HANDLING OF SECURITY REQUIREMENTS IN SOFTWARE DEVELOPMENT LIFECYCLE

DANIEL KEFER
ISSUES
REPEATING MISTAKES
SECURITY DOCUMENTATION
SECURITY BEHIND DEV PROCESSES AND TOOLING
APPROACH
ALIGN THE PROCESS
SCALE
KISS
1. Parameters
2. Requirements
3. Persist State
4. Create tickets
USE CASES

New assets

Production assets
INTERNALS
Based on JHipster
## Requirement Skeletons

<table>
<thead>
<tr>
<th>Short Name</th>
<th>Description</th>
<th>More Information</th>
<th>Motivation</th>
<th>Strategy</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA-01</td>
<td>3rd party code is identified, checked for security vulnerabilities and its update process is defined.</td>
<td><img src="https://via.placeholder.com/150" alt="Image" /></td>
<td><img src="https://via.placeholder.com/150" alt="Image" /></td>
<td><img src="https://via.placeholder.com/150" alt="Image" /></td>
<td><img src="https://via.placeholder.com/150" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>Implementation of automated tooling can support this task:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- <a href="https://www.owasp.org/index.php/OWASP_Dependency_Check">https://www.owasp.org/index.php/OWASP_Dependency_Check</a> (mapping of dependencies to CVEs)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- <a href="https://nodesecurity.io">https://nodesecurity.io</a> (evaluation of vulnerable packages for npm)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- <a href="http://retire.github.io/retire.js/">http://retire.github.io/retire.js/</a> (JavaScript libraries with known vulnerabilities)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Decrease the security risk being introduced by using vulnerable libraries. Be able to find out quickly if we're affected when new vulnerabilities are published.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA-02</td>
<td>No fundamentally different roles are present in the same application.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- internal employees and external customers should work on completely separated systems so that the privilege escalation probability and impact in case</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Short Name</td>
<td>Description</td>
<td>More Information</td>
<td>Motivation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>-----------------</td>
<td>------------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| SA-01      | 3rd party code is identified, checked for security vulnerabilities and its update process is defined. Implementation of automated tooling can support this task:  
- https://www.owasp.org/index.php/OWASP_Dependency_Check (mapping of dependencies to CVEs)  
- https://nodesecurity.github.io (evaluation of vulnerable packages for npm)  
- http://retirejs.github.io/retire.js/ (JavaScript libraries with known vulnerabilities)  
Decrease the security risk being introduced by using vulnerable libraries. Be able to find out quickly if we're affected when new vulnerabilities are published. | Task | |
| SA-02      | No fundamentally different roles are present in the same application. Example:  
- internal employees and external customers should work on completely separated systems so that the privilege escalation probability and impact in case | Task | |
Alternatives to Option Columns

<table>
<thead>
<tr>
<th>Short Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OE-01</td>
<td>All untrusted data output to any interface are properly escaped for the particular context using a common and standardized approach.</td>
</tr>
</tbody>
</table>

These interfaces can include (but are not limited to):
- SQL
- NoSQL
- Web Services
- LDAP
- ...

Parametrized queries should be used in all cases.

JAVA Application

Example of a prepared statement for SQL queries:

```java
String selectSQL = "SELECT USER_ID, USERNAME FROM DBUSER WHERE USER_ID = ?";
PreparedStatement preparedStatement = dbConnection.prepareStatement(selectSQL);
preparedStatement.setInt(1, 1001);
ResultSet rs = preparedStatement.executeQuery(selectSQL);
while (rs.next()) {
    String userid = rs.getString("USER_ID");
}
```
### Status Columns

<table>
<thead>
<tr>
<th>Short Name</th>
<th>Description</th>
<th>More Information</th>
<th>Motivation</th>
<th>Strategy</th>
<th>Comment</th>
<th>Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA-01</td>
<td>3rd party code is identified, checked for security vulnerabilities and its update process is defined.</td>
<td>Implemented automated tooling can support this task:</td>
<td>Decrease the security risk being introduced by using vulnerable libraries. Be able to find out quickly if we're affected when new vulnerabilities are published.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA-02</td>
<td>No fundamentally different roles are present in the same application. Example:</td>
<td></td>
<td>Task</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementation Type
AUTHENTICATION

Own authentication scheme

CAS (Central Authentication Service)
ROLES

Frontend User
User
Admin
JIRA INTEGRATION

Cross Origin Request Sharing

SecurityRAT inherits user‘s rights in JIRA
FUTURE PLANS
REQUIREMENTS

Continuous development (quality vs quantity)

Language-specific information
INTEGRATION

Issue trackers

Other tooling in use by developers
COMMUNITY

- Pull requests
- Derived projects
- Testing
- Issues
THANK YOU FOR YOUR ATTENTION!

https://securityrat.github.io