Fortify Standalone Report Generator

Developer Workbook

akka-actor-testkit-typed

Table of Contents

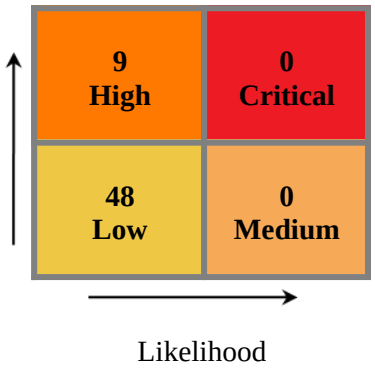
Executive Summary

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-testkit-typed project audit. The information contained in this workbook is targeted at project managers and developers.

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

Project Name:	akka-actor-testkit-typed
Project Version:	
SCA:	Results Present
WebInspect:	Results Not Present
WebInspect Agent:	Results Not Present
Other:	Results Not Present

Issues by Priority



Top Ten Critical Categories

This project does not contain any critical issues

Project Description

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

SCA

Date of Last Analysis:	Jun 16, 2022, 11:00 AM	**Engine Version:**	21.1.1.0009
Host Name:	Jacks-Work-MBP.local	**Certification:**	VALID
Number of Files:	41	**Lines of Code:**	1,585

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

Issue Breakdown by Fortify Categories

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

Category	Fortify Priority (audited/total)				Total Issues
	Critical	High	Medium	Low	
Code Correctness: Constructor Invokes Overridable Function	0	0	0	0 / 23	0 / 23
Code Correctness: Non-Static Inner Class Implements Serializable	0	0	0	0 / 16	0 / 16
Dead Code: Expression is Always false	0	0	0	0 / 1	0 / 1
Dead Code: Expression is Always true	0	0	0	0 / 1	0 / 1
Insecure Randomness	0	0 / 9	0	0	0 / 9
J2EE Bad Practices: Threads	0	0	0	0 / 4	0 / 4
Poor Error Handling: Overly Broad Catch	0	0	0	0 / 1	0 / 1
Redundant Null Check	0	0	0	0 / 1	0 / 1
System Information Leak	0	0	0	0 / 1	0 / 1

Results Outline

Code Correctness: Constructor Invokes Overridable Function (23 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");
    }
  }
}
```

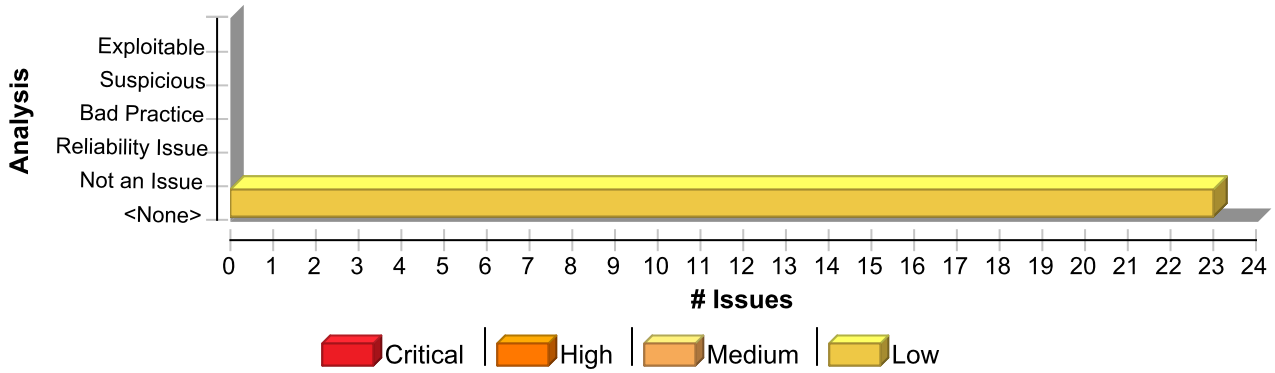
The code in `Example 1` prints "Attack successful!", since the `Attacker` class overrides the `validateUser()` function that is called from the constructor of the superclass `User`, and Java will first look in the subclass for functions called from the constructor.

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	23	0	0	23
Total	**23**	**0**	**0**	**23**

Code Correctness: Constructor Invokes Overridable Function	**Low**
Package: akka.actor.testkit.typed	
TestKitSettings.scala, line 78 (Code Correctness: Constructor Invokes Overridable Function)	**Low**
Issue Details	

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor.testkit.typed

TestKitSettings.scala, line 78 (Code Correctness: Constructor Invokes Overridable Function) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: dilated
Enclosing Method: TestKitSettings()
File: TestKitSettings.scala:78
Taint Flags:

```
75 val ThrowOnShutdownTimeout: Boolean = config.getBoolean("throw-on-shutdown-timeout")
76
77 /** Dilated with `TestTimeFactor`. */
78 val FilterLeeway: FiniteDuration = dilated(config.getMillisDuration("filter-leeway"))
79
80 /**
81 * Scala API: Scale the `duration` with the configured `TestTimeFactor`
```

TestKitSettings.scala, line 64 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: dilated
Enclosing Method: TestKitSettings()
File: TestKitSettings.scala:64
Taint Flags:

```
61 .requiring(tf => !tf.isInfinite && tf > 0, "timefactor must be positive finite double")
62
63 /** Dilated with `TestTimeFactor`. */
64 val SingleExpectDefaultTimeout: FiniteDuration = dilated(config.getMillisDuration("single-expect-default"))
65
66 /** Dilated with `TestTimeFactor`. */
67 val ExpectNoMessageDefaultTimeout: FiniteDuration = dilated(config.getMillisDuration("expect-no-message-default"))
```

TestKitSettings.scala, line 67 (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.testkit.typed

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

Sink Details

Sink: FunctionCall: dilated
Enclosing Method: TestKitSettings()
File: TestKitSettings.scala:67
Taint Flags:

```
64 val SingleExpectDefaultTimeout: FiniteDuration = dilated(config.getMillisDuration("single-expect-default"))
65
66 /** Dilated with `TestTimeFactor`. */
67 val ExpectNoMessageDefaultTimeout: FiniteDuration = dilated(config.getMillisDuration("expect-no-message-default"))
68
69 /** Dilated with `TestTimeFactor`. */
70 val DefaultTimeout: Timeout = Timeout(dilated(config.getMillisDuration("default-timeout")))
```

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

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: dilated
Enclosing Method: TestKitSettings()
File: TestKitSettings.scala:73
Taint Flags:

```
70 val DefaultTimeout: Timeout = Timeout(dilated(config.getMillisDuration("default-timeout")))
71
72 /** Dilated with `TestTimeFactor`. */
73 val DefaultActorSystemShutdownTimeout: FiniteDuration = dilated(config.getMillisDuration("system-shutdown-default"))
74
75 val ThrowOnShutdownTimeout: Boolean = config.getBoolean("throw-on-shutdown-timeout")
76
```

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

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: dilated
Enclosing Method: TestKitSettings()

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor.testkit.typed

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

File: TestKitSettings.scala:70
Taint Flags:

```
67 val ExpectNoMessageDefaultTimeout: FiniteDuration = dilated(config.getMillisDuration("expect-no-message-default"))
68
69 /** Dilated with `TestTimeFactor`. */
70 val DefaultTimeout: Timeout = Timeout(dilated(config.getMillisDuration("default-timeout")))
71
72 /** Dilated with `TestTimeFactor`. */
73 val DefaultActorSystemShutdownTimeout: FiniteDuration = dilated(config.getMillisDuration("system-shutdown-default"))
```

Package: akka.actor.testkit.typed.internal

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

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: rootPath
Enclosing Method: ActorSystemStub()
File: internal/ActorSystemStub.scala:46
Taint Flags:

```
43
44 private val rootPath: ActorPath = classic.RootActorPath(classic.Address("akka", name))
45
46 override val path: classic.ActorPath = rootPath / "user"
47
48 override val settings: Settings = {
49 val classLoader = getClass.getClassLoader
```

internal/TestProbeImpl.scala, line 88 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: queue
Enclosing Method: TestProbeImpl()
File: internal/TestProbeImpl.scala:88
Taint Flags:

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor.testkit.typed.internal

internal/TestProbeImpl.scala, line 88 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
85 */
86 private var lastWasNoMessage = false
87
88 private val testActor: ActorRef[M] =
89 system.systemActorOf(TestProbeImpl.testActor(queue, terminations), s"$name-${testActorId.incrementAndGet()}")
90
91 override def ref: ActorRef[M] = testActor
```

internal/StubbedActorContext.scala, line 80 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: selfInbox
Enclosing Method: StubbedActorContext()
File: internal/StubbedActorContext.scala:80
Taint Flags:

```
77 */
78 @InternalApi private[akka] val selfInbox = new TestInboxImpl[T](path)
79
80 override val self = selfInbox.ref
81 private var _children = TreeMap.empty[String, BehaviorTestKitImpl[_]]
82 private val childName = Iterator.from(0).map(Helpers.base64(_))
83 private val substituteLoggerFactory = new SubstituteLoggerFactory
```

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

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: context
Enclosing Method: BehaviorTestKitImpl()
File: internal/BehaviorTestKitImpl.scala:43
Taint Flags:

```
40
41 private var currentUncanonical = _initialBehavior
42 private var current = {
```

Code Correctness: Constructor Invokes Overridable Function	Low

Package: akka.actor.testkit.typed.internal

internal/BehaviorTestKitImpl.scala, line 43 (Code Correctness: Constructor Invokes Overridable Function) **Low**

```
43 try {
44 context.setCurrentActorThread()
45 Behavior.validateAsInitial(Behavior.start(_initialBehavior, context))
46 } finally {
```

internal/BehaviorTestKitImpl.scala, line 44 (Code Correctness: Constructor Invokes Overridable Function) **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: context
Enclosing Method: BehaviorTestKitImpl()
File: internal/BehaviorTestKitImpl.scala:44
Taint Flags:

```
41 private var currentUncanonical = _initialBehavior
42 private var current = {
43 try {
44 context.setCurrentActorThread()
45 Behavior.validateAsInitial(Behavior.start(_initialBehavior, context))
46 } finally {
47 context.clearCurrentActorThread()
```

internal/BehaviorTestKitImpl.scala, line 47 (Code Correctness: Constructor Invokes Overridable Function) **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: context
Enclosing Method: BehaviorTestKitImpl()
File: internal/BehaviorTestKitImpl.scala:47
Taint Flags:

```
44 context.setCurrentActorThread()
45 Behavior.validateAsInitial(Behavior.start(_initialBehavior, context))
46 } finally {
47 context.clearCurrentActorThread()
48 }
49 }
```

Code Correctness: Constructor Invokes Overridable Function	Low

Package: akka.actor.testkit.typed.internal

internal/BehaviorTestKitImpl.scala, line 47 (Code Correctness: Constructor Invokes Overridable Function) **Low**

```
50
```

internal/TestProbeImpl.scala, line 88 (Code Correctness: Constructor Invokes Overridable Function) **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: terminations
Enclosing Method: TestProbeImpl()
File: internal/TestProbeImpl.scala:88
Taint Flags:

```
85 */
86 private var lastWasNoMessage = false
87
88 private val testActor: ActorRef[M] =
89 system.systemActorOf(TestProbeImpl.testActor(queue, terminations), s"$name-${testActorId.incrementAndGet()}")
90
91 override def ref: ActorRef[M] = testActor
```

internal/StubbedActorContext.scala, line 84 (Code Correctness: Constructor Invokes Overridable Function) **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: substituteLoggerFactory
Enclosing Method: StubbedActorContext()
File: internal/StubbedActorContext.scala:84
Taint Flags:

```
81 private var _children = TreeMap.empty[String, BehaviorTestKitImpl[_]]
82 private val childName = Iterator.from(0).map(Helpers.base64(_))
83 private val substituteLoggerFactory = new SubstituteLoggerFactory
84 private val logger: Logger = substituteLoggerFactory.getLogger("StubbedLogger")
85 private var unhandled: List[T] = Nil
86
87 private[akka] def classicActorContext =
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor.testkit.typed.internal

internal/TestProbeImpl.scala, line 88 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: testActorId
Enclosing Method: TestProbeImpl()
File: internal/TestProbeImpl.scala:88
Taint Flags:

```
85 */
86 private var lastWasNoMessage = false
87
88 private val testActor: ActorRef[M] =
89 system.systemActorOf(TestProbeImpl.testActor(queue, terminations), s"$name-${testActorId.incrementAndGet()}")
90
91 override def ref: ActorRef[M] = testActor
```

internal/ActorSystemStub.scala, line 76 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: path
Enclosing Method: ActorSystemStub()
File: internal/ActorSystemStub.scala:76
Taint Flags:

```
73 // impl InternalRecipientRef
74 def isTerminated: Boolean = whenTerminated.isCompleted
75
76 val deadLettersInbox = new DebugRef[Any](path.parent / "deadLetters", true)
77 override def deadLetters[U]: ActorRef[U] = deadLettersInbox
78
79 override def ignoreRef[U]: ActorRef[U] = deadLettersInbox
```

internal/ActorSystemStub.scala, line 81 (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.testkit.typed.internal

internal/ActorSystemStub.scala, line 81 (Code Correctness: Constructor Invokes Overridable Function) — Low

Sink Details

Sink: FunctionCall: path
Enclosing Method: ActorSystemStub()
File: internal/ActorSystemStub.scala:81
Taint Flags:

```
78
79 override def ignoreRef[U]: ActorRef[U] = deadLettersInbox
80
81 val receptionistInbox = new TestInboxImpl[Receptionist.Command](path.parent / "receptionist")
82
83 override def receptionist: ActorRef[Receptionist.Command] = receptionistInbox.ref
84
```

internal/BehaviorTestKitImpl.scala, line 52 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: runAllTasks
Enclosing Method: BehaviorTestKitImpl()
File: internal/BehaviorTestKitImpl.scala:52
Taint Flags:

```
49 }
50
51 // execute any future tasks scheduled in Actor's constructor
52 runAllTasks()
53
54 override def retrieveEffect(): Effect = context.effectQueue.poll() match {
55 case null => NoEffects
```

internal/TestProbeImpl.scala, line 88 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: akka$actor$testkit$typed$internal$TestProbeImpl$$testActor
Enclosing Method: TestProbeImpl()

Code Correctness: Constructor Invokes Overridable Function	Low

Package: akka.actor.testkit.typed.internal

internal/TestProbeImpl.scala, line 88 (Code Correctness: Constructor Invokes Overridable Function) — Low

File: internal/TestProbeImpl.scala:88
Taint Flags:

```
85 */
86 private var lastWasNoMessage = false
87
88 private val testActor: ActorRef[M] =
89 system.systemActorOf(TestProbeImpl.testActor(queue, terminations), s"$name-${testActorId.incrementAndGet()}")
90
91 override def ref: ActorRef[M] = testActor
```

Package: akka.actor.testkit.typed.scaladsl

scaladsl/ActorTestKitBase.scala, line 47 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: testNameFromCallStack
Enclosing Method: ActorTestKitBase()
File: scaladsl/ActorTestKitBase.scala:47
Taint Flags:

```
44 /**
45 * Use a custom config for the actor system.
46 */
47 def this(config: Config) = this(ActorTestKit(ActorTestKitBase.testNameFromCallStack(), config))
48
49 /**
50 * Use a custom config for the actor system, and a custom [[akka.actor.testkit.typed.TestKitSettings]].
```

scaladsl/ActorTestKitBase.scala, line 53 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: testNameFromCallStack
Enclosing Method: ActorTestKitBase()
File: scaladsl/ActorTestKitBase.scala:53
Taint Flags:

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor.testkit.typed.scaladsl

scaladsl/ActorTestKitBase.scala, line 53 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
50 * Use a custom config for the actor system, and a custom [[akka.actor.testkit.typed.TestKitSettings]].
51 */
52 def this(config: Config, settings: TestKitSettings) =
53 this(ActorTestKit(ActorTestKitBase.testNameFromCallStack(), config, settings))
54
55 // delegates of the TestKit api for minimum fuss
56 /**
```

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

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: testNameFromCallStack
Enclosing Method: ActorTestKitBase()
File: scaladsl/ActorTestKitBase.scala:36
Taint Flags:

```
33 */
34 abstract class ActorTestKitBase(val testKit: ActorTestKit) {
35
36 def this() = this(ActorTestKit(ActorTestKitBase.testNameFromCallStack()))
37
38 /**
39 * Use a custom config for the actor system.
```

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

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: testNameFromCallStack
Enclosing Method: ActorTestKitBase()
File: scaladsl/ActorTestKitBase.scala:42
Taint Flags:

```
39 * Use a custom config for the actor system.
40 */
41 def this(config: String) =
```

Code Correctness: Constructor Invokes Overridable Function	Low

Package: akka.actor.testkit.typed.scaladsl

scaladsl/ActorTestKitBase.scala, line 42 (Code Correctness: Constructor Invokes Overridable Function) **Low**

```
42  this(ActorTestKit(ActorTestKitBase.testNameFromCallStack(), ConfigFactory.parseString(config)))
43
44  /**
45  * Use a custom config for the actor system.
```

scaladsl/ActorTestKit.scala, line 206 (Code Correctness: Constructor Invokes Overridable Function) **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: testKitSettings
Enclosing Method: ActorTestKit()
File: scaladsl/ActorTestKit.scala:206
Taint Flags:

```
203
204  private val childName: Iterator[String] = Iterator.from(0).map(_.toString)
205
206  implicit val timeout: Timeout = testKitSettings.DefaultTimeout
207
208  def scheduler: Scheduler = system.scheduler
209
```

Code Correctness: Non-Static Inner Class Implements Serializable (16 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

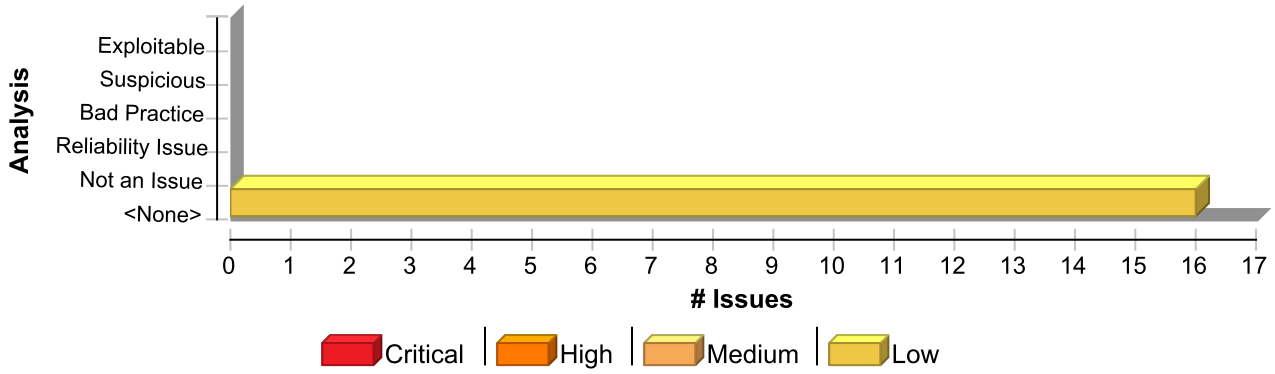
When using inner classes, they should not be serialized, or they should be changed to static-nested classes, since these do not have the drawbacks that non-static inner classes have when serialized. When a nested class is static it inherently has no association with instance variables (including those 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	16	0	0	16
Total	**16**	**0**	**0**	**16**

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

Package: akka.actor.testkit.typed

Effect.scala, line 92 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Effect$SpawnedAdapter
File: Effect.scala:92
Taint Flags:

```
89 * Spawning adapters is private[akka]
90 */
91 @InternalApi
92 private[akka] final class SpawnedAdapter[T](val name: String, val ref: ActorRef[T])
93 extends Effect
94 with Product1[String]
95 with Serializable {
```

Effect.scala, line 124 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Effect$SpawnedAnonymousAdapter
File: Effect.scala:124
Taint Flags:

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

Package: akka.actor.testkit.typed

Effect.scala, line 124 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
121 * The behavior spawned an anonymous adapter, through `context.spawnMessageAdapter`
122 */
123 @InternalApi
124 private[akka] final class SpawnedAnonymousAdapter[T](val ref: ActorRef[T])
125 extends Effect
126 with Product
127 with Serializable {
```

Effect.scala, line 166 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Effect$Stopped
File: Effect.scala:166
Taint Flags:

```
163 /**
164 * The behavior stopped `childName`
165 */
166 final case class Stopped(childName: String) extends Effect
167
168 /**
169 * The behavior started watching `other`, through `context.watch(other)`
```

Effect.scala, line 171 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Effect$Watched
File: Effect.scala:171
Taint Flags:

```
168 /**
169 * The behavior started watching `other`, through `context.watch(other)`
170 */
171 final case class Watched[T](other: ActorRef[T]) extends Effect
172
173 /**
174 * The behavior started watching `other`, through `context.watchWith(other, message)`
```

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

Package: akka.actor.testkit.typed

Effect.scala, line 221 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Effect$TimerScheduled$FixedRateModeWithInitialDelay
File: Effect.scala:221
Taint Flags:

```
218
219 sealed trait TimerMode
220 case object FixedRateMode extends TimerMode
221 case class FixedRateModeWithInitialDelay(initialDelay: FiniteDuration) extends TimerMode
222 case object FixedDelayMode extends TimerMode
223 case class FixedDelayModeWithInitialDelay(initialDelay: FiniteDuration) extends TimerMode
224 case object SingleMode extends TimerMode
```

Effect.scala, line 223 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Effect$TimerScheduled$FixedDelayModeWithInitialDelay
File: Effect.scala:223
Taint Flags:

```
220 case object FixedRateMode extends TimerMode
221 case class FixedRateModeWithInitialDelay(initialDelay: FiniteDuration) extends TimerMode
222 case object FixedDelayMode extends TimerMode
223 case class FixedDelayModeWithInitialDelay(initialDelay: FiniteDuration) extends TimerMode
224 case object SingleMode extends TimerMode
225
226 /*Java API*/
```

Effect.scala, line 155 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Effect$MessageAdapter
File: Effect.scala:155

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

Package: akka.actor.testkit.typed

Effect.scala, line 155 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Taint Flags:

```
152 /**
153 * The behavior create a message adapter for the messages of type clazz
154 */
155 final case class MessageAdapter[A, T](messageClass: Class[A], adapt: A => T) extends Effect {
156
157 /**
158 * JAVA API
```

Effect.scala, line 186 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Effect$ReceiveTimeoutSet
File: Effect.scala:186
Taint Flags:

```
183 /**
184 * The behavior set a new receive timeout, with `message` as timeout notification
185 */
186 final case class ReceiveTimeoutSet[T](d: FiniteDuration, message: T) extends Effect {
187
188 /**
189 * Java API
```

Effect.scala, line 202 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Effect$Scheduled
File: Effect.scala:202
Taint Flags:

```
199 * The behavior used `context.scheduleOnce` to schedule `message` to be sent to `target` after `delay`
200 * FIXME what about events scheduled through the scheduler?
201 */
202 final case class Scheduled[U](delay: FiniteDuration, target: ActorRef[U], message: U) extends Effect {
203 def duration(): java.time.Duration = delay.asJava
204 }
```

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

Package: akka.actor.testkit.typed

Effect.scala, line 202 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
205
```

Effect.scala, line 32 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Effect$Spawned
File: Effect.scala:32
Taint Flags:

```
29 /**
30 * The behavior spawned a named child with the given behavior (and optionally specific props)
31 */
32 final class Spawned[T](val behavior: Behavior[T], val childName: String, val props: Props, val ref: ActorRef[T])
33 extends Effect
34 with Product3[Behavior[T], String, Props]
35 with Serializable {
```

Effect.scala, line 176 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Effect$WatchedWith
File: Effect.scala:176
Taint Flags:

```
173 /**
174 * The behavior started watching `other`, through `context.watchWith(other, message)`
175 */
176 final case class WatchedWith[U, T](other: ActorRef[U], message: T) extends Effect
177
178 /**
179 * The behavior stopped watching `other`, through `context.unwatch(other)`
```

Effect.scala, line 237 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

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

Package: akka.actor.testkit.typed

Effect.scala, line 237 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink Details

Sink: Class: Effect$TimerCancelled
File: Effect.scala:237
Taint Flags:

```
234 /*Java API*/
235 def timerScheduled = TimerScheduled
236
237 final case class TimerCancelled(key: Any) extends Effect
238
239 /**
240 * Used to represent an empty list of effects - in other words, the behavior didn't do anything observable
```

Effect.scala, line 206 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Effect$TimerScheduled
File: Effect.scala:206
Taint Flags:

```
203 def duration(): java.time.Duration = delay.asJava
204 }
205
206 final case class TimerScheduled[U](
207 key: Any,
208 msg: U,
209 delay: FiniteDuration,
```

Effect.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: Effect$SpawnedAnonymous
File: Effect.scala:63
Taint Flags:

```
60 /**
61 * The behavior spawned an anonymous child with the given behavior (and optionally specific props)
62 */
63 final class SpawnedAnonymous[T](val behavior: Behavior[T], val props: Props, val ref: ActorRef[T])
```

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

Package: akka.actor.testkit.typed

Effect.scala, line 63 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
64 extends Effect
65 with Product2[Behavior[T], Props]
66 with Serializable {
```

Effect.scala, line 181 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Effect$Unwatched
File: Effect.scala:181
Taint Flags:

```
178 /**
179 * The behavior stopped watching `other`, through `context.unwatch(other)`
180 */
181 final case class Unwatched[T](other: ActorRef[T]) extends Effect
182
183 /**
184 * The behavior set a new receive timeout, with `message` as timeout notification
```

Package: akka.actor.testkit.typed.internal

internal/TestProbeImpl.scala, line 42 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TestProbeImpl$WatchActor
File: internal/TestProbeImpl.scala:42
Taint Flags:

```
39
40 @InternalApi
41 private[akka] object TestProbeImpl {
42 private final case class WatchActor[U](actor: ActorRef[U])
43 private case object Stop
44
45 private def testActor[M](queue: BlockingDeque[M], terminations: BlockingDeque[Terminated]): Behavior[M] =
```

Dead Code: Expression is Always false (1 issue)

Abstract

This expression will always evaluate to `false`.

Explanation

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

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

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

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

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

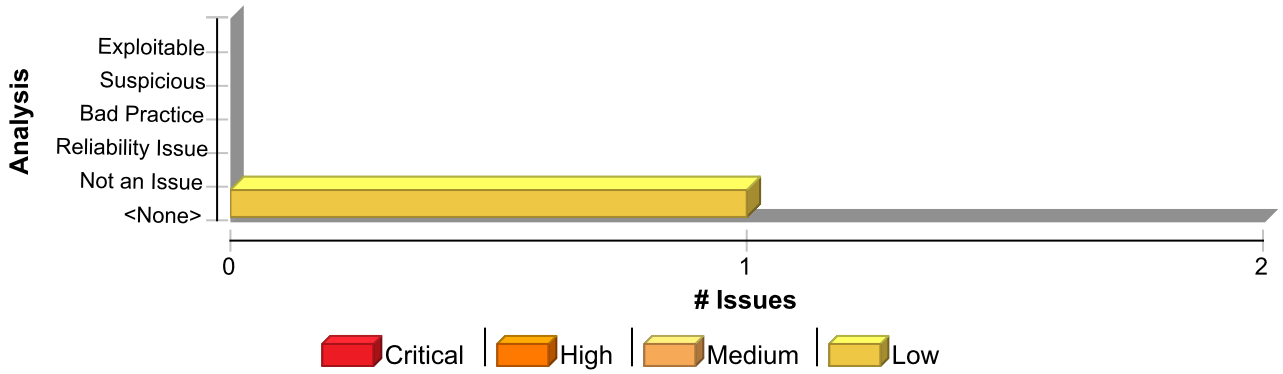
```
public void setUpCalls() {
  boolean firstCall = false;
  boolean secondCall = false;

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

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

Recommendation

In general, you should repair or remove unused code. It causes additional complexity and maintenance burden without

contributing to the functionality of the program.

Issue Summary



Engine Breakdown

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

Dead Code: Expression is Always false — Low

Package: akka.actor.testkit.typed.scaladsl

scaladsl/LogCapturing.scala, line 81 (Dead Code: Expression is Always false) — **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: withFixture()
File: scaladsl/LogCapturing.scala:81
Taint Flags:

```
78  s"<-- [${Console.BLUE}${self.getClass.getName}: ${test.name}${Console.RESET}] End of log messages of test that [$res]")
79  }
80
81  if (clearLogsAfterEachTest) {
82  capturingAppender.clear()
83  }
84
```

Dead Code: Expression is Always true (1 issue)

Abstract

This expression will always evaluate to `true`.

Explanation

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

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

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

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

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

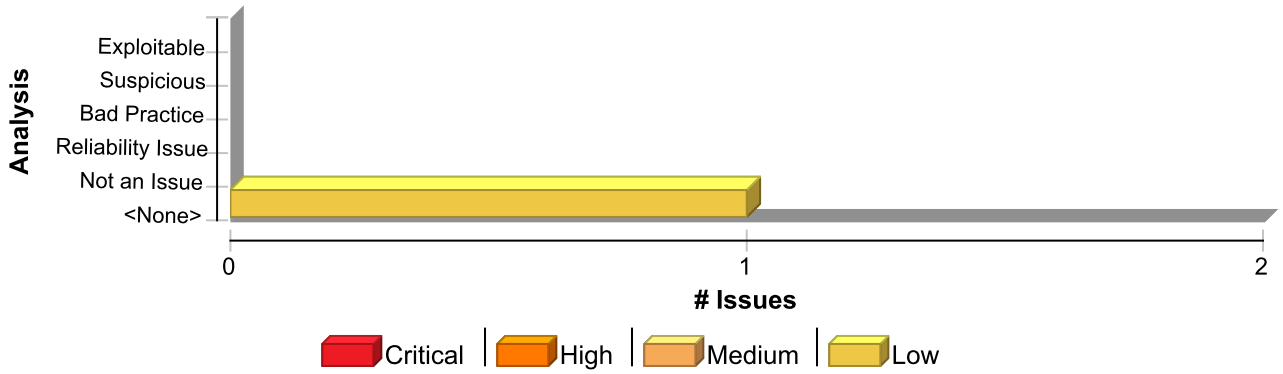
```
public void setUpCalls() {
  boolean firstCall = false;
  boolean secondCall = false;

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

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

Recommendation

In general, you should repair or remove unused code. It causes additional complexity and maintenance burden without

contributing to the functionality of the program.

Issue Summary



Engine Breakdown

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

Dead Code: Expression is Always true — Low

Package: akka.actor.testkit.typed.internal

internal/TestProbeImpl.scala, line 384 (Dead Code: Expression is Always true) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: poll()
File: internal/TestProbeImpl.scala:384
Taint Flags:

```
381 else null.asInstanceOf[A]
382 }
383
384 if (!failed) result
385 else {
386 Thread.sleep(t.toMillis)
387 poll((stop - now) min interval)
```

Insecure Randomness (9 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.

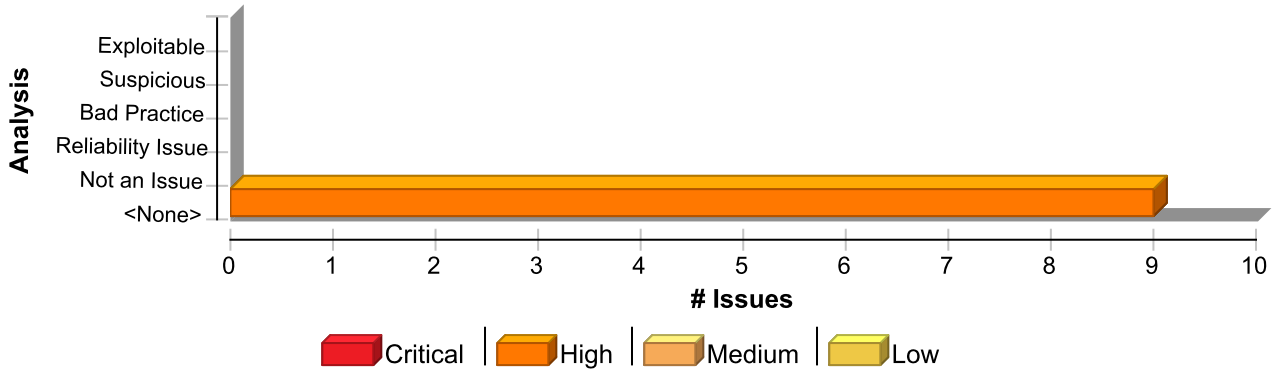
```
String GenerateReceiptURL(String baseUrl) {
    Random ranGen = new Random();
    ranGen.setSeed((new Date()).getTime());
    return (baseUrl + ranGen.nextInt(400000000) + ".html");
}
```

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	9	0	0	9
Total	**9**	**0**	**0**	**9**

Insecure Randomness — High

Package: akka.actor.testkit.typed.internal

internal/StubbedActorContext.scala, line 165 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: internalSpawnMessageAdapter()
File: internal/StubbedActorContext.scala:165
Taint Flags:

```
162 @InternalApi private[akka] def internalSpawnMessageAdapter[U](f: U => T, name: String): ActorRef[U] = {
163
164 val n = if (name != "") s"${childName.next()}-$name" else childName.next()
165 val p = (path / n).withUid(rnd().nextInt())
166 val i = new BehaviorTestKitImpl[U](system, p, BehaviorImpl.ignore)
167 _children += p.name -> i
168
```

internal/StubbedActorContext.scala, line 104 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: spawnAnonymous()
File: internal/StubbedActorContext.scala:104
Taint Flags:

Insecure Randomness — High

Package: akka.actor.testkit.typed.internal

internal/StubbedActorContext.scala, line 104 (Insecure Randomness) — High

```
101
102 override def spawnAnonymous[U](behavior: Behavior[U], props: Props = Props.empty): ActorRef[U] = {
103 checkCurrentActorThread()
104 val btk = new BehaviorTestKitImpl[U](system, (path / childName.next()).withUid(rnd().nextInt()), behavior)
105 _children += btk.context.self.path.name -> btk
106 btk.context.self
107 }
```

internal/StubbedActorContext.scala, line 113 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: spawn()
File: internal/StubbedActorContext.scala:113
Taint Flags:

```
110 _children.get(name) match {
111 case Some(_) => throw classic.InvalidActorNameException(s"actor name $name is already taken")
112 case None =>
113 val btk = new BehaviorTestKitImpl[U](system, (path / name).withUid(rnd().nextInt()), behavior)
114 _children += name -> btk
115 btk.context.self
116 }
```

internal/StubbedActorContext.scala, line 68 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: StubbedActorContext()
File: internal/StubbedActorContext.scala:68
Taint Flags:

```
65 extends ActorContextImpl[T] {
66
67 def this(system: ActorSystemStub, name: String, currentBehaviorProvider: () => Behavior[T]) = {
68 this(system, (system.path / name).withUid(rnd().nextInt()), currentBehaviorProvider)
69 }
70
```

Insecure Randomness — High

Package: akka.actor.testkit.typed.internal

internal/StubbedActorContext.scala, line 68 (Insecure Randomness) — High

```
71 def this(name: String, currentBehaviorProvider: () => Behavior[T]) = {
```

Package: akka.actor.testkit.typed.javadsl

javadsl/TestInbox.scala, line 24 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: create()
File: javadsl/TestInbox.scala:24
Taint Flags:

```
21 new TestInboxImpl((address / name).withUid(uid))
22 }
23 def create[T](): TestInbox[T] = {
24 val uid = ThreadLocalRandom.current().nextInt()
25 new TestInboxImpl((address / "inbox").withUid(uid))
26 }
27 }
```

javadsl/TestInbox.scala, line 20 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: create()
File: javadsl/TestInbox.scala:20
Taint Flags:

```
17 import akka.actor.testkit.typed.scaladsl.TestInbox.address
18
19 def create[T](name: String): TestInbox[T] = {
20 val uid = ThreadLocalRandom.current().nextInt()
21 new TestInboxImpl((address / name).withUid(uid))
22 }
23 def create[T](): TestInbox[T] = {
```

javadsl/BehaviorTestKit.scala, line 30 (Insecure Randomness) — High

Issue Details

Insecure Randomness — High

Package: akka.actor.testkit.typed.javadsl

javadsl/BehaviorTestKit.scala, line 30 (Insecure Randomness) — High

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: create()
File: javadsl/BehaviorTestKit.scala:30
Taint Flags:

```
27 @ApiMayChange
28 def create[T](initialBehavior: Behavior[T], name: String, config: Config): BehaviorTestKit[T] = {
29 val system = new ActorSystemStub("StubbedActorContext", config)
30 val uid = ThreadLocalRandom.current().nextInt()
31 new BehaviorTestKitImpl(system, (system.path / name).withUid(uid), initialBehavior)
32 }
33
```

Package: akka.actor.testkit.typed.scaladsl

scaladsl/BehaviorTestKit.scala, line 25 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: apply()
File: scaladsl/BehaviorTestKit.scala:25
Taint Flags:

```
22
23 def apply[T](initialBehavior: Behavior[T], name: String, config: Config): BehaviorTestKit[T] = {
24 val system = new ActorSystemStub("StubbedActorContext", config)
25 val uid = ThreadLocalRandom.current().nextInt()
26 new BehaviorTestKitImpl(system, (system.path / name).withUid(uid), initialBehavior)
27 }
28 def apply[T](initialBehavior: Behavior[T], name: String): BehaviorTestKit[T] = {
```

scaladsl/TestInbox.scala, line 19 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Insecure Randomness	High
Package: akka.actor.testkit.typed.scaladsl	
scaladsl/TestInbox.scala, line 19 (Insecure Randomness)	**High**

Sink: nextInt()
Enclosing Method: apply()
File: scaladsl/TestInbox.scala:19
Taint Flags:

```
16 @ApiMayChange
17 object TestInbox {
18  def apply[T](name: String = "inbox"): TestInbox[T] = {
19  val uid = ThreadLocalRandom.current().nextInt()
20  new TestInboxImpl((address / name).withUid(uid))
21  }
22
```

J2EE Bad Practices: Threads (4 issues)

Abstract

Thread management in a web application is forbidden in some circumstances and is always highly error prone.

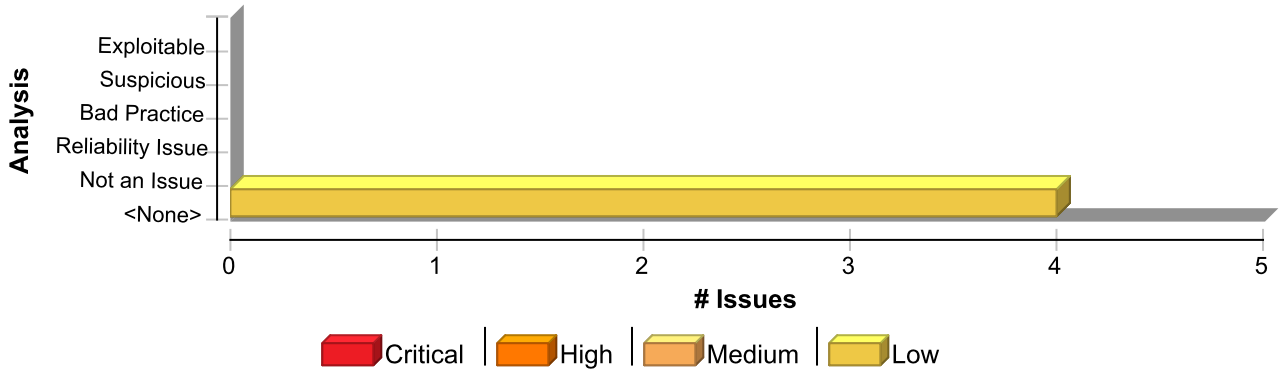
Explanation

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	4	0	0	4
Total	**4**	**0**	**0**	**4**

J2EE Bad Practices: Threads	Low
Package: akka.actor.testkit.typed.internal	
internal/LogbackUtil.scala, line 29 (J2EE Bad Practices: Threads)	**Low**

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: getLogbackLoggerInternal()
File: internal/LogbackUtil.scala:29
Taint Flags:

J2EE Bad Practices: Threads — Low

Package: akka.actor.testkit.typed.internal

internal/LogbackUtil.scala, line 29 (J2EE Bad Practices: Threads) — Low

```
26 case logger: ch.qos.logback.classic.Logger => logger
27 case _: org.slf4j.helpers.SubstituteLogger if count > 0 =>
28 // Wait for logging initialisation https://www.slf4j.org/codes.html#substituteLogger
29 Thread.sleep(50)
30 getLogbackLoggerInternal(loggerName, count - 1)
31 case null =>
32 throw new IllegalArgumentException(s"Couldn't find logger for [$loggerName].")
```

internal/TestProbeImpl.scala, line 386 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: sleep()
Enclosing Method: poll()
File: internal/TestProbeImpl.scala:386
Taint Flags:

```
383
384 if (!failed) result
385 else {
386 Thread.sleep(t.toMillis)
387 poll((stop - now) min interval)
388 }
389 }
```

internal/ActorSystemStub.scala, line 105 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: run()
Enclosing Method: newThread()
File: internal/ActorSystemStub.scala:105
Taint Flags:

```
102 override val startTime: Long = System.currentTimeMillis()
103 override def uptime: Long = System.currentTimeMillis() - startTime
104 override def threadFactory: java.util.concurrent.ThreadFactory = new ThreadFactory {
105 override def newThread(r: Runnable): Thread = new Thread(r)
106 }
107
```

J2EE Bad Practices: Threads — Low

Package: akka.actor.testkit.typed.internal

internal/ActorSystemStub.scala, line 105 (J2EE Bad Practices: Threads) — Low

```
108 override def printTree: String = "no tree for ActorSystemStub"
```

internal/ControlledExecutor.scala, line 22 (J2EE Bad Practices: Threads) — Low

Issue Details

Kingdom: Time and State
Scan Engine: SCA (Semantic)

Sink Details

Sink: run()
Enclosing Method: runOne()
File: internal/ControlledExecutor.scala:22
Taint Flags:

```
19
20 def queueSize: Int = tasks.size()
21
22 def runOne(): Unit = tasks.pop().run()
23
24 def runAll(): Unit = while (!tasks.isEmpty()) runOne()
25
```

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:

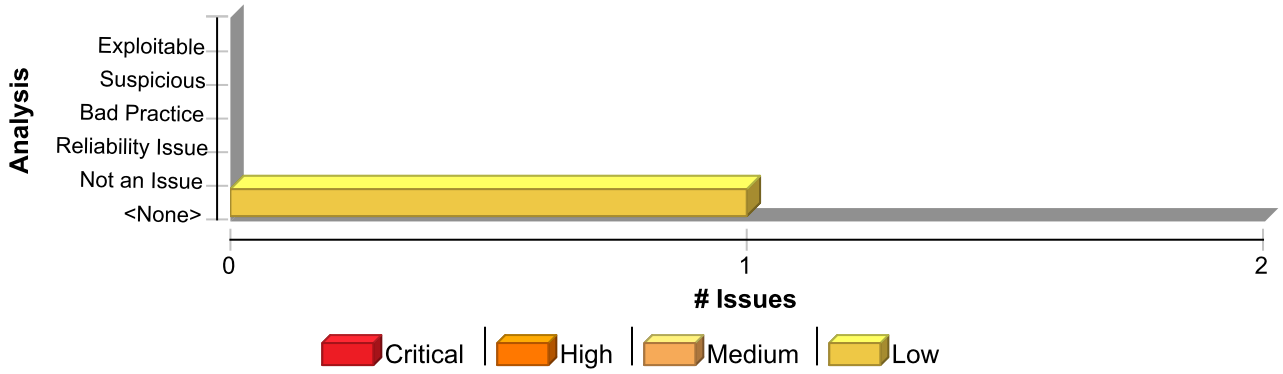
```
  try {
    doExchange();
  }
  catch (Exception e) {
    logger.error("doExchange failed", e);
  }
```

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: internal

internal/TestAppender.scala, line 99 (Poor Error Handling: Overly Broad Catch) — Low

Issue Details

Kingdom: Errors
Scan Engine: SCA (Structural)

Sink Details

Sink: CatchBlock
Enclosing Method: apply()
File: internal/TestAppender.scala:99
Taint Flags:

```
96  try {
97  f.apply(event)
98  } catch {
99  case _: Exception => false
100 })
101 }
102
```

Redundant Null Check (1 issue)

Abstract

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

Explanation

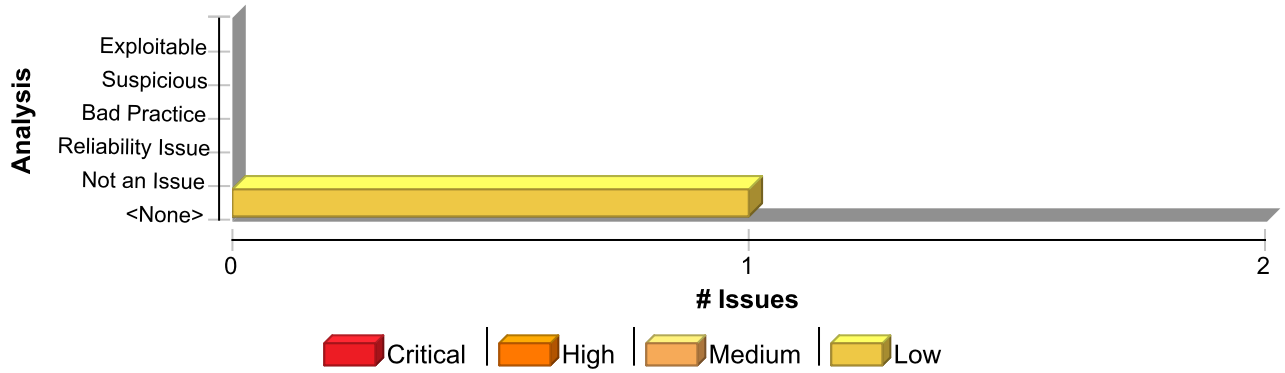
Null-pointer exceptions usually occur when one or more of the programmer's assumptions is violated. Specifically, dereference-after-check errors occur when a program makes an explicit check for `null`, but proceeds to dereference the object when it is known to be `null`. Errors of this type are often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

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

Recommendation

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

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Redundant Null Check	1	0	0	1
Total	**1**	**0**	**0**	**1**

Redundant Null Check — Low

Package: akka.actor.testkit.typed.internal

internal/TestProbeImpl.scala, line 343 (Redundant Null Check) — Low

Issue Details

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

Sink Details

Sink: Dereferenced : message
Enclosing Method: expectTerminated_internal()
File: internal/TestProbeImpl.scala:343
Taint Flags:

```
340 terminations.takeFirst
341 }
342 assert(message != null, s"timeout ($max) during expectTerminated waiting for actor [${actorRef.path}] to stop")
343 assert(message.ref == actorRef, s"expected [${actorRef.path}] to stop, but saw [${message.ref.path}] stop")
344 }
345
346 override def awaitAssert[A](a: => A, max: FiniteDuration, interval: FiniteDuration): A =
```

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;
}
...
```

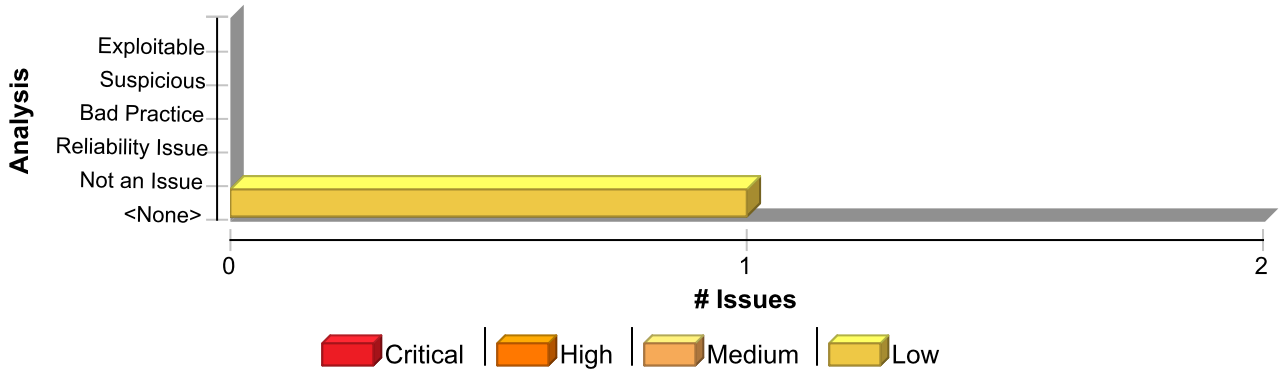
NFC Data Exchange Format (NDEF) message contains typed data, a URI, or a custom application payload. If the message contains information about the application, such as its name, MIME type, or device software version, this information could be leaked to an eavesdropper. In `Example 2`, Fortify Static 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.testkit.typed.internal

internal/ControlledExecutor.scala, line 31 (System Information Leak) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Semantic)

Sink Details

Sink: printStackTrace()
Enclosing Method: reportFailure()
File: internal/ControlledExecutor.scala:31
Taint Flags:

```
28 }
29
30 def reportFailure(cause: Throwable): Unit = {
31 cause.printStackTrace()
32 }
33 }
34
```