

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















Kotlin static code analysis

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

All rules 98  Vulnerability 10  Bug 17  Security Hotspot 15  Code Smell 56

Tags ▾

Search by name... 

Hard-coded credentials are security-sensitive
 Security Hotspot
Cipher algorithms should be robust
 Vulnerability
Encryption algorithms should be used with secure mode and padding scheme
 Vulnerability
Server hostnames should be verified during SSL/TLS connections
 Vulnerability
Server certificates should be verified during SSL/TLS connections
 Vulnerability
Cryptographic keys should be robust
 Vulnerability
Weak SSL/TLS protocols should not be used
 Vulnerability
"SecureRandom" seeds should not be predictable
 Vulnerability
Cipher Block Chaining IVs should be unpredictable
 Vulnerability
Hashes should include an unpredictable salt
 Vulnerability
Regular expressions should be syntactically valid
 Bug
"runFinalizersOnExit" should not be called
 Bug

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

"ScheduledThreadPoolExecutor" should not have 0 core threads	 Bug
Jump statements should not occur in "finally" blocks	 Bug
Using clear-text protocols is security-sensitive	 Security Hotspot
Accessing Android external storage is security-sensitive	 Security Hotspot
Receiving intents is security-sensitive	 Security Hotspot
Broadcasting intents is security-sensitive	 Security Hotspot
Using weak hashing algorithms is security-sensitive	 Security Hotspot
Using pseudorandom number generators (PRNGs) is security-sensitive	 Security Hotspot
Empty lines should not be tested with regex MULTILINE flag	 Code Smell
Cognitive Complexity of functions should not be too high	 Code Smell

- 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 deactivation 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:

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

val ftpClient = FTPClient(); // Sensitive

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

Unencrypted HTTP connections, when using [okhttp](#) library for instance, should be avoided:

```
val spec: ConnectionSpec = ConnectionSpec.Builder(Connect
    .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

val webView: WebView = findViewById(R.id.webview)
webView.getSettings().setMixedContentMode(MIXED_CONTENT_
```

Compliant Solution

Use instead these clients from [Apache commons net](#) and [JSch/ssh](#) library:

```
JSch jsch = JSch();

if(implicit) {
    // implicit mode is considered deprecated but offer th
    val ftpsClient = FTPSClient(true);
}
else {
    val ftpsClient = FTPSClient();
}

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

Perform HTTP encrypted connections, with [okhttp](#) library for instance:

```
val spec: ConnectionSpec =ConnectionSpec.Builder(Connect
    .build()
```

The most secure mode for Android WebView is MIXED_CONTENT_NEVER_ALLOW;

```
import android.webkit.WebView

val webView: WebView = findViewById(R.id.webview)
webView.getSettings().setMixedContentMode(MIXED_CONTENT_
```

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:

