Secrets
ABAP
Apex
C
C++
CloudFormation
COBOL
C#
CSS
Flex
Go
HTML
Java
JavaScript
Kotlin
Objective C
PHP
PL/I
PL/SQL
Python
RPG
Ruby
Scala
Swift
Terraform
Text
TypeScript
T-SQL
VB.NET
VB6
XML

Java static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVA code

All rules 632 | Vulnerability 53 | Bug 154 | Security Hotspot 36 | Code Smell 389 | Quick Fix 42

Tags | Search by name...

Assertions should not be used in production code
Bug

DateTimeFormatters should not use mismatched year and week numbers
Bug

Unicode Grapheme Clusters should be avoided inside regex character classes
Bug

Case insensitive Unicode regular expressions should enable the “UNICODE_CASE” flag
Bug

Assertions should not compare an object to itself
Bug

Regex alternatives should not be redundant
Bug

Alternatives in regular expressions should be grouped when used with anchors
Bug

AssertJ methods setting the assertion context should come before an assertion
Bug

AssertJ configuration should be applied
Bug

JUnit5 test classes and methods should not be silently ignored
Bug

"ThreadLocal" variables should be cleaned up when no longer used

Using clear-text protocols is security-sensitive

Analyze your code

Security Hotspot | Critical | cwe owasp

Clear-text protocols such as `ftp`, `telnet` or non-secure `http` lack encryption of transported data, as well as the capability to build an authenticated connection. It means that an attacker able to sniff traffic from the network can read, modify or corrupt the transported content. These protocols are not secure as they expose applications to an extensive range of risks:

- Sensitive data exposure
- Traffic redirected to a malicious endpoint
- Malware infected software update or installer
- Execution of client side code
- Corruption of critical information

Even in the context of isolated networks like offline environments or segmented cloud environments, the insider threat exists. Thus, attacks involving communications being sniffed or tampered with can still happen.

For example, attackers could successfully compromise prior security layers by:

- Bypassing isolation mechanisms
- Compromising a component of the network
- Getting the credentials of an internal IAM account (either from a service account or an actual person)

In such cases, encrypting communications would decrease the chances of attackers to successfully leak data or steal credentials from other network components. By layering various security practices (segmentation and encryption, for example), the application will follow the *defense-in-depth* principle.

Note that using the `http` protocol is being deprecated by major web browsers.

In the past, it has led to the following vulnerabilities:

- CVE-2019-6169
- CVE-2019-12327
- CVE-2019-11065

Ask Yourself Whether

- Application data needs to be protected against falsifications or leaks when transiting over the network.
- Application data transits over a network that is considered untrusted.
- Compliance rules require the service to encrypt data in transit.
- Your application renders web pages with a relaxed mixed content policy.
- OS level protections against clear-text traffic are deactivated.

There is a risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

- Make application data transit over a secure, authenticated and encrypted protocol like TLS or SSH. Here are a few alternatives to the most common clear-text protocols:
  - Use`ssh` as an alternative to `telnet`
  - Use `sftp`, `scp` or `ftps` instead of `ftp`
  - Use `https` instead of `http`

Bug

Strings and Boxed types should be compared using "equals()"

Bug

InputSteam.read() implementation should not return a signed byte

Bug

"compareTo" should not be overloaded

Bug

"iterator" should not return "this"

Bug

- Use `SMTP` over `SSL/TLS` or `SMTP` with `STARTTLS` instead of clear-text SMTP

- Enable encryption of cloud components communications whenever it's possible.
- Configure your application to block mixed content when rendering web pages.
- If available, enforce OS level deativation of all clear-text traffic

It is recommended to secure all transport channels (even local network) as it can take a single non secure connection to compromise an entire application or system.

Sensitive Code Example

These clients from Apache commons net libraries are based on unencrypted protocols and are not recommended:

```
TelnetClient telnet = new TelnetClient(); // Sensitive

FTPClient ftpClient = new FTPClient(); // Sensitive

SMTPClient smtpClient = new SMTPClient(); // Sensitive
```

Unencrypted HTTP connections, when using okhttp library for instance, should be avoided:

```
ConnectionSpec spec = new ConnectionSpec.Builder(ConnectionS
  .build();
```

Android WebView can be configured to allow a secure origin to load content from any other origin, even if that origin is insecure (mixed content);

```
import android.webkit.WebView

WebView webView = findViewById(R.id.webview)
webView.getSettings().setMixedContentMode(MIXED_CONTENT_ALWA
```

Compliant Solution

Use instead these clients from Apache commons net and JSch/ssh library:

```
JSch jsch = new JSch(); // Compliant

if(implicit) {
  // implicit mode is considered deprecated but offer the sa
  FTPSClient ftpsClient = new FTPSClient(true); // Compliant
}
else {
  FTPSClient ftpsClient = new FTPSClient(); // Compliant
}

if(implicit) {
  // implicit mode is considered deprecated but offer the sa
  SMTPSClient smtpsClient = new SMTPSClient(true); // Compli
}
else {
  SMTPSClient smtpsClient = new SMTPSClient(); // Compliant
  smtpsClient.connect("127.0.0.1", 25);
  if (smtpsClient.execTLS()) {
    // commands
  }
}
```

Perform HTTP encrypted connections, with okhttp library for instance:

```
ConnectionSpec spec = new ConnectionSpec.Builder(ConnectionS
  .build();
```

The most secure mode for Android WebView is `MIXED_CONTENT_NEVER_ALLOW`;

```
import android.webkit.WebView

WebView webView = findViewById(R.id.webview)
webView.getSettings().setMixedContentMode(MIXED_CONTENT_NEVE
```

Exceptions

No issue is reported for the following cases because they are not considered sensitive:

- Insecure protocol scheme followed by loopback addresses like 127.0.0.1 or `localhost`

See

- OWASP Top 10 2021 Category A2 - Cryptographic Failures
- OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
- Mobile AppSec Verification Standard - Network Communication Requirements
- OWASP Mobile Top 10 2016 Category M3 - Insecure Communication
- MITRE, CWE-200 - Exposure of Sensitive Information to an Unauthorized Actor
- MITRE, CWE-319 - Cleartext Transmission of Sensitive Information
- Google, Moving towards more secure web
- Mozilla, Deprecating non secure http

Available In:

sonarcloud | sonarqube