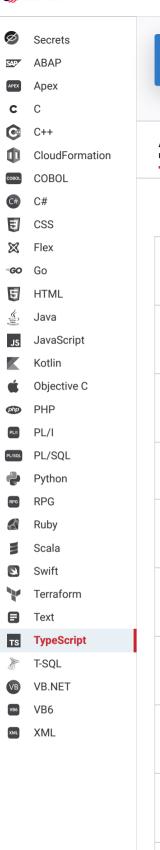
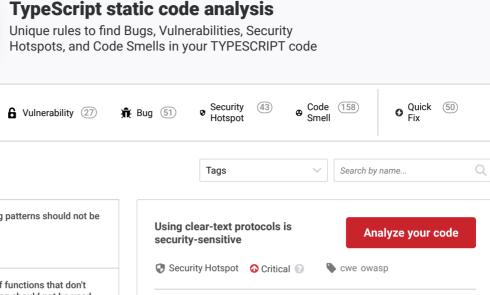


Products ✓







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.

- 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  $\mathtt{http}$  protocol is being deprecated by  $\underline{\mathsf{major}}$  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:
  - Usessh as an alternative to telnet
  - Use sftp, scp or ftps instead of ftp
  - Use https instead of http



Objects should not be created to be dropped immediately without being used



Identical expressions should not be used on both sides of a binary operator



All code should be reachable



Loops with at most one iteration

- 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**

```
url = "http://example.com"; // Sensitive
url = "ftp://anonymous@example.com"; // Sensitive
url = "telnet://anonymous@example.com"; // Sensitive
```

## For nodemailer:

```
const nodemailer = require("nodemailer");
let transporter = nodemailer.createTransport({
   secure: false, // Sensitive
   requireTLS: false // Sensitive
});
```

```
const nodemailer = require("nodemailer");
let transporter = nodemailer.createTransport({}); // Sensiti
```

#### For ftp:

```
var Client = require('ftp');
var c = new Client();
c.connect({
   'secure': false // Sensitive
});
```

#### For telnet-client:

```
const Telnet = require('telnet-client'); // Sensitive
```

#### **Compliant Solution**

```
url = "https://example.com"; // Compliant
url = "sftp://anonymous@example.com"; // Compliant
url = "ssh://anonymous@example.com"; // Compliant
```

For <u>nodemailer</u> one of the following options must be set:

```
const nodemailer = require("nodemailer");
let transporter = nodemailer.createTransport({
   secure: true, // Compliant
   requireTLS: true, // Compliant
   port: 465, // Compliant
   secured: true // Compliant
});
```

### For ftp:

```
var Client = require('ftp');
var c = new Client();
c.connect({
   'secure': true // Compliant
});
```

# Exceptions

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

 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

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.

Privacy Policy