Fortify Standalone Report Generator

Developer Workbook

akka-stream-testkit

Table of Contents

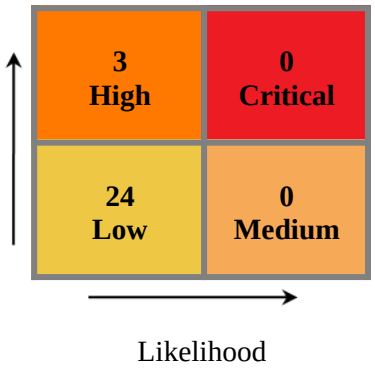
Executive Summary

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-stream-testkit 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-stream-testkit
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, 12:04 PM	**Engine Version:**	21.1.1.0009
Host Name:	Jacks-Work-MBP.local	**Certification:**	VALID
Number of Files:	18	**Lines of Code:**	1,288

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 / 8	0 / 8
Code Correctness: Non-Static Inner Class Implements Serializable	0	0	0	0 / 12	0 / 12
Dead Code: Expression is Always true	0	0	0	0 / 3	0 / 3
Insecure Randomness	0	0 / 3	0	0	0 / 3
System Information Leak	0	0	0	0 / 1	0 / 1

Results Outline

Code Correctness: Constructor Invokes Overridable Function (8 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");
    }
  }
}
```

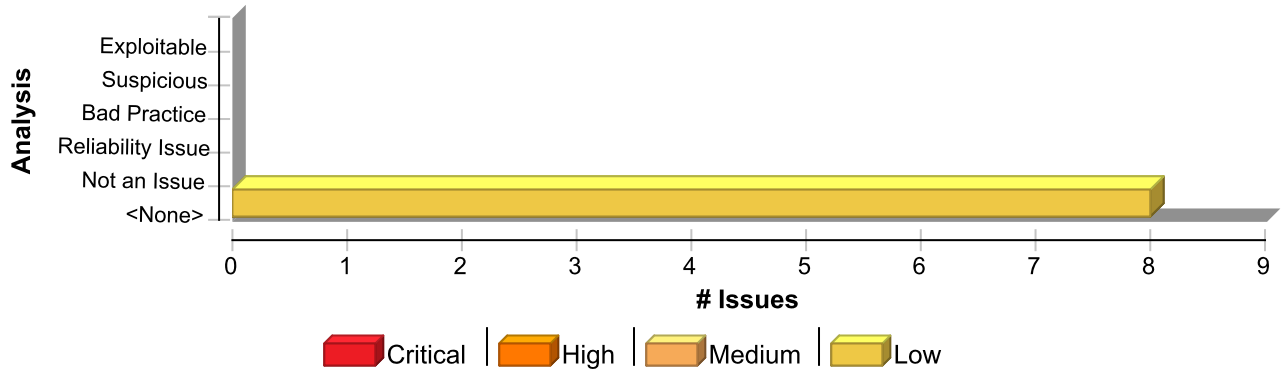
The code in `Example 1` prints "Attack successful!", since the `Attacker` class overrides the `validateUser()` function that is called from the constructor of the superclass `User`, and Java will first look in the subclass for functions called from the constructor.

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	8	0	0	8
Total	**8**	**0**	**0**	**8**

Code Correctness: Constructor Invokes Overridable Function	**Low**
Package: akka.stream.testkit	
main/scala/akka/stream/testkit/TestGraphStage.scala, line 118 (Code Correctness: Constructor Invokes Overridable Function)	**Low**
Issue Details	

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.stream.testkit

main/scala/akka/stream/testkit/TestGraphStage.scala, line 118 (Code Correctness: Constructor Invokes Overridable Function) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: out
Enclosing Method: TestSourceStage()
File: main/scala/akka/stream/testkit/TestGraphStage.scala:118
Taint Flags:

```
115 extends GraphStageWithMaterializedValue[SourceShape[T], M] {
116
117 val out = Outlet[T]("testSourceStage.out")
118 override val shape: SourceShape[T] = SourceShape.of(out)
119
120 override def createLogicAndMaterializedValue(inheritedAttributes: Attributes) = {
121 stageUnderTest.shape.out.id = out.id
```

main/scala/akka/stream/testkit/TestGraphStage.scala, line 54 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: in
Enclosing Method: TestSinkStage()
File: main/scala/akka/stream/testkit/TestGraphStage.scala:54
Taint Flags:

```
51 extends GraphStageWithMaterializedValue[SinkShape[T], M] {
52
53 val in = Inlet[T]("testSinkStage.in")
54 override val shape: SinkShape[T] = SinkShape.of(in)
55
56 override def createLogicAndMaterializedValue(inheritedAttributes: Attributes) = {
57 stageUnderTest.shape.in.id = in.id
```

test/scala/akka/stream/testkit/ChainSetup.scala, line 42 (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.stream.testkit

test/scala/akka/stream/testkit/ChainSetup.scala, line 42 (Code Correctness: Constructor Invokes Overridable Function) — Low

Sink Details

Sink: FunctionCall: s
Enclosing Method: ChainSetup()
File: test/scala/akka/stream/testkit/ChainSetup.scala:42
Taint Flags:

```
39 val upstream = TestPublisher.manualProbe[In]()
40 val downstream = TestSubscriber.probe[Out]()
41 private val s = Source.fromPublisher(upstream).via(stream(Flow[In].map(x => x).named("buh")))
42 val publisher = toPublisher(s, materializer)
43 val upstreamSubscription = upstream.expectSubscription()
44 publisher.subscribe(downstream)
45 val downstreamSubscription = downstream.expectSubscription()
```

test/scala/akka/stream/testkit/ChainSetup.scala, line 44 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: publisher
Enclosing Method: ChainSetup()
File: test/scala/akka/stream/testkit/ChainSetup.scala:44
Taint Flags:

```
41 private val s = Source.fromPublisher(upstream).via(stream(Flow[In].map(x => x).named("buh")))
42 val publisher = toPublisher(s, materializer)
43 val upstreamSubscription = upstream.expectSubscription()
44 publisher.subscribe(downstream)
45 val downstreamSubscription = downstream.expectSubscription()
46 }
47
```

test/scala/akka/stream/testkit/ChainSetup.scala, line 41 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: upstream
Enclosing Method: ChainSetup()

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.stream.testkit

test/scala/akka/stream/testkit/ChainSetup.scala, line 41 (Code Correctness: Constructor Invokes Overridable Function) — Low

File: test/scala/akka/stream/testkit/ChainSetup.scala:41
Taint Flags:

```
38
39 val upstream = TestPublisher.manualProbe[In]()
40 val downstream = TestSubscriber.probe[Out]()
41 private val s = Source.fromPublisher(upstream).via(stream(Flow[In].map(x => x).named("buh")))
42 val publisher = toPublisher(s, materializer)
43 val upstreamSubscription = upstream.expectSubscription()
44 publisher.subscribe(downstream)
```

test/scala/akka/stream/testkit/ChainSetup.scala, line 43 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: upstream
Enclosing Method: ChainSetup()
File: test/scala/akka/stream/testkit/ChainSetup.scala:43
Taint Flags:

```
40 val downstream = TestSubscriber.probe[Out]()
41 private val s = Source.fromPublisher(upstream).via(stream(Flow[In].map(x => x).named("buh")))
42 val publisher = toPublisher(s, materializer)
43 val upstreamSubscription = upstream.expectSubscription()
44 publisher.subscribe(downstream)
45 val downstreamSubscription = downstream.expectSubscription()
46 }
```

test/scala/akka/stream/testkit/ChainSetup.scala, line 44 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: downstream
Enclosing Method: ChainSetup()
File: test/scala/akka/stream/testkit/ChainSetup.scala:44
Taint Flags:

```
41 private val s = Source.fromPublisher(upstream).via(stream(Flow[In].map(x => x).named("buh")))
```

Code Correctness: Constructor Invokes Overridable Function	Low

Package: akka.stream.testkit

test/scala/akka/stream/testkit/ChainSetup.scala, line 44 (Code Correctness: Constructor Invokes Overridable Function) **Low**

```
42  val publisher = toPublisher(s, materializer)
43  val upstreamSubscription = upstream.expectSubscription()
44  publisher.subscribe(downstream)
45  val downstreamSubscription = downstream.expectSubscription()
46 }
47
```

test/scala/akka/stream/testkit/ChainSetup.scala, line 45 (Code Correctness: Constructor Invokes Overridable Function) **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: downstream
Enclosing Method: ChainSetup()
File: test/scala/akka/stream/testkit/ChainSetup.scala:45
Taint Flags:

```
42  val publisher = toPublisher(s, materializer)
43  val upstreamSubscription = upstream.expectSubscription()
44  publisher.subscribe(downstream)
45  val downstreamSubscription = downstream.expectSubscription()
46 }
47
48 undefined
```

Code Correctness: Non-Static Inner Class Implements Serializable (12 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

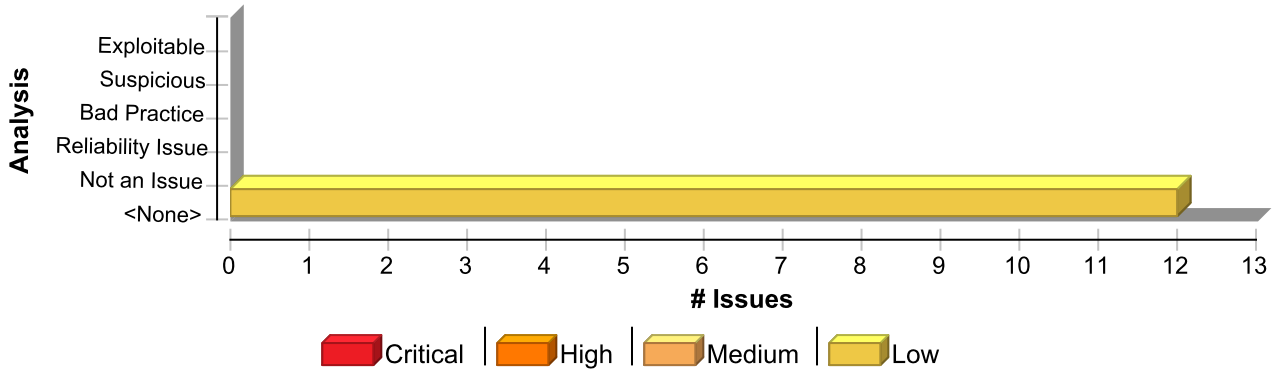
When using inner classes, they should not be serialized, or they should be changed to static-nested classes, since these do not have the drawbacks that non-static inner classes have when serialized. When a nested class is static it inherently has no association with instance variables (including those 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	12	0	0	12
Total	**12**	**0**	**0**	**12**

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

Package: akka.stream.impl.fusing

test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala, line 355 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: GraphInterpreterSpecKit$TestSetup$OnNext
File: test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala:355
Taint Flags:

```
352 case class OnComplete(source: GraphStageLogic) extends TestEvent
353 case class Cancel(source: GraphStageLogic, cause: Throwable) extends TestEvent
354 case class OnError(source: GraphStageLogic, cause: Throwable) extends TestEvent
355 case class OnNext(source: GraphStageLogic, elem: Any) extends TestEvent
356 case class RequestOne(source: GraphStageLogic) extends TestEvent
357 case class RequestAnother(source: GraphStageLogic) extends TestEvent
358 case class PreStart(source: GraphStageLogic) extends TestEvent
```

test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala, line 354 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: GraphInterpreterSpecKit$TestSetup$OnError

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

Package: akka.stream.impl.fusing

test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala, line 354 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

File: test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala:354
Taint Flags:

```
351
352 case class OnComplete(source: GraphStageLogic) extends TestEvent
353 case class Cancel(source: GraphStageLogic, cause: Throwable) extends TestEvent
354 case class OnError(source: GraphStageLogic, cause: Throwable) extends TestEvent
355 case class OnNext(source: GraphStageLogic, elem: Any) extends TestEvent
356 case class RequestOne(source: GraphStageLogic) extends TestEvent
357 case class RequestAnother(source: GraphStageLogic) extends TestEvent
```

test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala, line 586 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: GraphInterpreterSpecKit$OneBoundedSetupWithDecider$OnError
File: test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala:586
Taint Flags:

```
583
584 case object OnComplete extends TestEvent
585 case class Cancel(cause: Throwable) extends TestEvent
586 case class OnError(cause: Throwable) extends TestEvent
587 case class OnNext(elem: Any) extends TestEvent
588 case object RequestOne extends TestEvent
589 case object RequestAnother extends TestEvent
```

test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala, line 585 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: GraphInterpreterSpecKit$OneBoundedSetupWithDecider$Cancel
File: test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala:585
Taint Flags:

```
582 sealed trait TestEvent
583
584 case object OnComplete extends TestEvent
```

Code Correctness: Non-Static Inner Class Implements Serializable	Low

Package: akka.stream.impl.fusing

test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala, line 585 (Code Correctness: Non-Static Inner Class Implements Serializable) **Low**

```
585 case class Cancel(cause: Throwable) extends TestEvent
586 case class OnError(cause: Throwable) extends TestEvent
587 case class OnNext(elem: Any) extends TestEvent
588 case object RequestOne extends TestEvent
```

test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala, line 356 (Code Correctness: Non-Static Inner Class Implements Serializable) **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: GraphInterpreterSpecKit$TestSetup$RequestOne
File: test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala:356
Taint Flags:

```
353 case class Cancel(source: GraphStageLogic, cause: Throwable) extends TestEvent
354 case class OnError(source: GraphStageLogic, cause: Throwable) extends TestEvent
355 case class OnNext(source: GraphStageLogic, elem: Any) extends TestEvent
356 case class RequestOne(source: GraphStageLogic) extends TestEvent
357 case class RequestAnother(source: GraphStageLogic) extends TestEvent
358 case class PreStart(source: GraphStageLogic) extends TestEvent
359 case class PostStop(source: GraphStageLogic) extends TestEvent
```

test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala, line 352 (Code Correctness: Non-Static Inner Class Implements Serializable) **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: GraphInterpreterSpecKit$TestSetup$OnComplete
File: test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala:352
Taint Flags:

```
349 def source: GraphStageLogic
350 }
351
352 case class OnComplete(source: GraphStageLogic) extends TestEvent
353 case class Cancel(source: GraphStageLogic, cause: Throwable) extends TestEvent
354 case class OnError(source: GraphStageLogic, cause: Throwable) extends TestEvent
355 case class OnNext(source: GraphStageLogic, elem: Any) extends TestEvent
```

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

Package: akka.stream.impl.fusing

test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala, line 359 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: GraphInterpreterSpecKit$TestSetup$PostStop
File: test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala:359
Taint Flags:

```
356 case class RequestOne(source: GraphStageLogic) extends TestEvent
357 case class RequestAnother(source: GraphStageLogic) extends TestEvent
358 case class PreStart(source: GraphStageLogic) extends TestEvent
359 case class PostStop(source: GraphStageLogic) extends TestEvent
360
361 protected var lastEvent: Set[TestEvent] = Set.empty
362
```

test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala, line 587 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: GraphInterpreterSpecKit$OneBoundedSetupWithDecider$OnNext
File: test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala:587
Taint Flags:

```
584 case object OnComplete extends TestEvent
585 case class Cancel(cause: Throwable) extends TestEvent
586 case class OnError(cause: Throwable) extends TestEvent
587 case class OnNext(elem: Any) extends TestEvent
588 case object RequestOne extends TestEvent
589 case object RequestAnother extends TestEvent
590
```

test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala, line 358 (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.stream.impl.fusing

test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala, line 358 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: GraphInterpreterSpecKit$TestSetup$PreStart
File: test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala:358
Taint Flags:

```
355 case class OnNext(source: GraphStageLogic, elem: Any) extends TestEvent
356 case class RequestOne(source: GraphStageLogic) extends TestEvent
357 case class RequestAnother(source: GraphStageLogic) extends TestEvent
358 case class PreStart(source: GraphStageLogic) extends TestEvent
359 case class PostStop(source: GraphStageLogic) extends TestEvent
360
361 protected var lastEvent: Set[TestEvent] = Set.empty
```

test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala, line 353 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: GraphInterpreterSpecKit$TestSetup$Cancel
File: test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala:353
Taint Flags:

```
350 }
351
352 case class OnComplete(source: GraphStageLogic) extends TestEvent
353 case class Cancel(source: GraphStageLogic, cause: Throwable) extends TestEvent
354 case class OnError(source: GraphStageLogic, cause: Throwable) extends TestEvent
355 case class OnNext(source: GraphStageLogic, elem: Any) extends TestEvent
356 case class RequestOne(source: GraphStageLogic) extends TestEvent
```

test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala, line 357 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: GraphInterpreterSpecKit$TestSetup$RequestAnother
File: test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala:357
Taint Flags:

```
354 case class OnError(source: GraphStageLogic, cause: Throwable) extends TestEvent
355 case class OnNext(source: GraphStageLogic, elem: Any) extends TestEvent
```

Code Correctness: Non-Static Inner Class Implements Serializable	Low

Package: akka.stream.impl.fusing

test/scala/akka/stream/impl/fusing/GraphInterpreterSpecKit.scala, line 357 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

```
356  case class RequestOne(source: GraphStageLogic) extends TestEvent
357  case class RequestAnother(source: GraphStageLogic) extends TestEvent
358  case class PreStart(source: GraphStageLogic) extends TestEvent
359  case class PostStop(source: GraphStageLogic) extends TestEvent
360
```

Package: akka.stream.testkit

test/scala/akka/stream/testkit/ScriptedTest.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: ScriptedTest$ScriptException
File: test/scala/akka/stream/testkit/ScriptedTest.scala:29
Taint Flags:

```
26
27 trait ScriptedTest extends Matchers {
28
29  class ScriptException(msg: String) extends RuntimeException(msg)
30
31  def toPublisher[In, Out]: (Source[Out, _], Materializer) => Publisher[Out] =
32  (f, m) => f.runWith(Sink.asPublisher(false))(m)
```

Dead Code: Expression is Always true (3 issues)

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`.

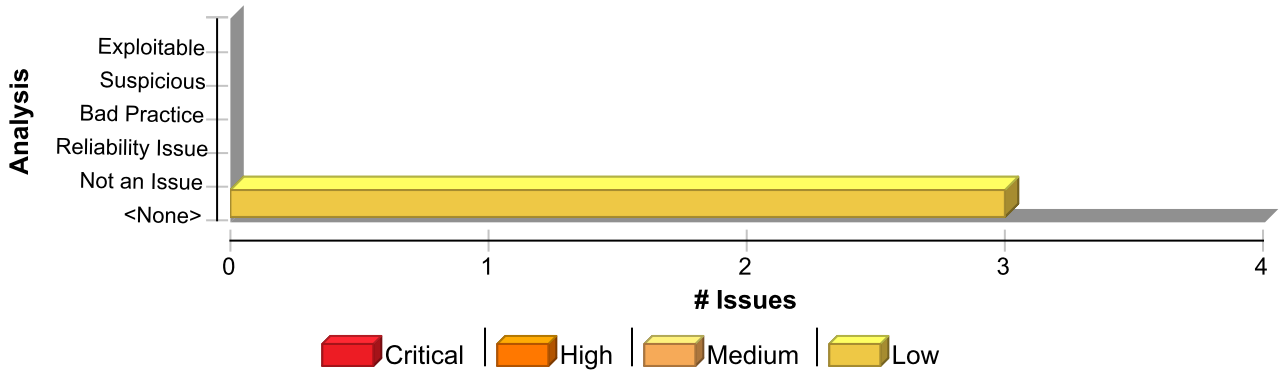
```
public void setUpCalls() {
  boolean firstCall = false;
  boolean secondCall = false;

  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	3	0	0	3
Total	**3**	**0**	**0**	**3**

Dead Code: Expression is Always true — Low

Package: akka.stream.testkit

main/scala/akka/stream/testkit/StreamTestKit.scala, line 573 (Dead Code: Expression is Always true) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: expectSubscriptionAndComplete()
File: main/scala/akka/stream/testkit/StreamTestKit.scala:573
Taint Flags:

```
570 */
571 def expectSubscriptionAndComplete(signalDemand: Boolean): Self = {
572 val sub = expectSubscription()
573 if (signalDemand) sub.request(1)
574 expectComplete()
575 self
576 }
```

main/scala/akka/stream/testkit/StreamTestKit.scala, line 545 (Dead Code: Expression is Always true) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Dead Code: Expression is Always true	Low

Package: akka.stream.testkit

main/scala/akka/stream/testkit/StreamTestKit.scala, line 545 (Dead Code: Expression is Always true)	Low

Sink Details

Sink: IfStatement
Enclosing Method: expectSubscriptionAndError()
File: main/scala/akka/stream/testkit/StreamTestKit.scala:545
Taint Flags:

```
542 */
543 def expectSubscriptionAndError(cause: Throwable, signalDemand: Boolean): Self = {
544 val sub = expectSubscription()
545 if (signalDemand) sub.request(1)
546 expectError(cause)
547 self
548 }
```

main/scala/akka/stream/testkit/StreamTestKit.scala, line 519 (Dead Code: Expression is Always true)	Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: expectSubscriptionAndError()
File: main/scala/akka/stream/testkit/StreamTestKit.scala:519
Taint Flags:

```
516 */
517 def expectSubscriptionAndError(signalDemand: Boolean): Throwable = {
518 val sub = expectSubscription()
519 if (signalDemand) sub.request(1)
520 expectError()
521 }
522
```

Insecure Randomness (3 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.

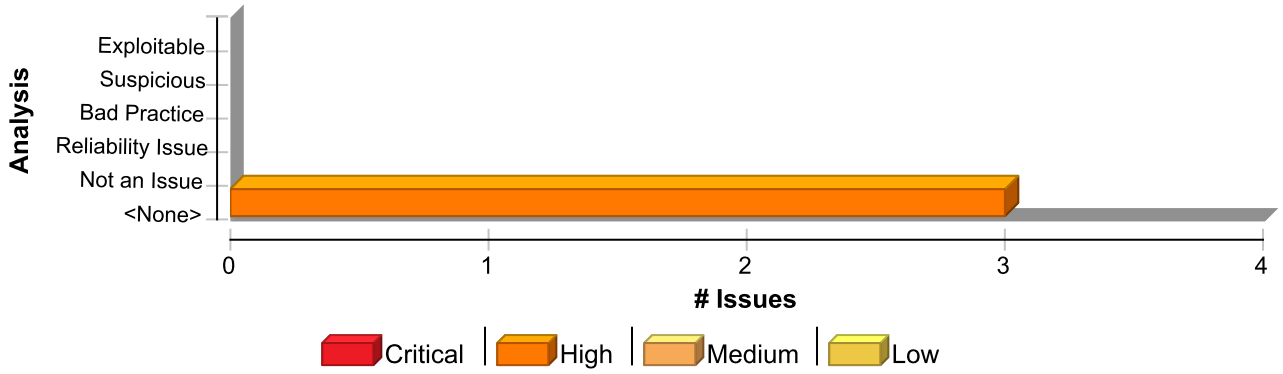
```
String GenerateReceiptURL(String baseUrl) {
    Random ranGen = new Random();
    ranGen.setSeed((new Date()).getTime());
    return (baseUrl + ranGen.nextInt(400000000) + ".html");
}
```

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	3	0	0	3
Total	**3**	**0**	**0**	**3**

Insecure Randomness — High

Package: akka.stream.testkit

test/scala/akka/stream/testkit/ScriptedTest.scala, line 150 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: getNextDemand()
File: test/scala/akka/stream/testkit/ScriptedTest.scala:150
Taint Flags:

```
147 remainingDemand = 0
148 1
149 } else {
150 val demand = ThreadLocalRandom.current().nextInt(1, max)
151 remainingDemand -= demand
152 demand
153 }
```

test/scala/akka/stream/testkit/ScriptedTest.scala, line 138 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: ScriptedTest$ScriptRunner()
File: test/scala/akka/stream/testkit/ScriptedTest.scala:138
Taint Flags:

Insecure Randomness — High

Package: akka.stream.testkit

test/scala/akka/stream/testkit/ScriptedTest.scala, line 138 (Insecure Randomness) — High

```
135
136 var _debugLog = Vector.empty[String]
137 var currentScript = script
138 var remainingDemand = script.expectedOutputs.size + ThreadLocalRandom.current().nextInt(1, maximumOverrun)
139 debugLog(s"starting with remainingDemand=$remainingDemand")
140 var pendingRequests = 0L
141 var outstandingDemand = 0L
```

test/scala/akka/stream/testkit/ScriptedTest.scala, line 201 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextBoolean()
Enclosing Method: doRun()
File: test/scala/akka/stream/testkit/ScriptedTest.scala:201
Taint Flags:

```
198 if (!currentScript.completed) {
199 val nextIdle = if (shakeIt()) 0 else idleRounds + 1
200
201 val tieBreak = ThreadLocalRandom.current().nextBoolean()
202 if (mayProvideInput && (!mayRequestMore || tieBreak)) {
203 val (input, nextScript) = currentScript.provideInput
204 debugLog(s"test environment produces [${input}]")
```

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;
}
...
```

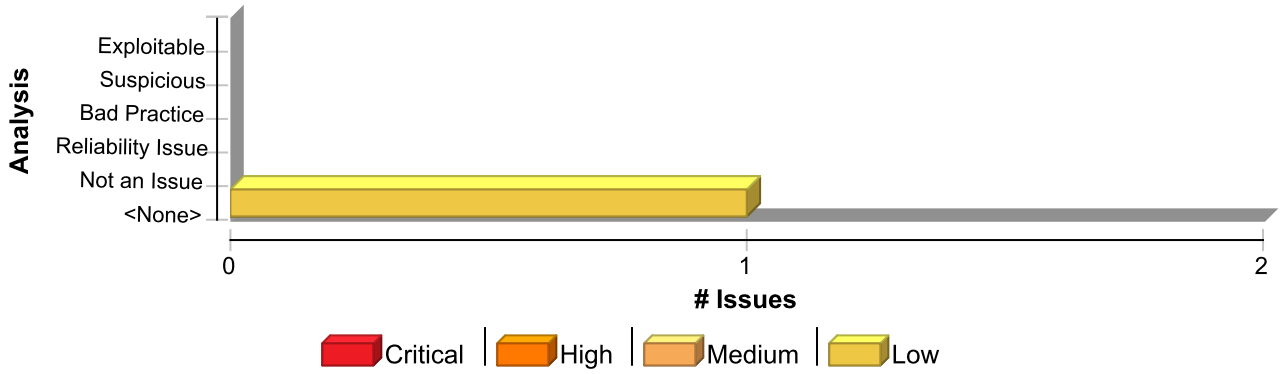
NFC Data Exchange Format (NDEF) message contains typed data, a URI, or a custom application payload. If the message contains information about the application, such as its name, MIME type, or device software version, this information could be leaked to an eavesdropper. In `Example 2`, Fortify Static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

main/scala/akka/stream/testkit/StreamTestKit.scala, line 310 (System Information Leak) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Semantic)

Sink Details

Sink: printStackTrace()
Enclosing Method: toString()
File: main/scala/akka/stream/testkit/StreamTestKit.scala:310
Taint Flags:

```
307 val str = new StringWriter
308 val out = new PrintWriter(str)
309 out.print("OnError(")
310 cause.printStackTrace(out)
311 out.print(")")
312 str.toString
313 }
```