C static code analysis

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

"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

"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
Bug

Functions with "noreturn" attribute should not return
Bug

"memcmp" should only be called with pointers to trivially copyable types with no padding
Bug

Server hostnames should be verified during SSL/TLS connections

Analyze your code

Vulnerability Critical

cwe symbolic-execution full-project privacy owasp ssl

To establish a SSL/TLS connection not vulnerable to man-in-the-middle attacks, it's essential to make sure the server presents the right certificate.

The certificate's hostname-specific data should match the server hostname.

It's not recommended to re-invent the wheel by implementing custom hostname verification.

TLS/SSL libraries provide built-in hostname verification functions that should be used.

Noncompliant Code Example

libcurl

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

CURL *curl;
curl_global_init(CURL_GLOBAL_DEFAULT);

curl = curl_easy_init();
curl_easy_setopt(curl, CURLOPT_URL, "https://example.com/");
curl_easy_setopt(curl, CURLOPT_SSL_VERIFYPEER, 1L);

curl_easy_setopt(curl, CURLOPT_SSL_VERIFYHOST, 0L); // Noncom

//Perform the request
curl_easy_perform(curl);
```

OpenSSL

```
#include <openssl/ssl.h>

SSL_CTX *ctx = get_ctx();
SSL *ssl = SSL_new(ctx);

// ...

// By default hostname validation is disabled
// `SSL_set1_host` is not called
SSL_set_verify(ssl, SSL_VERIFY_PEER, NULL);

// ...

SSL_connect(ssl); // Noncompliant
```

botan

```
#include <botan/tls_client.h>
#include <botan/tls_callbacks.h>
```

```
#include <botan/tls_session_manager.h>
#include <botan/tls_policy.h>
#include <botan/auto_rng.h>
#include <botan/certstor.h>
#include <botan/certstor_system.h>

class Callbacks : public Botan::TLS::Callbacks
{
// ...

virtual void tls_verify_cert_chain(
          const std::vector<Botan::X509_Certificate> &cert_ch
          const std::vector<std::shared_ptr<const Botan::OCSP
          const std::vector<Botan::Certificate_Store *> &trus
          Botan::Usage_Type usage,
          const std::string &hostname,
          const Botan::TLS::Policy &policy) override {} // No
};

class Client_Credentials : public Botan::Credentials_Manager
{
// ...
};

Callbacks callbacks;
Botan::AutoSeeded_RNG rng;
Botan::TLS::Session_Manager_In_Memory session_mgr(rng);
Client_Credentials creds;
Botan::TLS::Strict_Policy policy;

// open the tls connection
Botan::TLS::Client client(callbacks, session_mgr, creds, polic
                          Botan::TLS::Server_Information("exa
                          Botan::TLS::Protocol_Version::TLS_V
```

Compliant Solution

libcurl

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

CURL *curl;
curl_global_init(CURL_GLOBAL_DEFAULT);

curl = curl_easy_init();
curl_easy_setopt(curl, CURLOPT_URL, "https://example.com/");
curl_easy_setopt(curl, CURLOPT_SSL_VERIFYPEER, 1L);

curl_easy_setopt(curl, CURLOPT_SSL_VERIFYHOST, 2L); // Compli

//Perform the request
curl_easy_perform(curl);
```

OpenSSL

```
#include <openssl/ssl.h>

SSL_CTX *ctx = get_ctx();
SSL *ssl = SSL_new(ctx);

// ...

SSL_set1_host(ssl, HOST_NAME); // Compliant
SSL_set_verify(ssl, SSL_VERIFY_PEER, NULL);

// ...

SSL_connect(ssl);
```

botan

```
#include <botan/tls_client.h>
#include <botan/tls_callbacks.h>
#include <botan/tls_session_manager.h>
#include <botan/tls_policy.h>
#include <botan/auto_rng.h>
#include <botan/certstor.h>
#include <botan/certstor_system.h>
```

Stack allocated memory and non-owned memory should not be freed

Bug

Closed resources should not be accessed

Bug

Dynamically allocated memory should be released

Bug

Freed memory should not be used

```
// Compliant use the default implementation of tls_verify_cer
class Callbacks : public Botan::TLS::Callbacks
{
// ...
};

class Client_Credentials : public Botan::Credentials_Manager
{
// ...
};

Callbacks callbacks;
Botan::AutoSeeded_RNG rng;
Botan::TLS::Session_Manager_In_Memory session_mgr(rng);
Client_Credentials creds;
Botan::TLS::Strict_Policy policy;

// open the tls connection
Botan::TLS::Client client(callbacks, session_mgr, creds, poli
                          Botan::TLS::Server_Information("exa
                          Botan::TLS::Protocol_Version::TLS_V
```

See

- OWASP Top 10 2021 Category A2 - Cryptographic Failures
- OWASP Top 10 2021 Category A5 - Security Misconfiguration
- OWASP Top 10 2021 Category A7 - Identification and Authentication Failures
- OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
- OWASP Top 10 2017 Category A6 - Security Misconfiguration
- Mobile AppSec Verification Standard - Network Communication Requirements
- OWASP Mobile Top 10 2016 Category M3 - Insecure Communication
- MITRE, CWE-297 - Improper Validation of Certificate with Host Mismatch

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition