Fortify for Scala: A Commercial Back-end Compiler Plugin for Static Security Analysis

Seth Tisue
Lightbend, Inc.

Example vulnerability

```
def endpoint: Action[AnyContent] =
  Action { implicit request =>
    val address = request.getQueryString("address")
    val html = Html(s"Host ${address} pinged")
    Ok(html) as HTML
  }
```

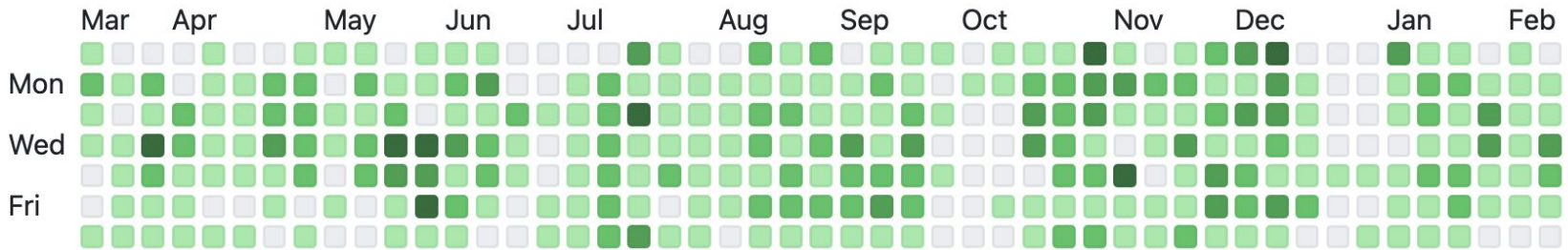
from https://github.com/playframework/play-webgoat

Example vulnerability report

```
[critical : Cross-Site Scripting]
Controller.scala(53) :  ->Result.as(this)
    Controller.scala(53) : <->Results$Status.apply(0->return)
    Controller.scala(50) : <=> (html)
    Controller.scala(50) : <->Html.apply(0->return)
    Controller.scala(50) : <->Object.toString(this->return)
    Controller.scala(45) : <=> (address)
    Controller.scala(45) : <- RequestHeader.getQueryString(return)
```

Who am I?

- Not an academic. (This is an **industry** talk.)
- Active in **Scala community since 2008**
- On the Scala compiler team at **Lightbend since 2015**
- Previously:
  NetLogo simulation language (2000–2014),
  targeting JVM, JS



Lightbend, Inc.

Started as **Scala + Akka**

cofounded by Martin Odersky in 2011 as Typesafe

Now mostly **Akka + Kalix**

“Build, Operate, and Secure Low Latency Systems”

Scala team still going strong 💪

We are the stewards of **Scala 2**…

…and also work on **Scala 3**

Introduction

Overview

Translation

Conclusion

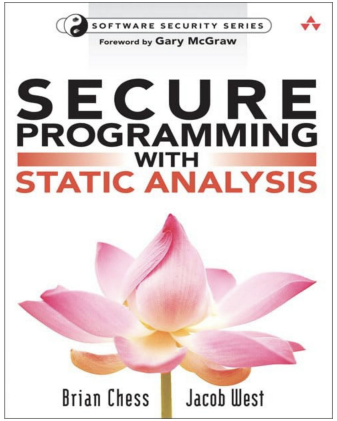
What is Fortify?

Fortify is a suite of security products from **OpenText**

2003	Fortify, Inc.
2010	$40 million/year revenue
2010	Hewlett Packard Enterprise
2017	Micro Focus
2023	OpenText

Fortify remains **industry-standard**

Fortify is **required** at large shops



2007

FORTIFY

FORTIFY
An HP Company

MICRO FOCUS
Fortify

Fortify
by opentext™

Dynamic vs. static

Dynamic: find vuln(erabilitie)s in *running* code

Static: find vulns in *source* code

Fortify SCA = Source Code Analyzer

Let’s just say “Fortify”

Why us?

Fortify helps **drive Scala adoption**

Fortify helps **fund open source**

Our implementation leverages
Scala compiler internals

Vulnerabilities

Fortify published a well-known taxonomy of security vulnerabilities:

https://vulncat.fortify.com/en

For our purposes here,

Linting

Localized pattern matching on ASTs

Dataflow analysis

As seen in example

Example was local;

Fortify can track dataflow through call graph

Fortify architecture

33 languages!

Step 1: **Translate** to “NST” (intermediate language)

Step 2: **Scan** (find vulns in NST code)

Scanner = rule engine

Built in rules (language, standard library)

Vendor rules

Custom rules

The collaboration

OpenText: invented NST, built scanner

OpenText: provided existing Java rules

Lightbend: built Scala ➡ NST translator

my expertise

Lightbend: wrote sample code with sample vulns

play-webgoat
akka-http-webgoat

OpenText: wrote rules to detect our vulns

Timeline

2016	collaboration begins
2017	version 1.0 for Scala 2
2021	plugin licensed, bundled with Fortify
2022	Scala 3 version planned
2023	Scala 3 version release candidate
2024	Scala 3 final release

Introduction

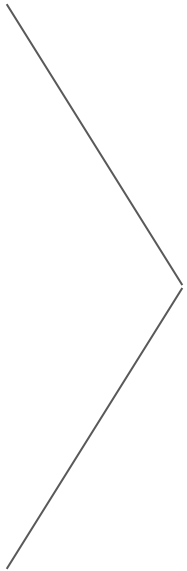
Overview

Translation

Conclusion

Translation: Scala to NST

1. Scala: classes + traits
   NST: **classes + interfaces**
2. Scala: objects only
   NST: **objects, primitive types, arrays**
3. Scala: generic types, higher-kinded types
   NST: **no generics**
4. Scala: arbitrary nesting of constructs
   NST: **top level only**
5. Scala: expression oriented
   NST: **statement oriented**
6. Scala: lambdas
   NST: **no lambdas**



very roughly:

NST ≈ Java 1.0

Scala compiler architecture

```
% ./scalac -Xshow-phases

phase name  description
----------  -----------
    parser  scan and parse sources
     typer  type the trees
   erasure  rewrite types to JVM model
lambdaLift  lift nested functions to class scope
   flatten  lift inner classes to package scope
  genBCode  generate JVM bytecode
```

Solution: Late-phase compiler plugin

```
% ./scalac -Xshow-phases

phase name  description
----------  -----------
    parser  scan and parse sources
     typer  type the trees
   erasure  rewrite types to JVM model
lambdaLift  lift nested functions to class scope
   flatten  lift inner classes to package scope
   fortify  generate NST
  genBCode  generate JVM bytecode
```

Let scalac do the work!

Good news:

- Nested constructs flattened
- Generics erased, casts inserted

Bad news:

- Still expression oriented
- Still entirely object-oriented
- Complex tree shapes:
    - lambdas
    - pattern matching

Challenge: Late-phase ASTs not documented

Solution #1:

Put arbitrary Scala code in and see what tree shapes come out.

Handle those tree shapes.

Repeat.

Solution #2:

Steal!

Solution: Steal! (from Scala.JS)

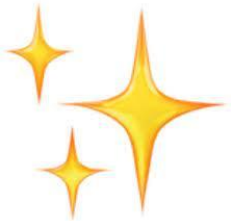
Good news:

- Scala.JS source code is de facto documentation of what to expect

Bad news:

- JavaScript is untyped, NST is typed
- JavaScript has lambdas, NST does not

Thank you,
Sébastien Doeraene!



Challenge: NST not well-documented

Challenge:

- Founders gone
- Experts scarce
- Documentation scanty
- NST code *cannot be executed!*

Solution:

- **Reverse engineer Java translator**
- Vulns driven, customer driven

Anything unusual here?

Nothing truly novel

Endless aligning of weird details of both languages

We must emit the “closure” (declarations for external references)

Compilers 101: Introducing temporaries

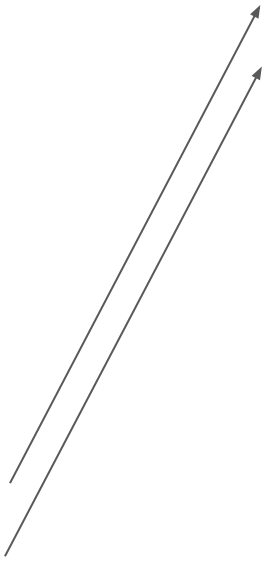
Statements vs. expressions — we must:

- Track if we are in “expression context” or “statement context”
- Translate many constructs differently according to mode
- Introduce temporaries as needed

Compilers 101: Introducing temporaries

```
// Scala
println(
  foo.bar(baz) match
    case _: A => 1
    case _: B => 2
)
```

```
// NST
:int: ~t0;
:int: ~t1;
~t1 :=: Foo~bar(foo, baz);
:if: (__instanceof(~t1, <A>)) {
  ~t0 :=: 1;
} :else: {
  ...
}
scala~Predef@~~println(~t0);
```



~t0 is needed because…?

~t1 is needed because…?

Introduction
Overview
Translation
Conclusion

How to use it: Translation

from https://github.com/playframework/play-webgoat

fortify.sbt:

```
// enable the plugin
addCompilerPlugin(
  "com.lightbend" %% "scala-fortify" % "1.1.0-RC2"
    cross CrossVersion.patch)

// configure the plugin
scalacOptions ++= Seq(
  "-P:fortify:build=play-webgoat")
```

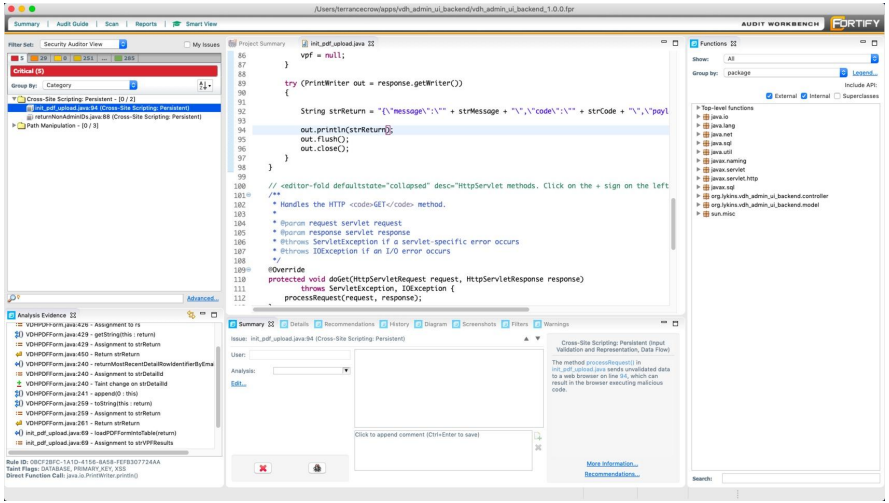
How to use it: Scanning

from https://github.com/playframework/play-webgoat

`.github/workflows/fortify.yml`:

```
sourceanalyzer \
  -b play-webgoat \
  -scan
```

GUI



A lesson about industry

Big shops have:

- Developers (“What’s security?”)
- Build experts (“What’s Fortify?”)
- Security experts (“What’s sbt?”)

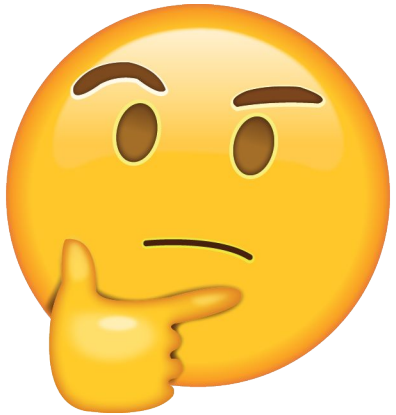
They’re not the same people

Sometimes they don’t even know each other

DOES IT WORK?

Is there real value?

Have companies prevented incidents with this?



What’s next? Nothing!

Maintenance.

Rules. I hope OpenText resumes work on this.

What’s next?

Give back to open source:

- Build a toy version as open source
  - emit Java 1.0 code?
- Document, present!