

- 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



## C++ static code analysis

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

All rules 578

Vulnerability 13

Bug 111

Security Hotspot 18

Code Smell 436

Quick Fix 68

Tags

Search by name...



"memset" should not be used to delete sensitive data

Vulnerability

POSIX functions should not be called with arguments that trigger buffer overflows

Vulnerability

XML parsers should not be vulnerable to XXE attacks

Vulnerability

Function-like macros should not be invoked without all of their arguments

Bug

The address of an automatic object should not be assigned to another object that may persist after the first object has ceased to exist

Bug

Assigning to an optional should directly target the optional

Bug

Result of the standard remove algorithms should not be ignored

Bug

"std::scoped\_lock" should be created with constructor arguments

Bug

Objects should not be sliced

Bug

Immediately dangling references should not be created

Bug

"pthread\_mutex\_t" should be unlocked in the reverse order they were locked

Bug

"pthread\_mutex\_t" should be properly initialized and destroyed

Bug

"pthread\_mutex\_t" should not be consecutively locked or unlocked twice

### Using clear-text protocols is security-sensitive

Analyze your code

Security Hotspot Critical cwe symbolic-execution 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
- 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.

 Bug
<b>"std::move" and "std::forward" should not be confused</b>  Bug
<b>A call to "wait()" on a "std::condition_variable" should have a condition</b>  Bug
<b>A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast</b>  Bug
<b>Functions with "noreturn" attribute should not return</b>  Bug
<b>RAII objects should not be temporary</b>  Bug
<b>"memcmp" should only be called with pointers to trivially copyable types with no padding</b>  Bug
<b>"memcpy", "memmove", and "memset" should only be called with pointers to trivially copyable types</b>  Bug
<b>"std::auto_ptr" should not be used</b>  Bug
<b>Destructors should be "noexcept"</b>  Bug

Sensitive Code Example

```
char* http_url = "http://example.com"; // Sensitive
char* ftp_url = "ftp://anonymous@example.com"; // Sensitive
char* telnet_url = "telnet://anonymous@example.com"; // Sensitive
```

```
#include <curl/curl.h>

CURL *curl_ftp = curl_easy_init();
curl_easy_setopt(curl_ftp, CURLOPT_URL, "ftp://example.com/");

CURL *curl_smtp = curl_easy_init();
curl_easy_setopt(curl_smtp, CURLOPT_URL, "smtp://example.com:25");
```

Compliant Solution

```
char* https_url = "https://example.com" # Compliant
char* sftp_url = "sftp://anonymous@example.com" # Compliant
char* ssh_url = "ssh://anonymous@example.com" # Compliant
```

```
#include <curl/curl.h>

CURL *curl_ftps = curl_easy_init();
curl_easy_setopt(curl_ftps, CURLOPT_URL, "ftp://example.com/");
curl_easy_setopt(curl_ftps, CURLOPT_USE_SSL, CURLUSESSL_ALL);

CURL *curl_smtp_tls = curl_easy_init();
curl_easy_setopt(curl_smtp_tls, CURLOPT_URL, "smtp://example.com:25");
curl_easy_setopt(curl_smtp_tls, CURLOPT_USE_SSL, CURLUSESSL_ALL);
```

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:  
 |  Developer Edition