Fortify Standalone Report Generator

Developer Workbook

akka-actor-tests

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-actor-tests 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-actor-tests
Project Version:	
SCA:	Results Present
WebInspect:	Results Not Present
WebInspect Agent:	Results Not Present
Other:	Results Not Present

Issues by Priority

Impact	**57** **High**	**0** **Critical**
	650 **Low**	**0** **Medium**
	Likelihood	

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, 11:10 AM	**Engine Version:**	21.1.1.0009
Host Name:	Jacks-Work-MBP.local	**Certification:**	VALID
Number of Files:	150	**Lines of Code:**	15,884

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 / 6	0 / 6
Code Correctness: Call to System.gc()	0	0	0	0 / 1	0 / 1
Code Correctness: Class Does Not Implement equals	0	0	0	0 / 96	0 / 96
Code Correctness: Constructor Invokes Overridable Function	0	0	0	0 / 114	0 / 114
Code Correctness: Erroneous String Compare	0	0	0	0 / 8	0 / 8
Code Correctness: Non-Static Inner Class Implements Serializable	0	0	0	0 / 107	0 / 107
Dead Code: Expression is Always false	0	0	0	0 / 173	0 / 173
Dead Code: Expression is Always true	0	0	0	0 / 1	0 / 1
Insecure Randomness	0	0 / 26	0	0	0 / 26
Insecure Randomness: Hardcoded Seed	0	0 / 2	0	0	0 / 2
J2EE Bad Practices: Leftover Debug Code	0	0	0	0 / 1	0 / 1
J2EE Bad Practices: Sockets	0	0	0	0 / 32	0 / 32
J2EE Bad Practices: Threads	0	0	0	0 / 96	0 / 96
Null Dereference	0	0 / 1	0	0	0 / 1
Often Misused: Authentication	0	0 / 25	0	0	0 / 25
Poor Error Handling: Empty Catch Block	0	0	0	0 / 1	0 / 1
Poor Error Handling: Overly Broad Catch	0	0	0	0 / 1	0 / 1
Poor Style: Value Never Read	0	0	0	0 / 4	0 / 4
Resource Injection	0	0	0	0 / 1	0 / 1
Setting Manipulation	0	0	0	0 / 1	0 / 1
System Information Leak	0	0	0	0 / 1	0 / 1
System Information Leak: External	0	0	0	0 / 2	0 / 2
Unchecked Return Value	0	0	0	0 / 4	0 / 4
Unreleased Resource: Sockets	0	0 / 2	0	0	0 / 2
Unreleased Resource: Streams	0	0 / 1	0	0	0 / 1

Results Outline

Code Correctness: Byte Array to String Conversion (6 issues)

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.

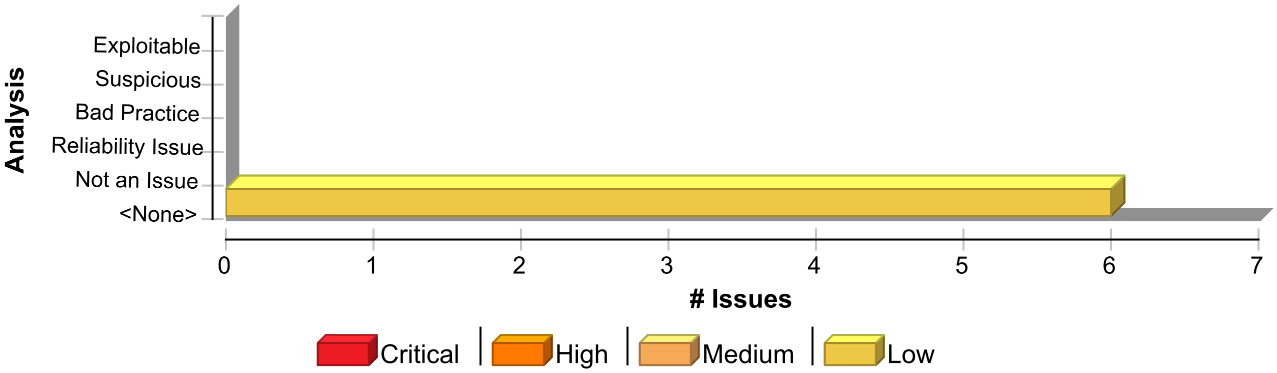
Recommendation

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	6	0	0	6
Total	**6**	**0**	**0**	**6**

Code Correctness: Byte Array to String Conversion

Low

Package: akka.serialization

scala/akka/serialization/AsyncSerializeSpec.scala, line 52 (Code Correctness: Byte Array to String Conversion)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Semantic)

Sink Details

Sink: String()
Enclosing Method: fromBinaryAsync()
File: scala/akka/serialization/AsyncSerializeSpec.scala:52
Taint Flags:

```
49 override def fromBinaryAsync(bytes: Array[Byte], manifest: String): Future[AnyRef] = {
50 manifest match {
51 case "1" => Future.successful(Message1(new String(bytes)))
52 case "2" => Future.successful(Message2(new String(bytes)))
53 case _ => throw new IllegalArgumentException(s"Unknown manifest $manifest")
54 }
55 }
```

scala/akka/serialization/AsyncSerializeSpec.scala, line 51 (Code Correctness: Byte Array to String Conversion)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Semantic)

Sink Details

Sink: String()

Code Correctness: Byte Array to String Conversion

Low

Package: akka.serialization

scala/akka/serialization/AsyncSerializeSpec.scala, line 51 (Code Correctness: Byte Array to String Conversion)

Low

Enclosing Method: fromBinaryAsync()
File: scala/akka/serialization/AsyncSerializeSpec.scala:51
Taint Flags:

```
48
49  override def fromBinaryAsync(bytes: Array[Byte], manifest: String): Future[AnyRef] = {
50  manifest match {
51  case "1" => Future.successful(Message1(new String(bytes)))
52  case "2" => Future.successful(Message2(new String(bytes)))
53  case _ => throw new IllegalArgumentException(s"Unknown manifest $manifest")
54  }
```

scala/akka/serialization/AsyncSerializeSpec.scala, line 79 (Code Correctness: Byte Array to String Conversion)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Semantic)

Sink Details

Sink: String()
Enclosing Method: fromBinaryAsyncCS()
File: scala/akka/serialization/AsyncSerializeSpec.scala:79
Taint Flags:

```
76  override def fromBinaryAsyncCS(bytes: Array[Byte], manifest: String): CompletionStage[AnyRef] = {
77  manifest match {
78  case "1" => CompletableFuture.completedFuture(Message3(new String(bytes)))
79  case "2" => CompletableFuture.completedFuture(Message4(new String(bytes)))
80  case _ => throw new IllegalArgumentException(s"Unknown manifest $manifest")
81  }
82  }
```

scala/akka/serialization/AsyncSerializeSpec.scala, line 78 (Code Correctness: Byte Array to String Conversion)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Semantic)

Sink Details

Sink: String()
Enclosing Method: fromBinaryAsyncCS()
File: scala/akka/serialization/AsyncSerializeSpec.scala:78
Taint Flags:

Code Correctness: Byte Array to String Conversion

Low

Package: akka.serialization

scala/akka/serialization/AsyncSerializeSpec.scala, line 78 (Code Correctness: Byte Array to String Conversion)

Low

```
75
76  override def fromBinaryAsyncCS(bytes: Array[Byte], manifest: String): CompletionStage[AnyRef] = {
77  manifest match {
78  case "1" => CompletableFuture.completedFuture(Message3(new String(bytes)))
79  case "2" => CompletableFuture.completedFuture(Message4(new String(bytes)))
80  case _ => throw new IllegalArgumentException(s"Unknown manifest $manifest")
81  }
```

Package: scala.akka.event

scala/akka/event/LoggerSpec.scala, line 191 (Code Correctness: Byte Array to String Conversion)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Semantic)

Sink Details

Sink: String()
Enclosing Method: apply()
File: scala/akka/event/LoggerSpec.scala:191
Taint Flags:

```
188  out.close()
189  }
190
191  val logMessages = new String(out.toByteArray).split("\n")
192  logMessages.head should include("msg1")
193  logMessages.last should include("msg3")
194  logMessages.size should ===(3)
```

Package: scala.akka.util

scala/akka/util/ByteStringSpec.scala, line 1224 (Code Correctness: Byte Array to String Conversion)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Semantic)

Sink Details

Sink: String()
Enclosing Method: apply()
File: scala/akka/util/ByteStringSpec.scala:1224
Taint Flags:

Code Correctness: Byte Array to String Conversion — Low

Package: scala.akka.util

scala/akka/util/ByteStringSpec.scala, line 1224 (Code Correctness: Byte Array to String Conversion) — Low

```
1221 iterator.copyToArray(array, 0, 2)
1222 iterator.copyToArray(array, 2, 2)
1223 iterator.copyToArray(array, 4, 2)
1224 assert(new String(array) === "123456")
1225 }
1226
1227 "calling copyToArray with length passing end of destination" in {
```

Code Correctness: Call to System.gc() (1 issue)

Abstract

Explicit requests for garbage collection are a bellwether indicating likely performance problems.

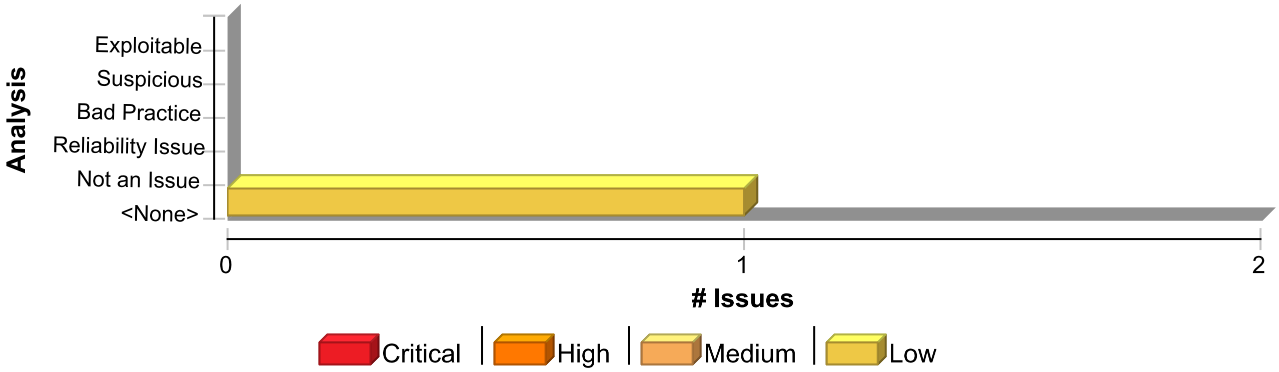
Explanation

At some point in every Java developer's career, a problem surfaces that appears to be so mysterious, impenetrable, and impervious to debugging that there seems to be no alternative but to blame the garbage collector. Especially when the bug is related to time and state, there may be a hint of empirical evidence to support this theory: inserting a call to `System.gc()` sometimes seems to make the problem go away. In almost every case we have seen, calling `System.gc()` is the wrong thing to do. In fact, calling `System.gc()` can cause performance problems if it is invoked too often.

Recommendation

When it seems as though calling `System.gc()` has solved a problem, look for other explanations, particularly ones that involve time and interaction between threads, processes, or the JVM and the operating system. I/O buffering, synchronization, and race conditions are all likely culprits.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Code Correctness: Call to System.gc()	1	0	0	1
Total	**1**	**0**	**0**	**1**

Code Correctness: Call to System.gc()	Low
Package: akka.actor	
scala/akka/actor/SupervisorHierarchySpec.scala, line 644 (Code Correctness: Call to System.gc())	**Low**

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Code Correctness: Call to System.gc() Low

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 644 (Code Correctness: Call to System.gc()) Low

Sink: FunctionCall: gc
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:644
Taint Flags:

```
641 val next = weak.filter(_.get ne null)
642 if (next.nonEmpty) {
643 context.system.scheduler.scheduleOnce(workSchedule, self, GCcheck(next))(context.dispatcher)
644 System.gc()
645 stay()
646 } else {
647 testActor ! "stressTestSuccessful"
```

Code Correctness: Class Does Not Implement equals (96 issues)

Abstract

The `equals()` method is called on an object that does not implement `equals()`.

Explanation

When comparing objects, developers usually want to compare properties of objects. However, calling `equals()` on a class (or any super class/interface) that does not explicitly implement `equals()` results in a call to the `equals()` method inherited from `java.lang.Object`. Instead of comparing object member fields or other properties, `Object.equals()` compares two object instances to see if they are the same. Although there are legitimate uses of `Object.equals()`, it is often an indication of buggy code. **Example 1:**

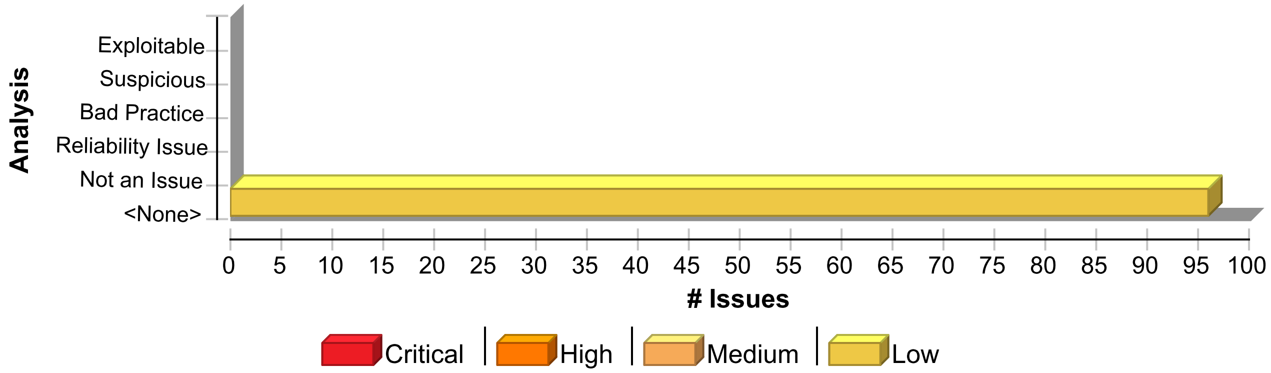
```
public class AccountGroup
{
    private int gid;

    public int getGid()
    {
        return gid;
    }

    public void setGid(int newGid)
    {
        gid = newGid;
    }
}
...
public class CompareGroup
{
    public boolean compareGroups(AccountGroup group1, AccountGroup group2)
    {
        return group1.equals(group2);   //equals() is not implemented in
AccountGroup
    }
}
```

Recommendation

Verify that the use of `Object.equals()` is really the method you intend to call. If not, implement an `equals()` method or use a different method for comparing objects. **Example 2:** The following code adds an `equals()` method to the example from the Explanation section.

```
public class AccountGroup
{
    private int gid;

    public int getGid()
    {
        return gid;
    }

    public void setGid(int newGid)
    {
        gid = newGid;
    }
    public boolean equals(Object o)
    {
        if (!(o instanceof AccountGroup))
            return false;
        AccountGroup other = (AccountGroup) o;
        return (gid == other.getGid());
    }
}
...
public class CompareGroup
{
    public static boolean compareGroups(AccountGroup group1, AccountGroup 
group2)
    {
        return group1.equals(group2);
    }
}
```

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Code Correctness: Class Does Not Implement equals	96	0	0	96
Total	**96**	**0**	**0**	**96**

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/SupervisorSpec.scala, line 463 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorSpec.scala:463

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/SupervisorSpec.scala, line 463 (Code Correctness: Class Does Not Implement equals) — Low

Taint Flags:

```
460 }
461
462 def receive = {
463 case Terminated(t) if t.path == child.path => testActor ! "child terminated"
464 case l: TestLatch => child ! l
465 case "test" => sender() ! "green"
466 case "testchild" => child.forward("test")
```

scala/akka/actor/RestartStrategySpec.scala, line 176 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/RestartStrategySpec.scala:176
Taint Flags:

```
173
174 def receive = {
175 case Ping => countDownLatch.countDown()
176 case Crash => throw new Exception("Crashing...")
177 }
178 override def postRestart(reason: Throwable) = {
179 if (!restartLatch.isOpen)
```

scala/akka/actor/FSMActorSpec.scala, line 173 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:173
Taint Flags:

```
170 system.eventStream.subscribe(testActor, classOf[Logging.Error])
171 fsm ! "go"
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/FSMActorSpec.scala, line 173 (Code Correctness: Class Does Not Implement equals) — Low

```
172 expectMsgPF(1 second, hint = "Next state 2 does not exist") {
173 case Logging.Error(_, `name`, _, "Next state 2 does not exist") => true
174 }
175 system.eventStream.unsubscribe(testActor)
176 }
```

scala/akka/actor/ActorSelectionSpec.scala, line 62 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorSelectionSpec.scala:62
Taint Flags:

```
59 def identify(selection: ActorSelection): Option[ActorRef] = {
60 selection.tell(Identify(selection), idProbe.ref)
61 val result = idProbe.expectMsgPF() {
62 case ActorIdentity(`selection`, ref) => ref
63 }
64 val asked = Await.result((selection ? Identify(selection)).mapTo[ActorIdentity], timeout.duration)
65 asked.ref should ===(result)
```

scala/akka/actor/SchedulerSpec.scala, line 330 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/SchedulerSpec.scala:330
Taint Flags:

```
327 val props = Props(new Actor {
328 def receive = {
329 case Ping => pingLatch.countDown()
330 case Crash => throw new Exception("CRASH")
331 }
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/SchedulerSpec.scala, line 330 (Code Correctness: Class Does Not Implement equals) — Low

```
332
333  override def postRestart(reason: Throwable) = restartLatch.open()
```

scala/akka/actor/FSMActorSpec.scala, line 291 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:291
Taint Flags:

```
288  expectMsg(1 second, Logging.Debug(name, fsmClass, "transition 1 -> 2"))
289  fsm ! "stop"
290  expectMsgPF(1 second, hint = "processing Event(stop,null)") {
291  case Logging.Debug(`name`, `fsmClass`, s: String)
292  if s.startsWith("processing Event(stop,null) from Actor[") =>
293  true
294  }
```

scala/akka/actor/FSMActorSpec.scala, line 207 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:207
Taint Flags:

```
204  case Event(2, null) => stop(FSM.Normal, expected)
205  }
206  onTermination {
207  case StopEvent(FSM.Normal, 1, `expected`) => testActor ! "green"
208  }
209  }))
210  actor ! 2
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 600 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:600
Taint Flags:

```
597
598 when(Stopping, stateTimeout = 5.seconds.dilated) {
599 case this.Event(PongOfDeath, _) => stay()
600 case this.Event(Terminated(r), _) if r == hierarchy =>
601 @nowarn
602 val undead = children.filterNot(_.isTerminated)
603 if (undead.nonEmpty) {
```

scala/akka/actor/ActorSelectionSpec.scala, line 241 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: check()
File: scala/akka/actor/ActorSelectionSpec.scala:241
Taint Flags:

```
238 askNode(looker, SelectString(target.path.toString)) should ===(Some(target))
239 askNode(looker, SelectString(target.path.toString + "/")) should ===(Some(target))
240 }
241 if (target != root)
242 askNode(c1, SelectString("../.." + target.path.elements.mkString("/", "/", "/"))) should ===(Some(target))
243 }
244 for (target <- Seq(root, syst, user)) check(target)
```

scala/akka/actor/ActorRefSpec.scala, line 421 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/ActorRefSpec.scala, line 421 (Code Correctness: Class Does Not Implement equals) — Low

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:421
Taint Flags:

```
418 val ref = system.actorOf(Props(new Actor {
419 def receive = {
420 case 5 => sender() ! "five"
421 case 0 => sender() ! "null"
422 }
423 }))
424
```

scala/akka/actor/DeathWatchSpec.scala, line 123 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/DeathWatchSpec.scala:123
Taint Flags:

```
120 "The Death Watch" must {
121 def expectTerminationOf(actorRef: ActorRef) =
122 expectMsgPF(5 seconds, "" + actorRef + ": Stopped or Already terminated when linking") {
123 case WrappedTerminated(Terminated(`actorRef`)) => true
124 }
125
126 "notify with one Terminated message when an Actor is stopped" in {
```

scala/akka/actor/DeathWatchSpec.scala, line 46 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()

Code Correctness: Class Does Not Implement equals

Low

Package: akka.actor

scala/akka/actor/DeathWatchSpec.scala, line 46 (Code Correctness: Class Does Not Implement equals)

Low

File: scala/akka/actor/DeathWatchSpec.scala:46
Taint Flags:

```
43 }), "kid"))
44 currentKid.forward("NKOTB")
45 context.become {
46 case Terminated(`currentKid`) =>
47 testActor ! "GREEN"
48 context.unbecome()
49 }
```

scala/akka/actor/ActorSelectionSpec.scala, line 292 (Code Correctness: Class Does Not Implement equals)

Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorSelectionSpec.scala:292
Taint Flags:

```
289 implicit val sender = c1
290 ActorSelection(c21, "../../*") ! GetSender(testActor)
291 val actors = Set() ++ receiveWhile(messages = 2) {
292 case `c1` => lastSender
293 }
294 actors should ===(Set(c1, c2))
295 expectNoMessage()
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 662 (Code Correctness: Class Does Not Implement equals)

Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:662
Taint Flags:

```
659 if (!e.msg.startsWith("not resumed") || !ignoreNotResumedLogs)
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 662 (Code Correctness: Class Does Not Implement equals) — Low

```
660 errors :+= sender() -> e
661 stay()
662 case this.Event(Terminated(r), _) if r == hierarchy =>
663 printErrors()
664 testActor ! "stressTestFailed"
665 stop()
```

scala/akka/actor/SupervisorMiscSpec.scala, line 104 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorMiscSpec.scala:104
Taint Flags:

```
101 if (newKid eq kid) "Failure: context.actorOf returned the same instance!"
102 else if (!kid.isTerminated) "Kid is zombie"
103 else if (newKid.isTerminated) "newKid was stillborn"
104 else if (kid.path != newKid.path) "The kids do not share the same path"
105 else "green"
106 testActor ! result
107 } catch {
```

scala/akka/actor/RestartStrategySpec.scala, line 48 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/RestartStrategySpec.scala:48
Taint Flags:

```
45
46 def receive = {
47 case Ping => countDownLatch.countDown()
48 case Crash => throw new Exception("Crashing...")
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/RestartStrategySpec.scala, line 48 (Code Correctness: Class Does Not Implement equals) — Low

```
49  }
50
51  override def postRestart(reason: Throwable) = {
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 697 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: getErrorsUp()
File: scala/akka/actor/SupervisorHierarchySpec.scala:697
Taint Flags:

```
694  case h: Hierarchy => errors :+= target -> ErrorLog("forced", h.log)
695  case _ => errors :+= target -> ErrorLog("fetched", stateCache.get(target.path).log)
696  }
697  if (target != hierarchy) getErrorsUp(l.getParent)
698  case _ => throw new IllegalArgumentException()
699  }
700  }
```

scala/akka/actor/ActorSelectionSpec.scala, line 302 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorSelectionSpec.scala:302
Taint Flags:

```
299  implicit val sender = c2
300  ActorSelection(c21, "../../*/c21") ! GetSender(testActor)
301  val actors = receiveWhile(messages = 2) {
302  case `c2` => lastSender
303  }
304  actors should ===(Seq(c21))
305  expectNoMessage()
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/SupervisorMiscSpec.scala, line 97 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorMiscSpec.scala:97
Taint Flags:

```
94  val parent = system.actorOf(Props(new Actor {
95  val kid = context.watch(context.actorOf(Props.empty, "foo"))
96  def receive = {
97  case Terminated(`kid`) =>
98  try {
99  val newKid = context.actorOf(Props.empty, "foo")
100  val result =
```

scala/akka/actor/RestartStrategySpec.scala, line 47 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/RestartStrategySpec.scala:47
Taint Flags:

```
44  val employeeProps = Props(new Actor {
45
46  def receive = {
47  case Ping => countDownLatch.countDown()
48  case Crash => throw new Exception("Crashing...")
49  }
50
```

scala/akka/actor/RestartStrategySpec.scala, line 231 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Code Correctness: Class Does Not Implement equals

Low

Package: akka.actor

scala/akka/actor/RestartStrategySpec.scala, line 231 (Code Correctness: Class Does Not Implement equals)

Low

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/RestartStrategySpec.scala:231
Taint Flags:

```
228
229  def receive = {
230  case Ping => countDownLatch.countDown()
231  case Crash => throw new Exception("Crashing...")
232  }
233
234  override def postRestart(reason: Throwable) = {
```

scala/akka/actor/RestartStrategySpec.scala, line 119 (Code Correctness: Class Does Not Implement equals)

Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/RestartStrategySpec.scala:119
Taint Flags:

```
116  val employeeProps = Props(new Actor {
117
118  def receive = {
119  case Ping =>
120  if (!pingLatch.isOpen) pingLatch.open() else secondPingLatch.open()
121  case Crash => throw new Exception("Crashing...")
122  }
```

scala/akka/actor/RestartStrategySpec.scala, line 175 (Code Correctness: Class Does Not Implement equals)

Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/RestartStrategySpec.scala, line 175 (Code Correctness: Class Does Not Implement equals) — Low

File: scala/akka/actor/RestartStrategySpec.scala:175
Taint Flags:

```
172  val employeeProps = Props(new Actor {
173
174  def receive = {
175  case Ping => countDownLatch.countDown()
176  case Crash => throw new Exception("Crashing...")
177  }
178  override def postRestart(reason: Throwable) = {
```

scala/akka/actor/RestartStrategySpec.scala, line 90 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/RestartStrategySpec.scala:90
Taint Flags:

```
87  val employeeProps = Props(new Actor {
88
89  def receive = {
90  case Crash => throw new Exception("Crashing...")
91  }
92
93  override def postRestart(reason: Throwable) = {
```

scala/akka/actor/ExtensionSpec.scala, line 52 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: FailingTestExtension()
File: scala/akka/actor/ExtensionSpec.scala:52
Taint Flags:

```
49  system.actorOf(Props.empty, "uniqueName")
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/ExtensionSpec.scala, line 52 (Code Correctness: Class Does Not Implement equals) — Low

```
50
51  // Always fail, but 'hide' this from IntelliJ to avoid compilation issues:
52  if (42.toString == "42")
53  throw new FailingTestExtension.TestException
54 }
55
```

scala/akka/actor/SupervisorSpec.scala, line 443 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorSpec.scala:443
Taint Flags:

```
440  "not lose system messages when a NonFatal exception occurs when processing a system message" in {
441  val parent = system.actorOf(Props(new Actor {
442  override val supervisorStrategy = OneForOneStrategy()({
443  case e: IllegalStateException if e.getMessage == "OHNOES" => throw e
444  case _ => SupervisorStrategy.Restart
445  })
446  val child = context.watch(context.actorOf(Props(new Actor {
```

scala/akka/actor/SupervisorSpec.scala, line 48 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorSpec.scala:48
Taint Flags:

```
45  def receive = {
46  case Ping =>
47  sendTo ! PingMessage
48  if (sender() != sendTo)
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/SupervisorSpec.scala, line 48 (Code Correctness: Class Does Not Implement equals) — Low

```
49 sender() ! PongMessage
50 case Die =>
51 throw new RuntimeException(ExceptionMessage)
```

scala/akka/actor/FSMActorSpec.scala, line 47 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:47
Taint Flags:

```
44 soFar + digit match {
45 case incomplete if incomplete.length < code.length =>
46 stay().using(CodeState(incomplete, code))
47 case codeTry if (codeTry == code) => {
48 doUnlock()
49 goto(Open).using(CodeState("", code)).forMax(timeout)
50 }
```

scala/akka/actor/LocalActorRefProviderSpec.scala, line 46 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/LocalActorRefProviderSpec.scala:46
Taint Flags:

```
43 val child = context.actorOf(Props.empty, name = childName)
44 def receive = {
45 case "lookup" =>
46 if (childName == child.path.name) {
47 val resolved = system.asInstanceOf[ExtendedActorSystem].provider.resolveActorRef(child.path)
48 sender() ! resolved
49 } else sender() ! s"$childName is not ${child.path.name}!"
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/SupervisorMiscSpec.scala, line 143 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: handleChildTerminated()
File: scala/akka/actor/SupervisorMiscSpec.scala:143
Taint Flags:

```
140 child: ActorRef,
141 children: Iterable[ActorRef]): Unit = {
142 val newKid = context.actorOf(Props.empty, child.path.name)
143 testActor ! { if ((newKid ne child) && newKid.path == child.path) "green" else "red" }
144 }
145 }
146
```

scala/akka/actor/RestartStrategySpec.scala, line 121 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/RestartStrategySpec.scala:121
Taint Flags:

```
118 def receive = {
119 case Ping =>
120 if (!pingLatch.isOpen) pingLatch.open() else secondPingLatch.open()
121 case Crash => throw new Exception("Crashing...")
122 }
123 override def postRestart(reason: Throwable) = {
124 if (!restartLatch.isOpen)
```

scala/akka/actor/DeathWatchSpec.scala, line 231 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Code Correctness: Class Does Not Implement equals

Low

Package: akka.actor

scala/akka/actor/DeathWatchSpec.scala, line 231 (Code Correctness: Class Does Not Implement equals)

Low

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/DeathWatchSpec.scala:231
Taint Flags:

```
228 failed ! Kill
229 val result = receiveWhile(3 seconds, messages = 3) {
230 case FF(Failed(_, _: ActorKilledException, _)) if lastSender eq failed => 1
231 case FF(Failed(_, DeathPactException(`failed`), _)) if lastSender eq brother => 2
232 case WrappedTerminated(Terminated(`brother`)) => 3
233 }
234 testActor.isTerminated should not be true
```

scala/akka/actor/SchedulerSpec.scala, line 329 (Code Correctness: Class Does Not Implement equals)

Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/SchedulerSpec.scala:329
Taint Flags:

```
326 system.actorOf(Props(new Supervisor(AllForOneStrategy(3, 1 second)(List(classOf[Exception])))))
327 val props = Props(new Actor {
328 def receive = {
329 case Ping => pingLatch.countDown()
330 case Crash => throw new Exception("CRASH")
331 }
332
```

scala/akka/actor/ActorRefSpec.scala, line 420 (Code Correctness: Class Does Not Implement equals)

Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/ActorRefSpec.scala, line 420 (Code Correctness: Class Does Not Implement equals) — Low

File: scala/akka/actor/ActorRefSpec.scala:420
Taint Flags:

```
417 val timeout = Timeout(20.seconds)
418 val ref = system.actorOf(Props(new Actor {
419 def receive = {
420 case 5 => sender() ! "five"
421 case 0 => sender() ! "null"
422 }
423 }))
```

scala/akka/actor/FSMActorSpec.scala, line 283 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:283
Taint Flags:

```
280 system.eventStream.subscribe(testActor, classOf[Logging.Debug])
281 fsm ! "go"
282 expectMsgPF(1 second, hint = "processing Event(go,null)") {
283 case Logging.Debug(`name`, `fsmClass`, s: String)
284 if s.startsWith("processing Event(go,null) from Actor[") =>
285 true
286 }
```

scala/akka/actor/RestartStrategySpec.scala, line 230 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/RestartStrategySpec.scala:230
Taint Flags:

```
227 val employeeProps = Props(new Actor {
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/RestartStrategySpec.scala, line 230 (Code Correctness: Class Does Not Implement equals) — Low

```
228
229 def receive = {
230 case Ping => countDownLatch.countDown()
231 case Crash => throw new Exception("Crashing...")
232 }
233
```

scala/akka/actor/SupervisorSpec.scala, line 72 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorSpec.scala:72
Taint Flags:

```
69
70 def receive = {
71 case Die => temp.forward(Die)
72 case Terminated(`temp`) => sendTo ! "terminated"
73 case Status.Failure(_) => /*Ignore*/
74 }
75 }
```

scala/akka/actor/ActorWithStashSpec.scala, line 81 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorWithStashSpec.scala:81
Taint Flags:

```
78 context.stop(watched)
79
80 def receive = {
81 case Terminated(`watched`) =>
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/ActorWithStashSpec.scala, line 81 (Code Correctness: Class Does Not Implement equals) — Low

```
82  if (!stashed) {
83  stash()
84  stashed = true
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 256 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: postRestart()
File: scala/akka/actor/SupervisorHierarchySpec.scala:256
Taint Flags:

```
253  }
254  cause match {
255  case f: Failure if f.failPost > 0 => { f.failPost -= 1; throw f }
256  case PostRestartException(`self`, f: Failure, _) if f.failPost > 0 => { f.failPost -= 1; throw f }
257  case _ =>
258  }
259  }
```

scala/akka/actor/ForwardActorSpec.scala, line 39 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/ForwardActorSpec.scala:39
Taint Flags:

```
36
37  "forward actor reference when invoking forward on tell" in {
38  val replyTo = system.actorOf(Props(new Actor {
39  def receive = { case ExpectedMessage => testActor ! ExpectedMessage }
40  }))
41
42  val chain = createForwardingChain(system)
```

Code Correctness: Class Does Not Implement equals

Low

Package: akka.actor

scala/akka/actor/LocalActorRefProviderSpec.scala, line 105 (Code Correctness: Class Does Not Implement equals)

Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/LocalActorRefProviderSpec.scala:105
Taint Flags:

```
102 val GetChild = "GetChild"
103 val a = watch(system.actorOf(Props(new Actor {
104 val child = context.actorOf(Props.empty)
105 def receive = { case `GetChild` => sender() ! child }
106 })))
107 a.tell(GetChild, testActor)
108 val child = expectMsgType[ActorRef]
```

scala/akka/actor/TypedActorSpec.scala, line 401 (Code Correctness: Class Does Not Implement equals)

Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/TypedActorSpec.scala:401
Taint Flags:

```
398 filterEvents(EventFilter[IllegalStateException]("expected")) {
399 val boss = system.actorOf(Props(new Actor {
400 override val supervisorStrategy = OneForOneStrategy() {
401 case e: IllegalStateException if e.getMessage == "expected" => SupervisorStrategy.Resume
402 }
403 def receive = {
404 case p: TypedProps[_] => context.sender() ! akka.actor.TypedActor(context).typedActorOf(p)
```

scala/akka/actor/DeathWatchSpec.scala, line 232 (Code Correctness: Class Does Not Implement equals)

Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/DeathWatchSpec.scala, line 232 (Code Correctness: Class Does Not Implement equals) — Low

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/DeathWatchSpec.scala:232
Taint Flags:

```
229 val result = receiveWhile(3 seconds, messages = 3) {
230 case FF(Failed(_, _: ActorKilledException, _)) if lastSender eq failed => 1
231 case FF(Failed(_, DeathPactException(`failed`), _)) if lastSender eq brother => 2
232 case WrappedTerminated(Terminated(`brother`)) => 3
233 }
234 testActor.isTerminated should not be true
235 result should ===(Seq(1, 2, 3))
```

scala/akka/actor/FSMActorSpec.scala, line 204 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:204
Taint Flags:

```
201 val actor = system.actorOf(Props(new Actor with FSM[Int, String] {
202 startWith(1, null)
203 when(1) {
204 case Event(2, null) => stop(FSM.Normal, expected)
205 }
206 onTermination {
207 case StopEvent(FSM.Normal, 1, `expected`) => testActor ! "green"
```

scala/akka/actor/FSMActorSpec.scala, line 359 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor

scala/akka/actor/FSMActorSpec.scala, line 359 (Code Correctness: Class Does Not Implement equals) — Low

File: scala/akka/actor/FSMActorSpec.scala:359
Taint Flags:

```
356 p.ref ! StateTimeout
357 stay()
358
359 case Event(OverrideTimeoutToInf, _) =>
360 p.ref ! OverrideTimeoutToInf
361 stay().forMax(Duration.Inf)
362 }
```

Package: akka.dispatch

scala/akka/dispatch/MailboxConfigSpec.scala, line 303 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/dispatch/MailboxConfigSpec.scala:303
Taint Flags:

```
300 }).withDispatcher(dispatcherId)))
301 def receive = {
302 case Ping => a.tell(Ping, b)
303 case Terminated(`a` | `b`) => if (context.children.isEmpty) context.stop(self)
304 }
305 }))
306 watch(runner)
```

scala/akka/dispatch/MailboxConfigSpec.scala, line 303 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/dispatch/MailboxConfigSpec.scala:303
Taint Flags:

Code Correctness: Class Does Not Implement equals — Low

Package: akka.dispatch

scala/akka/dispatch/MailboxConfigSpec.scala, line 303 (Code Correctness: Class Does Not Implement equals) — Low

```
300 }).withDispatcher(dispatcherId)))
301 def receive = {
302 case Ping => a.tell(Ping, b)
303 case Terminated(`a` | `b`) => if (context.children.isEmpty) context.stop(self)
304 }
305 }))
306 watch(runner)
```

Package: akka.event

scala/akka/event/LoggingReceiveSpec.scala, line 114 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:114
Taint Flags:

```
111
112 actor ! "bah"
113 expectMsgPF() {
114 case UnhandledMessage("bah", _, `actor`) => true
115 }
116 }
117 }
```

scala/akka/event/LoggingReceiveSpec.scala, line 164 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:164
Taint Flags:

```
161 actor ! "buh"
162 fishForSpecificMessage() {
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.event

scala/akka/event/LoggingReceiveSpec.scala, line 164 (Code Correctness: Class Does Not Implement equals) — Low

```
163 case Logging.Info(src, _, msg)
164 if src == actor.path.toString && msg == "received handled message buh from " + self =>
165 ()
166 }
167 expectMsg("x")
```

scala/akka/event/LoggingReceiveSpec.scala, line 236 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:236
Taint Flags:

```
233
234 supervisor.unwatch(actor)
235 fishForMessage(hint = "no longer watched by") {
236 case Logging.Debug(`aname`, `sclass`, msg: String) if msg.startsWith("no longer watched by") => true
237 case _ => false
238 }
239 }
```

scala/akka/event/LoggingReceiveSpec.scala, line 262 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:262
Taint Flags:

```
259 val aclass = classOf[TestLogActor]
260
261 expectMsgAllPF(messages = 2) {
262 case Logging.Debug(`aname`, `aclass`, msg: String)
263 if msg.startsWith("started (" + classOf[TestLogActor].getName) =>
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.event

scala/akka/event/LoggingReceiveSpec.scala, line 262 (Code Correctness: Class Does Not Implement equals) — Low

```
264 0
265 case Logging.Debug(`sname`, `sclass`, msg: String) if msg == s"now supervising TestActor[$aname]" => 1
```

scala/akka/event/LoggingReceiveSpec.scala, line 253 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:253
Taint Flags:

```
250 val sclass = classOf[TestLogActor]
251
252 expectMsgAllPF(messages = 2) {
253 case Logging.Debug(`sname`, `sclass`, msg: String) if msg.startsWith("started") => 0
254 case Logging.Debug(_, _, msg: String) if msg.startsWith("now supervising") => 1
255 }
256
```

scala/akka/event/LoggingReceiveSpec.scala, line 272 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:272
Taint Flags:

```
269 actor ! Kill
270 expectMsgAllPF(messages = 3) {
271 case Logging.Error(_: ActorKilledException, `aname`, _, "Kill") => 0
272 case Logging.Debug(`aname`, `aclass`, "restarting") => 1
273 case Logging.Debug(`aname`, `aclass`, "restarted") => 2
274 }
275 }
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.event

scala/akka/event/LoggingReceiveSpec.scala, line 273 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:273
Taint Flags:

```
270 expectMsgAllPF(messages = 3) {
271 case Logging.Error(_: ActorKilledException, `aname`, _, "Kill") => 0
272 case Logging.Debug(`aname`, `aclass`, "restarting") => 1
273 case Logging.Debug(`aname`, `aclass`, "restarted") => 2
274 }
275 }
276
```

scala/akka/event/LoggingReceiveSpec.scala, line 212 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:212
Taint Flags:

```
209 TestActorRef[TestLogActor](Props[TestLogActor](), supervisor, "none")
210
211 fishForMessage(hint = "now supervising") {
212 case Logging.Debug(`sname`, _, msg: String) if msg.startsWith("now supervising") => true
213 case _ => false
214 }
215 }
```

scala/akka/event/LoggingReceiveSpec.scala, line 265 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Code Correctness: Class Does Not Implement equals — Low

Package: akka.event

scala/akka/event/LoggingReceiveSpec.scala, line 265 (Code Correctness: Class Does Not Implement equals) — Low

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:265
Taint Flags:

```
262 case Logging.Debug(`aname`, `aclass`, msg: String)
263 if msg.startsWith("started (" + classOf[TestLogActor].getName) =>
264 0
265 case Logging.Debug(`sname`, `sclass`, msg: String) if msg == s"now supervising TestActor[$aname]" => 1
266 }
267
268 EventFilter[ActorKilledException](occurrences = 1).intercept {
```

scala/akka/event/LoggingReceiveSpec.scala, line 230 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:230
Taint Flags:

```
227
228 supervisor.watch(actor)
229 fishForMessage(hint = "now watched by") {
230 case Logging.Debug(`aname`, `sclass`, msg: String) if msg.startsWith("now watched by") => true
231 case _ => false
232 }
233
```

scala/akka/event/LoggingReceiveSpec.scala, line 265 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()

Code Correctness: Class Does Not Implement equals — Low

Package: akka.event

scala/akka/event/LoggingReceiveSpec.scala, line 265 (Code Correctness: Class Does Not Implement equals) — Low

File: scala/akka/event/LoggingReceiveSpec.scala:265
Taint Flags:

```
262 case Logging.Debug(`aname`, `aclass`, msg: String)
263 if msg.startsWith("started (" + classOf[TestLogActor].getName) =>
264 0
265 case Logging.Debug(`sname`, `sclass`, msg: String) if msg == s"now supervising TestActor[$aname]" => 1
266 }
267
268 EventFilter[ActorKilledException](occurrences = 1).intercept {
```

scala/akka/event/LoggingReceiveSpec.scala, line 205 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:205
Taint Flags:

```
202 val sname = supervisor.path.toString
203
204 fishForMessage(hint = "now supervising") {
205 case Logging.Debug(`lname`, _, msg: String) if msg.startsWith("now supervising") => true
206 case _ => false
207 }
208
```

scala/akka/event/EventBusSpec.scala, line 298 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: matches()
File: scala/akka/event/EventBusSpec.scala:298
Taint Flags:

```
295 protected def compareClassifiers(a: Classifier, b: Classifier): Int = a.compareTo(b)
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.event

scala/akka/event/EventBusSpec.scala, line 298 (Code Correctness: Class Does Not Implement equals) — Low

```
296  protected def compareSubscribers(a: Subscriber, b: Subscriber): Int = akka.util.Helpers.compareIdentityHash(a, b)
297
298  protected def matches(classifier: Classifier, event: Event): Boolean = event.toString == classifier
299
300  protected def publish(event: Event, subscriber: Subscriber): Unit = subscriber(event)
301  }
```

scala/akka/event/LoggerSpec.scala, line 130 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: mdc()
File: scala/akka/event/LoggerSpec.scala:130
Taint Flags:

```
127  val always = Map("requestId" -> reqId)
128  val cmim = "Current Message in MDC"
129  val perMessage = currentMessage match {
130  case `cmim` => Map[String, Any]("currentMsg" -> cmim, "currentMsgLength" -> cmim.length)
131  case _ => Map()
132  }
133  always ++ perMessage
```

scala/akka/event/LoggingReceiveSpec.scala, line 186 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:186
Taint Flags:

```
183  val name = actor.path.toString
184  actor ! PoisonPill
185  fishForMessage(hint = "received AutoReceiveMessage Envelope(PoisonPill") {
186  case Logging.Debug(`name`, _, msg: String)
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.event

scala/akka/event/LoggingReceiveSpec.scala, line 186 (Code Correctness: Class Does Not Implement equals) — Low

```
187 if msg.startsWith("received AutoReceiveMessage Envelope(PoisonPill") =>
188 true
189 case _ => false
```

scala/akka/event/LoggingReceiveSpec.scala, line 271 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:271
Taint Flags:

```
268 EventFilter[ActorKilledException](occurrences = 1).intercept {
269 actor ! Kill
270 expectMsgAllPF(messages = 3) {
271 case Logging.Error(_: ActorKilledException, `aname`, _, "Kill") => 0
272 case Logging.Debug(`aname`, `aclass`, "restarting") => 1
273 case Logging.Debug(`aname`, `aclass`, "restarted") => 2
274 }
```

Package: akka.io

scala/akka/io/TcpIntegrationSpecSupport.scala, line 53 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: establishNewClientConnection()
File: scala/akka/io/TcpIntegrationSpecSupport.scala:53
Taint Flags:

```
50 connectCommander.sender() ! Register(clientHandler.ref)
51
52 bindHandler.expectMsgType[Connected] match {
53 case Connected(`localAddress`, `endpoint`) => //ok
54 case other => fail(s"No match: ${other}")
55 }
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.io

scala/akka/io/TcpIntegrationSpecSupport.scala, line 53 (Code Correctness: Class Does Not Implement equals) — Low

```
56 val serverHandler = TestProbe()
```

scala/akka/io/TcpIntegrationSpecSupport.scala, line 46 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: establishNewClientConnection()
File: scala/akka/io/TcpIntegrationSpecSupport.scala:46
Taint Flags:

```
43 val connectCommander = TestProbe()(clientSystem)
44 connectCommander.send(IO(Tcp)(clientSystem), Connect(endpoint, options = connectOptions))
45 val localAddress = connectCommander.expectMsgType[Connected] match {
46 case Connected(`endpoint`, localAddress) => localAddress
47 case Connected(other, _) => fail(s"No match: $other")
48 }
49 val clientHandler = TestProbe()(clientSystem)
```

scala/akka/io/TcpIntegrationSpecSupport.scala, line 53 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: establishNewClientConnection()
File: scala/akka/io/TcpIntegrationSpecSupport.scala:53
Taint Flags:

```
50 connectCommander.sender() ! Register(clientHandler.ref)
51
52 bindHandler.expectMsgType[Connected] match {
53 case Connected(`localAddress`, `endpoint`) => //ok
54 case other => fail(s"No match: ${other}")
55 }
56 val serverHandler = TestProbe()
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.util

scala/akka/util/ByteStringSpec.scala, line 317 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/util/ByteStringSpec.scala:317
Taint Flags:

```
314
315 reference.zipWithIndex
316 .collect({ // Since there is no partial put on LongBuffer, we need to collect only the interesting bytes
317 case (r, i) if byteOrder == ByteOrder.LITTLE_ENDIAN && i % elemSize < nBytes => r
318 case (r, i) if byteOrder == ByteOrder.BIG_ENDIAN && i % elemSize >= (elemSize - nBytes) => r
319 })
320 .toSeq == builder.result()
```

scala/akka/util/ByteStringSpec.scala, line 318 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: applyOrElse()
File: scala/akka/util/ByteStringSpec.scala:318
Taint Flags:

```
315 reference.zipWithIndex
316 .collect({ // Since there is no partial put on LongBuffer, we need to collect only the interesting bytes
317 case (r, i) if byteOrder == ByteOrder.LITTLE_ENDIAN && i % elemSize < nBytes => r
318 case (r, i) if byteOrder == ByteOrder.BIG_ENDIAN && i % elemSize >= (elemSize - nBytes) => r
319 })
320 .toSeq == builder.result()
321 }
```

Package: scala.akka.actor

scala/akka/actor/ActorSelectionSpec.scala, line 373 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse

Code Correctness: Class Does Not Implement equals — Low

Package: scala.akka.actor

scala/akka/actor/ActorSelectionSpec.scala, line 373 (Code Correctness: Class Does Not Implement equals) — Low

Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/actor/ActorSelectionSpec.scala:373
Taint Flags:

```
370 probe
371 .receiveN(2)
372 .map {
373 case ActorIdentity(1, r) => r
374 case _ => throw new IllegalArgumentException()
375 }
376 .toSet should ===(Set[Option[ActorRef]](Some(b1), Some(b2)))
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 307 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/actor/SupervisorHierarchySpec.scala:307
Taint Flags:

```
304 */
305 val name = ref.path.name
306 if (pongsToGo == 0) {
307 if (!context.child(name).exists(_ != ref)) {
308 listener ! Died(ref.path)
309 val kids = stateCache.get(self.path).kids(ref.path)
310 val props = Props(new Hierarchy(kids, breadth, listener, myLevel + 1, random)).withDispatcher("hierarchy")
```

scala/akka/actor/ActorSelectionSpec.scala, line 125 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Code Correctness: Class Does Not Implement equals — Low

Package: scala.akka.actor

scala/akka/actor/ActorSelectionSpec.scala, line 125 (Code Correctness: Class Does Not Implement equals) — Low

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/actor/ActorSelectionSpec.scala:125
Taint Flags:

```
122
123  // wait till path is freed
124  awaitCond {
125  system.asInstanceOf[ExtendedActorSystem].provider.resolveActorRef(a1.path) != a1
126  }
127
128  val a2 = system.actorOf(p, name)
```

Package: scala.akka.actor.dispatch

scala/akka/actor/dispatch/DispatchersSpec.scala, line 116 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/actor/dispatch/DispatchersSpec.scala:116
Taint Flags:

```
113  val throughput = "throughput"
114  val id = "id"
115
116  def instance(dispatcher: MessageDispatcher): MessageDispatcher => Boolean = _ == dispatcher
117  def ofType[T <: MessageDispatcher: ClassTag]: MessageDispatcher => Boolean =
118  _.getClass == implicitly[ClassTag[T]].runtimeClass
119
```

Package: scala.akka.dispatch

scala/akka/dispatch/ForkJoinPoolStarvationSpec.scala, line 55 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Code Correctness: Class Does Not Implement equals — Low

Package: scala.akka.dispatch

scala/akka/dispatch/ForkJoinPoolStarvationSpec.scala, line 55 (Code Correctness: Class Does Not Implement equals) — Low

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/dispatch/ForkJoinPoolStarvationSpec.scala:55
Taint Flags:

```
52
53 "not starve tasks arriving from external dispatchers under high internal traffic" in {
54 // TODO issue #31117: starvation with JDK 17 FJP
55 if (System.getProperty("java.specification.version") == "17")
56 pending
57
58 // Two busy actors that will occupy the threads of the dispatcher
```

Package: scala.akka.dispatch.sysmsg

scala/akka/dispatch/sysmsg/SystemMessageListSpec.scala, line 114 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/dispatch/sysmsg/SystemMessageListSpec.scala:114
Taint Flags:

```
111 (list.tail.tail.tail.tail.tail.head eq create5) should ===(true)
112 (list.tail.tail.tail.tail.tail.tail.head eq null) should ===(true)
113
114 ENil.reversePrepend(LNil) == ENil should ===(true)
115 (ENil.reversePrepend(create0 :: LNil).head eq create0) should ===(true)
116 ((create0 :: ENil).reversePrepend(LNil).head eq create0) should ===(true)
117 }
```

scala/akka/dispatch/sysmsg/SystemMessageListSpec.scala, line 21 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()

Code Correctness: Class Does Not Implement equals — Low

Package: scala.akka.dispatch.sysmsg

scala/akka/dispatch/sysmsg/SystemMessageListSpec.scala, line 21 (Code Correctness: Class Does Not Implement equals) — Low

File: scala/akka/dispatch/sysmsg/SystemMessageListSpec.scala:21
Taint Flags:

```
18 "handle empty lists correctly" in {
19 LNil.head should ===(null)
20 LNil.isEmpty should ===(true)
21 (LNil.reverse == ENil) should ===(true)
22 }
23
24 "able to append messages" in {
```

Package: scala.akka.io.dns.internal

scala/akka/io/dns/internal/AsyncDnsManagerSpec.scala, line 41 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/io/dns/internal/AsyncDnsManagerSpec.scala:41
Taint Flags:

```
38 "support ipv6" in {
39 dns ! Resolve("::1") // ::1 will short circuit the resolution
40 expectMsgType[Resolved] match {
41 case Resolved("::1", Seq(AAAARecord("::1", Ttl.effectivelyForever, _)), Nil) =>
42 case other => fail(other.toString)
43 }
44 }
```

Package: scala.akka.pattern

scala/akka/pattern/StatusReplySpec.scala, line 32 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()

Code Correctness: Class Does Not Implement equals

Low

Package: scala.akka.pattern

scala/akka/pattern/StatusReplySpec.scala, line 32 (Code Correctness: Class Does Not Implement equals)

Low

File: scala/akka/pattern/StatusReplySpec.scala:32
Taint Flags:

```
29  "pattern match success (Ack)" in {
30  // like in a classic actor receive Any => ...
31  (StatusReply.Ack: Any) match {
32  case StatusReply.Ack =>
33  case _ => fail()
34  }
35  }
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 660 (Code Correctness: Class Does Not Implement equals)

Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/pattern/CircuitBreakerSpec.scala:660
Taint Flags:

```
657  val harmlessException = new TestException
658  val harmlessExceptionAsSuccess: Try[String] => Boolean = {
659  case Success(_) => false
660  case Failure(ex) => ex != harmlessException
661  }
662
663  breaker().withCircuitBreaker(Future(throw harmlessException), harmlessExceptionAsSuccess)
```

scala/akka/pattern/StatusReplySpec.scala, line 25 (Code Correctness: Class Does Not Implement equals)

Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/pattern/StatusReplySpec.scala:25
Taint Flags:

```
22  // like in a classic actor receive Any => ...
```

Code Correctness: Class Does Not Implement equals — Low

Package: scala.akka.pattern

scala/akka/pattern/StatusReplySpec.scala, line 25 (Code Correctness: Class Does Not Implement equals) — Low

```
23 (StatusReply.Success("woho!"): Any) match {
24 case StatusReply.Success(_: Int) => fail()
25 case StatusReply.Success(text: String) if text == "woho!" =>
26 case _ => fail()
27 }
28 }
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 162 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/pattern/CircuitBreakerSpec.scala:162
Taint Flags:

```
159 val breaker = shortResetTimeoutCb()
160 intercept[TestException] { breaker().withSyncCircuitBreaker(throwException) }
161 checkLatch(breaker.halfOpenLatch)
162 assert("hi" == breaker().withSyncCircuitBreaker(sayHi))
163 checkLatch(breaker.closedLatch)
164 }
165
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 343 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/pattern/CircuitBreakerSpec.scala:343
Taint Flags:

```
340 val harmlessException = new TestException
341 val harmlessExceptionAsSuccess: Try[String] => Boolean = {
342 case Success(_) => false
343 case Failure(ex) => ex != harmlessException
```

Code Correctness: Class Does Not Implement equals — Low

Package: scala.akka.pattern

scala/akka/pattern/CircuitBreakerSpec.scala, line 343 (Code Correctness: Class Does Not Implement equals) — Low

```
344 }
345
346 intercept[TestException] {
```

Package: scala.akka.util

scala/akka/util/ByteStringSpec.scala, line 789 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/util/ByteStringSpec.scala:789
Taint Flags:

```
786 "behave as expected" when {
787 "created from and decoding to String" in {
788 check { (s: String) =>
789 ByteString(s, "UTF-8").decodeString("UTF-8") == s
790 }
791 }
792
```

scala/akka/util/DoubleLinkedListSpec.scala, line 118 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/util/DoubleLinkedListSpec.scala:118
Taint Flags:

```
115 check(list, List("a", "b"))
116 list.getLastOrElseAppend(_.value == "b", c) shouldBe b
117 check(list, List("a", "b"))
118 list.getLastOrElseAppend(_.value == "c", c) shouldBe c
119 check(list, List("a", "b", "c"))
120 }
```

Code Correctness: Class Does Not Implement equals — Low

Package: scala.akka.util

scala/akka/util/DoubleLinkedListSpec.scala, line 118 (Code Correctness: Class Does Not Implement equals) — Low

```
121
```

scala/akka/util/DoubleLinkedListSpec.scala, line 130 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/util/DoubleLinkedListSpec.scala:130
Taint Flags:

```
127 list.prepend(c)
128 list.prepend(a)
129 check(list, List("a", "c"))
130 list.getNextOrElseInsert(a, _.value == "c", b) shouldBe c
131 check(list, List("a", "c"))
132 list.getNextOrElseInsert(a, _.value == "b", b) shouldBe b
133 check(list, List("a", "b", "c"))
```

scala/akka/util/DurationSpec.scala, line 46 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/util/DurationSpec.scala:46
Taint Flags:

```
43 (minf - inf) should ===(minf)
44 (minf + minf) should ===(minf)
45 assert(inf == inf)
46 assert(minf == minf)
47 inf.compareTo(inf) should ===(0)
48 inf.compareTo(one) should ===(1)
49 minf.compareTo(minf) should ===(0)
```

Code Correctness: Class Does Not Implement equals — Low

Package: scala.akka.util

scala/akka/util/DurationSpec.scala, line 54 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/util/DurationSpec.scala:54
Taint Flags:

```
51  assert(inf != minf)
52  assert(minf != inf)
53  assert(one != inf)
54  assert(minf != one)
55  }
56
57  /*"check its range" in {
```

scala/akka/util/DurationSpec.scala, line 53 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/util/DurationSpec.scala:53
Taint Flags:

```
50  minf.compareTo(one) should ===(-1)
51  assert(inf != minf)
52  assert(minf != inf)
53  assert(one != inf)
54  assert(minf != one)
55  }
56
```

scala/akka/util/DurationSpec.scala, line 45 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Code Correctness: Class Does Not Implement equals — Low

Package: scala.akka.util

scala/akka/util/DurationSpec.scala, line 45 (Code Correctness: Class Does Not Implement equals) — Low

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/util/DurationSpec.scala:45
Taint Flags:

```
42 (inf - minf) should ===(inf)
43 (minf - inf) should ===(minf)
44 (minf + minf) should ===(minf)
45 assert(inf == inf)
46 assert(minf == minf)
47 inf.compareTo(inf) should ===(0)
48 inf.compareTo(one) should ===(1)
```

scala/akka/util/DoubleLinkedListSpec.scala, line 102 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/util/DoubleLinkedListSpec.scala:102
Taint Flags:

```
99 list.prepend(c)
100 list.prepend(b)
101 check(list, List("b", "c"))
102 list.getFirstOrElsePrepend(_.value == "b", a) shouldBe b
103 check(list, List("b", "c"))
104 list.getFirstOrElsePrepend(_.value == "a", a) shouldBe a
105 check(list, List("a", "b", "c"))
```

scala/akka/util/DoubleLinkedListSpec.scala, line 116 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()

Code Correctness: Class Does Not Implement equals — Low

Package: scala.akka.util

scala/akka/util/DoubleLinkedListSpec.scala, line 116 (Code Correctness: Class Does Not Implement equals) — Low

File: scala/akka/util/DoubleLinkedListSpec.scala:116
Taint Flags:

```
113 list.append(a)
114 list.append(b)
115 check(list, List("a", "b"))
116 list.getLastOrElseAppend(_.value == "b", c) shouldBe b
117 check(list, List("a", "b"))
118 list.getLastOrElseAppend(_.value == "c", c) shouldBe c
119 check(list, List("a", "b", "c"))
```

scala/akka/util/DoubleLinkedListSpec.scala, line 132 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/util/DoubleLinkedListSpec.scala:132
Taint Flags:

```
129 check(list, List("a", "c"))
130 list.getNextOrElseInsert(a, _.value == "c", b) shouldBe c
131 check(list, List("a", "c"))
132 list.getNextOrElseInsert(a, _.value == "b", b) shouldBe b
133 check(list, List("a", "b", "c"))
134 }
135
```

scala/akka/util/DoubleLinkedListSpec.scala, line 146 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/util/DoubleLinkedListSpec.scala:146
Taint Flags:

```
143 check(list, List("a", "c"))
```

Code Correctness: Class Does Not Implement equals — Low

Package: scala.akka.util

scala/akka/util/DoubleLinkedListSpec.scala, line 146 (Code Correctness: Class Does Not Implement equals) — Low

```
144 list.getPreviousOrElseInsert(c, _.value == "a", b) shouldBe a
145 check(list, List("a", "c"))
146 list.getPreviousOrElseInsert(c, _.value == "b", b) shouldBe b
147 check(list, List("a", "b", "c"))
148 }
149
```

scala/akka/util/DoubleLinkedListSpec.scala, line 104 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/util/DoubleLinkedListSpec.scala:104
Taint Flags:

```
101 check(list, List("b", "c"))
102 list.getFirstOrElsePrepend(_.value == "b", a) shouldBe b
103 check(list, List("b", "c"))
104 list.getFirstOrElsePrepend(_.value == "a", a) shouldBe a
105 check(list, List("a", "b", "c"))
106 }
107
```

scala/akka/util/DoubleLinkedListSpec.scala, line 144 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/util/DoubleLinkedListSpec.scala:144
Taint Flags:

```
141 list.append(a)
142 list.append(c)
143 check(list, List("a", "c"))
144 list.getPreviousOrElseInsert(c, _.value == "a", b) shouldBe a
```

Code Correctness: Class Does Not Implement equals — Low

Package: scala.akka.util

scala/akka/util/DoubleLinkedListSpec.scala, line 144 (Code Correctness: Class Does Not Implement equals) — Low

```
145 check(list, List("a", "c"))
146 list.getPreviousOrElseInsert(c, _.value == "b", b) shouldBe b
147 check(list, List("a", "b", "c"))
```

scala/akka/util/DurationSpec.scala, line 52 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/util/DurationSpec.scala:52
Taint Flags:

```
49 minf.compareTo(minf) should ===(0)
50 minf.compareTo(one) should ===(-1)
51 assert(inf != minf)
52 assert(minf != inf)
53 assert(one != inf)
54 assert(minf != one)
55 }
```

scala/akka/util/DurationSpec.scala, line 51 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/util/DurationSpec.scala:51
Taint Flags:

```
48 inf.compareTo(one) should ===(1)
49 minf.compareTo(minf) should ===(0)
50 minf.compareTo(one) should ===(-1)
51 assert(inf != minf)
52 assert(minf != inf)
53 assert(one != inf)
54 assert(minf != one)
```

Code Correctness: Constructor Invokes Overridable Function (114 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.

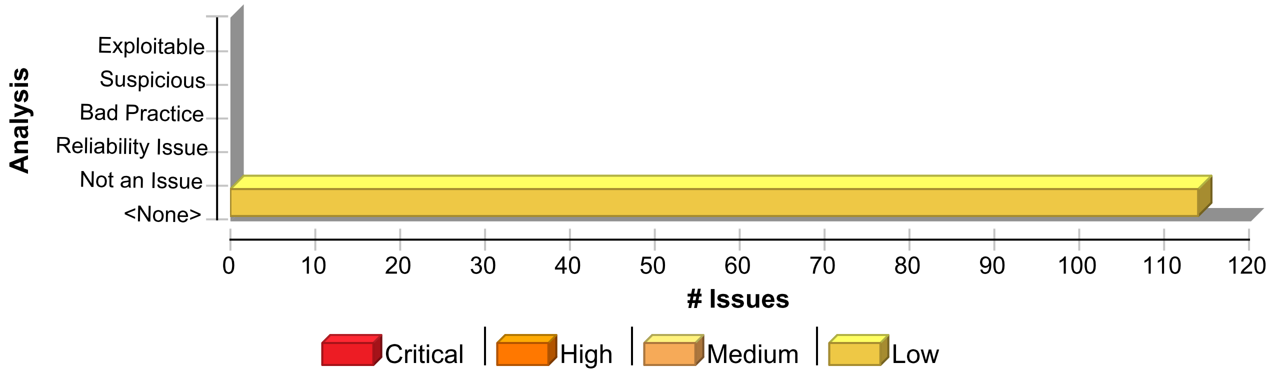
Recommendation

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	114	0	0	114
Total	**114**	**0**	**0**	**114**

Code Correctness: Constructor Invokes Overridable Function	**Low**
Package: akka.actor	
scala/akka/actor/ActorCreationPerfSpec.scala, line 129 (Code Correctness: Constructor Invokes Overridable Function)	**Low**

Issue Details

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/ActorCreationPerfSpec.scala, line 129 (Code Correctness: Constructor Invokes Overridable Function) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: ActorCreationKey
Enclosing Method: ActorCreationPerfSpec()
File: scala/akka/actor/ActorCreationPerfSpec.scala:129
Taint Flags:

```
126
127  def metricsConfig = system.settings.config
128  val ActorCreationKey = MetricKey.fromString("actor-creation")
129  val BlockingTimeKey = ActorCreationKey / "synchronous-part"
130  val TotalTimeKey = ActorCreationKey / "total"
131
132  val warmUp = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.warmUp")
```

scala/akka/actor/ActorCreationPerfSpec.scala, line 130 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: ActorCreationKey
Enclosing Method: ActorCreationPerfSpec()
File: scala/akka/actor/ActorCreationPerfSpec.scala:130
Taint Flags:

```
127  def metricsConfig = system.settings.config
128  val ActorCreationKey = MetricKey.fromString("actor-creation")
129  val BlockingTimeKey = ActorCreationKey / "synchronous-part"
130  val TotalTimeKey = ActorCreationKey / "total"
131
132  val warmUp = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.warmUp")
133  val nrOfActors = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.numberOfActors")
```

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 71 (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.actor

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 71 (Code Correctness: Constructor Invokes Overridable Function) — Low

Sink Details

Sink: FunctionCall: aliasedDispatcherId2
Enclosing Method: ActorWithBoundedStashSpec()
File: scala/akka/actor/ActorWithBoundedStashSpec.scala:71
Taint Flags:

```
68  val mailboxId1 = "my-mailbox-1"
69  val mailboxId2 = "my-mailbox-2"
70
71  val testConf: Config = ConfigFactory.parseString(s"""
72  $dispatcherId1 {
73  mailbox-type = "${classOf[Bounded10].getName}"
74  stash-capacity = 20
```

scala/akka/actor/ActorCreationPerfSpec.scala, line 132 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: metricsConfig
Enclosing Method: ActorCreationPerfSpec()
File: scala/akka/actor/ActorCreationPerfSpec.scala:132
Taint Flags:

```
129  val BlockingTimeKey = ActorCreationKey / "synchronous-part"
130  val TotalTimeKey = ActorCreationKey / "total"
131
132  val warmUp = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.warmUp")
133  val nrOfActors = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.numberOfActors")
134  val nrOfRepeats = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.numberOfRepeats")
135  override val reportMetricsEnabled = metricsConfig.getBoolean("akka.test.actor.ActorPerfSpec.report-metrics")
```

scala/akka/actor/ActorCreationPerfSpec.scala, line 133 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: metricsConfig
Enclosing Method: ActorCreationPerfSpec()

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/ActorCreationPerfSpec.scala, line 133 (Code Correctness: Constructor Invokes Overridable Function) — Low

File: scala/akka/actor/ActorCreationPerfSpec.scala:133
Taint Flags:

```
130  val TotalTimeKey = ActorCreationKey / "total"
131
132  val warmUp = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.warmUp")
133  val nrOfActors = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.numberOfActors")
134  val nrOfRepeats = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.numberOfRepeats")
135  override val reportMetricsEnabled = metricsConfig.getBoolean("akka.test.actor.ActorPerfSpec.report-metrics")
136  override val forceGcEnabled = metricsConfig.getBoolean("akka.test.actor.ActorPerfSpec.force-gc")
```

scala/akka/actor/ActorCreationPerfSpec.scala, line 134 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: metricsConfig
Enclosing Method: ActorCreationPerfSpec()
File: scala/akka/actor/ActorCreationPerfSpec.scala:134
Taint Flags:

```
131
132  val warmUp = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.warmUp")
133  val nrOfActors = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.numberOfActors")
134  val nrOfRepeats = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.numberOfRepeats")
135  override val reportMetricsEnabled = metricsConfig.getBoolean("akka.test.actor.ActorPerfSpec.report-metrics")
136  override val forceGcEnabled = metricsConfig.getBoolean("akka.test.actor.ActorPerfSpec.force-gc")
137
```

scala/akka/actor/ActorCreationPerfSpec.scala, line 135 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: metricsConfig
Enclosing Method: ActorCreationPerfSpec()
File: scala/akka/actor/ActorCreationPerfSpec.scala:135
Taint Flags:

```
132  val warmUp = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.warmUp")
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/ActorCreationPerfSpec.scala, line 135 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
133  val nrOfActors = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.numberOfActors")
134  val nrOfRepeats = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.numberOfRepeats")
135  override val reportMetricsEnabled = metricsConfig.getBoolean("akka.test.actor.ActorPerfSpec.report-metrics")
136  override val forceGcEnabled = metricsConfig.getBoolean("akka.test.actor.ActorPerfSpec.force-gc")
137
138  def runWithCounterInside(metricName: String, scenarioName: String, number: Int, propsCreator: () => Props): Unit = {
```

scala/akka/actor/ActorCreationPerfSpec.scala, line 136 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: metricsConfig
Enclosing Method: ActorCreationPerfSpec()
File: scala/akka/actor/ActorCreationPerfSpec.scala:136
Taint Flags:

```
133  val nrOfActors = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.numberOfActors")
134  val nrOfRepeats = metricsConfig.getInt("akka.test.actor.ActorPerfSpec.numberOfRepeats")
135  override val reportMetricsEnabled = metricsConfig.getBoolean("akka.test.actor.ActorPerfSpec.report-metrics")
136  override val forceGcEnabled = metricsConfig.getBoolean("akka.test.actor.ActorPerfSpec.force-gc")
137
138  def runWithCounterInside(metricName: String, scenarioName: String, number: Int, propsCreator: () => Props): Unit = {
139  val hist = histogram(BlockingTimeKey / metricName)
```

scala/akka/actor/ActorCreationPerfSpec.scala, line 120 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: ActorCreationPerfSpec()
File: scala/akka/actor/ActorCreationPerfSpec.scala:120
Taint Flags:

```
117 }
118
119 class ActorCreationPerfSpec
120  extends AkkaSpec(ActorCreationPerfSpec.config)
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/ActorCreationPerfSpec.scala, line 120 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
121  with ImplicitSender
122  with MetricsKit
123  with BeforeAndAfterAll {
```

scala/akka/actor/ActorSelectionSpec.scala, line 41 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: p
Enclosing Method: ActorSelectionSpec()
File: scala/akka/actor/ActorSelectionSpec.scala:41
Taint Flags:

```
38 class ActorSelectionSpec extends AkkaSpec with DefaultTimeout {
39  import ActorSelectionSpec._
40
41  val c1 = system.actorOf(p, "c1")
42  val c2 = system.actorOf(p, "c2")
43  val c21 = Await.result((c2 ? Create("c21")).mapTo[ActorRef], timeout.duration)
44
```

scala/akka/actor/ActorSelectionSpec.scala, line 42 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: p
Enclosing Method: ActorSelectionSpec()
File: scala/akka/actor/ActorSelectionSpec.scala:42
Taint Flags:

```
39  import ActorSelectionSpec._
40
41  val c1 = system.actorOf(p, "c1")
42  val c2 = system.actorOf(p, "c2")
43  val c21 = Await.result((c2 ? Create("c21")).mapTo[ActorRef], timeout.duration)
44
45  val sysImpl = system.asInstanceOf[ActorSystemImpl]
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/ConsistencySpec.scala, line 18 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: factor
Enclosing Method: ConsistencySpec()
File: scala/akka/actor/ConsistencySpec.scala:18
Taint Flags:

```
15 val minThreads = 1
16 val maxThreads = 2000
17 val factor = 1.5d
18 val threads = ThreadPoolConfig.scaledPoolSize(minThreads, factor, maxThreads) // Make sure we have more threads than cores
19
20 val config = s"""
21 consistency-dispatcher {
```

scala/akka/actor/ConsistencySpec.scala, line 20 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: factor
Enclosing Method: ConsistencySpec()
File: scala/akka/actor/ConsistencySpec.scala:20
Taint Flags:

```
17 val factor = 1.5d
18 val threads = ThreadPoolConfig.scaledPoolSize(minThreads, factor, maxThreads) // Make sure we have more threads than cores
19
20 val config = s"""
21 consistency-dispatcher {
22 throughput = 1
23 executor = "fork-join-executor"
```

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 71 (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.actor

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 71 (Code Correctness: Constructor Invokes Overridable Function) — Low

Sink Details

Sink: FunctionCall: aliasedDispatcherId1
Enclosing Method: ActorWithBoundedStashSpec()
File: scala/akka/actor/ActorWithBoundedStashSpec.scala:71
Taint Flags:

```
68 val mailboxId1 = "my-mailbox-1"
69 val mailboxId2 = "my-mailbox-2"
70
71 val testConf: Config = ConfigFactory.parseString(s"""
72 $dispatcherId1 {
73 mailbox-type = "${classOf[Bounded10].getName}"
74 stash-capacity = 20
```

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 71 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: aliasedDispatcherId1
Enclosing Method: ActorWithBoundedStashSpec()
File: scala/akka/actor/ActorWithBoundedStashSpec.scala:71
Taint Flags:

```
68 val mailboxId1 = "my-mailbox-1"
69 val mailboxId2 = "my-mailbox-2"
70
71 val testConf: Config = ConfigFactory.parseString(s"""
72 $dispatcherId1 {
73 mailbox-type = "${classOf[Bounded10].getName}"
74 stash-capacity = 20
```

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 94 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: testConf
Enclosing Method: ActorWithBoundedStashSpec()

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 94 (Code Correctness: Constructor Invokes Overridable Function) — Low

File: scala/akka/actor/ActorWithBoundedStashSpec.scala:94
Taint Flags:

```
91 }
92
93 class ActorWithBoundedStashSpec
94  extends AkkaSpec(ActorWithBoundedStashSpec.testConf)
95  with BeforeAndAfterEach
96  with DefaultTimeout
97  with ImplicitSender {
```

scala/akka/actor/ActorConfigurationVerificationSpec.scala, line 39 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: ActorConfigurationVerificationSpec()
File: scala/akka/actor/ActorConfigurationVerificationSpec.scala:39
Taint Flags:

```
36 }
37
38 class ActorConfigurationVerificationSpec
39  extends AkkaSpec(ActorConfigurationVerificationSpec.config)
40  with DefaultTimeout
41  with BeforeAndAfterEach {
42  import ActorConfigurationVerificationSpec._
```

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 71 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: mailboxId1
Enclosing Method: ActorWithBoundedStashSpec()
File: scala/akka/actor/ActorWithBoundedStashSpec.scala:71
Taint Flags:

```
68  val mailboxId1 = "my-mailbox-1"
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 71 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
69  val mailboxId2 = "my-mailbox-2"
70
71  val testConf: Config = ConfigFactory.parseString(s"""
72  $dispatcherId1 {
73  mailbox-type = "${classOf[Bounded10].getName}"
74  stash-capacity = 20
```

scala/akka/actor/SupervisorSpec.scala, line 108 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: SupervisorSpec()
File: scala/akka/actor/SupervisorSpec.scala:108
Taint Flags:

```
105 }
106
107 class SupervisorSpec
108  extends AkkaSpec(SupervisorSpec.config)
109  with BeforeAndAfterEach
110  with ImplicitSender
111  with DefaultTimeout {
```

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 71 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: dispatcherId1
Enclosing Method: ActorWithBoundedStashSpec()
File: scala/akka/actor/ActorWithBoundedStashSpec.scala:71
Taint Flags:

```
68  val mailboxId1 = "my-mailbox-1"
69  val mailboxId2 = "my-mailbox-2"
70
71  val testConf: Config = ConfigFactory.parseString(s"""
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 71 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
72  $dispatcherId1 {
73  mailbox-type = "${classOf[Bounded10].getName}"
74  stash-capacity = 20
```

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 71 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: dispatcherId1
Enclosing Method: ActorWithBoundedStashSpec()
File: scala/akka/actor/ActorWithBoundedStashSpec.scala:71
Taint Flags:

```
68  val mailboxId1 = "my-mailbox-1"
69  val mailboxId2 = "my-mailbox-2"
70
71  val testConf: Config = ConfigFactory.parseString(s"""
72  $dispatcherId1 {
73  mailbox-type = "${classOf[Bounded10].getName}"
74  stash-capacity = 20
```

scala/akka/actor/ActorSelectionSpec.scala, line 47 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: sysImpl
Enclosing Method: ActorSelectionSpec()
File: scala/akka/actor/ActorSelectionSpec.scala:47
Taint Flags:

```
44
45  val sysImpl = system.asInstanceOf[ActorSystemImpl]
46
47  val user = sysImpl.guardian
48  val syst = sysImpl.systemGuardian
49  val root = sysImpl.lookupRoot
50
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/ActorSelectionSpec.scala, line 48 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: sysImpl
Enclosing Method: ActorSelectionSpec()
File: scala/akka/actor/ActorSelectionSpec.scala:48
Taint Flags:

```
45  val sysImpl = system.asInstanceOf[ActorSystemImpl]
46
47  val user = sysImpl.guardian
48  val syst = sysImpl.systemGuardian
49  val root = sysImpl.lookupRoot
50
51  def empty(path: String) =
```

scala/akka/actor/ActorSelectionSpec.scala, line 49 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: sysImpl
Enclosing Method: ActorSelectionSpec()
File: scala/akka/actor/ActorSelectionSpec.scala:49
Taint Flags:

```
46
47  val user = sysImpl.guardian
48  val syst = sysImpl.systemGuardian
49  val root = sysImpl.lookupRoot
50
51  def empty(path: String) =
52  new EmptyLocalActorRef(sysImpl.provider, path match {
```

scala/akka/actor/SupervisorSpec.scala, line 115 (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.actor

scala/akka/actor/SupervisorSpec.scala, line 115 (Code Correctness: Constructor Invokes Overridable Function) — Low

Sink Details

Sink: FunctionCall: Timeout
Enclosing Method: SupervisorSpec()
File: scala/akka/actor/SupervisorSpec.scala:115
Taint Flags:

```
112
113  import SupervisorSpec._
114
115  val DilatedTimeout = Timeout.dilated
116
117  // =====================================================
118  // Creating actors and supervisors
```

scala/akka/actor/ActorSystemSpec.scala, line 118 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: ActorSystemSpec()
File: scala/akka/actor/ActorSystemSpec.scala:118
Taint Flags:

```
115 }
116
117 @nowarn
118 class ActorSystemSpec extends AkkaSpec(ActorSystemSpec.config) with ImplicitSender {
119
120  import ActorSystemSpec.FastActor
121
```

scala/akka/actor/LocalActorRefProviderSpec.scala, line 37 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: LocalActorRefProviderSpec()

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/LocalActorRefProviderSpec.scala, line 37 (Code Correctness: Constructor Invokes Overridable Function) — Low

File: scala/akka/actor/LocalActorRefProviderSpec.scala:37
Taint Flags:

```
34 }
35
36 @nowarn
37 class LocalActorRefProviderSpec extends AkkaSpec(LocalActorRefProviderSpec.config) {
38  "An LocalActorRefProvider" must {
39
40  "find child actor with URL encoded name" in {
```

scala/akka/actor/DeadLetterSupressionSpec.scala, line 27 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: deadActor
Enclosing Method: DeadLetterSupressionSpec()
File: scala/akka/actor/DeadLetterSupressionSpec.scala:27
Taint Flags:

```
24  import DeadLetterSupressionSpec._
25
26  val deadActor = system.actorOf(TestActors.echoActorProps)
27  watch(deadActor)
28  deadActor ! PoisonPill
29  expectTerminated(deadActor)
30
```

scala/akka/actor/DeadLetterSupressionSpec.scala, line 28 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: deadActor
Enclosing Method: DeadLetterSupressionSpec()
File: scala/akka/actor/DeadLetterSupressionSpec.scala:28
Taint Flags:

```
25
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/DeadLetterSupressionSpec.scala, line 28 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
26 val deadActor = system.actorOf(TestActors.echoActorProps)
27 watch(deadActor)
28 deadActor ! PoisonPill
29 expectTerminated(deadActor)
30
31 s"must suppress message from default dead-letters logging (sent to dead: ${Logging.simpleName(deadActor)})" in {
```

scala/akka/actor/DeadLetterSupressionSpec.scala, line 29 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: deadActor
Enclosing Method: DeadLetterSupressionSpec()
File: scala/akka/actor/DeadLetterSupressionSpec.scala:29
Taint Flags:

```
26 val deadActor = system.actorOf(TestActors.echoActorProps)
27 watch(deadActor)
28 deadActor ! PoisonPill
29 expectTerminated(deadActor)
30
31 s"must suppress message from default dead-letters logging (sent to dead: ${Logging.simpleName(deadActor)})" in {
32 val deadListener = TestProbe()
```

scala/akka/actor/DeadLetterSupressionSpec.scala, line 31 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: deadActor
Enclosing Method: DeadLetterSupressionSpec()
File: scala/akka/actor/DeadLetterSupressionSpec.scala:31
Taint Flags:

```
28 deadActor ! PoisonPill
29 expectTerminated(deadActor)
30
31 s"must suppress message from default dead-letters logging (sent to dead: ${Logging.simpleName(deadActor)})" in {
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/DeadLetterSupressionSpec.scala, line 31 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
32 val deadListener = TestProbe()
33 system.eventStream.subscribe(deadListener.ref, classOf[DeadLetter])
34
```

scala/akka/actor/ConsistencySpec.scala, line 18 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: maxThreads
Enclosing Method: ConsistencySpec()
File: scala/akka/actor/ConsistencySpec.scala:18
Taint Flags:

```
15 val minThreads = 1
16 val maxThreads = 2000
17 val factor = 1.5d
18 val threads = ThreadPoolConfig.scaledPoolSize(minThreads, factor, maxThreads) // Make sure we have more threads than cores
19
20 val config = s"""
21 consistency-dispatcher {
```

scala/akka/actor/ConsistencySpec.scala, line 20 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: maxThreads
Enclosing Method: ConsistencySpec()
File: scala/akka/actor/ConsistencySpec.scala:20
Taint Flags:

```
17 val factor = 1.5d
18 val threads = ThreadPoolConfig.scaledPoolSize(minThreads, factor, maxThreads) // Make sure we have more threads than cores
19
20 val config = s"""
21 consistency-dispatcher {
22 throughput = 1
23 executor = "fork-join-executor"
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/SupervisorMiscSpec.scala, line 33 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: SupervisorMiscSpec()
File: scala/akka/actor/SupervisorMiscSpec.scala:33
Taint Flags:

```
30 }
31
32 @nowarn
33 class SupervisorMiscSpec extends AkkaSpec(SupervisorMiscSpec.config) with DefaultTimeout {
34
35  "A Supervisor" must {
36
```

scala/akka/actor/ActorMailboxSpec.scala, line 232 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: mailboxConf
Enclosing Method: ActorMailboxSpec()
File: scala/akka/actor/ActorMailboxSpec.scala:232
Taint Flags:

```
229
230  import ActorMailboxSpec._
231
232  def this() = this(ActorMailboxSpec.mailboxConf)
233
234  def checkMailboxQueue(props: Props, name: String, types: Seq[Class[_]]): MessageQueue = {
235  val actor = system.actorOf(props, name)
```

scala/akka/actor/DeadLetterSuspensionSpec.scala, line 44 (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.actor

scala/akka/actor/DeadLetterSuspensionSpec.scala, line 44 (Code Correctness: Constructor Invokes Overridable Function) — Low

Sink Details

Sink: FunctionCall: deadActor
Enclosing Method: DeadLetterSuspensionSpec()
File: scala/akka/actor/DeadLetterSuspensionSpec.scala:44
Taint Flags:

```
41 import DeadLetterSuspensionSpec._
42
43 private val deadActor = system.actorOf(TestActors.echoActorProps)
44 watch(deadActor)
45 deadActor ! PoisonPill
46 expectTerminated(deadActor)
47
```

scala/akka/actor/DeadLetterSuspensionSpec.scala, line 45 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: deadActor
Enclosing Method: DeadLetterSuspensionSpec()
File: scala/akka/actor/DeadLetterSuspensionSpec.scala:45
Taint Flags:

```
42
43 private val deadActor = system.actorOf(TestActors.echoActorProps)
44 watch(deadActor)
45 deadActor ! PoisonPill
46 expectTerminated(deadActor)
47
48 private val droppingActor = system.actorOf(Dropping.props(), "droppingActor")
```

scala/akka/actor/DeadLetterSuspensionSpec.scala, line 46 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: deadActor
Enclosing Method: DeadLetterSuspensionSpec()

Code Correctness: Constructor Invokes Overridable Function Low

Package: akka.actor

scala/akka/actor/DeadLetterSuspensionSpec.scala, line 46 (Code Correctness: Constructor Invokes Overridable Function) Low

File: scala/akka/actor/DeadLetterSuspensionSpec.scala:46
Taint Flags:

```
43  private val deadActor = system.actorOf(TestActors.echoActorProps)
44  watch(deadActor)
45  deadActor ! PoisonPill
46  expectTerminated(deadActor)
47
48  private val droppingActor = system.actorOf(Dropping.props(), "droppingActor")
49  private val unhandledActor = system.actorOf(Unandled.props(), "unhandledActor")
```

scala/akka/actor/ExtensionSpec.scala, line 42 (Code Correctness: Constructor Invokes Overridable Function) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: createCount
Enclosing Method: InstanceCountingExtension()
File: scala/akka/actor/ExtensionSpec.scala:42
Taint Flags:

```
39 }
40
41 class InstanceCountingExtension extends Extension {
42  InstanceCountingExtension.createCount.incrementAndGet()
43 }
44
45 // Dont't place inside ActorSystemSpec object, since it will not be garbage collected and reference to system remains
```

scala/akka/actor/DeployerSpec.scala, line 81 (Code Correctness: Constructor Invokes Overridable Function) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: deployerConf
Enclosing Method: DeployerSpec()
File: scala/akka/actor/DeployerSpec.scala:81
Taint Flags:

```
78
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/DeployerSpec.scala, line 81 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
79 }
80
81 class DeployerSpec extends AkkaSpec(DeployerSpec.deployerConf) {
82  "A Deployer" must {
83
84  "be able to parse 'akka.actor.deployment._' with all default values" in {
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 750 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: SupervisorHierarchySpec()
File: scala/akka/actor/SupervisorHierarchySpec.scala:750
Taint Flags:

```
747
748 }
749
750 class SupervisorHierarchySpec extends AkkaSpec(SupervisorHierarchySpec.config) with DefaultTimeout with ImplicitSender {
751  import SupervisorHierarchySpec._
752
753  override def expectedTestDuration = 2.minutes
```

scala/akka/actor/FSMTimingSpec.scala, line 17 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: fsm
Enclosing Method: FSMTimingSpec()
File: scala/akka/actor/FSMTimingSpec.scala:17
Taint Flags:

```
14  import FSM._
15
16  val fsm = system.actorOf(Props(new StateMachine(testActor)))
17  fsm ! SubscribeTransitionCallBack(testActor)
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/FSMTimingSpec.scala, line 17 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
18  expectMsg(1 second, CurrentState(fsm, Initial))
19
20  ignoreMsg {
```

scala/akka/actor/FSMTimingSpec.scala, line 18 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: fsm
Enclosing Method: FSMTimingSpec()
File: scala/akka/actor/FSMTimingSpec.scala:18
Taint Flags:

```
15
16  val fsm = system.actorOf(Props(new StateMachine(testActor)))
17  fsm ! SubscribeTransitionCallBack(testActor)
18  expectMsg(1 second, CurrentState(fsm, Initial))
19
20  ignoreMsg {
21  case Transition(_, bs: FSMTimingSpec.State, _) if bs eq Initial => true // SI-5900 workaround
```

scala/akka/actor/ActorSelectionSpec.scala, line 43 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: c2
Enclosing Method: ActorSelectionSpec()
File: scala/akka/actor/ActorSelectionSpec.scala:43
Taint Flags:

```
40
41  val c1 = system.actorOf(p, "c1")
42  val c2 = system.actorOf(p, "c2")
43  val c21 = Await.result((c2 ? Create("c21")).mapTo[ActorRef], timeout.duration)
44
45  val sysImpl = system.asInstanceOf[ActorSystemImpl]
46
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/ConsistencySpec.scala, line 18 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: minThreads
Enclosing Method: ConsistencySpec()
File: scala/akka/actor/ConsistencySpec.scala:18
Taint Flags:

```
15 val minThreads = 1
16 val maxThreads = 2000
17 val factor = 1.5d
18 val threads = ThreadPoolConfig.scaledPoolSize(minThreads, factor, maxThreads) // Make sure we have more threads than cores
19
20 val config = s"""
21 consistency-dispatcher {
```

scala/akka/actor/ConsistencySpec.scala, line 20 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: minThreads
Enclosing Method: ConsistencySpec()
File: scala/akka/actor/ConsistencySpec.scala:20
Taint Flags:

```
17 val factor = 1.5d
18 val threads = ThreadPoolConfig.scaledPoolSize(minThreads, factor, maxThreads) // Make sure we have more threads than cores
19
20 val config = s"""
21 consistency-dispatcher {
22 throughput = 1
23 executor = "fork-join-executor"
```

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 71 (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.actor

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 71 (Code Correctness: Constructor Invokes Overridable Function) — Low

Sink Details

Sink: FunctionCall: dispatcherId2
Enclosing Method: ActorWithBoundedStashSpec()
File: scala/akka/actor/ActorWithBoundedStashSpec.scala:71
Taint Flags:

```
68  val mailboxId1 = "my-mailbox-1"
69  val mailboxId2 = "my-mailbox-2"
70
71  val testConf: Config = ConfigFactory.parseString(s"""
72  $dispatcherId1 {
73  mailbox-type = "${classOf[Bounded10].getName}"
74  stash-capacity = 20
```

scala/akka/actor/TypedActorSpec.scala, line 253 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: TypedActorSpec()
File: scala/akka/actor/TypedActorSpec.scala:253
Taint Flags:

```
250
251 @nowarn
252 class TypedActorSpec
253  extends AkkaSpec(TypedActorSpec.config)
254  with BeforeAndAfterEach
255  with BeforeAndAfterAll
256  with DefaultTimeout {
```

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 71 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: mailboxId2
Enclosing Method: ActorWithBoundedStashSpec()

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 71 (Code Correctness: Constructor Invokes Overridable Function) — Low

File: scala/akka/actor/ActorWithBoundedStashSpec.scala:71
Taint Flags:

```
68  val mailboxId1 = "my-mailbox-1"
69  val mailboxId2 = "my-mailbox-2"
70
71  val testConf: Config = ConfigFactory.parseString(s"""
72  $dispatcherId1 {
73  mailbox-type = "${classOf[Bounded10].getName}"
74  stash-capacity = 20
```

scala/akka/actor/ConsistencySpec.scala, line 56 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: ConsistencySpec()
File: scala/akka/actor/ConsistencySpec.scala:56
Taint Flags:

```
53  }
54 }
55
56 class ConsistencySpec extends AkkaSpec(ConsistencySpec.config) {
57  import ConsistencySpec._
58
59  override def expectedTestDuration: FiniteDuration = 5.minutes
```

Package: akka.actor.dispatch

scala/akka/actor/dispatch/DispatchersSpec.scala, line 125 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: df
Enclosing Method: DispatchersSpec()
File: scala/akka/actor/dispatch/DispatchersSpec.scala:125
Taint Flags:

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor.dispatch

scala/akka/actor/dispatch/DispatchersSpec.scala, line 125 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
122
123  def validTypes = typesAndValidators.keys.toList
124
125  val defaultDispatcherConfig = settings.config.getConfig("akka.actor.default-dispatcher")
126
127  lazy val allDispatchers: Map[String, MessageDispatcher] = {
128  import akka.util.ccompat.JavaConverters._
```

scala/akka/actor/dispatch/BalancingDispatcherSpec.scala, line 21 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: BalancingDispatcherSpec()
File: scala/akka/actor/dispatch/BalancingDispatcherSpec.scala:21
Taint Flags:

```
18  """
19 }
20
21 class BalancingDispatcherSpec extends AkkaSpec(BalancingDispatcherSpec.config) {
22
23  val delayableActorDispatcher = "pooled-dispatcher"
24
```

scala/akka/actor/dispatch/DispatcherActorSpec.scala, line 58 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: DispatcherActorSpec()
File: scala/akka/actor/dispatch/DispatcherActorSpec.scala:58
Taint Flags:

```
55  }
56 }
57
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor.dispatch

scala/akka/actor/dispatch/DispatcherActorSpec.scala, line 58 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
58 class DispatcherActorSpec extends AkkaSpec(DispatcherActorSpec.config) with DefaultTimeout {
59  import DispatcherActorSpec._
60
61  "A Dispatcher and an Actor" must {
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 619 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: BalancingDispatcherModelSpec()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:619
Taint Flags:

```
616 }
617
618 @nowarn
619 class BalancingDispatcherModelSpec extends ActorModelSpec(BalancingDispatcherModelSpec.config) {
620  import ActorModelSpec._
621
622  val dispatcherCount = new AtomicInteger()
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 550 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: dispatcherType
Enclosing Method: DispatcherModelSpec()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:550
Taint Flags:

```
547
548  override def dispatcherType = "Dispatcher"
549
550  "A " + dispatcherType must {
551  "process messages in parallel" in {
552  val probeA, probeB = TestProbe()
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor.dispatch

scala/akka/actor/dispatch/ActorModelSpec.scala, line 550 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
553  implicit val dispatcher = interceptedDispatcher()
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 536 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: DispatcherModelSpec()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:536
Taint Flags:

```
533  }
534 }
535
536 class DispatcherModelSpec extends ActorModelSpec(DispatcherModelSpec.config) {
537  import ActorModelSpec._
538
539  val dispatcherCount = new AtomicInteger()
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 633 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: dispatcherType
Enclosing Method: BalancingDispatcherModelSpec()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:633
Taint Flags:

```
630
631  override def dispatcherType = "Balancing Dispatcher"
632
633  "A " + dispatcherType must {
634  "process messages in parallel" in {
635  implicit val dispatcher = interceptedDispatcher()
636  val aStart, aStop, bParallel = new CountDownLatch(1)
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor.dispatch

scala/akka/actor/dispatch/DispatchersSpec.scala, line 103 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: DispatchersSpec()
File: scala/akka/actor/dispatch/DispatchersSpec.scala:103
Taint Flags:

```
100  }
101 }
102
103 class DispatchersSpec extends AkkaSpec(DispatchersSpec.config) with ImplicitSender {
104  import DispatchersSpec._
105  val df = system.dispatchers
106  import df._
```

scala/akka/actor/dispatch/PinnedActorSpec.scala, line 34 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: PinnedActorSpec()
File: scala/akka/actor/dispatch/PinnedActorSpec.scala:34
Taint Flags:

```
31  }
32 }
33
34 class PinnedActorSpec extends AkkaSpec(PinnedActorSpec.config) with BeforeAndAfterEach with DefaultTimeout {
35  import PinnedActorSpec._
36
37  "A PinnedActor" must {
```

Package: akka.dispatch

scala/akka/dispatch/MailboxConfigSpec.scala, line 246 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.dispatch

scala/akka/dispatch/MailboxConfigSpec.scala, line 246 (Code Correctness: Constructor Invokes Overridable Function) — Low

Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: CustomMailboxSpec()
File: scala/akka/dispatch/MailboxConfigSpec.scala:246
Taint Flags:

```
243  }
244 }
245
246 class CustomMailboxSpec extends AkkaSpec(CustomMailboxSpec.config) {
247  "Dispatcher configuration" must {
248  "support custom mailboxType" in {
249  val actor = system.actorOf(Props.empty.withDispatcher("my-dispatcher"))
```

scala/akka/dispatch/ControlAwareDispatcherSpec.scala, line 23 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: ControlAwareDispatcherSpec()
File: scala/akka/dispatch/ControlAwareDispatcherSpec.scala:23
Taint Flags:

```
20  case object ImportantMessage extends ControlMessage
21 }
22
23 class ControlAwareDispatcherSpec extends AkkaSpec(ControlAwareDispatcherSpec.config) with DefaultTimeout {
24  import ControlAwareDispatcherSpec.ImportantMessage
25
26  "A ControlAwareDispatcher" must {
```

scala/akka/dispatch/PriorityDispatcherSpec.scala, line 45 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Code Correctness: Constructor Invokes Overridable Function

Low

Package: akka.dispatch

scala/akka/dispatch/PriorityDispatcherSpec.scala, line 45 (Code Correctness: Constructor Invokes Overridable Function)

Low

Sink: FunctionCall: config
Enclosing Method: PriorityDispatcherSpec()
File: scala/akka/dispatch/PriorityDispatcherSpec.scala:45
Taint Flags:

```
42
43 }
44
45 class PriorityDispatcherSpec extends AkkaSpec(PriorityDispatcherSpec.config) with DefaultTimeout {
46  import PriorityDispatcherSpec._
47
48  "A PriorityDispatcher" must {
```

scala/akka/dispatch/MailboxConfigSpec.scala, line 285 (Code Correctness: Constructor Invokes Overridable Function)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: mailboxConf
Enclosing Method: SingleConsumerOnlyMailboxVerificationSpec()
File: scala/akka/dispatch/MailboxConfigSpec.scala:285
Taint Flags:

```
282 }
283
284 class SingleConsumerOnlyMailboxVerificationSpec
285  extends AkkaSpec(SingleConsumerOnlyMailboxVerificationSpec.mailboxConf) {
286  import SingleConsumerOnlyMailboxVerificationSpec.Ping
287
288  def pathologicalPingPong(dispatcherId: String): Unit = {
```

scala/akka/dispatch/ForkJoinPoolStarvationSpec.scala, line 46 (Code Correctness: Constructor Invokes Overridable Function)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: ForkJoinPoolStarvationSpec()
File: scala/akka/dispatch/ForkJoinPoolStarvationSpec.scala:46
Taint Flags:

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.dispatch

scala/akka/dispatch/ForkJoinPoolStarvationSpec.scala, line 46 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
43
44 }
45
46 class ForkJoinPoolStarvationSpec extends AkkaSpec(ForkJoinPoolStarvationSpec.config) with ImplicitSender {
47  import ForkJoinPoolStarvationSpec._
48
49  val Iterations = 1000
```

scala/akka/dispatch/StablePriorityDispatcherSpec.scala, line 46 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: StablePriorityDispatcherSpec()
File: scala/akka/dispatch/StablePriorityDispatcherSpec.scala:46
Taint Flags:

```
43
44 }
45
46 class StablePriorityDispatcherSpec extends AkkaSpec(StablePriorityDispatcherSpec.config) with DefaultTimeout {
47  import StablePriorityDispatcherSpec._
48
49  "A StablePriorityDispatcher" must {
```

Package: akka.event

scala/akka/event/EventStreamSpec.scala, line 33 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: configUnhandled
Enclosing Method: EventStreamSpec()
File: scala/akka/event/EventStreamSpec.scala:33
Taint Flags:

```
30  }
31  """)
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.event

scala/akka/event/EventStreamSpec.scala, line 33 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
32
33  val configUnhandledWithDebug =
34  ConfigFactory.parseString("akka.actor.debug.event-stream = on").withFallback(configUnhandled)
35
36  final case class M(i: Int)
```

scala/akka/event/EventStreamSpec.scala, line 68 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: EventStreamSpec()
File: scala/akka/event/EventStreamSpec.scala:68
Taint Flags:

```
65  class CCATBT extends CC with ATT with BTT
66 }
67
68 class EventStreamSpec extends AkkaSpec(EventStreamSpec.config) {
69
70  import EventStreamSpec._
71
```

scala/akka/event/LoggingReceiveSpec.scala, line 47 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: appLogging
Enclosing Method: LoggingReceiveSpec()
File: scala/akka/event/LoggingReceiveSpec.scala:47
Taint Flags:

```
44  case _: Logging.Info => true
45  case _ => false
46  })
47  appLogging.eventStream.publish(filter)
48  appAuto.eventStream.publish(filter)
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.event

scala/akka/event/LoggingReceiveSpec.scala, line 47 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
49 appLifecycle.eventStream.publish(filter)
50
```

scala/akka/event/LoggingReceiveSpec.scala, line 47 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: filter
Enclosing Method: LoggingReceiveSpec()
File: scala/akka/event/LoggingReceiveSpec.scala:47
Taint Flags:

```
44 case _: Logging.Info => true
45 case _ => false
46 })
47 appLogging.eventStream.publish(filter)
48 appAuto.eventStream.publish(filter)
49 appLifecycle.eventStream.publish(filter)
50
```

scala/akka/event/LoggingReceiveSpec.scala, line 48 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: filter
Enclosing Method: LoggingReceiveSpec()
File: scala/akka/event/LoggingReceiveSpec.scala:48
Taint Flags:

```
45 case _ => false
46 })
47 appLogging.eventStream.publish(filter)
48 appAuto.eventStream.publish(filter)
49 appLifecycle.eventStream.publish(filter)
50
51 def ignoreMute(t: TestKit): Unit = {
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.event

scala/akka/event/LoggingReceiveSpec.scala, line 49 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: filter
Enclosing Method: LoggingReceiveSpec()
File: scala/akka/event/LoggingReceiveSpec.scala:49
Taint Flags:

```
46  })
47  appLogging.eventStream.publish(filter)
48  appAuto.eventStream.publish(filter)
49  appLifecycle.eventStream.publish(filter)
50
51  def ignoreMute(t: TestKit): Unit = {
52  t.ignoreMsg {
```

scala/akka/event/LoggingReceiveSpec.scala, line 33 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: LoggingReceiveSpec()
File: scala/akka/event/LoggingReceiveSpec.scala:33
Taint Flags:

```
30  val config = ConfigFactory.parseString("""
31  akka.loglevel=DEBUG # test verifies debug
32  """).withFallback(AkkaSpec.testConf)
33  val appLogging =
34  ActorSystem("logging", ConfigFactory.parseMap(Map("akka.actor.debug.receive" -> true).asJava).withFallback(config))
35  val appAuto = ActorSystem(
36  "autoreceive",
```

scala/akka/event/LoggingReceiveSpec.scala, line 35 (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.event

scala/akka/event/LoggingReceiveSpec.scala, line 35 (Code Correctness: Constructor Invokes Overridable Function) — Low

Sink Details

Sink: FunctionCall: config
Enclosing Method: LoggingReceiveSpec()
File: scala/akka/event/LoggingReceiveSpec.scala:35
Taint Flags:

```
32 """).withFallback(AkkaSpec.testConf)
33 val appLogging =
34 ActorSystem("logging", ConfigFactory.parseMap(Map("akka.actor.debug.receive" -> true).asJava).withFallback(config))
35 val appAuto = ActorSystem(
36 "autoreceive",
37 ConfigFactory.parseMap(Map("akka.actor.debug.autoreceive" -> true).asJava).withFallback(config))
38 val appLifecycle = ActorSystem(
```

scala/akka/event/LoggingReceiveSpec.scala, line 38 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: LoggingReceiveSpec()
File: scala/akka/event/LoggingReceiveSpec.scala:38
Taint Flags:

```
35 val appAuto = ActorSystem(
36 "autoreceive",
37 ConfigFactory.parseMap(Map("akka.actor.debug.autoreceive" -> true).asJava).withFallback(config))
38 val appLifecycle = ActorSystem(
39 "lifecycle",
40 ConfigFactory.parseMap(Map("akka.actor.debug.lifecycle" -> true).asJava).withFallback(config))
41
```

scala/akka/event/LoggingReceiveSpec.scala, line 48 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: appAuto
Enclosing Method: LoggingReceiveSpec()

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.event

scala/akka/event/LoggingReceiveSpec.scala, line 48 (Code Correctness: Constructor Invokes Overridable Function) — Low

File: scala/akka/event/LoggingReceiveSpec.scala:48
Taint Flags:

```
45 case _ => false
46 })
47 appLogging.eventStream.publish(filter)
48 appAuto.eventStream.publish(filter)
49 appLifecycle.eventStream.publish(filter)
50
51 def ignoreMute(t: TestKit): Unit = {
```

scala/akka/event/LoggingReceiveSpec.scala, line 49 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: appLifecycle
Enclosing Method: LoggingReceiveSpec()
File: scala/akka/event/LoggingReceiveSpec.scala:49
Taint Flags:

```
46 })
47 appLogging.eventStream.publish(filter)
48 appAuto.eventStream.publish(filter)
49 appLifecycle.eventStream.publish(filter)
50
51 def ignoreMute(t: TestKit): Unit = {
52 t.ignoreMsg {
```

Package: akka.event.jul

scala/akka/event/jul/JavaLoggerSpec.scala, line 42 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: logger
Enclosing Method: JavaLoggerSpec()
File: scala/akka/event/jul/JavaLoggerSpec.scala:42
Taint Flags:

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.event.jul

scala/akka/event/jul/JavaLoggerSpec.scala, line 42 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
39 class JavaLoggerSpec extends AkkaSpec(JavaLoggerSpec.config) {
40
41  val logger = logging.Logger.getLogger(classOf[JavaLoggerSpec.LogProducer].getName)
42  logger.setUseParentHandlers(false) // turn off output of test LogRecords
43  logger.addHandler(new logging.Handler {
44  def publish(record: logging.LogRecord): Unit = {
45  testActor ! record
```

scala/akka/event/jul/JavaLoggerSpec.scala, line 43 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: logger
Enclosing Method: JavaLoggerSpec()
File: scala/akka/event/jul/JavaLoggerSpec.scala:43
Taint Flags:

```
40
41  val logger = logging.Logger.getLogger(classOf[JavaLoggerSpec.LogProducer].getName)
42  logger.setUseParentHandlers(false) // turn off output of test LogRecords
43  logger.addHandler(new logging.Handler {
44  def publish(record: logging.LogRecord): Unit = {
45  testActor ! record
46  }
```

scala/akka/event/jul/JavaLoggerSpec.scala, line 39 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: JavaLoggerSpec()
File: scala/akka/event/jul/JavaLoggerSpec.scala:39
Taint Flags:

```
36 }
37
38 @deprecated("Use SLF4J instead.", "2.6.0")
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.event.jul

scala/akka/event/jul/JavaLoggerSpec.scala, line 39 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
39 class JavaLoggerSpec extends AkkaSpec(JavaLoggerSpec.config) {
40
41  val logger = logging.Logger.getLogger(classOf[JavaLoggerSpec.LogProducer].getName)
42  logger.setUseParentHandlers(false) // turn off output of test LogRecords
```

Package: akka.io.dns

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 51 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: defaultTimeout
Enclosing Method: AsyncDnsResolverIntegrationSpec()
File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:51
Taint Flags:

```
48  import AsyncDnsResolverIntegrationSpec._
49
50  override implicit val patience: PatienceConfig =
51  PatienceConfig(defaultTimeout.duration + 1.second, Span(100, Millis))
52
53  override val hostPort = dockerDnsServerPort
54
```

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 46 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: conf
Enclosing Method: AsyncDnsResolverIntegrationSpec()
File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:46
Taint Flags:

```
43 }
44
45 class AsyncDnsResolverIntegrationSpec
46  extends DockerBindDnsService(AsyncDnsResolverIntegrationSpec.conf)
47  with WithLogCapturing {
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.io.dns

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 46 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
48 import AsyncDnsResolverIntegrationSpec._
49
```

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 53 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: dockerDnsServerPort
Enclosing Method: AsyncDnsResolverIntegrationSpec()
File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:53
Taint Flags:

```
50 override implicit val patience: PatienceConfig =
51 PatienceConfig(defaultTimeout.duration + 1.second, Span(100, Millis))
52
53 override val hostPort = dockerDnsServerPort
54
55 "Resolver" must {
56 if (!dockerAvailable()) {
```

Package: akka.routing

scala/akka/routing/ConfiguredLocalRoutingSpec.scala, line 106 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: ConfiguredLocalRoutingSpec()
File: scala/akka/routing/ConfiguredLocalRoutingSpec.scala:106
Taint Flags:

```
103 }
104
105 class ConfiguredLocalRoutingSpec
106  extends AkkaSpec(ConfiguredLocalRoutingSpec.config)
107  with DefaultTimeout
108  with ImplicitSender {
109  import ConfiguredLocalRoutingSpec._
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.routing

scala/akka/routing/ConfiguredLocalRoutingSpec.scala, line 106 (Code Correctness: Constructor Invokes Overridable Function) — Low

scala/akka/routing/RoutingSpec.scala, line 51 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: RoutingSpec()
File: scala/akka/routing/RoutingSpec.scala:51
Taint Flags:

```
48
49 }
50
51 class RoutingSpec extends AkkaSpec(RoutingSpec.config) with DefaultTimeout with ImplicitSender {
52  implicit val ec: ExecutionContextExecutor = system.dispatcher
53  import RoutingSpec._
54
```

scala/akka/routing/ConsistentHashingRouterSpec.scala, line 57 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: ConsistentHashingRouterSpec()
File: scala/akka/routing/ConsistentHashingRouterSpec.scala:57
Taint Flags:

```
54 }
55
56 class ConsistentHashingRouterSpec
57  extends AkkaSpec(ConsistentHashingRouterSpec.config)
58  with DefaultTimeout
59  with ImplicitSender {
60  import ConsistentHashingRouterSpec._
```

Code Correctness: Constructor Invokes Overridable Function | Low

Package: akka.routing

scala/akka/routing/ResizerSpec.scala, line 40 (Code Correctness: Constructor Invokes Overridable Function) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: ResizerSpec()
File: scala/akka/routing/ResizerSpec.scala:40
Taint Flags:

```
37
38 }
39
40 class ResizerSpec extends AkkaSpec(ResizerSpec.config) with DefaultTimeout with ImplicitSender {
41
42  import akka.routing.ResizerSpec._
43
```

Package: akka.serialization

scala/akka/serialization/SerializationSetupSpec.scala, line 71 (Code Correctness: Constructor Invokes Overridable Function) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: bootstrapSettings
Enclosing Method: SerializationSetupSpec()
File: scala/akka/serialization/SerializationSetupSpec.scala:71
Taint Flags:

```
68  }
69  """)),
70  None)
71  val actorSystemSettings = ActorSystemSetup(bootstrapSettings, serializationSettings)
72
73  val noJavaSerializationSystem = ActorSystem(
74  "SerializationSettingsSpec" + "NoJavaSerialization",
```

scala/akka/serialization/SerializationSetupSpec.scala, line 158 (Code Correctness: Constructor Invokes Overridable Function) | Low

Issue Details

Kingdom: Code Quality

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.serialization

scala/akka/serialization/SerializationSetupSpec.scala, line 158 (Code Correctness: Constructor Invokes Overridable Function) — Low

Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: addedJavaSerializationSettings
Enclosing Method: SerializationSetupSpec()
File: scala/akka/serialization/SerializationSetupSpec.scala:158
Taint Flags:

```
155 val addedJavaSerializationViaSettingsSystem =
156 ActorSystem(
157 "addedJavaSerializationSystem",
158 ActorSystemSetup(addedJavaSerializationProgramaticallyButDisabledSettings, addedJavaSerializationSettings))
159
160 "Disabling java serialization" should {
161
```

scala/akka/serialization/SerializationSetupSpec.scala, line 158 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: addedJavaSerializationProgramaticallyButDisabledSettings
Enclosing Method: SerializationSetupSpec()
File: scala/akka/serialization/SerializationSetupSpec.scala:158
Taint Flags:

```
155 val addedJavaSerializationViaSettingsSystem =
156 ActorSystem(
157 "addedJavaSerializationSystem",
158 ActorSystemSetup(addedJavaSerializationProgramaticallyButDisabledSettings, addedJavaSerializationSettings))
159
160 "Disabling java serialization" should {
161
```

scala/akka/serialization/PrimitivesSerializationSpec.scala, line 20 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.serialization

scala/akka/serialization/PrimitivesSerializationSpec.scala, line 20 (Code Correctness: Constructor Invokes Overridable Function) — Low

Sink: FunctionCall: serializationTestOverrides
Enclosing Method: PrimitivesSerializationSpec()
File: scala/akka/serialization/PrimitivesSerializationSpec.scala:20
Taint Flags:

```
17 object PrimitivesSerializationSpec {
18  val serializationTestOverrides = ""
19
20  val testConfig = ConfigFactory.parseString(serializationTestOverrides).withFallback(AkkaSpec.testConf)
21 }
22
23 class PrimitivesSerializationSpec extends AkkaSpec(PrimitivesSerializationSpec.testConfig) {
```

scala/akka/serialization/PrimitivesSerializationSpec.scala, line 23 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: testConfig
Enclosing Method: PrimitivesSerializationSpec()
File: scala/akka/serialization/PrimitivesSerializationSpec.scala:23
Taint Flags:

```
20  val testConfig = ConfigFactory.parseString(serializationTestOverrides).withFallback(AkkaSpec.testConf)
21 }
22
23 class PrimitivesSerializationSpec extends AkkaSpec(PrimitivesSerializationSpec.testConfig) {
24
25  val buffer = {
26  val b = ByteBuffer.allocate(4096)
```

scala/akka/serialization/SerializationSetupSpec.scala, line 71 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: serializationSettings
Enclosing Method: SerializationSetupSpec()
File: scala/akka/serialization/SerializationSetupSpec.scala:71
Taint Flags:

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.serialization

scala/akka/serialization/SerializationSetupSpec.scala, line 71 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
68  }
69  """)),
70  None)
71  val actorSystemSettings = ActorSystemSetup(bootstrapSettings, serializationSettings)
72
73  val noJavaSerializationSystem = ActorSystem(
74  "SerializationSettingsSpec" + "NoJavaSerialization",
```

scala/akka/serialization/SerializationSetupSpec.scala, line 89 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: actorSystemSettings
Enclosing Method: SerializationSetupSpec()
File: scala/akka/serialization/SerializationSetupSpec.scala:89
Taint Flags:

```
86 }
87
88 class SerializationSetupSpec
89  extends AkkaSpec(ActorSystem("SerializationSettingsSpec", SerializationSetupSpec.actorSystemSettings)) {
90
91  import SerializationSetupSpec._
92
```

scala/akka/serialization/SerializationSetupSpec.scala, line 84 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: noJavaSerializationSystem
Enclosing Method: SerializationSetupSpec()
File: scala/akka/serialization/SerializationSetupSpec.scala:84
Taint Flags:

```
81  }
82  }
83  """.stripMargin))
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.serialization

scala/akka/serialization/SerializationSetupSpec.scala, line 84 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
84  val noJavaSerializer = new DisabledJavaSerializer(noJavaSerializationSystem.asInstanceOf[ExtendedActorSystem])
85
86 }
87
```

scala/akka/serialization/AsyncSerializeSpec.scala, line 95 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: AsyncSerializeSpec()
File: scala/akka/serialization/AsyncSerializeSpec.scala:95
Taint Flags:

```
92
93 }
94
95 class AsyncSerializeSpec extends AkkaSpec(AsyncSerializeSpec.config) {
96  import AsyncSerializeSpec._
97
98  val ser = SerializationExtension(system)
```

Package: akka.testkit

scala/akka/testkit/CallingThreadDispatcherModelSpec.scala, line 48 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: config
Enclosing Method: CallingThreadDispatcherModelSpec()
File: scala/akka/testkit/CallingThreadDispatcherModelSpec.scala:48
Taint Flags:

```
45
46 }
47
48 class CallingThreadDispatcherModelSpec extends ActorModelSpec(CallingThreadDispatcherModelSpec.config) {
49  import ActorModelSpec._
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.testkit

scala/akka/testkit/CallingThreadDispatcherModelSpec.scala, line 48 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
50
51  val dispatcherCount = new AtomicInteger()
```

Package: akka.util

scala/akka/util/ZipfianGenerator.scala, line 37 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: akka$util$ZipfianGenerator$$zeta
Enclosing Method: ZipfianGenerator()
File: scala/akka/util/ZipfianGenerator.scala:37
Taint Flags:

```
34  private val n = max - min + 1
35  private val alpha = 1.0 / (1.0 - theta)
36  private val zeta2 = ZipfianGenerator.zeta(2, theta)
37  private val zetaN = ZipfianGenerator.zeta(n, theta)
38  private val eta = (1 - Math.pow(2.0 / n, 1 - theta)) / (1 - zeta2 / zetaN)
39  private val random = new scala.util.Random(seed)
40
```

scala/akka/util/ByteStringSpec.scala, line 114 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: arbSlice
Enclosing Method: ByteStringSpec()
File: scala/akka/util/ByteStringSpec.scala:114
Taint Flags:

```
111  Gen.containerOfN[Array, Short](n, arbitrary[Short])
112  }
113  }
114  implicit val arbitraryShortArraySlice: Arbitrary[ArraySlice[Short]] = arbSlice(arbitraryShortArray)
115  val arbitraryIntArray: Arbitrary[Array[Int]] = Arbitrary {
116  Gen.sized { n =>
117  Gen.containerOfN[Array, Int](n, arbitrary[Int])
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.util

scala/akka/util/ByteStringSpec.scala, line 114 (Code Correctness: Constructor Invokes Overridable Function) — Low

scala/akka/util/ZipfianGenerator.scala, line 36 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: akka$util$ZipfianGenerator$$zeta
Enclosing Method: ZipfianGenerator()
File: scala/akka/util/ZipfianGenerator.scala:36
Taint Flags:

```
33 final class ZipfianGenerator(min: Int, max: Int, theta: Double, seed: Int) {
34  private val n = max - min + 1
35  private val alpha = 1.0 / (1.0 - theta)
36  private val zeta2 = ZipfianGenerator.zeta(2, theta)
37  private val zetaN = ZipfianGenerator.zeta(n, theta)
38  private val eta = (1 - Math.pow(2.0 / n, 1 - theta)) / (1 - zeta2 / zetaN)
39  private val random = new scala.util.Random(seed)
```

scala/akka/util/ZipfianGenerator.scala, line 38 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: zetaN
Enclosing Method: ZipfianGenerator()
File: scala/akka/util/ZipfianGenerator.scala:38
Taint Flags:

```
35  private val alpha = 1.0 / (1.0 - theta)
36  private val zeta2 = ZipfianGenerator.zeta(2, theta)
37  private val zetaN = ZipfianGenerator.zeta(n, theta)
38  private val eta = (1 - Math.pow(2.0 / n, 1 - theta)) / (1 - zeta2 / zetaN)
39  private val random = new scala.util.Random(seed)
40
41  def next(): Int = {
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.util

scala/akka/util/ByteStringSpec.scala, line 108 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: arbitraryByteArray
Enclosing Method: ByteStringSpec()
File: scala/akka/util/ByteStringSpec.scala:108
Taint Flags:

```
105 Gen.containerOfN[Array, Byte](n, arbitrary[Byte])
106 }
107 }
108 implicit val arbitraryByteArraySlice: Arbitrary[ArraySlice[Byte]] = arbSlice(arbitraryByteArray)
109 val arbitraryShortArray: Arbitrary[Array[Short]] = Arbitrary {
110 Gen.sized { n =>
111 Gen.containerOfN[Array, Short](n, arbitrary[Short])
```

scala/akka/util/ByteStringSpec.scala, line 114 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: arbitraryShortArray
Enclosing Method: ByteStringSpec()
File: scala/akka/util/ByteStringSpec.scala:114
Taint Flags:

```
111 Gen.containerOfN[Array, Short](n, arbitrary[Short])
112 }
113 }
114 implicit val arbitraryShortArraySlice: Arbitrary[ArraySlice[Short]] = arbSlice(arbitraryShortArray)
115 val arbitraryIntArray: Arbitrary[Array[Int]] = Arbitrary {
116 Gen.sized { n =>
117 Gen.containerOfN[Array, Int](n, arbitrary[Int])
```

scala/akka/util/ByteStringSpec.scala, line 108 (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.util

scala/akka/util/ByteStringSpec.scala, line 108 (Code Correctness: Constructor Invokes Overridable Function) — Low

Sink Details

Sink: FunctionCall: arbSlice
Enclosing Method: ByteStringSpec()
File: scala/akka/util/ByteStringSpec.scala:108
Taint Flags:

```
105 Gen.containerOfN[Array, Byte](n, arbitrary[Byte])
106 }
107 }
108 implicit val arbitraryByteArraySlice: Arbitrary[ArraySlice[Byte]] = arbSlice(arbitraryByteArray)
109 val arbitraryShortArray: Arbitrary[Array[Short]] = Arbitrary {
110 Gen.sized { n =>
111 Gen.containerOfN[Array, Short](n, arbitrary[Short])
```

scala/akka/util/ByteStringSpec.scala, line 120 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: arbitraryIntArray
Enclosing Method: ByteStringSpec()
File: scala/akka/util/ByteStringSpec.scala:120
Taint Flags:

```
117 Gen.containerOfN[Array, Int](n, arbitrary[Int])
118 }
119 }
120 implicit val arbitraryIntArraySlice: Arbitrary[ArraySlice[Int]] = arbSlice(arbitraryIntArray)
121 val arbitraryLongArray: Arbitrary[Array[Long]] = Arbitrary {
122 Gen.sized { n =>
123 Gen.containerOfN[Array, Long](n, arbitrary[Long])
```

scala/akka/util/ZipfianGenerator.scala, line 37 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: n
Enclosing Method: ZipfianGenerator()

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.util

scala/akka/util/ZipfianGenerator.scala, line 37 (Code Correctness: Constructor Invokes Overridable Function) — Low

File: scala/akka/util/ZipfianGenerator.scala:37
Taint Flags:

```
34  private val n = max - min + 1
35  private val alpha = 1.0 / (1.0 - theta)
36  private val zeta2 = ZipfianGenerator.zeta(2, theta)
37  private val zetaN = ZipfianGenerator.zeta(n, theta)
38  private val eta = (1 - Math.pow(2.0 / n, 1 - theta)) / (1 - zeta2 / zetaN)
39  private val random = new scala.util.Random(seed)
40
```

scala/akka/util/ZipfianGenerator.scala, line 38 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: n
Enclosing Method: ZipfianGenerator()
File: scala/akka/util/ZipfianGenerator.scala:38
Taint Flags:

```
35  private val alpha = 1.0 / (1.0 - theta)
36  private val zeta2 = ZipfianGenerator.zeta(2, theta)
37  private val zetaN = ZipfianGenerator.zeta(n, theta)
38  private val eta = (1 - Math.pow(2.0 / n, 1 - theta)) / (1 - zeta2 / zetaN)
39  private val random = new scala.util.Random(seed)
40
41  def next(): Int = {
```

scala/akka/util/ZipfianGenerator.scala, line 38 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: zeta2
Enclosing Method: ZipfianGenerator()
File: scala/akka/util/ZipfianGenerator.scala:38
Taint Flags:

```
35  private val alpha = 1.0 / (1.0 - theta)
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.util

scala/akka/util/ZipfianGenerator.scala, line 38 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
36 private val zeta2 = ZipfianGenerator.zeta(2, theta)
37 private val zetaN = ZipfianGenerator.zeta(n, theta)
38 private val eta = (1 - Math.pow(2.0 / n, 1 - theta)) / (1 - zeta2 / zetaN)
39 private val random = new scala.util.Random(seed)
40
41 def next(): Int = {
```

scala/akka/util/ByteStringSpec.scala, line 132 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: arbSlice
Enclosing Method: ByteStringSpec()
File: scala/akka/util/ByteStringSpec.scala:132
Taint Flags:

```
129 Gen.containerOfN[Array, Float](n, arbitrary[Float])
130 }
131 }
132 implicit val arbitraryFloatArraySlice: Arbitrary[ArraySlice[Float]] = arbSlice(arbitraryFloatArray)
133 val arbitraryDoubleArray: Arbitrary[Array[Double]] = Arbitrary {
134 Gen.sized { n =>
135 Gen.containerOfN[Array, Double](n, arbitrary[Double])
```

scala/akka/util/ByteStringSpec.scala, line 120 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: arbSlice
Enclosing Method: ByteStringSpec()
File: scala/akka/util/ByteStringSpec.scala:120
Taint Flags:

```
117 Gen.containerOfN[Array, Int](n, arbitrary[Int])
118 }
119 }
120 implicit val arbitraryIntArraySlice: Arbitrary[ArraySlice[Int]] = arbSlice(arbitraryIntArray)
```

Code Correctness: Constructor Invokes Overridable Function | Low

Package: akka.util

scala/akka/util/ByteStringSpec.scala, line 120 (Code Correctness: Constructor Invokes Overridable Function) | Low

```
121 val arbitraryLongArray: Arbitrary[Array[Long]] = Arbitrary {
122 Gen.sized { n =>
123 Gen.containerOfN[Array, Long](n, arbitrary[Long])
```

scala/akka/util/ByteStringSpec.scala, line 126 (Code Correctness: Constructor Invokes Overridable Function) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: arbitraryLongArray
Enclosing Method: ByteStringSpec()
File: scala/akka/util/ByteStringSpec.scala:126
Taint Flags:

```
123 Gen.containerOfN[Array, Long](n, arbitrary[Long])
124 }
125 }
126 implicit val arbitraryLongArraySlice: Arbitrary[ArraySlice[Long]] = arbSlice(arbitraryLongArray)
127 val arbitraryFloatArray: Arbitrary[Array[Float]] = Arbitrary {
128 Gen.sized { n =>
129 Gen.containerOfN[Array, Float](n, arbitrary[Float])
```

scala/akka/util/ByteStringSpec.scala, line 132 (Code Correctness: Constructor Invokes Overridable Function) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: arbitraryFloatArray
Enclosing Method: ByteStringSpec()
File: scala/akka/util/ByteStringSpec.scala:132
Taint Flags:

```
129 Gen.containerOfN[Array, Float](n, arbitrary[Float])
130 }
131 }
132 implicit val arbitraryFloatArraySlice: Arbitrary[ArraySlice[Float]] = arbSlice(arbitraryFloatArray)
133 val arbitraryDoubleArray: Arbitrary[Array[Double]] = Arbitrary {
134 Gen.sized { n =>
135 Gen.containerOfN[Array, Double](n, arbitrary[Double])
```

Code Correctness: Constructor Invokes Overridable Function | Low

Package: akka.util

scala/akka/util/ByteStringSpec.scala, line 138 (Code Correctness: Constructor Invokes Overridable Function) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: arbSlice
Enclosing Method: ByteStringSpec()
File: scala/akka/util/ByteStringSpec.scala:138
Taint Flags:

```
135 Gen.containerOfN[Array, Double](n, arbitrary[Double])
136 }
137 }
138 implicit val arbitraryDoubleArraySlice: Arbitrary[ArraySlice[Double]] = arbSlice(arbitraryDoubleArray)
139
140 type ArrayNumBytes[A] = (Array[A], Int)
141
```

scala/akka/util/ByteStringSpec.scala, line 126 (Code Correctness: Constructor Invokes Overridable Function) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: arbSlice
Enclosing Method: ByteStringSpec()
File: scala/akka/util/ByteStringSpec.scala:126
Taint Flags:

```
123 Gen.containerOfN[Array, Long](n, arbitrary[Long])
124 }
125 }
126 implicit val arbitraryLongArraySlice: Arbitrary[ArraySlice[Long]] = arbSlice(arbitraryLongArray)
127 val arbitraryFloatArray: Arbitrary[Array[Float]] = Arbitrary {
128 Gen.sized { n =>
129 Gen.containerOfN[Array, Float](n, arbitrary[Float])
```

scala/akka/util/ByteStringSpec.scala, line 138 (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.util

scala/akka/util/ByteStringSpec.scala, line 138 (Code Correctness: Constructor Invokes Overridable Function) — Low

Sink Details

Sink: FunctionCall: arbitraryDoubleArray
Enclosing Method: ByteStringSpec()
File: scala/akka/util/ByteStringSpec.scala:138
Taint Flags:

```
135 Gen.containerOfN[Array, Double](n, arbitrary[Double])
136 }
137 }
138 implicit val arbitraryDoubleArraySlice: Arbitrary[ArraySlice[Double]] = arbSlice(arbitraryDoubleArray)
139
140 type ArrayNumBytes[A] = (Array[A], Int)
141
```

Code Correctness: Erroneous String Compare (8 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");
  }
```

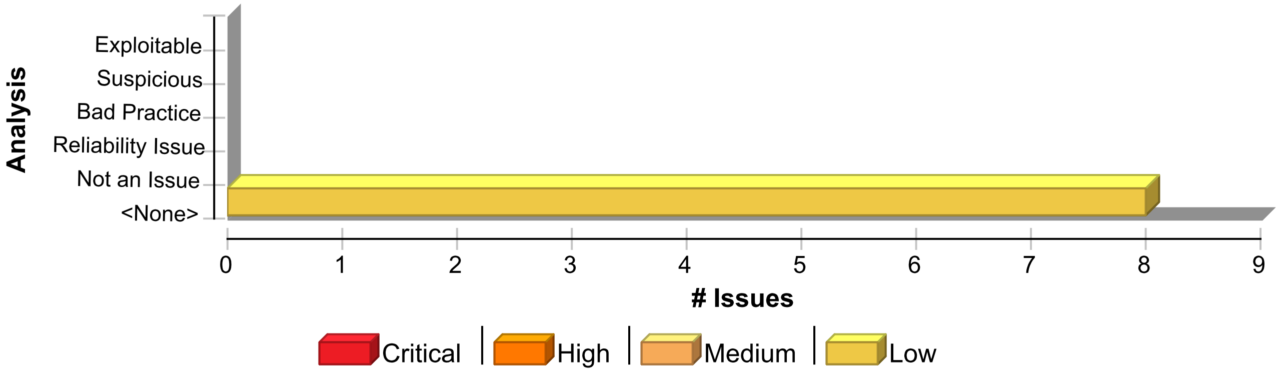
The `==` and `!=` operators will only behave as expected when they are used to compare strings contained in objects that are equal. The most common way for this to occur is for the strings to be interned, whereby the strings are added to a pool of objects maintained by the `String` class. Once a string 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	8	0	0	8
Total	**8**	**0**	**0**	**8**

Code Correctness: Erroneous String Compare — Low

Package: akka.serialization

scala/akka/serialization/AsyncSerializeSpec.scala, line 77 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: fromBinaryAsyncCS()
File: scala/akka/serialization/AsyncSerializeSpec.scala:77
Taint Flags:

```
74 }
75
76 override def fromBinaryAsyncCS(bytes: Array[Byte], manifest: String): CompletionStage[AnyRef] = {
77 manifest match {
78 case "1" => CompletableFuture.completedFuture(Message3(new String(bytes)))
79 case "2" => CompletableFuture.completedFuture(Message4(new String(bytes)))
80 case _ => throw new IllegalArgumentException(s"Unknown manifest $manifest")
```

scala/akka/serialization/AsyncSerializeSpec.scala, line 50 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: fromBinaryAsync()
File: scala/akka/serialization/AsyncSerializeSpec.scala:50
Taint Flags:

```
47 }
48
49 override def fromBinaryAsync(bytes: Array[Byte], manifest: String): Future[AnyRef] = {
50 manifest match {
51 case "1" => Future.successful(Message1(new String(bytes)))
52 case "2" => Future.successful(Message2(new String(bytes)))
53 case _ => throw new IllegalArgumentException(s"Unknown manifest $manifest")
```

scala/akka/serialization/AsyncSerializeSpec.scala, line 50 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Code Correctness: Erroneous String Compare

Low

Package: akka.serialization

scala/akka/serialization/AsyncSerializeSpec.scala, line 50 (Code Correctness: Erroneous String Compare)

Low

Sink Details

Sink: Operation
Enclosing Method: fromBinaryAsync()
File: scala/akka/serialization/AsyncSerializeSpec.scala:50
Taint Flags:

```
47 }
48
49 override def fromBinaryAsync(bytes: Array[Byte], manifest: String): Future[AnyRef] = {
50 manifest match {
51 case "1" => Future.successful(Message1(new String(bytes)))
52 case "2" => Future.successful(Message2(new String(bytes)))
53 case _ => throw new IllegalArgumentException(s"Unknown manifest $manifest")
```

scala/akka/serialization/AsyncSerializeSpec.scala, line 77 (Code Correctness: Erroneous String Compare)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: fromBinaryAsyncCS()
File: scala/akka/serialization/AsyncSerializeSpec.scala:77
Taint Flags:

```
74 }
75
76 override def fromBinaryAsyncCS(bytes: Array[Byte], manifest: String): CompletionStage[AnyRef] = {
77 manifest match {
78 case "1" => CompletableFuture.completedFuture(Message3(new String(bytes)))
79 case "2" => CompletableFuture.completedFuture(Message4(new String(bytes)))
80 case _ => throw new IllegalArgumentException(s"Unknown manifest $manifest")
```

Package: akka.util

scala/akka/util/LineNumberSpecCodeForScala.scala, line 20 (Code Correctness: Erroneous String Compare)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation

Code Correctness: Erroneous String Compare | Low

Package: akka.util

scala/akka/util/LineNumberSpecCodeForScala.scala, line 20 (Code Correctness: Erroneous String Compare) | Low

Enclosing Method: applyOrElse()
File: scala/akka/util/LineNumberSpecCodeForScala.scala:20
Taint Flags:

```
17  Integer.parseInt(s)
18  }
19
20  val partial: PartialFunction[String, Unit] = {
21  case "a" =>
22  case "b" =>
23  }
```

scala/akka/util/LineNumberSpecCodeForScala.scala, line 20 (Code Correctness: Erroneous String Compare) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: applyOrElse()
File: scala/akka/util/LineNumberSpecCodeForScala.scala:20
Taint Flags:

```
17  Integer.parseInt(s)
18  }
19
20  val partial: PartialFunction[String, Unit] = {
21  case "a" =>
22  case "b" =>
23  }
```

Package: scala.akka.actor

scala/akka/actor/CoordinatedShutdownSpec.scala, line 187 (Code Correctness: Erroneous String Compare) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: apply()
File: scala/akka/actor/CoordinatedShutdownSpec.scala:187

Code Correctness: Erroneous String Compare — Low

Package: scala.akka.actor

scala/akka/actor/CoordinatedShutdownSpec.scala, line 187 (Code Correctness: Erroneous String Compare) — Low

Taint Flags:

```
184 }
185 whenReady(co.run(UnknownReason).flatMap(_ => messagesFut), timeout(250.milliseconds)) { messages =>
186 messages.distinct.size shouldEqual 2
187 messages.foreach {
188 case "copy1" | "copy3" => // OK
189 case other => fail(s"Unexpected probe message ${other}!")
190 }
```

scala/akka/actor/CoordinatedShutdownSpec.scala, line 187 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: apply()
File: scala/akka/actor/CoordinatedShutdownSpec.scala:187
Taint Flags:

```
184 }
185 whenReady(co.run(UnknownReason).flatMap(_ => messagesFut), timeout(250.milliseconds)) { messages =>
186 messages.distinct.size shouldEqual 2
187 messages.foreach {
188 case "copy1" | "copy3" => // OK
189 case other => fail(s"Unexpected probe message ${other}!")
190 }
```

Code Correctness: Non-Static Inner Class Implements Serializable (107 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

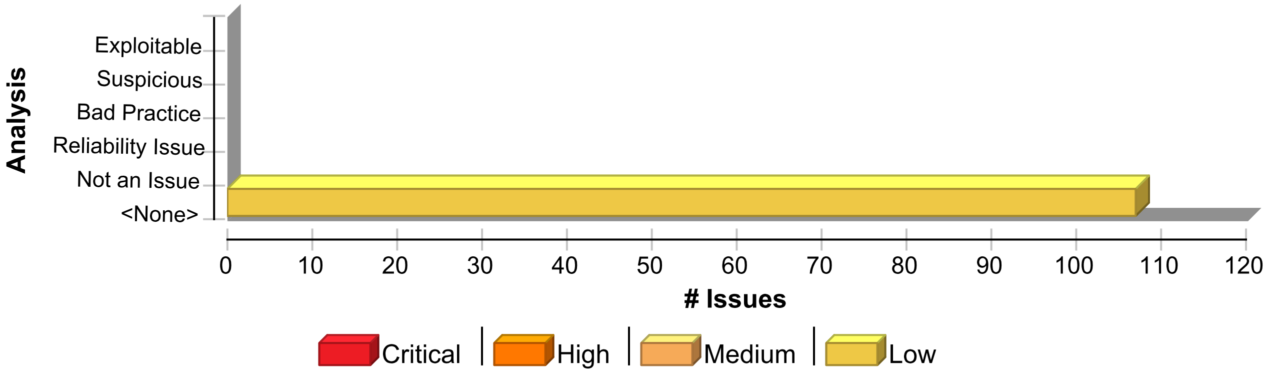
When using inner classes, they should not be serialized, or they should be changed to static-nested classes, since these do not have the drawbacks that non-static inner classes have when serialized. When a nested class is static it inherently has no association with instance variables (including those 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	107	0	0	107
Total	**107**	**0**	**0**	**107**

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

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 346 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: SupervisorHierarchySpec$Fail
File: scala/akka/actor/SupervisorHierarchySpec.scala:346
Taint Flags:

```
343
344 sealed trait Action
345 final case class Ping(ref: ActorRef) extends Action
346 final case class Fail(ref: ActorRef, directive: Directive) extends Action
347
348 sealed trait State
349 case object Idle extends State
```

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 60 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorWithBoundedStashSpec$Bounded10

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

Package: akka.actor

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 60 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

File: scala/akka/actor/ActorWithBoundedStashSpec.scala:60
Taint Flags:

```
57 }
58
59 // bounded deque-based mailbox with capacity 10
60 class Bounded10(@unused settings: Settings, @unused config: Config) extends BoundedDequeBasedMailbox(10, 500 millis)
61
62 class Bounded100(@unused settings: Settings, @unused config: Config) extends BoundedDequeBasedMailbox(100, 500 millis)
63
```

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

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: SupervisorHierarchySpec$Failure
File: scala/akka/actor/SupervisorHierarchySpec.scala:69
Taint Flags:

```
66 case object PongOfDeath
67 final case class Event(msg: Any, identity: Long) { val time: Long = System.nanoTime }
68 final case class ErrorLog(msg: String, log: Vector[Event])
69 final case class Failure(
70 directive: Directive,
71 stop: Boolean,
72 depth: Int,
```

scala/akka/actor/FunctionRefSpec.scala, line 16 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: FunctionRefSpec$Forwarded
File: scala/akka/actor/FunctionRefSpec.scala:16
Taint Flags:

```
13
14 case class GetForwarder(replyTo: ActorRef)
15 case class DropForwarder(ref: FunctionRef)
```

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

Package: akka.actor

scala/akka/actor/FunctionRefSpec.scala, line 16 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
16  case class Forwarded(msg: Any, sender: ActorRef)
17
18  class Super extends Actor {
19  def receive = {
```

scala/akka/actor/ActorRefSpec.scala, line 22 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorRefSpec$ReplyTo
File: scala/akka/actor/ActorRefSpec.scala:22
Taint Flags:

```
19
20 object ActorRefSpec {
21
22  final case class ReplyTo(sender: ActorRef)
23
24  class ReplyActor extends Actor {
25  var replyTo: ActorRef = null
```

scala/akka/actor/TimerSpec.scala, line 17 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TimerSpec$Tick
File: scala/akka/actor/TimerSpec.scala:17
Taint Flags:

```
14
15 object TimerSpec {
16  sealed trait Command
17  case class Tick(n: Int) extends Command
18  case object Bump extends Command
19  case class SlowThenBump(latch: TestLatch) extends Command with NoSerializationVerificationNeeded
20  case object End extends Command
```

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

Package: akka.actor

scala/akka/actor/TimerSpec.scala, line 31 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TimerSpec$Exc
File: scala/akka/actor/TimerSpec.scala:31
Taint Flags:

```
28 case class GotPostStop(timerActive: Boolean) extends Event
29 case class GotPreRestart(timerActive: Boolean) extends Event
30
31 class Exc extends RuntimeException("simulated exc") with NoStackTrace
32
33 def target(monitor: ActorRef, interval: FiniteDuration, repeat: Boolean, initial: () => Int): Props =
34 Props(new Target(monitor, interval, repeat, initial))
```

scala/akka/actor/DeathWatchSpec.scala, line 67 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: DeathWatchSpec$WrappedTerminated
File: scala/akka/actor/DeathWatchSpec.scala:67
Taint Flags:

```
64 * Forwarding `Terminated` to non-watching testActor is not possible,
65 * and therefore the `Terminated` message is wrapped.
66 */
67 final case class WrappedTerminated(t: Terminated)
68
69 final case class W(ref: ActorRef)
70 final case class U(ref: ActorRef)
```

scala/akka/actor/DeathWatchSpec.scala, line 79 (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.actor

scala/akka/actor/DeathWatchSpec.scala, line 79 (Code Correctness: Non-Static Inner Class Implements Serializable)

Low

Sink: Class: DeathWatchSpec$WatchWithVerifier$StartStashing
File: scala/akka/actor/DeathWatchSpec.scala:79
Taint Flags:

```
76  case class WatchThis(ref: ActorRef)
77  case object Watching
78  case class CustomWatchMsg(ref: ActorRef)
79  case class StartStashing(numberOfMessagesToStash: Int)
80  case object StashingStarted
81
82  def props(probe: ActorRef) = Props(new WatchWithVerifier(probe))
```

scala/akka/actor/Bench.scala, line 12 (Code Correctness: Non-Static Inner Class Implements Serializable)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Chameneos$Meet
File: scala/akka/actor/Bench.scala:12
Taint Flags:

```
9 object Chameneos {
10
11  sealed trait ChameneosEvent
12  final case class Meet(from: ActorRef, colour: Colour) extends ChameneosEvent
13  final case class Change(colour: Colour) extends ChameneosEvent
14  final case class MeetingCount(count: Int) extends ChameneosEvent
15  case object Exit extends ChameneosEvent
```

scala/akka/actor/FSMActorSpec.scala, line 100 (Code Correctness: Non-Static Inner Class Implements Serializable)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: FSMActorSpec$CodeState
File: scala/akka/actor/FSMActorSpec.scala:100
Taint Flags:

```
97  private def doUnlock(): Unit = unlockedLatch.open()
98  }
```

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

Package: akka.actor

scala/akka/actor/FSMActorSpec.scala, line 100 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
99
100  final case class CodeState(soFar: String, code: String)
101 }
102
103 class FSMActorSpec extends AkkaSpec(Map("akka.actor.debug.fsm" -> true)) with ImplicitSender {
```

scala/akka/actor/TimerSpec.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: TimerSpec$GotPreRestart
File: scala/akka/actor/TimerSpec.scala:29
Taint Flags:

```
26  sealed trait Event
27  case class Tock(n: Int) extends Event
28  case class GotPostStop(timerActive: Boolean) extends Event
29  case class GotPreRestart(timerActive: Boolean) extends Event
30
31  class Exc extends RuntimeException("simulated exc") with NoStackTrace
32
```

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 62 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorWithBoundedStashSpec$Bounded100
File: scala/akka/actor/ActorWithBoundedStashSpec.scala:62
Taint Flags:

```
59  // bounded deque-based mailbox with capacity 10
60  class Bounded10(@unused settings: Settings, @unused config: Config) extends BoundedDequeBasedMailbox(10, 500 millis)
61
62  class Bounded100(@unused settings: Settings, @unused config: Config) extends BoundedDequeBasedMailbox(100, 500 millis)
63
64  val dispatcherId1 = "my-dispatcher-1"
65  val dispatcherId2 = "my-dispatcher-2"
```

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

Package: akka.actor

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 62 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

scala/akka/actor/SupervisorHierarchySpec.scala, line 35 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: SupervisorHierarchySpec$FireWorkerException
File: scala/akka/actor/SupervisorHierarchySpec.scala:35
Taint Flags:

```
32
33 object SupervisorHierarchySpec {
34
35  class FireWorkerException(msg: String) extends Exception(msg)
36
37  /**
38  * For testing Supervisor behavior, normally you don't supply the strategy
```

scala/akka/actor/TypedActorSpec.scala, line 121 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TypedActorSpec$Bar
File: scala/akka/actor/TypedActorSpec.scala:121
Taint Flags:

```
118  throw new IllegalStateException(s"expected $foo $s $i $o")
119  }
120
121  class Bar extends Foo with Serializable {
122
123  import akka.actor.TypedActor.dispatcher
124
```

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

Issue Details

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

Package: akka.actor

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

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: SupervisorHierarchySpec$Dump
File: scala/akka/actor/SupervisorHierarchySpec.scala:81
Taint Flags:

```
78  with NoStackTrace {
79  override def toString = productPrefix + productIterator.mkString("(", ",", ")")
80  }
81  final case class Dump(level: Int)
82
83  val config = ConfigFactory.parseString("""
84  hierarchy {
```

scala/akka/actor/FunctionRefSpec.scala, line 15 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: FunctionRefSpec$DropForwarder
File: scala/akka/actor/FunctionRefSpec.scala:15
Taint Flags:

```
12 object FunctionRefSpec {
13
14  case class GetForwarder(replyTo: ActorRef)
15  case class DropForwarder(ref: FunctionRef)
16  case class Forwarded(msg: Any, sender: ActorRef)
17
18  class Super extends Actor {
```

scala/akka/actor/DeathWatchSpec.scala, line 76 (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.actor

scala/akka/actor/DeathWatchSpec.scala, line 76 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: DeathWatchSpec$WatchWithVerifier$WatchThis
File: scala/akka/actor/DeathWatchSpec.scala:76
Taint Flags:

```
73 final case class Latches(t1: TestLatch, t2: TestLatch) extends NoSerializationVerificationNeeded
74
75 object WatchWithVerifier {
76 case class WatchThis(ref: ActorRef)
77 case object Watching
78 case class CustomWatchMsg(ref: ActorRef)
79 case class StartStashing(numberOfMessagesToStash: Int)
```

scala/akka/actor/Bench.scala, line 14 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Chameneos$MeetingCount
File: scala/akka/actor/Bench.scala:14
Taint Flags:

```
11 sealed trait ChameneosEvent
12 final case class Meet(from: ActorRef, colour: Colour) extends ChameneosEvent
13 final case class Change(colour: Colour) extends ChameneosEvent
14 final case class MeetingCount(count: Int) extends ChameneosEvent
15 case object Exit extends ChameneosEvent
16
17 abstract sealed class Colour
```

scala/akka/actor/UidClashTest.scala, line 16 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: UidClashTest$TerminatedForNonWatchedActor
File: scala/akka/actor/UidClashTest.scala:16
Taint Flags:

```
13
14 object UidClashTest {
```

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

Package: akka.actor

scala/akka/actor/UidClashTest.scala, line 16 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
15
16 class TerminatedForNonWatchedActor
17 extends Exception("Received Terminated for actor that was not actually watched")
18 with NoStackTrace
19
```

scala/akka/actor/FSMTimingSpec.scala, line 187 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: FSMTimingSpec$Unhandled
File: scala/akka/actor/FSMTimingSpec.scala:187
Taint Flags:

```
184 case object Cancel
185 case object SetHandler
186
187 final case class Unhandled(msg: AnyRef)
188
189 class StateMachine(tester: ActorRef) extends Actor with FSM[State, Int] {
190 import FSM._
```

scala/akka/actor/ActorSelectionSpec.scala, line 20 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorSelectionSpec$GetSender
File: scala/akka/actor/ActorSelectionSpec.scala:20
Taint Flags:

```
17 trait Query
18 final case class SelectString(path: String) extends Query
19 final case class SelectPath(path: ActorPath) extends Query
20 final case class GetSender(to: ActorRef) extends Query
21 final case class Forward(path: String, msg: Any) extends Query
22
23 val p = Props[Node]()
```

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

Package: akka.actor

scala/akka/actor/ActorSelectionSpec.scala, line 20 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

scala/akka/actor/TimerSpec.scala, line 28 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TimerSpec$GotPostStop
File: scala/akka/actor/TimerSpec.scala:28
Taint Flags:

```
25
26  sealed trait Event
27  case class Tock(n: Int) extends Event
28  case class GotPostStop(timerActive: Boolean) extends Event
29  case class GotPreRestart(timerActive: Boolean) extends Event
30
31  class Exc extends RuntimeException("simulated exc") with NoStackTrace
```

scala/akka/actor/ActorSelectionSpec.scala, line 15 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorSelectionSpec$Create
File: scala/akka/actor/ActorSelectionSpec.scala:15
Taint Flags:

```
12
13 object ActorSelectionSpec {
14
15  final case class Create(child: String)
16
17  trait Query
18  final case class SelectString(path: String) extends Query
```

scala/akka/actor/FunctionRefSpec.scala, line 14 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

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

Package: akka.actor

scala/akka/actor/FunctionRefSpec.scala, line 14 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: FunctionRefSpec$GetForwarder
File: scala/akka/actor/FunctionRefSpec.scala:14
Taint Flags:

```
11
12 object FunctionRefSpec {
13
14  case class GetForwarder(replyTo: ActorRef)
15  case class DropForwarder(ref: FunctionRef)
16  case class Forwarded(msg: Any, sender: ActorRef)
17
```

scala/akka/actor/ActorSelectionSpec.scala, line 18 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorSelectionSpec$SelectString
File: scala/akka/actor/ActorSelectionSpec.scala:18
Taint Flags:

```
15  final case class Create(child: String)
16
17  trait Query
18  final case class SelectString(path: String) extends Query
19  final case class SelectPath(path: ActorPath) extends Query
20  final case class GetSender(to: ActorRef) extends Query
21  final case class Forward(path: String, msg: Any) extends Query
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 342 (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.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 342 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: SupervisorHierarchySpec$GCcheck
File: scala/akka/actor/SupervisorHierarchySpec.scala:342
Taint Flags:

```
339 }
340
341 case object Work
342 final case class GCcheck(kids: Vector[WeakReference[ActorRef]])
343
344 sealed trait Action
345 final case class Ping(ref: ActorRef) extends Action
```

scala/akka/actor/TimerSpec.scala, line 27 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TimerSpec$Tock
File: scala/akka/actor/TimerSpec.scala:27
Taint Flags:

```
24 case object AutoReceive extends Command
25
26 sealed trait Event
27 case class Tock(n: Int) extends Event
28 case class GotPostStop(timerActive: Boolean) extends Event
29 case class GotPreRestart(timerActive: Boolean) extends Event
30
```

scala/akka/actor/DeathWatchSpec.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: DeathWatchSpec$U
File: scala/akka/actor/DeathWatchSpec.scala:70
Taint Flags:

```
67 final case class WrappedTerminated(t: Terminated)
68
```

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

Package: akka.actor

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

```
69 final case class W(ref: ActorRef)
70 final case class U(ref: ActorRef)
71 final case class FF(fail: Failed)
72
73 final case class Latches(t1: TestLatch, t2: TestLatch) extends NoSerializationVerificationNeeded
```

scala/akka/actor/DeathWatchSpec.scala, line 73 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: DeathWatchSpec$Latches
File: scala/akka/actor/DeathWatchSpec.scala:73
Taint Flags:

```
70 final case class U(ref: ActorRef)
71 final case class FF(fail: Failed)
72
73 final case class Latches(t1: TestLatch, t2: TestLatch) extends NoSerializationVerificationNeeded
74
75 object WatchWithVerifier {
76 case class WatchThis(ref: ActorRef)
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 63 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: SupervisorHierarchySpec$Died
File: scala/akka/actor/SupervisorHierarchySpec.scala:63
Taint Flags:

```
60 }
61
62 final case class Ready(ref: ActorRef)
63 final case class Died(path: ActorPath)
64 case object Abort
65 case object PingOfDeath
66 case object PongOfDeath
```

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

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 63 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

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

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: DeathWatchSpec$W
File: scala/akka/actor/DeathWatchSpec.scala:69
Taint Flags:

```
66  */
67  final case class WrappedTerminated(t: Terminated)
68
69  final case class W(ref: ActorRef)
70  final case class U(ref: ActorRef)
71  final case class FF(fail: Failed)
72
```

scala/akka/actor/DeathWatchSpec.scala, line 71 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: DeathWatchSpec$FF
File: scala/akka/actor/DeathWatchSpec.scala:71
Taint Flags:

```
68
69  final case class W(ref: ActorRef)
70  final case class U(ref: ActorRef)
71  final case class FF(fail: Failed)
72
73  final case class Latches(t1: TestLatch, t2: TestLatch) extends NoSerializationVerificationNeeded
74
```

scala/akka/actor/TimerSpec.scala, line 23 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

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

Package: akka.actor

scala/akka/actor/TimerSpec.scala, line 23 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TimerSpec$SlowThenThrow
File: scala/akka/actor/TimerSpec.scala:23
Taint Flags:

```
20  case object End extends Command
21  case class Throw(e: Throwable) extends Command
22  case object Cancel extends Command
23  case class SlowThenThrow(latch: TestLatch, e: Throwable) extends Command with NoSerializationVerificationNeeded
24  case object AutoReceive extends Command
25
26  sealed trait Event
```

scala/akka/actor/ExtensionSpec.scala, line 30 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: FailingTestExtension$TestException
File: scala/akka/actor/ExtensionSpec.scala:30
Taint Flags:

```
27  def lookup = this
28  def createExtension(s: ExtendedActorSystem) = new FailingTestExtension(s)
29
30  class TestException extends IllegalArgumentException("ERR") with NoStackTrace
31 }
32
33 object InstanceCountingExtension extends ExtensionId[InstanceCountingExtension] with ExtensionIdProvider {
```

scala/akka/actor/ActorSelectionSpec.scala, line 21 (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.actor

scala/akka/actor/ActorSelectionSpec.scala, line 21 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: ActorSelectionSpec$Forward
File: scala/akka/actor/ActorSelectionSpec.scala:21
Taint Flags:

```
18 final case class SelectString(path: String) extends Query
19 final case class SelectPath(path: ActorPath) extends Query
20 final case class GetSender(to: ActorRef) extends Query
21 final case class Forward(path: String, msg: Any) extends Query
22
23 val p = Props[Node]()
24
```

scala/akka/actor/ActorCreationPerfSpec.scala, line 36 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorCreationPerfSpec$Create
File: scala/akka/actor/ActorCreationPerfSpec.scala:36
Taint Flags:

```
33 }
34 """)
35
36 final case class Create(number: Int, props: () => Props)
37 case object Created
38 case object IsAlive
39 case object Alive
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 129 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: SupervisorHierarchySpec$HierarchyState
File: scala/akka/actor/SupervisorHierarchySpec.scala:129
Taint Flags:

```
126 * upon Restart or would have to be managed by the highest supervisor (which
127 * is undesirable).
```

Code Correctness: Non-Static Inner Class Implements Serializable

Low

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 129 (Code Correctness: Non-Static Inner Class Implements Serializable)

Low

```
128  */
129  final case class HierarchyState(log: Vector[Event], kids: Map[ActorPath, Int], failConstr: Failure)
130  val stateCache = new ConcurrentHashMap[ActorPath, HierarchyState]()
131  @volatile var ignoreFailConstr = false
132
```

scala/akka/actor/TimerSpec.scala, line 21 (Code Correctness: Non-Static Inner Class Implements Serializable)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TimerSpec$Throw
File: scala/akka/actor/TimerSpec.scala:21
Taint Flags:

```
18  case object Bump extends Command
19  case class SlowThenBump(latch: TestLatch) extends Command with NoSerializationVerificationNeeded
20  case object End extends Command
21  case class Throw(e: Throwable) extends Command
22  case object Cancel extends Command
23  case class SlowThenThrow(latch: TestLatch, e: Throwable) extends Command with NoSerializationVerificationNeeded
24  case object AutoReceive extends Command
```

scala/akka/actor/ActorSelectionSpec.scala, line 19 (Code Correctness: Non-Static Inner Class Implements Serializable)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorSelectionSpec$SelectPath
File: scala/akka/actor/ActorSelectionSpec.scala:19
Taint Flags:

```
16
17  trait Query
18  final case class SelectString(path: String) extends Query
19  final case class SelectPath(path: ActorPath) extends Query
20  final case class GetSender(to: ActorRef) extends Query
21  final case class Forward(path: String, msg: Any) extends Query
22
```

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

Package: akka.actor

scala/akka/actor/ActorSelectionSpec.scala, line 19 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

scala/akka/actor/SupervisorHierarchySpec.scala, line 68 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: SupervisorHierarchySpec$ErrorLog
File: scala/akka/actor/SupervisorHierarchySpec.scala:68
Taint Flags:

```
65  case object PingOfDeath
66  case object PongOfDeath
67  final case class Event(msg: Any, identity: Long) { val time: Long = System.nanoTime }
68  final case class ErrorLog(msg: String, log: Vector[Event])
69  final case class Failure(
70  directive: Directive,
71  stop: Boolean,
```

scala/akka/actor/TypedActorSpec.scala, line 247 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TypedActorSpec$WithStringSerializedClass
File: scala/akka/actor/TypedActorSpec.scala:247
Taint Flags:

```
244  }
245  }
246
247  case class WithStringSerializedClass()
248
249 }
250
```

scala/akka/actor/ActorSystemSpec.scala, line 68 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

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

Package: akka.actor

scala/akka/actor/ActorSystemSpec.scala, line 68 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorSystemSpec$FastActor
File: scala/akka/actor/ActorSystemSpec.scala:68
Taint Flags:

```
65 }
66
67 @nowarn
68 final case class FastActor(latch: TestLatch, testActor: ActorRef) extends Actor {
69 val ref1 = context.actorOf(Props.empty)
70 context.actorSelection(ref1.path.toString).tell(Identify(ref1), testActor)
71 latch.countDown()
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 345 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: SupervisorHierarchySpec$Ping
File: scala/akka/actor/SupervisorHierarchySpec.scala:345
Taint Flags:

```
342 final case class GCcheck(kids: Vector[WeakReference[ActorRef]])
343
344 sealed trait Action
345 final case class Ping(ref: ActorRef) extends Action
346 final case class Fail(ref: ActorRef, directive: Directive) extends Action
347
348 sealed trait State
```

scala/akka/actor/Bench.scala, line 13 (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.actor

scala/akka/actor/Bench.scala, line 13 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: Chameneos$Change
File: scala/akka/actor/Bench.scala:13
Taint Flags:

```
10
11  sealed trait ChameneosEvent
12  final case class Meet(from: ActorRef, colour: Colour) extends ChameneosEvent
13  final case class Change(colour: Colour) extends ChameneosEvent
14  final case class MeetingCount(count: Int) extends ChameneosEvent
15  case object Exit extends ChameneosEvent
16
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 67 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: SupervisorHierarchySpec$Event
File: scala/akka/actor/SupervisorHierarchySpec.scala:67
Taint Flags:

```
64  case object Abort
65  case object PingOfDeath
66  case object PongOfDeath
67  final case class Event(msg: Any, identity: Long) { val time: Long = System.nanoTime }
68  final case class ErrorLog(msg: String, log: Vector[Event])
69  final case class Failure(
70  directive: Directive,
```

scala/akka/actor/ActorMailboxSpec.scala, line 215 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorMailboxSpec$MCBoundedMailbox
File: scala/akka/actor/ActorMailboxSpec.scala:215
Taint Flags:

```
212  classOf[UnboundedControlAwareMessageQueueSemantics])
213
```

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

Package: akka.actor

scala/akka/actor/ActorMailboxSpec.scala, line 215 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
214 trait MCBoundedMessageQueueSemantics extends MessageQueue with MultipleConsumerSemantics
215 final case class MCBoundedMailbox(capacity: Int, pushTimeOut: FiniteDuration)
216 extends MailboxType
217 with ProducesMessageQueue[MCBoundedMessageQueueSemantics] {
218
```

scala/akka/actor/UidClashTest.scala, line 22 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: UidClashTest$EvilCollidingActorRef
File: scala/akka/actor/UidClashTest.scala:22
Taint Flags:

```
19
20 @volatile var oldActor: ActorRef = _
21
22 private[akka] class EvilCollidingActorRef(
23 override val provider: ActorRefProvider,
24 override val path: ActorPath,
25 val eventStream: EventStream)
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 62 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: SupervisorHierarchySpec$Ready
File: scala/akka/actor/SupervisorHierarchySpec.scala:62
Taint Flags:

```
59 }
60 }
61
62 final case class Ready(ref: ActorRef)
63 final case class Died(path: ActorPath)
64 case object Abort
65 case object PingOfDeath
```

Code Correctness: Non-Static Inner Class Implements Serializable

Low

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 62 (Code Correctness: Non-Static Inner Class Implements Serializable)

Low

scala/akka/actor/TimerSpec.scala, line 19 (Code Correctness: Non-Static Inner Class Implements Serializable)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TimerSpec$SlowThenBump
File: scala/akka/actor/TimerSpec.scala:19
Taint Flags:

```
16 sealed trait Command
17 case class Tick(n: Int) extends Command
18 case object Bump extends Command
19 case class SlowThenBump(latch: TestLatch) extends Command with NoSerializationVerificationNeeded
20 case object End extends Command
21 case class Throw(e: Throwable) extends Command
22 case object Cancel extends Command
```

scala/akka/actor/DeathWatchSpec.scala, line 78 (Code Correctness: Non-Static Inner Class Implements Serializable)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: DeathWatchSpec$WatchWithVerifier$CustomWatchMsg
File: scala/akka/actor/DeathWatchSpec.scala:78
Taint Flags:

```
75 object WatchWithVerifier {
76 case class WatchThis(ref: ActorRef)
77 case object Watching
78 case class CustomWatchMsg(ref: ActorRef)
79 case class StartStashing(numberOfMessagesToStash: Int)
80 case object StashingStarted
81
```

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

Package: akka.actor.dispatch

scala/akka/actor/dispatch/ActorModelSpec.scala, line 53 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorModelSpec$InterruptNicely
File: scala/akka/actor/dispatch/ActorModelSpec.scala:53
Taint Flags:

```
50
51  case object Interrupt extends ActorModelMessage
52
53  final case class InterruptNicely(expect: Any) extends ActorModelMessage
54
55  case object Restart extends ActorModelMessage
56
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 49 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorModelSpec$WaitAck
File: scala/akka/actor/dispatch/ActorModelSpec.scala:49
Taint Flags:

```
46
47  final case class Wait(time: Long) extends ActorModelMessage
48
49  final case class WaitAck(time: Long, latch: CountDownLatch) extends ActorModelMessage
50
51  case object Interrupt extends ActorModelMessage
52
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 33 (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.actor.dispatch

scala/akka/actor/dispatch/ActorModelSpec.scala, line 33 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: ActorModelSpec$Reply
File: scala/akka/actor/dispatch/ActorModelSpec.scala:33
Taint Flags:

```
30
31  final case class TryReply(expect: Any) extends ActorModelMessage
32
33  final case class Reply(expect: Any) extends ActorModelMessage
34
35  final case class Forward(to: ActorRef, msg: Any) extends ActorModelMessage
36
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 35 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorModelSpec$Forward
File: scala/akka/actor/dispatch/ActorModelSpec.scala:35
Taint Flags:

```
32
33  final case class Reply(expect: Any) extends ActorModelMessage
34
35  final case class Forward(to: ActorRef, msg: Any) extends ActorModelMessage
36
37  final case class CountDown(latch: CountDownLatch) extends ActorModelMessage
38
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 43 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorModelSpec$Meet
File: scala/akka/actor/dispatch/ActorModelSpec.scala:43
Taint Flags:

```
40
41  final case class AwaitLatch(latch: CountDownLatch) extends ActorModelMessage
```

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

Package: akka.actor.dispatch

scala/akka/actor/dispatch/ActorModelSpec.scala, line 43 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
42
43  final case class Meet(acknowledge: CountDownLatch, waitFor: CountDownLatch) extends ActorModelMessage
44
45  final case class CountDownNStop(latch: CountDownLatch) extends ActorModelMessage
46
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 37 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorModelSpec$CountDown
File: scala/akka/actor/dispatch/ActorModelSpec.scala:37
Taint Flags:

```
34
35  final case class Forward(to: ActorRef, msg: Any) extends ActorModelMessage
36
37  final case class CountDown(latch: CountDownLatch) extends ActorModelMessage
38
39  final case class Increment(counter: AtomicLong) extends ActorModelMessage
40
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 47 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorModelSpec$Wait
File: scala/akka/actor/dispatch/ActorModelSpec.scala:47
Taint Flags:

```
44
45  final case class CountDownNStop(latch: CountDownLatch) extends ActorModelMessage
46
47  final case class Wait(time: Long) extends ActorModelMessage
48
49  final case class WaitAck(time: Long, latch: CountDownLatch) extends ActorModelMessage
50
```

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

Package: akka.actor.dispatch

scala/akka/actor/dispatch/ActorModelSpec.scala, line 47 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

scala/akka/actor/dispatch/ActorModelSpec.scala, line 41 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorModelSpec$AwaitLatch
File: scala/akka/actor/dispatch/ActorModelSpec.scala:41
Taint Flags:

```
38
39  final case class Increment(counter: AtomicLong) extends ActorModelMessage
40
41  final case class AwaitLatch(latch: CountDownLatch) extends ActorModelMessage
42
43  final case class Meet(acknowledge: CountDownLatch, waitFor: CountDownLatch) extends ActorModelMessage
44
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 39 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorModelSpec$Increment
File: scala/akka/actor/dispatch/ActorModelSpec.scala:39
Taint Flags:

```
36
37  final case class CountDown(latch: CountDownLatch) extends ActorModelMessage
38
39  final case class Increment(counter: AtomicLong) extends ActorModelMessage
40
41  final case class AwaitLatch(latch: CountDownLatch) extends ActorModelMessage
42
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 59 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

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

Package: akka.actor.dispatch

scala/akka/actor/dispatch/ActorModelSpec.scala, line 59 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorModelSpec$ThrowException
File: scala/akka/actor/dispatch/ActorModelSpec.scala:59
Taint Flags:

```
56
57  case object DoubleStop extends ActorModelMessage
58
59  final case class ThrowException(e: Throwable) extends ActorModelMessage
60
61  val Ping = "Ping"
62  val Pong = "Pong"
```

scala/akka/actor/dispatch/DispatchersSpec.scala, line 97 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: DispatchersSpec$R
File: scala/akka/actor/dispatch/DispatchersSpec.scala:97
Taint Flags:

```
94  }
95
96  // Workaround to narrow the type of unapplySeq of Regex since the unapplySeq(Any) will be removed in Scala 2.13
97  case class R(s: String) {
98  private val r = s.r
99  def unapplySeq(arg: CharSequence) = r.unapplySeq(arg)
100  }
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 45 (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.actor.dispatch

scala/akka/actor/dispatch/ActorModelSpec.scala, line 45 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: ActorModelSpec$CountDownNStop
File: scala/akka/actor/dispatch/ActorModelSpec.scala:45
Taint Flags:

```
42
43  final case class Meet(acknowledge: CountDownLatch, waitFor: CountDownLatch) extends ActorModelMessage
44
45  final case class CountDownNStop(latch: CountDownLatch) extends ActorModelMessage
46
47  final case class Wait(time: Long) extends ActorModelMessage
48
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 31 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorModelSpec$TryReply
File: scala/akka/actor/dispatch/ActorModelSpec.scala:31
Taint Flags:

```
28
29  sealed trait ActorModelMessage extends NoSerializationVerificationNeeded
30
31  final case class TryReply(expect: Any) extends ActorModelMessage
32
33  final case class Reply(expect: Any) extends ActorModelMessage
34
```

Package: akka.event

scala/akka/event/EventStreamSpec.scala, line 38 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: EventStreamSpec$SetTarget
File: scala/akka/event/EventStreamSpec.scala:38
Taint Flags:

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

Package: akka.event

scala/akka/event/EventStreamSpec.scala, line 38 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
35
36  final case class M(i: Int)
37
38  final case class SetTarget(ref: ActorRef)
39
40  class MyLog extends Actor {
41  var dst: ActorRef = context.system.deadLetters
```

scala/akka/event/EventStreamSpec.scala, line 36 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: EventStreamSpec$M
File: scala/akka/event/EventStreamSpec.scala:36
Taint Flags:

```
33  val configUnhandledWithDebug =
34  ConfigFactory.parseString("akka.actor.debug.event-stream = on").withFallback(configUnhandled)
35
36  final case class M(i: Int)
37
38  final case class SetTarget(ref: ActorRef)
39
```

scala/akka/event/LoggerSpec.scala, line 87 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: LoggerSpec$SetTarget
File: scala/akka/event/LoggerSpec.scala:87
Taint Flags:

```
84  }
85  """).withFallback(AkkaSpec.testConf)
86
87  final case class SetTarget(ref: ActorRef, qualifier: Int)
88
```

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

Package: akka.event

scala/akka/event/LoggerSpec.scala, line 87 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
89  class TestLogger1 extends TestLogger(1)
90  class TestLogger2 extends TestLogger(2)
```

scala/akka/event/EventBusSpec.scala, line 165 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorEventBusSpec$Notification
File: scala/akka/event/EventBusSpec.scala:165
Taint Flags:

```
162  def publish(event: Event, subscriber: Subscriber) = subscriber ! event
163  }
164
165  case class Notification(ref: ActorRef, payload: Int)
166 }
167
168 class ActorEventBusSpec(conf: Config) extends EventBusSpec("ActorEventBus", conf) {
```

Package: akka.event.jul

scala/akka/event/jul/JavaLoggerSpec.scala, line 35 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: JavaLoggerSpec$SimulatedExc
File: scala/akka/event/jul/JavaLoggerSpec.scala:35
Taint Flags:

```
32  }
33  }
34
35  class SimulatedExc extends RuntimeException("Simulated error") with NoStackTrace
36 }
37
38 @deprecated("Use SLF4J instead.", "2.6.0")
```

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

Package: akka.io

scala/akka/io/TcpConnectionSpec.scala, line 39 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TcpConnectionSpec$Registration
File: scala/akka/io/TcpConnectionSpec.scala:39
Taint Flags:

```
36 object TcpConnectionSpec {
37  case class Ack(i: Int) extends Event
38  object Ack extends Ack(0)
39  final case class Registration(channel: SelectableChannel, initialOps: Int) extends NoSerializationVerificationNeeded
40 }
41
42 class TcpConnectionSpec extends AkkaSpec("""
```

scala/akka/io/TcpListenerSpec.scala, line 200 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TcpListenerSpec$RegisterChannel
File: scala/akka/io/TcpListenerSpec.scala:200
Taint Flags:

```
197
198 }
199 object TcpListenerSpec {
200  final case class RegisterChannel(channel: SelectableChannel, initialOps: Int)
201  extends NoSerializationVerificationNeeded
202 }
203
```

scala/akka/io/TcpConnectionSpec.scala, line 37 (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.io

scala/akka/io/TcpConnectionSpec.scala, line 37 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: TcpConnectionSpec$Ack
File: scala/akka/io/TcpConnectionSpec.scala:37
Taint Flags:

```
34 import akka.util.{ ByteString, Helpers }
35
36 object TcpConnectionSpec {
37  case class Ack(i: Int) extends Event
38  object Ack extends Ack(0)
39  final case class Registration(channel: SelectableChannel, initialOps: Int) extends NoSerializationVerificationNeeded
40 }
```

Package: akka.pattern

scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala, line 25 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TestActor$NormalException
File: scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala:25
Taint Flags:

```
22
23  class StoppingException extends TestException("stopping exception")
24
25  class NormalException extends TestException("normal exception")
26
27  def props(probe: ActorRef): Props = Props(new TestActor(probe))
28 }
```

scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala, line 21 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TestActor$TestException
File: scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala:21
Taint Flags:

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

Package: akka.pattern

scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala, line 21 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
18
19 object TestActor {
20
21  class TestException(msg: String) extends Exception(msg)
22
23  class StoppingException extends TestException("stopping exception")
24
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 23 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: CircuitBreakerSpec$TestException
File: scala/akka/pattern/CircuitBreakerSpec.scala:23
Taint Flags:

```
20
21 object CircuitBreakerSpec {
22
23  class TestException extends RuntimeException
24  case class CBSuccess(value: FiniteDuration)
25  case class CBFailure(value: FiniteDuration)
26  case class CBTimeout(value: FiniteDuration)
```

scala/akka/pattern/PromiseRefSpec.scala, line 14 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: PromiseRefSpec$Request
File: scala/akka/pattern/PromiseRefSpec.scala:14
Taint Flags:

```
11 import akka.testkit.{ AkkaSpec, ImplicitSender, TestProbe }
12
13 object PromiseRefSpec {
14  case class Request(replyTo: ActorRef)
15  case object Response
```

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

Package: akka.pattern

scala/akka/pattern/PromiseRefSpec.scala, line 14 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
16
17  case object FirstMessage
```

scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala, line 23 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TestActor$StoppingException
File: scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala:23
Taint Flags:

```
20
21  class TestException(msg: String) extends Exception(msg)
22
23  class StoppingException extends TestException("stopping exception")
24
25  class NormalException extends TestException("normal exception")
26
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 25 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: CircuitBreakerSpec$CBFailure
File: scala/akka/pattern/CircuitBreakerSpec.scala:25
Taint Flags:

```
22
23  class TestException extends RuntimeException
24  case class CBSuccess(value: FiniteDuration)
25  case class CBFailure(value: FiniteDuration)
26  case class CBTimeout(value: FiniteDuration)
27
28  class Breaker(val instance: CircuitBreaker)(implicit system: ActorSystem) {
```

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

Package: akka.pattern

scala/akka/pattern/PatternSpec.scala, line 17 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: PatternSpec$Work
File: scala/akka/pattern/PatternSpec.scala:17
Taint Flags:

```
14 import akka.testkit.{ AkkaSpec, TestLatch }
15
16 object PatternSpec {
17  final case class Work(duration: Duration)
18  class TargetActor extends Actor {
19  def receive = {
20  case (testLatch: TestLatch, duration: FiniteDuration) =>
```

scala/akka/pattern/BackoffSupervisorSpec.scala, line 18 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: BackoffSupervisorSpec$TestException
File: scala/akka/pattern/BackoffSupervisorSpec.scala:18
Taint Flags:

```
15
16 object BackoffSupervisorSpec {
17
18  class TestException extends RuntimeException with NoStackTrace
19
20  object Child {
21  def props(probe: ActorRef): Props =
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 24 (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.pattern

scala/akka/pattern/CircuitBreakerSpec.scala, line 24 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: CircuitBreakerSpec$CBSuccess
File: scala/akka/pattern/CircuitBreakerSpec.scala:24
Taint Flags:

```
21 object CircuitBreakerSpec {
22
23  class TestException extends RuntimeException
24  case class CBSuccess(value: FiniteDuration)
25  case class CBFailure(value: FiniteDuration)
26  case class CBTimeout(value: FiniteDuration)
27
```

scala/akka/pattern/CircuitBreakerStressSpec.scala, line 23 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: CircuitBreakerStressSpec$Result
File: scala/akka/pattern/CircuitBreakerStressSpec.scala:23
Taint Flags:

```
20 object CircuitBreakerStressSpec {
21  case object JobDone
22  case object GetResult
23  case class Result(doneCount: Int, timeoutCount: Int, failCount: Int, circCount: Int)
24
25  class StressActor(breaker: CircuitBreaker) extends Actor with ActorLogging with PipeToSupport {
26  import context.dispatcher
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 26 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: CircuitBreakerSpec$CBTimeout
File: scala/akka/pattern/CircuitBreakerSpec.scala:26
Taint Flags:

```
23  class TestException extends RuntimeException
24  case class CBSuccess(value: FiniteDuration)
```

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

Package: akka.pattern

scala/akka/pattern/CircuitBreakerSpec.scala, line 26 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
25  case class CBFailure(value: FiniteDuration)
26  case class CBTimeout(value: FiniteDuration)
27
28  class Breaker(val instance: CircuitBreaker)(implicit system: ActorSystem) {
29  val probe = TestProbe()
```

Package: akka.pattern.extended

scala/akka/pattern/extended/ExplicitAskSpec.scala, line 16 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ExplicitAskSpec$Response
File: scala/akka/pattern/extended/ExplicitAskSpec.scala:16
Taint Flags:

```
13
14 object ExplicitAskSpec {
15  case class Request(respondTo: ActorRef)
16  case class Response(sentFrom: ActorRef)
17 }
18
19 class ExplicitAskSpec extends AkkaSpec {
```

scala/akka/pattern/extended/ExplicitAskSpec.scala, line 15 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ExplicitAskSpec$Request
File: scala/akka/pattern/extended/ExplicitAskSpec.scala:15
Taint Flags:

```
12 import akka.util.Timeout
13
14 object ExplicitAskSpec {
15  case class Request(respondTo: ActorRef)
16  case class Response(sentFrom: ActorRef)
17 }
```

Code Correctness: Non-Static Inner Class Implements Serializable

Low

Package: akka.pattern.extended

scala/akka/pattern/extended/ExplicitAskSpec.scala, line 15 (Code Correctness: Non-Static Inner Class Implements Serializable)

Low

```
18
```

Package: akka.routing

scala/akka/routing/ConfiguredLocalRoutingSpec.scala, line 71 (Code Correctness: Non-Static Inner Class Implements Serializable)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ConfiguredLocalRoutingSpec$MyRouter
File: scala/akka/routing/ConfiguredLocalRoutingSpec.scala:71
Taint Flags:

```
68 }
69 """
70
71 class MyRouter(config: Config) extends CustomRouterConfig {
72 override def createRouter(system: ActorSystem): Router = Router(MyRoutingLogic(config))
73 }
74
```

scala/akka/routing/ConsistentHashingRouterSpec.scala, line 47 (Code Correctness: Non-Static Inner Class Implements Serializable)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ConsistentHashingRouterSpec$Msg
File: scala/akka/routing/ConsistentHashingRouterSpec.scala:47
Taint Flags:

```
44 }
45 }
46
47 final case class Msg(key: Any, data: String) extends ConsistentHashable {
48 override def consistentHashKey = key
49 }
50
```

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

Package: akka.routing

scala/akka/routing/ConsistentHashingRouterSpec.scala, line 51 (Code Correctness: Non-Static Inner Class Implements Serializable) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ConsistentHashingRouterSpec$MsgKey
File: scala/akka/routing/ConsistentHashingRouterSpec.scala:51
Taint Flags:

```
48  override def consistentHashKey = key
49  }
50
51  final case class MsgKey(name: String)
52
53  final case class Msg2(key: Any, data: String)
54 }
```

scala/akka/routing/ScatterGatherFirstCompletedSpec.scala, line 25 (Code Correctness: Non-Static Inner Class Implements Serializable) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ScatterGatherFirstCompletedSpec$Stop
File: scala/akka/routing/ScatterGatherFirstCompletedSpec.scala:25
Taint Flags:

```
22  def receive = { case _ => }
23  }
24
25  final case class Stop(id: Option[Int] = None)
26
27  def newActor(id: Int, shudownLatch: Option[TestLatch] = None)(implicit system: ActorSystem) =
28  system.actorOf(
```

scala/akka/routing/MetricsBasedResizerSpec.scala, line 44 (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.routing

scala/akka/routing/MetricsBasedResizerSpec.scala, line 44 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: MetricsBasedResizerSpec$TestRouter
File: scala/akka/routing/MetricsBasedResizerSpec.scala:44
Taint Flags:

```
41
42  def routees(num: Int = 10)(implicit system: ActorSystem, timeout: Timeout) = (1 to num).map(_ => routee).toVector
43
44  case class TestRouter(routees: Vector[ActorRefRoutee])(implicit system: ActorSystem, timeout: Timeout) {
45
46  var msgs: Set[TestLatch] = Set()
47
```

scala/akka/routing/MetricsBasedResizerSpec.scala, line 24 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: MetricsBasedResizerSpec$Latches
File: scala/akka/routing/MetricsBasedResizerSpec.scala:24
Taint Flags:

```
21
22 object MetricsBasedResizerSpec {
23
24  case class Latches(first: TestLatch, second: TestLatch)
25
26  /**
27  * The point of these Actors is that their mailbox size will be queried
```

scala/akka/routing/ConfiguredLocalRoutingSpec.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: ConfiguredLocalRoutingSpec$MyRoutingLogic
File: scala/akka/routing/ConfiguredLocalRoutingSpec.scala:75
Taint Flags:

```
72  override def createRouter(system: ActorSystem): Router = Router(MyRoutingLogic(config))
73  }
```

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

Package: akka.routing

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

```
74
75  final case class MyRoutingLogic(config: Config) extends RoutingLogic {
76  override def select(message: Any, routees: immutable.IndexedSeq[Routee]): Routee =
77  MyRoutee(config.getString(message.toString))
78  }
```

scala/akka/routing/ConfiguredLocalRoutingSpec.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: ConfiguredLocalRoutingSpec$MyRoutee
File: scala/akka/routing/ConfiguredLocalRoutingSpec.scala:80
Taint Flags:

```
77  MyRoutee(config.getString(message.toString))
78  }
79
80  final case class MyRoutee(reply: String) extends Routee {
81  override def send(message: Any, sender: ActorRef): Unit =
82  sender ! reply
83  }
```

scala/akka/routing/ConsistentHashingRouterSpec.scala, line 53 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ConsistentHashingRouterSpec$Msg2
File: scala/akka/routing/ConsistentHashingRouterSpec.scala:53
Taint Flags:

```
50
51  final case class MsgKey(name: String)
52
53  final case class Msg2(key: Any, data: String)
54 }
55
56 class ConsistentHashingRouterSpec
```

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

Package: akka.routing

scala/akka/routing/ConsistentHashingRouterSpec.scala, line 53 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Package: akka.serialization

scala/akka/serialization/DisabledJavaSerializerWarningSpec.scala, line 15 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: DisabledJavaSerializerWarningSpec$Msg
File: scala/akka/serialization/DisabledJavaSerializerWarningSpec.scala:15
Taint Flags:

```
12 import akka.testkit._
13
14 object DisabledJavaSerializerWarningSpec {
15   final case class Msg(s: String)
16 }
17
18 class DisabledJavaSerializerWarningSpec extends AkkaSpec("""
```

scala/akka/serialization/AsyncSerializeSpec.scala, line 18 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: AsyncSerializeSpec$Message3
File: scala/akka/serialization/AsyncSerializeSpec.scala:18
Taint Flags:

```
15
16   case class Message1(str: String)
17   case class Message2(str: String)
18   case class Message3(str: String)
19   case class Message4(str: String)
20
21   val config = ConfigFactory.parseString(s"""
```

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

Package: akka.serialization

scala/akka/serialization/AsyncSerializeSpec.scala, line 17 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: AsyncSerializeSpec$Message2
File: scala/akka/serialization/AsyncSerializeSpec.scala:17
Taint Flags:

```
14 object AsyncSerializeSpec {
15
16  case class Message1(str: String)
17  case class Message2(str: String)
18  case class Message3(str: String)
19  case class Message4(str: String)
20
```

scala/akka/serialization/AsyncSerializeSpec.scala, line 19 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: AsyncSerializeSpec$Message4
File: scala/akka/serialization/AsyncSerializeSpec.scala:19
Taint Flags:

```
16  case class Message1(str: String)
17  case class Message2(str: String)
18  case class Message3(str: String)
19  case class Message4(str: String)
20
21  val config = ConfigFactory.parseString(s"""
22  akka {
```

scala/akka/serialization/AsyncSerializeSpec.scala, line 16 (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.serialization

scala/akka/serialization/AsyncSerializeSpec.scala, line 16 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: AsyncSerializeSpec$Message1
File: scala/akka/serialization/AsyncSerializeSpec.scala:16
Taint Flags:

```
13
14 object AsyncSerializeSpec {
15
16  case class Message1(str: String)
17  case class Message2(str: String)
18  case class Message3(str: String)
19  case class Message4(str: String)
```

Package: akka.util

scala-2.13/akka/util/TypedMultiMapSpec.scala, line 13 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TypedMultiMapSpec$Key
File: scala-2.13/akka/util/TypedMultiMapSpec.scala:13
Taint Flags:

```
10
11 object TypedMultiMapSpec {
12  trait AbstractKey { type Type }
13  final case class Key[T](t: T) extends AbstractKey { final override type Type = T }
14  final case class MyValue[T](t: T)
15
16  type KV[K <: AbstractKey] = MyValue[K#Type]
```

scala-2.13/akka/util/TypedMultiMapSpec.scala, line 14 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TypedMultiMapSpec$MyValue
File: scala-2.13/akka/util/TypedMultiMapSpec.scala:14
Taint Flags:

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

Package: akka.util

scala-2.13/akka/util/TypedMultiMapSpec.scala, line 14 (Code Correctness: Non-Static Inner Class Implements Serializable) | Low

```
11 object TypedMultiMapSpec {
12  trait AbstractKey { type Type }
13  final case class Key[T](t: T) extends AbstractKey { final override type Type = T }
14  final case class MyValue[T](t: T)
15
16  type KV[K <: AbstractKey] = MyValue[K#Type]
17 }
```

scala/akka/util/MessageBufferSpec.scala, line 172 (Code Correctness: Non-Static Inner Class Implements Serializable) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: MessageBufferSpec$DummyActorRef
File: scala/akka/util/MessageBufferSpec.scala:172
Taint Flags:

```
169 }
170
171 object MessageBufferSpec {
172  final private[akka] class DummyActorRef(val id: String) extends MinimalActorRef {
173
174  override def toString: String = id
175
```

scala/akka/util/ByteStringSpec.scala, line 60 (Code Correctness: Non-Static Inner Class Implements Serializable) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ByteStringSpec$ByteStringGrouped
File: scala/akka/util/ByteStringSpec.scala:60
Taint Flags:

```
57  } yield (xs, from, until)
58  }
59
60  case class ByteStringGrouped(bs: ByteString, size: Int)
61
```

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

Package: akka.util

scala/akka/util/ByteStringSpec.scala, line 60 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
62 implicit val arbitraryByteStringGrouped: Arbitrary[ByteStringGrouped] = Arbitrary {
63 for {
```

scala/akka/util/BoundedBlockingQueueSpec.scala, line 810 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: QueueSetupHelper$TestCondition$Manual
File: scala/akka/util/BoundedBlockingQueueSpec.scala:810
Taint Flags:

```
807 awaitEvent: QueueEvent)
808 extends Condition {
809
810 case class Manual(waitTime: Long = 0, waitingThread: Option[Thread] = None)
811
812 @volatile private var waiting: Option[Manual] = None
813
```

scala/akka/util/BoundedBlockingQueueSpec.scala, line 782 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: QueueSetupHelper$TestBackingQueue
File: scala/akka/util/BoundedBlockingQueueSpec.scala:782
Taint Flags:

```
779 /**
780 * Backing queue that records all poll and offer calls in `events`
781 */
782 class TestBackingQueue(events: mutable.Buffer[QueueEvent]) extends util.LinkedList[String] {
783
784 override def poll(): String = {
785 events += Poll()
```

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

Package: akka.util

scala/akka/util/BoundedBlockingQueueSpec.scala, line 771 (Code Correctness: Non-Static Inner Class Implements Serializable) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: QueueSetupHelper$TestContext
File: scala/akka/util/BoundedBlockingQueueSpec.scala:771
Taint Flags:

```
768
769  import akka.util.QueueTestEvents._
770
771  case class TestContext(
772  queue: BoundedBlockingQueue[String],
773  events: mutable.Buffer[QueueEvent],
774  notEmpty: TestCondition,
```

scala/akka/util/DoubleLinkedListSpec.scala, line 11 (Code Correctness: Non-Static Inner Class Implements Serializable) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: DoubleLinkedListSpec$Node
File: scala/akka/util/DoubleLinkedListSpec.scala:11
Taint Flags:

```
8 import org.scalatest.wordspec.AnyWordSpec
9
10 object DoubleLinkedListSpec {
11  private case class Node(value: String) {
12  var less, more: OptionVal[Node] = OptionVal.None
13  }
14 }
```

Dead Code: Expression is Always false (173 issues)

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();
  }
}
```

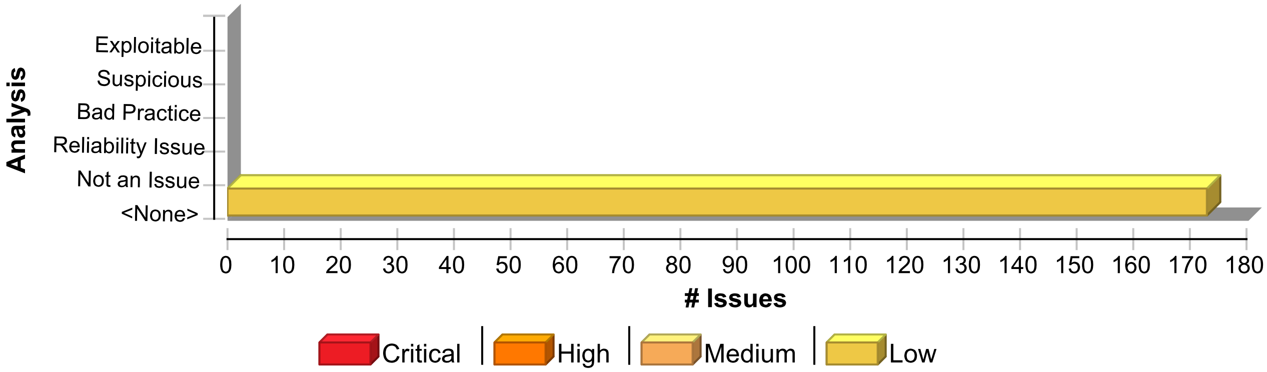
Example 2: The following method never sets the variable `firstCall` to `true`. (The variable `firstCall` is mistakenly set to `false` after the first conditional statement.) The result is that the first part of the expression `firstCall && secondCall` will always evaluate to `false`.

```
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	173	0	0	173
Total	**173**	**0**	**0**	**173**

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 791 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:791
Taint Flags:

```
788 context.actorOf(Props(new CountDownActor(countDownMessages, SupervisorStrategy.defaultStrategy))))
789
790 def receive = {
791 case "killCrasher" => crasher ! Kill
792 case Terminated(_) => countDownMax.countDown()
793 }
794 }))
```

scala/akka/actor/ActorSystemDispatcherSpec.scala, line 52 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/ActorSystemDispatcherSpec.scala, line 52 (Dead Code: Expression is Always false) — Low

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorSystemDispatcherSpec.scala:52
Taint Flags:

```
49 try {
50 val ref = system2.actorOf(Props(new Actor {
51 def receive = {
52 case "ping" => sender() ! "pong"
53 }
54 }))
55
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 56 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:56
Taint Flags:

```
53 class Resumer extends Actor {
54 override def supervisorStrategy = OneForOneStrategy() { case _ => SupervisorStrategy.Resume }
55 def receive = {
56 case "spawn" => sender() ! context.actorOf(Props[Resumer]())
57 case "fail" => throw new Exception("expected")
58 case "ping" => sender() ! "pong"
59 }
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 58 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 58 (Dead Code: Expression is Always false) — Low

File: scala/akka/actor/SupervisorHierarchySpec.scala:58
Taint Flags:

```
55 def receive = {
56 case "spawn" => sender() ! context.actorOf(Props[Resumer]())
57 case "fail" => throw new Exception("expected")
58 case "ping" => sender() ! "pong"
59 }
60 }
61
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 671 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:671
Taint Flags:

```
668 printErrors()
669 testActor ! "timeout in Failed"
670 stop()
671 case this.Event("pong", _) => stay() // don’t care?
672 case this.Event(Work, _) => stay()
673 case this.Event(Died(_), _) => stay()
674 }
```

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 28 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorWithBoundedStashSpec.scala:28
Taint Flags:

```
25 stash()
```

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/ActorWithBoundedStashSpec.scala, line 28 (Dead Code: Expression is Always false) — Low

```
26 sender() ! "ok"
27
28 case "world" =>
29 context.become(afterWorldBehavior)
30 unstashAll()
31
```

scala/akka/actor/ActorRefSpec.scala, line 62 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:62
Taint Flags:

```
59
60 def receive = {
61 case "complex" => replyActor ! "complexRequest"
62 case "complex2" => replyActor ! "complexRequest2"
63 case "simple" => replyActor ! "simpleRequest"
64 case "complexReply" => {
65 latch.countDown()
```

scala/akka/actor/HotSwapSpec.scala, line 105 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/HotSwapSpec.scala:105
Taint Flags:

```
102 val a = system.actorOf(Props(new Actor {
103 def receive = {
104 case "state" => sender() ! "0"
105 case "swap" =>
106 context.become({
107 case "state" => sender() ! "1"
```

Dead Code: Expression is Always false

Low

Package: akka.actor

scala/akka/actor/HotSwapSpec.scala, line 105 (Dead Code: Expression is Always false)

Low

```
108 case "swapped" => sender() ! "swapped"
```

scala/akka/actor/FSMTransitionSpec.scala, line 151 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMTransitionSpec.scala:151
Taint Flags:

```
148 val fsmref = system.actorOf(Props(new Actor with FSM[Int, ActorRef] {
149 startWith(0, null)
150 when(0) {
151 case Event("switch", _) => goto(1).using(sender())
152 }
153 onTransition {
154 case x -> y => nextStateData ! (x -> y)
```

scala/akka/actor/FSMActorSpec.scala, line 310 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:310
Taint Flags:

```
307 override def logDepth = 3
308 startWith(1, 0)
309 when(1) {
310 case Event("count", c) => stay().using(c + 1)
311 case Event("log", _) => stay().replying(getLog)
312 }
313 })
```

scala/akka/actor/ActorSystemSpec.scala, line 323 (Dead Code: Expression is Always false)

Low

Issue Details

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/ActorSystemSpec.scala, line 323 (Dead Code: Expression is Always false) Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorSystemSpec.scala:323
Taint Flags:

```
320 .withFallback(AkkaSpec.testConf))
321 val a = system.actorOf(Props(new Actor {
322 def receive = {
323 case "die" => throw new Exception("hello")
324 }
325 }))
326 val probe = TestProbe()
```

scala/akka/actor/SupervisorSpec.scala, line 465 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorSpec.scala:465
Taint Flags:

```
462 def receive = {
463 case Terminated(t) if t.path == child.path => testActor ! "child terminated"
464 case l: TestLatch => child ! l
465 case "test" => sender() ! "green"
466 case "testchild" => child.forward("test")
467 case "testchildAndAck" => child.forward("test"); sender() ! "ack"
468 }
```

scala/akka/actor/FSMActorSpec.scala, line 227 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/FSMActorSpec.scala, line 227 (Dead Code: Expression is Always false) — Low

File: scala/akka/actor/FSMActorSpec.scala:227
Taint Flags:

```
224 case Event("stop", _) => stop()
225 }
226 onTransition {
227 case "not-started" -> "started" =>
228 for (timerName <- timerNames) startSingleTimer(timerName, (), 10 seconds)
229 }
230 onTermination {
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 829 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:829
Taint Flags:

```
826 case _ => Await.ready(latch, 4.seconds.dilated); SupervisorStrategy.Resume
827 }
828 def receive = {
829 case "spawn" => sender() ! context.actorOf(Props[Resumer]())
830 }
831 }), "slowResumer")
832 slowResumer ! "spawn"
```

scala/akka/actor/FSMTransitionSpec.scala, line 21 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMTransitionSpec.scala:21
Taint Flags:

```
18 class SendAnyTransitionFSM(target: ActorRef) extends Actor with FSM[Int, Int] {
19 startWith(0, 0)
```

Dead Code: Expression is Always false

Low

Package: akka.actor

scala/akka/actor/FSMTransitionSpec.scala, line 21 (Dead Code: Expression is Always false)

Low

```
20  when(0) {
21  case Event("stay", _) => stay()
22  case Event(_, _) => goto(0)
23  }
24  onTransition { case from -> to => target ! (from -> to) }
```

scala/akka/actor/LocalActorRefProviderSpec.scala, line 150 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/LocalActorRefProviderSpec.scala:150
Taint Flags:

```
147  "only create one instance of an actor from within the same message invocation" in {
148  val supervisor = system.actorOf(Props(new Actor {
149  def receive = {
150  case "" =>
151  val a, b = context.actorOf(Props.empty, "duplicate")
152  }
153  }))
```

scala/akka/actor/HotSwapSpec.scala, line 83 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/HotSwapSpec.scala:83
Taint Flags:

```
80  case "swap" =>
81  context.become({
82  case "swapped" => sender() ! "swapped"
83  case "revert" => context.unbecome()
84  })
85  }
```

Dead Code: Expression is Always false

Low

Package: akka.actor

scala/akka/actor/HotSwapSpec.scala, line 83 (Dead Code: Expression is Always false)

Low

```
86 }))
```

scala/akka/actor/ActorWithStashSpec.scala, line 41 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorWithStashSpec.scala:41
Taint Flags:

```
38
39 class StashingTwiceActor extends Actor with Stash {
40 def receive = {
41 case "hello" =>
42 try {
43 stash()
44 stash()
```

scala/akka/actor/ActorSystemSpec.scala, line 57 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorSystemSpec.scala:57
Taint Flags:

```
54
55 class Terminater extends Actor {
56 def receive = {
57 case "run" => context.stop(self)
58 }
59 }
60
```

scala/akka/actor/SupervisorMiscSpec.scala, line 162 (Dead Code: Expression is Always false)

Low

Issue Details

Dead Code: Expression is Always false

Low

Package: akka.actor

scala/akka/actor/SupervisorMiscSpec.scala, line 162 (Dead Code: Expression is Always false)

Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorMiscSpec.scala:162
Taint Flags:

```
159 case _: Exception => testActor ! sender(); SupervisorStrategy.Stop
160 }
161 def receive = {
162 case "doit" => context.actorOf(Props.empty, "child") ! Kill
163 }
164 }))
165 EventFilter[ActorKilledException](occurrences = 1).intercept {
```

scala/akka/actor/SupervisorMiscSpec.scala, line 47 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorMiscSpec.scala:47
Taint Flags:

```
44 val workerProps = Props(new Actor {
45 override def postRestart(cause: Throwable): Unit = { countDownLatch.countDown() }
46 def receive = {
47 case "status" => this.sender() ! "OK"
48 case _ => this.context.stop(self)
49 }
50 })
```

scala/akka/actor/ActorRefSpec.scala, line 36 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/ActorRefSpec.scala, line 36 (Dead Code: Expression is Always false) — Low

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:36
Taint Flags:

```
33 case "complexRequest2" =>
34 val worker = context.actorOf(Props[WorkerActor]())
35 worker ! ReplyTo(sender())
36 case "workDone" => replyTo ! "complexReply"
37 case "simpleRequest" => sender() ! "simpleReply"
38 }
39 }
```

scala/akka/actor/ActorRefSpec.scala, line 37 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:37
Taint Flags:

```
34 val worker = context.actorOf(Props[WorkerActor]())
35 worker ! ReplyTo(sender())
36 case "workDone" => replyTo ! "complexReply"
37 case "simpleRequest" => sender() ! "simpleReply"
38 }
39 }
40
```

scala/akka/actor/ActorWithStashSpec.scala, line 60 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorWithStashSpec.scala:60
Taint Flags:

```
57 unstashAll()
```

Dead Code: Expression is Always false

Low

Package: akka.actor

scala/akka/actor/ActorWithStashSpec.scala, line 60 (Dead Code: Expression is Always false)

Low

```
58  context.become {
59  case "write" => // do writing...
60  case "close" =>
61  unstashAll()
62  context.unbecome()
63  case _ => stash()
```

scala/akka/actor/FSMActorSpec.scala, line 311 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:311
Taint Flags:

```
308  startWith(1, 0)
309  when(1) {
310  case Event("count", c) => stay().using(c + 1)
311  case Event("log", _) => stay().replying(getLog)
312  }
313  })
314  fsmref ! "log"
```

scala/akka/actor/ActorSystemDispatcherSpec.scala, line 127 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorSystemDispatcherSpec.scala:127
Taint Flags:

```
124  try {
125  val ref = system2.actorOf(Props(new Actor {
126  def receive = {
127  case "ping" => sender() ! "pong"
128  }
```

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/ActorSystemDispatcherSpec.scala, line 127 (Dead Code: Expression is Always false) — Low

```
129 }).withDispatcher(Dispatchers.InternalDispatcherId))
130
```

scala/akka/actor/ActorRefSpec.scala, line 44 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:44
Taint Flags:

```
41 class WorkerActor() extends Actor {
42 import context.system
43 def receive = {
44 case "work" => {
45 work()
46 sender() ! "workDone"
47 context.stop(self)
```

scala/akka/actor/FSMActorSpec.scala, line 224 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:224
Taint Flags:

```
221 case Event("start", _) => goto("started").replying("starting")
222 }
223 when("started", stateTimeout = 10 seconds) {
224 case Event("stop", _) => stop()
225 }
226 onTransition {
227 case "not-started" -> "started" =>
```

scala/akka/actor/ActorRefSpec.scala, line 64 (Dead Code: Expression is Always false) — Low

Issue Details

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/ActorRefSpec.scala, line 64 (Dead Code: Expression is Always false) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:64
Taint Flags:

```
61  case "complex" => replyActor ! "complexRequest"
62  case "complex2" => replyActor ! "complexRequest2"
63  case "simple" => replyActor ! "simpleRequest"
64  case "complexReply" => {
65  latch.countDown()
66  }
67  case "simpleReply" => {
```

scala/akka/actor/SupervisorMiscSpec.scala, line 110 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorMiscSpec.scala:110
Taint Flags:

```
107  } catch {
108  case NonFatal(e) => testActor ! e
109  }
110  case "engage" => context.stop(kid)
111  }
112  }))
113  parent ! "engage"
```

scala/akka/actor/ActorLifeCycleSpec.scala, line 141 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/ActorLifeCycleSpec.scala, line 141 (Dead Code: Expression is Always false) — Low

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorLifeCycleSpec.scala:141
Taint Flags:

```
138 a ! "hello"
139 expectMsg(42)
140 a ! Become(ctx => {
141 case "fail" => throw new RuntimeException("buh")
142 case _ => ctx.sender() ! 43
143 })
144 expectMsg("ok")
```

scala/akka/actor/ActorRefSpec.scala, line 84 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:84
Taint Flags:

```
81 val fail = new InnerActor
82
83 def receive = {
84 case "self" => sender() ! self
85 case x => inner.forward(x)
86 }
87 }
```

scala/akka/actor/ActorWithStashSpec.scala, line 65 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorWithStashSpec.scala:65
Taint Flags:

Dead Code: Expression is Always false Low

Package: akka.actor

scala/akka/actor/ActorWithStashSpec.scala, line 65 (Dead Code: Expression is Always false) Low

```
62 context.unbecome()
63 case _ => stash()
64 }
65 case "done" => state.finished.await()
66 case _ => stash()
67 }
68 }
```

scala/akka/actor/ActorWithStashSpec.scala, line 150 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorWithStashSpec.scala:150
Taint Flags:

```
147
148 val employeeProps = Props(new Actor with Stash {
149 def receive = {
150 case "crash" =>
151 throw new Exception("Crashing...")
152
153 // when restartLatch is not yet open, stash all messages != "crash"
```

scala/akka/actor/FSMActorSpec.scala, line 265 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:265
Taint Flags:

```
262 val fsm = TestActorRef(new Actor with LoggingFSM[Int, Null] {
263 startWith(1, null)
264 when(1) {
265 case Event("go", _) =>
```

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/FSMActorSpec.scala, line 265 (Dead Code: Expression is Always false) — Low

```
266 startSingleTimer("t", FSM.Shutdown, 1.5 seconds)
267 goto(2)
268 }
```

scala/akka/actor/ActorWithStashSpec.scala, line 59 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorWithStashSpec.scala:59
Taint Flags:

```
56 case "open" =>
57 unstashAll()
58 context.become {
59 case "write" => // do writing...
60 case "close" =>
61 unstashAll()
62 context.unbecome()
```

scala/akka/actor/HotSwapSpec.scala, line 104 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/HotSwapSpec.scala:104
Taint Flags:

```
101
102 val a = system.actorOf(Props(new Actor {
103 def receive = {
104 case "state" => sender() ! "0"
105 case "swap" =>
106 context.become({
107 case "state" => sender() ! "1"
```

Dead Code: Expression is Always false

Low

Package: akka.actor

scala/akka/actor/SupervisorSpec.scala, line 237 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorSpec.scala:237
Taint Flags:

```
234 override def postStop(): Unit = { postStops += 1; testActor ! ("postStop" + postStops) }
235 def receive = {
236 case "crash" => { testActor ! "crashed"; throw new RuntimeException("Expected") }
237 case "ping" => sender() ! "pong"
238 }
239 }
240 val master = system.actorOf(Props(new Actor {
```

scala/akka/actor/ActorRefSpec.scala, line 28 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:28
Taint Flags:

```
25 var replyTo: ActorRef = null
26
27 def receive = {
28 case "complexRequest" => {
29 replyTo = sender()
30 val worker = context.actorOf(Props[WorkerActor]())
31 worker ! "work"
```

scala/akka/actor/ActorRefSpec.scala, line 67 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/ActorRefSpec.scala, line 67 (Dead Code: Expression is Always false) — Low

Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:67
Taint Flags:

```
64 case "complexReply" => {
65 latch.countDown()
66 }
67 case "simpleReply" => {
68 latch.countDown()
69 }
70 }
```

scala/akka/actor/HotSwapSpec.scala, line 107 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/HotSwapSpec.scala:107
Taint Flags:

```
104 case "state" => sender() ! "0"
105 case "swap" =>
106 context.become({
107 case "state" => sender() ! "1"
108 case "swapped" => sender() ! "swapped"
109 case "crash" => throw new Exception("Crash (expected)!")
110 })
```

scala/akka/actor/FSMActorSpec.scala, line 270 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:270
Taint Flags:

```
267 goto(2)
268 }
269 when(2) {
```

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/FSMActorSpec.scala, line 270 (Dead Code: Expression is Always false) — Low

```
270 case Event("stop", _) =>
271 cancelTimer("t")
272 stop()
273 }
```

scala/akka/actor/ActorRefSpec.scala, line 104 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:104
Taint Flags:

```
101 val fail = new InnerActor
102
103 def receive = {
104 case "innerself" => sender() ! self
105 case other => sender() ! other
106 }
107 }
```

scala/akka/actor/ActorLifeCycleSpec.scala, line 27 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorLifeCycleSpec.scala:27
Taint Flags:

```
24 val currentGen = generationProvider.getAndIncrement()
25 override def preStart(): Unit = { report("preStart") }
26 override def postStop(): Unit = { report("postStop") }
27 def receive = { case "status" => sender() ! message("OK") }
28 }
29
30 }
```

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/HotSwapSpec.scala, line 108 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/HotSwapSpec.scala:108
Taint Flags:

```
105 case "swap" =>
106 context.become({
107 case "state" => sender() ! "1"
108 case "swapped" => sender() ! "swapped"
109 case "crash" => throw new Exception("Crash (expected)!")
110 })
111 sender() ! "swapped"
```

scala/akka/actor/HotSwapSpec.scala, line 80 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/HotSwapSpec.scala:80
Taint Flags:

```
77 val a = system.actorOf(Props(new Actor {
78 def receive = {
79 case "init" => sender() ! "init"
80 case "swap" =>
81 context.become({
82 case "swapped" => sender() ! "swapped"
83 case "revert" => context.unbecome()
```

scala/akka/actor/FSMTransitionSpec.scala, line 35 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/FSMTransitionSpec.scala, line 35 (Dead Code: Expression is Always false) — Low

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMTransitionSpec.scala:35
Taint Flags:

```
32 case Event("tick", _) => goto(1)
33 }
34 when(1) {
35 case Event("tick", _) => goto(0)
36 }
37 whenUnhandled {
38 case Event("reply", _) => stay().replying("reply")
```

scala/akka/actor/SupervisorMiscSpec.scala, line 147 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorMiscSpec.scala:147
Taint Flags:

```
144 }
145 }
146
147 def receive = { case "engage" => context.stop(context.actorOf(Props.empty, "Robert")) }
148 }))
149 parent ! "engage"
150 expectMsg("green")
```

scala/akka/actor/HotSwapSpec.scala, line 79 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/HotSwapSpec.scala:79
Taint Flags:

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/HotSwapSpec.scala, line 79 (Dead Code: Expression is Always false) — Low

```
76 "be able to revert hotswap its behavior with unbecome" in {
77 val a = system.actorOf(Props(new Actor {
78 def receive = {
79 case "init" => sender() ! "init"
80 case "swap" =>
81 context.become({
82 case "swapped" => sender() ! "swapped"
```

scala/akka/actor/ActorRefSpec.scala, line 95 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:95
Taint Flags:

```
92
93 class InnerActor extends Actor {
94 def receive = {
95 case "innerself" => sender() ! self
96 case other => sender() ! other
97 }
98 }
```

scala/akka/actor/DeathWatchSpec.scala, line 40 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/DeathWatchSpec.scala:40
Taint Flags:

```
37
38 class NKOTBWatcher(testActor: ActorRef) extends Actor {
39 def receive = {
40 case "NKOTB" =>
41 val currentKid = context.watch(context.actorOf(Props(new Actor {
42 def receive = { case "NKOTB" => context.stop(self) }
```

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/DeathWatchSpec.scala, line 40 (Dead Code: Expression is Always false) — Low

```
43  }), "kid"))
```

scala/akka/actor/ConsistencySpec.scala, line 51 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ConsistencySpec.scala:51
Taint Flags:

```
48  }
49
50  lastStep = step
51  case "done" => sender() ! "done"; context.stop(self)
52  }
53  }
54 }
```

scala/akka/actor/FSMActorSpec.scala, line 56 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:56
Taint Flags:

```
53  }
54  }
55  }
56  case Event("hello", _) => stay().replying("world")
57  case Event("bye", _) => stop(FSM.Shutdown)
58  }
59
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 566 (Dead Code: Expression is Always false) — Low

Issue Details

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 566 (Dead Code: Expression is Always false) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:566
Taint Flags:

```
563 }
564
565 when(Finishing) {
566 case this.Event("pong", _) =>
567 pingChildren -= sender()
568 idleChildren :+= sender()
569 if (pingChildren.isEmpty) goto(LastPing) else stay()
```

scala/akka/actor/FSMActorSpec.scala, line 330 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:330
Taint Flags:

```
327 val fsmref = system.actorOf(Props(new Actor with FSM[Int, Int] {
328 startWith(0, 0)
329 when(0)(transform {
330 case Event("go", _) => stay()
331 }.using {
332 case _ => goto(1)
333 })
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 57 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 57 (Dead Code: Expression is Always false) — Low

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:57
Taint Flags:

```
54  override def supervisorStrategy = OneForOneStrategy() { case _ => SupervisorStrategy.Resume }
55  def receive = {
56  case "spawn" => sender() ! context.actorOf(Props[Resumer]())
57  case "fail" => throw new Exception("expected")
58  case "ping" => sender() ! "pong"
59  }
60  }
```

scala/akka/actor/ActorRefSpec.scala, line 451 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:451
Taint Flags:

```
448  override def postRestart(reason: Throwable) = latch.countDown()
449  }))
450
451  def receive = { case "sendKill" => ref ! Kill }
452  }))
453
454  boss ! "sendKill"
```

scala/akka/actor/FSMTransitionSpec.scala, line 38 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMTransitionSpec.scala:38
Taint Flags:

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/FSMTransitionSpec.scala, line 38 (Dead Code: Expression is Always false) — Low

```
35 case Event("tick", _) => goto(0)
36 }
37 whenUnhandled {
38 case Event("reply", _) => stay().replying("reply")
39 }
40 initialize()
41 override def preRestart(reason: Throwable, msg: Option[Any]): Unit = { target ! "restarted" }
```

scala/akka/actor/DeathWatchSpec.scala, line 167 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/DeathWatchSpec.scala:167
Taint Flags:

```
164 context.watch(terminal)
165 context.unwatch(terminal)
166 def receive = {
167 case "ping" => sender() ! "pong"
168 case t: Terminated => testActor ! WrappedTerminated(t)
169 }
170 }).withDeploy(Deploy.local))
```

scala/akka/actor/FSMActorSpec.scala, line 165 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:165
Taint Flags:

```
162 val fsm = TestActorRef(new Actor with FSM[Int, Null] {
163 startWith(1, null)
164 when(1) {
165 case Event("go", _) => goto(2)
166 }
```

Dead Code: Expression is Always false

Low

Package: akka.actor

scala/akka/actor/FSMActorSpec.scala, line 165 (Dead Code: Expression is Always false)

Low

```
167  })
168  val name = fsm.path.toString
```

scala/akka/actor/FSMActorSpec.scala, line 57 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:57
Taint Flags:

```
54  }
55  }
56  case Event("hello", _) => stay().replying("world")
57  case Event("bye", _) => stop(FSM.Shutdown)
58  }
59
60  when(Open) {
```

scala/akka/actor/FSMActorSpec.scala, line 150 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:150
Taint Flags:

```
147  val tester = system.actorOf(Props(new Actor {
148  def receive = {
149  case Hello => lock ! "hello"
150  case "world" => answerLatch.open()
151  case Bye => lock ! "bye"
152  }
153  }))
```

scala/akka/actor/ActorWithStashSpec.scala, line 31 (Dead Code: Expression is Always false)

Low

Issue Details

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/ActorWithStashSpec.scala, line 31 (Dead Code: Expression is Always false) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorWithStashSpec.scala:31
Taint Flags:

```
28 }
29
30 def receive = {
31 case "hello" =>
32 state.s = "hello"
33 unstashAll()
34 context.become(greeted)
```

scala/akka/actor/ActorRefSpec.scala, line 61 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:61
Taint Flags:

```
58 class SenderActor(replyActor: ActorRef, latch: TestLatch) extends Actor {
59
60 def receive = {
61 case "complex" => replyActor ! "complexRequest"
62 case "complex2" => replyActor ! "complexRequest2"
63 case "simple" => replyActor ! "simpleRequest"
64 case "complexReply" => {
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 704 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 704 (Dead Code: Expression is Always false) — Low

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:704
Taint Flags:

```
701
702 def printErrors(): Unit = {
703 errors.collect {
704 case (origin, ErrorLog("dump", _)) => getErrors(origin, 1)
705 case (origin, ErrorLog(msg, _)) if msg.startsWith("not resumed") => getErrorsUp(origin)
706 }
707 val merged = errors.sortBy(_._1.toString).flatMap {
```

scala/akka/actor/ReceiveTimeoutSpec.scala, line 46 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ReceiveTimeoutSpec.scala:46
Taint Flags:

```
43 }
44
45 def receive = {
46 case "crash" =>
47 restarting.set(true)
48 probe ! "crashing"
49 throw TestException("boom bang")
```

scala/akka/actor/SupervisorSpec.scala, line 467 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorSpec.scala:467
Taint Flags:

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/SupervisorSpec.scala, line 467 (Dead Code: Expression is Always false) — Low

```
464 case l: TestLatch => child ! l
465 case "test" => sender() ! "green"
466 case "testchild" => child.forward("test")
467 case "testchildAndAck" => child.forward("test"); sender() ! "ack"
468 }
469 }))
470
```

scala/akka/actor/SupervisorSpec.scala, line 466 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorSpec.scala:466
Taint Flags:

```
463 case Terminated(t) if t.path == child.path => testActor ! "child terminated"
464 case l: TestLatch => child ! l
465 case "test" => sender() ! "green"
466 case "testchild" => child.forward("test")
467 case "testchildAndAck" => child.forward("test"); sender() ! "ack"
468 }
469 }))
```

scala/akka/actor/ActorLifeCycleSpec.scala, line 162 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorLifeCycleSpec.scala:162
Taint Flags:

```
159 import akka.pattern._
160
161 override def receive: Receive = {
162 case "ping" =>
163 val replyTo = sender()
```

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/ActorLifeCycleSpec.scala, line 162 (Dead Code: Expression is Always false) — Low

```
164
165 context.stop(self)
```

scala/akka/actor/FSMTransitionSpec.scala, line 48 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMTransitionSpec.scala:48
Taint Flags:

```
45 startWith(0, 0)
46 when(0) {
47 case Event("tick", _) => goto(1).using(1)
48 case Event("stay", _) => stay()
49 }
50 when(1) {
51 case _ => goto(1)
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 297 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:297
Taint Flags:

```
294 setFlags(f.directive)
295 stateCache.put(self.path, stateCache.get(self.path).copy(failConstr = f.copy()))
296 throw f
297 case "ping" => { Thread.sleep((random.nextFloat() * 1.03).toLong); sender() ! "pong" }
298 case Dump(0) => abort("dump")
299 case Dump(level) => context.children.foreach(_ ! Dump(level - 1))
300 case Terminated(ref) =>
```

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/LocalActorRefProviderSpec.scala, line 45 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/LocalActorRefProviderSpec.scala:45
Taint Flags:

```
42 val a = system.actorOf(Props(new Actor {
43 val child = context.actorOf(Props.empty, name = childName)
44 def receive = {
45 case "lookup" =>
46 if (childName == child.path.name) {
47 val resolved = system.asInstanceOf[ExtendedActorSystem].provider.resolveActorRef(child.path)
48 sender() ! resolved
```

scala/akka/actor/HotSwapSpec.scala, line 65 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/HotSwapSpec.scala:65
Taint Flags:

```
62 val a = system.actorOf(Props(new Actor {
63 def receive = {
64 case "init" => sender() ! "init"
65 case "swap" => context.become({ case x: String => context.sender() ! x })
66 }
67 }))
68
```

scala/akka/actor/HotSwapSpec.scala, line 82 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/HotSwapSpec.scala, line 82 (Dead Code: Expression is Always false) — Low

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/HotSwapSpec.scala:82
Taint Flags:

```
79 case "init" => sender() ! "init"
80 case "swap" =>
81 context.become({
82 case "swapped" => sender() ! "swapped"
83 case "revert" => context.unbecome()
84 })
85 }
```

scala/akka/actor/FSMActorSpec.scala, line 173 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:173
Taint Flags:

```
170 system.eventStream.subscribe(testActor, classOf[Logging.Error])
171 fsm ! "go"
172 expectMsgPF(1 second, hint = "Next state 2 does not exist") {
173 case Logging.Error(_, `name`, _, "Next state 2 does not exist") => true
174 }
175 system.eventStream.unsubscribe(testActor)
176 }
```

scala/akka/actor/SupervisorMiscSpec.scala, line 120 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorMiscSpec.scala:120
Taint Flags:

```
117 "not be able to recreate child when old child is alive" in {
```

Dead Code: Expression is Always false

Low

Package: akka.actor

scala/akka/actor/SupervisorMiscSpec.scala, line 120 (Dead Code: Expression is Always false)

Low

```
118 val parent = system.actorOf(Props(new Actor {
119 def receive = {
120 case "engage" =>
121 try {
122 val kid = context.actorOf(Props.empty, "foo")
123 context.stop(kid)
```

scala/akka/actor/ActorWithStashSpec.scala, line 158 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorWithStashSpec.scala:158
Taint Flags:

```
155 stash()
156
157 // when restartLatch is open, must receive "hello"
158 case "hello" =>
159 hasMsgLatch.open()
160 }
161
```

scala/akka/actor/FSMActorSpec.scala, line 227 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:227
Taint Flags:

```
224 case Event("stop", _) => stop()
225 }
226 onTransition {
227 case "not-started" -> "started" =>
228 for (timerName <- timerNames) startSingleTimer(timerName, (), 10 seconds)
```

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/FSMActorSpec.scala, line 227 (Dead Code: Expression is Always false) — Low

```
229  }
230  onTermination {
```

scala/akka/actor/ActorRefSpec.scala, line 63 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:63
Taint Flags:

```
60  def receive = {
61  case "complex" => replyActor ! "complexRequest"
62  case "complex2" => replyActor ! "complexRequest2"
63  case "simple" => replyActor ! "simpleRequest"
64  case "complexReply" => {
65  latch.countDown()
66  }
```

scala/akka/actor/ActorSystemSpec.scala, line 346 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorSystemSpec.scala:346
Taint Flags:

```
343  .withFallback(AkkaSpec.testConf))
344  val a = system.actorOf(Props(new Actor {
345  def receive = {
346  case "die" => throw new Exception("hello")
347  }
348  }))
349  EventFilter[Exception]("hello").intercept {
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 545 (Dead Code: Expression is Always false) — Low

Issue Details

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 545 (Dead Code: Expression is Always false) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:545
Taint Flags:

```
542 case this.Event(Died(path), _) =>
543 bury(path)
544 stay()
545 case this.Event("pong", _) =>
546 pingChildren -= sender()
547 idleChildren :+= sender()
548 stay()
```

scala/akka/actor/SupervisorSpec.scala, line 450 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorSpec.scala:450
Taint Flags:

```
447 override def postRestart(reason: Throwable): Unit = testActor ! "child restarted"
448 def receive = {
449 case l: TestLatch => { Await.ready(l, 5 seconds); throw new IllegalStateException("OHNOES") }
450 case "test" => sender() ! "child green"
451 }
452 }), "child"))
453
```

scala/akka/actor/FSMTransitionSpec.scala, line 47 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/FSMTransitionSpec.scala, line 47 (Dead Code: Expression is Always false) — Low

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMTransitionSpec.scala:47
Taint Flags:

```
44 class OtherFSM(target: ActorRef) extends Actor with FSM[Int, Int] {
45 startWith(0, 0)
46 when(0) {
47 case Event("tick", _) => goto(1).using(1)
48 case Event("stay", _) => stay()
49 }
50 when(1) {
```

scala/akka/actor/FSMTransitionSpec.scala, line 32 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMTransitionSpec.scala:32
Taint Flags:

```
29 class MyFSM(target: ActorRef) extends Actor with FSM[Int, Unit] {
30 startWith(0, ())
31 when(0) {
32 case Event("tick", _) => goto(1)
33 }
34 when(1) {
35 case Event("tick", _) => goto(0)
```

scala/akka/actor/ActorRefSpec.scala, line 75 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:75
Taint Flags:

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/ActorRefSpec.scala, line 75 (Dead Code: Expression is Always false) — Low

```
72
73  class OuterActor(val inner: ActorRef) extends Actor {
74  def receive = {
75  case "self" => sender() ! self
76  case x => inner.forward(x)
77  }
78  }
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 583 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:583
Taint Flags:

```
580  }
581
582  when(LastPing) {
583  case this.Event("pong", _) =>
584  pingChildren -= sender()
585  idleChildren :+= sender()
586  if (pingChildren.isEmpty) goto(Stopping) else stay()
```

scala/akka/actor/HotSwapSpec.scala, line 109 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/HotSwapSpec.scala:109
Taint Flags:

```
106  context.become({
107  case "state" => sender() ! "1"
108  case "swapped" => sender() ! "swapped"
109  case "crash" => throw new Exception("Crash (expected)!")
110  })
```

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/HotSwapSpec.scala, line 109 (Dead Code: Expression is Always false) — Low

```
111 sender() ! "swapped"
112 }
```

scala/akka/actor/TimerSpec.scala, line 329 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/TimerSpec.scala:329
Taint Flags:

```
326 case StopStashing =>
327 context.become(notStashing)
328 unstashAll()
329 case "scheduled" =>
330 probe ! "saw-scheduled"
331 stash()
332 }
```

scala/akka/actor/HotSwapSpec.scala, line 64 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/HotSwapSpec.scala:64
Taint Flags:

```
61 "be able to hotswap its behavior with become(..)" in {
62 val a = system.actorOf(Props(new Actor {
63 def receive = {
64 case "init" => sender() ! "init"
65 case "swap" => context.become({ case x: String => context.sender() ! x })
66 }
67 }))
```

scala/akka/actor/ActorRefSpec.scala, line 33 (Dead Code: Expression is Always false) — Low

Issue Details

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/ActorRefSpec.scala, line 33 (Dead Code: Expression is Always false) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorRefSpec.scala:33
Taint Flags:

```
30 val worker = context.actorOf(Props[WorkerActor]())
31 worker ! "work"
32 }
33 case "complexRequest2" =>
34 val worker = context.actorOf(Props[WorkerActor]())
35 worker ! ReplyTo(sender())
36 case "workDone" => replyTo ! "complexReply"
```

scala/akka/actor/SupervisorSpec.scala, line 236 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorSpec.scala:236
Taint Flags:

```
233 override def preStart(): Unit = { preStarts += 1; testActor ! ("preStart" + preStarts) }
234 override def postStop(): Unit = { postStops += 1; testActor ! ("postStop" + postStops) }
235 def receive = {
236 case "crash" => { testActor ! "crashed"; throw new RuntimeException("Expected") }
237 case "ping" => sender() ! "pong"
238 }
239 }
```

scala/akka/actor/DeathWatchSpec.scala, line 42 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/DeathWatchSpec.scala, line 42 (Dead Code: Expression is Always false) — Low

File: scala/akka/actor/DeathWatchSpec.scala:42
Taint Flags:

```
39 def receive = {
40 case "NKOTB" =>
41 val currentKid = context.watch(context.actorOf(Props(new Actor {
42 def receive = { case "NKOTB" => context.stop(self) }
43 }), "kid"))
44 currentKid.forward("NKOTB")
45 context.become {
```

scala/akka/actor/FSMTransitionSpec.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: applyOrElse()
File: scala/akka/actor/FSMTransitionSpec.scala:157
Taint Flags:

```
154 case x -> y => nextStateData ! (x -> y)
155 }
156 when(1) {
157 case Event("test", _) =>
158 try {
159 sender() ! s"failed: $nextStateData"
160 } catch {
```

scala/akka/actor/FSMActorSpec.scala, line 221 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/FSMActorSpec.scala:221
Taint Flags:

```
218 lazy val fsmref = TestFSMRef(new Actor with FSM[String, Null] {
219 startWith("not-started", null)
220 when("not-started") {
```

Dead Code: Expression is Always false — Low

Package: akka.actor

scala/akka/actor/FSMActorSpec.scala, line 221 (Dead Code: Expression is Always false) — Low

```
221 case Event("start", _) => goto("started").replying("starting")
222 }
223 when("started", stateTimeout = 10 seconds) {
224 case Event("stop", _) => stop()
```

scala/akka/actor/ActorWithStashSpec.scala, line 56 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorWithStashSpec.scala:56
Taint Flags:

```
53 class ActorWithProtocol extends Actor with Stash {
54 import context.system
55 def receive = {
56 case "open" =>
57 unstashAll()
58 context.become {
59 case "write" => // do writing...
```

scala/akka/actor/ActorWithStashSpec.scala, line 24 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/ActorWithStashSpec.scala:24
Taint Flags:

```
21 class StashingActor extends Actor with Stash {
22 import context.system
23 def greeted: Receive = {
24 case "bye" =>
25 state.s = "bye"
26 state.finished.await()
27 case _ => // do nothing
```

Dead Code: Expression is Always false — Low

Package: akka.actor.dispatch

scala/akka/actor/dispatch/DispatcherActorSpec.scala, line 43 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/DispatcherActorSpec.scala:43
Taint Flags:

```
40 """"
41 class TestActor extends Actor {
42 def receive = {
43 case "Hello" => sender() ! "World"
44 case "Failure" => throw new RuntimeException("Expected exception; to test fault-tolerance")
45 }
46 }
```

scala/akka/actor/dispatch/PinnedActorSpec.scala, line 42 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/PinnedActorSpec.scala:42
Taint Flags:

```
39 "support tell" in {
40 val oneWay = new CountDownLatch(1)
41 val actor = system.actorOf(
42 Props(new Actor { def receive = { case "OneWay" => oneWay.countDown() } }).withDispatcher("pinned-dispatcher"))
43 actor ! "OneWay"
44 assert(oneWay.await(1, TimeUnit.SECONDS))
45 system.stop(actor)
```

scala/akka/actor/dispatch/DispatcherActorSpec.scala, line 87 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Dead Code: Expression is Always false — Low

Package: akka.actor.dispatch

scala/akka/actor/dispatch/DispatcherActorSpec.scala, line 87 (Dead Code: Expression is Always false) — Low

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/DispatcherActorSpec.scala:87
Taint Flags:

```
84
85  val slowOne = system.actorOf(Props(new Actor {
86  def receive = {
87  case "hogexecutor" => { sender() ! "OK"; start.await() }
88  case "ping" => if (works.get) latch.countDown()
89  }
90  }).withDispatcher(throughputDispatcher))
```

scala/akka/actor/dispatch/DispatcherActorSpec.scala, line 44 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/DispatcherActorSpec.scala:44
Taint Flags:

```
41  class TestActor extends Actor {
42  def receive = {
43  case "Hello" => sender() ! "World"
44  case "Failure" => throw new RuntimeException("Expected exception; to test fault-tolerance")
45  }
46  }
47
```

scala/akka/actor/dispatch/PinnedActorSpec.scala, line 29 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()

Dead Code: Expression is Always false — Low

Package: akka.actor.dispatch

scala/akka/actor/dispatch/PinnedActorSpec.scala, line 29 (Dead Code: Expression is Always false) — Low

File: scala/akka/actor/dispatch/PinnedActorSpec.scala:29
Taint Flags:

```
26  class TestActor extends Actor {
27  def receive = {
28  case "Hello" => sender() ! "World"
29  case "Failure" => throw new RuntimeException("Expected exception; to test fault-tolerance")
30  }
31  }
32 }
```

scala/akka/actor/dispatch/DispatcherActorSpec.scala, line 115 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/DispatcherActorSpec.scala:115
Taint Flags:

```
112
113  val fastOne = system.actorOf(Props(new Actor {
114  def receive = {
115  case "ping" => if (works.get) latch.countDown(); context.stop(self)
116  }
117  }).withDispatcher(throughputDispatcher))
118
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 385 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:385
Taint Flags:

```
382  val waitTime = (20 seconds).dilated.toMillis
```

Dead Code: Expression is Always false — Low

Package: akka.actor.dispatch

scala/akka/actor/dispatch/ActorModelSpec.scala, line 385 (Dead Code: Expression is Always false) — Low

```
383 val boss = system.actorOf(Props(new Actor {
384 def receive = {
385 case "run" => for (_ <- 1 to num) context.watch(context.actorOf(props)) ! cachedMessage
386 case Terminated(_) => stopLatch.countDown()
387 }
388 }).withDispatcher("boss"))
```

scala/akka/actor/dispatch/DispatcherActorSpec.scala, line 122 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/DispatcherActorSpec.scala:122
Taint Flags:

```
119 val slowOne = system.actorOf(Props(new Actor {
120 def receive = {
121 case "hogexecutor" => { ready.countDown(); start.await() }
122 case "ping" => { works.set(false); context.stop(self) }
123 }
124 }).withDispatcher(throughputDispatcher))
125
```

scala/akka/actor/dispatch/DispatcherActorSpec.scala, line 53 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/DispatcherActorSpec.scala:53
Taint Flags:

```
50 }
51 class OneWayTestActor extends Actor {
52 def receive = {
53 case "OneWay" => OneWayTestActor.oneWay.countDown()
```

Dead Code: Expression is Always false — Low

Package: akka.actor.dispatch

scala/akka/actor/dispatch/DispatcherActorSpec.scala, line 53 (Dead Code: Expression is Always false) — Low

```
54  }
55  }
56 }
```

scala/akka/actor/dispatch/DispatcherActorSpec.scala, line 88 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/DispatcherActorSpec.scala:88
Taint Flags:

```
85  val slowOne = system.actorOf(Props(new Actor {
86  def receive = {
87  case "hogexecutor" => { sender() ! "OK"; start.await() }
88  case "ping" => if (works.get) latch.countDown()
89  }
90  }).withDispatcher(throughputDispatcher))
91
```

scala/akka/actor/dispatch/DispatcherActorSpec.scala, line 83 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/DispatcherActorSpec.scala:83
Taint Flags:

```
80  val latch = new CountDownLatch(100)
81  val start = new CountDownLatch(1)
82  val fastOne = system.actorOf(
83  Props(new Actor { def receive = { case "sabotage" => works.set(false) } }).withDispatcher(throughputDispatcher))
84
85  val slowOne = system.actorOf(Props(new Actor {
86  def receive = {
```

Dead Code: Expression is Always false — Low

Package: akka.actor.dispatch

scala/akka/actor/dispatch/PinnedActorSpec.scala, line 28 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/PinnedActorSpec.scala:28
Taint Flags:

```
25
26  class TestActor extends Actor {
27  def receive = {
28  case "Hello" => sender() ! "World"
29  case "Failure" => throw new RuntimeException("Expected exception; to test fault-tolerance")
30  }
31  }
```

scala/akka/actor/dispatch/DispatcherActorSpec.scala, line 121 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/DispatcherActorSpec.scala:121
Taint Flags:

```
118
119  val slowOne = system.actorOf(Props(new Actor {
120  def receive = {
121  case "hogexecutor" => { ready.countDown(); start.await() }
122  case "ping" => { works.set(false); context.stop(self) }
123  }
124  }).withDispatcher(throughputDispatcher))
```

Package: akka.actor.routing

scala/akka/actor/routing/ListenerSpec.scala, line 33 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality

Dead Code: Expression is Always false — Low

Package: akka.actor.routing

scala/akka/actor/routing/ListenerSpec.scala, line 33 (Dead Code: Expression is Always false) — Low

Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/routing/ListenerSpec.scala:33
Taint Flags:

```
30 def newListener =
31 system.actorOf(Props(new Actor {
32 def receive = {
33 case "bar" =>
34 barCount.incrementAndGet
35 barLatch.countDown()
36 case "foo" =>
```

scala/akka/actor/routing/ListenerSpec.scala, line 36 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/routing/ListenerSpec.scala:36
Taint Flags:

```
33 case "bar" =>
34 barCount.incrementAndGet
35 barLatch.countDown()
36 case "foo" =>
37 fooLatch.countDown()
38 }
39 }))
```

scala/akka/actor/routing/ListenerSpec.scala, line 26 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Dead Code: Expression is Always false — Low

Package: akka.actor.routing

scala/akka/actor/routing/ListenerSpec.scala, line 26 (Dead Code: Expression is Always false) — Low

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/actor/routing/ListenerSpec.scala:26
Taint Flags:

```
23
24 val broadcast = system.actorOf(Props(new Actor with Listeners {
25 def receive = listenerManagement.orElse {
26 case "foo" => gossip("bar")
27 }
28 }))
29
```

Package: akka.dataflow

scala/akka/dataflow/Future2Actor.scala, line 47 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/dataflow/Future2Actor.scala:47
Taint Flags:

```
44 val actor = system.actorOf(Props(new Actor {
45 def receive = {
46 case "do" => Future(31).pipeTo(context.sender())
47 case "ex" => Future(throw new AssertionError).pipeTo(context.sender())
48 }
49 }))
50 Await.result(actor ? "do", timeout.duration) should ===(31)
```

scala/akka/dataflow/Future2Actor.scala, line 46 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/dataflow/Future2Actor.scala:46
Taint Flags:

Dead Code: Expression is Always false — Low

Package: akka.dataflow

scala/akka/dataflow/Future2Actor.scala, line 46 (Dead Code: Expression is Always false) — Low

```
43 "support reply via sender" in {
44 val actor = system.actorOf(Props(new Actor {
45 def receive = {
46 case "do" => Future(31).pipeTo(context.sender())
47 case "ex" => Future(throw new AssertionError).pipeTo(context.sender())
48 }
49 }))
```

Package: akka.dispatch

scala/akka/dispatch/ForkJoinPoolStarvationSpec.scala, line 39 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/dispatch/ForkJoinPoolStarvationSpec.scala:39
Taint Flags:

```
36 class InnocentActor extends Actor {
37
38 override def receive = {
39 case "ping" =>
40 sender() ! "All fine"
41 }
42 }
```

scala/akka/dispatch/ForkJoinPoolStarvationSpec.scala, line 31 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/dispatch/ForkJoinPoolStarvationSpec.scala:31
Taint Flags:

```
28 self ! "tick"
29
30 override def receive = {
```

Dead Code: Expression is Always false — Low

Package: akka.dispatch

scala/akka/dispatch/ForkJoinPoolStarvationSpec.scala, line 31 (Dead Code: Expression is Always false) — Low

```
31  case "tick" =>
32  self ! "tick"
33  }
34  }
```

Package: akka.event

scala/akka/event/LoggingReceiveSpec.scala, line 272 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:272
Taint Flags:

```
269  actor ! Kill
270  expectMsgAllPF(messages = 3) {
271  case Logging.Error(_: ActorKilledException, `aname`, _, "Kill") => 0
272  case Logging.Debug(`aname`, `aclass`, "restarting") => 1
273  case Logging.Debug(`aname`, `aclass`, "restarted") => 2
274  }
275  }
```

scala/akka/event/LoggerSpec.scala, line 262 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggerSpec.scala:262
Taint Flags:

```
259
260  ref ! "Current Message removed from MDC"
261  probe.expectMsgPF(max = 3.seconds) {
262  case w @ Warning(_, _, "Current Message removed from MDC") if w.mdc.size == 1 && w.mdc("requestId") == 4 =>
263  }
264
```

Dead Code: Expression is Always false — Low

Package: akka.event

scala/akka/event/LoggerSpec.scala, line 262 (Dead Code: Expression is Always false) — Low

```
265 } finally {
```

scala/akka/event/LoggerSpec.scala, line 253 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggerSpec.scala:253
Taint Flags:

```
250
251 ref ! "Current Message in MDC"
252 probe.expectMsgPF(max = 3.seconds) {
253 case w @ Warning(_, _, "Current Message in MDC")
254 if w.mdc.size == 3 &&
255 w.mdc("requestId") == 3 &&
256 w.mdc("currentMsg") == "Current Message in MDC" &&
```

scala/akka/event/LoggingReceiveSpec.scala, line 271 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:271
Taint Flags:

```
268 EventFilter[ActorKilledException](occurrences = 1).intercept {
269 actor ! Kill
270 expectMsgAllPF(messages = 3) {
271 case Logging.Error(_: ActorKilledException, `aname`, _, "Kill") => 0
272 case Logging.Debug(`aname`, `aclass`, "restarting") => 1
273 case Logging.Debug(`aname`, `aclass`, "restarted") => 2
274 }
```

scala/akka/event/LoggingReceiveSpec.scala, line 97 (Dead Code: Expression is Always false) — Low

Issue Details

Dead Code: Expression is Always false — Low

Package: akka.event

scala/akka/event/LoggingReceiveSpec.scala, line 97 (Dead Code: Expression is Always false) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:97
Taint Flags:

```
94  }
95
96  val actor = TestActorRef(new Actor {
97  def switch: Actor.Receive = { case "becomenull" => context.become(r, false) }
98  def receive =
99  switch.orElse(LoggingReceive {
100  case _ => sender() ! "x"
```

scala/akka/event/LoggingReceiveSpec.scala, line 273 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:273
Taint Flags:

```
270  expectMsgAllPF(messages = 3) {
271  case Logging.Error(_: ActorKilledException, `aname`, _, "Kill") => 0
272  case Logging.Debug(`aname`, `aclass`, "restarting") => 1
273  case Logging.Debug(`aname`, `aclass`, "restarted") => 2
274  }
275  }
276
```

scala/akka/event/LoggerSpec.scala, line 243 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Dead Code: Expression is Always false — Low

Package: akka.event

scala/akka/event/LoggerSpec.scala, line 243 (Dead Code: Expression is Always false) — Low

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggerSpec.scala:243
Taint Flags:

```
240
241 ref ! "Processing new Request"
242 probe.expectMsgPF(max = 3.seconds) {
243 case w @ Warning(_, _, "Processing new Request") if w.mdc.size == 1 && w.mdc("requestId") == 1 =>
244 }
245
246 ref ! "Processing another Request"
```

scala/akka/event/LoggerSpec.scala, line 130 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: mdc()
File: scala/akka/event/LoggerSpec.scala:130
Taint Flags:

```
127 val always = Map("requestId" -> reqId)
128 val cmim = "Current Message in MDC"
129 val perMessage = currentMessage match {
130 case `cmim` => Map[String, Any]("currentMsg" -> cmim, "currentMsgLength" -> cmim.length)
131 case _ => Map()
132 }
133 always ++ perMessage
```

scala/akka/event/LoggerSpec.scala, line 248 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggerSpec.scala:248
Taint Flags:

```
245
246 ref ! "Processing another Request"
```

Dead Code: Expression is Always false | Low

Package: akka.event

scala/akka/event/LoggerSpec.scala, line 248 (Dead Code: Expression is Always false) | Low

```
247 probe.expectMsgPF(max = 3.seconds) {
248 case w @ Warning(_, _, "Processing another Request") if w.mdc.size == 1 && w.mdc("requestId") == 2 =>
249 }
250
251 ref ! "Current Message in MDC"
```

scala/akka/event/LoggingReceiveSpec.scala, line 143 (Dead Code: Expression is Always false) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:143
Taint Flags:

```
140 val a = system.actorOf(Props(new Actor with DiagnosticActorLogging {
141 override def mdc(currentMessage: Any) = myMDC
142 def receive = LoggingReceive {
143 case "hello" =>
144 }
145 }))
146 a ! "hello"
```

scala/akka/event/LoggerSpec.scala, line 159 (Dead Code: Expression is Always false) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggerSpec.scala:159
Taint Flags:

```
156 // since logging is asynchronous ensure that it propagates
157 if (shouldLog) {
158 probe.fishForMessage() {
159 case "Danger! Danger!" => true
160 case _ => false
161 }
162 } else {
```

Dead Code: Expression is Always false Low

Package: akka.event

scala/akka/event/LoggingReceiveSpec.scala, line 114 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/event/LoggingReceiveSpec.scala:114
Taint Flags:

```
111
112 actor ! "bah"
113 expectMsgPF() {
114 case UnhandledMessage("bah", _, `actor`) => true
115 }
116 }
117 }
```

Package: akka.pattern

scala/akka/pattern/BackoffSupervisorSpec.scala, line 27 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/pattern/BackoffSupervisorSpec.scala:27
Taint Flags:

```
24
25 class Child(probe: ActorRef) extends Actor {
26 def receive: Receive = {
27 case "boom" => throw new TestException
28 case msg => probe ! msg
29 }
30 }
```

scala/akka/pattern/BackoffSupervisorSpec.scala, line 39 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality

Dead Code: Expression is Always false

Low

Package: akka.pattern

scala/akka/pattern/BackoffSupervisorSpec.scala, line 39 (Dead Code: Expression is Always false)

Low

Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/pattern/BackoffSupervisorSpec.scala:39
Taint Flags:

```
36
37 class ManualChild(probe: ActorRef) extends Actor {
38 def receive: Receive = {
39 case "boom" => throw new TestException
40 case msg =>
41 probe ! msg
42 context.parent ! BackoffSupervisor.Reset
```

scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala, line 36 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala:36
Taint Flags:

```
33
34 def receive: Receive = {
35 case "DIE" => context.stop(self)
36 case "THROW" => throw new TestActor.NormalException
37 case "THROW_STOPPING_EXCEPTION" => throw new TestActor.StoppingException
38 case ("TO_PARENT", msg) => context.parent ! msg
39 case other => probe ! other
```

scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala, line 37 (Dead Code: Expression is Always false)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Dead Code: Expression is Always false — Low

Package: akka.pattern

scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala, line 37 (Dead Code: Expression is Always false) — Low

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala:37
Taint Flags:

```
34 def receive: Receive = {
35 case "DIE" => context.stop(self)
36 case "THROW" => throw new TestActor.NormalException
37 case "THROW_STOPPING_EXCEPTION" => throw new TestActor.StoppingException
38 case ("TO_PARENT", msg) => context.parent ! msg
39 case other => probe ! other
40 }
```

scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala, line 35 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala:35
Taint Flags:

```
32 probe ! "STARTED"
33
34 def receive: Receive = {
35 case "DIE" => context.stop(self)
36 case "THROW" => throw new TestActor.NormalException
37 case "THROW_STOPPING_EXCEPTION" => throw new TestActor.StoppingException
38 case ("TO_PARENT", msg) => context.parent ! msg
```

scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala, line 143 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala:143
Taint Flags:

Dead Code: Expression is Always false — Low

Package: akka.pattern

scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala, line 143 (Dead Code: Expression is Always false) — Low

```
140
141 class SlowlyFailingActor(latch: CountDownLatch) extends Actor {
142 def receive: Receive = {
143 case "THROW" =>
144 sender() ! "THROWN"
145 throw new NormalException
146 case "PING" =>
```

scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala, line 146 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala:146
Taint Flags:

```
143 case "THROW" =>
144 sender() ! "THROWN"
145 throw new NormalException
146 case "PING" =>
147 sender() ! "PONG"
148 }
149
```

scala/akka/pattern/BackoffSupervisorSpec.scala, line 207 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/pattern/BackoffSupervisorSpec.scala:207
Taint Flags:

```
204 "use provided actor while stopped and withHandlerWhileStopped is specified" in {
205 val handler = system.actorOf(Props(new Actor {
206 override def receive: Receive = {
```

Dead Code: Expression is Always false — Low

Package: akka.pattern

scala/akka/pattern/BackoffSupervisorSpec.scala, line 207 (Dead Code: Expression is Always false) — Low

```
207 case "still there?" =>
208 sender() ! "not here!"
209 }
210 }))
```

scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala, line 38 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala:38
Taint Flags:

```
35 case "DIE" => context.stop(self)
36 case "THROW" => throw new TestActor.NormalException
37 case "THROW_STOPPING_EXCEPTION" => throw new TestActor.StoppingException
38 case ("TO_PARENT", msg) => context.parent ! msg
39 case other => probe ! other
40 }
41 }
```

Package: akka.routing

scala/akka/routing/BroadcastSpec.scala, line 39 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/BroadcastSpec.scala:39
Taint Flags:

```
36 val counter2 = new AtomicInteger
37 val actor2 = system.actorOf(Props(new Actor {
38 def receive = {
39 case "end" => doneLatch.countDown()
40 case msg: Int => counter2.addAndGet(msg)
41 }
```

Dead Code: Expression is Always false — Low

Package: akka.routing

scala/akka/routing/BroadcastSpec.scala, line 39 (Dead Code: Expression is Always false) — Low

```
42  }))
```

scala/akka/routing/RoundRobinSpec.scala, line 88 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/RoundRobinSpec.scala:88
Taint Flags:

```
85
86  val actor = system.actorOf(RoundRobinPool(5).props(routeeProps = Props(new Actor {
87  def receive = {
88  case "hello" => helloLatch.countDown()
89  }
90
91  override def postStop(): Unit = {
```

scala/akka/routing/ScatterGatherFirstCompletedSpec.scala, line 58 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/ScatterGatherFirstCompletedSpec.scala:58
Taint Flags:

```
55  val counter1 = new AtomicInteger
56  val actor1 = system.actorOf(Props(new Actor {
57  def receive = {
58  case "end" => doneLatch.countDown()
59  case msg: Int => counter1.addAndGet(msg)
60  }
61  }))
```

scala/akka/routing/RoundRobinSpec.scala, line 134 (Dead Code: Expression is Always false) — Low

Issue Details

Dead Code: Expression is Always false — Low

Package: akka.routing

scala/akka/routing/RoundRobinSpec.scala, line 134 (Dead Code: Expression is Always false) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/RoundRobinSpec.scala:134
Taint Flags:

```
131 val paths = (1 to connectionCount).map { n =>
132 val ref = system.actorOf(Props(new Actor {
133 def receive = {
134 case "hit" => sender() ! self.path.name
135 case "end" => doneLatch.countDown()
136 }
137 }), name = "target-" + n)
```

scala/akka/routing/TailChoppingSpec.scala, line 24 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/TailChoppingSpec.scala:24
Taint Flags:

```
21
22 def receive = {
23 case "stop" => context.stop(self)
24 case "times" => sender() ! times
25 case _ =>
26 times += 1
27 Thread.sleep(sleepTime.toMillis)
```

scala/akka/routing/ConfiguredLocalRoutingSpec.scala, line 87 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Dead Code: Expression is Always false — Low

Package: akka.routing

scala/akka/routing/ConfiguredLocalRoutingSpec.scala, line 87 (Dead Code: Expression is Always false) — Low

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/ConfiguredLocalRoutingSpec.scala:87
Taint Flags:

```
84
85 class EchoProps extends Actor {
86 def receive = {
87 case "get" => sender() ! context.props
88 }
89 }
90
```

scala/akka/routing/ScatterGatherFirstCompletedSpec.scala, line 66 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/ScatterGatherFirstCompletedSpec.scala:66
Taint Flags:

```
63 val counter2 = new AtomicInteger
64 val actor2 = system.actorOf(Props(new Actor {
65 def receive = {
66 case "end" => doneLatch.countDown()
67 case msg: Int => counter2.addAndGet(msg)
68 }
69 }))
```

scala/akka/routing/RandomSpec.scala, line 27 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/RandomSpec.scala:27

Dead Code: Expression is Always false — Low

Package: akka.routing

scala/akka/routing/RandomSpec.scala, line 27 (Dead Code: Expression is Always false) — Low

Taint Flags:

```
24
25  val actor = system.actorOf(RandomPool(7).props(Props(new Actor {
26  def receive = {
27  case "hello" => sender() ! "world"
28  }
29
30  override def postStop(): Unit = {
```

scala/akka/routing/BroadcastSpec.scala, line 71 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/BroadcastSpec.scala:71
Taint Flags:

```
68  val counter2 = new AtomicInteger
69  val actor2 = system.actorOf(Props(new Actor {
70  def receive = {
71  case "end" => doneLatch.countDown()
72  case msg: Int => counter2.addAndGet(msg)
73  }
74  }))
```

scala/akka/routing/RoundRobinSpec.scala, line 135 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/RoundRobinSpec.scala:135
Taint Flags:

```
132  val ref = system.actorOf(Props(new Actor {
133  def receive = {
134  case "hit" => sender() ! self.path.name
135  case "end" => doneLatch.countDown()
```

Dead Code: Expression is Always false — Low

Package: akka.routing

scala/akka/routing/RoundRobinSpec.scala, line 135 (Dead Code: Expression is Always false) — Low

```
136 }
137 }), name = "target-" + n)
138 ref.path.toStringWithoutAddress
```

scala/akka/routing/RoutingSpec.scala, line 200 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/RoutingSpec.scala:200
Taint Flags:

```
197 "start in-line for context.actorOf()" in {
198 system.actorOf(Props(new Actor {
199 def receive = {
200 case "start" =>
201 (context.actorOf(RoundRobinPool(2).props(routeeProps = Props(new Actor {
202 def receive = { case x => sender() ! x }
203 }))) ? "hello").pipeTo(sender())
```

scala/akka/routing/RoundRobinSpec.scala, line 34 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/RoundRobinSpec.scala:34
Taint Flags:

```
31
32 val actor = system.actorOf(RoundRobinPool(5).props(routeeProps = Props(new Actor {
33 def receive = {
34 case "hello" => helloLatch.countDown()
35 }
36
37 override def postStop(): Unit = {
```

Dead Code: Expression is Always false — Low

Package: akka.routing

scala/akka/routing/RoundRobinSpec.scala, line 64 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/RoundRobinSpec.scala:64
Taint Flags:

```
61  val actor = system.actorOf(RoundRobinPool(connectionCount).props(routeeProps = Props(new Actor {
62  lazy val id = counter.getAndIncrement()
63  def receive = {
64  case "hit" => sender() ! id
65  case "end" => doneLatch.countDown()
66  }
67  })), "round-robin")
```

scala/akka/routing/RoundRobinSpec.scala, line 183 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/RoundRobinSpec.scala:183
Taint Flags:

```
180  val childProps = Props(new Actor {
181  def receive = {
182  case "hit" => sender() ! self.path.name
183  case "end" => context.stop(self)
184  }
185  })
186
```

scala/akka/routing/RandomSpec.scala, line 64 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Dead Code: Expression is Always false Low

Package: akka.routing

scala/akka/routing/RandomSpec.scala, line 64 (Dead Code: Expression is Always false) Low

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/RandomSpec.scala:64
Taint Flags:

```
61 lazy val id = counter.getAndIncrement()
62 def receive = {
63 case "hit" => sender() ! id
64 case "end" => doneLatch.countDown()
65 }
66 })), name = "random")
67
```

scala/akka/routing/RandomSpec.scala, line 90 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/RandomSpec.scala:90
Taint Flags:

```
87
88 val actor = system.actorOf(RandomPool(6).props(routeeProps = Props(new Actor {
89 def receive = {
90 case "hello" => helloLatch.countDown()
91 }
92
93 override def postStop(): Unit = {
```

scala/akka/routing/RoundRobinSpec.scala, line 65 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/RoundRobinSpec.scala:65
Taint Flags:

```
62 lazy val id = counter.getAndIncrement()
63 def receive = {
```

Dead Code: Expression is Always false Low

Package: akka.routing

scala/akka/routing/RoundRobinSpec.scala, line 65 (Dead Code: Expression is Always false) Low

```
64 case "hit" => sender() ! id
65 case "end" => doneLatch.countDown()
66 }
67 })), "round-robin")
68
```

scala/akka/routing/BroadcastSpec.scala, line 31 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/BroadcastSpec.scala:31
Taint Flags:

```
28 val counter1 = new AtomicInteger
29 val actor1 = system.actorOf(Props(new Actor {
30 def receive = {
31 case "end" => doneLatch.countDown()
32 case msg: Int => counter1.addAndGet(msg)
33 }
34 }))
```

scala/akka/routing/RandomSpec.scala, line 63 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/RandomSpec.scala:63
Taint Flags:

```
60 val actor = system.actorOf(RandomPool(connectionCount).props(routeeProps = Props(new Actor {
61 lazy val id = counter.getAndIncrement()
62 def receive = {
63 case "hit" => sender() ! id
64 case "end" => doneLatch.countDown()
65 }
66 })), name = "random")
```

Dead Code: Expression is Always false — Low

Package: akka.routing

scala/akka/routing/ResizerSpec.scala, line 176 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/ResizerSpec.scala:176
Taint Flags:

```
173 def receive = {
174 case d: FiniteDuration =>
175 Thread.sleep(d.dilated.toMillis); sender() ! "done"
176 case "echo" => sender() ! "reply"
177 }
178 })))
179
```

scala/akka/routing/RouteeCreationSpec.scala, line 42 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/RouteeCreationSpec.scala:42
Taint Flags:

```
39 }
40 })))
41 val gotit = receiveWhile(messages = N) {
42 case "two" => lastSender.toString
43 }
44 expectNoMessage(100.millis)
45 if (gotit.size != N) {
```

scala/akka/routing/RouteeCreationSpec.scala, line 38 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Dead Code: Expression is Always false — Low

Package: akka.routing

scala/akka/routing/RouteeCreationSpec.scala, line 38 (Dead Code: Expression is Always false) — Low

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/RouteeCreationSpec.scala:38
Taint Flags:

```
35 system.actorOf(RoundRobinPool(N).props(Props(new Actor {
36 context.parent ! "one"
37 def receive = {
38 case "one" => testActor.forward("two")
39 }
40 })))
41 val gotit = receiveWhile(messages = N) {
```

scala/akka/routing/TailChoppingSpec.scala, line 63 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/TailChoppingSpec.scala:63
Taint Flags:

```
60 val counter2 = new AtomicInteger
61 val actor2 = system.actorOf(Props(new Actor {
62 def receive = {
63 case "end" => doneLatch.countDown()
64 case msg: Int => counter2.addAndGet(msg)
65 }
66 }))
```

scala/akka/routing/TailChoppingSpec.scala, line 23 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()

Dead Code: Expression is Always false — Low

Package: akka.routing

scala/akka/routing/TailChoppingSpec.scala, line 23 (Dead Code: Expression is Always false) — Low

File: scala/akka/routing/TailChoppingSpec.scala:23
Taint Flags:

```
20 var times: Int = _
21
22 def receive = {
23 case "stop" => context.stop(self)
24 case "times" => sender() ! times
25 case _ =>
26 times += 1
```

scala/akka/routing/TailChoppingSpec.scala, line 55 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/TailChoppingSpec.scala:55
Taint Flags:

```
52 val counter1 = new AtomicInteger
53 val actor1 = system.actorOf(Props(new Actor {
54 def receive = {
55 case "end" => doneLatch.countDown()
56 case msg: Int => counter1.addAndGet(msg)
57 }
58 }))
```

scala/akka/routing/BroadcastSpec.scala, line 61 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/BroadcastSpec.scala:61
Taint Flags:

```
58 val counter1 = new AtomicInteger
59 val actor1 = system.actorOf(Props(new Actor {
```

Dead Code: Expression is Always false — Low

Package: akka.routing

scala/akka/routing/BroadcastSpec.scala, line 61 (Dead Code: Expression is Always false) — Low

```
60 def receive = {
61 case "end" => doneLatch.countDown()
62 case msg: Int =>
63 counter1.addAndGet(msg)
64 sender() ! "ack"
```

scala/akka/routing/RoundRobinSpec.scala, line 182 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: applyOrElse()
File: scala/akka/routing/RoundRobinSpec.scala:182
Taint Flags:

```
179
180 val childProps = Props(new Actor {
181 def receive = {
182 case "hit" => sender() ! self.path.name
183 case "end" => context.stop(self)
184 }
185 })
```

Package: akka.util

scala/akka/util/ByteStringSpec.scala, line 185 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: likeVecIts()
File: scala/akka/util/ByteStringSpec.scala:185
Taint Flags:

```
182 val (bsAIt, bsBIt) = (a.iterator, b.iterator)
183 val (vecAIt, vecBIt) = (Vector(a: _*).iterator.buffered, Vector(b: _*).iterator.buffered)
184 (body(bsAIt, bsBIt) == body(vecAIt, vecBIt)) &&
185 (!strict || (bsAIt.toSeq -> bsBIt.toSeq) == (vecAIt.toSeq -> vecBIt.toSeq))
186 }
187
```

Dead Code: Expression is Always false — Low

Package: akka.util

scala/akka/util/ByteStringSpec.scala, line 185 (Dead Code: Expression is Always false) — Low

```
188 def likeVecBld(body: Builder[Byte, _] => Unit): Boolean = {
```

Package: scala.akka.actor

scala/akka/actor/TypedActorSpec.scala, line 214 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: apply()
File: scala/akka/actor/TypedActorSpec.scala:214
Taint Flags:

```
211
212 override def onReceive(msg: Any, sender: ActorRef): Unit = {
213 ensureContextAvailable(msg match {
214 case "pigdog" => sender ! "dogpig"
215 case _ =>
216 })
217 }
```

Package: scala.akka.io.dns

scala/akka/io/dns/DockerBindDnsService.scala, line 68 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: apply()
File: scala/akka/io/dns/DockerBindDnsService.scala:68
Taint Flags:

```
65 .asScala
66 .find(_.names().asScala.exists(_.contains(containerName)))
67 .foreach(c => {
68 if ("running" == c.state()) {
69 client.killContainer(c.id)
70 }
71 client.removeContainer(c.id)
```

Dead Code: Expression is Always false Low

Package: scala.akka.io.dns.internal

scala/akka/io/dns/internal/AsyncDnsManagerSpec.scala, line 41 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: apply()
File: scala/akka/io/dns/internal/AsyncDnsManagerSpec.scala:41
Taint Flags:

```
38 "support ipv6" in {
39 dns ! Resolve("::1") // ::1 will short circuit the resolution
40 expectMsgType[Resolved] match {
41 case Resolved("::1", Seq(AAAARecord("::1", Ttl.effectivelyForever, _)), Nil) =>
42 case other => fail(other.toString)
43 }
44 }
```

scala/akka/io/dns/internal/AsyncDnsManagerSpec.scala, line 41 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: apply()
File: scala/akka/io/dns/internal/AsyncDnsManagerSpec.scala:41
Taint Flags:

```
38 "support ipv6" in {
39 dns ! Resolve("::1") // ::1 will short circuit the resolution
40 expectMsgType[Resolved] match {
41 case Resolved("::1", Seq(AAAARecord("::1", Ttl.effectivelyForever, _)), Nil) =>
42 case other => fail(other.toString)
43 }
44 }
```

Package: scala.akka.pattern

scala/akka/pattern/AskSpec.scala, line 230 (Dead Code: Expression is Always false) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Dead Code: Expression is Always false — Low

Package: scala.akka.pattern

scala/akka/pattern/AskSpec.scala, line 230 (Dead Code: Expression is Always false) — Low

Sink Details

Sink: IfStatement
Enclosing Method: apply()
File: scala/akka/pattern/AskSpec.scala:230
Taint Flags:

```
227  }))
228
229  val f = (act ? "ask").mapTo[String]
230  val (promiseActorRef, "ask") = p.expectMsgType[(ActorRef, String)]
231
232  watch(promiseActorRef)
233  promiseActorRef ! "complete"
```

scala/akka/pattern/AskSpec.scala, line 251 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: apply()
File: scala/akka/pattern/AskSpec.scala:251
Taint Flags:

```
248  }), "myName")
249
250  (act ? "ask").mapTo[String]
251  val (promiseActorRef, "ask") = p.expectMsgType[(ActorRef, String)]
252
253  promiseActorRef.path.name should startWith("myName")
254
```

scala/akka/pattern/AskSpec.scala, line 256 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: apply()
File: scala/akka/pattern/AskSpec.scala:256
Taint Flags:

Dead Code: Expression is Always false — Low

Package: scala.akka.pattern

scala/akka/pattern/AskSpec.scala, line 256 (Dead Code: Expression is Always false) — Low

```
253 promiseActorRef.path.name should startWith("myName")
254
255 (system.actorSelection("/user/myName") ? "ask").mapTo[String]
256 val (promiseActorRefForSelection, "ask") = p.expectMsgType[(ActorRef, String)]
257 promiseActorRefForSelection.path.name should startWith("_user_myName")
258 }
259 }
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 162 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: apply()
File: scala/akka/pattern/CircuitBreakerSpec.scala:162
Taint Flags:

```
159 val breaker = shortResetTimeoutCb()
160 intercept[TestException] { breaker().withSyncCircuitBreaker(throwException) }
161 checkLatch(breaker.halfOpenLatch)
162 assert("hi" == breaker().withSyncCircuitBreaker(sayHi))
163 checkLatch(breaker.closedLatch)
164 }
165
```

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();
  }
}
```

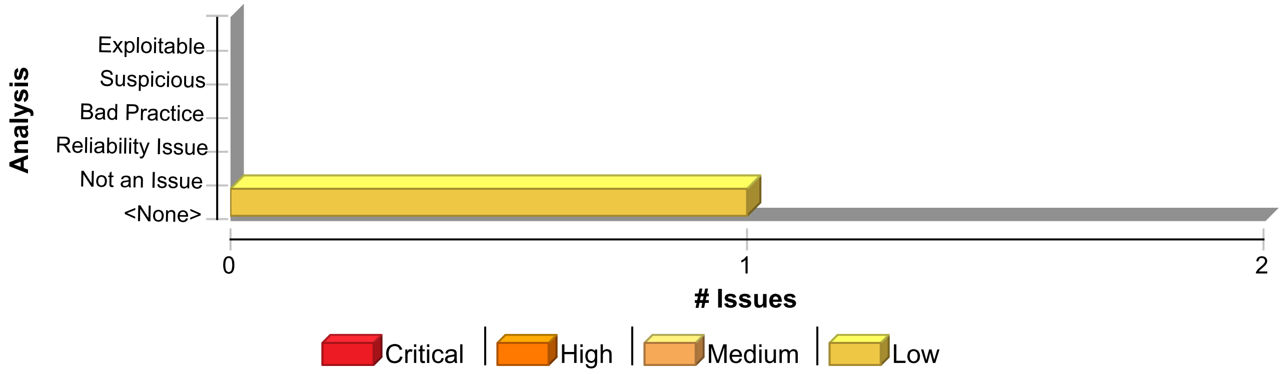
Example 2: The following method tries to check the variables `firstCall` and `secondCall`. (The variable `firstCall` is mistakenly set to `true` instead of being checked.) The result is that the first part of the expression `firstCall = true && secondCall == true` will always evaluate to `true`.

```
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: scala.akka.actor

scala/akka/actor/ActorSystemSpec.scala, line 294 (Dead Code: Expression is Always true) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: apply()
File: scala/akka/actor/ActorSystemSpec.scala:294
Taint Flags:

```
291 case _: IllegalStateException => failing = true
292 }
293
294 if (!failing && system.uptime >= 10) {
295 println(created.last)
296 println(system.asInstanceOf[ExtendedActorSystem].printTree)
297 fail("System didn't terminate within 5 seconds")
```

Insecure Randomness (26 issues)

Abstract

Standard pseudorandom number generators cannot withstand cryptographic attacks.

Explanation

Insecure randomness errors occur when a function that can produce predictable values is used as a source of randomness in a security-sensitive context. Computers are deterministic machines, and as such are unable to produce true randomness. Pseudorandom Number Generators (PRNGs) approximate randomness algorithmically, starting with a seed from which subsequent values are calculated. There are two types of PRNGs: statistical and cryptographic. Statistical PRNGs provide useful statistical properties, but their output is highly predictable and form an easy to reproduce numeric stream that is unsuitable for use in cases where security depends on generated values being unpredictable. Cryptographic PRNGs address this problem by generating output that is more difficult to predict. For a value to be cryptographically secure, it must be impossible or highly improbable for an attacker to distinguish between the generated random value and a truly random value. In general, if a PRNG algorithm is not advertised as being cryptographically secure, then it is probably a statistical PRNG and should not be used in security-sensitive contexts, where its use can lead to serious vulnerabilities such as easy-to-guess temporary passwords, predictable cryptographic keys, session hijacking, and DNS spoofing. **Example:** The following code uses a statistical PRNG to create a URL for a receipt that remains active for some period of time after a purchase.

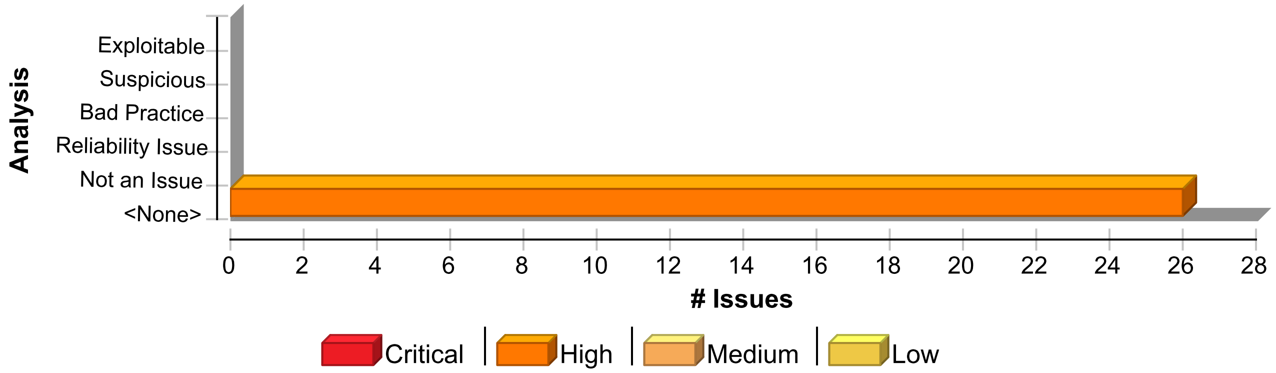
```
String GenerateReceiptURL(String baseUrl) {
    Random ranGen = new Random();
    ranGen.setSeed((new Date()).getTime());
    return (baseUrl + ranGen.nextInt(400000000) + ".html");
}
```

This code uses the `Random.nextInt()` function to generate "unique" identifiers for the receipt pages it generates. Since `Random.nextInt()` is a statistical PRNG, it is easy for an attacker to guess the strings it generates. Although the underlying design of the receipt system is also faulty, it would be more secure if it used a random number generator that did not produce predictable receipt identifiers, such as a cryptographic PRNG.

Recommendation

When unpredictability is critical, as is the case with most security-sensitive uses of randomness, use a cryptographic PRNG. Regardless of the PRNG you choose, always use a value with sufficient entropy to seed the algorithm. (Do not use values such as the current time because it offers only negligible entropy.) The Java language provides a cryptographic PRNG in `java.security.SecureRandom`. As is the case with other algorithm-based classes in `java.security`, `SecureRandom` provides an implementation-independent wrapper around a particular set of algorithms. When you request an instance of a `SecureRandom` object using `SecureRandom.getInstance()`, you can request a specific implementation of the algorithm. If the algorithm is available, then it is given as a `SecureRandom` object. If it is unavailable or if you do not specify a particular implementation, then you are given a `SecureRandom` implementation selected by the system. Sun provides a single `SecureRandom` implementation with the Java distribution named `SHA1PRNG`, which Sun describes as computing: "The SHA-1 hash over a true-random seed value concatenated with a 64-bit counter which is incremented by 1 for each operation. From the 160-bit SHA-1 output, only 64 bits are used [1]." However, the specifics of the Sun implementation of the `SHA1PRNG` algorithm are poorly documented, and it is unclear what sources of entropy the implementation uses and therefore what amount of true randomness exists in its output. Although there is speculation on the Web about the Sun implementation, there is no evidence to contradict the claim that the algorithm is cryptographically strong and can be used safely in security-sensitive contexts.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Insecure Randomness	26	0	0	26
Total	**26**	**0**	**0**	**26**

Insecure Randomness — High

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 169 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: preStart()
File: scala/akka/actor/SupervisorHierarchySpec.scala:169
Taint Flags:

```
166 val s = size - 1 // subtract myself
167 val kidInfo: Map[ActorPath, Int] =
168 if (s > 0) {
169 val kids = random.nextInt(Math.min(breadth, s)) + 1
170 val sizes = s / kids
171 var rest = s % kids
172 val propsTemplate = Props.empty.withDispatcher("hierarchy")
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 500 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextFloat()
Enclosing Method: akka$actor$SupervisorHierarchySpec$StressTest$$random012()
File: scala/akka/actor/SupervisorHierarchySpec.scala:500
Taint Flags:

Insecure Randomness High

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 500 (Insecure Randomness) High

```
497
498 val workSchedule = 50.millis
499
500 private def random012: Int = random.nextFloat() match {
501 case x if x > 0.1 => 0
502 case x if x > 0.03 => 1
503 case _ => 2
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 297 (Insecure Randomness) High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextFloat()
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:297
Taint Flags:

```
294 setFlags(f.directive)
295 stateCache.put(self.path, stateCache.get(self.path).copy(failConstr = f.copy()))
296 throw f
297 case "ping" => { Thread.sleep((random.nextFloat() * 1.03).toLong); sender() ! "pong" }
298 case Dump(0) => abort("dump")
299 case Dump(level) => context.children.foreach(_ ! Dump(level - 1))
300 case Terminated(ref) =>
```

scala/akka/actor/SchedulerSpec.scala, line 485 (Insecure Randomness) High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextBoolean()
Enclosing Method: delay()
File: scala/akka/actor/SchedulerSpec.scala:485
Taint Flags:

```
482 }
483 rounds
484 }
485 def delay = if (ThreadLocalRandom.current.nextBoolean) step * 2 else step
486 val N = 1000000
487 (1 to N).foreach(_ =>
```

Insecure Randomness — High

Package: akka.actor

scala/akka/actor/SchedulerSpec.scala, line 485 (Insecure Randomness) — **High**

```
488 sched.scheduleOnce(delay, new Scheduler.TaskRunOnClose {
```

Package: akka.pattern

scala/akka/pattern/CircuitBreakerStressSpec.scala, line 36 (Insecure Randomness) — **High**

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: akka$pattern$CircuitBreakerStressSpec$StressActor$$job()
File: scala/akka/pattern/CircuitBreakerStressSpec.scala:36
Taint Flags:

```
33 private def job = {
34 val promise = Promise[JobDone.type]()
35
36 context.system.scheduler.scheduleOnce(ThreadLocalRandom.current.nextInt(300).millisecond) {
37 promise.success(JobDone)
38 }
39
```

Package: akka.util

scala/akka/util/IndexSpec.scala, line 131 (Insecure Randomness) — **High**

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: executeRandomTask()
File: scala/akka/util/IndexSpec.scala:131
Taint Flags:

```
128 }
129 }
130
131 def executeRandomTask() = Random.nextInt(4) match {
132 case 0 => putTask()
133 case 1 => removeTask1()
134 case 2 => removeTask2()
```

Insecure Randomness — High

Package: akka.util

scala/akka/util/ZipfianGenerator.scala, line 42 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextDouble()
Enclosing Method: next()
File: scala/akka/util/ZipfianGenerator.scala:42
Taint Flags:

```
39  private val random = new scala.util.Random(seed)
40
41  def next(): Int = {
42  val u = random.nextDouble()
43  val uz = u * zetaN
44  if (uz < 1.0) min
45  else if (uz < 1.0 + Math.pow(0.5, theta)) min + 1
```

Package: scala.akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 435 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextFloat()
Enclosing Method: apply()
File: scala/akka/actor/SupervisorHierarchySpec.scala:435
Taint Flags:

```
432  var idleChildren = Vector.empty[ActorRef]
433  var pingChildren = Set.empty[ActorRef]
434
435  val nextJob = Iterator.continually(random.nextFloat() match {
436  case x if x >= 0.5 =>
437  // ping one child
438  val pick = ((x - 0.5) * 2 * idleChildren.size).toInt
```

scala/akka/actor/SchedulerSpec.scala, line 440 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Insecure Randomness High

Package: scala.akka.actor

scala/akka/actor/SchedulerSpec.scala, line 440 (Insecure Randomness) High

Sink: nextInt()
Enclosing Method: apply()
File: scala/akka/actor/SchedulerSpec.scala:440
Taint Flags:

```
437 val r = ThreadLocalRandom.current
438 val N = 1000000
439 val tasks = for (_ <- 1 to N) yield {
440 val next = r.nextInt(3000)
441 val now = System.nanoTime
442 system.scheduler.scheduleOnce(next.millis) {
443 val stop = System.nanoTime
```

scala/akka/actor/SchedulerSpec.scala, line 165 (Insecure Randomness) High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: apply()
File: scala/akka/actor/SchedulerSpec.scala:165
Taint Flags:

```
162 val r = ThreadLocalRandom.current()
163 val N = 100000
164 for (_ <- 1 to N) {
165 val next = r.nextInt(3000)
166 val now = System.nanoTime
167 system.scheduler.scheduleOnce(next.millis) {
168 val stop = System.nanoTime
```

Package: scala.akka.io

scala/akka/io/TcpConnectionSpec.scala, line 200 (Insecure Randomness) High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextBytes()
Enclosing Method: apply()
File: scala/akka/io/TcpConnectionSpec.scala:200
Taint Flags:

```
197 val bufferSize = 512 * 1024 // 256kB are too few
```

Insecure Randomness High

Package: scala.akka.io

scala/akka/io/TcpConnectionSpec.scala, line 200 (Insecure Randomness) High

```
198 val random = new Random(0)
199 val testBytes = new Array[Byte](bufferSize)
200 random.nextBytes(testBytes)
201 val testData = ByteString(testBytes)
202
203 val writer = TestProbe()
```

Package: scala.akka.routing

scala/akka/routing/MetricsBasedResizerSpec.scala, line 51 (Insecure Randomness) High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
File: scala/akka/routing/MetricsBasedResizerSpec.scala:51
Taint Flags:

```
48 def mockSend(
49 await: Boolean,
50 l: TestLatch = TestLatch(),
51 routeeIdx: Int = Random.nextInt(routees.length)): Latches = {
52 val target = routees(routeeIdx)
53 val first = TestLatch()
54 val latches = Latches(first, l)
```

Package: scala.akka.serialization

scala/akka/serialization/PrimitivesSerializationSpec.scala, line 123 (Insecure Randomness) High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextString()
Enclosing Method: apply()
File: scala/akka/serialization/PrimitivesSerializationSpec.scala:123
Taint Flags:

```
120 }
121
122 "StringSerializer" must {
123 val random = Random.nextString(256)
124 Seq("empty string" -> "", "hello" -> "hello", "árvíztrütvefúrógép" -> "árvíztrütvefúrógép", "random" -> random)
```

Insecure Randomness — High

Package: scala.akka.serialization

scala/akka/serialization/PrimitivesSerializationSpec.scala, line 123 (Insecure Randomness) — High

```
125 .foreach {
126 case (scenario, item) =>
```

Package: scala.akka.util

scala/akka/util/IndexSpec.scala, line 118 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: apply()
File: scala/akka/util/IndexSpec.scala:118
Taint Flags:

```
115 index.put(Random.nextInt(nrOfKeys), Random.nextInt(nrOfValues))
116 }
117 def removeTask1() = Future {
118 index.remove(Random.nextInt(nrOfKeys / 2), Random.nextInt(nrOfValues))
119 }
120 def removeTask2() = Future {
121 index.remove(Random.nextInt(nrOfKeys / 2))
```

scala/akka/util/IndexSpec.scala, line 115 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: apply()
File: scala/akka/util/IndexSpec.scala:115
Taint Flags:

```
112 index.put(key, value)
113 //Tasks to be executed in parallel
114 def putTask() = Future {
115 index.put(Random.nextInt(nrOfKeys), Random.nextInt(nrOfValues))
116 }
117 def removeTask1() = Future {
118 index.remove(Random.nextInt(nrOfKeys / 2), Random.nextInt(nrOfValues))
```

Insecure Randomness — High

Package: scala.akka.util

scala/akka/util/TokenBucketSpec.scala, line 233 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: apply()
File: scala/akka/util/TokenBucketSpec.scala:233
Taint Flags:

```
230
231 if (untilNextElement == 0) {
232 // Allow cost of zer
233 val cost = Random.nextInt(maxCost + 1)
234 idealBucket -= cost // This can go negative
235 bucket.currentTime = startTime + time
236 val delay = bucket.offer(cost)
```

scala/akka/util/IndexSpec.scala, line 121 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: apply()
File: scala/akka/util/IndexSpec.scala:121
Taint Flags:

```
118 index.remove(Random.nextInt(nrOfKeys / 2), Random.nextInt(nrOfValues))
119 }
120 def removeTask2() = Future {
121 index.remove(Random.nextInt(nrOfKeys / 2))
122 }
123 def readTask() = Future {
124 val key = Random.nextInt(nrOfKeys)
```

scala/akka/util/IndexSpec.scala, line 118 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()

Insecure Randomness — High

Package: scala.akka.util

scala/akka/util/IndexSpec.scala, line 118 (Insecure Randomness) — High

Enclosing Method: apply()
File: scala/akka/util/IndexSpec.scala:118
Taint Flags:

```
115  index.put(Random.nextInt(nrOfKeys), Random.nextInt(nrOfValues))
116  }
117  def removeTask1() = Future {
118  index.remove(Random.nextInt(nrOfKeys / 2), Random.nextInt(nrOfValues))
119  }
120  def removeTask2() = Future {
121  index.remove(Random.nextInt(nrOfKeys / 2))
```

scala/akka/util/ImmutableIntMapSpec.scala, line 134 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: apply()
File: scala/akka/util/ImmutableIntMapSpec.scala:134
Taint Flags:

```
131
132  (1 to 1000).foreach { i =>
133  withClue(s"seed=$seed, iteration=$i") {
134  val key = rnd.nextInt(100)
135  val value = rnd.nextPrintableChar()
136  rnd.nextInt(3) match {
137  case 0 | 1 =>
```

scala/akka/util/ImmutableIntMapSpec.scala, line 135 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextPrintableChar()
Enclosing Method: apply()
File: scala/akka/util/ImmutableIntMapSpec.scala:135
Taint Flags:

```
132  (1 to 1000).foreach { i =>
133  withClue(s"seed=$seed, iteration=$i") {
134  val key = rnd.nextInt(100)
```

Insecure Randomness High

Package: scala.akka.util

scala/akka/util/ImmutableIntMapSpec.scala, line 135 (Insecure Randomness) High

```
135 val value = rnd.nextPrintableChar()
136 rnd.nextInt(3) match {
137 case 0 | 1 =>
138 longMap = longMap.updated(key, value)
```

scala/akka/util/TokenBucketSpec.scala, line 241 (Insecure Randomness) High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: apply()
File: scala/akka/util/TokenBucketSpec.scala:241
Taint Flags:

```
238 if (Debug) println(s" ARRIVAL cost: $cost at: $nextEmit")
239 if (delay == 0) {
240 (idealBucket >= 0) should be(true)
241 untilNextElement = time + Random.nextInt(arrivalPeriod)
242 } else delaying = true
243 }
244
```

scala/akka/util/TokenBucketSpec.scala, line 226 (Insecure Randomness) High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: apply()
File: scala/akka/util/TokenBucketSpec.scala:226
Taint Flags:

```
223 if (delaying && idealBucket == 0) {
224 // Actual emit time should equal to what the optimized token bucket calculates
225 time.toLong should ===(nextEmit)
226 untilNextElement = time + Random.nextInt(arrivalPeriod)
227 if (Debug) println(s" EMITTING")
228 delaying = false
229 }
```

Insecure Randomness High

Package: scala.akka.util

scala/akka/util/TokenBucketSpec.scala, line 211 (Insecure Randomness) High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: apply()
File: scala/akka/util/TokenBucketSpec.scala:211
Taint Flags:

```
208
209 var idealBucket = capacity
210 var untilNextTick = period
211 var untilNextElement = Random.nextInt(arrivalPeriod) + 1
212 var nextEmit = 0L
213 var delaying = false
214
```

scala/akka/util/ImmutableIntMapSpec.scala, line 136 (Insecure Randomness) High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: apply()
File: scala/akka/util/ImmutableIntMapSpec.scala:136
Taint Flags:

```
133 withClue(s"seed=$seed, iteration=$i") {
134 val key = rnd.nextInt(100)
135 val value = rnd.nextPrintableChar()
136 rnd.nextInt(3) match {
137 case 0 | 1 =>
138 longMap = longMap.updated(key, value)
139 reference = reference.updated(key, value)
```

scala/akka/util/IndexSpec.scala, line 124 (Insecure Randomness) High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()

Insecure Randomness High

Package: scala.akka.util

scala/akka/util/IndexSpec.scala, line 124 (Insecure Randomness) High

Enclosing Method: apply()
File: scala/akka/util/IndexSpec.scala:124
Taint Flags:

```
121 index.remove(Random.nextInt(nrOfKeys / 2))
122 }
123 def readTask() = Future {
124 val key = Random.nextInt(nrOfKeys)
125 val values = index.valueIterator(key)
126 if (key >= nrOfKeys / 2) {
127 values.isEmpty should ===(false)
```

scala/akka/util/IndexSpec.scala, line 115 (Insecure Randomness) High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: apply()
File: scala/akka/util/IndexSpec.scala:115
Taint Flags:

```
112 index.put(key, value)
113 //Tasks to be executed in parallel
114 def putTask() = Future {
115 index.put(Random.nextInt(nrOfKeys), Random.nextInt(nrOfValues))
116 }
117 def removeTask1() = Future {
118 index.remove(Random.nextInt(nrOfKeys / 2), Random.nextInt(nrOfValues))
```

Insecure Randomness: Hardcoded Seed (2 issues)

Abstract

Functions that generate random or pseudorandom values, which are passed a seed, should not be called with a constant argument.

Explanation

Functions that generate random or pseudorandom values, which are passed a seed, should not be called with a constant argument. If a pseudorandom number generator (such as `Random`) is seeded with a specific value (using a function such as `Random.setSeed()`), the values returned by `Random.nextInt()` and similar methods which return or assign values are predictable for an attacker that can collect a number of PRNG outputs. **Example 1:** The values produced by the `Random` object `s` are predictable from the `Random` object `r`.

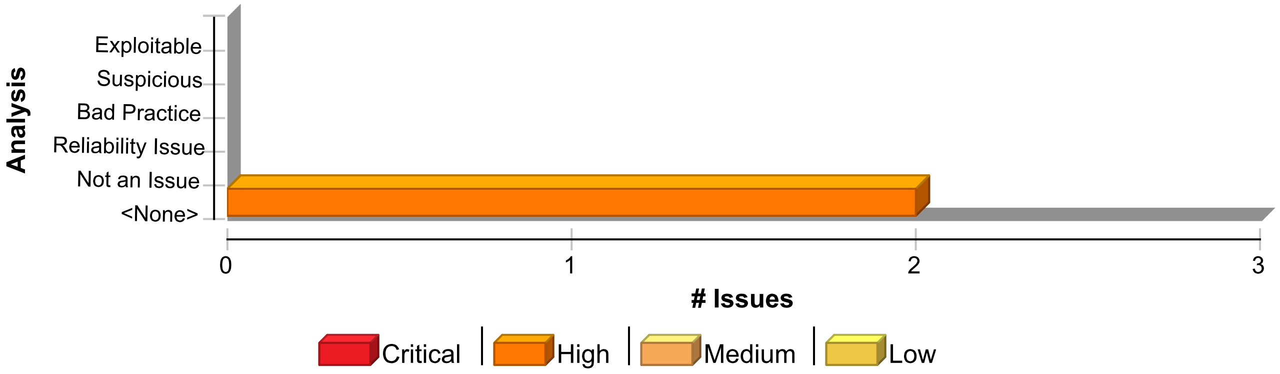
```
        Random r = new Random();
        r.setSeed(12345);
        int i = r.nextInt();
        byte[] b = new byte[4];
        r.nextBytes(b);

        Random s = new Random();
        s.setSeed(12345);
        int j = s.nextInt();
        byte[] c = new byte[4];
        s.nextBytes(c);
```

In this example, pseudorandom number generators: `r` and `s` were identically seeded, so `i == j`, and corresponding values of arrays `b[]` and `c[]` are equal.

Recommendation

Use a cryptographic PRNG seeded with hardware-based sources of randomness, such as ring oscillators, disk drive timing, thermal noise, or radioactive decay. Doing so makes the sequence of data produced by `Random.nextInt()` and similar methods much harder to predict than setting the seed to a constant.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Insecure Randomness: Hardcoded Seed	2	0	0	2
Total	**2**	**0**	**0**	**2**

Insecure Randomness: Hardcoded Seed High

Package: akka.util

scala/akka/util/ZipfianGenerator.scala, line 39 (Insecure Randomness: Hardcoded Seed) High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: Random()
Enclosing Method: ZipfianGenerator()
File: scala/akka/util/ZipfianGenerator.scala:39
Taint Flags:

```
36  private val zeta2 = ZipfianGenerator.zeta(2, theta)
37  private val zetaN = ZipfianGenerator.zeta(n, theta)
38  private val eta = (1 - Math.pow(2.0 / n, 1 - theta)) / (1 - zeta2 / zetaN)
39  private val random = new scala.util.Random(seed)
40
41  def next(): Int = {
42  val u = random.nextDouble()
```

Package: scala.akka.io

scala/akka/io/TcpConnectionSpec.scala, line 198 (Insecure Randomness: Hardcoded Seed) High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: Random()
Enclosing Method: apply()
File: scala/akka/io/TcpConnectionSpec.scala:198
Taint Flags:

```
195  "send big buffers to network correctly" in new EstablishedConnectionTest() {
196  run {
197  val bufferSize = 512 * 1024 // 256kB are too few
198  val random = new Random(0)
199  val testBytes = new Array[Byte](bufferSize)
200  random.nextBytes(testBytes)
201  val testData = ByteString(testBytes)
```

J2EE Bad Practices: Leftover Debug Code (1 issue)

Abstract

Debug code can create unintended entry points in a deployed web application.

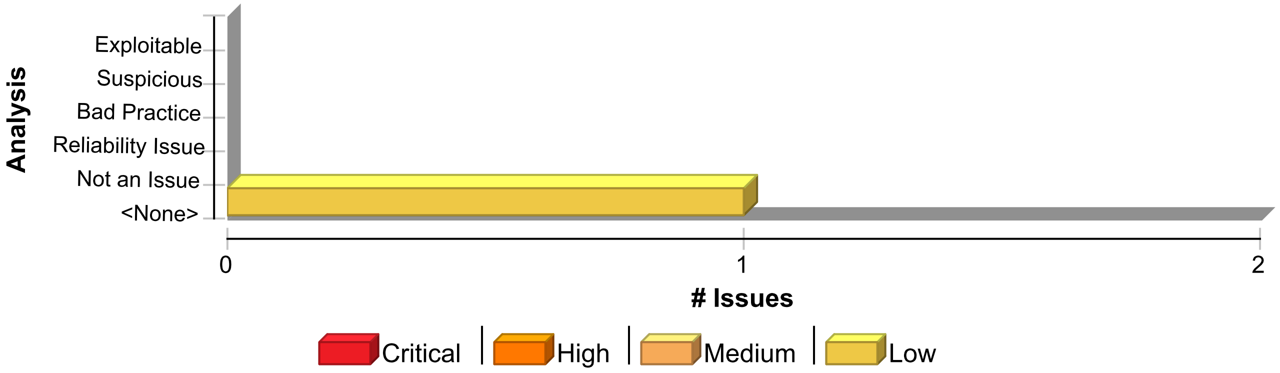
Explanation

A common development practice is to add "back door" code specifically designed for debugging or testing purposes that is not intended to be shipped or deployed with the application. When this sort of debug code is accidentally left in the application, the application is open to unintended modes of interaction. These back door entry points create security risks because they are not considered during design or testing and fall outside of the expected operating conditions of the application. The most common example of forgotten debug code is a `main()` method appearing in a web application. Although this is an acceptable practice during product development, classes that are part of a production J2EE application should not define a `main()`.

Recommendation

Remove debug code before deploying a production version of an application. Regardless of whether a direct security threat can be articulated, it is unlikely that there is a legitimate reason for such code to remain in the application after the early stages of development.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
J2EE Bad Practices: Leftover Debug Code	1	0	0	1
Total	**1**	**0**	**0**	**1**

J2EE Bad Practices: Leftover Debug Code — Low

Package: akka.actor

scala/akka/actor/Bench.scala, line 121 (J2EE Bad Practices: Leftover Debug Code) **Low**

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

J2EE Bad Practices: Leftover Debug Code | Low

Package: akka.actor

scala/akka/actor/Bench.scala, line 121 (J2EE Bad Practices: Leftover Debug Code) | Low

Sink: Function: main
Enclosing Method: main()
File: scala/akka/actor/Bench.scala:121
Taint Flags:

```
118  system.terminate()
119  }
120
121  def main(args: Array[String]): Unit = run()
122 }
123
124 undefined
```

J2EE Bad Practices: Sockets (32 issues)

Abstract

Socket-based communication in web applications is prone to error.

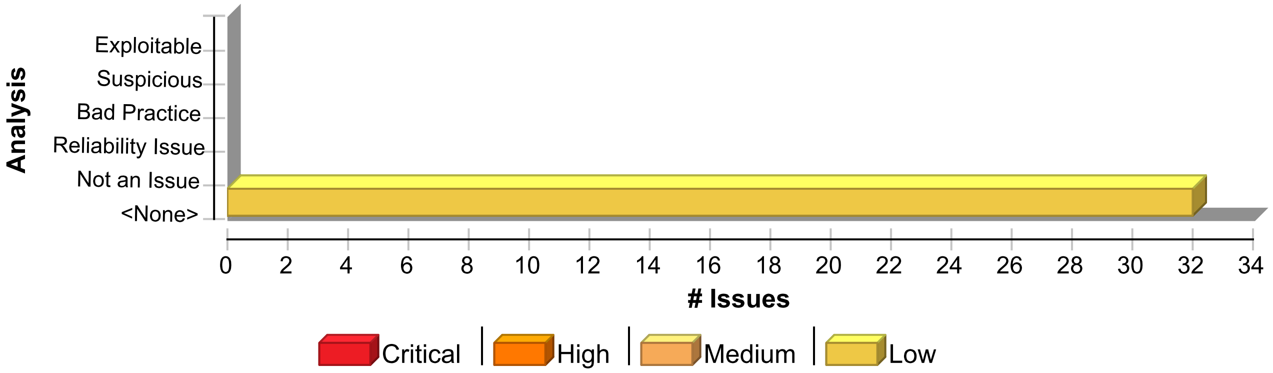
Explanation

The J2EE standard permits the use of sockets only for the purpose of communication with legacy systems when no higher-level protocol is available. Authoring your own communication protocol requires wrestling with difficult security issues, including: - In-band versus out-of-band signaling - Compatibility between protocol versions - Channel security - Error handling - Network constraints (firewalls) - Session management Without significant scrutiny by a security expert, chances are good that a custom communication protocol will suffer from security problems. Many of the same issues apply to a custom implementation of a standard protocol. While there are usually more resources available that address security concerns related to implementing a standard protocol, these resources are also available to attackers.

Recommendation

Replace a custom communication protocol with an industry standard protocol or framework. Consider whether you can use a protocol such as HTTP, FTP, SMTP, CORBA, RMI/IIOP, EJB, or SOAP. Consider the security track record of the protocol implementation you choose.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
J2EE Bad Practices: Sockets	32	0	0	32
Total	**32**	**0**	**0**	**32**

J2EE Bad Practices: Sockets	Low

Package: akka.io

scala/akka/io/UdpIntegrationSpec.scala, line 23 (J2EE Bad Practices: Sockets)	Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

J2EE Bad Practices: Sockets — Low

Package: akka.io

scala/akka/io/UdpIntegrationSpec.scala, line 23 (J2EE Bad Practices: Sockets) — Low

Sink: InetSocketAddress()
Enclosing Method: bindUdp()
File: scala/akka/io/UdpIntegrationSpec.scala:23
Taint Flags:

```
20
21  def bindUdp(handler: ActorRef): InetSocketAddress = {
22  val commander = TestProbe()
23  commander.send(IO(Udp), Bind(handler, new InetSocketAddress("127.0.0.1", 0)))
24  commander.expectMsgType[Bound].localAddress
25  }
26
```

scala/akka/io/TcpConnectionSpec.scala, line 61 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: InetSocketAddress()
Enclosing Method: liftedTree1()
File: scala/akka/io/TcpConnectionSpec.scala:61
Taint Flags:

```
58  val serverSocket = ServerSocketChannel.open()
59  serverSocket.socket.bind(new InetSocketAddress("127.0.0.1", 0))
60  try {
61  val clientSocket = SocketChannel.open(new InetSocketAddress("127.0.0.1", serverSocket.socket().getLocalPort))
62  val clientSocketOnServer = acceptServerSideConnection(serverSocket)
63  clientSocketOnServer.socket.setSoLinger(true, 0)
64  clientSocketOnServer.close()
```

scala/akka/io/TcpListenerSpec.scala, line 165 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: Socket()
Enclosing Method: attemptConnectionToEndpoint()
File: scala/akka/io/TcpListenerSpec.scala:165
Taint Flags:

```
162  bindCommander.expectMsgType[Bound]
163  }
```

J2EE Bad Practices: Sockets — Low

Package: akka.io

scala/akka/io/TcpListenerSpec.scala, line 165 (J2EE Bad Practices: Sockets) — Low

```
164
165 def attemptConnectionToEndpoint(): Unit = new Socket(endpoint.getHostName, endpoint.getPort)
166
167 def listener = parentRef.underlyingActor.listener
168
```

scala/akka/io/TcpConnectionSpec.scala, line 74 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: InetSocketAddress()
Enclosing Method: ConnectionRefusedMessagePrefix$lzycompute()
File: scala/akka/io/TcpConnectionSpec.scala:74
Taint Flags:

```
71
72 lazy val ConnectionRefusedMessagePrefix: String = {
73 val serverSocket = ServerSocketChannel.open()
74 serverSocket.socket.bind(new InetSocketAddress("127.0.0.1", 0))
75 try {
76 serverSocket.close()
77 val clientSocket = SocketChannel.open(new InetSocketAddress("127.0.0.1", serverSocket.socket().getLocalPort))
```

scala/akka/io/TcpConnectionSpec.scala, line 59 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: InetSocketAddress()
Enclosing Method: ConnectionResetByPeerMessage$lzycompute()
File: scala/akka/io/TcpConnectionSpec.scala:59
Taint Flags:

```
56
57 lazy val ConnectionResetByPeerMessage: String = {
58 val serverSocket = ServerSocketChannel.open()
59 serverSocket.socket.bind(new InetSocketAddress("127.0.0.1", 0))
60 try {
61 val clientSocket = SocketChannel.open(new InetSocketAddress("127.0.0.1", serverSocket.socket().getLocalPort))
62 val clientSocketOnServer = acceptServerSideConnection(serverSocket)
```

J2EE Bad Practices: Sockets — Low

Package: akka.io

scala/akka/io/TcpConnectionSpec.scala, line 643 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: InetSocketAddress()
Enclosing Method: TcpConnectionSpec$$anon$27()
File: scala/akka/io/TcpConnectionSpec.scala:643
Taint Flags:

```
640
641 "report failed connection attempt when target cannot be resolved" in
642 new UnacceptedConnectionTest() {
643 val address = new InetSocketAddress("notthere.local", 666)
644 override lazy val connectionActor = createConnectionActorWithoutRegistration(serverAddress = address)
645 run {
646 connectionActor ! newChannelRegistration
```

scala/akka/io/TcpConnectionSpec.scala, line 77 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: InetSocketAddress()
Enclosing Method: liftedTree2()
File: scala/akka/io/TcpConnectionSpec.scala:77
Taint Flags:

```
74 serverSocket.socket.bind(new InetSocketAddress("127.0.0.1", 0))
75 try {
76 serverSocket.close()
77 val clientSocket = SocketChannel.open(new InetSocketAddress("127.0.0.1", serverSocket.socket().getLocalPort))
78 clientSocket.finishConnect()
79 clientSocket.write(ByteBuffer.allocate(1))
80 null
```

scala/akka/io/TcpIntegrationSpec.scala, line 179 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

J2EE Bad Practices: Sockets

Low

Package: akka.io

scala/akka/io/TcpIntegrationSpec.scala, line 179 (J2EE Bad Practices: Sockets)

Low

Sink: ServerSocket()
Enclosing Method: TcpIntegrationSpec$$anon$12()
File: scala/akka/io/TcpIntegrationSpec.scala:179
Taint Flags:

```
176 }
177
178 "handle tcp connection actor death properly" in new TestSetup(shouldBindServer = false) {
179 val serverSocket = new ServerSocket(endpoint.getPort(), 100, endpoint.getAddress())
180 val connectCommander = TestProbe()
181 connectCommander.send(IO(Tcp), Connect(endpoint))
182
```

Package: scala.akka.io

scala/akka/io/TcpIntegrationSpec.scala, line 171 (J2EE Bad Practices: Sockets)

Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: InetSocketAddress()
Enclosing Method: apply()
File: scala/akka/io/TcpIntegrationSpec.scala:171
Taint Flags:

```
168 "don't report Connected when endpoint isn't responding" in {
169 val connectCommander = TestProbe()
170 // a "random" endpoint hopefully unavailable since it's in the test-net IP range
171 val endpoint = new InetSocketAddress("192.0.2.1", 23825)
172 connectCommander.send(IO(Tcp), Connect(endpoint))
173 // expecting CommandFailed or no reply (within timeout)
174 val replies = connectCommander.receiveWhile(1.second) { case m: Connected => m }
```

scala/akka/io/UdpIntegrationSpec.scala, line 117 (J2EE Bad Practices: Sockets)

Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: InetSocketAddress()
Enclosing Method: apply()
File: scala/akka/io/UdpIntegrationSpec.scala:117
Taint Flags:

```
114 "call DatagramChannelCreator.create method when opening channel" in {
```

J2EE Bad Practices: Sockets — Low

Package: scala.akka.io

scala/akka/io/UdpIntegrationSpec.scala, line 117 (J2EE Bad Practices: Sockets) — Low

```
115 val commander = TestProbe()
116 val assertOption = AssertOpenDatagramChannel()
117 commander.send(IO(Udp), Bind(testActor, new InetSocketAddress("127.0.0.1", 0), options = List(assertOption)))
118 commander.expectMsgType[Bound]
119 assert(assertOption.openCalled === 1)
120 }
```

scala/akka/io/UdpIntegrationSpec.scala, line 109 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: InetSocketAddress()
Enclosing Method: apply()
File: scala/akka/io/UdpIntegrationSpec.scala:109
Taint Flags:

```
106 "call SocketOption.afterConnect method after binding." in {
107 val commander = TestProbe()
108 val assertOption = AssertAfterChannelBind()
109 commander.send(IO(Udp), Bind(testActor, new InetSocketAddress("127.0.0.1", 0), options = List(assertOption)))
110 commander.expectMsgType[Bound]
111 assert(assertOption.afterCalled === 1)
112 }
```

scala/akka/io/UdpIntegrationSpec.scala, line 101 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: InetSocketAddress()
Enclosing Method: apply()
File: scala/akka/io/UdpIntegrationSpec.scala:101
Taint Flags:

```
98 "call SocketOption.beforeBind method before bind." in {
99 val commander = TestProbe()
100 val assertOption = AssertBeforeBind()
101 commander.send(IO(Udp), Bind(testActor, new InetSocketAddress("127.0.0.1", 0), options = List(assertOption)))
102 commander.expectMsgType[Bound]
103 assert(assertOption.beforeCalled === 1)
104 }
```

J2EE Bad Practices: Sockets — Low

Package: scala.akka.io

scala/akka/io/UdpIntegrationSpec.scala, line 101 (J2EE Bad Practices: Sockets) — Low

scala/akka/io/UdpIntegrationSpec.scala, line 53 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: InetSocketAddress()
Enclosing Method: apply()
File: scala/akka/io/UdpIntegrationSpec.scala:53
Taint Flags:

```
50  }
51
52  "be able to deliver subsequent messages after address resolution failure" in {
53  val unresolvableServerAddress = new InetSocketAddress("some-unresolvable-host", 10000)
54  val cmd = Send(ByteString("Can't be delivered"), unresolvableServerAddress)
55  val simpleSender = createSimpleSender()
56  simpleSender ! cmd
```

scala/akka/io/TcpConnectionSpec.scala, line 836 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: ServerSocket()
Enclosing Method: apply()
File: scala/akka/io/TcpConnectionSpec.scala:836
Taint Flags:

```
833  // This test needs the OP_CONNECT workaround on Windows, see original report #15033 and parent ticket #15766
834
835  val bindAddress = SocketUtil.temporaryServerAddress()
836  val serverSocket = new ServerSocket(bindAddress.getPort, 100, bindAddress.getAddress)
837  val connectionProbe = TestProbe()
838
839  connectionProbe.send(IO(Tcp), Connect(bindAddress))
```

scala/akka/io/UdpConnectedIntegrationSpec.scala, line 64 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

J2EE Bad Practices: Sockets — Low

Package: scala.akka.io

scala/akka/io/UdpConnectedIntegrationSpec.scala, line 64 (J2EE Bad Practices: Sockets) — Low

Sink Details

Sink: createUnresolved()
Enclosing Method: apply()
File: scala/akka/io/UdpConnectedIntegrationSpec.scala:64
Taint Flags:

```
61  val serverAddress = "doesnotexist.local"
62  val commander = TestProbe()
63  val handler = TestProbe()
64  val command = UdpConnected.Connect(handler.ref, InetSocketAddress.createUnresolved(serverAddress, 1234), None)
65  commander.send(IO(UdpConnected), command)
66  commander.expectMsg(6.seconds, UdpConnected.CommandFailed(command))
67  }
```

scala/akka/io/UdpConnectedIntegrationSpec.scala, line 55 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: createUnresolved()
Enclosing Method: apply()
File: scala/akka/io/UdpConnectedIntegrationSpec.scala:55
Taint Flags:

```
52  val serverAddress = "doesnotexist.local"
53  val commander = TestProbe()
54  val handler = TestProbe()
55  val command = UdpConnected.Connect(handler.ref, InetSocketAddress.createUnresolved(serverAddress, 1234), None)
56  commander.send(IO(UdpConnected), command)
57  commander.expectMsg(10.seconds, UdpConnected.CommandFailed(command))
58  }
```

scala/akka/io/UdpConnectedIntegrationSpec.scala, line 64 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: createUnresolved()
Enclosing Method: apply()
File: scala/akka/io/UdpConnectedIntegrationSpec.scala:64
Taint Flags:

```
61  val serverAddress = "doesnotexist.local"
```

J2EE Bad Practices: Sockets — Low

Package: scala.akka.io

scala/akka/io/UdpConnectedIntegrationSpec.scala, line 64 (J2EE Bad Practices: Sockets) — Low

```
62 val commander = TestProbe()
63 val handler = TestProbe()
64 val command = UdpConnected.Connect(handler.ref, InetSocketAddress.createUnresolved(serverAddress, 1234), None)
65 commander.send(IO(UdpConnected), command)
66 commander.expectMsg(6.seconds, UdpConnected.CommandFailed(command))
67 }
```

scala/akka/io/UdpConnectedIntegrationSpec.scala, line 55 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: createUnresolved()
Enclosing Method: apply()
File: scala/akka/io/UdpConnectedIntegrationSpec.scala:55
Taint Flags:

```
52 val serverAddress = "doesnotexist.local"
53 val commander = TestProbe()
54 val handler = TestProbe()
55 val command = UdpConnected.Connect(handler.ref, InetSocketAddress.createUnresolved(serverAddress, 1234), None)
56 commander.send(IO(UdpConnected), command)
57 commander.expectMsg(10.seconds, UdpConnected.CommandFailed(command))
58 }
```

Package: scala.akka.io.dns

scala/akka/io/dns/NameserverAddressParserSpec.scala, line 18 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: InetSocketAddress()
Enclosing Method: apply()
File: scala/akka/io/dns/NameserverAddressParserSpec.scala:18
Taint Flags:

```
15 DnsSettings.parseNameserverAddress("8.8.8.8:153") shouldEqual new InetSocketAddress("8.8.8.8", 153)
16 }
17 "handle explicit port in IPv6 address" in {
18 DnsSettings.parseNameserverAddress("[2001:4860:4860::8888]:153") shouldEqual new InetSocketAddress(
19 "2001:4860:4860::8888",
```

J2EE Bad Practices: Sockets — Low

Package: scala.akka.io.dns

scala/akka/io/dns/NameserverAddressParserSpec.scala, line 18 (J2EE Bad Practices: Sockets) — Low

```
20  153)
21  }
```

scala/akka/io/dns/NameserverAddressParserSpec.scala, line 26 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: InetSocketAddress()
Enclosing Method: apply()
File: scala/akka/io/dns/NameserverAddressParserSpec.scala:26
Taint Flags:

```
23  DnsSettings.parseNameserverAddress("8.8.8.8") shouldEqual new InetSocketAddress("8.8.8.8", 53)
24  }
25  "handle default port in IPv6 address" in {
26  DnsSettings.parseNameserverAddress("[2001:4860:4860::8888]") shouldEqual new InetSocketAddress(
27  "2001:4860:4860::8888",
28  53)
29  }
```

scala/akka/io/dns/NameserverAddressParserSpec.scala, line 15 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: InetSocketAddress()
Enclosing Method: apply()
File: scala/akka/io/dns/NameserverAddressParserSpec.scala:15
Taint Flags:

```
12 class NameserverAddressParserSpec extends AnyWordSpec with Matchers {
13  "Parser" should {
14  "handle explicit port in IPv4 address" in {
15  DnsSettings.parseNameserverAddress("8.8.8.8:153") shouldEqual new InetSocketAddress("8.8.8.8", 153)
16  }
17  "handle explicit port in IPv6 address" in {
18  DnsSettings.parseNameserverAddress("[2001:4860:4860::8888]:153") shouldEqual new InetSocketAddress(
```

J2EE Bad Practices: Sockets — Low

Package: scala.akka.io.dns

scala/akka/io/dns/NameserverAddressParserSpec.scala, line 23 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: InetSocketAddress()
Enclosing Method: apply()
File: scala/akka/io/dns/NameserverAddressParserSpec.scala:23
Taint Flags:

```
20 153)
21 }
22 "handle default port in IPv4 address" in {
23 DnsSettings.parseNameserverAddress("8.8.8.8") shouldEqual new InetSocketAddress("8.8.8.8", 53)
24 }
25 "handle default port in IPv6 address" in {
26 DnsSettings.parseNameserverAddress("[2001:4860:4860::8888]") shouldEqual new InetSocketAddress(
```

Package: scala.akka.io.dns.internal

scala/akka/io/dns/internal/TcpDnsClientSpec.scala, line 26 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: createUnresolved()
Enclosing Method: apply()
File: scala/akka/io/dns/internal/TcpDnsClientSpec.scala:26
Taint Flags:

```
23 Message(42, MessageFlags(), questions = Seq(Question("akka.io", RecordType.A, RecordClass.IN)))
24 val exampleResponseMessage = Message(42, MessageFlags(answer = true))
25 val dnsServerAddress = InetSocketAddress.createUnresolved("foo", 53)
26 val localAddress = InetSocketAddress.createUnresolved("localhost", 13441)
27
28 "reconnect when the server closes the connection" in {
29 val tcpExtensionProbe = TestProbe()
```

scala/akka/io/dns/internal/TcpDnsClientSpec.scala, line 25 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

J2EE Bad Practices: Sockets — Low

Package: scala.akka.io.dns.internal

scala/akka/io/dns/internal/TcpDnsClientSpec.scala, line 25 (J2EE Bad Practices: Sockets) — Low

Sink Details

Sink: createUnresolved()
Enclosing Method: apply()
File: scala/akka/io/dns/internal/TcpDnsClientSpec.scala:25
Taint Flags:

```
22 val exampleRequestMessage =
23 Message(42, MessageFlags(), questions = Seq(Question("akka.io", RecordType.A, RecordClass.IN)))
24 val exampleResponseMessage = Message(42, MessageFlags(answer = true))
25 val dnsServerAddress = InetSocketAddress.createUnresolved("foo", 53)
26 val localAddress = InetSocketAddress.createUnresolved("localhost", 13441)
27
28 "reconnect when the server closes the connection" in {
```

scala/akka/io/dns/internal/DnsClientSpec.scala, line 25 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: createUnresolved()
Enclosing Method: apply()
File: scala/akka/io/dns/internal/DnsClientSpec.scala:25
Taint Flags:

```
22 Message(42, MessageFlags(), questions = Seq(Question("akka.io", RecordType.A, RecordClass.IN)))
23 val exampleResponseMessage = Message(42, MessageFlags(answer = true))
24 val exampleResponse = Answer(42, Nil)
25 val dnsServerAddress = InetSocketAddress.createUnresolved("foo", 53)
26
27 "not connect to the DNS server over TCP eagerly" in {
28 val udpExtensionProbe = TestProbe()
```

scala/akka/io/dns/internal/TcpDnsClientSpec.scala, line 25 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: createUnresolved()
Enclosing Method: apply()
File: scala/akka/io/dns/internal/TcpDnsClientSpec.scala:25
Taint Flags:

```
22 val exampleRequestMessage =
```

J2EE Bad Practices: Sockets — Low

Package: scala.akka.io.dns.internal

scala/akka/io/dns/internal/TcpDnsClientSpec.scala, line 25 (J2EE Bad Practices: Sockets) — Low

```
23 Message(42, MessageFlags(), questions = Seq(Question("akka.io", RecordType.A, RecordClass.IN)))
24 val exampleResponseMessage = Message(42, MessageFlags(answer = true))
25 val dnsServerAddress = InetSocketAddress.createUnresolved("foo", 53)
26 val localAddress = InetSocketAddress.createUnresolved("localhost", 13441)
27
28 "reconnect when the server closes the connection" in {
```

scala/akka/io/dns/internal/DnsClientSpec.scala, line 25 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: createUnresolved()
Enclosing Method: apply()
File: scala/akka/io/dns/internal/DnsClientSpec.scala:25
Taint Flags:

```
22 Message(42, MessageFlags(), questions = Seq(Question("akka.io", RecordType.A, RecordClass.IN)))
23 val exampleResponseMessage = Message(42, MessageFlags(answer = true))
24 val exampleResponse = Answer(42, Nil)
25 val dnsServerAddress = InetSocketAddress.createUnresolved("foo", 53)
26
27 "not connect to the DNS server over TCP eagerly" in {
28 val udpExtensionProbe = TestProbe()
```

scala/akka/io/dns/internal/TcpDnsClientSpec.scala, line 26 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: createUnresolved()
Enclosing Method: apply()
File: scala/akka/io/dns/internal/TcpDnsClientSpec.scala:26
Taint Flags:

```
23 Message(42, MessageFlags(), questions = Seq(Question("akka.io", RecordType.A, RecordClass.IN)))
24 val exampleResponseMessage = Message(42, MessageFlags(answer = true))
25 val dnsServerAddress = InetSocketAddress.createUnresolved("foo", 53)
26 val localAddress = InetSocketAddress.createUnresolved("localhost", 13441)
27
28 "reconnect when the server closes the connection" in {
29 val tcpExtensionProbe = TestProbe()
```

J2EE Bad Practices: Sockets Low

Package: scala.akka.io.dns.internal

scala/akka/io/dns/internal/TcpDnsClientSpec.scala, line 26 (J2EE Bad Practices: Sockets) Low

scala/akka/io/dns/internal/DnsClientSpec.scala, line 66 (J2EE Bad Practices: Sockets) Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: createUnresolved()
Enclosing Method: apply()
File: scala/akka/io/dns/internal/DnsClientSpec.scala:66
Taint Flags:

```
63 client ! exampleRequest
64
65 udpExtensionProbe.expectMsgType[Udp.Bind]
66 udpExtensionProbe.lastSender ! Udp.Bound(InetSocketAddress.createUnresolved("localhost", 41325))
67
68 expectMsgType[Udp.Send]
69 client ! Udp.Received(Message(exampleRequest.id, MessageFlags(truncated = true)).write(), dnsServerAddress)
```

scala/akka/io/dns/internal/DnsClientSpec.scala, line 43 (J2EE Bad Practices: Sockets) Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: createUnresolved()
Enclosing Method: apply()
File: scala/akka/io/dns/internal/DnsClientSpec.scala:43
Taint Flags:

```
40 client ! exampleRequest
41
42 udpExtensionProbe.expectMsgType[Udp.Bind]
43 udpExtensionProbe.lastSender ! Udp.Bound(InetSocketAddress.createUnresolved("localhost", 41325))
44
45 expectMsgType[Udp.Send]
46 client ! Udp.Received(exampleResponseMessage.write(), dnsServerAddress)
```

scala/akka/io/dns/internal/DnsClientSpec.scala, line 66 (J2EE Bad Practices: Sockets) Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

J2EE Bad Practices: Sockets — Low

Package: scala.akka.io.dns.internal

scala/akka/io/dns/internal/DnsClientSpec.scala, line 66 (J2EE Bad Practices: Sockets) — Low

Sink Details

Sink: createUnresolved()
Enclosing Method: apply()
File: scala/akka/io/dns/internal/DnsClientSpec.scala:66
Taint Flags:

```
63 client ! exampleRequest
64
65 udpExtensionProbe.expectMsgType[Udp.Bind]
66 udpExtensionProbe.lastSender ! Udp.Bound(InetSocketAddress.createUnresolved("localhost", 41325))
67
68 expectMsgType[Udp.Send]
69 client ! Udp.Received(Message(exampleRequest.id, MessageFlags(truncated = true)).write(), dnsServerAddress)
```

scala/akka/io/dns/internal/DnsClientSpec.scala, line 43 (J2EE Bad Practices: Sockets) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: createUnresolved()
Enclosing Method: apply()
File: scala/akka/io/dns/internal/DnsClientSpec.scala:43
Taint Flags:

```
40 client ! exampleRequest
41
42 udpExtensionProbe.expectMsgType[Udp.Bind]
43 udpExtensionProbe.lastSender ! Udp.Bound(InetSocketAddress.createUnresolved("localhost", 41325))
44
45 expectMsgType[Udp.Send]
46 client ! Udp.Received(exampleResponseMessage.write(), dnsServerAddress)
```

J2EE Bad Practices: Threads (96 issues)

Abstract

Thread management in a web application is forbidden in some circumstances and is always highly error prone.

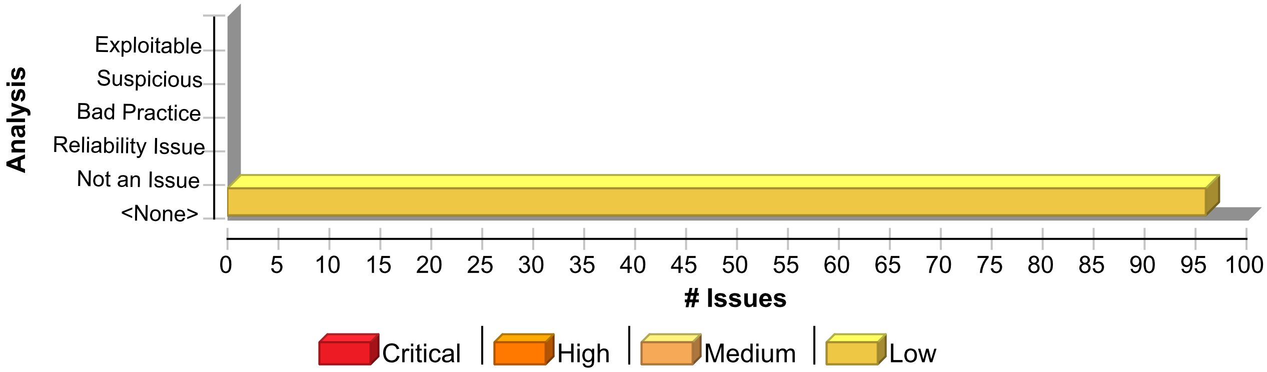
Explanation

Thread management in a web application is forbidden by the J2EE standard in some circumstances and is always highly error prone. Managing threads is difficult and is likely to interfere in unpredictable ways with the behavior of the application container. Even without interfering with the container, thread management usually leads to bugs that are hard to detect and diagnose like deadlock, race conditions, and other synchronization errors.

Recommendation

Avoid managing threads directly from within the web application. Instead use standards such as message driven beans and the EJB timer service that are provided by the application container.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
J2EE Bad Practices: Threads	96	0	0	96
Total	**96**	**0**	**0**	**96**

J2EE Bad Practices: Threads — Low

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 297 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:297
Taint Flags:

J2EE Bad Practices: Threads — Low

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 297 (J2EE Bad Practices: Threads) — Low

```
294 setFlags(f.directive)
295 stateCache.put(self.path, stateCache.get(self.path).copy(failConstr = f.copy()))
296 throw f
297 case "ping" => { Thread.sleep((random.nextFloat() * 1.03).toLong); sender() ! "pong" }
298 case Dump(0) => abort("dump")
299 case Dump(level) => context.children.foreach(_ ! Dump(level - 1))
300 case Terminated(ref) =>
```

scala/akka/actor/SchedulerSpec.scala, line 711 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: interrupt()
Enclosing Method: waitNanos()
File: scala/akka/actor/SchedulerSpec.scala:711
Taint Flags:

```
708 case null => 0L
709 })
710 catch {
711 case _: InterruptedException => Thread.currentThread.interrupt()
712 }
713 }
714
```

scala/akka/actor/TypedActorSpec.scala, line 147 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: optionPigdog()
File: scala/akka/actor/TypedActorSpec.scala:147
Taint Flags:

```
144 def optionPigdog(): Option[String] = Some(pigdog())
145
146 def optionPigdog(delay: FiniteDuration): Option[String] = {
147 Thread.sleep(delay.toMillis)
148 Some(pigdog())
149 }
```

J2EE Bad Practices: Threads — Low

Package: akka.actor

scala/akka/actor/TypedActorSpec.scala, line 147 (J2EE Bad Practices: Threads) — Low

```
150
```

scala/akka/actor/TypedActorSpec.scala, line 152 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: joptionPigdog()
File: scala/akka/actor/TypedActorSpec.scala:152
Taint Flags:

```
149 }
150
151 def joptionPigdog(delay: FiniteDuration): JOption[String] = {
152 Thread.sleep(delay.toMillis)
153 JOption.some(pigdog())
154 }
155
```

scala/akka/actor/ActorRefSpec.scala, line 55 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: akka$actor$ActorRefSpec$WorkerActor$$work()
File: scala/akka/actor/ActorRefSpec.scala:55
Taint Flags:

```
52 }
53 }
54
55 private def work(): Unit = Thread.sleep(1.second.dilated.toMillis)
56 }
57
58 class SenderActor(replyActor: ActorRef, latch: TestLatch) extends Actor {
```

scala/akka/actor/Bench.scala, line 116 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State

J2EE Bad Practices: Threads

Low

Package: akka.actor

scala/akka/actor/Bench.scala, line 116 (J2EE Bad Practices: Threads)

Low

Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: run()
File: scala/akka/actor/Bench.scala:116
Taint Flags:

```
113 Chameneos.start = System.currentTimeMillis
114 val system = ActorSystem()
115 system.actorOf(Props(new Mall(1000000, 4)))
116 Thread.sleep(10000)
117 println("Elapsed: " + (end - start))
118 system.terminate()
119 }
```

scala/akka/actor/SchedulerSpec.scala, line 683 (J2EE Bad Practices: Threads)

Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: run()
Enclosing Method: execute()
File: scala/akka/actor/SchedulerSpec.scala:683
Taint Flags:

```
680 }
681
682 val localEC = new ExecutionContext {
683 def execute(runnable: Runnable): Unit = { runnable.run() }
684 def reportFailure(t: Throwable): Unit = { t.printStackTrace() }
685 }
686
```

scala/akka/actor/TypedActorSpec.scala, line 135 (J2EE Bad Practices: Threads)

Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: futurePigdog()
File: scala/akka/actor/TypedActorSpec.scala:135

J2EE Bad Practices: Threads — Low

Package: akka.actor

scala/akka/actor/TypedActorSpec.scala, line 135 (J2EE Bad Practices: Threads) — Low

Taint Flags:

```
132  }
133
134  def futurePigdog(delay: FiniteDuration, numbered: Int): Future[String] = {
135  Thread.sleep(delay.toMillis)
136  Future.successful(pigdog() + numbered)
137  }
138
```

scala/akka/actor/TypedActorSpec.scala, line 130 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: futurePigdog()
File: scala/akka/actor/TypedActorSpec.scala:130
Taint Flags:

```
127  def futurePigdog(): Future[String] = Future.successful(pigdog())
128
129  def futurePigdog(delay: FiniteDuration): Future[String] = {
130  Thread.sleep(delay.toMillis)
131  futurePigdog()
132  }
133
```

Package: akka.actor.dispatch

scala/akka/actor/dispatch/ActorModelSpec.scala, line 84 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:84
Taint Flags:

```
81  def receive = {
82  case AwaitLatch(latch) => { ack(); latch.await(); busy.switchOff(()) }
83  case Meet(sign, wait) => { ack(); sign.countDown(); wait.await(); busy.switchOff(()) }
```

J2EE Bad Practices: Threads — Low

Package: akka.actor.dispatch

scala/akka/actor/dispatch/ActorModelSpec.scala, line 84 (J2EE Bad Practices: Threads) — Low

```
84 case Wait(time) => { ack(); Thread.sleep(time); busy.switchOff(()) }
85 case WaitAck(time, l) => { ack(); Thread.sleep(time); l.countDown(); busy.switchOff(()) }
86 case Reply(msg) => { ack(); sender() ! msg; busy.switchOff(()) }
87 case TryReply(msg) => { ack(); sender().tell(msg, null); busy.switchOff(()) }
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 85 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:85
Taint Flags:

```
82 case AwaitLatch(latch) => { ack(); latch.await(); busy.switchOff(()) }
83 case Meet(sign, wait) => { ack(); sign.countDown(); wait.await(); busy.switchOff(()) }
84 case Wait(time) => { ack(); Thread.sleep(time); busy.switchOff(()) }
85 case WaitAck(time, l) => { ack(); Thread.sleep(time); l.countDown(); busy.switchOff(()) }
86 case Reply(msg) => { ack(); sender() ! msg; busy.switchOff(()) }
87 case TryReply(msg) => { ack(); sender().tell(msg, null); busy.switchOff(()) }
88 case Forward(to, msg) => { ack(); to.forward(msg); busy.switchOff(()) }
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 398 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: start()
Enclosing Method: flood()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:398
Taint Flags:

```
395 Future {
396 keepAliveLatch.await(waitTime, TimeUnit.MILLISECONDS)
397 }(dispatcher)
398 }).start()
399 boss ! "run"
400 assertCountDown(cachedMessage.latch, waitTime, "Counting down from " + num)
401 assertCountDown(stopLatch, waitTime, "Expected all children to stop")
```

J2EE Bad Practices: Threads Low

Package: akka.actor.dispatch

scala/akka/actor/dispatch/BalancingDispatcherSpec.scala, line 31 (J2EE Bad Practices: Threads) Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/BalancingDispatcherSpec.scala:31
Taint Flags:

```
28
29  def receive = {
30  case _: Int => {
31  Thread.sleep(delay)
32  invocationCount += 1
33  finishedCounter.countDown()
34  }
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 393 (J2EE Bad Practices: Threads) Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: run()
Enclosing Method: flood()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:393
Taint Flags:

```
390  // this future is meant to keep the dispatcher alive until the end of the test run even if
391  // the boss doesn't create children fast enough to keep the dispatcher from becoming empty
392  // and it needs to be on a separate thread to not deadlock the calling thread dispatcher
393  new Thread(new Runnable {
394  def run() =
395  Future {
396  keepAliveLatch.await(waitTime, TimeUnit.MILLISECONDS)
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 347 (J2EE Bad Practices: Threads) Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

J2EE Bad Practices: Threads — Low

Package: akka.actor.dispatch

scala/akka/actor/dispatch/ActorModelSpec.scala, line 347 (J2EE Bad Practices: Threads) — Low

Sink: start()
Enclosing Method: spawn()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:347
Taint Flags:

```
344 catch {
345 case e: Throwable => system.eventStream.publish(Error(e, "spawn", this.getClass, "error in spawned thread"))
346 }
347 }).start()
348 }
349
350 "not process messages for a suspended actor" in {
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 248 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: await()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:248
Taint Flags:

```
245 var done = false
246 try {
247 done = condition
248 if (!done) Thread.sleep(25)
249 } catch {
250 case _: InterruptedException =>
251 }
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 97 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: interrupt()
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:97
Taint Flags:

```
94 ack(); sender() ! Status.Failure(new ActorInterruptedException(new InterruptedException("Ping!")));
95 busy.switchOff(()); throw new InterruptedException("Ping!")
```

J2EE Bad Practices: Threads — Low

Package: akka.actor.dispatch

scala/akka/actor/dispatch/ActorModelSpec.scala, line 97 (J2EE Bad Practices: Threads) — Low

```
96  }
97  case InterruptNicely(msg) => { ack(); sender() ! msg; busy.switchOff(()); Thread.currentThread().interrupt() }
98  case ThrowException(e: Throwable) => { ack(); busy.switchOff(()); throw e }
99  case DoubleStop => { ack(); context.stop(self); context.stop(self); busy.switchOff }
100  }
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 341 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: Thread()
Enclosing Method: ActorModelSpec$$anon$1()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:341
Taint Flags:

```
338  }
339
340  def spawn(f: => Unit): Unit = {
341  (new Thread {
342  override def run(): Unit =
343  try f
344  catch {
```

scala/akka/actor/dispatch/DispatcherActorsSpec.scala, line 20 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: applyOrElse()
File: scala/akka/actor/dispatch/DispatcherActorsSpec.scala:20
Taint Flags:

```
17
18  def receive = {
19  case _: Int => {
20  Thread.sleep(50) // slow actor
21  finishedCounter.countDown()
22  }
23  }
```

J2EE Bad Practices: Threads — Low

Package: akka.dispatch

scala/akka/dispatch/ExecutionContextSpec.scala, line 262 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: run()
Enclosing Method: execute()
File: scala/akka/dispatch/ExecutionContextSpec.scala:262
Taint Flags:

```
259 val submissions = new AtomicInteger(0)
260 val counter = new AtomicInteger(0)
261 val underlying = new ExecutionContext {
262 override def execute(r: Runnable): Unit = { submissions.incrementAndGet(); ExecutionContext.global.execute(r) }
263 override def reportFailure(t: Throwable): Unit = { ExecutionContext.global.reportFailure(t) }
264 }
265 val throughput = 25
```

scala/akka/dispatch/ExecutionContextSpec.scala, line 186 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: run()
Enclosing Method: run()
File: scala/akka/dispatch/ExecutionContextSpec.scala:186
Taint Flags:

```
183 ec.execute(new RunBatch {
184 override def run = {
185 // enqueue a task to the batch
186 ec.execute(new RunBatch {
187 override def run = blocking {
188 x = 1
189 }
```

scala/akka/dispatch/ExecutionContextSpec.scala, line 226 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

J2EE Bad Practices: Threads — Low

Package: akka.dispatch

scala/akka/dispatch/ExecutionContextSpec.scala, line 226 (J2EE Bad Practices: Threads) — Low

Sink: run()
Enclosing Method: execute()
File: scala/akka/dispatch/ExecutionContextSpec.scala:226
Taint Flags:

```
223 val submissions = new AtomicInteger(0)
224 val counter = new AtomicInteger(0)
225 val underlying = new ExecutionContext {
226 override def execute(r: Runnable): Unit = { submissions.incrementAndGet(); ExecutionContext.global.execute(r) }
227 override def reportFailure(t: Throwable): Unit = { ExecutionContext.global.reportFailure(t) }
228 }
229 val throughput = 25
```

Package: akka.event

scala/akka/event/LoggerSpec.scala, line 114 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: aroundReceive()
File: scala/akka/event/LoggerSpec.scala:114
Taint Flags:

```
111 msg match {
112 case event: LogEvent =>
113 if (event.message.toString.startsWith("msg1"))
114 Thread.sleep(500) // slow
115 super.aroundReceive(r, msg)
116 case _ => super.aroundReceive(r, msg)
117 }
```

Package: akka.io.dns.internal

scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala, line 219 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: AsyncDnsResolverSpec$$anon$14()

J2EE Bad Practices: Threads — Low

Package: akka.io.dns.internal

scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala, line 219 (J2EE Bad Practices: Threads) — Low

File: scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala:219
Taint Flags:

```
216
217  senderProbe.expectMsg(Resolved("cats.com", im.Seq(ipv4Record)))
218
219  Thread.sleep(200)
220  r ! Resolve("cats.com", Ip(ipv4 = true, ipv6 = false))
221  dnsClient1.expectMsg(Question4(2, "cats.com"))
222  dnsClient1.reply(Answer(2, im.Seq(ipv4Record)))
```

Package: akka.routing

scala/akka/routing/ScatterGatherFirstCompletedSpec.scala, line 35 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: applyOrElse()
File: scala/akka/routing/ScatterGatherFirstCompletedSpec.scala:35
Taint Flags:

```
32  case Stop(Some(_id)) if (_id == id) => context.stop(self)
33  case _id: Int if (_id == id) =>
34  case _ => {
35  Thread.sleep(100 * id)
36  sender() ! id
37  }
38  }
```

scala/akka/routing/TailChoppingSpec.scala, line 27 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: applyOrElse()
File: scala/akka/routing/TailChoppingSpec.scala:27
Taint Flags:

J2EE Bad Practices: Threads — Low

Package: akka.routing

scala/akka/routing/TailChoppingSpec.scala, line 27 (J2EE Bad Practices: Threads) — Low

```
24  case "times" => sender() ! times
25  case _ =>
26  times += 1
27  Thread.sleep(sleepTime.toMillis)
28  sender() ! "ack"
29  }
30  }), "Actor:" + id)
```

scala/akka/routing/ResizerSpec.scala, line 219 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: applyOrElse()
File: scala/akka/routing/ResizerSpec.scala:219
Taint Flags:

```
216  val router = system.actorOf(RoundRobinPool(nrOfInstances = 0, resizer = Some(resizer)).props(Props(new Actor {
217  def receive = {
218  case n: Int if n <= 0 => // done
219  case n: Int => Thread.sleep((n millis).dilated.toMillis)
220  }
221  })))
222
```

scala/akka/routing/ResizerSpec.scala, line 175 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: applyOrElse()
File: scala/akka/routing/ResizerSpec.scala:175
Taint Flags:

```
172  val router = system.actorOf(RoundRobinPool(nrOfInstances = 0, resizer = Some(resizer)).props(Props(new Actor {
173  def receive = {
174  case d: FiniteDuration =>
175  Thread.sleep(d.dilated.toMillis); sender() ! "done"
176  case "echo" => sender() ! "reply"
177  }
```

J2EE Bad Practices: Threads — Low

Package: akka.routing

scala/akka/routing/ResizerSpec.scala, line 175 (J2EE Bad Practices: Threads) — Low

```
178 })))
```

Package: akka.util

scala/akka/util/BoundedBlockingQueueSpec.scala, line 819 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: interrupt()
Enclosing Method: advanceTime()
File: scala/akka/util/BoundedBlockingQueueSpec.scala:819
Taint Flags:

```
816 case Some(manual) =>
817 val newWaitTime = manual.waitTime - timespan.toNanos
818 waiting = if (newWaitTime <= 0 && manual.waitingThread.isDefined) {
819 manual.waitingThread.get.interrupt()
820 Some(Manual(newWaitTime, None))
821 } else {
822 Some(manual.copy(waitTime = newWaitTime))
```

scala/akka/util/BoundedBlockingQueueSpec.scala, line 900 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: Thread()
Enclosing Method: DefaultExecutionContext$$anon$1$$anon$2()
File: scala/akka/util/BoundedBlockingQueueSpec.scala:900
Taint Flags:

```
897 object DefaultExecutionContext {
898 implicit val ec: ExecutionContextExecutor = ExecutionContext.fromExecutor(new Executor {
899 override def execute(command: Runnable): Unit =
900 new Thread() {
901 override def run(): Unit = command.run()
902 }.start()
903 })
```

scala/akka/util/BoundedBlockingQueueSpec.scala, line 901 (J2EE Bad Practices: Threads) — Low

Issue Details

J2EE Bad Practices: Threads — Low

Package: akka.util

scala/akka/util/BoundedBlockingQueueSpec.scala, line 901 (J2EE Bad Practices: Threads) Low

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: run()
Enclosing Method: run()
File: scala/akka/util/BoundedBlockingQueueSpec.scala:901
Taint Flags:

```
898  implicit val ec: ExecutionContextExecutor = ExecutionContext.fromExecutor(new Executor {
899  override def execute(command: Runnable): Unit =
900  new Thread() {
901  override def run(): Unit = command.run()
902  }.start()
903  })
904 }
```

scala/akka/util/BoundedBlockingQueueSpec.scala, line 902 (J2EE Bad Practices: Threads) Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: start()
Enclosing Method: execute()
File: scala/akka/util/BoundedBlockingQueueSpec.scala:902
Taint Flags:

```
899  override def execute(command: Runnable): Unit =
900  new Thread() {
901  override def run(): Unit = command.run()
902  }.start()
903  })
904 }
905
```

scala/akka/util/SwitchSpec.scala, line 90 (J2EE Bad Practices: Threads) Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: Thread()
Enclosing Method: SwitchSpec$$anon$1()

J2EE Bad Practices: Threads — Low

Package: akka.util

scala/akka/util/SwitchSpec.scala, line 90 (J2EE Bad Practices: Threads) — Low

File: scala/akka/util/SwitchSpec.scala:90
Taint Flags:

```
87  }
88
89  val latch = new CountDownLatch(1)
90  new Thread {
91  override def run(): Unit = {
92  s.switchOff(())
93  latch.countDown()
```

Package: scala.akka.actor

scala/akka/actor/SchedulerSpec.scala, line 479 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/SchedulerSpec.scala:479
Taint Flags:

```
476  while (Try(sched.scheduleOnce(Duration.Zero, new Scheduler.TaskRunOnClose {
477  override def run(): Unit = ()
478  })(localEC)).isSuccess) {
479  Thread.sleep(1)
480  driver.wakeUp(step)
481  rounds += 1
482  }
```

scala/akka/actor/SchedulerSpec.scala, line 404 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: run()
File: scala/akka/actor/SchedulerSpec.scala:404
Taint Flags:

```
401  val latch = new TestLatch(n)
402  val startTime = System.nanoTime
```

J2EE Bad Practices: Threads — Low

Package: scala.akka.actor

scala/akka/actor/SchedulerSpec.scala, line 404 (J2EE Bad Practices: Threads) — Low

```
403 system.scheduler.scheduleWithFixedDelay(125.millis, 125.millis) { () =>
404 Thread.sleep(100)
405 latch.countDown()
406 }
407 Await.ready(latch, 6.seconds)
```

scala/akka/actor/SchedulerSpec.scala, line 388 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: run()
File: scala/akka/actor/SchedulerSpec.scala:388
Taint Flags:

```
385 val latch = new TestLatch(n)
386 val startTime = System.nanoTime
387 system.scheduler.scheduleAtFixedRate(225.millis, 225.millis) { () =>
388 Thread.sleep(100)
389 latch.countDown()
390 }
391 Await.ready(latch, 6.seconds)
```

scala/akka/actor/ActorSystemSpec.scala, line 289 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/ActorSystemSpec.scala:289
Taint Flags:

```
286 val t = system.actorOf(Props[ActorSystemSpec.Terminater]())
287 failing should not be true // because once failing => always failing (it’s due to shutdown)
288 created :+= t
289 if (created.size % 1000 == 0) Thread.sleep(50) // in case of unfair thread scheduling
290 } catch {
291 case _: IllegalStateException => failing = true
292 }
```

J2EE Bad Practices: Threads — Low

Package: scala.akka.actor

scala/akka/actor/ActorLifeCycleSpec.scala, line 169 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/ActorLifeCycleSpec.scala:169
Taint Flags:

```
166
167  Future {
168  latch.await()
169  Thread.sleep(50)
170  "po"
171  }
172  // Here, we implicitly close over the actor instance and access the context
```

scala/akka/actor/SchedulerSpec.scala, line 307 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/SchedulerSpec.scala:307
Taint Flags:

```
304  val timeout = collectCancellable(scheduleAdapter.schedule(initialDelay, delay, () => {
305  ticks.incrementAndGet()
306  }))
307  Thread.sleep((initialDelay + 200.millis.dilated).toMillis)
308  timeout.cancel()
309  Thread.sleep((delay + 100.millis.dilated).toMillis)
310
```

scala/akka/actor/SchedulerSpec.scala, line 294 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

J2EE Bad Practices: Threads — Low

Package: scala.akka.actor

scala/akka/actor/SchedulerSpec.scala, line 294 (J2EE Bad Practices: Threads) — Low

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/SchedulerSpec.scala:294
Taint Flags:

```
291 }))
292 Thread.sleep(10.millis.dilated.toMillis)
293 timeout.cancel()
294 Thread.sleep((initialDelay + 100.millis.dilated).toMillis)
295
296 ticks.get should ===(0)
297 }
```

scala/akka/actor/CoordinatedShutdownSpec.scala, line 404 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/CoordinatedShutdownSpec.scala:404
Taint Flags:

```
401 co.addTask("a", "a2") { () =>
402 Future {
403 // to verify that b is not performed before a also in case of failure
404 Thread.sleep(100)
405 testActor ! "A"
406 Done
407 }
```

scala/akka/actor/CoordinatedShutdownSpec.scala, line 145 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/CoordinatedShutdownSpec.scala:145
Taint Flags:

```
142 co.addTask("b", "b2") { () =>
143 Future {
```

J2EE Bad Practices: Threads — Low

Package: scala.akka.actor

scala/akka/actor/CoordinatedShutdownSpec.scala, line 145 (J2EE Bad Practices: Threads) — Low

```
144 // to verify that c is not performed before b
145 Thread.sleep(100)
146 testActor ! "B"
147 Done
148 }
```

scala/akka/actor/RestartStrategySpec.scala, line 207 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/RestartStrategySpec.scala:207
Taint Flags:

```
204 Await.ready(secondRestartLatch, 10 seconds)
205 Await.ready(countDownLatch, 10 seconds)
206
207 sleep(700L)
208
209 employee ! Crash
210 Await.ready(stopLatch, 10 seconds)
```

scala/akka/actor/RestartStrategySpec.scala, line 153 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/RestartStrategySpec.scala:153
Taint Flags:

```
150 Await.ready(secondPingLatch, 10 seconds)
151
152 // sleep to go out of the restart strategy's time range
153 sleep(700L)
154
155 // now crash again... should and post restart ping
156 employee ! Crash
```

J2EE Bad Practices: Threads — Low

Package: scala.akka.actor

scala/akka/actor/SchedulerSpec.scala, line 309 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/SchedulerSpec.scala:309
Taint Flags:

```
306 }))
307 Thread.sleep((initialDelay + 200.millis.dilated).toMillis)
308 timeout.cancel()
309 Thread.sleep((delay + 100.millis.dilated).toMillis)
310
311 ticks.get should ===(1)
312 }
```

scala/akka/actor/SchedulerSpec.scala, line 612 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/SchedulerSpec.scala:612
Taint Flags:

```
609 withScheduler() { (sched, driver) =>
610 import system.dispatcher
611 val counter = new AtomicInteger
612 Future { Thread.sleep(5); driver.close(); sched.close() }
613 val headroom = 200
614 var overrun = headroom
615 val cap = 1000000
```

scala/akka/actor/SchedulerSpec.scala, line 292 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

J2EE Bad Practices: Threads — Low

Package: scala.akka.actor

scala/akka/actor/SchedulerSpec.scala, line 292 (J2EE Bad Practices: Threads) — Low

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/SchedulerSpec.scala:292
Taint Flags:

```
289 val timeout = collectCancellable(scheduleAdapter.schedule(initialDelay, delay, () => {
290 ticks.incrementAndGet()
291 }))
292 Thread.sleep(10.millis.dilated.toMillis)
293 timeout.cancel()
294 Thread.sleep((initialDelay + 100.millis.dilated).toMillis)
295
```

scala/akka/actor/ActorSystemSpec.scala, line 218 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/ActorSystemSpec.scala:218
Taint Flags:

```
215 var callbackWasRun = false
216
217 system2.registerOnTermination {
218 Thread.sleep(50.millis.dilated.toMillis)
219 callbackWasRun = true
220 }
221 import system.dispatcher
```

scala/akka/actor/ActorSystemSpec.scala, line 198 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/ActorSystemSpec.scala:198
Taint Flags:

```
195
196 for (i <- 1 to count) {
```

J2EE Bad Practices: Threads — Low

Package: scala.akka.actor

scala/akka/actor/ActorSystemSpec.scala, line 198 (J2EE Bad Practices: Threads) — Low

```
197 system2.registerOnTermination {
198 Thread.sleep((i % 3).millis.dilated.toMillis)
199 result.add(i)
200 latch.countDown()
201 }
```

scala/akka/actor/SchedulerSpec.scala, line 448 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/SchedulerSpec.scala:448
Taint Flags:

```
445 }
446 }
447 // get somewhat into the middle of things
448 Thread.sleep(500)
449 val cancellations = for (t <- tasks) yield {
450 t.cancel()
451 if (t.isCancelled) 1 else 0
```

scala/akka/actor/RestartStrategySpec.scala, line 266 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/RestartStrategySpec.scala:266
Taint Flags:

```
263 Await.ready(stopLatch, 10 seconds)
264
265 Await.ready(maxNoOfRestartsLatch, 10 seconds)
266 sleep(500L)
267 assert(employee.isTerminated)
268 }
269 }
```

J2EE Bad Practices: Threads — Low

Package: scala.akka.actor

scala/akka/actor/RestartStrategySpec.scala, line 211 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/RestartStrategySpec.scala:211
Taint Flags:

```
208
209  employee ! Crash
210  Await.ready(stopLatch, 10 seconds)
211  sleep(500L)
212  assert(employee.isTerminated)
213  }
214
```

scala/akka/actor/DeadLetterSuspensionSpec.scala, line 79 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/DeadLetterSuspensionSpec.scala:79
Taint Flags:

```
76  droppingActor ! 6
77
78  // let suspend-duration elapse
79  Thread.sleep(2050)
80
81  // re-enabled
82  EventFilter
```

scala/akka/actor/SchedulerSpec.scala, line 143 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

J2EE Bad Practices: Threads — Low

Package: scala.akka.actor

scala/akka/actor/SchedulerSpec.scala, line 143 (J2EE Bad Practices: Threads) — Low

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/SchedulerSpec.scala:143
Taint Flags:

```
140
141  (1 to 300).foreach { _ =>
142  collectCancellable(system.scheduler.scheduleOnce(20.millis, actor, Msg(System.nanoTime)))
143  Thread.sleep(5)
144  }
145
146  Await.ready(ticks, 3 seconds)
```

scala/akka/actor/FSMTimingSpec.scala, line 110 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/FSMTimingSpec.scala:110
Taint Flags:

```
107  fsm ! TestCancelStateTimerInNamedTimerMessage
108  fsm ! Tick
109  expectMsg(500 millis, Tick)
110  Thread.sleep(200) // this is ugly: need to wait for StateTimeout to be queued
111  resume(fsm)
112  expectMsg(
113  500 millis,
```

scala/akka/actor/DeadLetterSupressionSpec.scala, line 84 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/DeadLetterSupressionSpec.scala:84
Taint Flags:

```
81  allListener.expectMsg(200.millis, SuppressedDeadLetter(SuppressedMsg, testActor, system.deadLetters))
82  allListener.expectMsg(200.millis, DeadLetter(NormalMsg, testActor, system.deadLetters))
```

J2EE Bad Practices: Threads — Low

Package: scala.akka.actor

scala/akka/actor/DeadLetterSupressionSpec.scala, line 84 (J2EE Bad Practices: Threads) — Low

```
83
84 Thread.sleep(200)
85 deadListener.expectNoMessage(Duration.Zero)
86 suppressedListener.expectNoMessage(Duration.Zero)
87 allListener.expectNoMessage(Duration.Zero)
```

Package: scala.akka.actor.dispatch

scala/akka/actor/dispatch/ActorModelSpec.scala, line 444 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:444
Taint Flags:

```
441 assert(Await.result(f6, timeout.duration) === "bar2")
442 assert(intercept[ActorInterruptedException](Await.result(f5, timeout.duration)).getCause.getMessage === "Ping!")
443 c.cancel()
444 Thread.sleep(300) // give the EventFilters a chance of catching all messages
445 }
446 }
447
```

scala/akka/actor/dispatch/BalancingDispatcherSpec.scala, line 76 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/dispatch/BalancingDispatcherSpec.scala:76
Taint Flags:

```
73
74 // now send some messages to actors to keep the dispatcher dispatching messages
75 for (i <- 1 to 10) {
76 Thread.sleep(150)
77 if (i % 2 == 0) {
78 fast ! i
```

J2EE Bad Practices: Threads — Low

Package: scala.akka.actor.dispatch

scala/akka/actor/dispatch/BalancingDispatcherSpec.scala, line 76 (J2EE Bad Practices: Threads) — Low

```
79  sentToFast += 1
```

scala/akka/actor/dispatch/DispatcherActorSpec.scala, line 130 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/actor/dispatch/DispatcherActorSpec.scala:130
Taint Flags:

```
127  slowOne ! "ping"
128  fastOne ! "ping"
129  assert(ready.await(2, TimeUnit.SECONDS) === true)
130  Thread.sleep(deadline.toMillis + 10) // wait just a bit more than the deadline
131  start.countDown()
132  assert(latch.await(2, TimeUnit.SECONDS) === true)
133  }
```

Package: scala.akka.dispatch

scala/akka/dispatch/ExecutionContextSpec.scala, line 65 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/dispatch/ExecutionContextSpec.scala:65
Taint Flags:

```
62  if (callingThreadLock.get != 0) p.tryFailure(new IllegalStateException("Batch was executed inline!"))
63  else if (count.incrementAndGet == 100) p.trySuccess(()) //Done
64  else if (lock.compareAndSet(0, 1)) {
65  try Thread.sleep(10)
66  finally lock.compareAndSet(1, 0)
67  } else p.tryFailure(new IllegalStateException("Executed batch in parallel!"))
68  }
```

J2EE Bad Practices: Threads — Low

Package: scala.akka.dispatch

scala/akka/dispatch/ExecutionContextSpec.scala, line 212 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/dispatch/ExecutionContextSpec.scala:212
Taint Flags:

```
209 perform(_ + 4)
210 perform(_ * 2)
211 sec.size() should ===(2)
212 Thread.sleep(500)
213 sec.size() should ===(2)
214 counter.get should ===(2)
215 sec.resume()
```

scala/akka/dispatch/ExecutionContextSpec.scala, line 115 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/dispatch/ExecutionContextSpec.scala:115
Taint Flags:

```
112 // trigger the resubmitUnbatched() call
113 blocking { () }
114 // make sure that the other task runs to completion before continuing
115 Thread.sleep(500)
116 // now try again to blockOn()
117 blocking { () }
118 }
```

scala/akka/dispatch/ExecutionContextSpec.scala, line 183 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: run()

J2EE Bad Practices: Threads — Low

Package: scala.akka.dispatch

scala/akka/dispatch/ExecutionContextSpec.scala, line 183 (J2EE Bad Practices: Threads) — Low

Enclosing Method: apply()
File: scala/akka/dispatch/ExecutionContextSpec.scala:183
Taint Flags:

```
180 }
181 val ec = system.dispatchers.lookup(CallingThreadDispatcher.Id)
182 var x = 0
183 ec.execute(new RunBatch {
184 override def run = {
185 // enqueue a task to the batch
186 ec.execute(new RunBatch {
```

Package: scala.akka.io

scala/akka/io/TcpConnectionSpec.scala, line 631 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/io/TcpConnectionSpec.scala:631
Taint Flags:

```
628
629 key.isConnectable should ===(true)
630 connectionActor.toString // force the lazy val
631 Thread.sleep(300)
632 selector.send(connectionActor, ChannelConnectable)
633 userHandler.expectMsg(CommandFailed(Connect(UnboundAddress)))
634
```

scala/akka/io/UdpConnectedIntegrationSpec.scala, line 145 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/io/UdpConnectedIntegrationSpec.scala:145
Taint Flags:

```
142
```

J2EE Bad Practices: Threads — Low

Package: scala.akka.io

scala/akka/io/UdpConnectedIntegrationSpec.scala, line 145 (J2EE Bad Practices: Threads) Low

```
143 server ! Udp.Unbind
144 expectMsg(Udp.Unbound)
145 Thread.sleep(1000) // if it stops that takes a bit of time, give it that time
146
147 // bug was that the commander would fail on next read/write
148 clientCommander ! UdpConnected.Send(ByteString("data to trigger fail"), 1)
```

Package: scala.akka.pattern

scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala, line 234 (J2EE Bad Practices: Threads) Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/pattern/BackoffOnRestartSupervisorSpec.scala:234
Taint Flags:

```
231 probe.expectMsg("STARTED")
232 }
233 // Now wait the length of our window, and throw again. We should still restart.
234 Thread.sleep(1000)
235 supervisor ! "THROW"
236 probe.expectMsg("STARTED")
237 // Now we'll issue three more requests, and should be terminated.
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 698 (J2EE Bad Practices: Threads) Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/pattern/CircuitBreakerSpec.scala:698
Taint Flags:

```
695 "invoke onCallTimeout if call timeouts" taggedAs TimingTest in {
696 val breaker = shortCallTimeoutCb()
697
698 breaker().withCircuitBreaker(Future(Thread.sleep(250.millis.dilated.toMillis)))
699 checkLatch(breaker.callTimeoutLatch)
```

J2EE Bad Practices: Threads — Low

Package: scala.akka.pattern

scala/akka/pattern/CircuitBreakerSpec.scala, line 698 (J2EE Bad Practices: Threads) — Low

```
700
701  val timeout = breaker.probe.expectMsgType[CBTimeout]
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 672 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/pattern/CircuitBreakerSpec.scala:672
Taint Flags:

```
669  val breaker = shortCallTimeoutCb()
670
671  val fut = breaker().withCircuitBreaker(Future {
672  Thread.sleep(150.millis.dilated.toMillis)
673  throwException
674  })
675  checkLatch(breaker.openLatch)
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 593 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/pattern/CircuitBreakerSpec.scala:593
Taint Flags:

```
590  breaker().withCircuitBreaker(Future(throwException))
591  checkLatch(breaker.halfOpenLatch)
592
593  breaker().withCircuitBreaker(Future(Thread.sleep(250.millis.dilated.toMillis)))
594  breaker().withCircuitBreaker(Future(sayHi))
595  checkLatch(breaker.callBreakerOpenLatch)
596  }
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 579 (J2EE Bad Practices: Threads) — Low

Issue Details

J2EE Bad Practices: Threads — Low

Package: scala.akka.pattern

scala/akka/pattern/CircuitBreakerSpec.scala, line 579 (J2EE Bad Practices: Threads) — Low

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/pattern/CircuitBreakerSpec.scala:579
Taint Flags:

```
576  breaker().withCircuitBreaker(Future(throwException))
577  checkLatch(breaker.halfOpenLatch)
578
579  breaker().withCircuitBreaker(Future(Thread.sleep(200.millis.dilated.toMillis)))
580  checkLatch(breaker.callTimeoutLatch)
581
582  breaker.probe.expectMsgType[CBFailure]
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 450 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/pattern/CircuitBreakerSpec.scala:450
Taint Flags:

```
447  val e1 = intercept[CircuitBreakerOpenException] { breaker().withSyncCircuitBreaker(sayHi) }
448  val shortRemainingDuration = e1.remainingDuration
449
450  Thread.sleep(1000.millis.dilated.toMillis)
451  checkLatch(breaker.halfOpenLatch)
452
453  // transit to open again
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 401 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()

J2EE Bad Practices: Threads — Low

Package: scala.akka.pattern

scala/akka/pattern/CircuitBreakerSpec.scala, line 401 (J2EE Bad Practices: Threads) — Low

File: scala/akka/pattern/CircuitBreakerSpec.scala:401
Taint Flags:

```
398 "invoke onCallTimeout if call timeouts" taggedAs TimingTest in {
399 val breaker = shortCallTimeoutCb()
400
401 intercept[TimeoutException](breaker().withSyncCircuitBreaker(Thread.sleep(250.millis.dilated.toMillis)))
402 checkLatch(breaker.callTimeoutLatch)
403
404 val timeout = breaker.probe.expectMsgType[CBTimeout]
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 370 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/pattern/CircuitBreakerSpec.scala:370
Taint Flags:

```
367 val breaker = shortCallTimeoutCb()
368 intercept[TimeoutException] {
369 breaker().withSyncCircuitBreaker {
370 Thread.sleep(200.millis.dilated.toMillis)
371 }
372 }
373 breaker().currentFailureCount should ===(1)
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 267 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/pattern/CircuitBreakerSpec.scala:267
Taint Flags:

```
264 intercept[TestException] { breaker().withSyncCircuitBreaker(throwException) }
265 checkLatch(breaker.halfOpenLatch)
266
267 breaker().withCircuitBreaker(Future(Thread.sleep(250.millis.dilated.toMillis)))
```

J2EE Bad Practices: Threads — Low

Package: scala.akka.pattern

scala/akka/pattern/CircuitBreakerSpec.scala, line 267 (J2EE Bad Practices: Threads) — Low

```
268 intercept[CircuitBreakerOpenException] { breaker().withSyncCircuitBreaker(sayHi) }
269
270 checkLatch(breaker.callBreakerOpenLatch)
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 253 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/pattern/CircuitBreakerSpec.scala:253
Taint Flags:

```
250 intercept[TestException] { breaker().withSyncCircuitBreaker(throwException) }
251 checkLatch(breaker.halfOpenLatch)
252
253 intercept[TimeoutException] { breaker().withSyncCircuitBreaker(Thread.sleep(200.millis.dilated.toMillis)) }
254 checkLatch(breaker.callTimeoutLatch)
255
256 breaker.probe.expectMsgType[CBFailure]
```

scala/akka/pattern/CircuitBreakerSpec.scala, line 380 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/pattern/CircuitBreakerSpec.scala:380
Taint Flags:

```
377 val breaker = shortCallTimeoutCb()
378 Future {
379 breaker().withSyncCircuitBreaker {
380 Thread.sleep(1.second.dilated.toMillis)
381 }
382 }
383 within(900.millis) {
```

scala/akka/pattern/CircuitBreakerStressSpec.scala, line 77 (J2EE Bad Practices: Threads) — Low

Issue Details

J2EE Bad Practices: Threads | Low

Package: scala.akka.pattern

scala/akka/pattern/CircuitBreakerStressSpec.scala, line 77 (J2EE Bad Practices: Threads) | Low

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/pattern/CircuitBreakerStressSpec.scala:77
Taint Flags:

```
74 a ! JobDone
75 }
76 // let them work for a while
77 Thread.sleep(3000)
78 stressActors.foreach { a =>
79 a ! GetResult
80 val result = expectMsgType[Result]
```

Package: scala.akka.routing

scala/akka/routing/ResizerSpec.scala, line 226 (J2EE Bad Practices: Threads) | Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/routing/ResizerSpec.scala:226
Taint Flags:

```
223 // put some pressure on the router
224 for (_ <- 0 until 15) {
225 router ! 150
226 Thread.sleep((20 millis).dilated.toMillis)
227 }
228
229 val z = routeeSize(router)
```

scala/akka/routing/ResizerSpec.scala, line 190 (J2EE Bad Practices: Threads) | Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

J2EE Bad Practices: Threads — Low

Package: scala.akka.routing

scala/akka/routing/ResizerSpec.scala, line 190 (J2EE Bad Practices: Threads) — Low

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/routing/ResizerSpec.scala:190
Taint Flags:

```
187 for (_ <- 0 until loops) {
188 router ! d
189 // sending in too quickly will result in skipped resize due to many resizeInProgress conflicts
190 Thread.sleep(20.millis.dilated.toMillis)
191 }
192 within((d * loops / resizer.lowerBound) + 2.seconds.dilated) {
193 for (_ <- 0 until loops) expectMsg("done")
```

scala/akka/routing/MetricsBasedResizerSpec.scala, line 274 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/routing/MetricsBasedResizerSpec.scala:274
Taint Flags:

```
271 msgs1.foreach(_.second.open()) //process two messages
272
273 // make sure some time passes in-between
274 Thread.sleep(300)
275
276 // wait for routees to update their mail boxes
277 msgs2.foreach(l => Await.ready(l.first, timeout.duration))
```

scala/akka/routing/MetricsBasedResizerSpec.scala, line 243 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/routing/MetricsBasedResizerSpec.scala:243
Taint Flags:

J2EE Bad Practices: Threads — Low

Package: scala.akka.routing

scala/akka/routing/MetricsBasedResizerSpec.scala, line 243 (J2EE Bad Practices: Threads) — Low

```
240 msgs1.foreach(_.second.open()) //process two messages
241
242 // make sure some time passes in-between
243 Thread.sleep(300)
244
245 // wait for routees to update their mail boxes
246 msgs2.foreach(l => Await.ready(l.first, timeout.duration))
```

scala/akka/routing/ResizerSpec.scala, line 237 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/routing/ResizerSpec.scala:237
Taint Flags:

```
234 // let it cool down
235 awaitCond({
236 router ! 0 // trigger resize
237 Thread.sleep((20 millis).dilated.toMillis)
238 routeeSize(router) < z
239 }, interval = 500.millis.dilated)
240
```

scala/akka/routing/ResizerSpec.scala, line 232 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/routing/ResizerSpec.scala:232
Taint Flags:

```
229 val z = routeeSize(router)
230 z should be > (2)
231
232 Thread.sleep((300 millis).dilated.toMillis)
233
```

J2EE Bad Practices: Threads — Low

Package: scala.akka.routing

scala/akka/routing/ResizerSpec.scala, line 232 (J2EE Bad Practices: Threads) — Low

```
234 // let it cool down
235 awaitCond({
```

Package: scala.akka.util

scala/akka/util/BoundedBlockingQueueSpec.scala, line 446 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/util/BoundedBlockingQueueSpec.scala:446
Taint Flags:

```
443 notEmpty.advanceTime(99.milliseconds)
444 latch.await(3, TimeUnit.SECONDS)
445 // queue.poll() must happen first
446 Thread.sleep(50) // this is why this test is tagged as TimingTest
447 f.isCompleted should be(false)
448 queue.put("Hello")
449
```

scala/akka/util/BoundedBlockingQueueSpec.scala, line 411 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/util/BoundedBlockingQueueSpec.scala:411
Taint Flags:

```
408
409 latch.await(3, TimeUnit.SECONDS)
410 // queue.poll() must happen first
411 Thread.sleep(50) // this is why this test is tagged as TimingTest
412 f.isCompleted should be(false)
413 notEmpty.advanceTime(99.milliseconds)
414
```

J2EE Bad Practices: Threads Low

Package: scala.akka.util

scala/akka/util/BoundedBlockingQueueSpec.scala, line 343 (J2EE Bad Practices: Threads) Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/util/BoundedBlockingQueueSpec.scala:343
Taint Flags:

```
340 // Cause `notFull` signal, but don't fill the queue
341 latch.await(3, TimeUnit.SECONDS)
342 // queue.offer() must happen first
343 Thread.sleep(50) // this is why this test is tagged as TimingTest
344 f.isCompleted should be(false)
345 lock.lockInterruptibly()
346 notFull.signal()
```

scala/akka/util/BoundedBlockingQueueSpec.scala, line 321 (J2EE Bad Practices: Threads) Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/util/BoundedBlockingQueueSpec.scala:321
Taint Flags:

```
318
319 latch.await(3, TimeUnit.SECONDS)
320 // queue.offer() must happen first
321 Thread.sleep(50) // this is why this test is tagged as TimingTest
322 f.isCompleted should be(false)
323 queue.take()
324
```

scala/akka/util/BoundedBlockingQueueSpec.scala, line 290 (J2EE Bad Practices: Threads) Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()

J2EE Bad Practices: Threads Low

Package: scala.akka.util

scala/akka/util/BoundedBlockingQueueSpec.scala, line 290 (J2EE Bad Practices: Threads) Low

Enclosing Method: apply()
File: scala/akka/util/BoundedBlockingQueueSpec.scala:290
Taint Flags:

```
287
288 latch.await(3, TimeUnit.SECONDS)
289 // queue.offer() must happen first
290 Thread.sleep(50) // this is why this test is tagged as TimingTest
291 f.isCompleted should be(false)
292 notFull.advanceTime(99.milliseconds)
293
```

scala/akka/util/BoundedBlockingQueueSpec.scala, line 221 (J2EE Bad Practices: Threads) Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/util/BoundedBlockingQueueSpec.scala:221
Taint Flags:

```
218 // Cause `notFull` signal, but don't fill the queue
219 latch.await(3, TimeUnit.SECONDS)
220 // queue.take() must happen first
221 Thread.sleep(50) // this is why this test is tagged as TimingTest
222 f.isCompleted should be(false)
223 lock.lockInterruptibly()
224 notEmpty.signal()
```

scala/akka/util/BoundedBlockingQueueSpec.scala, line 200 (J2EE Bad Practices: Threads) Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/util/BoundedBlockingQueueSpec.scala:200
Taint Flags:

```
197
198 latch.await(3, TimeUnit.SECONDS)
199 // queue.take() must happen first
```

J2EE Bad Practices: Threads — Low

Package: scala.akka.util

scala/akka/util/BoundedBlockingQueueSpec.scala, line 200 (J2EE Bad Practices: Threads) Low

```
200 Thread.sleep(50) // this is why this test is tagged as TimingTest
201 f.isCompleted should be(false)
202 queue.put("a")
203
```

scala/akka/util/BoundedBlockingQueueSpec.scala, line 184 (J2EE Bad Practices: Threads) Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/util/BoundedBlockingQueueSpec.scala:184
Taint Flags:

```
181
182 latch.await(3, TimeUnit.SECONDS)
183 // queue.take() must happen first
184 Thread.sleep(50) // this is why this test is tagged as TimingTest
185 events should contain(awaitNotEmpty)
186 events should not contain (poll)
187 }
```

scala/akka/util/BoundedBlockingQueueSpec.scala, line 145 (J2EE Bad Practices: Threads) Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/util/BoundedBlockingQueueSpec.scala:145
Taint Flags:

```
142
143 latch.await(3, TimeUnit.SECONDS)
144 // queue.put() must happen first
145 Thread.sleep(50) // this is why this test is tagged as TimingTest
146 f.isCompleted should be(false)
147 lock.lockInterruptibly()
148 notFull.signal()
```

J2EE Bad Practices: Threads — Low

Package: scala.akka.util

scala/akka/util/BoundedBlockingQueueSpec.scala, line 124 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/util/BoundedBlockingQueueSpec.scala:124
Taint Flags:

```
121
122 latch.await(3, TimeUnit.SECONDS)
123 // queue.take() must happen first
124 Thread.sleep(50) // this is why this test is tagged as TimingTest
125 f.isCompleted should be(false)
126 queue.take()
127
```

scala/akka/util/DurationSpec.scala, line 92 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: apply()
File: scala/akka/util/DurationSpec.scala:92
Taint Flags:

```
89 // view bounds vs. very local type inference vs. operator precedence: sigh
90 dead.timeLeft should be > (1 second: Duration)
91 dead2.timeLeft should be > (1 second: Duration)
92 Thread.sleep(1.second.toMillis)
93 dead.timeLeft should be < (1 second: Duration)
94 dead2.timeLeft should be < (1 second: Duration)
95 }
```

scala/akka/util/SwitchSpec.scala, line 84 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()

J2EE Bad Practices: Threads — Low

Package: scala.akka.util

scala/akka/util/SwitchSpec.scala, line 84 (J2EE Bad Practices: Threads) — Low

Enclosing Method: apply()
File: scala/akka/util/SwitchSpec.scala:84
Taint Flags:

```
81 val s = new Switch(false)
82
83 s.locked {
84 Thread.sleep(500)
85 s.switchOn(())
86 s.isOn should ===(true)
87 }
```

scala/akka/util/SwitchSpec.scala, line 95 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: start()
Enclosing Method: apply()
File: scala/akka/util/SwitchSpec.scala:95
Taint Flags:

```
92 s.switchOff(())
93 latch.countDown()
94 }
95 }.start()
96
97 latch.await(5, TimeUnit.SECONDS)
98 s.isOff should ===(true)
```

Null Dereference (1 issue)

Abstract

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

Explanation

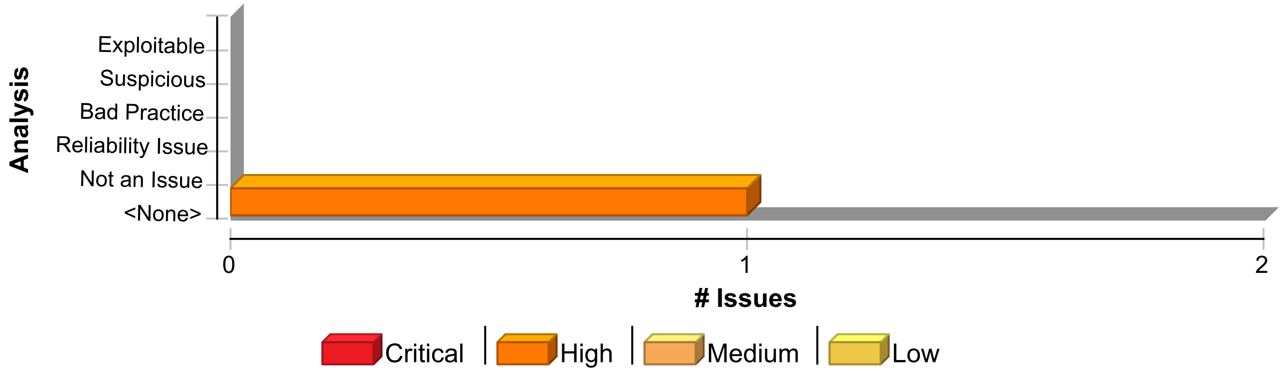
Null-pointer exceptions usually occur when one or more of the programmer's assumptions is violated. A dereference-after-store error occurs when a program explicitly sets an object to `null` and dereferences it later. This error is often the result of a programmer initializing a variable to `null` when it is declared. Most null-pointer issues result in general software reliability problems, but if attackers can intentionally trigger a null-pointer dereference, they can use the resulting exception to bypass security logic or to cause the application to reveal debugging information that will be valuable in planning subsequent attacks. **Example:** In the following code, the programmer explicitly sets the variable `foo` to `null`. Later, the programmer dereferences `foo` before checking the object for a `null` value.

```
Foo 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
Null Dereference	1	0	0	1
Total	**1**	**0**	**0**	**1**

Null Dereference **High**

Package: scala.akka.pattern

scala/akka/pattern/StatusReplySpec.scala, line 38 (Null Dereference) **High**

Issue Details

Null Dereference — High

Package: scala.akka.pattern

scala/akka/pattern/StatusReplySpec.scala, line 38 (Null Dereference) — High

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

Sink Details

Sink: Dereferenced : null
Enclosing Method: apply()
File: scala/akka/pattern/StatusReplySpec.scala:38
Taint Flags:

```
35 }
36 "not throw exception if null" in {
37 (null: StatusReply[_]) match {
38 case StatusReply.Success(_) => fail()
39 case StatusReply.Error(_) => fail()
40 case _ =>
41 }
```

Often Misused: Authentication (25 issues)

Abstract

Attackers may spoof DNS entries. Do not rely on DNS names for security.

Explanation

Many DNS servers are susceptible to spoofing attacks, so you should assume that your software will someday run in an environment with a compromised DNS server. If attackers are allowed to make DNS updates (sometimes called DNS cache poisoning), they can route your network traffic through their machines or make it appear as if their IP addresses are part of your domain. Do not base the security of your system on DNS names. **Example:** The following code uses a DNS lookup to determine whether an inbound request is from a trusted host. If an attacker can poison the DNS cache, they can gain trusted status.

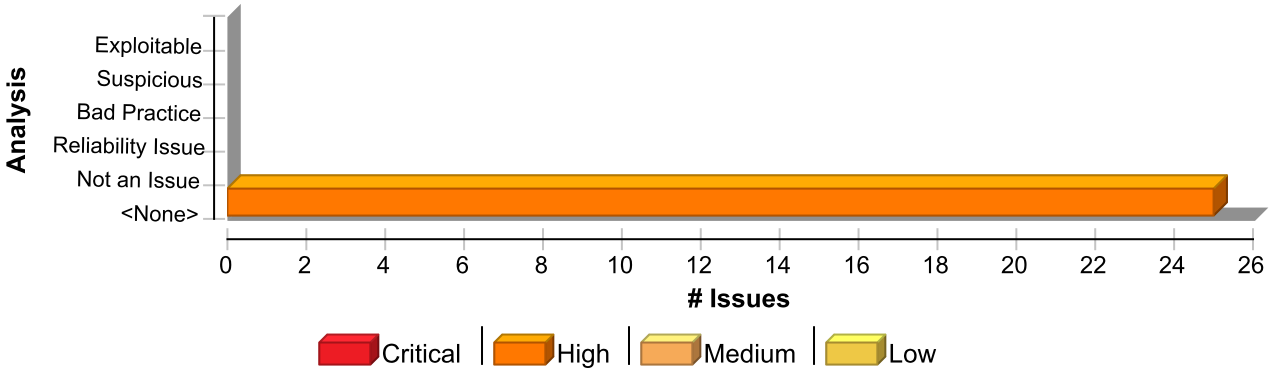
```
String ip = request.getRemoteAddr();
InetAddress addr = InetAddress.getByName(ip);
if (addr.getCanonicalHostName().endsWith("trustme.com")) {
trusted = true;
}
```

IP addresses are more reliable than DNS names, but they can also be spoofed. Attackers may easily forge the source IP address of the packets they send, but response packets will return to the forged IP address. To see the response packets, the attacker has to sniff the traffic between the victim machine and the forged IP address. In order to accomplish the required sniffing, attackers typically attempt to locate themselves on the same subnet as the victim machine. Attackers may be able to circumvent this requirement by using source routing, but source routing is disabled across much of the Internet today. In summary, IP address verification can be a useful part of an authentication scheme, but it should not be the single factor required for authentication.

Recommendation

You can increase confidence in a domain name lookup if you check to make sure that the host's forward and backward DNS entries match. Attackers will not be able to spoof both the forward and the reverse DNS entries without controlling the nameservers for the target domain. This is not a foolproof approach however: attackers may be able to convince the domain registrar to turn over the domain to a malicious nameserver. Basing authentication on DNS entries is simply a risky proposition. While no authentication mechanism is foolproof, there are better alternatives than host-based authentication. Password systems offer decent security, but are susceptible to bad password choices, insecure password transmission, and bad password management. A cryptographic scheme like SSL is worth considering, but such schemes are often so complex that they bring with them the risk of significant implementation errors, and key material can always be stolen. In many situations, multi-factor authentication including a physical token offers the most security available at a reasonable price.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Often Misused: Authentication	25	0	0	25
Total	**25**	**0**	**0**	**25**

Often Misused: Authentication — High

Package: akka.io.dns.internal

scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala, line 118 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: AsyncDnsResolverSpec$$anon$8()
File: scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala:118
Taint Flags:

```
115 dnsClient1.expectNoMessage(50.millis)
116 val answer = senderProbe.expectMsgType[Resolved]
117 answer.records.collect { case r: ARecord => r }.toSet shouldEqual Set(
118 ARecord("127.0.0.1", Ttl.effectivelyForever, InetAddress.getByName("127.0.0.1")))
119 }
120
121 "response immediately for IPv6 address" in new Setup {
```

scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala, line 81 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: AsyncDnsResolverSpec$$anon$4()
File: scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala:81
Taint Flags:

```
78 val ipv4Record = ARecord("cats.com", ttl, InetAddress.getByName("127.0.0.1"))
79 dnsClient1.reply(Answer(1, im.Seq(ipv4Record)))
80 dnsClient1.expectMsg(Question6(2, "cats.com"))
81 val ipv6Record = AAAARecord("cats.com", ttl, InetAddress.getByName("::1").asInstanceOf[Inet6Address])
82 dnsClient1.reply(Answer(2, im.Seq(ipv6Record)))
83 senderProbe.expectMsg(Resolved("cats.com", im.Seq(ipv4Record, ipv6Record)))
84 }
```

Often Misused: Authentication — High

Package: akka.io.dns.internal

scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala, line 187 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: AsyncDnsResolverSpec$$anon$13()
File: scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala:187
Taint Flags:

```
184 override val r = resolver(List(dnsClient1.ref), configWithSmallTtl)
185 val recordTtl = Ttl.fromPositive(100.seconds)
186
187 val ipv4Record = ARecord("cats.com", recordTtl, InetAddress.getByName("127.0.0.1"))
188
189 r ! Resolve("cats.com", Ip(ipv4 = true, ipv6 = false))
190 dnsClient1.expectMsg(Question4(1, "cats.com"))
```

scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala, line 140 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: AsyncDnsResolverSpec$$anon$10()
File: scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala:140
Taint Flags:

```
137 r ! Resolve("cats.com", Srv)
138 dnsClient1.expectMsg(SrvQuestion(1, "cats.com"))
139 val srvRecs = im.Seq(SRVRecord("cats.com", Ttl.fromPositive(5000.seconds), 1, 1, 1, "a.cats.com"))
140 val aRecs = im.Seq(ARecord("a.cats.com", Ttl.fromPositive(1.seconds), InetAddress.getByName("127.0.0.1")))
141 dnsClient1.reply(Answer(1, srvRecs, aRecs))
142 dnsClient2.expectNoMessage(50.millis)
143 senderProbe.expectMsg(Resolved("cats.com", srvRecs, aRecs))
```

scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala, line 78 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Often Misused: Authentication High

Package: akka.io.dns.internal

scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala, line 78 (Often Misused: Authentication) High

Sink Details

Sink: getByName()
Enclosing Method: AsyncDnsResolverSpec$$anon$4()
File: scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala:78
Taint Flags:

```
75  r ! Resolve("cats.com", Ip(ipv4 = true, ipv6 = true))
76  dnsClient1.expectMsg(Question4(1, "cats.com"))
77  val ttl = Ttl.fromPositive(100.seconds)
78  val ipv4Record = ARecord("cats.com", ttl, InetAddress.getByName("127.0.0.1"))
79  dnsClient1.reply(Answer(1, im.Seq(ipv4Record)))
80  dnsClient1.expectMsg(Question6(2, "cats.com"))
81  val ipv6Record = AAAARecord("cats.com", ttl, InetAddress.getByName("::1").asInstanceOf[Inet6Address])
```

scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala, line 207 (Often Misused: Authentication) High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: AsyncDnsResolverSpec$$anon$14()
File: scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala:207
Taint Flags:

```
204  override val r = resolver(List(dnsClient1.ref), configWithSmallTtl)
205  val recordTtl = Ttl.fromPositive(100.seconds)
206
207  val ipv4Record = ARecord("cats.com", recordTtl, InetAddress.getByName("127.0.0.1"))
208
209  r ! Resolve("cats.com", Ip(ipv4 = true, ipv6 = false))
210  dnsClient1.expectMsg(Question4(1, "cats.com"))
```

scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala, line 168 (Often Misused: Authentication) High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: AsyncDnsResolverSpec$$anon$12()

Often Misused: Authentication — High

Package: akka.io.dns.internal

scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala, line 168 (Often Misused: Authentication) — High

File: scala/akka/io/dns/internal/AsyncDnsResolverSpec.scala:168
Taint Flags:

```
165
166  "don't use resolver until record in cache will expired" in new Setup {
167  val recordTtl = Ttl.fromPositive(100.seconds)
168  val ipv4Record = ARecord("cats.com", recordTtl, InetAddress.getByName("127.0.0.1"))
169
170  r ! Resolve("cats.com", Ip(ipv4 = true, ipv6 = false))
171  dnsClient1.expectMsg(Question4(1, "cats.com"))
```

Package: scala.akka.io

scala/akka/io/SimpleDnsCacheSpec.scala, line 50 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/SimpleDnsCacheSpec.scala:50
Taint Flags:

```
47  val cacheEntry =
48  DnsProtocol.Resolved(
49  "test.local",
50  immutable.Seq(ARecord("test.local", ttl, InetAddress.getByName("127.0.0.1"))))
51  cache.put(("test.local", Ip()), cacheEntry, ttl)
52
53  cache.cached(DnsProtocol.Resolve("test.local")) should ===(Some(cacheEntry))
```

scala/akka/io/SimpleDnsCacheSpec.scala, line 31 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/SimpleDnsCacheSpec.scala:31
Taint Flags:

```
28  val ttl = Ttl.fromPositive(5000.millis)
```

Often Misused: Authentication — High

Package: scala.akka.io

scala/akka/io/SimpleDnsCacheSpec.scala, line 31 (Often Misused: Authentication) — High

```
29 val cacheEntry = DnsProtocol.Resolved(
30 "test.local",
31 immutable.Seq(ARecord("test.local", ttl, InetAddress.getByName("127.0.0.1"))))
32 cache.put(("test.local", Ip()), cacheEntry, ttl)
33
34 cache.cached(DnsProtocol.Resolve("test.local")) should ===(Some(cacheEntry))
```

Package: scala.akka.io.dns

scala/akka/io/dns/DnsSettingsSpec.scala, line 50 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/dns/DnsSettingsSpec.scala:50
Taint Flags:

```
47 eas,
48 ConfigFactory.parseString("nameservers = [\"127.0.0.1\", \"127.0.0.2\"]").withFallback(defaultConfig))
49
50 dnsSettings.NameServers.map(_.getAddress) shouldEqual List(
51 InetAddress.getByName("127.0.0.1"),
52 InetAddress.getByName("127.0.0.2"))
53 }
```

scala/akka/io/dns/DnsSettingsSpec.scala, line 42 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/dns/DnsSettingsSpec.scala:42
Taint Flags:

```
39 val dnsSettings =
40 new DnsSettings(eas, ConfigFactory.parseString("nameservers = \"127.0.0.1\"").withFallback(defaultConfig))
41
42 dnsSettings.NameServers.map(_.getAddress) shouldEqual List(InetAddress.getByName("127.0.0.1"))
43 }
44
```

Often Misused: Authentication — High

Package: scala.akka.io.dns

scala/akka/io/dns/DnsSettingsSpec.scala, line 42 (Often Misused: Authentication) — High

```
45  "parse a list of name servers" in {
```

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 192 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:192
Taint Flags:

```
189  answer.name shouldEqual "localhost"
190  answer.records.size shouldEqual 1
191  answer.records.head.name shouldEqual "localhost"
192  answer.records.head.asInstanceOf[ARecord].ip shouldEqual InetAddress.getByName("127.0.0.1")
193  }
194  }
195
```

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 105 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:105
Taint Flags:

```
102  val answer = resolve(name)
103  answer.name shouldEqual name
104
105  answer.records.collect { case r: ARecord => r.ip }.toSet shouldEqual Set(
106  InetAddress.getByName("192.168.1.23"),
107  InetAddress.getByName("192.168.1.24"))
108
```

Often Misused: Authentication High

Package: scala.akka.io.dns

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 109 (Often Misused: Authentication) High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:109
Taint Flags:

```
106 InetAddress.getByName("192.168.1.23"),
107 InetAddress.getByName("192.168.1.24"))
108
109 answer.records.collect { case r: AAAARecord => r.ip }.toSet shouldEqual Set(
110 InetAddress.getByName("fd4d:36b2:3eca:a2d8:0:0:0:4"),
111 InetAddress.getByName("fd4d:36b2:3eca:a2d8:0:0:0:5"))
112 }
```

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 87 (Often Misused: Authentication) High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:87
Taint Flags:

```
84 val name = "aaaa-single.foo.test"
85 val answer = resolve(name)
86 answer.name shouldEqual name
87 answer.records.map(_.asInstanceOf[AAAARecord].ip) shouldEqual Seq(
88 InetAddress.getByName("fd4d:36b2:3eca:a2d8:0:0:0:1"))
89 }
90
```

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 181 (Often Misused: Authentication) High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Often Misused: Authentication — High

Package: scala.akka.io.dns

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 181 (Often Misused: Authentication) — High

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:181
Taint Flags:

```
178 answer.name shouldEqual expectedName
179 answer.records.size shouldEqual 1
180 answer.records.head.name shouldEqual expectedName
181 answer.records.head.asInstanceOf[ARecord].ip shouldEqual InetAddress.getByName("192.168.1.20")
182 }
183 }
184
```

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 169 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:169
Taint Flags:

```
166 answer.name shouldEqual expectedName
167 answer.records.size shouldEqual 1
168 answer.records.head.name shouldEqual expectedName
169 answer.records.head.asInstanceOf[ARecord].ip shouldEqual InetAddress.getByName("192.168.1.20")
170 }
171 }
172
```

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 145 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()

Often Misused: Authentication

High

Package: scala.akka.io.dns

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 145 (Often Misused: Authentication)

High

File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:145
Taint Flags:

```
142 "resolve same address twice" in {
143 resolve("a-single.foo.test").records.map(_.asInstanceOf[ARecord].ip) shouldEqual Seq(
144 InetAddress.getByName("192.168.1.20"))
145 resolve("a-single.foo.test").records.map(_.asInstanceOf[ARecord].ip) shouldEqual Seq(
146 InetAddress.getByName("192.168.1.20"))
147 }
148
```

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 143 (Often Misused: Authentication)

High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:143
Taint Flags:

```
140 }
141
142 "resolve same address twice" in {
143 resolve("a-single.foo.test").records.map(_.asInstanceOf[ARecord].ip) shouldEqual Seq(
144 InetAddress.getByName("192.168.1.20"))
145 resolve("a-single.foo.test").records.map(_.asInstanceOf[ARecord].ip) shouldEqual Seq(
146 InetAddress.getByName("192.168.1.20"))
```

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 70 (Often Misused: Authentication)

High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:70
Taint Flags:

```
67 answer.name shouldEqual name
```

Often Misused: Authentication — High

Package: scala.akka.io.dns

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 70 (Often Misused: Authentication) — High

```
68  answer.records.size shouldEqual 1
69  answer.records.head.name shouldEqual name
70  answer.records.head.asInstanceOf[ARecord].ip shouldEqual InetAddress.getByName("192.168.1.20")
71  }
72  }
73
```

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 95 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:95
Taint Flags:

```
92  val name = "aaaa-double.foo.test"
93  val answer = resolve(name)
94  answer.name shouldEqual name
95  answer.records.map(_.asInstanceOf[AAAARecord].ip).toSet shouldEqual Set(
96  InetAddress.getByName("fd4d:36b2:3eca:a2d8:0:0:0:2"),
97  InetAddress.getByName("fd4d:36b2:3eca:a2d8:0:0:0:3"))
98  }
```

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 127 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:127
Taint Flags:

```
124  val answer = resolve(name)
125  answer.name shouldEqual name
126  answer.records.collect { case r: CNameRecord => r.canonicalName }.toSet shouldEqual Set("a-double.foo.test")
127  answer.records.collect { case r: ARecord => r.ip }.toSet shouldEqual Set(
```

Often Misused: Authentication — High

Package: scala.akka.io.dns

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 127 (Often Misused: Authentication) — High

```
128 InetAddress.getByName("192.168.1.21"),
129 InetAddress.getByName("192.168.1.22"))
130 }
```

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 78 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:78
Taint Flags:

```
75 val name = "a-double.foo.test"
76 val answer = resolve(name)
77 answer.name shouldEqual name
78 answer.records.map(_.asInstanceOf[ARecord].ip).toSet shouldEqual Set(
79 InetAddress.getByName("192.168.1.21"),
80 InetAddress.getByName("192.168.1.22"))
81 }
```

scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala, line 119 (Often Misused: Authentication) — High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/dns/AsyncDnsResolverIntegrationSpec.scala:119
Taint Flags:

```
116 val answer = (IO(Dns) ? DnsProtocol.Resolve(name)).mapTo[DnsProtocol.Resolved].futureValue
117 answer.name shouldEqual name
118 answer.records.collect { case r: CNameRecord => r.canonicalName }.toSet shouldEqual Set("a-single.bar.example")
119 answer.records.collect { case r: ARecord => r.ip }.toSet shouldEqual Set(InetAddress.getByName("192.168.2.20"))
120 }
121
122 "resolve internal CNAME record" in {
```

Often Misused: Authentication High

Package: scala.akka.io.dns.internal

scala/akka/io/dns/internal/AsyncDnsManagerSpec.scala, line 34 (Often Misused: Authentication) High

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: getByName()
Enclosing Method: apply()
File: scala/akka/io/dns/internal/AsyncDnsManagerSpec.scala:34
Taint Flags:

```
31 "Async DNS Manager" must {
32 "adapt reply back to old protocol when old protocol Dns.Resolve is received" in {
33 dns ! akka.io.Dns.Resolve("127.0.0.1") // 127.0.0.1 will short circuit the resolution
34 val oldProtocolReply = akka.io.Dns.Resolved("127.0.0.1", InetAddress.getByName("127.0.0.1") :: Nil)
35 expectMsg(oldProtocolReply)
36 }
37
```

Poor Error Handling: Empty Catch Block (1 issue)

Abstract

Ignoring an exception can cause the program to overlook unexpected states and conditions.

Explanation

Just about every serious attack on a software system begins with the violation of a programmer's assumptions. After the attack, the programmer's assumptions seem flimsy and poorly founded, but before an attack many programmers would defend their assumptions well past the end of their lunch break. Two dubious assumptions that are easy to spot in code are "this method call can never fail" and "it doesn't matter if this call fails". When a programmer ignores an exception, they implicitly state that they are operating under one of these assumptions. **Example 1:** The following code excerpt ignores a rarely-thrown exception from `doExchange()`.

```
try {
  doExchange();
}
catch (RareException e) {
  // this can never happen
}
```

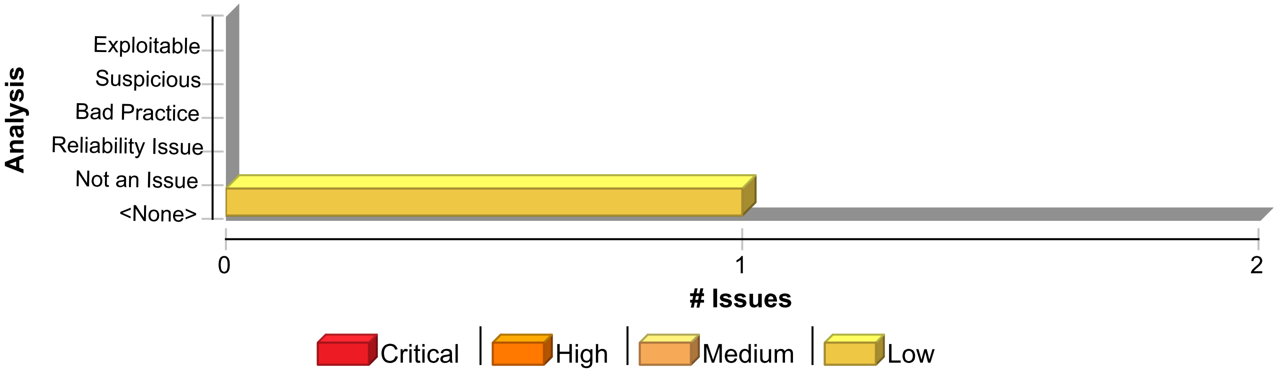
If a `RareException` were to ever be thrown, the program would continue to execute as though nothing unusual had occurred. The program records no evidence indicating the special situation, potentially frustrating any later attempt to explain the program's behavior.

Recommendation

At a minimum, log the fact that the exception was thrown so that it will be possible to come back later and make sense of the resulting program behavior. Better yet, abort the current operation. If the exception is being ignored because the caller cannot properly handle it but the context makes it inconvenient or impossible for the caller to declare that it throws the exception itself, consider throwing a `RuntimeException` or an `Error`, both of which are unchecked exceptions. As of JDK 1.4, `RuntimeException` has a constructor that makes it easy to wrap another exception.
Example 2: The code in `Example 1` could be rewritten in the following way:

```
try {
  doExchange();
}
catch (RareException e) {
  throw new RuntimeException("This can never happen", e);
}
```

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Poor Error Handling: Empty Catch Block	1	0	0	1
Total	**1**	**0**	**0**	**1**

Poor Error Handling: Empty Catch Block Low

Package: akka.actor

scala/akka/actor/SchedulerSpec.scala, line 738 (Poor Error Handling: Empty Catch Block) Low

Issue Details

Kingdom: Errors
Scan Engine: SCA (Structural)

Sink Details

Sink: CatchBlock
Enclosing Method: withScheduler()
File: scala/akka/actor/SchedulerSpec.scala:738
Taint Flags:

```
735 try {
736 driver.close()
737 sched.close()
738 } catch { case _: Exception => }
739 throw ex
740 }
741 driver.close()
```

Poor Error Handling: Overly Broad Catch (1 issue)

Abstract

The catch block handles a broad swath of exceptions, potentially trapping dissimilar issues or problems that should not be dealt with at this point in the program.

Explanation

Multiple catch blocks can get repetitive, but "condensing" catch blocks by catching a high-level class such as `Exception` can obscure exceptions that deserve special treatment or that should not be caught at this point in the program. Catching an overly broad exception essentially defeats the purpose of Java's typed exceptions, and can become particularly dangerous if the program grows and begins to throw new types of exceptions. The new exception types will not receive any attention. **Example:** The following code excerpt handles three types of exceptions in an identical fashion.

```
  try {
    doExchange();
  }
  catch (IOException e) {
    logger.error("doExchange failed", e);
  }
  catch (InvocationTargetException e) {
    logger.error("doExchange failed", e);
  }
  catch (SQLException e) {
    logger.error("doExchange failed", e);
  }
```

At first blush, it may seem preferable to deal with these exceptions in a single catch block, as follows:

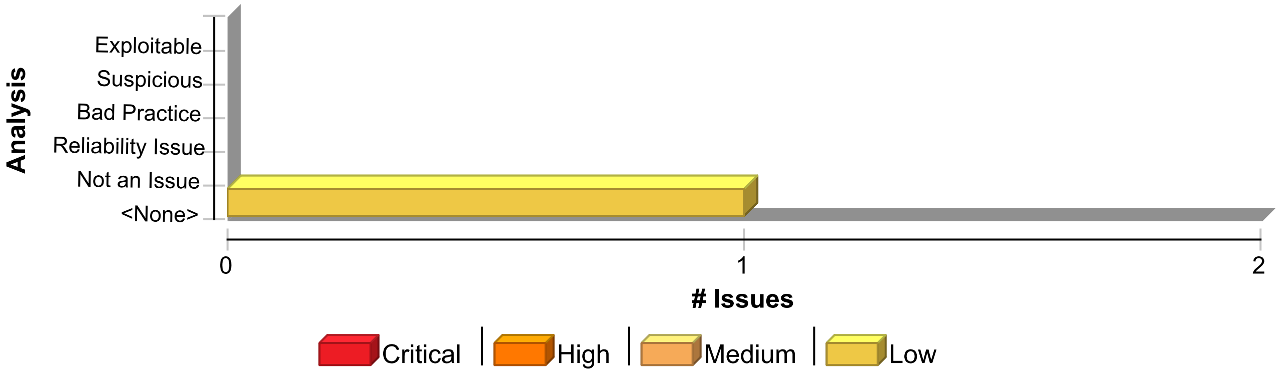
```
  try {
    doExchange();
  }
  catch (Exception e) {
    logger.error("doExchange failed", e);
  }
```

However, if `doExchange()` is modified to throw a new type of exception that should be handled in some different kind of way, the broad catch block will prevent the compiler from pointing out the situation. Further, the new catch block will now also handle exceptions derived from `RuntimeException` such as `ClassCastException`, and `NullPointerException`, which is not the programmer's intent.

Recommendation

Do not catch broad exception classes such as `Exception`, `Throwable`, `Error`, or `RuntimeException` except at the very top level of the program or thread.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Poor Error Handling: Overly Broad Catch	1	0	0	1
Total	**1**	**0**	**0**	**1**

Poor Error Handling: Overly Broad Catch

Low

Package: akka.actor

scala/akka/actor/SchedulerSpec.scala, line 738 (Poor Error Handling: Overly Broad Catch)

Low

Issue Details

Kingdom: Errors
Scan Engine: SCA (Structural)

Sink Details

Sink: CatchBlock
Enclosing Method: withScheduler()
File: scala/akka/actor/SchedulerSpec.scala:738
Taint Flags:

```
735 try {
736 driver.close()
737 sched.close()
738 } catch { case _: Exception => }
739 throw ex
740 }
741 driver.close()
```

Poor Style: Value Never Read (4 issues)

Abstract

The variable's value is assigned but never used, making it a dead store.

Explanation

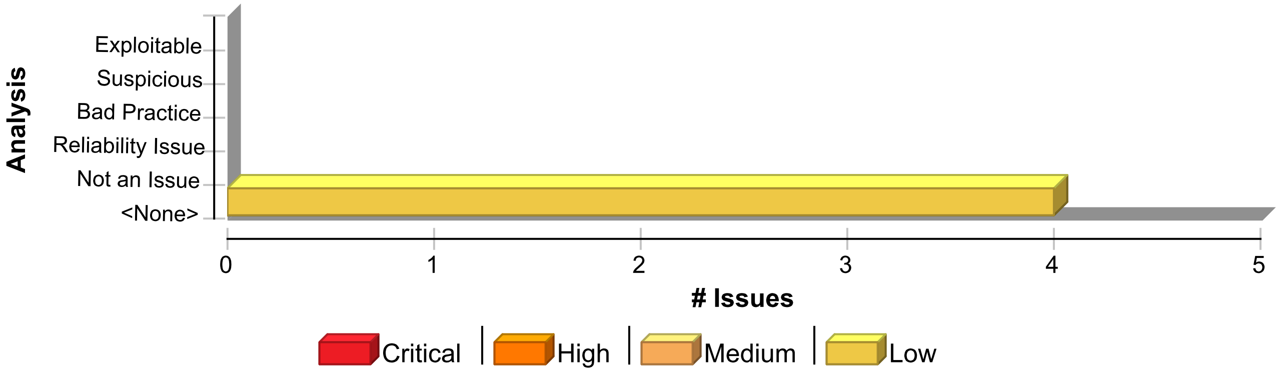
This variable's value is not used. After the assignment, the variable is either assigned another value or goes out of scope. **Example:** The following code excerpt assigns to the variable `r` and then overwrites the value without using it.

```
  r = getName();
  r = getNewBuffer(buf);
```

Recommendation

Remove unnecessary assignments in order to make the code easier to understand and maintain.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Poor Style: Value Never Read	4	0	0	4
Total	**4**	**0**	**0**	**4**

Poor Style: Value Never Read — Low

Package: akka.actor

scala/akka/actor/LocalActorRefProviderSpec.scala, line 151 (Poor Style: Value Never Read) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: VariableAccess: b
Enclosing Method: applyOrElse()
File: scala/akka/actor/LocalActorRefProviderSpec.scala:151

Poor Style: Value Never Read Low

Package: akka.actor

scala/akka/actor/LocalActorRefProviderSpec.scala, line 151 (Poor Style: Value Never Read) Low

Taint Flags:

```
148 val supervisor = system.actorOf(Props(new Actor {
149 def receive = {
150 case "" =>
151 val a, b = context.actorOf(Props.empty, "duplicate")
152 }
153 }))
154 EventFilter[InvalidActorNameException](occurrences = 1).intercept {
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 523 (Poor Style: Value Never Read) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: VariableAccess: dir
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:523
Taint Flags:

```
520 case this.Event(Work, x) if x > 0 =>
521 nextJob.next() match {
522 case Ping(ref) => ref ! "ping"
523 case Fail(ref, dir) =>
524 val f = Failure(
525 dir,
526 stop = random012 > 0,
```

scala/akka/actor/SupervisorHierarchySpec.scala, line 523 (Poor Style: Value Never Read) Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: VariableAccess: ref~1
Enclosing Method: applyOrElse()
File: scala/akka/actor/SupervisorHierarchySpec.scala:523
Taint Flags:

```
520 case this.Event(Work, x) if x > 0 =>
521 nextJob.next() match {
522 case Ping(ref) => ref ! "ping"
523 case Fail(ref, dir) =>
```

Poor Style: Value Never Read — Low

Package: akka.actor

scala/akka/actor/SupervisorHierarchySpec.scala, line 523 (Poor Style: Value Never Read) — Low

```
524 val f = Failure(
525 dir,
526 stop = random012 > 0,
```

scala/akka/actor/LocalActorRefProviderSpec.scala, line 151 (Poor Style: Value Never Read) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: VariableAccess: a
Enclosing Method: applyOrElse()
File: scala/akka/actor/LocalActorRefProviderSpec.scala:151
Taint Flags:

```
148 val supervisor = system.actorOf(Props(new Actor {
149 def receive = {
150 case "" =>
151 val a, b = context.actorOf(Props.empty, "duplicate")
152 }
153 }))
154 EventFilter[InvalidActorNameException](occurrences = 1).intercept {
```

Resource Injection (1 issue)

Abstract

Allowing user input to control resource identifiers could enable an attacker to access or modify otherwise protected system resources.

Explanation

A resource injection issue occurs when the following two conditions are met: 1. An attacker is able to specify the identifier used to access a system resource. For example, an attacker may be able to specify a port number to be used to connect to a network resource. 2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted. For example, the program may give the attacker the ability to transmit sensitive information to a third-party server. Note: Resource injections involving resources stored on the file system are reported in a separate category named path manipulation. See the path manipulation description for further details of this vulnerability.
Example 1: The following code uses a port number read from an HTTP request to create a socket.

```
String remotePort = request.getParameter("remotePort");
...
ServerSocket srvr = new ServerSocket(remotePort);
Socket skt = srvr.accept();
...
```

Some think that in the mobile world, classic web application vulnerabilities, such as resource injection, do not make sense -- why would the user attack themself? However, keep in mind that the essence of mobile platforms is applications that are downloaded from various sources and run alongside each other on the same device. The likelihood of running a piece of malware next to a banking application is high, which necessitates expanding the attack surface of mobile applications to include inter-process communication. **Example 2:** The following code uses a URL read from an Android intent to load the page in `WebView`.

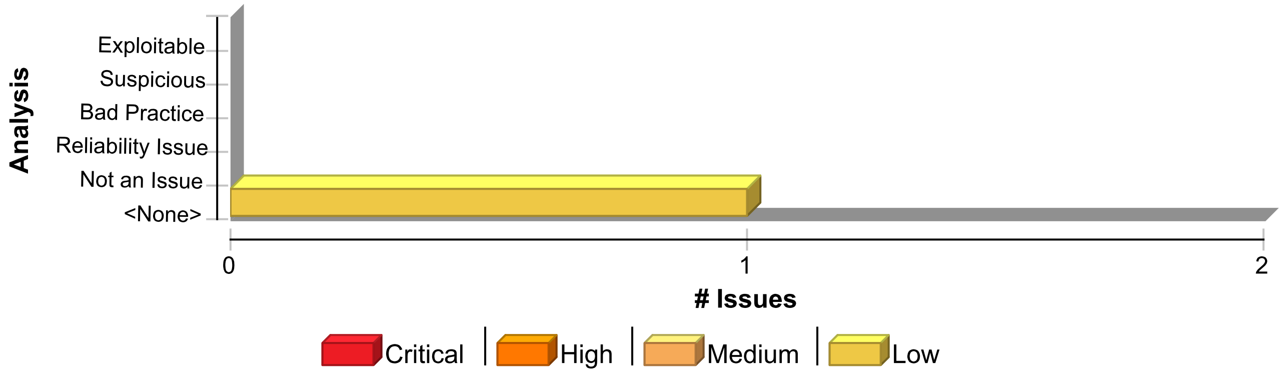
```
...
    WebView webview = new WebView(this);
    setContentView(webview);
        String url = this.getIntent().getExtras().getString("url");
    webview.loadUrl(url);
...
```

The kind of resource affected by user input indicates the kind of content that may be dangerous. For example, data containing special characters like period, slash, and backslash are risky when used in methods that interact with the file system. Similarly, data that contains URLs and URIs is risky for functions that create remote connections.

Recommendation

The best way to prevent resource injection is with a level of indirection: create a list of legitimate resource names that a user is allowed to specify, and only allow the user to select from the list. With this approach the input provided by the user is never used directly to specify the resource name. In some situations this approach is impractical because the set of legitimate resource names is too large or too hard to maintain. Programmers often resort to implementing a deny list in these situations. A deny list is used to selectively reject or escape potentially dangerous characters before using the input. However, any such list of unsafe characters is likely to be incomplete and will almost certainly become out of date. A better approach is to create a list of characters that are permitted to appear in the resource name and accept input composed exclusively of characters in the approved set.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Resource Injection	1	0	0	1
Total	**1**	**0**	**0**	**1**

Resource Injection — Low

Package: akka.util

scala/akka/util/ByteStringInitializationSpec.scala, line 25 (Resource Injection) — Low

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.io.InputStream.read()
From: akka.util.ByteStringInitializationSpec$$anon$1.slurp
File: scala/akka/util/ByteStringInitializationSpec.scala:33

```
30 }
31
32 def slurp(is: InputStream, res: ArrayBuilder[Byte]): Array[Byte] = {
33 val read = is.read(buffer)
34 if (read == 0) throw new IllegalStateException
35 else if (read > 0) slurp(is, res ++= buffer.take(read))
36 else res.result()
```

Sink Details

Sink: java.lang.ClassLoader.defineClass()
Enclosing Method: loadClass()
File: scala/akka/util/ByteStringInitializationSpec.scala:25
Taint Flags: NUMBER, STREAM

```
22 val buffer = new Array[Byte](1000000)
23 override def loadClass(name: String): Class[_] =
24 if (!name.startsWith("akka")) outerCl.loadClass(name)
25 else {
26 val classFile = name.replace(".", "/") + ".class"
```

Resource Injection Low

Package: akka.util

scala/akka/util/ByteStringInitializationSpec.scala, line 25 (Resource Injection) Low

```
val is = outerCl.getResourceAsStream(classFile)
val res = slurp(is, new mutable.ArrayBuilder.ofByte)
```

Setting Manipulation (1 issue)

Abstract

Allowing external control of system settings can disrupt service or cause an application to behave in unexpected ways.

Explanation

Setting manipulation vulnerabilities occur when an attacker can control values that govern the behavior of the system, manage specific resources, or in some way affect the functionality of the application. Because setting manipulation covers a diverse set of functions, any attempt to illustrate it will inevitably be incomplete. Rather than searching for a tight-knit relationship between the functions addressed in the setting manipulation category, take a step back and consider the sorts of system values that an attacker should not be allowed to control. **Example 1:** The following Java code snippet reads a string from an `HttpServletRequest` and sets it as the active catalog for a database `Connection`.

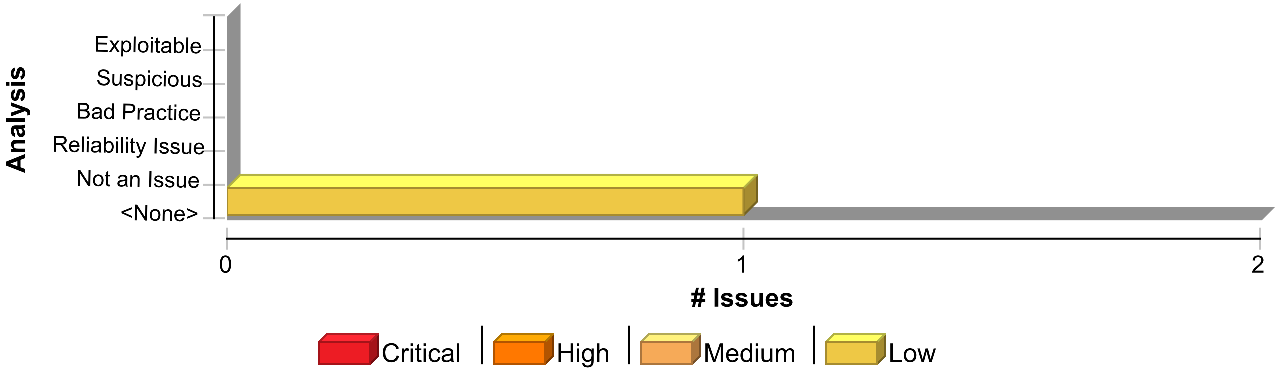
```
...
conn.setCatalog(request.getParamter("catalog"));
...
```

In this example, an attacker could cause an error by providing a nonexistent catalog name or connect to an unauthorized portion of the database. In general, do not allow user-provided or otherwise untrusted data to control sensitive values. The leverage that an attacker gains by controlling these values is not always immediately obvious, but do not underestimate the creativity of your attacker.

Recommendation

Do not allow untrusted data to control sensitive values. In many cases where this error occurs, the application expects a particular input to hold only a very small range of values. If possible, instead of relying on the input to remain within an expected range, the application should guarantee reasonable behavior by using the input only to select from a predetermined set of safe values. If the input is maliciously crafted, the value passed to the sensitive function should default to some safe selection from this set. Even if the set of safe values cannot be known in advance, it is often possible to validate that the input falls within some safe range of values. If neither of these forms of validation is possible, you may have to redesign the application to avoid the need to accept potentially dangerous values from the user.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Setting Manipulation	1	0	0	1
Total	**1**	**0**	**0**	**1**

Setting Manipulation — Low

Package: scala.akka.io

scala/akka/io/InetAddressDnsResolverSpec.scala, line 107 (Setting Manipulation) — Low

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.lang.System.getProperty()
From: akka.io.InetAddressDnsResolverSpec.withNewSystemProperty
File: scala/akka/io/InetAddressDnsResolverSpec.scala:102

```
99  }
100
101  private def withNewSystemProperty[T](property: String, testValue: String)(test: => T): T = {
102  val oldValue = Option(System.getProperty(property))
103  try {
104  System.setProperty(property, testValue)
105  test
```

Sink Details

Sink: java.lang.System.setProperty()
Enclosing Method: apply()
File: scala/akka/io/InetAddressDnsResolverSpec.scala:107
Taint Flags: PROPERTY

```
104  System.setProperty(property, testValue)
105  test
106  } finally {
107  oldValue.foreach(v => System.setProperty(property, v))
108  }
109  }
110
```

System Information Leak (1 issue)

Abstract

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

Explanation

An information leak occurs when system data or debug information leaves the program through an output stream or logging function. **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 remote user. For example, with scripting mechanisms it is trivial to redirect output information from "Standard error" or "Standard output" into a file or another program. Alternatively, the system that the program runs on could have a remote logging mechanism such as a "syslog" server that sends the logs to a remote device. During development, you have no way of knowing where this information might end up being displayed. 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. Here is another scenario, specific to the mobile world. Most mobile devices now implement a Near-Field Communication (NFC) protocol for quickly sharing information between devices using radio communication. It works by bringing devices to close proximity or simply having them touch each other. Even though the communication range of NFC is limited to just a few centimeters, eavesdropping, data modification and various other types of attacks are possible, since NFC alone does not ensure secure communication. **Example 2:** The Android platform provides support for NFC. The following code creates a message that gets pushed to the other device within the range.

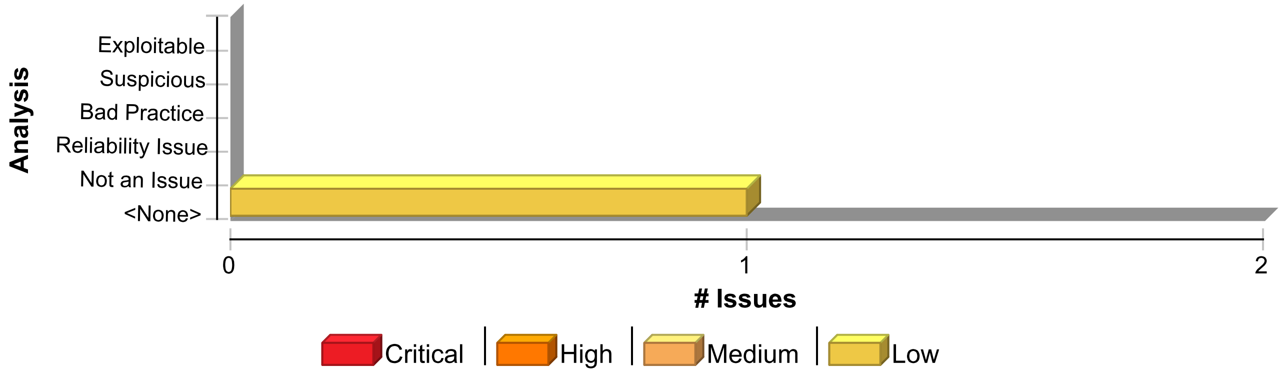
```
...
public static final String TAG = "NfcActivity";
private static final String DATA_SPLITTER = "__:DATA:__";
private static final String MIME_TYPE = "application/my.applications.mimetype";
...
public NdefMessage createNdefMessage(NfcEvent event) {
    TelephonyManager tm =
(TelephonyManager)Context.getSystemService(Context.TELEPHONY_SERVICE);
    String VERSION = tm.getDeviceSoftwareVersion();
    String text = TAG + DATA_SPLITTER + VERSION;
    NdefRecord record = new NdefRecord(NdefRecord.TNF_MIME_MEDIA,
            MIME_TYPE.getBytes(), new byte[0], text.getBytes());
    NdefRecord[] records = { record };
    NdefMessage msg = new NdefMessage(records);
    return msg;
}
...
```

NFC Data Exchange Format (NDEF) message contains typed data, a URI, or a custom application payload. If the message contains information about the application, such as its name, MIME type, or device software version, this information could be leaked to an eavesdropper. In `Example 2`, Fortify Static Code Analyzer reports a System Information Leak vulnerability on the return statement.

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. If you are concerned about leaking system data via NFC on an Android device, you could do one of the following three things. Do not include system data in the messages pushed to other devices in range, encrypt the payload of the message, or establish a secure communication channel at a higher layer.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
System Information Leak	1	0	0	1
Total	**1**	**0**	**0**	**1**

System Information Leak — Low

Package: akka.actor

scala/akka/actor/SchedulerSpec.scala, line 684 (System Information Leak) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Semantic)

Sink Details

Sink: printStackTrace()
Enclosing Method: reportFailure()
File: scala/akka/actor/SchedulerSpec.scala:684
Taint Flags:

```
681
682 val localEC = new ExecutionContext {
683 def execute(runnable: Runnable): Unit = { runnable.run() }
684 def reportFailure(t: Throwable): Unit = { t.printStackTrace() }
685 }
686
687 @nowarn
```

System Information Leak: External (2 issues)

Abstract

Revealing system data or debugging information could enable an adversary to use system information to plan an attack.

Explanation

An external information leak occurs when system data or debug information leaves the program to a remote machine via a socket or network connection. External leaks can help an attacker by revealing specific data about operating systems, full pathnames, the existence of usernames, or locations of configuration files, and are more serious than internal information leaks, which are more difficult for an attacker to access. **Example 1:** The following code leaks System details in the HTTP response:

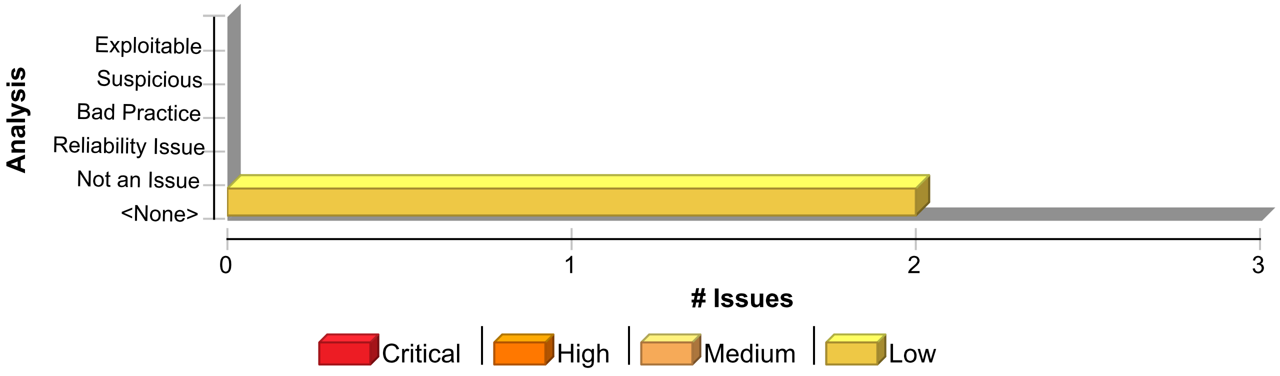
```
def doSomething() = Action { request =>
    ...
    Ok(Html(Properties.osName)) as HTML
}
```

This information can be exposed to a remote 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.

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. Because of this, never send information to a resource directly outside the program.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
System Information Leak: External	2	0	0	2
Total	**2**	**0**	**0**	**2**

System Information Leak: External — Low

Package: akka.actor

scala/akka/actor/SupervisorSpec.scala, line 59 (System Information Leak: External) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.lang.Throwable.getMessage()
From: akka.actor.SupervisorSpec$PingPongActor.postRestart
File: scala/akka/actor/SupervisorSpec.scala:59

```
56  }
57
58  override def postRestart(reason: Throwable): Unit = {
59  sendTo ! reason.getMessage
60  }
61  }
62
```

Sink Details

Sink: akka.actor.FunctionRef.!()
Enclosing Method: postRestart()
File: scala/akka/actor/SupervisorSpec.scala:59
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
56  }
57
58  override def postRestart(reason: Throwable): Unit = {
59  sendTo ! reason.getMessage
60  }
61  }
62
```

Package: akka.util

scala/akka/util/MessageBufferSpec.scala, line 172 (System Information Leak: External) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.lang.Throwable.getMessage()
From: akka.actor.SupervisorSpec$PingPongActor.postRestart
File: scala/akka/actor/SupervisorSpec.scala:59

```
56  }
```

System Information Leak: External — Low

Package: akka.util

scala/akka/util/MessageBufferSpec.scala, line 172 (System Information Leak: External) — Low

```

 override def postRestart(reason: Throwable): Unit = {
 sendTo ! reason.getMessage
 }
 }

```

Sink Details

Sink: akka.util.MessageBufferSpec.DummyActorRef.!()
Enclosing Method: !()
File: scala/akka/util/MessageBufferSpec.scala:172
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
169 }
170
171 object MessageBufferSpec {
172  final private[akka] class DummyActorRef(val id: String) extends MinimalActorRef {
173
174  override def toString: String = id
175
```

Unchecked Return Value (4 issues)

Abstract

Ignoring a method's return value can cause the program to overlook unexpected states and conditions.

Explanation

It is not uncommon for Java programmers to misunderstand `read()` and related methods that are part of many `java.io` classes. Most errors and unusual events in Java result in an exception being thrown. (This is one of the advantages that Java has over languages like C: Exceptions make it easier for programmers to think about what can go wrong.) But the stream and reader classes do not consider it unusual or exceptional if only a small amount of data becomes available. These classes simply add the small amount of data to the return buffer, and set the return value to the number of bytes or characters read. There is no guarantee that the amount of data returned is equal to the amount of data requested. This behavior makes it important for programmers to examine the return value from `read()` and other IO methods to ensure that they receive the amount of data they expect. **Example:** The following code loops through a set of users, reading a private data file for each user. The programmer assumes that the files are always exactly 1 kilobyte in size and therefore ignores the return value from `read()`. If an attacker can create a smaller file, the program will recycle the remainder of the data from the previous user and handle it as though it belongs to the attacker.

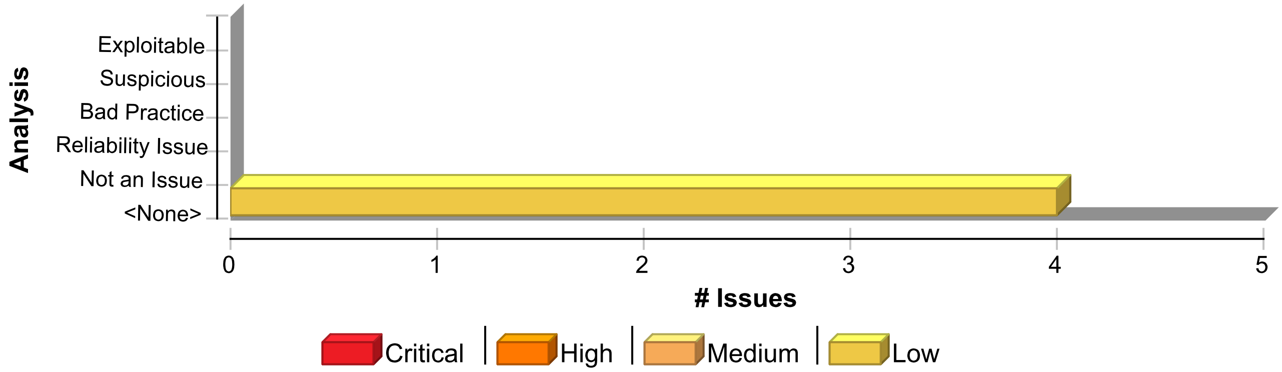
```
FileInputStream fis;
byte[] byteArray = new byte[1024];
for (Iterator i=users.iterator(); i.hasNext();) {
    String userName = (String) i.next();
    String pFileName = PFILE_ROOT + "/" + userName;
    FileInputStream fis = new FileInputStream(pFileName);
    fis.read(byteArray); // the file is always 1k bytes
    fis.close();
    processPFile(userName, byteArray);
}
```

Recommendation

```
  FileInputStream fis;
  byte[] byteArray = new byte[1024];
  for (Iterator i=users.iterator(); i.hasNext();) {
    String userName = (String) i.next();
    String pFileName = PFILE_ROOT + "/" + userName;
    fis = new FileInputStream(pFileName);
    int bRead = 0;
    while (bRead < 1024) {
        int rd = fis.read(byteArray, bRead, 1024 - bRead);
        if (rd == -1) {
          throw new IOException("file is unusually small");
        }
        bRead += rd;
    }
    // could add check to see if file is too large here
    fis.close();
    processPFile(userName, byteArray);
  }
```

Note: Because the fix for this problem is relatively complicated, you might be tempted to use a simpler approach, such as checking the size of the file before you begin reading. Such an approach would render the application vulnerable to a file system race condition, whereby an attacker could replace a well-formed file with a malicious file between the file size check and the call to read data from the file.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Unchecked Return Value	4	0	0	4
Total	**4**	**0**	**0**	**4**

Unchecked Return Value — Low

Package: akka.actor

scala/akka/actor/Ticket669Spec.scala, line 24 (Unchecked Return Value) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: interrupted()
Enclosing Method: atStartup()
File: scala/akka/actor/Ticket669Spec.scala:24
Taint Flags:

```
21
22  // TODO: does this really make sense?
23  override def atStartup(): Unit = {
24  Thread.interrupted() //remove interrupted status.
25  }
26
27  "A supervised actor with lifecycle PERMANENT" should {
```

Package: scala.akka.actor.dispatch

scala/akka/actor/dispatch/ActorModelSpec.scala, line 426 (Unchecked Return Value) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Unchecked Return Value — Low

Package: scala.akka.actor.dispatch

scala/akka/actor/dispatch/ActorModelSpec.scala, line 426 (Unchecked Return Value) — Low

Sink: interrupted()
Enclosing Method: apply()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:426
Taint Flags:

```
423  Thread.interrupted() // CallingThreadDispatcher may necessitate this
424  val f4 = a ? Reply("foo2")
425  val f5 = a ? Interrupt
426  Thread.interrupted() // CallingThreadDispatcher may necessitate this
427  val f6 = a ? Reply("bar2")
428
429  val c = system.scheduler.scheduleOnce(2.seconds) {
```

scala/akka/actor/dispatch/ActorModelSpec.scala, line 423 (Unchecked Return Value) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: interrupted()
Enclosing Method: apply()
File: scala/akka/actor/dispatch/ActorModelSpec.scala:423
Taint Flags:

```
420  val f1 = a ? Reply("foo")
421  val f2 = a ? Reply("bar")
422  val f3 = a ? Interrupt
423  Thread.interrupted() // CallingThreadDispatcher may necessitate this
424  val f4 = a ? Reply("foo2")
425  val f5 = a ? Interrupt
426  Thread.interrupted() // CallingThreadDispatcher may necessitate this
```

Package: scala.akka.util

scala/akka/util/ByteStringSpec.scala, line 1187 (Unchecked Return Value) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Semantic)

Sink Details

Sink: skip()
Enclosing Method: apply()
File: scala/akka/util/ByteStringSpec.scala:1187
Taint Flags:

```
1184  val input = bytes.iterator
```

Unchecked Return Value — Low

Package: scala.akka.util

scala/akka/util/ByteStringSpec.scala, line 1187 (Unchecked Return Value) — Low

```
1185  val output = new Array[Byte](bytes.length)
1186
1187  input.asInputStream.skip(a)
1188
1189  val toRead = b - a
1190  var (nRead, eof) = (0, false)
```

Unreleased Resource: Sockets (2 issues)

Abstract

The program can potentially fail to release a socket.

Explanation

The program can potentially fail to release a socket. Resource leaks have at least two common causes: - Error conditions and other exceptional circumstances. - Confusion over which part of the program is responsible for releasing the resource. Most unreleased resource issues result in general software reliability problems. However, if an attacker can intentionally trigger a resource leak, the attacker may be able to launch a denial of service attack by depleting the resource pool. **Example 1:** The following method never closes the socket it opens. In a busy environment, this can result in the JVM using up all of its sockets.

```
private void echoSocket(String host, int port) throws UnknownHostException,
SocketException, IOException
{
  Socket sock = new Socket(host, port);
  BufferedReader reader = new BufferedReader(new
InputStreamReader(sock.getInputStream()));

  while ((String socketData = reader.readLine()) != null) {
    System.out.println(socketData);
  }
}
```

Example 2: Under normal conditions, the following fix properly closes the socket and any associated streams. But if an exception occurs while reading the input or writing the data to screen, the socket object will not be closed. If this happens often enough, the system will run out of sockets and not be able to handle any further connections.

```
private void echoSocket(String host, int port) throws UnknownHostException,
SocketException, IOException
{
  Socket sock = new Socket(host, port);
  BufferedReader reader = new BufferedReader(new
InputStreamReader(sock.getInputStream()));

  while ((String socketData = reader.readLine()) != null) {
    System.out.println(socketData);
  }
  sock.close();
}
```

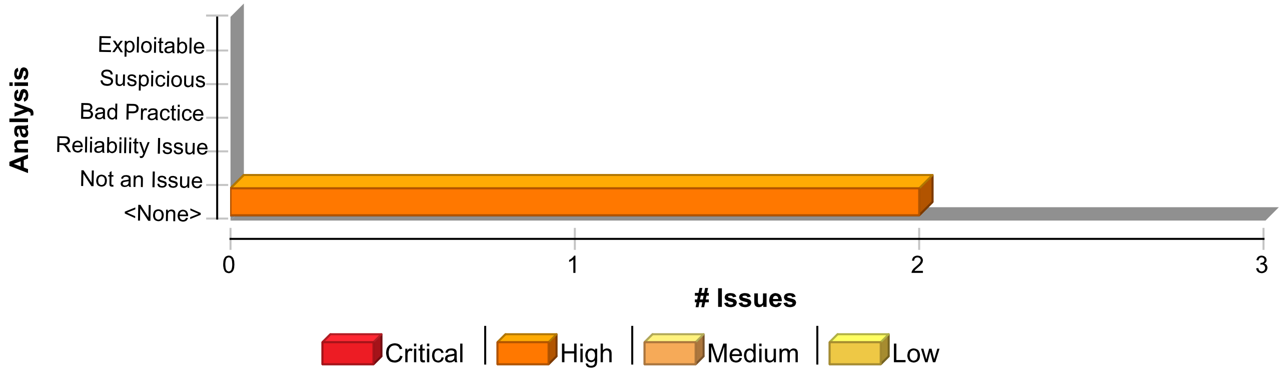
Recommendation

Release socket resources in a `finally` block. The code for `Example 2` should be rewritten as follows:

```
private void echoSocket(String host, int port) throws UnknownHostException,
SocketException, IOException
{
  Socket sock;
  BufferedReader reader;

  try {
    sock = new Socket(host, port);
    reader = new BufferedReader(new InputStreamReader(sock.getInputStream()));

    while ((String socketData = reader.readLine()) != null) {
        System.out.println(socketData);
    }
  }
  finally {
    safeClose(sock);
  }
}

public static void safeClose(Socket s) {
  if (s != null && !s.isClosed()) {
    try {
      s.close();
    } catch (IOException e) {
      log(e);
    }
  }
}
```

This solution uses a helper function to log the exceptions that might occur when trying to close the socket. Presumably this helper function will be reused whenever a socket needs to be closed. Also, the `echoSocket()` method does not initialize the `sock` socket object to `null`. Instead, it checks to ensure that `sock` is not `null` before calling `safeClose()`. Without the `null` check, the Java compiler reports that `sock` might not be initialized. This choice takes advantage of Java's ability to detect uninitialized variables. If `sock` is initialized to `null` in a more complex method, cases in which `sock` is used without being initialized will not be detected by the compiler.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Unreleased Resource: Sockets	2	0	0	2
Total	**2**	**0**	**0**	**2**

Unreleased Resource: Sockets — High

Package: akka.io

scala/akka/io/TcpListenerSpec.scala, line 165 (Unreleased Resource: Sockets) — High

Issue Details

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

Sink Details

Unreleased Resource: Sockets — High

Package: akka.io

scala/akka/io/TcpListenerSpec.scala, line 165 (Unreleased Resource: Sockets) — High

Sink: new Socket(...)
Enclosing Method: attemptConnectionToEndpoint()
File: scala/akka/io/TcpListenerSpec.scala:165
Taint Flags:

```
162 bindCommander.expectMsgType[Bound]
163 }
164
165 def attemptConnectionToEndpoint(): Unit = new Socket(endpoint.getHostName, endpoint.getPort)
166
167 def listener = parentRef.underlyingActor.listener
168
```

Package: scala.akka.io

scala/akka/io/TcpConnectionSpec.scala, line 836 (Unreleased Resource: Sockets) — High

Issue Details

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

Sink Details

Sink: serverSocket = new ServerSocket(...)
Enclosing Method: apply()
File: scala/akka/io/TcpConnectionSpec.scala:836
Taint Flags:

```
833 // This test needs the OP_CONNECT workaround on Windows, see original report #15033 and parent ticket #15766
834
835 val bindAddress = SocketUtil.temporaryServerAddress()
836 val serverSocket = new ServerSocket(bindAddress.getPort, 100, bindAddress.getAddress)
837 val connectionProbe = TestProbe()
838
839 connectionProbe.send(IO(Tcp), Connect(bindAddress))
```

Unreleased Resource: Streams (1 issue)

Abstract

The program can potentially fail to release a system resource.

Explanation

The program can potentially fail to release a system resource. Resource leaks have at least two common causes: - Error conditions and other exceptional circumstances. - Confusion over which part of the program is responsible for releasing the resource. Most unreleased resource issues result in general software reliability problems. However, if an attacker can intentionally trigger a resource leak, the attacker may be able to launch a denial of service attack by depleting the resource pool. **Example:** The following method never closes the file handle it opens. The `finalize()` method for `FileInputStream` eventually calls `close()`, but there is no guarantee as to how long it will take before the `finalize()` method will be invoked. In a busy environment, this can result in the JVM using up all of its file handles.

```
private void processFile(String fName) throws FileNotFoundException,
IOException {
  FileInputStream fis = new FileInputStream(fName);
  int sz;
  byte[] byteArray = new byte[BLOCK_SIZE];
  while ((sz = fis.read(byteArray)) != -1) {
    processBytes(byteArray, sz);
  }
}
```

Recommendation

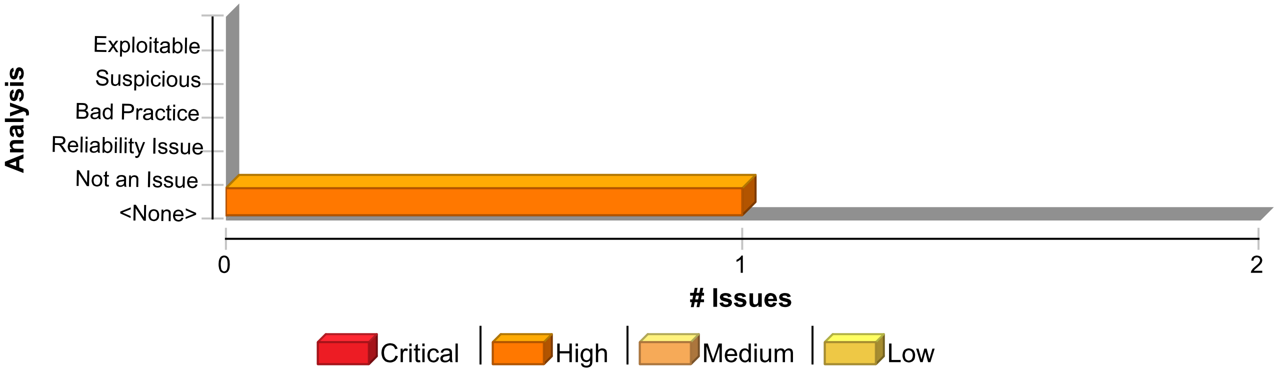
1. Never rely on `finalize()` to reclaim resources. In order for an object's `finalize()` method to be invoked, the garbage collector must determine that the object is eligible for garbage collection. Because the garbage collector is not required to run unless the JVM is low on memory, there is no guarantee that an object's `finalize()` method will be invoked in an expedient fashion. When the garbage collector finally does run, it may cause a large number of resources to be reclaimed in a short period of time, which can lead to "bursty" performance and lower overall system throughput. This effect becomes more pronounced as the load on the system increases. Finally, if it is possible for a resource reclamation operation to hang (if it requires communicating over a network to a database, for example), then the thread that is executing the `finalize()` method will hang. 2. Release resources in a `finally` block. The code for the Example should be rewritten as follows:

```
public void processFile(String fName) throws FileNotFoundException,
IOException {
  FileInputStream fis;
  try {
    fis = new FileInputStream(fName);
    int sz;
    byte[] byteArray = new byte[BLOCK_SIZE];
    while ((sz = fis.read(byteArray)) != -1) {
      processBytes(byteArray, sz);
    }
  }
  finally {
    if (fis != null) {
      safeClose(fis);
    }
  }
}
```

```
public static void safeClose(FileInputStream fis) {
  if (fis != null) {
    try {
      fis.close();
    } catch (IOException e) {
      log(e);
    }
  }
}
```

This solution uses a helper function to log the exceptions that might occur when trying to close the stream. Presumably this helper function will be reused whenever a stream needs to be closed. Also, the `processFile` method does not initialize the `fis` object to `null`. Instead, it checks to ensure that `fis` is not `null` before calling `safeClose()`. Without the `null` check, the Java compiler reports that `fis` might not be initialized. This choice takes advantage of Java's ability to detect uninitialized variables. If `fis` is initialized to `null` in a more complex method, cases in which `fis` is used without being initialized will not be detected by the compiler.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Unreleased Resource: Streams	1	0	0	1
Total	**1**	**0**	**0**	**1**

Unreleased Resource: Streams — High

Package: akka.util

scala/akka/util/ByteStringInitializationSpec.scala, line 27 (Unreleased Resource: Streams) High

Issue Details

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

Sink Details

Sink: is = getResourceAsStream(...)
Enclosing Method: loadClass()
File: scala/akka/util/ByteStringInitializationSpec.scala:27
Taint Flags:

```
24  if (!name.startsWith("akka")) outerCl.loadClass(name)
25  else {
```

Unreleased Resource: Streams — High

Package: akka.util

scala/akka/util/ByteStringInitializationSpec.scala, line 27 (Unreleased Resource: Streams) — High

```
26 val classFile = name.replace(".", "/") + ".class"
27 val is = outerCl.getResourceAsStream(classFile)
28 val res = slurp(is, new mutable.ArrayBuilder.ofByte)
29 defineClass(name, res, 0, res.length)
30 }
```