C++ static code analysis: Weak SSL/TLS protocols should not be used

4-6 minutes

This rule raises an issue when an insecure TLS protocol version (i.e. a protocol different from "TLSv1.2", "TLSv1.3", "DTLSv1.2", or "DTLSv1.3") is used or allowed.

It is recommended to enforce TLS 1.2 as the minimum protocol version and to disallow older versions like TLS 1.0. Failure to do so could open the door to downgrade attacks: a malicious actor who is able to intercept the connection could modify the requested protocol version and downgrade

it to a less secure version.

Noncompliant Code Example

libcurl

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

CURL *curl;
curl_global_init(CURL_GLOBAL_DEFAULT);

// CURL_SSLVERSION_DEFAULT is the
default option for CURLOPT_SSLVERSION
// It means legacy version TLSv1 and
TLSv1.1 are enabled
curl = curl_easy_init(); // Noncompliant
curl_easy_setopt(curl, CURLOPT_URL,
"https://example.com/");

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

```
#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_SSLVERSION,
CURL_SSLVERSION_TLSv1); //
Noncompliant;  legacy version TLSv1 and
TLSv1.1 are enabled

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

OpenSSL

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

const SSL_METHOD *method =
TLS_method(); // Noncompliant;  legacy
```

version TLSv1 and TLSv1.1 are enabled

```
SSL_CTX *ctx = SSL_CTX_new(method);

SSL *ssl = SSL_new(ctx);

// ...

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>

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::Policy policy; // Noncompliant:
Default Policy has TLSv1.0 and DLTv1.0 as
minimal versions

// open the tls connection
Botan::TLS::Client client(callbacks,
session_mgr, creds, policy, rng,
```

```
Botan::TLS::Server_Information("example.com",
443),

Botan::TLS::Protocol_Version::TLS_V12);
```

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_SSLVERSION,
CURL_SSLVERSION_TLSv1_2); //
Compliant; enables TLSv1.2 / TLSv1.3
```

version only

// Perform the request
curl_easy_perform(curl);

OpenSSL

#include <openssl/ssl.h>

const SSL_METHOD *method =
TLS_method();

SSL_CTX *ctx = SSL_CTX_new(method);
SSL_CTX_set_min_proto_version(ctx,
TLS1_2_VERSION); // Compliant; enables
TLSv1.2 / TLSv1.3 version only

SSL *ssl = SSL_new(ctx);

// ...

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>

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; //
Compliant: Strict_Policy has TLSv1.2 and
TLSv1.2 as minimal versions

// open the tls connection
Botan::TLS::Client client(callbacks,
session_mgr, creds, policy, rng,

Botan::TLS::Server_Information("example.com",
443),

Botan::TLS::Protocol_Version::TLS_V12);
```

See

- OWASP Top 10 2021 Category A2 - Cryptographic Failures

- 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-327 - Inadequate Encryption Strength
- MITRE, CWE-326 - Use of a Broken or Risky Cryptographic Algorithm
- SANS Top 25 - Porous Defenses
- SSL and TLS Deployment Best Practices - Use secure protocols