Fortify Standalone Report Generator

Developer Workbook

akka-actor-typed

Table of Contents

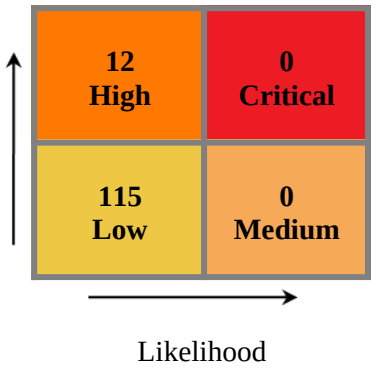
Executive Summary

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-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-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, 11:14 AM	**Engine Version:**	21.1.1.0009
Host Name:	Jacks-Work-MBP.local	**Certification:**	VALID
Number of Files:	86	**Lines of Code:**	4,588

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: Class Does Not Implement equals	0	0	0	0 / 36	0 / 36
Code Correctness: Constructor Invokes Overridable Function	0	0	0	0 / 14	0 / 14
Code Correctness: Erroneous String Compare	0	0	0	0 / 1	0 / 1
Code Correctness: Invalid Call to Object.equals()	0	0	0	0 / 1	0 / 1
Code Correctness: Non-Static Inner Class Implements Serializable	0	0	0	0 / 53	0 / 53
Dead Code: Expression is Always true	0	0	0	0 / 1	0 / 1
Insecure Randomness	0	0 / 3	0	0	0 / 3
Null Dereference	0	0 / 6	0	0	0 / 6
Redundant Null Check	0	0	0	0 / 1	0 / 1
System Information Leak: Internal	0	0	0	0 / 8	0 / 8
Weak SecurityManager Check: Overridable Method	0	0 / 3	0	0	0 / 3

Results Outline

Code Correctness: Class Does Not Implement equals (36 issues)

Abstract

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

Explanation

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

```
public class AccountGroup
{
    private int gid;

    public int getGid()
    {
        return gid;
    }

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

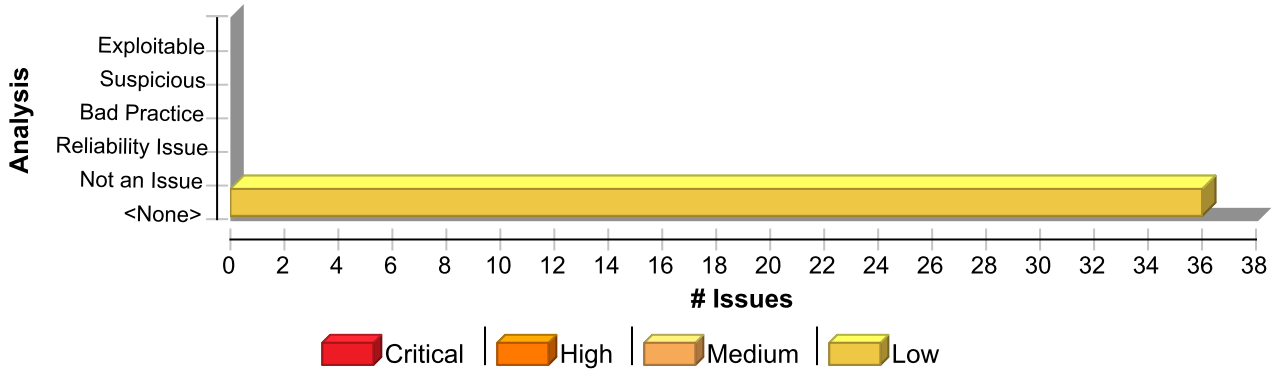
Recommendation

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

```
public class AccountGroup
{
    private int gid;

    public int getGid()
    {
        return gid;
    }

    public void setGid(int newGid)
```

```
    {
        gid = newGid;
    }

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

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Code Correctness: Class Does Not Implement equals	36	0	0	36
Total	**36**	**0**	**0**	**36**

Code Correctness: Class Does Not Implement equals	**Low**
Package: akka.actor.typed	
scala/akka/actor/typed/Behavior.scala, line 239 (Code Correctness: Class Does Not Implement equals)	**Low**

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor.typed

scala/akka/actor/typed/Behavior.scala, line 239 (Code Correctness: Class Does Not Implement equals) — Low

Sink: FunctionCall: equals
Enclosing Method: interpretSignal()
File: scala/akka/actor/typed/Behavior.scala:239
Taint Flags:

```
236 val result = interpret(behavior, ctx, signal, isSignal = true)
237 // we need to throw here to allow supervision of deathpact exception
238 signal match {
239 case Terminated(ref) if result == BehaviorImpl.UnhandledBehavior => throw DeathPactException(ref)
240 case _ => result
241 }
242 }
```

scala/akka/actor/typed/MessageAndSignals.scala, line 106 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: equals()
File: scala/akka/actor/typed/MessageAndSignals.scala:106
Taint Flags:

```
103 override def hashCode(): Int = ref.hashCode()
104
105 override def equals(obj: Any): Boolean = obj match {
106 case ChildFailed(`ref`, `cause`) => true
107 case _ => false
108 }
109 }
```

scala/akka/actor/typed/MessageAndSignals.scala, line 106 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: equals()
File: scala/akka/actor/typed/MessageAndSignals.scala:106
Taint Flags:

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor.typed

scala/akka/actor/typed/MessageAndSignals.scala, line 106 (Code Correctness: Class Does Not Implement equals) — Low

```
103 override def hashCode(): Int = ref.hashCode()
104
105 override def equals(obj: Any): Boolean = obj match {
106 case ChildFailed(`ref`, `cause`) => true
107 case _ => false
108 }
109 }
```

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

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

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

```
78 override def hashCode(): Int = ref.hashCode()
79
80 override def equals(obj: Any): Boolean = obj match {
81 case Terminated(`ref`) => true
82 case _ => false
83 }
84 }
```

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala, line 713 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: retryRequest()
File: scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala:713
Taint Flags:

```
710
711 // in case the Request or the SequencedMessage triggering the Request is lost
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala, line 713 (Code Correctness: Class Does Not Implement equals) — Low

```
712 private def retryRequest(s: State[A]): State[A] = {
713 if (s.producerController == context.system.deadLetters) {
714 s
715 } else {
716 val newRequestedSeqNr = if (resendLost) s.requestedSeqNr else s.receivedSeqNr + flowControlWindow / 2
```

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

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: updatedRegistering()
File: scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala:104
Taint Flags:

```
101 def updatedRegistering(seqMsg: SequencedMessage[A]): Option[ActorRef[ProducerController.Command[A]]] = {
102 registering match {
103 case None => None
104 case s @ Some(reg) => if (seqMsg.producerController == reg) None else s
105 }
106 }
107
```

scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala, line 246 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: scheduleNext()
File: scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala:246
Taint Flags:

```
243 case f: FiniteDuration => f
244 case _ => maxBackoff
245 }
246 if (newInterval != _interval) {
247 _interval = newInterval
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala, line 246 (Code Correctness: Class Does Not Implement equals) — Low

```
248  timers.startTimerWithFixedDelay(Retry, _interval)
249  }
```

scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala, line 406 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: logChangedProducer()
File: scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala:406
Taint Flags:

```
403  }
404
405  private def logChangedProducer(s: State[A], seqMsg: SequencedMessage[A]): Unit = {
406  if (s.producerController == context.system.deadLetters) {
407  context.log.debugN(
408  "Associated with new ProducerController [{}], seqNr [{}].",
409  seqMsg.producerController,
```

scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala, line 99 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: isProducerChanged()
File: scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala:99
Taint Flags:

```
96  seqMsg.seqNr == receivedSeqNr + 1
97
98  def isProducerChanged(seqMsg: SequencedMessage[A]): Boolean =
99  seqMsg.producerController != producerController || receivedSeqNr == 0
100
101  def updatedRegistering(seqMsg: SequencedMessage[A]): Option[ActorRef[ProducerController.Command[A]]] = {
102  registering match {
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala, line 663 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: receiveStart()
File: scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala:663
Taint Flags:

```
660 start: Start[A],
661 nextBehavior: State[A] => Behavior[InternalCommand]): Behavior[InternalCommand] = {
662 ConsumerControllerImpl.enforceLocalConsumer(start.deliverTo)
663 if (start.deliverTo == s.consumer) {
664 nextBehavior(s)
665 } else {
666 // if consumer is restarted it may send Start again
```

scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala, line 677 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: receiveRegisterToProducerController()
File: scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala:677
Taint Flags:

```
674 s: State[A],
675 reg: RegisterToProducerController[A],
676 nextBehavior: State[A] => Behavior[InternalCommand]): Behavior[InternalCommand] = {
677 if (reg.producerController != s.producerController) {
678 context.log.debug2(
679 "Register to new ProducerController [{}], previous was [{}].",
680 reg.producerController,
```

scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala, line 648 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala, line 648 (Code Correctness: Class Does Not Implement equals) — Low

Sink Details

Sink: FunctionCall: equals
Enclosing Method: receiveRetry()
File: scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala:648
Taint Flags:

```
645
646 private def receiveRetry(s: State[A], nextBehavior: () => Behavior[InternalCommand]): Behavior[InternalCommand] = {
647 retryTimer.scheduleNext()
648 if (retryTimer.interval() != retryTimer.minBackoff)
649 context.log.debug("Schedule next retry in [{} ms]", retryTimer.interval().toMillis)
650 s.registering match {
651 case None => nextBehavior()
```

Package: akka.actor.typed.internal

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

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: equals()
File: scala/akka/actor/typed/internal/ActorRefImpl.scala:39
Taint Flags:

```
36 * Equals takes path and the unique id of the actor cell into account.
37 */
38 final override def equals(that: Any): Boolean = that match {
39 case other: ActorRef[_] => path.uid == other.path.uid && path == other.path
40 case _ => false
41 }
42
```

scala/akka/actor/typed/internal/InterceptorImpl.scala, line 98 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor.typed.internal

scala/akka/actor/typed/internal/InterceptorImpl.scala, line 98 (Code Correctness: Class Does Not Implement equals) — Low

Enclosing Method: deduplicate()
File: scala/akka/actor/typed/internal/InterceptorImpl.scala:98
Taint Flags:

```
95
96 private def deduplicate(interceptedResult: Behavior[I], ctx: TypedActorContext[O]): Behavior[O] = {
97 val started = Behavior.start(interceptedResult, ctx.asInstanceOf[TypedActorContext[I]])
98 if (started == BehaviorImpl.UnhandledBehavior || started == BehaviorImpl.SameBehavior || !Behavior.isAlive(started)) {
99 started.unsafeCast[O]
100 } else {
101 // returned behavior could be nested in setups, so we need to start before we deduplicate
```

scala/akka/actor/typed/internal/InterceptorImpl.scala, line 218 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: isSame()
File: scala/akka/actor/typed/internal/InterceptorImpl.scala:218
Taint Flags:

```
215 override def isSame(other: BehaviorInterceptor[Any, Any]): Boolean = other match {
216 // If they use the same pf instance we can allow it, to have one way to workaround defining
217 // "recursive" narrowed behaviors.
218 case TransformMessagesInterceptor(`matcher`) => true
219 case TransformMessagesInterceptor(otherMatcher) =>
220 // there is no safe way to allow this
221 throw new IllegalStateException(
```

scala/akka/actor/typed/internal/StashBufferImpl.scala, line 220 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: interpretUnstashedMessages()
File: scala/akka/actor/typed/internal/StashBufferImpl.scala:220
Taint Flags:

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor.typed.internal

scala/akka/actor/typed/internal/StashBufferImpl.scala, line 220 (Code Correctness: Class Does Not Implement equals) — Low

```
217 val actualInitialBehavior =
218 if (Behavior.isUnhandled(started))
219 throw new IllegalArgumentException("Cannot unstash with unhandled as starting behavior")
220 else if (started == BehaviorImpl.same) {
221 currentBehaviorWhenUnstashInProgress match {
222 case OptionVal.Some(c) => c
223 case _ => ctx.asScala.currentBehavior
```

scala/akka/actor/typed/internal/InterceptorImpl.scala, line 98 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: deduplicate()
File: scala/akka/actor/typed/internal/InterceptorImpl.scala:98
Taint Flags:

```
95
96 private def deduplicate(interceptedResult: Behavior[I], ctx: TypedActorContext[O]): Behavior[O] = {
97 val started = Behavior.start(interceptedResult, ctx.asInstanceOf[TypedActorContext[I]])
98 if (started == BehaviorImpl.UnhandledBehavior || started == BehaviorImpl.SameBehavior || !Behavior.isAlive(started)) {
99 started.unsafeCast[O]
100 } else {
101 // returned behavior could be nested in setups, so we need to start before we deduplicate
```

scala/akka/actor/typed/internal/InterceptorImpl.scala, line 190 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: isSame()
File: scala/akka/actor/typed/internal/InterceptorImpl.scala:190
Taint Flags:

```
187
188 // only once in the same behavior stack
189 override def isSame(other: BehaviorInterceptor[Any, Any]): Boolean = other match {
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor.typed.internal

scala/akka/actor/typed/internal/InterceptorImpl.scala, line 190 (Code Correctness: Class Does Not Implement equals) — Low

```
190 case a: LogMessagesInterceptor => a.opts == opts
191 case _ => false
192 }
193 }
```

scala/akka/actor/typed/internal/StashBufferImpl.scala, line 199 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: interpretOne()
File: scala/akka/actor/typed/internal/StashBufferImpl.scala:199
Taint Flags:

```
196 }
197
198 val actualNext =
199 if (interpretResult == BehaviorImpl.same) b2
200 else if (Behavior.isUnhandled(interpretResult)) {
201 ctx.asScala.onUnhandled(message)
202 b2
```

scala/akka/actor/typed/internal/InterceptorImpl.scala, line 134 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: isSame()
File: scala/akka/actor/typed/internal/InterceptorImpl.scala:134
Taint Flags:

```
131
132 // only once to the same actor in the same behavior stack
133 override def isSame(other: BehaviorInterceptor[Any, Any]): Boolean = other match {
134 case MonitorInterceptor(`actorRef`) => true
135 case _ => false
136 }
```

Code Correctness: Class Does Not Implement equals	Low

Package: akka.actor.typed.internal

scala/akka/actor/typed/internal/InterceptorImpl.scala, line 134 (Code Correctness: Class Does Not Implement equals) **Low**

```
137
```

Package: akka.actor.typed.internal.adapter

scala/akka/actor/typed/internal/adapter/ActorAdapter.scala, line 337 (Code Correctness: Class Does Not Implement equals) **Low**

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: receiveSignal()
File: scala/akka/actor/typed/internal/adapter/ActorAdapter.scala:337
Taint Flags:

```
334 override def receive(ctx: TypedActorContext[T], msg: T): Behavior[T] =
335 throw new IllegalStateException("Stopping, should never receieve a message")
336 override def receiveSignal(ctx: TypedActorContext[T], msg: Signal): Behavior[T] = {
337 if (msg != PostStop)
338 throw new IllegalArgumentException(
339 s"The ComposedStoppingBehavior should only ever receive a PostStop signal, but received $msg")
340 // first pass the signal to the previous behavior, so that it and potential interceptors
```

scala/akka/actor/typed/internal/adapter/ActorContextAdapter.scala, line 82 (Code Correctness: Class Does Not Implement equals) **Low**

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: stop()
File: scala/akka/actor/typed/internal/adapter/ActorContextAdapter.scala:82
Taint Flags:

```
79 // child that was already stopped
80 }
81 }
82 } else if (self == child) {
83 throw new IllegalArgumentException(
84 "Only direct children of an actor can be stopped through the actor context, " +
85 s"but you tried to stop [$self] by passing its ActorRef to the `stop` method. " +
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor.typed.internal.adapter

scala/akka/actor/typed/internal/adapter/ActorContextAdapter.scala, line 76 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: stop()
File: scala/akka/actor/typed/internal/adapter/ActorContextAdapter.scala:76
Taint Flags:

```
73 cell.removeFunctionRef(f)
74 case c =>
75 classicContext.child(child.path.name) match {
76 case Some(`c`) =>
77 classicContext.stop(c)
78 case _ =>
79 // child that was already stopped
```

scala/akka/actor/typed/internal/adapter/ActorContextAdapter.scala, line 69 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: stop()
File: scala/akka/actor/typed/internal/adapter/ActorContextAdapter.scala:69
Taint Flags:

```
66
67 override def stop[U](child: ActorRef[U]): Unit = {
68 checkCurrentActorThread()
69 if (child.path.parent == self.path) { // only if a direct child
70 toClassic(child) match {
71 case f: akka.actor.FunctionRef =>
72 val cell = classicContext.asInstanceOf[akka.actor.ActorCell]
```

Package: akka.actor.typed.internal.receptionist

scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala, line 74 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor.typed.internal.receptionist

scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala, line 74 (Code Correctness: Class Does Not Implement equals) — Low

Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: serviceInstances()
File: scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala:74
Taint Flags:

```
71 def isForKey(key: ServiceKey[_]): Boolean = key == this.key
72
73 def serviceInstances[M](key: ServiceKey[M]): Set[ActorRef[M]] = {
74 if (key != this.key)
75 throw new IllegalArgumentException(s"Wrong key [$key] used, must use listing key [${this.key}]")
76 _serviceInstances.asInstanceOf[Set[ActorRef[M]]]
77 }
```

scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala, line 40 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: serviceInstance()
File: scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala:40
Taint Flags:

```
37 extends Receptionist.Registered {
38 def isForKey(key: ServiceKey[_]): Boolean = key == this.key
39 def serviceInstance[M](key: ServiceKey[M]): ActorRef[M] = {
40 if (key != this.key)
41 throw new IllegalArgumentException(s"Wrong key [$key] used, must use listing key [${this.key}]")
42 _serviceInstance.asInstanceOf[ActorRef[M]]
43 }
```

scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala, line 53 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor.typed.internal.receptionist

scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala, line 53 (Code Correctness: Class Does Not Implement equals) — Low

Sink: FunctionCall: equals
Enclosing Method: serviceInstance()
File: scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala:53
Taint Flags:

```
50 extends Receptionist.Deregistered {
51 def isForKey(key: ServiceKey[_]): Boolean = key == this.key
52 def serviceInstance[M](key: ServiceKey[M]): ActorRef[M] = {
53 if (key != this.key)
54 throw new IllegalArgumentException(s"Wrong key [$key] used, must use listing key [${this.key}]")
55 _serviceInstance.asInstanceOf[ActorRef[M]]
56 }
```

scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala, line 38 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: isForKey()
File: scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala:38
Taint Flags:

```
35
36 final case class Registered[T] private[akka] (key: ServiceKey[T], _serviceInstance: ActorRef[T])
37 extends Receptionist.Registered {
38 def isForKey(key: ServiceKey[_]): Boolean = key == this.key
39 def serviceInstance[M](key: ServiceKey[M]): ActorRef[M] = {
40 if (key != this.key)
41 throw new IllegalArgumentException(s"Wrong key [$key] used, must use listing key [${this.key}]")
```

scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala, line 71 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: isForKey()
File: scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala:71
Taint Flags:

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor.typed.internal.receptionist

scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala, line 71 (Code Correctness: Class Does Not Implement equals) — Low

```
68 servicesWereAddedOrRemoved: Boolean)
69 extends Receptionist.Listing {
70
71 def isForKey(key: ServiceKey[_]): Boolean = key == this.key
72
73 def serviceInstances[M](key: ServiceKey[M]): Set[ActorRef[M]] = {
74 if (key != this.key)
```

scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala, line 51 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: isForKey()
File: scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala:51
Taint Flags:

```
48
49 final case class Deregistered[T] private[akka] (key: ServiceKey[T], _serviceInstance: ActorRef[T])
50 extends Receptionist.Deregistered {
51 def isForKey(key: ServiceKey[_]): Boolean = key == this.key
52 def serviceInstance[M](key: ServiceKey[M]): ActorRef[M] = {
53 if (key != this.key)
54 throw new IllegalArgumentException(s"Wrong key [$key] used, must use listing key [${this.key}]")
```

scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala, line 83 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: allServiceInstances()
File: scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala:83
Taint Flags:

```
80 serviceInstances(key).asJava
81
82 override def allServiceInstances[M](key: ServiceKey[M]): Set[ActorRef[M]] = {
```

Code Correctness: Class Does Not Implement equals — Low

Package: akka.actor.typed.internal.receptionist

scala/akka/actor/typed/internal/receptionist/ReceptionistMessages.scala, line 83 (Code Correctness: Class Does Not Implement equals) — Low

```
83 if (key != this.key)
84 throw new IllegalArgumentException(s"Wrong key [$key] used, must use listing key [${this.key}]")
85 _allServiceInstances.asInstanceOf[Set[ActorRef[M]]]
86 }
```

Package: akka.actor.typed.javadsl

scala/akka/actor/typed/javadsl/ReceiveBuilder.scala, line 132 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: test()
File: scala/akka/actor/typed/javadsl/ReceiveBuilder.scala:132
Taint Flags:

```
129 */
130 def onSignalEquals(signal: Signal, handler: Creator[Behavior[T]]): ReceiveBuilder[T] =
131 withSignal(signal.getClass, OptionVal.Some(new JPredicate[Signal] {
132 override def test(param: Signal): Boolean = param == signal
133 }), new JFunction[Signal, Behavior[T]] {
134 override def apply(param: Signal): Behavior[T] = handler.create()
135 })
```

Package: scala.akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 574 (Code Correctness: Class Does Not Implement equals) — Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:574
Taint Flags:

```
571 }
572
573 val newState2 = removedWorkers.foldLeft(newState) { (acc, c) =>
```

Code Correctness: Class Does Not Implement equals	Low

Package: scala.akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 574 (Code Correctness: Class Does Not Implement equals) — Low

```
574 acc.out.find { case (_, outState) => outState.consumerController == c } match {
575 case Some((key, outState)) =>
576 context.log.debug2("Deregistered worker [{}], with producerId [{}].", c, key)
577 context.stop(outState.producerController)
```

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

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala:600
Taint Flags:

```
597 case msg: SequencedMessage[_] =>
598 flightRecorder.consumerReceivedPreviousInProgress(msg.producerId, msg.seqNr, stashBuffer.size + 1)
599 val expectedSeqNr = seqMsg.seqNr + stashBuffer.size + 1
600 if (msg.seqNr < expectedSeqNr && msg.producerController == seqMsg.producerController) {
601 flightRecorder.consumerDuplicate(msg.producerId, expectedSeqNr, msg.seqNr)
602 context.log.debug("Received duplicate SequencedMessage seqNr [{}].", msg.seqNr)
603 } else if (stashBuffer.isFull) {
```

Package: scala.akka.actor.typed.internal.pubsub

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

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/actor/typed/internal/pubsub/TopicImpl.scala:125
Taint Flags:

```
122
123 case Unsubscribe(subscriber) =>
124 context.unwatch(subscriber)
125 localSubscribers = localSubscribers.filterNot(_ == subscriber)
126 if (localSubscribers.isEmpty) {
```

Code Correctness: Class Does Not Implement equals	Low

Package: scala.akka.actor.typed.internal.pubsub

scala/akka/actor/typed/internal/pubsub/TopicImpl.scala, line 125 (Code Correctness: Class Does Not Implement equals)	Low

```
127 context.log.debug("Last local subscriber [{}] unsubscribed, deregistering from receptionist", subscriber)
128 // that was the lost subscriber, deregister from the receptionist
```

Package: scala.akka.actor.typed.javadsl

scala/akka/actor/typed/javadsl/BehaviorBuilder.scala, line 137 (Code Correctness: Class Does Not Implement equals)	Low

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: apply()
File: scala/akka/actor/typed/javadsl/BehaviorBuilder.scala:137
Taint Flags:

```
134 * @return a new behavior builder with the specified handling appended
135 */
136 def onSignalEquals(signal: Signal, handler: Creator[Behavior[T]]): BehaviorBuilder[T] =
137 withSignal(signal.getClass, OptionVal.Some(_.equals(signal)), (_: Signal) => handler.create())
138
139 private def withMessage[M <: T](
140 clazz: OptionVal[Class[M]],
```

Code Correctness: Constructor Invokes Overridable Function (14 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.

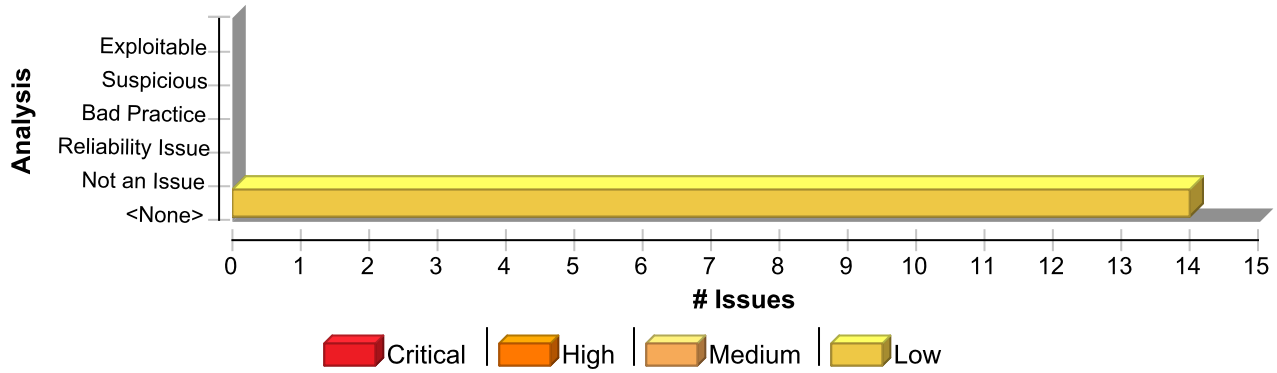
Recommendation

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	14	0	0	14
Total	**14**	**0**	**0**	**14**

Code Correctness: Constructor Invokes Overridable Function	**Low**
Package: akka.actor.typed	
scala/akka/actor/typed/SupervisorStrategy.scala, line 40 (Code Correctness: Constructor Invokes Overridable Function)	**Low**
Issue Details	

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor.typed

scala/akka/actor/typed/SupervisorStrategy.scala, line 40 (Code Correctness: Constructor Invokes Overridable Function) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: SupervisorStrategy$Stop
Enclosing Method: SupervisorStrategy()
File: scala/akka/actor/typed/SupervisorStrategy.scala:40
Taint Flags:

```
37 /**
38 * Stop the actor
39 */
40 val stop: SupervisorStrategy = Stop(loggingEnabled = true, logLevel = Level.ERROR)
41
42 /**
43 * Scala API: It supports exponential back-off between the given `minBackoff` and
```

scala/akka/actor/typed/SupervisorStrategy.scala, line 35 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: SupervisorStrategy$Restart
Enclosing Method: SupervisorStrategy()
File: scala/akka/actor/typed/SupervisorStrategy.scala:35
Taint Flags:

```
32 * (restarting would be dangerous as it could lead to an infinite restart-loop)
33 */
34 val restart: RestartSupervisorStrategy =
35 Restart(maxRestarts = -1, withinTimeRange = Duration.Zero)
36
37 /**
38 * Stop the actor
```

scala/akka/actor/typed/ActorRef.scala, line 97 (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.typed

scala/akka/actor/typed/ActorRef.scala, line 97 (Code Correctness: Constructor Invokes Overridable Function) — Low

Sink Details

Sink: FunctionCall: toAddress
Enclosing Method: SerializedActorRef()
File: scala/akka/actor/typed/ActorRef.scala:97
Taint Flags:

```
94  import akka.serialization.JavaSerializer.currentSystem
95
96  def this(actorRef: ActorRef[T]) =
97  this(SerializedActorRef.toAddress(actorRef))
98
99  @throws(classOf[java.io.ObjectStreamException])
100 def readResolve(): AnyRef = currentSystem.value match {
```

scala/akka/actor/typed/ActorSystem.scala, line 321 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: typedConfig
Enclosing Method: Settings()
File: scala/akka/actor/typed/ActorSystem.scala:321
Taint Flags:

```
318
319 private val typedConfig = config.getConfig("akka.actor.typed")
320
321 val RestartStashCapacity: Int =
322 typedConfig.getInt("restart-stash-capacity").requiring(_ >= 0, "restart-stash-capacity must be >= 0")
323 }
324 * Be careful to not schedule any operations, such as `onComplete`, on the dispatchers (`ExecutionContext`)
```

scala/akka/actor/typed/SupervisorStrategy.scala, line 25 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: SupervisorStrategy$Resume
Enclosing Method: SupervisorStrategy()

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor.typed

scala/akka/actor/typed/SupervisorStrategy.scala, line 25 (Code Correctness: Constructor Invokes Overridable Function) — Low

File: scala/akka/actor/typed/SupervisorStrategy.scala:25
Taint Flags:

```
22 * If the actor behavior is deferred and throws an exception on startup the actor is stopped
23 * (restarting would be dangerous as it could lead to an infinite restart-loop)
24 */
25 val resume: SupervisorStrategy = Resume(loggingEnabled = true, logLevel = Level.ERROR)
26
27 /**
28 * Restart immediately without any limit on number of restart retries. A limit can be
```

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 284 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: producerControllerSettings
Enclosing Method: WorkPullingProducerControllerImpl()
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:284
Taint Flags:

```
281
282 private val producerControllerSettings = settings.producerControllerSettings
283 private val traceEnabled = context.log.isTraceEnabled
284 private val durableQueueAskTimeout: Timeout = producerControllerSettings.durableQueueRequestTimeout
285 private val workerAskTimeout: Timeout = settings.internalAskTimeout
286
287 private val workerRequestNextAdapter: ActorRef[ProducerController.RequestNext[A]] =
```

Package: akka.actor.typed.internal

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

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: _first
Enclosing Method: StashBufferImpl()

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor.typed.internal

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

File: scala/akka/actor/typed/internal/StashBufferImpl.scala:49
Taint Flags:

```
46
47 import StashBufferImpl.Node
48
49 private var _size: Int = if (_first eq null) 0 else 1
50
51 private var currentBehaviorWhenUnstashInProgress: OptionVal[Behavior[T]] = OptionVal.None
52
```

Package: akka.actor.typed.internal.adapter

scala/akka/actor/typed/internal/adapter/ActorSystemAdapter.scala, line 82 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: provider
Enclosing Method: ActorSystemAdapter()
File: scala/akka/actor/typed/internal/adapter/ActorSystemAdapter.scala:82
Taint Flags:

```
79 // Members declared in akka.actor.typed.ActorSystem
80 override def deadLetters[U]: ActorRef[U] = ActorRefAdapter(system.deadLetters)
81
82 private val cachedIgnoreRef: ActorRef[Nothing] = ActorRefAdapter(provider.ignoreRef)
83 override def ignoreRef[U]: ActorRef[U] = cachedIgnoreRef.unsafeUpcast[U]
84
85 override def dispatchers: Dispatchers = new Dispatchers {
```

scala/akka/actor/typed/internal/adapter/ActorSystemAdapter.scala, line 100 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: classicSystem
Enclosing Method: ActorSystemAdapter()
File: scala/akka/actor/typed/internal/adapter/ActorSystemAdapter.scala:100
Taint Flags:

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor.typed.internal.adapter

scala/akka/actor/typed/internal/adapter/ActorSystemAdapter.scala, line 100 (Code Correctness: Constructor Invokes Overridable Function) — **Low**

```
97  override val log: Logger = LoggerFactory.getLogger(classOf[ActorSystem[_]])
98  override def logConfiguration(): Unit = classicSystem.logConfiguration()
99  override def name: String = classicSystem.name
100 override val scheduler: Scheduler = new SchedulerAdapter(classicSystem.scheduler)
101 override def settings: Settings = new Settings(classicSystem.settings)
102 override def startTime: Long = classicSystem.startTime
103 override def threadFactory: java.util.concurrent.ThreadFactory = system.threadFactory
```

Package: akka.actor.typed.internal.pubsub

scala/akka/actor/typed/internal/pubsub/TopicImpl.scala, line 74 (Code Correctness: Constructor Invokes Overridable Function) — **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: receptionist
Enclosing Method: TopicImpl()
File: scala/akka/actor/typed/internal/pubsub/TopicImpl.scala:74
Taint Flags:

```
71 private var localSubscribers = Set.empty[ActorRef[T]]
72
73 private val receptionist = context.system.receptionist
74 private val receptionistAdapter = context.messageAdapter[Receptionist.Listing] {
75 case topicServiceKey.Listing(topics) => TopicInstancesUpdated(topics)
76 case _ => throw new IllegalArgumentException() // FIXME exhaustiveness check fails on receptionist listing match
77 }
```

scala/akka/actor/typed/internal/pubsub/TopicImpl.scala, line 74 (Code Correctness: Constructor Invokes Overridable Function) — **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: topicServiceKey
Enclosing Method: TopicImpl()
File: scala/akka/actor/typed/internal/pubsub/TopicImpl.scala:74
Taint Flags:

```
71 private var localSubscribers = Set.empty[ActorRef[T]]
72
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.actor.typed.internal.pubsub

scala/akka/actor/typed/internal/pubsub/TopicImpl.scala, line 74 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
73 private val receptionist = context.system.receptionist
74 private val receptionistAdapter = context.messageAdapter[Receptionist.Listing] {
75 case topicServiceKey.Listing(topics) => TopicInstancesUpdated(topics)
76 case _ => throw new IllegalArgumentException() // FIXME exhaustiveness check fails on receptionist listing match
77 }
```

scala/akka/actor/typed/internal/pubsub/TopicImpl.scala, line 74 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: receptionistAdapter
Enclosing Method: TopicImpl()
File: scala/akka/actor/typed/internal/pubsub/TopicImpl.scala:74
Taint Flags:

```
71 private var localSubscribers = Set.empty[ActorRef[T]]
72
73 private val receptionist = context.system.receptionist
74 private val receptionistAdapter = context.messageAdapter[Receptionist.Listing] {
75 case topicServiceKey.Listing(topics) => TopicInstancesUpdated(topics)
76 case _ => throw new IllegalArgumentException() // FIXME exhaustiveness check fails on receptionist listing match
77 }
```

Package: akka.actor.typed.internal.routing

scala/akka/actor/typed/internal/routing/PoolRouterImpl.scala, line 80 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: akka$actor$typed$internal$routing$PoolRouterImpl$$onRouteesChanged
Enclosing Method: PoolRouterImpl()
File: scala/akka/actor/typed/internal/routing/PoolRouterImpl.scala:80
Taint Flags:

```
77 context.watch(child)
78 child
79 }
80 onRouteesChanged()
```

Code Correctness: Constructor Invokes Overridable Function	Low

Package: akka.actor.typed.internal.routing

scala/akka/actor/typed/internal/routing/PoolRouterImpl.scala, line 80 (Code Correctness: Constructor Invokes Overridable Function)	Low

```
81
82  private def onRouteesChanged(): Unit = {
83  val children = context.children.toSet.asInstanceOf[Set[ActorRef[T]]]
```

Package: akka.actor.typed.javadsl

scala/akka/actor/typed/javadsl/ReceiveBuilder.scala, line 180 (Code Correctness: Constructor Invokes Overridable Function)	Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: akka$actor$typed$javadsl$ReceiveBuilder$$adapterExceptionSignalHandler
Enclosing Method: ReceiveBuilder()
File: scala/akka/actor/typed/javadsl/ReceiveBuilder.scala:180
Taint Flags:

```
177
178  /** INTERNAL API */
179  @InternalApi
180  private val _defaultSignalHandlers = adapterExceptionSignalHandler :: Nil
181
182  /** INTERNAL API */
183  @InternalApi
```

Code Correctness: Erroneous String Compare (1 issue)

Abstract

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

Explanation

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

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

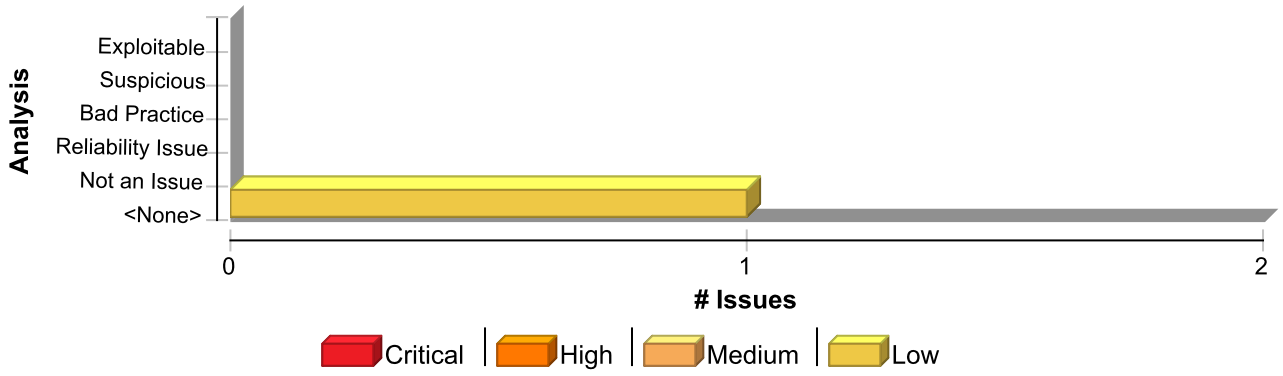
The `==` and `!=` operators will only behave as expected when they are used to compare strings contained in objects that are equal. The most common way for this to occur is for the strings to be interned, whereby the strings are added to a pool of objects maintained by the `String` class. Once a string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

Recommendation

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

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

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Code Correctness: Erroneous String Compare	1	0	0	1
Total	**1**	**0**	**0**	**1**

Code Correctness: Erroneous String Compare	Low
Package: akka.actor.typed.delivery	
scala/akka/actor/typed/delivery/ProducerController.scala, line 158 (Code Correctness: Erroneous String Compare)	**Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: apply()
File: scala/akka/actor/typed/delivery/ProducerController.scala:158
Taint Flags:

```
155 * `akka.reliable-delivery.producer-controller`.
156 */
157 def apply(config: Config): Settings = {
158 val chunkLargeMessagesBytes = toRootLowerCase(config.getString("chunk-large-messages")) match {
159 case "off" => 0
160 case _ =>
161 config.getBytes("chunk-large-messages").requiring(_ <= Int.MaxValue, "Too large chunk-large-messages.").toInt
```

Code Correctness: Invalid Call to Object.equals() (1 issue)

Abstract

The program calls `Object.equals()` on an array instead of `java.util.Arrays.equals()`.

Explanation

Calling `Object.equals()` against an array is a mistake in most cases, since this will check the equality of the arrays' addresses, instead of the equality of the arrays' elements, and should usually be replaced by `java.util.Arrays.equals()`. **Example 1:** The following tries to check two arrays using the `Object.equals()` function.

```
...
  int[] arr1 = new int[10];
  int[] arr2 = new int[10];
  ...
  if (arr1.equals(arr2)){
    //treat arrays as if identical elements
  }
...
```

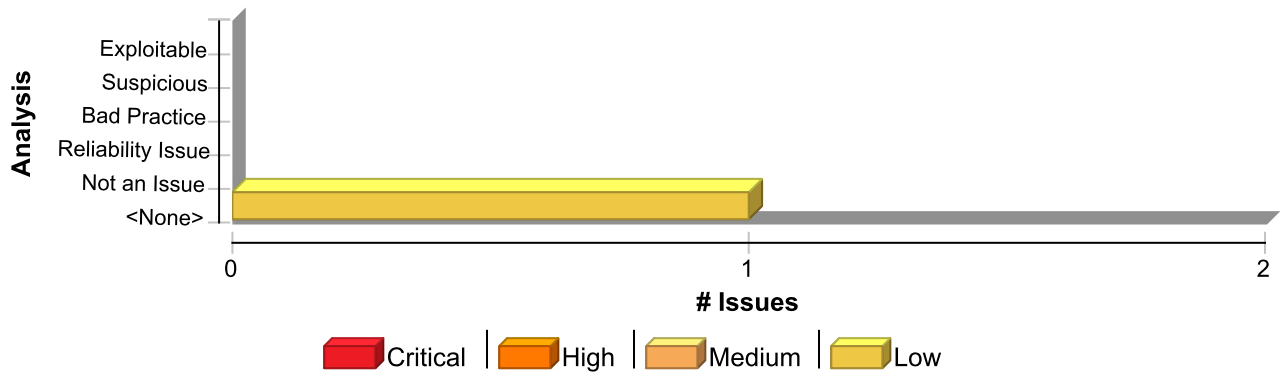
This will almost always result in code that is never executed, unless at some point there is an assignment of one array to the other.

Recommendation

When using arrays `Object.equals()` can be used to verify that two array objects are pointing to the same address, which is a small use case. If trying to verify that two arrays contain the same elements, in the same order, `java.util.Arrays.equals()` should instead be used. **Example 2:** The following fixes `Example 1` using `java.util.Arrays.equals()`.

```
import java.util.Arrays;
...
  int[] arr1 = new int[10];
  int[] arr2 = new int[10];
  ...
  if (Arrays.equals(arr1, arr2)){
    //treat arrays as if identical elements
  }
...
```

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Code Correctness: Invalid Call to Object.equals()	1	0	0	1
Total	**1**	**0**	**0**	**1**

Code Correctness: Invalid Call to Object.equals() — Low

Package: akka.actor.typed.internal.routing

scala/akka/actor/typed/internal/routing/RoutingLogic.scala, line 66 (Code Correctness: Invalid Call to Object.equals()) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals
Enclosing Method: routeesUpdated()
File: scala/akka/actor/typed/internal/routing/RoutingLogic.scala:66
Taint Flags:

```
63 var idx = 0
64 while (idx < currentRoutees.length &&
65 idx < sortedNewRoutees.length &&
66 currentRoutees(idx) == sortedNewRoutees(idx)) {
67 idx += 1
68 }
69 idx
```

Code Correctness: Non-Static Inner Class Implements Serializable (53 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

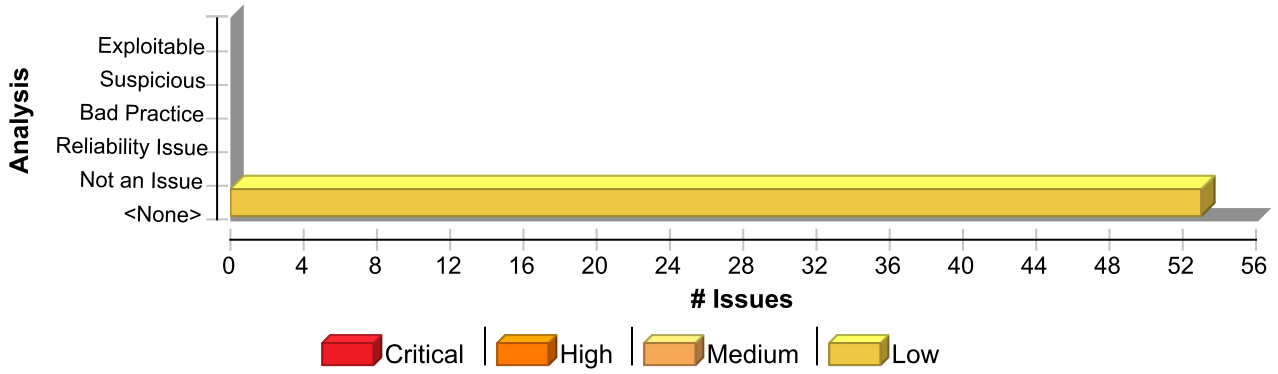
When using inner classes, they should not be serialized, or they should be changed to static-nested classes, since these do not have the drawbacks that non-static inner classes have when serialized. When a nested class is static it inherently has no association with instance variables (including those 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	53	0	0	53
Total	**53**	**0**	**0**	**53**

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

Package: akka.actor.typed

scala/akka/actor/typed/SupervisorStrategy.scala, line 147 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: SupervisorStrategy$Restart
File: scala/akka/actor/typed/SupervisorStrategy.scala:147
Taint Flags:

```
144 /**
145 * INTERNAL API
146 */
147 @InternalApi private[akka] final case class Restart(
148 maxRestarts: Int,
149 withinTimeRange: FiniteDuration,
150 loggingEnabled: Boolean = true,
```

scala/akka/actor/typed/SupervisorStrategy.scala, line 125 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: SupervisorStrategy$Stop

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

Package: akka.actor.typed

scala/akka/actor/typed/SupervisorStrategy.scala, line 125 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

File: scala/akka/actor/typed/SupervisorStrategy.scala:125
Taint Flags:

```
122 /**
123 * INTERNAL API
124 */
125 @InternalApi private[akka] case class Stop(loggingEnabled: Boolean, logLevel: Level) extends SupervisorStrategy {
126 override def withLoggingEnabled(enabled: Boolean) =
127 copy(loggingEnabled = enabled)
128 override def withLogLevel(level: Level): SupervisorStrategy =
```

scala/akka/actor/typed/LogOptions.scala, line 53 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: LogOptions$LogOptionsImpl
File: scala/akka/actor/typed/LogOptions.scala:53
Taint Flags:

```
50 * INTERNAL API
51 */
52 @InternalApi
53 private[akka] final case class LogOptionsImpl(enabled: Boolean, level: Level, logger: Option[Logger])
54 extends LogOptions {
55
56 /**
```

scala/akka/actor/typed/SupervisorStrategy.scala, line 115 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: SupervisorStrategy$Resume
File: scala/akka/actor/typed/SupervisorStrategy.scala:115
Taint Flags:

```
112 /**
113 * INTERNAL API
114 */
```

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

Package: akka.actor.typed

scala/akka/actor/typed/SupervisorStrategy.scala, line 115 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
115 @InternalApi private[akka] case class Resume(loggingEnabled: Boolean, logLevel: Level) extends SupervisorStrategy {
116 override def withLoggingEnabled(enabled: Boolean): SupervisorStrategy =
117 copy(loggingEnabled = enabled)
118 override def withLogLevel(level: Level): SupervisorStrategy =
```

scala/akka/actor/typed/SupervisorStrategy.scala, line 179 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: SupervisorStrategy$Backoff
File: scala/akka/actor/typed/SupervisorStrategy.scala:179
Taint Flags:

```
176 /**
177 * INTERNAL API
178 */
179 @InternalApi private[akka] final case class Backoff(
180 minBackoff: FiniteDuration,
181 maxBackoff: FiniteDuration,
182 randomFactor: Double,
```

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 99 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ProducerControllerImpl$Resend
File: scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala:99
Taint Flags:

```
96 confirmedSeqNr <= requestUpToSeqNr,
97 s"confirmedSeqNr [$confirmedSeqNr] should be <= requestUpToSeqNr [$requestUpToSeqNr]")
98 }
99 final case class Resend(fromSeqNr: SeqNr) extends InternalCommand with DeliverySerializable with DeadLetterSuppression
100 final case class Ack(confirmedSeqNr: SeqNr)
101 extends InternalCommand
102 with DeliverySerializable
```

Code Correctness: Non-Static Inner Class Implements Serializable	Low

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 99 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 111 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ProducerControllerImpl$LoadStateFailed
File: scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala:111
Taint Flags:

```
108 private case object SendChunk extends InternalCommand
109
110 private case class LoadStateReply[A](state: DurableProducerQueue.State[A]) extends InternalCommand
111 private case class LoadStateFailed(attempt: Int) extends InternalCommand
112 private case class StoreMessageSentReply(ack: DurableProducerQueue.StoreMessageSentAck)
113 private case class StoreMessageSentFailed[A](messageSent: DurableProducerQueue.MessageSent[A], attempt: Int)
114 extends InternalCommand
```

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 64 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: WorkPullingProducerControllerImpl$StoreMessageSentCompleted
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:64
Taint Flags:

```
61 private case class StoreMessageSentReply(ack: DurableProducerQueue.StoreMessageSentAck)
62 private case class StoreMessageSentFailed[A](messageSent: DurableProducerQueue.MessageSent[A], attempt: Int)
63 extends InternalCommand
64 private case class StoreMessageSentCompleted[A](messageSent: DurableProducerQueue.MessageSent[A])
65 extends InternalCommand
66 private case object DurableQueueTerminated extends InternalCommand
67
```

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 55 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

Issue Details

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

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 55 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: WorkPullingProducerControllerImpl$AskTimeout
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:55
Taint Flags:

```
52 private final case class WorkerRequestNext[A](next: ProducerController.RequestNext[A]) extends InternalCommand
53
54 private final case class Ack(outKey: OutKey, confirmedSeqNr: OutSeqNr) extends InternalCommand
55 private final case class AskTimeout(outKey: OutKey, outSeqNr: OutSeqNr) extends InternalCommand
56
57 private case object RegisterConsumerDone extends InternalCommand
58
```

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 105 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: WorkPullingProducerControllerImpl$Msg
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:105
Taint Flags:

```
102 private final case class CurrentWorkers[A](workers: Set[ActorRef[ConsumerController.Command[A]]])
103 extends InternalCommand
104
105 private final case class Msg[A](msg: A, wasStashed: Boolean, replyTo: Option[ActorRef[Done]]) extends InternalCommand
106
107 private final case class ResendDurableMsg[A](
108 msg: A,
```

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 105 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

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

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 105 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: ProducerControllerImpl$Msg
File: scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala:105
Taint Flags:

```
102 with DeliverySerializable
103 with DeadLetterSuppression
104
105 private case class Msg[A](msg: A) extends InternalCommand
106 private case object ResendFirst extends InternalCommand
107 case object ResendFirstUnconfirmed extends InternalCommand
108 private case object SendChunk extends InternalCommand
```

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 107 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: WorkPullingProducerControllerImpl$ResendDurableMsg
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:107
Taint Flags:

```
104
105 private final case class Msg[A](msg: A, wasStashed: Boolean, replyTo: Option[ActorRef[Done]]) extends InternalCommand
106
107 private final case class ResendDurableMsg[A](
108 msg: A,
109 oldConfirmationQualifier: ConfirmationQualifier,
110 oldSeqNr: TotalSeqNr)
```

scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala, line 84 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ConsumerControllerImpl$State
File: scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala:84
Taint Flags:

```
81
82 private final case class ConsumerTerminated(consumer: ActorRef[_]) extends InternalCommand
```

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

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala, line 84 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
83
84 private final case class State[A](
85 producerController: ActorRef[ProducerControllerImpl.InternalCommand],
86 producerId: String,
87 consumer: ActorRef[ConsumerController.Delivery[A]],
```

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 54 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: WorkPullingProducerControllerImpl$Ack
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:54
Taint Flags:

```
51
52 private final case class WorkerRequestNext[A](next: ProducerController.RequestNext[A]) extends InternalCommand
53
54 private final case class Ack(outKey: OutKey, confirmedSeqNr: OutSeqNr) extends InternalCommand
55 private final case class AskTimeout(outKey: OutKey, outSeqNr: OutSeqNr) extends InternalCommand
56
57 private case object RegisterConsumerDone extends InternalCommand
```

scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala, line 82 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ConsumerControllerImpl$ConsumerTerminated
File: scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala:82
Taint Flags:

```
79
80 private case object Retry extends InternalCommand
81
82 private final case class ConsumerTerminated(consumer: ActorRef[_]) extends InternalCommand
83
84 private final case class State[A](
85 producerController: ActorRef[ProducerControllerImpl.InternalCommand],
```

Code Correctness: Non-Static Inner Class Implements Serializable	Low

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/ConsumerControllerImpl.scala, line 82 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 91 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ProducerControllerImpl$Request
File: scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala:91
Taint Flags:

```
88 /** For commands defined in public ProducerController */
89 trait UnsealedInternalCommand extends InternalCommand
90
91 final case class Request(confirmedSeqNr: SeqNr, requestUpToSeqNr: SeqNr, supportResend: Boolean, viaTimeout: Boolean)
92 extends InternalCommand
93 with DeliverySerializable
94 with DeadLetterSuppression {
```

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 61 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: WorkPullingProducerControllerImpl$StoreMessageSentReply
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:61
Taint Flags:

```
58
59 private case class LoadStateReply[A](state: DurableProducerQueue.State[A]) extends InternalCommand
60 private case class LoadStateFailed(attempt: Int) extends InternalCommand
61 private case class StoreMessageSentReply(ack: DurableProducerQueue.StoreMessageSentAck)
62 private case class StoreMessageSentFailed[A](messageSent: DurableProducerQueue.MessageSent[A], attempt: Int)
63 extends InternalCommand
64 private case class StoreMessageSentCompleted[A](messageSent: DurableProducerQueue.MessageSent[A])
```

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 83 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

Issue Details

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

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 83 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: WorkPullingProducerControllerImpl$State
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:83
Taint Flags:

```
80 msg: A,
81 replyTo: Option[ActorRef[Done]])
82
83 private final case class State[A](
84 currentSeqNr: TotalSeqNr, // only updated when durableQueue is enabled
85 workers: Set[ActorRef[ConsumerController.Command[A]]],
86 out: Map[OutKey, OutState[A]],
```

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 102 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: WorkPullingProducerControllerImpl$CurrentWorkers
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:102
Taint Flags:

```
99 private case class HandOver(oldConfirmationQualifier: ConfirmationQualifier, oldSeqNr: TotalSeqNr)
100
101 // registration of workers via Receptionist
102 private final case class CurrentWorkers[A](workers: Set[ActorRef[ConsumerController.Command[A]]])
103 extends InternalCommand
104
105 private final case class Msg[A](msg: A, wasStashed: Boolean, replyTo: Option[ActorRef[Done]]) extends InternalCommand
```

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 119 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

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

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 119 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: ProducerControllerImpl$State
File: scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala:119
Taint Flags:

```
116 extends InternalCommand
117 private case object DurableQueueTerminated extends InternalCommand
118
119 private final case class State[A](
120 requested: Boolean,
121 currentSeqNr: SeqNr,
122 confirmedSeqNr: SeqNr,
```

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 52 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: WorkPullingProducerControllerImpl$WorkerRequestNext
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:52
Taint Flags:

```
49 private type OutSeqNr = Long
50 private type OutKey = String
51
52 private final case class WorkerRequestNext[A](next: ProducerController.RequestNext[A]) extends InternalCommand
53
54 private final case class Ack(outKey: OutKey, confirmedSeqNr: OutSeqNr) extends InternalCommand
55 private final case class AskTimeout(outKey: OutKey, outSeqNr: OutSeqNr) extends InternalCommand
```

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 110 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ProducerControllerImpl$LoadStateReply
File: scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala:110
Taint Flags:

```
107 case object ResendFirstUnconfirmed extends InternalCommand
108 private case object SendChunk extends InternalCommand
```

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

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 110 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
109
110 private case class LoadStateReply[A](state: DurableProducerQueue.State[A]) extends InternalCommand
111 private case class LoadStateFailed(attempt: Int) extends InternalCommand
112 private case class StoreMessageSentReply(ack: DurableProducerQueue.StoreMessageSentAck)
113 private case class StoreMessageSentFailed[A](messageSent: DurableProducerQueue.MessageSent[A], attempt: Int)
```

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 99 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: WorkPullingProducerControllerImpl$HandOver
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:99
Taint Flags:

```
96
97 private case class PreselectedWorker(outKey: OutKey, confirmationQualifier: ConfirmationQualifier)
98
99 private case class HandOver(oldConfirmationQualifier: ConfirmationQualifier, oldSeqNr: TotalSeqNr)
100
101 // registration of workers via Receptionist
102 private final case class CurrentWorkers[A](workers: Set[ActorRef[ConsumerController.Command[A]]])
```

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 97 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: WorkPullingProducerControllerImpl$PreselectedWorker
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:97
Taint Flags:

```
94 producer: ActorRef[WorkPullingProducerController.RequestNext[A]],
95 requested: Boolean)
96
97 private case class PreselectedWorker(outKey: OutKey, confirmationQualifier: ConfirmationQualifier)
98
99 private case class HandOver(oldConfirmationQualifier: ConfirmationQualifier, oldSeqNr: TotalSeqNr)
100
```

Code Correctness: Non-Static Inner Class Implements Serializable	Low

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 97 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 77 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: WorkPullingProducerControllerImpl$Unconfirmed
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:77
Taint Flags:

```
74 def confirmationQualifier: ConfirmationQualifier = producerController.path.name
75 }
76
77 private final case class Unconfirmed[A](
78 totalSeqNr: TotalSeqNr,
79 outSeqNr: OutSeqNr,
80 msg: A,
```

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 115 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ProducerControllerImpl$StoreMessageSentCompleted
File: scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala:115
Taint Flags:

```
112 private case class StoreMessageSentReply(ack: DurableProducerQueue.StoreMessageSentAck)
113 private case class StoreMessageSentFailed[A](messageSent: DurableProducerQueue.MessageSent[A], attempt: Int)
114 extends InternalCommand
115 private case class StoreMessageSentCompleted[A](messageSent: DurableProducerQueue.MessageSent[A])
116 extends InternalCommand
117 private case object DurableQueueTerminated extends InternalCommand
118
```

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 62 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

Issue Details

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

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 62 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: WorkPullingProducerControllerImpl$StoreMessageSentFailed
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:62
Taint Flags:

```
59 private case class LoadStateReply[A](state: DurableProducerQueue.State[A]) extends InternalCommand
60 private case class LoadStateFailed(attempt: Int) extends InternalCommand
61 private case class StoreMessageSentReply(ack: DurableProducerQueue.StoreMessageSentAck)
62 private case class StoreMessageSentFailed[A](messageSent: DurableProducerQueue.MessageSent[A], attempt: Int)
63 extends InternalCommand
64 private case class StoreMessageSentCompleted[A](messageSent: DurableProducerQueue.MessageSent[A])
65 extends InternalCommand
```

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 60 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: WorkPullingProducerControllerImpl$LoadStateFailed
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:60
Taint Flags:

```
57 private case object RegisterConsumerDone extends InternalCommand
58
59 private case class LoadStateReply[A](state: DurableProducerQueue.State[A]) extends InternalCommand
60 private case class LoadStateFailed(attempt: Int) extends InternalCommand
61 private case class StoreMessageSentReply(ack: DurableProducerQueue.StoreMessageSentAck)
62 private case class StoreMessageSentFailed[A](messageSent: DurableProducerQueue.MessageSent[A], attempt: Int)
63 extends InternalCommand
```

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 68 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

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

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 68 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: WorkPullingProducerControllerImpl$OutState
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:68
Taint Flags:

```
65 extends InternalCommand
66 private case object DurableQueueTerminated extends InternalCommand
67
68 private final case class OutState[A](
69 producerController: ActorRef[ProducerController.Command[A]],
70 consumerController: ActorRef[ConsumerController.Command[A]],
71 seqNr: OutSeqNr,
```

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 112 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ProducerControllerImpl$StoreMessageSentReply
File: scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala:112
Taint Flags:

```
109
110 private case class LoadStateReply[A](state: DurableProducerQueue.State[A]) extends InternalCommand
111 private case class LoadStateFailed(attempt: Int) extends InternalCommand
112 private case class StoreMessageSentReply(ack: DurableProducerQueue.StoreMessageSentAck)
113 private case class StoreMessageSentFailed[A](messageSent: DurableProducerQueue.MessageSent[A], attempt: Int)
114 extends InternalCommand
115 private case class StoreMessageSentCompleted[A](messageSent: DurableProducerQueue.MessageSent[A])
```

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 100 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ProducerControllerImpl$Ack
File: scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala:100
Taint Flags:

```
97 s"confirmedSeqNr [$confirmedSeqNr] should be <= requestUpToSeqNr [$requestUpToSeqNr]")
98 }
```

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

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 100 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
99  final case class Resend(fromSeqNr: SeqNr) extends InternalCommand with DeliverySerializable with DeadLetterSuppression
100 final case class Ack(confirmedSeqNr: SeqNr)
101 extends InternalCommand
102 with DeliverySerializable
103 with DeadLetterSuppression
```

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 59 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: WorkPullingProducerControllerImpl$LoadStateReply
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:59
Taint Flags:

```
56
57 private case object RegisterConsumerDone extends InternalCommand
58
59 private case class LoadStateReply[A](state: DurableProducerQueue.State[A]) extends InternalCommand
60 private case class LoadStateFailed(attempt: Int) extends InternalCommand
61 private case class StoreMessageSentReply(ack: DurableProducerQueue.StoreMessageSentAck)
62 private case class StoreMessageSentFailed[A](messageSent: DurableProducerQueue.MessageSent[A], attempt: Int)
```

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 113 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ProducerControllerImpl$StoreMessageSentFailed
File: scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala:113
Taint Flags:

```
110 private case class LoadStateReply[A](state: DurableProducerQueue.State[A]) extends InternalCommand
111 private case class LoadStateFailed(attempt: Int) extends InternalCommand
112 private case class StoreMessageSentReply(ack: DurableProducerQueue.StoreMessageSentAck)
113 private case class StoreMessageSentFailed[A](messageSent: DurableProducerQueue.MessageSent[A], attempt: Int)
114 extends InternalCommand
115 private case class StoreMessageSentCompleted[A](messageSent: DurableProducerQueue.MessageSent[A])
116 extends InternalCommand
```

Code Correctness: Non-Static Inner Class Implements Serializable	Low

Package: akka.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/ProducerControllerImpl.scala, line 113 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

Package: akka.actor.typed.internal

scala/akka/actor/typed/internal/TimerSchedulerImpl.scala, line 23 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TimerSchedulerImpl$Timer
File: scala/akka/actor/typed/internal/TimerSchedulerImpl.scala:23
Taint Flags:

```
20 * INTERNAL API
21 */
22 @InternalApi private[akka] object TimerSchedulerImpl {
23 final case class Timer[T](key: Any, msg: T, repeat: Boolean, generation: Int, task: Cancellable)
24 sealed class TimerMsg(val key: Any, val generation: Int, val owner: AnyRef) {
25 override def toString = s"TimerMsg(key=$key, generation=$generation, owner=$owner)"
26 }
```

scala/akka/actor/typed/internal/ActorContextImpl.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: ActorContextImpl$LoggingContext
File: scala/akka/actor/typed/internal/ActorContextImpl.scala:63
Taint Flags:

```
60 }
61 }
62
63 final case class LoggingContext(
64 logger: Logger,
65 tagsString: String,
66 akkaSource: String,
```

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

Package: akka.actor.typed.internal

scala/akka/actor/typed/internal/Supervision.scala, line 188 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: RestartSupervisor$ScheduledRestart
File: scala/akka/actor/typed/internal/Supervision.scala:188
Taint Flags:

```
185 }
186 }
187
188 final case class ScheduledRestart(owner: RestartSupervisor[_, _ <: Throwable]) extends DeadLetterSuppression
189 final case class ResetRestartCount(current: Int, owner: RestartSupervisor[_, _ <: Throwable])
190 extends DeadLetterSuppression
191 }
```

scala/akka/actor/typed/internal/Supervision.scala, line 189 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: RestartSupervisor$ResetRestartCount
File: scala/akka/actor/typed/internal/Supervision.scala:189
Taint Flags:

```
186 }
187
188 final case class ScheduledRestart(owner: RestartSupervisor[_, _ <: Throwable]) extends DeadLetterSuppression
189 final case class ResetRestartCount(current: Int, owner: RestartSupervisor[_, _ <: Throwable])
190 extends DeadLetterSuppression
191 }
192
```

scala/akka/actor/typed/internal/TimerSchedulerImpl.scala, line 46 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

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

Package: akka.actor.typed.internal

scala/akka/actor/typed/internal/TimerSchedulerImpl.scala, line 46 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: TimerSchedulerImpl$FixedDelayMode
File: scala/akka/actor/typed/internal/TimerSchedulerImpl.scala:46
Taint Flags:

```
43 private case class FixedRateMode(initialDelay: FiniteDuration) extends TimerMode {
44 override def repeat: Boolean = true
45 }
46 private case class FixedDelayMode(initialDelay: FiniteDuration) extends TimerMode {
47 override def repeat: Boolean = true
48 }
49 private case object SingleMode extends TimerMode {
```

scala/akka/actor/typed/internal/TimerSchedulerImpl.scala, line 43 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TimerSchedulerImpl$FixedRateMode
File: scala/akka/actor/typed/internal/TimerSchedulerImpl.scala:43
Taint Flags:

```
40 private sealed trait TimerMode {
41 def repeat: Boolean
42 }
43 private case class FixedRateMode(initialDelay: FiniteDuration) extends TimerMode {
44 override def repeat: Boolean = true
45 }
46 private case class FixedDelayMode(initialDelay: FiniteDuration) extends TimerMode {
```

Package: akka.actor.typed.internal.adapter

scala/akka/actor/typed/internal/adapter/ActorAdapter.scala, line 39 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ActorAdapter$TypedActorFailedException
File: scala/akka/actor/typed/internal/adapter/ActorAdapter.scala:39
Taint Flags:

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

Package: akka.actor.typed.internal.adapter

scala/akka/actor/typed/internal/adapter/ActorAdapter.scala, line 39 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
36 *
37 * Should only be thrown if the parent is known to be an `ActorAdapter`.
38 */
39 final case class TypedActorFailedException(cause: Throwable) extends RuntimeException
40
41 private val DummyReceive: classic.Actor.Receive = {
42 case _ => throw new RuntimeException("receive should never be called on the typed ActorAdapter")
```

Package: akka.actor.typed.internal.pubsub

scala/akka/actor/typed/internal/pubsub/TopicImpl.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: TopicImpl$MessagePublished
File: scala/akka/actor/typed/internal/pubsub/TopicImpl.scala:42
Taint Flags:

```
39 final case class GetTopicStats[T](replyTo: ActorRef[TopicStats]) extends Topic.Command[T]
40 final case class TopicStats(localSubscriberCount: Int, topicInstanceCount: Int) extends Topic.TopicStats
41 final case class TopicInstancesUpdated[T](topics: Set[ActorRef[TopicImpl.Command[T]]]) extends Command[T]
42 final case class MessagePublished[T](message: T) extends Command[T]
43 final case class SubscriberTerminated[T](subscriber: ActorRef[T]) extends Command[T]
44 }
45
```

scala/akka/actor/typed/internal/pubsub/TopicImpl.scala, line 40 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TopicImpl$TopicStats
File: scala/akka/actor/typed/internal/pubsub/TopicImpl.scala:40
Taint Flags:

```
37
38 // internal messages, note that the protobuf serializer for those sent remotely is defined in akka-cluster-typed
39 final case class GetTopicStats[T](replyTo: ActorRef[TopicStats]) extends Topic.Command[T]
40 final case class TopicStats(localSubscriberCount: Int, topicInstanceCount: Int) extends Topic.TopicStats
```

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

Package: akka.actor.typed.internal.pubsub

scala/akka/actor/typed/internal/pubsub/TopicImpl.scala, line 40 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
41 final case class TopicInstancesUpdated[T](topics: Set[ActorRef[TopicImpl.Command[T]]]) extends Command[T]
42 final case class MessagePublished[T](message: T) extends Command[T]
43 final case class SubscriberTerminated[T](subscriber: ActorRef[T]) extends Command[T]
```

scala/akka/actor/typed/internal/pubsub/TopicImpl.scala, line 39 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TopicImpl$GetTopicStats
File: scala/akka/actor/typed/internal/pubsub/TopicImpl.scala:39
Taint Flags:

```
36 final case class Unsubscribe[T](subscriber: ActorRef[T]) extends Topic.Command[T]
37
38 // internal messages, note that the protobuf serializer for those sent remotely is defined in akka-cluster-typed
39 final case class GetTopicStats[T](replyTo: ActorRef[TopicStats]) extends Topic.Command[T]
40 final case class TopicStats(localSubscriberCount: Int, topicInstanceCount: Int) extends Topic.TopicStats
41 final case class TopicInstancesUpdated[T](topics: Set[ActorRef[TopicImpl.Command[T]]]) extends Command[T]
42 final case class MessagePublished[T](message: T) extends Command[T]
```

scala/akka/actor/typed/internal/pubsub/TopicImpl.scala, line 35 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TopicImpl$Subscribe
File: scala/akka/actor/typed/internal/pubsub/TopicImpl.scala:35
Taint Flags:

```
32 if (message == null)
33 throw InvalidMessageException("[null] is not an allowed message")
34 }
35 final case class Subscribe[T](subscriber: ActorRef[T]) extends Topic.Command[T]
36 final case class Unsubscribe[T](subscriber: ActorRef[T]) extends Topic.Command[T]
37
38 // internal messages, note that the protobuf serializer for those sent remotely is defined in akka-cluster-typed
```

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

Package: akka.actor.typed.internal.pubsub

scala/akka/actor/typed/internal/pubsub/TopicImpl.scala, line 43 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TopicImpl$SubscriberTerminated
File: scala/akka/actor/typed/internal/pubsub/TopicImpl.scala:43
Taint Flags:

```
40 final case class TopicStats(localSubscriberCount: Int, topicInstanceCount: Int) extends Topic.TopicStats
41 final case class TopicInstancesUpdated[T](topics: Set[ActorRef[TopicImpl.Command[T]]]) extends Command[T]
42 final case class MessagePublished[T](message: T) extends Command[T]
43 final case class SubscriberTerminated[T](subscriber: ActorRef[T]) extends Command[T]
44 }
45
46 /**
```

scala/akka/actor/typed/internal/pubsub/TopicImpl.scala, line 41 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TopicImpl$TopicInstancesUpdated
File: scala/akka/actor/typed/internal/pubsub/TopicImpl.scala:41
Taint Flags:

```
38 // internal messages, note that the protobuf serializer for those sent remotely is defined in akka-cluster-typed
39 final case class GetTopicStats[T](replyTo: ActorRef[TopicStats]) extends Topic.Command[T]
40 final case class TopicStats(localSubscriberCount: Int, topicInstanceCount: Int) extends Topic.TopicStats
41 final case class TopicInstancesUpdated[T](topics: Set[ActorRef[TopicImpl.Command[T]]]) extends Command[T]
42 final case class MessagePublished[T](message: T) extends Command[T]
43 final case class SubscriberTerminated[T](subscriber: ActorRef[T]) extends Command[T]
44 }
```

scala/akka/actor/typed/internal/pubsub/TopicImpl.scala, line 36 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

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

Package: akka.actor.typed.internal.pubsub

scala/akka/actor/typed/internal/pubsub/TopicImpl.scala, line 36 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: TopicImpl$Unsubscribe
File: scala/akka/actor/typed/internal/pubsub/TopicImpl.scala:36
Taint Flags:

```
33 throw InvalidMessageException("[null] is not an allowed message")
34 }
35 final case class Subscribe[T](subscriber: ActorRef[T]) extends Topic.Command[T]
36 final case class Unsubscribe[T](subscriber: ActorRef[T]) extends Topic.Command[T]
37
38 // internal messages, note that the protobuf serializer for those sent remotely is defined in akka-cluster-typed
39 final case class GetTopicStats[T](replyTo: ActorRef[TopicStats]) extends Topic.Command[T]
```

scala/akka/actor/typed/internal/pubsub/TopicImpl.scala, line 31 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: TopicImpl$Publish
File: scala/akka/actor/typed/internal/pubsub/TopicImpl.scala:31
Taint Flags:

```
28 trait Command[T]
29
30 // actual public messages but internal to ease bincomp evolution
31 final case class Publish[T](message: T) extends Topic.Command[T] {
32 if (message == null)
33 throw InvalidMessageException("[null] is not an allowed message")
34 }
```

Package: akka.actor.typed.internal.receptionist

scala/akka/actor/typed/internal/receptionist/LocalReceptionist.scala, line 43 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: LocalReceptionist$RegisteredActorTerminated
File: scala/akka/actor/typed/internal/receptionist/LocalReceptionist.scala:43
Taint Flags:

Code Correctness: Non-Static Inner Class Implements Serializable	Low

Package: akka.actor.typed.internal.receptionist

scala/akka/actor/typed/internal/receptionist/LocalReceptionist.scala, line 43 (Code Correctness: Non-Static Inner Class Implements Serializable) **Low**

```
40 private type Subscriber[K <: AbstractServiceKey] = Platform.Subscriber[K]
41
42 private sealed trait InternalCommand
43 private final case class RegisteredActorTerminated[T](ref: ActorRef[T]) extends InternalCommand
44 private final case class SubscriberTerminated[T](ref: ActorRef[ReceptionistMessages.Listing[T]])
45 extends InternalCommand
46
```

scala/akka/actor/typed/internal/receptionist/LocalReceptionist.scala, line 44 (Code Correctness: Non-Static Inner Class Implements Serializable) **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: LocalReceptionist$SubscriberTerminated
File: scala/akka/actor/typed/internal/receptionist/LocalReceptionist.scala:44
Taint Flags:

```
41
42 private sealed trait InternalCommand
43 private final case class RegisteredActorTerminated[T](ref: ActorRef[T]) extends InternalCommand
44 private final case class SubscriberTerminated[T](ref: ActorRef[ReceptionistMessages.Listing[T]])
45 extends InternalCommand
46
47 private object State {
```

scala/akka/actor/typed/internal/receptionist/LocalReceptionist.scala, line 62 (Code Correctness: Non-Static Inner Class Implements Serializable) **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: LocalReceptionist$State
File: scala/akka/actor/typed/internal/receptionist/LocalReceptionist.scala:62
Taint Flags:

```
59 * @param subscriptions current subscriptions per service key
60 * @param subscriptionsPerActor current subscriptions per subscriber (needed since a subscriber can subscribe to several keys) FIXME is
it really needed?
61 */
62 private final case class State(
```

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

Package: akka.actor.typed.internal.receptionist

scala/akka/actor/typed/internal/receptionist/LocalReceptionist.scala, line 62 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
63 services: TypedMultiMap[AbstractServiceKey, Service],
64 servicesPerActor: Map[ActorRef[_], Set[AbstractServiceKey]],
65 subscriptions: TypedMultiMap[AbstractServiceKey, Subscriber],
```

Package: akka.actor.typed.javadsl

scala/akka/actor/typed/javadsl/ReceiveBuilder.scala, line 161 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ReceiveBuilder$Case
File: scala/akka/actor/typed/javadsl/ReceiveBuilder.scala:161
Taint Flags:

```
158
159 /** INTERNAL API */
160 @InternalApi
161 private[javadsl] final case class Case[BT, MT](
162 `type`: OptionVal[Class[_ <: MT]],
163 test: OptionVal[JPredicate[MT]],
164 handler: JFunction[MT, Behavior[BT]])
```

scala/akka/actor/typed/javadsl/BehaviorBuilder.scala, line 165 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: BehaviorBuilder$Case
File: scala/akka/actor/typed/javadsl/BehaviorBuilder.scala:165
Taint Flags:

```
162
163 /** INTERNAL API */
164 @InternalApi
165 private[javadsl] final case class Case[BT, MT](
166 `type`: OptionVal[Class[_ <: MT]],
167 test: OptionVal[MT => Boolean],
168 handler: JFunction[MT, Behavior[BT]])
```

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`.

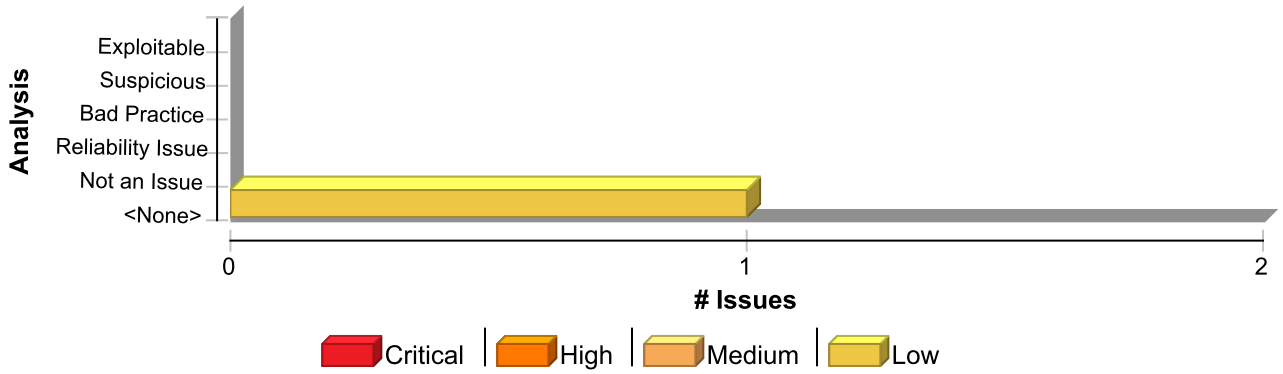
```
public void setUpCalls() {
  boolean firstCall = false;
  boolean secondCall = false;

  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.typed.internal.adapter	
scala/akka/actor/typed/internal/adapter/ActorAdapter.scala, line 201 (Dead Code: Expression is Always true)	**Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: withSafelyAdapted()
File: scala/akka/actor/typed/internal/adapter/ActorAdapter.scala:201
Taint Flags:

```
198 failed = true
199 null.asInstanceOf[U]
200 }
201 if (!failed) {
202 if (adapted != null) body(adapted)
203 else
204 ctx.log.warn(
```

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.

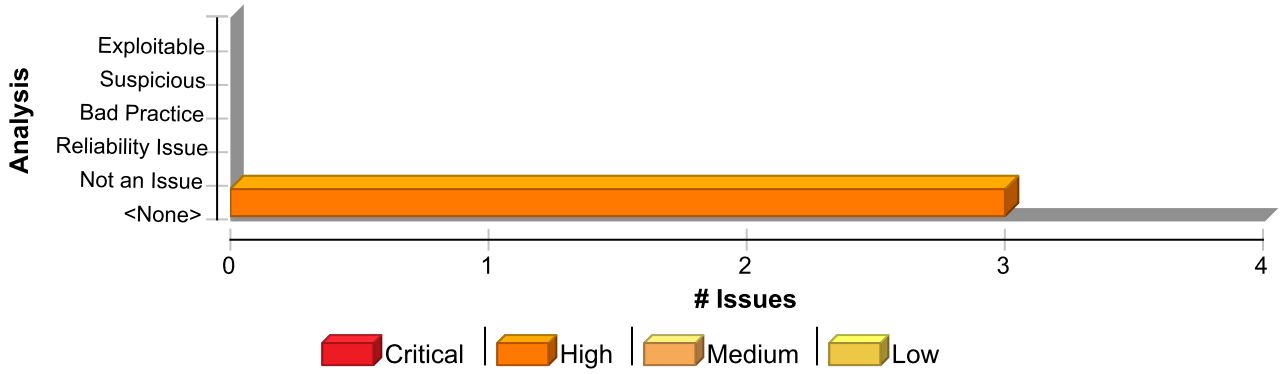
```
String GenerateReceiptURL(String baseUrl) {
    Random ranGen = new Random();
    ranGen.setSeed((new Date()).getTime());
    return (baseUrl + ranGen.nextInt(400000000) + ".html");
}
```

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.actor.typed.delivery.internal

scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala, line 397 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: selectWorker()
File: scala/akka/actor/typed/delivery/internal/WorkPullingProducerControllerImpl.scala:397
Taint Flags:

```
394 if (workers.isEmpty) {
395 None
396 } else {
397 val i = ThreadLocalRandom.current().nextInt(workers.size)
398 Some(workers(i))
399 }
400 }
```

Package: akka.actor.typed.internal

scala/akka/actor/typed/internal/Supervision.scala, line 180 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextDouble()

Insecure Randomness — High

Package: akka.actor.typed.internal

scala/akka/actor/typed/internal/Supervision.scala, line 180 (Insecure Randomness) — High

Enclosing Method: calculateDelay()
File: scala/akka/actor/typed/internal/Supervision.scala:180
Taint Flags:

```
177 minBackoff: FiniteDuration,
178 maxBackoff: FiniteDuration,
179 randomFactor: Double): FiniteDuration = {
180 val rnd = 1.0 + ThreadLocalRandom.current().nextDouble() * randomFactor
181 val calculatedDuration = Try(maxBackoff.min(minBackoff * math.pow(2, restartCount)) * rnd).getOrElse(maxBackoff)
182 calculatedDuration match {
183 case f: FiniteDuration => f
```

Package: akka.actor.typed.internal.routing

scala/akka/actor/typed/internal/routing/RoutingLogic.scala, line 82 (Insecure Randomness) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: nextInt()
Enclosing Method: selectRoutee()
File: scala/akka/actor/typed/internal/routing/RoutingLogic.scala:82
Taint Flags:

```
79 private var currentRoutees: Array[ActorRef[T]] = _
80
81 override def selectRoutee(msg: T): ActorRef[T] = {
82 val selectedIdx = ThreadLocalRandom.current().nextInt(currentRoutees.length)
83 currentRoutees(selectedIdx)
84 }
85
```

Null Dereference (6 issues)

Abstract

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

Explanation

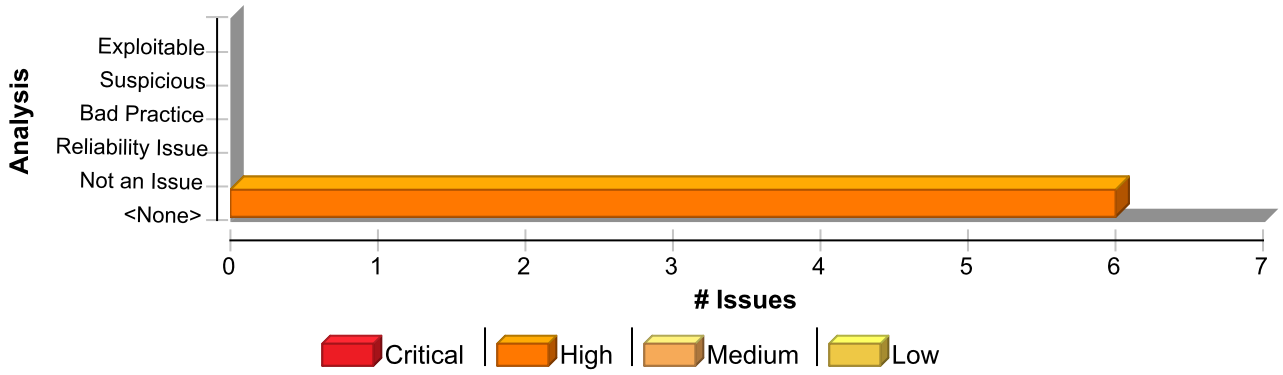
Null-pointer exceptions usually occur when one or more of the programmer's assumptions is violated. A dereference-after-store error occurs when a program explicitly sets an object to `null` and dereferences it later. This error is often the result of a programmer initializing a variable to `null` when it is declared. Most null-pointer issues result in general software reliability problems, but if attackers can intentionally trigger a null-pointer dereference, they can use the resulting exception to bypass security logic or to cause the application to reveal debugging information that will be valuable in planning subsequent attacks. **Example:** In the following code, the programmer explicitly sets the variable `foo` to `null`. Later, the programmer dereferences `foo` before checking the object for a `null` value.

```
Foo foo = null;
...
foo.setBar(val);
...
}
```

Recommendation

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

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Null Dereference	6	0	0	6
Total	**6**	**0**	**0**	**6**

Null Dereference	**High**
Package: akka.actor.typed.internal.adapter	
scala/akka/actor/typed/internal/adapter/PropsAdapter.scala, line 43 (Null Dereference)	**High**
Issue Details	

Null Dereference — High

Package: akka.actor.typed.internal.adapter

scala/akka/actor/typed/internal/adapter/PropsAdapter.scala, line 43 (Null Dereference) — High

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

Sink Details

Sink: Dereferenced : props.firstOrElse(MODULE$.empty(), MODULE$.apply(ActorTags.class))
Enclosing Method: apply()
File: scala/akka/actor/typed/internal/adapter/PropsAdapter.scala:43
Taint Flags:

```
40
41 val localDeploy = mailboxProps.withDeploy(Deploy.local) // disallow remote deployment for typed actors
42
43 val tags = props.firstOrElse[ActorTags](ActorTagsImpl.empty).tags
44 if (tags.isEmpty) localDeploy
45 else localDeploy.withActorTags(tags)
46 }
```

scala/akka/actor/typed/internal/adapter/PropsAdapter.scala, line 26 (Null Dereference) — High

Issue Details

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

Sink Details

Sink: Dereferenced :
Enclosing Method: apply()
File: scala/akka/actor/typed/internal/adapter/PropsAdapter.scala:26
Taint Flags:

```
23
24 val dispatcherProps = (props.firstOrElse[DispatcherSelector](DispatcherDefault.empty) match {
25 case _: DispatcherDefault => classicProps
26 case DispatcherFromConfig(name, _) => classicProps.withDispatcher(name)
27 case _: DispatcherSameAsParent => classicProps.withDispatcher(Deploy.DispatcherSameAsParent)
28 case unknown => throw new RuntimeException(s"Unsupported dispatcher selector: $unknown")
29 }).withDeploy(Deploy.local) // disallow remote deployment for typed actors
```

scala/akka/actor/typed/internal/adapter/PropsAdapter.scala, line 28 (Null Dereference) — High

Issue Details

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

Sink Details

Sink: Dereferenced :
Enclosing Method: apply()

Null Dereference — High

Package: akka.actor.typed.internal.adapter

scala/akka/actor/typed/internal/adapter/PropsAdapter.scala, line 28 (Null Dereference) — High

File: scala/akka/actor/typed/internal/adapter/PropsAdapter.scala:28
Taint Flags:

```
25 case _: DispatcherDefault => classicProps
26 case DispatcherFromConfig(name, _) => classicProps.withDispatcher(name)
27 case _: DispatcherSameAsParent => classicProps.withDispatcher(Deploy.DispatcherSameAsParent)
28 case unknown => throw new RuntimeException(s"Unsupported dispatcher selector: $unknown")
29 }).withDeploy(Deploy.local) // disallow remote deployment for typed actors
30
31 val mailboxProps = props.firstOrElse[MailboxSelector](MailboxSelector.default()) match {
```

scala/akka/actor/typed/internal/adapter/PropsAdapter.scala, line 33 (Null Dereference) — High

Issue Details

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

Sink Details

Sink: Dereferenced : props.firstOrElse(MODULE$.default(), MODULE$.apply(MailboxSelector.class))
Enclosing Method: apply()
File: scala/akka/actor/typed/internal/adapter/PropsAdapter.scala:33
Taint Flags:

```
30
31 val mailboxProps = props.firstOrElse[MailboxSelector](MailboxSelector.default()) match {
32 case _: DefaultMailboxSelector => dispatcherProps
33 case BoundedMailboxSelector(capacity, _) =>
34 // specific support in classic Mailboxes
35 dispatcherProps.withMailbox(s"${Mailboxes.BoundedCapacityPrefix}$capacity")
36 case MailboxFromConfigSelector(path, _) =>
```

scala/akka/actor/typed/internal/adapter/PropsAdapter.scala, line 36 (Null Dereference) — High

Issue Details

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

Sink Details

Sink: Dereferenced : props.firstOrElse(MODULE$.default(), MODULE$.apply(MailboxSelector.class))
Enclosing Method: apply()
File: scala/akka/actor/typed/internal/adapter/PropsAdapter.scala:36
Taint Flags:

```
33 case BoundedMailboxSelector(capacity, _) =>
34 // specific support in classic Mailboxes
35 dispatcherProps.withMailbox(s"${Mailboxes.BoundedCapacityPrefix}$capacity")
36 case MailboxFromConfigSelector(path, _) =>
```

Null Dereference — High

Package: akka.actor.typed.internal.adapter

scala/akka/actor/typed/internal/adapter/PropsAdapter.scala, line 36 (Null Dereference) — High

```
37 dispatcherProps.withMailbox(path)
38 case unknown => throw new RuntimeException(s"Unsupported mailbox selector: $unknown")
39 }
```

scala/akka/actor/typed/internal/adapter/PropsAdapter.scala, line 38 (Null Dereference) — High

Issue Details

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

Sink Details

Sink: Dereferenced : props.firstOrElse(MODULE$.default(), MODULE$.apply(MailboxSelector.class))
Enclosing Method: apply()
File: scala/akka/actor/typed/internal/adapter/PropsAdapter.scala:38
Taint Flags:

```
35 dispatcherProps.withMailbox(s"${Mailboxes.BoundedCapacityPrefix}$capacity")
36 case MailboxFromConfigSelector(path, _) =>
37 dispatcherProps.withMailbox(path)
38 case unknown => throw new RuntimeException(s"Unsupported mailbox selector: $unknown")
39 }
40
41 val localDeploy = mailboxProps.withDeploy(Deploy.local) // disallow remote deployment for typed actors
```

Redundant Null Check (1 issue)

Abstract

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

Explanation

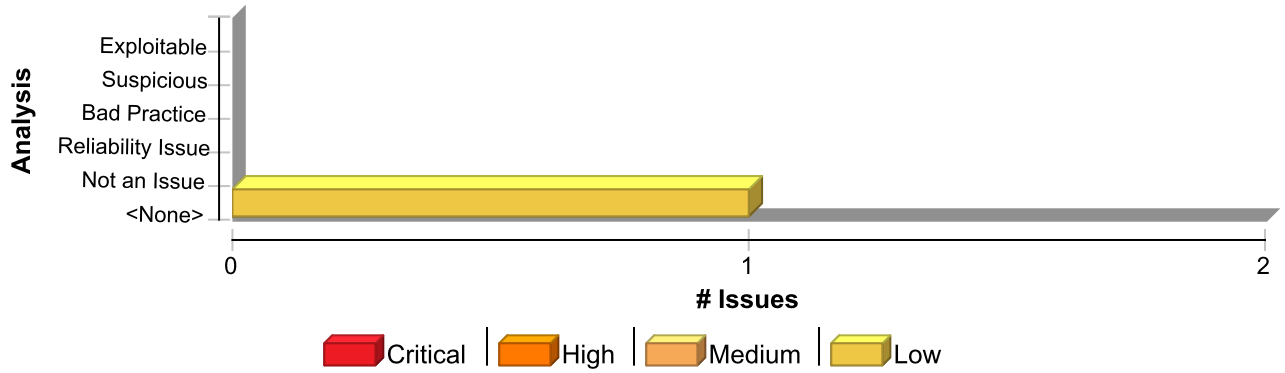
Null-pointer exceptions usually occur when one or more of the programmer's assumptions is violated. Specifically, dereference-after-check errors occur when a program makes an explicit check for `null`, but proceeds to dereference the object when it is known to be `null`. Errors of this type are often 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.typed.internal

scala/akka/actor/typed/internal/StashBufferImpl.scala, line 138 (Redundant Null Check) — Low

Issue Details

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

Sink Details

Sink: Dereferenced : node
Enclosing Method: exists()
File: scala/akka/actor/typed/internal/StashBufferImpl.scala:138
Taint Flags:

```
135 var hasElement = false
136 var node = _first
137 while (node != null && !hasElement) {
138 hasElement = predicate(node.message)
139 node = node.next
140 }
141 hasElement
```

System Information Leak: Internal (8 issues)

Abstract

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

Explanation

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

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

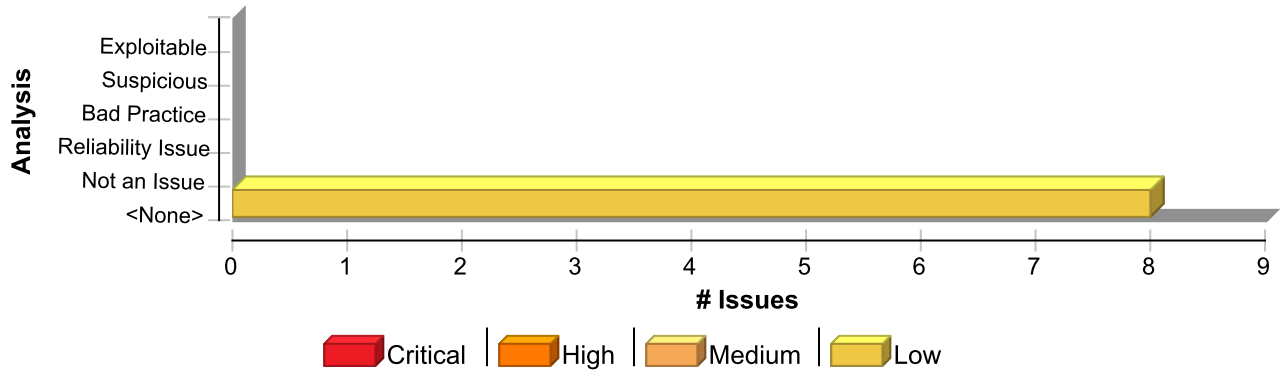
Depending upon the system configuration, this information can be dumped to a console, written to a log file, or exposed to a user. In some cases, the error message provides the attacker with the precise type of attack to which the system is vulnerable. For example, a database error message can reveal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

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

Recommendation

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

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
System Information Leak: Internal	8	0	0	8
Total	**8**	**0**	**0**	**8**

System Information Leak: Internal — Low

Package: akka.actor.typed.internal

scala/akka/actor/typed/internal/Supervision.scala, line 103 (System Information Leak: Internal) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.lang.Throwable.getMessage()
From: akka.actor.typed.internal.AbstractSupervisor.log
File: scala/akka/actor/typed/internal/Supervision.scala:94

```
91  if (strategy.loggingEnabled) {
92  val unwrapped = UnstashException.unwrap(t)
93  val errorCountStr = if (errorCount >= 0) s" [$errorCount]" else ""
94  val logMessage = s"Supervisor $this saw failure$errorCountStr: ${unwrapped.getMessage}"
95  val logger = ctx.asScala.log
96  val logLevel = strategy match {
97  case b: Backoff => if (errorCount > b.criticalLogLevelAfter) b.criticalLogLevel else strategy.logLevel
```

Sink Details

Sink: org.slf4j.Logger.info()
Enclosing Method: log()
File: scala/akka/actor/typed/internal/Supervision.scala:103
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
100  logLevel match {
101  case Level.ERROR => logger.error(logMessage, unwrapped)
102  case Level.WARN => logger.warn(logMessage, unwrapped)
103  case Level.INFO => logger.info(logMessage, unwrapped)
104  case Level.DEBUG => logger.debug(logMessage, unwrapped)
105  case Level.TRACE => logger.trace(logMessage, unwrapped)
106  }
```

scala/akka/actor/typed/internal/Supervision.scala, line 105 (System Information Leak: Internal) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

System Information Leak: Internal	Low

Package: akka.actor.typed.internal

scala/akka/actor/typed/internal/Supervision.scala, line 105 (System Information Leak: Internal) **Low**

Source Details

Source: java.lang.Throwable.getMessage()
From: akka.actor.typed.internal.AbstractSupervisor.log
File: scala/akka/actor/typed/internal/Supervision.scala:94

```
91  if (strategy.loggingEnabled) {
92  val unwrapped = UnstashException.unwrap(t)
93  val errorCountStr = if (errorCount >= 0) s" [$errorCount]" else ""
94  val logMessage = s"Supervisor $this saw failure$errorCountStr: ${unwrapped.getMessage}"
95  val logger = ctx.asScala.log
96  val logLevel = strategy match {
97  case b: Backoff => if (errorCount > b.criticalLogLevelAfter) b.criticalLogLevel else strategy.logLevel
```

Sink Details

Sink: org.slf4j.Logger.trace()
Enclosing Method: log()
File: scala/akka/actor/typed/internal/Supervision.scala:105
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
102  case Level.WARN => logger.warn(logMessage, unwrapped)
103  case Level.INFO => logger.info(logMessage, unwrapped)
104  case Level.DEBUG => logger.debug(logMessage, unwrapped)
105  case Level.TRACE => logger.trace(logMessage, unwrapped)
106  }
107  }
108  }
```

scala/akka/actor/typed/internal/Supervision.scala, line 101 (System Information Leak: Internal) **Low**

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.lang.Throwable.getMessage()
From: akka.actor.typed.internal.AbstractSupervisor.log
File: scala/akka/actor/typed/internal/Supervision.scala:94

```
91  if (strategy.loggingEnabled) {
92  val unwrapped = UnstashException.unwrap(t)
93  val errorCountStr = if (errorCount >= 0) s" [$errorCount]" else ""
94  val logMessage = s"Supervisor $this saw failure$errorCountStr: ${unwrapped.getMessage}"
95  val logger = ctx.asScala.log
```

System Information Leak: Internal — Low

Package: akka.actor.typed.internal

scala/akka/actor/typed/internal/Supervision.scala, line 101 (System Information Leak: Internal) — Low

```
96  val logLevel = strategy match {
97  case b: Backoff => if (errorCount > b.criticalLogLevelAfter) b.criticalLogLevel else strategy.logLevel
```

Sink Details

Sink: org.slf4j.Logger.error()
Enclosing Method: log()
File: scala/akka/actor/typed/internal/Supervision.scala:101
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
98  case _ => strategy.logLevel
99  }
100 logLevel match {
101 case Level.ERROR => logger.error(logMessage, unwrapped)
102 case Level.WARN => logger.warn(logMessage, unwrapped)
103 case Level.INFO => logger.info(logMessage, unwrapped)
104 case Level.DEBUG => logger.debug(logMessage, unwrapped)
```

scala/akka/actor/typed/internal/Supervision.scala, line 104 (System Information Leak: Internal) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.lang.Throwable.getMessage()
From: akka.actor.typed.internal.AbstractSupervisor.log
File: scala/akka/actor/typed/internal/Supervision.scala:94

```
91  if (strategy.loggingEnabled) {
92  val unwrapped = UnstashException.unwrap(t)
93  val errorCountStr = if (errorCount >= 0) s" [$errorCount]" else ""
94  val logMessage = s"Supervisor $this saw failure$errorCountStr: ${unwrapped.getMessage}"
95  val logger = ctx.asScala.log
96  val logLevel = strategy match {
97  case b: Backoff => if (errorCount > b.criticalLogLevelAfter) b.criticalLogLevel else strategy.logLevel
```

Sink Details

Sink: org.slf4j.Logger.debug()
Enclosing Method: log()
File: scala/akka/actor/typed/internal/Supervision.scala:104
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
101 case Level.ERROR => logger.error(logMessage, unwrapped)
```

System Information Leak: Internal	Low

Package: akka.actor.typed.internal

scala/akka/actor/typed/internal/Supervision.scala, line 104 (System Information Leak: Internal) **Low**

```
102  case Level.WARN => logger.warn(logMessage, unwrapped)
103  case Level.INFO => logger.info(logMessage, unwrapped)
104  case Level.DEBUG => logger.debug(logMessage, unwrapped)
105  case Level.TRACE => logger.trace(logMessage, unwrapped)
106  }
107  }
```

scala/akka/actor/typed/internal/Supervision.scala, line 102 (System Information Leak: Internal) **Low**

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.lang.Throwable.getMessage()
From: akka.actor.typed.internal.AbstractSupervisor.log
File: scala/akka/actor/typed/internal/Supervision.scala:94

```
91  if (strategy.loggingEnabled) {
92  val unwrapped = UnstashException.unwrap(t)
93  val errorCountStr = if (errorCount >= 0) s" [$errorCount]" else ""
94  val logMessage = s"Supervisor $this saw failure$errorCountStr: ${unwrapped.getMessage}"
95  val logger = ctx.asScala.log
96  val logLevel = strategy match {
97  case b: Backoff => if (errorCount > b.criticalLogLevelAfter) b.criticalLogLevel else strategy.logLevel
```

Sink Details

Sink: org.slf4j.Logger.warn()
Enclosing Method: log()
File: scala/akka/actor/typed/internal/Supervision.scala:102
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
99  }
100  logLevel match {
101  case Level.ERROR => logger.error(logMessage, unwrapped)
102  case Level.WARN => logger.warn(logMessage, unwrapped)
103  case Level.INFO => logger.info(logMessage, unwrapped)
104  case Level.DEBUG => logger.debug(logMessage, unwrapped)
105  case Level.TRACE => logger.trace(logMessage, unwrapped)
```

System Information Leak: Internal — Low

Package: akka.actor.typed.internal.adapter

scala/akka/actor/typed/internal/adapter/ActorAdapter.scala, line 246 (System Information Leak: Internal) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.lang.Throwable.getMessage()
From: akka.actor.typed.internal.adapter.ActorAdapter$$anonfun$2.applyOrElse
File: scala/akka/actor/typed/internal/adapter/ActorAdapter.scala:243

```
240  case ex: InvocationTargetException if ex.getCause ne null => ex.getCause.getMessage
241  case ex => ex.getMessage
242  }
243  case e => e.getMessage
244  }
245  // log at Error as that is what the supervision strategy would have done.
246  ctx.log.error(logMessage, ex)
```

Sink Details

Sink: org.slf4j.Logger.error()
Enclosing Method: applyOrElse()
File: scala/akka/actor/typed/internal/adapter/ActorAdapter.scala:246
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
243  case e => e.getMessage
244  }
245  // log at Error as that is what the supervision strategy would have done.
246  ctx.log.error(logMessage, ex)
247  if (isTypedActor)
248  classic.SupervisorStrategy.Stop
249  else
```

scala/akka/actor/typed/internal/adapter/ActorAdapter.scala, line 246 (System Information Leak: Internal) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.lang.Throwable.getMessage()
From: akka.actor.typed.internal.adapter.ActorAdapter$$anonfun$2.applyOrElse
File: scala/akka/actor/typed/internal/adapter/ActorAdapter.scala:240

```
237  val logMessage = ex match {
```

System Information Leak: Internal	Low

Package: akka.actor.typed.internal.adapter

scala/akka/actor/typed/internal/adapter/ActorAdapter.scala, line 246 (System Information Leak: Internal) **Low**

```
238  case e: ActorInitializationException if e.getCause ne null =>
239  e.getCause match {
240  case ex: InvocationTargetException if ex.getCause ne null => ex.getCause.getMessage
241  case ex => ex.getMessage
242  }
243  case e => e.getMessage
```

Sink Details

Sink: org.slf4j.Logger.error()
Enclosing Method: applyOrElse()
File: scala/akka/actor/typed/internal/adapter/ActorAdapter.scala:246
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
243  case e => e.getMessage
244  }
245  // log at Error as that is what the supervision strategy would have done.
246  ctx.log.error(logMessage, ex)
247  if (isTypedActor)
248  classic.SupervisorStrategy.Stop
249  else
```

scala/akka/actor/typed/internal/adapter/ActorAdapter.scala, line 246 (System Information Leak: Internal) **Low**

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.lang.Throwable.getMessage()
From: akka.actor.typed.internal.adapter.ActorAdapter$$anonfun$2.applyOrElse
File: scala/akka/actor/typed/internal/adapter/ActorAdapter.scala:241

```
238  case e: ActorInitializationException if e.getCause ne null =>
239  e.getCause match {
240  case ex: InvocationTargetException if ex.getCause ne null => ex.getCause.getMessage
241  case ex => ex.getMessage
242  }
243  case e => e.getMessage
244  }
```

Sink Details

System Information Leak: Internal	Low
Package: akka.actor.typed.internal.adapter	
scala/akka/actor/typed/internal/adapter/ActorAdapter.scala, line 246 (System Information Leak: Internal)	**Low**

Sink: org.slf4j.Logger.error()
Enclosing Method: applyOrElse()
File: scala/akka/actor/typed/internal/adapter/ActorAdapter.scala:246
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
243 case e => e.getMessage
244 }
245 // log at Error as that is what the supervision strategy would have done.
246 ctx.log.error(logMessage, ex)
247 if (isTypedActor)
248 classic.SupervisorStrategy.Stop
249 else
```

Weak SecurityManager Check: Overridable Method (3 issues)

Abstract

Non-final methods that perform security checks may be overridden in ways that bypass security checks.

Explanation

If a method is overridden by a child class, the child class can bypass security checks in the parent class. **Example 1:** In the following code, `doSecurityCheck()` performs a security check and can be overridden by a child class.

```
public class BadSecurityCheck {
    private int id;

    public BadSecurityCheck() {
        doSecurityCheck();
        id = 1;
    }
    protected void doSecurityCheck() {
        SecurityManager sm = System.getSecurityManager();
        if (sm != null) {
            sm.checkPermission(new SomePermission("SomeAction"));
        }
    }
}
```

In this example, if the `SecurityManager` permission is not allowed, a `SecurityException` exception will be thrown, which is a runtime exception and will stop the program from executing any further. Since `BadSecurityCheck` is not `final`, and the method `doSecurityCheck()` is `protected` and not `final`, it means that this class can be subclassed to override this function. **Example 2:** In the following code, `doSecurityCheck()` is overridden by a subclass:

```
public class EvilSubclass extends BadSecurityCheck {
    private int id;

    public EvilSubclass() {
        super();
    }
    protected void doSecurityCheck() {
        //do nothing
    }
}
```

When `EvilSubclass` is instantiated, the constructor first calls `super()`, to invoke the constructor of the superclass. This in turn calls the function `doSecurityCheck()`, but Java will first look for the function within the subclass prior to looking in the superclass, thus invoking the attacker controlled method that bypasses the security check, so `id` will still be set to `1`. This category was derived from the Cigital Java Rulepack.

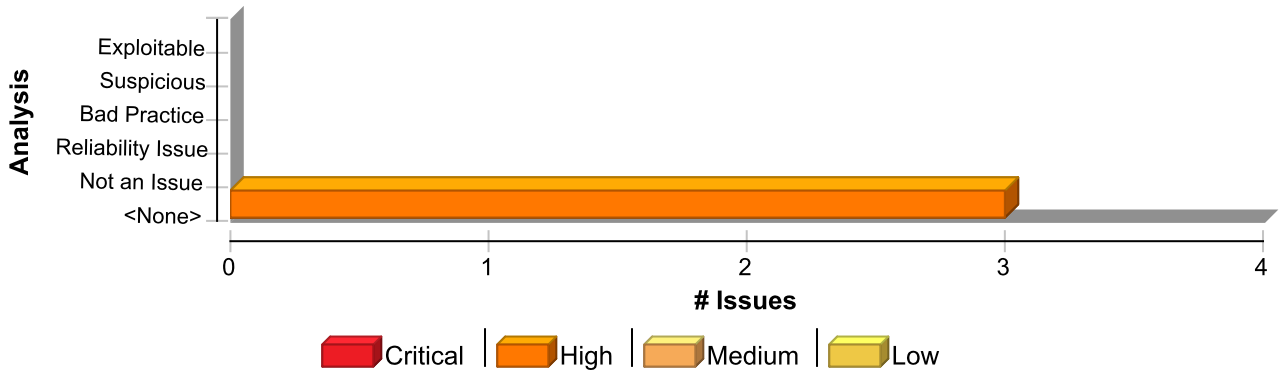
Recommendation

Make sure any methods that perform security operations (e.g. methods from SecurityManager or AccessController) are declared in `final` classes or the methods themselves are declared final. **Example 2:** The following code declared the class `GoodSecurityCheck` as `final` so none of its methods can be overridden.

```
public final class GoodSecurityCheck {
    private int id;

    public GoodSecurityCheck() {
        doSecurityCheck();
        id = 1;
    }
```

```
    void doSecurityCheck() {
        SecurityManager sm = System.getSecurityManager();
        if (sm != null) {
            sm.checkPermission(new SomePermission("SomeAction"));
        }
    }
}
```

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Weak SecurityManager Check: Overridable Method	3	0	0	3
Total	**3**	**0**	**0**	**3**

Weak SecurityManager Check: Overridable Method — High

Package: akka.actor.typed.internal

scala/akka/actor/typed/internal/LoggerClass.scala, line 19 (Weak SecurityManager Check: Overridable Method) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Function: LoggerClass$TrickySecurityManager
Enclosing Method: LoggerClass$TrickySecurityManager()
File: scala/akka/actor/typed/internal/LoggerClass.scala:19
Taint Flags:

```
16 private[akka] object LoggerClass {
17
18  // just to get access to the class context
19  private final class TrickySecurityManager extends SecurityManager {
20  def getClassStack: Array[Class[_]] = getClassContext
21  }
22
```

Weak SecurityManager Check: Overridable Method	High

Package: akka.actor.typed.internal

scala/akka/actor/typed/internal/LoggerClass.scala, line 19 (Weak SecurityManager Check: Overridable Method)	High

scala/akka/actor/typed/internal/LoggerClass.scala, line 20 (Weak SecurityManager Check: Overridable Method)	High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Function: getClassStack
Enclosing Method: getClassStack()
File: scala/akka/actor/typed/internal/LoggerClass.scala:20
Taint Flags:

```
17
18 // just to get access to the class context
19 private final class TrickySecurityManager extends SecurityManager {
20 def getClassStack: Array[Class[_]] = getClassContext
21 }
22
23 private val defaultPrefixesToSkip = List("scala.runtime", "akka.actor.typed.internal")
```

scala/akka/actor/typed/internal/LoggerClass.scala, line 28 (Weak SecurityManager Check: Overridable Method)	High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Function: detectLoggerClassFromStack
Enclosing Method: detectLoggerClassFromStack()
File: scala/akka/actor/typed/internal/LoggerClass.scala:28
Taint Flags:

```
25 /**
26 * Try to extract a logger class from the call stack, if not possible the provided default is used
27 */
28 def detectLoggerClassFromStack(default: Class[_], additionalPrefixesToSkip: List[String] = Nil): Class[_] = {
29 // TODO use stack walker API when we no longer need to support Java 8
30 try {
31 def skip(name: String): Boolean = {
```