

DIY: Java Static Analysis

#sonarqube
#sonarlint

Nicolas PERU - @benzonico

Ego boost

- Nicolas PERU - @benzonico
 - Java developer@SonarSource
 - Developer in language team
 - Geneva Jug enthusiast
 - Cycle around the Leman

Sonar Java Plugin

- Back Story



Challenge

Get the language.

```
import java.io.*;
import java.net.*;
import java.security.*;
import protection;

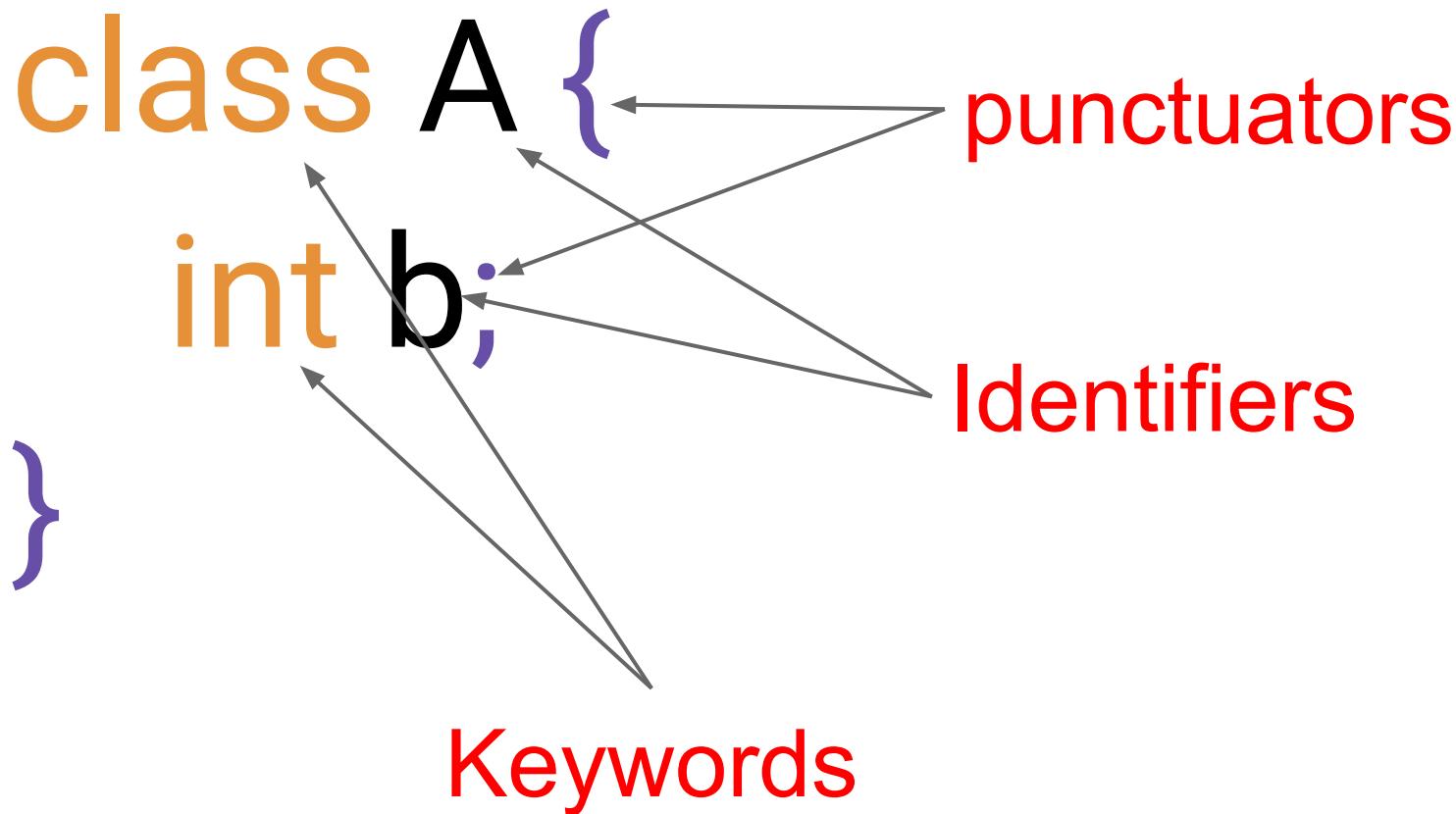
public class Client {
    public void sendAuthentication(String host, int port, String user, String password) throws IOException {
        long t1 = (new Date()).getTime();
        double q1 = Math.random();
        byte[] protected1 = Protection.marshal(q1);
        long t2 = (new Date()).getTime();
        double q2 = Math.random();
        byte[] protected2 = Protection.marshal(q2);
        OutputStream out = new DataOutputStream(new Socket(host, port).getOutputStream());
        out.writeUTF(user);
        out.writeInt(protected1.length);
        out.write(protected2);
        out.flush();
    }

    public static void main(String[] args) {
        String host = args[0];
        int port = 7999;
        String user = "John";
        String password = "shrubbery";
        Socket s = new Socket(host, port);
        Client client = new Client();
        client.sendAuthentication(host, port, user, password);
    }
}
```

Lexical Analysis

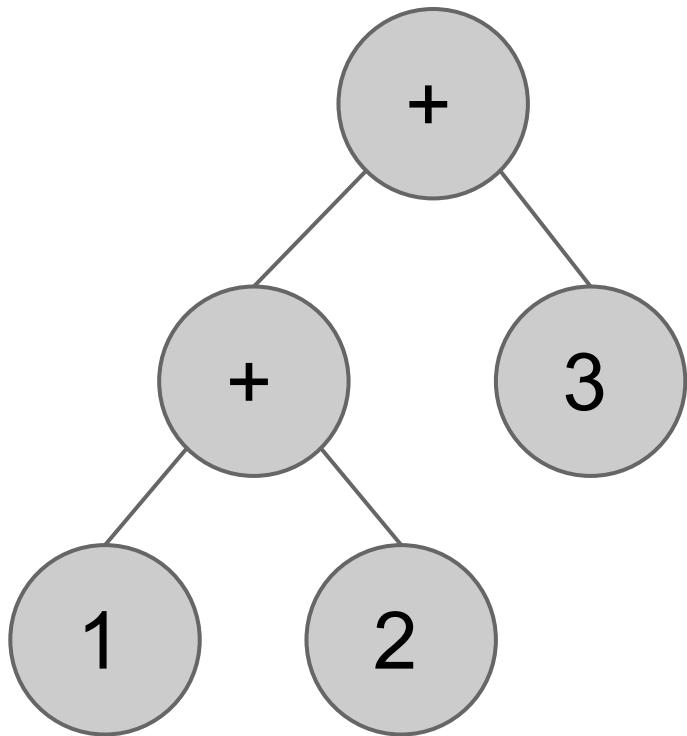
```
class A {  
    int b;  
}
```

Syntactic Analysis



Syntax Tree

1 + 2 + 3



```
interface BinaryExpressionTree {  
    ExpressionTree leftOperand();  
    SyntaxToken operatorToken();  
    ExpressionTree rightOperand();  
}
```

Semantic Analysis

```
class A {  
    int b;  
    A(int b) {  
        this.b = b;  
    }  
}
```

Your turn now : Custom rules !



Beyond Semantic: Symbolic Execution

Beyond Semantic: Symbolic Execution

```
Object myObject = new Object();
// ...
if ( a ) { myObject = null; }
// ...
if ( !a ) { /* ... */ }
else {
    myObject.toString();
}
```

Beyond Semantic: Symbolic Execution

```
Object myObject = new Object();
```

```
// ...
```

```
if ( a ) { myObject = null; }
```

```
// ...
```

```
if ( !a ) { /* ... */ }
```

```
else {
```

```
    myObject.toString();
```

```
}
```

Program State#0
myObject != null

Beyond Semantic: Symbolic Execution

```
Object myObject = new Object();
```

```
// ...
```

```
if ( a ) { myObject = null; }
```

```
// ...
```

```
if ( !a ) { /* ... */ }
```

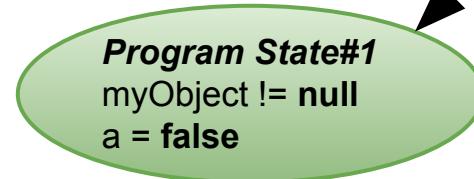
```
else {
```

```
    myObject.toString();
```

```
}
```

Program State#0
myObject != null

Program State#1
myObject != null
a = false



Beyond Semantic: Symbolic Execution

```
Object myObject = new Object();
```

```
// ...
```

```
if ( a ) { myObject = null; }
```

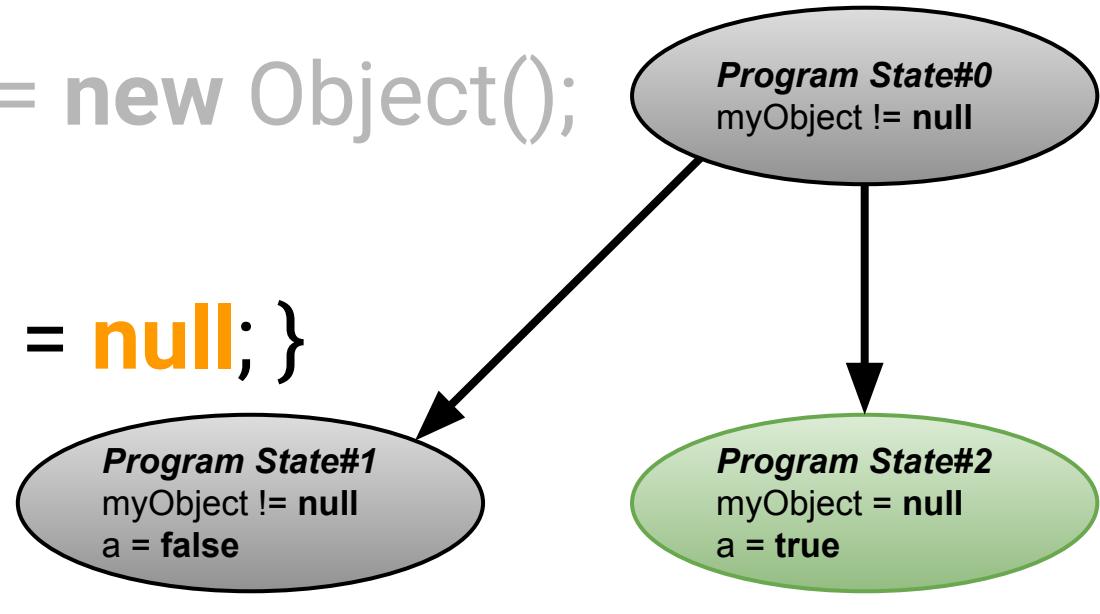
```
// ...
```

```
if ( !a ) { /* ... */ }
```

```
else {
```

```
    myObject.toString();
```

```
}
```



Beyond Semantic: Symbolic Execution

```
Object myObject = new Object();
```

Program State#1
myObject != null
a = false

```
// ...
```

```
if ( a ) { myObject = null; }
```

```
// ...
```

```
if ( !a ) { /* ... */ }
```

```
else {
```

```
    myObject.toString();
```

```
}
```

Program State#2
myObject = null
a = true

Beyond Semantic: Symbolic Execution

```
Object myObject = null; myObject.toString();  
// ...  
if ( a ) { myObject = null; }  
// ...  
if ( !a ) { /* ... */ }  
else {  
    myObject.toString();  
}
```

Program State#1
myObject != **null**
a = **false**

Beyond Semantic: Symbolic Execution

The diagram illustrates the control flow of a Java code snippet. It starts with the declaration of an object:

```
Object myObject =
```

This is followed by two comments:

```
// ...
```

Then, an if statement is reached:

```
if ( a ) { myObject = null; }
```

A grey oval labeled "Program State#1" contains the condition and the assignment:

*myObject != null
a = false*

An arrow points from this state to a green oval labeled "Program State#3".

From "Program State#3", another arrow points to the next part of the code:

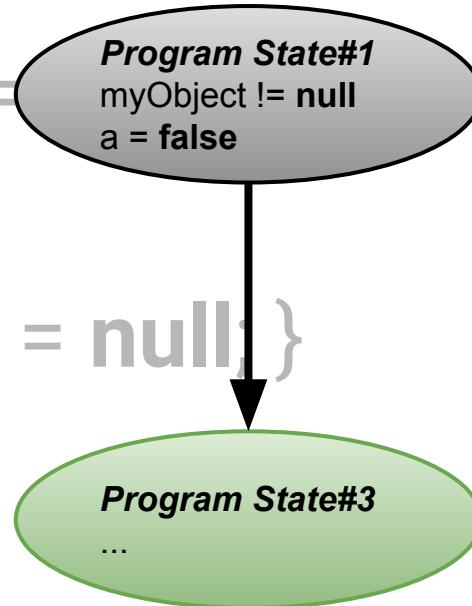
```
// ...
```

Another if statement is encountered:

```
if ( !a ) { /* ... */ }
```

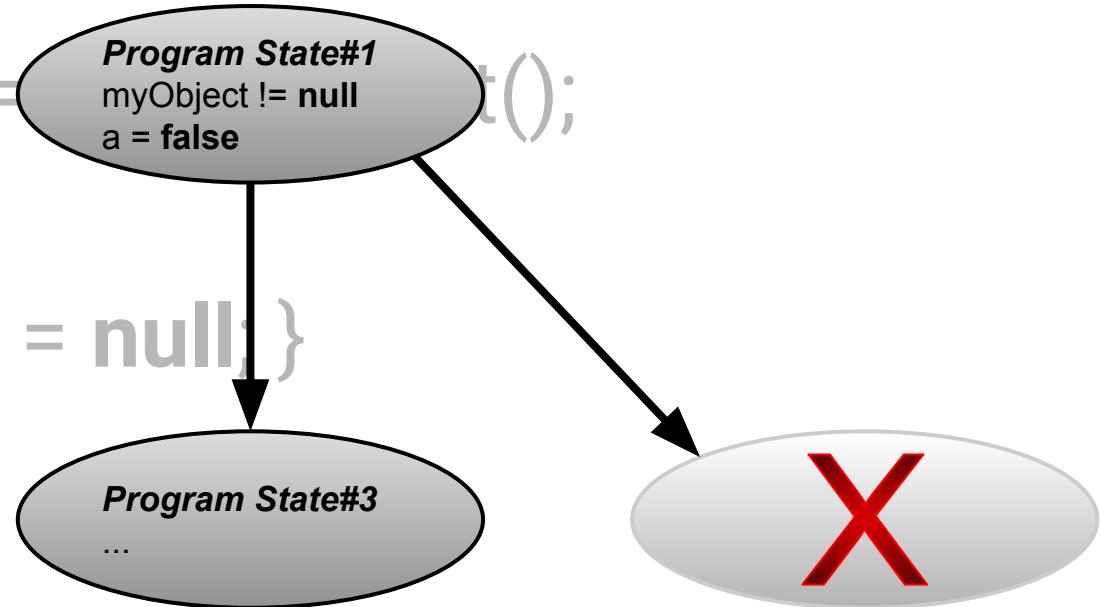
From "Program State#3", an arrow points to the final part of the code:

```
else {  
    myObject.toString();  
}
```



Beyond Semantic: Symbolic Execution

```
Object myObject = ...  
// ...  
if ( a ) { myObject = null; }  
// ...  
if ( !a ) { /* ... */ }  
else {  
    myObject.toString();  
}  
}
```



Beyond Semantic: Symbolic Execution

```
Object myObject = new Object();
```

Program State#1
myObject != null
a = false

```
// ...
```

```
if ( a ) { myObject = null; }
```

```
// ...
```

```
if ( !a ) { /* ... */ }
```

```
else {
```

```
    myObject.toString();
```

```
}
```

Program State#2
myObject = null
a = true

Beyond Semantic: Symbolic Execution

```
Object myObject = new Object();
```

```
// ...
```

```
if ( a ) { myObject = null; }
```

```
// ...
```

```
if ( !a ) { /* ... */ }
```

```
else {
```

```
    myObject.toString();
```

```
}
```

Program State#2
myObject = null
a = true

Beyond Semantic: Symbolic Execution

```
Object myObject = new Object();
```

```
// ...
```

```
if ( a ) { myObject = null; }
```

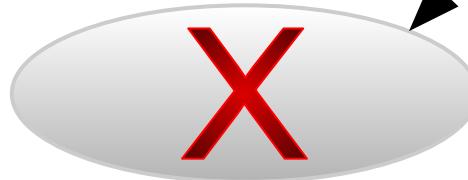
```
// ...
```

```
if ( !a ) { /* ... */ }
```

```
else {
```

```
    myObject.toString();
```

```
}
```



Program State#2
myObject = null
a = true

Beyond Semantic: Symbolic Execution

```
Object myObject = new Object();
```

```
// ...
```

```
if ( a ) { myObject = null; }
```

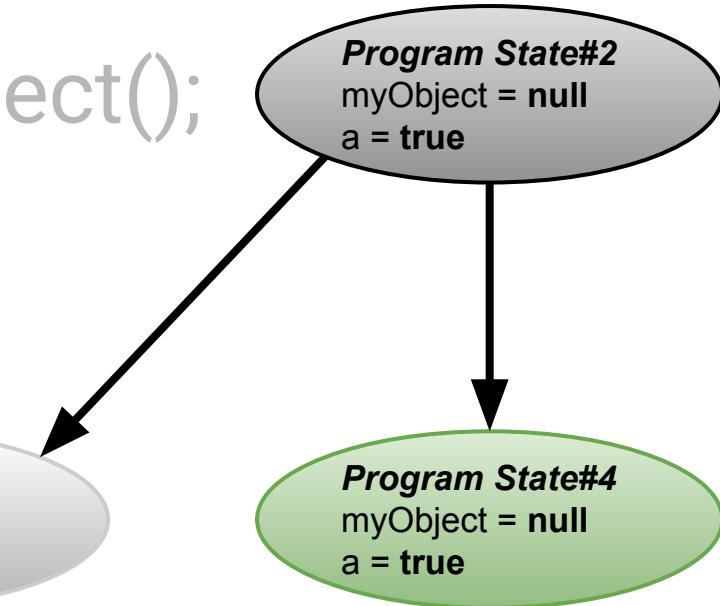
```
// ...
```

```
if ( !a ) { /* ... */ }
```

```
else {
```

```
    myObject.toString();
```

```
}
```



Beyond Semantic: Symbolic Execution

```
Object myObject = new Object();
```

```
// ...
```

```
if ( a ) { myObject = null; }
```

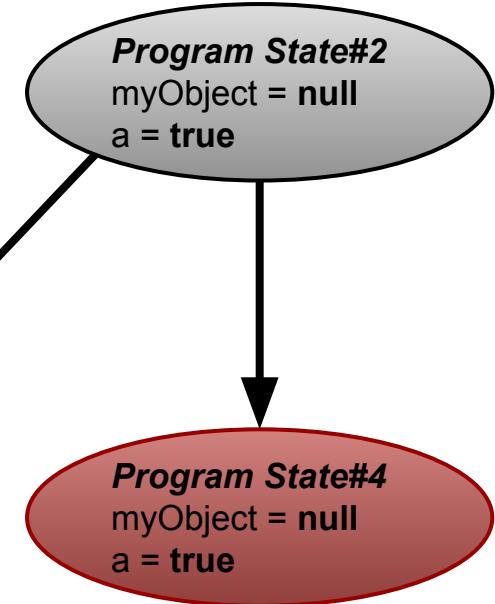
```
// ...
```

```
if ( !a ) { /* ... */ }
```

```
else {
```

```
myObject.toString(); // NPE
```

```
}
```



Symbolic Execution challenges

- Complex conditions

```
if (a + 1 < (b * 10 - 39) ) {  
    if ( b > a / 10 + 4 ) { ... } // Always true  
}
```

Symbolic Execution challenges

- Complex conditions

```
if (a + 1 < (b * 10 - 39) ) {  
    if ( b > a / 10 + 4 ) { ... } // Always true  
}
```

- Explosion of states

Uhoh ?!

```
Entity to = stanza.getTo();
boolean isServerInfoRequest = false;
boolean isComponentInfoRequest = false;
Entity serverEntity = serverRuntimeContext.getServerEntity();
if (to == null || to.equals(serverEntity)) {
    isServerInfoRequest = true; // this can only be meant to query the server
} else if (serverRuntimeContext.getComponentStanzaProcessor(to) != null) {
    isComponentInfoRequest = true; // this is a query to a component
} else if (!to.isNodeSet()) {
    isServerInfoRequest = serverEntity.equals(to);
    if (!isServerInfoRequest) {
```

Change this condition so that it does not always evaluate to "true" ...

2 years ago ▾ L100 Qo

Bug Major Open Not assigned 15min effort

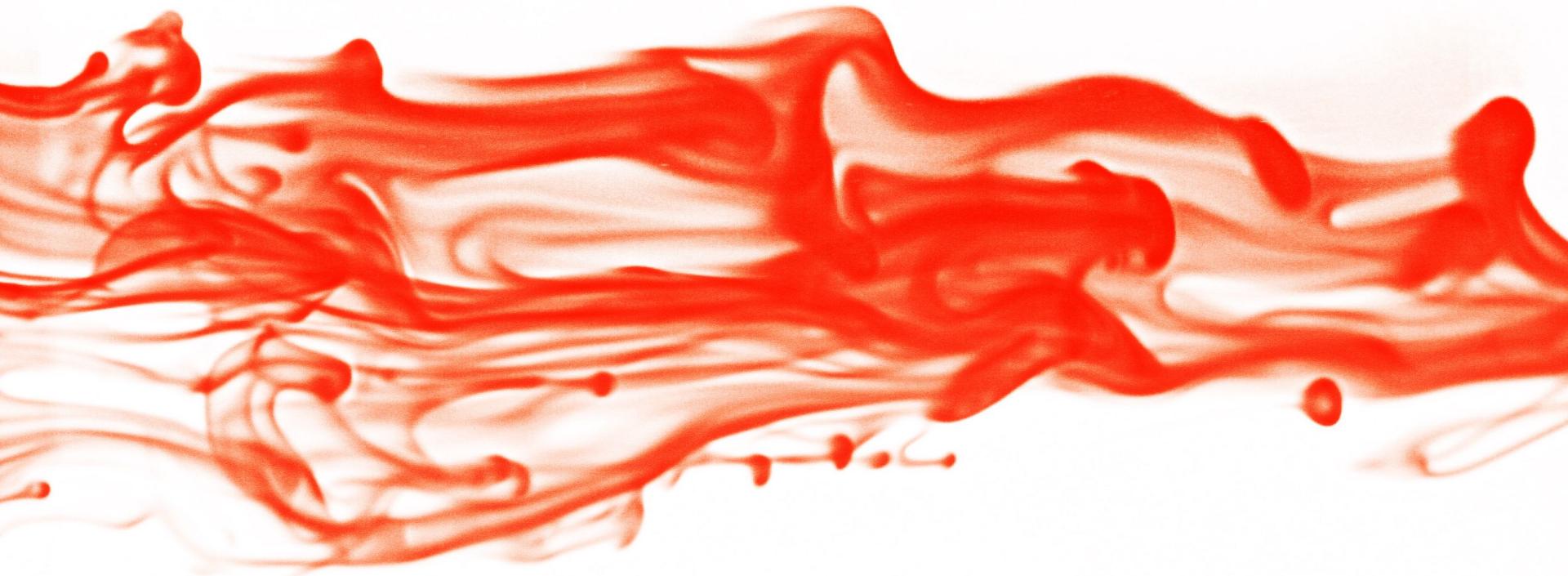
cwe, misra

From [apache vysper](#)

sonarsource

What's next ?

Taint Analysis for vulnerabilities



sonarsource



WE NEED
YOU

jobs@sonarsource.com

sonarsource

