wolfSSL QuickStart Guide

Introduction to wolfSSL

The wolfSSL embedded SSL library (formerly CyaSSL) is a lightweight SSL/TLS library written in ANSI C and targeted for embedded, RTOS, and resource-constrained environments - primarily because of its small size, speed, and feature set. It is commonly used in standard operating environments as well because of its royalty-free pricing and excellent cross platform support. wolfSSL supports industry standards up to the current TLS 1.2 and DTLS 1.2 levels, is up to 20 times smaller than OpenSSL, and offers progressive ciphers such as ChaCha20, Curve25519, NTRU, and Blake2b. User benchmarking and feedback reports dramatically better performance when using wolfSSL over OpenSSL.

The wolfSSL library is designed to facilitate secure communication, as well as offering a suite of cryptographic algorithms and a command line tool. In this quickstart guide, we will cover basic installation and setup, as well as simple use cases. For a more comprehensive tour, see the manual linked to in the Helpful Links section towards the bottom. A number of other useful links are also available for those new to the wonderful world of secure communication.

Before You Get Started

Required

- make
- Autoconf
- C compiler

Optional

- git
- Basic knowledge of the C language
- Basic knowledge of server/client communication
- Basic knowledge of SSL/TLS

The more you know, the easier it will be to get going. There are a number of links in the Helpful Links section to read up on SSL/TLS.

Installation and Compilation

The wolfSSL library requires the use of autoconf and a C compiler in order to operate. wolfSSL supports a majority of operating systems, a full list of which can be seen on the wolfSSL wikipedia article.

Getting the Code

Download the source from one of the following locations: wolfSSL webpage or GitHub.

Note: it is necessary to run `bash autogen.sh` if the source is downloaded from the GitHub repo.

Setup

After you have downloaded the source code run the following commands from the wolfssl/ directory:

```
./configure
make
sudo make install
```

`./configure` uses autoconf to configure wolfSSL to default settings; a full list of possible options can be found in the manual. `make` will compile the wolfSSL library. `sudo make install` will allow one to use wolfSSL from anywhere by including the package with `#include <wolfssl/ssl.h>` and linking wolfSSL at compilation (e.g. using the -lwolfssl option).

That's it!

Hello, World! Server/Client Example

Now that wolfSSL has been compiled and installed we can include it and use any of its functions. Here we demonstrate a simple client/server interaction using wolfSSL to secure a message.

client.c

```
#include <wolfssl/options.h>
#include <wolfssl/ssl.h>
#include <wolfssl/test.h>
#include <errno.h>

#define SERV_PORT 11111

int main()
{
    int sockfd;
    WOLFSSL_CTX* ctx;
    WOLFSSL* ssl;
    WOLFSSL_METHOD* method;
    struct  sockaddr_in servAddr;
    const char message[] = "Hello, World!";

    /* create and set up socket */
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    memset(&servAddr, 0, sizeof(servAddr));
    servAddr.sin_family = AF_INET;
    servAddr.sin_port = htons(SERV_PORT);

    /* connect to socket */
    connect(sockfd, (struct sockaddr *) &servAddr, sizeof(servAddr));

    /* initialize wolfssl library */
    wolfSSL_Init();
    method = wolfTLSv1_2_client_method(); /* use TLS v1.2 */

    /* make new ssl context */
    if ( (ctx = wolfSSL_CTX_new(method)) == NULL) {
        err_sys("wolfSSL_CTX_new error");
    }

    /* make new wolfSSL struct */
    if ( (ssl = wolfSSL_new(ctx)) == NULL) {
        err_sys("wolfSSL_new error");
    }

    /* Add cert to ctx */
    if (wolfSSL_CTX_load_verify_locations(ctx, "certs/ca-cert.pem", 0
                SSL_SUCCESS) {
        err_sys("Error loading certs/ca-cert.pem");
    }

    /* Connect wolfssl to the socket, server, then send message */
    wolfSSL_set_fd(ssl, sockfd);
    wolfSSL_connect(ssl);
    wolfSSL_write(ssl, message, strlen(message));

    /* frees all data before client termination */
    wolfSSL_free(ssl);
    wolfSSL_CTX_free(ctx);
    wolfSSL_Cleanup();
}
```

This client attempts to access a server on port 11111, with localhost being the default. Then it attempts to send a single message of "Hello, World!", encrypted, to the server. Then it exits.

server.c

```
#include <wolfssl/options.h>
#include <wolfssl/ssl.h>
#include <wolfssl/test.h>
#include <errno.h>

#define SERV_PORT 11111
#define MAX_LINE 4096

int main(int argc, char** argv)
```

```
{
    int listenfd, connfd;
    WOLFSSL_CTX* ctx;
    WOLFSSL* ssl;
    int n;
    char buf[MAX_LINE];
    WOLFSSL_METHOD* method;

    /* Initialize wolfSSL library */
    wolfSSL_Init();

    /* Get encryption method */
    method = wolfTLSv1_2_server_method();

    /* Create wolfSSL_CTX */
    if ( (ctx = wolfSSL_CTX_new(method)) == NULL)
        err_sys("wolfSSL_CTX_new error");

    /* Load server certs into ctx */
    if (wolfSSL_CTX_use_certificate_file(ctx, "certs/server-cert.pem"
                SSL_FILETYPE_PEM) != SSL_SUCCESS)
        err_sys("Error loading certs/server-cert.pem");

    /* Load server key into ctx */
    if (wolfSSL_CTX_use_PrivateKey_file(ctx, "certs/server-key.pem",
                SSL_FILETYPE_PEM) != SSL_SUCCESS)
        err_sys("Error loading certs/server-key.pem");

    tcp_accept(&listenfd, &connfd, NULL, SERV_PORT, 0, 0, 0);

    /* Create wolfSSL object */
    if ( (ssl = wolfSSL_new(ctx)) == NULL)
        err_sys("wolfSSL_new error");

    wolfSSL_set_fd(ssl, connfd);

    if ( (n = wolfSSL_read(ssl, buf, (sizeof(buf) -1))) > 0) {
        printf("%s\n", buf);
        if (wolfSSL_write(ssl, buf, n) != n)
            err_sys("wolfSSL_write error");
    }
    if (n <0)
        printf("wolfSSL_read error = %d\n", wolfSSL_get_error(ssl, n)
    else if (n == 0)
        printf("Connection close by peer\n");
```

```
                wolfSSL_free(ssl);
                close(connfd);

                wolfSSL_CTX_free(ctx);
                wolfSSL_Cleanup();
                exit(EXIT_SUCCESS);
            }
```

The server waits on the port until it hears a request, and then prints a single message before exiting. To compile this file, it is important to link the wolfSSL library with the `-lwolfssl` option, this is automatically handled by the makefile which can be obtained with this code on GitHub.

Note: If you run into any errors or unexpected behavior, check out the full tutorial found in chapter 11 of the official manual.

Mini API

Below is a small subset of the wolfSSL API. These functions are the ones assumed to be the most valuable to a new user. More information is obtainable in the official manual.

`int wolfSSL_Init(void)`

Initializes the wolfSSL library for use. Must be called once per application and before any other call to the library.

`WOLFSSL_CTX* wolfSSL_CTX_new(WOLFSSL_METHOD* method)`

This function creates a new SSL context, taking a desired SSL/TLS protocol method for input.

`WOLFSSL* wolfSSL_new(WOLFSSL_CTX* ctx)`

This function creates a new SSL session, taking an already created SSL context as input.

`int wolfSSL_set_fd(WOLFSSL* ssl, int fd)`

This function assigns a file descriptor (fd) as the input/output facility for the SSL connection. Typically this will be a socket file descriptor.

`WOLFSSL_METHOD* wolfTLSv1_2_client_method()`

The wolfTLSv1_2_client_method() function is used to indicate that the application is a client and will only support the TLS 1.2 protocol. This function allocates memory for and initializes a new WOLFSSL_METHOD structure to be used when creating the SSL/TLS context with wolfSSL_CTX_new().

```
WOLFSSL_METHOD* wolfTLSv1_2_server_method()
```

The wolfTLSv1_2_server_method() function is used to indicate that the application is a server and will only support the TLS 1.2 protocol. This function allocates memory for and initializes a new WOLFSSL_METHOD structure to be used when creating the SSL/TLS context with wolfSSL_CTX_new().

```
int wolfSSL_connect(WOLFSSL* ssl)
```

This function is called on the client side and initiates an SSL/TLS handshake with a server. When this function is called, the underlying communication channel has already been set up.

```
void wolfSSL_write(WOLFSSL* ssl, const void* data, int sz)
```

This function writes sz bytes from the buffer, data, to the SSL connection, ssl. If necessary, wolfSSL_write() will negotiate an SSL/TLS session if the handshake has not already been performed yet by wolfSSL_connect() or wolfSSL_accept().

```
void wolfSSL_read(WOLFSSL* ssl, void* data, int sz)
```

This function reads sz bytes from the SSL session (ssl) internal read buffer into the buffer data. The bytes read are removed from the internal receive buffer.

```
void wolfSSL_free(WOLFSSL* ssl)
```

This function frees an allocated WOLFSSL object.

```
void wolfSSL_CTX_free(WOLFSSL_CTX* ctx)
```

This function frees an allocated WOLFSSL_CTX object. This function decrements the CTX reference count and only frees the context when the reference count has reached 0.

```
void wolfSSL_Cleanup(void)
```

Un-initializes the wolfSSL library from further use. Doesn't have to be called, though it will free any resources used by the library.

```
int wolfSSL_CTX_use_certificate_file(WOLFSSL_CTX* ctx, const char* file, int
format)
```

This function loads a certificate file into the SSL context (WOLFSSL_CTX). The file is provided by the file argument. The format argument specifies the format type of the file - either SSL_FILETYPE_ASN1 or SSL_FILETYPE_PEM. Please see the examples for proper usage.

```
int wolfSSL_CTX_use_PrivateKey_file(WOLFSSL_CTX* ctx, const char* file, int
format)
```

This function loads a private key file into the SSL context (WOLFSSL_CTX). The file is provided by the file argument. The format argument specifies the format type of the file - SSL_FILETYPE_ASN1or SSL_FILETYPE_PEM. Please see the examples for proper usage.

FAQ

- **Using the echoclient & echoserver with Visual Studio**
  - The echoserver and echoclient are examples found in the examples directory of the wolfSSL source. Check out the tutorial in the links below.
- **Configure options**
  - --enable-opensslextra enables OpenSSL compatability, providing wolfSSL functions with OpenSSL names.
  - --enable-pwdbased enables password based encryption. This is often paired with the opensslextra.
  - --enable-dtls, this enables DTLS which is useful for a UDP connection, or just trying to run the wolfSSL examples.
  - More information on configure options can be found in chapter 2.5 of the manual.
- **Cross compiling**

- When cross compiling, you'll need to specify the host to `./configure`, such as:`./configure --host=arm-linux`
- You may also need to specify the compiler, linker, etc. that you want to use:`./configure --host=arm-linux CC=arm-linux-gcc AR=arm-linux-ar RANLIB=arm-linux`
- More information on cross compiling can be found in chapter 2.6 of the manual.

Helpful Links

In general, these are links which will be useful for using both wolfSSL, as well as just networked and secure applications in general. Furthermore, there is a more comprehensive tutorial that can be found in chapter 11 of the official wolfSSL manual, these examples also do appropriate error checking, which is worth taking a look at. For a more comprehensive API, check out chapter 17 of the official manual.

- wolfSSL official manual
- wolfSSL Wikipedia page
- Comparison of TLS libraries
- TLS Wikipedia page
- OSI model Wikipedia page
- Visual Studio tutorial
- wolfSSL GitHub