BString: A String-based Framework to Improve Application Security

MSc Thesis (final presentation)

19 January 2022

Christian Zürcher
Two major problems of our time

Data leaks

Remote Code Execution
(RCE)

Important:
These threats usually originate from String values.
Two major problems of our time

https://owasp.org/www-project-top-ten/
Security measures for strings already exist...

- Restrictive data types
- Data encryption
- Taint and data flow analyses
..., but they lack fundamental features

- limited interoperability between different tools
- no shared configuration
- no standardized set up
- limited automation
Our idea: a generalizable security framework for String

Step 1: **Improving** the String class of OpenJDK

Step 2: **Compiling** the modified JDK/JRE

Step 3: **Using** the additional functionality **without** sacrificing any compatibility
Provided API

```java
public interface IStringBehavior {

    public String applyOnCreation(String s);

    public String applyOnRead(String s);

    public boolean attachToChild();

    public boolean recordHistory();

    ...

}
```
Behind the scenes
Challenges

Don’t break the system:
● String is used throughout the Java VM
● Changes must not alter the existing behavior
● Recursions are very easy to introduce

Native code complications:
● The string pooling uses native code
● Translation between different data types
RQ1: What are the restrictions when used with existing Java code?
## Compatibility

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Package</th>
<th>Version</th>
<th>Compatible?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Commons (IO)</td>
<td>Commons-io</td>
<td>2.8.0</td>
<td>✓</td>
</tr>
<tr>
<td>Apache Commons (Logging)</td>
<td>Commons-logging</td>
<td>1.2</td>
<td>✓</td>
</tr>
<tr>
<td>Apache HttpClient</td>
<td>Org.apache.httpcomponents</td>
<td>4.5.13</td>
<td>✓</td>
</tr>
<tr>
<td>Gson</td>
<td>Com.google.code.gson</td>
<td>2.8.5</td>
<td>✗ (reflection)</td>
</tr>
<tr>
<td>JavaMail</td>
<td>Com.sun.mail</td>
<td>1.6.0</td>
<td>✓</td>
</tr>
<tr>
<td>Log4J (core)</td>
<td>Org.apache.logging.log4j</td>
<td>2.14.1</td>
<td>✓</td>
</tr>
<tr>
<td>Logback (classic)</td>
<td>Ch.qos.logback</td>
<td>1.3.0-alpha5</td>
<td>✓</td>
</tr>
<tr>
<td>SLF4J</td>
<td>Slf4j-simple</td>
<td>2.0.0-alpha1</td>
<td>✗ (custom byte buffer)</td>
</tr>
<tr>
<td>SLF4J (API)</td>
<td>Org.slf4j</td>
<td>2.0.0-alpha1</td>
<td>✓</td>
</tr>
<tr>
<td>Spring</td>
<td>Org.springframework</td>
<td>5.3.6</td>
<td>✓</td>
</tr>
<tr>
<td>Square Okhttp</td>
<td>Com.squareup.okhttp3</td>
<td>5.0.0-alpha.2</td>
<td>✓</td>
</tr>
<tr>
<td>Square Okio</td>
<td>Com.squareup.okhokio</td>
<td>2.10.0</td>
<td>✓</td>
</tr>
<tr>
<td>Square Retrofit</td>
<td>Com.squareup.okhretrofit</td>
<td>2.9.0</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 4.1: Evaluation of popular Java libraries
Restrictions

- Native code
- Value conversion
- Reflection
- Concurrency
- Scope
RQ2: What are the security gains and threats?
Security gains

- Date type emulation
- In-memory encryption
- Off-memory encryption
- Taint analysis
- Data flow analysis
Security threats

- String or application hijacking
- Developer confusion
## Performance evaluation

<table>
<thead>
<tr>
<th>[milliseconds]</th>
<th>String initialization</th>
<th>Read value</th>
<th>Attaching behavior to new String</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>4</td>
<td>166</td>
<td>11</td>
</tr>
<tr>
<td>without behavior</td>
<td>13</td>
<td>171</td>
<td>20</td>
</tr>
<tr>
<td>with empty behavior</td>
<td>16</td>
<td>193</td>
<td>22</td>
</tr>
<tr>
<td>with behavior attachment</td>
<td>16</td>
<td>191</td>
<td>32</td>
</tr>
<tr>
<td>with history</td>
<td>59</td>
<td>213</td>
<td>87</td>
</tr>
<tr>
<td>with encryption on read in IO</td>
<td>2,388</td>
<td>9,331</td>
<td>19</td>
</tr>
<tr>
<td>with encryption and decryption</td>
<td>5,112</td>
<td>7,982</td>
<td>timeout</td>
</tr>
</tbody>
</table>
DEMO
Conclusion

- only one framework required
- limited interoperability between different tools
- no shared configuration
- no standardized set up
- limited automation
Summary

Two major problems of our time

Data leaks  Remote Code Execution (RCE)

Important:
These threats usually originate from String values.

Compatibility

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Package</th>
<th>Version</th>
<th>Compatible?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Commons (IO)</td>
<td>Commons-io</td>
<td>2.8.0</td>
<td></td>
</tr>
<tr>
<td>Apache Commons (Logging)</td>
<td>Commons-logging</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>Apache HttpClient</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gson</td>
<td>Com.google.code.gson</td>
<td>2.8.5</td>
<td>(reflection)</td>
</tr>
<tr>
<td>JavaMail</td>
<td>Com.sun.mail</td>
<td>1.6.0</td>
<td></td>
</tr>
<tr>
<td>Log4j (core)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logback (classic)</td>
<td>Ch.qos.logback</td>
<td>1.3.0-alpha1</td>
<td></td>
</tr>
<tr>
<td>SLF4J</td>
<td>SLF4J-simple</td>
<td>2.0.0-alpha1</td>
<td>(custom byte buffer)</td>
</tr>
<tr>
<td>SLF4J (API)</td>
<td>Org.slf4j</td>
<td>2.0.0-alpha1</td>
<td></td>
</tr>
<tr>
<td>Spring</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Square Okhttp</td>
<td>Com.squareup.okhttp</td>
<td>3.0.0-alpha1</td>
<td></td>
</tr>
<tr>
<td>Square Okio</td>
<td>Com.squareup.okio</td>
<td>2.10.0</td>
<td></td>
</tr>
<tr>
<td>Square Retrofit</td>
<td>Com.squareup.okretrift</td>
<td>2.9.0</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: Evaluation of popular Java libraries

Provided API

```java
public interface IStringBehavior {
    public String applyOnCreation(String s);
    public boolean applyOnRead(String s);
    public boolean inheritToChild();
    public boolean recordHistory();
    ...
}
```

Security gains

- Date type emulation
- In-memory encryption
- Off-memory encryption
- Taint analysis
- Data flow analysis