wolfSentry Documentation

2024-03-01

Contents

1 はじめに

このマニュアルは、wolfSentry に組み込まれた侵入検知保護システム (IDPS) の技術ガイドとして書かれています。wolfSentry を構築して開始する方法を説明し、構築オプション、機能、移植性の強化、サポートなどの概要を提供します。

1.1 wolfSentry を選ぶ理由

wolfSentry は、サイズとリソースのオーバーヘッドの両方で非常に軽量ですが、組み込みのニーズに対して非常に強力な IDPS です。# wolfSentry のビルド

wolfSentry は移植性を念頭に置いて作成されており、通常、ほとんどのシステムで簡単に構築できるはずです。もしも wolfSentry の構築に問題がある場合は、遠慮なくサポート フォーラムを通じてサポートを求めてください。(https://www.wolfssl.com/forums) または support@wolfssl.com まで直接お問い合わせください。

1.2 wolfSentry ソースコードの取得

wolfSentry の最新バージョンは、wolfSSL Web サイトから ZIP ファイルとしてダウンロードできます。

https://wolfssl.jp/download/

ZIP ファイルをダウンロードしたら、「unzip」コマンドを使用してファイルを解凍します。ネイティブの行末を使用するには、unzip を使用する場合は、 -a 修飾子を有効にします。unzip のマニュアル ページから、-a 修飾子の機能は次のように説明されています：

> -a オプションを使用すると、zip によってテキスト ファイルとして識別されるファイル (zipinfo リストで「b」ではなく「t」ラベルが付いているファイル) が自動的に抽出され、行末、ファイル終了文字、および文字が変換されます。必要に応じて設定します。リスト、'b' ではなく) そのように自動的に抽出され、行末を変換し、end- ファイルの文字と、必要に応じて文字セット自体。[...]

1.3 依存関係

デフォルトのビルドでは、wolfSentry は POSIX ランタイム、特にヒープ アロケータ、clock_gettime、stdio、セマフォ、および文字列 API に依存しています。ただし、これらの依存関係は、さまざまなビルド時のオプションで回避できます。特に、次の指定を用いると：

```
make STATIC=1 SINGLETHREADED=1 NO_STDIO=1
  ↪ EXTRA_CFLAGS='-DWOLFSENTRY_NO_CLOCK_BUILTIN
  ↪ -DWOLFSENTRY_NO_MALLOC_BUILTIN'
```

少数の基本的な文字列関数のみに依存する libwolfsentry.a を生成します。次に、アロケータと時間のコールバックを wolfsentry_init() に渡される wolfsentry_host_platform_interface 構造体に '設定する必要があります。

1.4 ビルドとテスト

GNU Make をサポートするプラットフォームでは、make を実行すると、通常どおり wolfSentry がビルドされます。他のプラットフォームでは、src ディレクトリとその json サブディレクトリにある C ファイルを使用してコンパイルする必要があります。

テスト スイートをビルドして実行するには、make -j test または make V=1 -j test を詳細に使用できます。

1.4.1 ビルド オプション

make に追加できるいくつかのフラグと、ヘッダー ファイルまたは CFLAGS のいずれかでビルド時の定義として使用されるフラグがあります。make フラグを使用するには、以下を使用できます。

```
make SINGLETHREADED=1 EXTRA_CFLAGS='-DWOLFSENTRY_NO_CLOCK_BUILTIN'
```

これらは、ビルド時に wolfSentry がビルドされる wolfsentry_options.h ファイルに保存されます。make を使用していない場合は、次のテンプレートを使用してこのファイルを作成できます。

```
#ifndef WOLFSENTRY_OPTIONS_H
#define WOLFSENTRY_OPTIONS_H

#endif /* WOLFSENTRY_OPTIONS_H */
```

次の表に、可能なオプションを示します。

make オプション	説明
ーーーーー	ーーーー
'V'	詳細な make 出力

|| 例えば make V=1 -j test| | | USER_MAKE_CONF | 含めるユーザー定義の Makefile | || 例えば make -j USER_MAKE_CONF=Makefile.settings | | | SRC_TOP | ソース コードの最上位ディレクトリ (デフォルトは「pwd -P」) | | | BUILD_TOP| 別の場所 (ソース ツリーの外部またはサブディレクトリ内) にアーティファクトを使用してビルドする | || 例えば make BUILD_TOP=./build -j test| | | デバッグ | 使用するコンパイラ デバッグ フラグ (デフォルトは -ggdb) | | | オプティム | 使用するオプティマイザ フラグ (デフォルトは -O3) | | | C_WARNFLAGS | 使用する警告フラグ (デフォルト [1] を参照) | | | NO_STDIO | プラットフォームに「STDIO」がありません | | | NO_JSON | JSON 構成サポートをコンパイルしないでください | | | USER_SETTINGS_FILE| インクルードする追加のヘッダー ファイル | | | シングルヘッド | シングル スレッドの使用にスレッド セーフ セマンティクスを使用しない | || 例えば make -j SINGLETHREADED=1 テスト | | | 静的 | 静的バイナリをビルドする | | | 剥ぎ取られた | デバッグ シンボルのバイナリを削除 | | | BUILD_DYNAMIC | 動的ライブラリを構築 | | | VERY_QUIET | 非常に静かなビルドを有効にする | | | TAR | make dist の GNU tar バイナリへのパス。macOS では gtar に設定する必要があります | | | バージョン | コンパイルするバージョン番号 (デフォルトは [2] を参照) |

プリプロセッサ マクロ	説明
WOLFSENTRY_NO_STDIO	プラットフォームに「STDIO」がありません
WOLFSENTRY_NO_JSON	JSON 構成サポートをコンパイルしないでください
WOLFSENTRY_USER_SETTINGS_FILE	インクルードする追加のヘッダー ファイル
WOLFSENTRY_SINGLETHREADED	シングル スレッドの使用にスレッド セーフ セマンティクスを使用しない
CENTIJSON_USE_LOCALE	JSON パーサーはロケール依存の文字を使用する必要があります
WOLFSENTRY_NO_PROTOCOL_NAMES	構成ファイルのプロトコル名のサポートを無効にする
DEBUG_JSON	JSON パーサーにデバッグ printf() を追加
WOLFSENTRY_NO_ERROR_STRINGS	エラー文字列関数へのエラー コードを無効にする
WOLFSENTRY_NO_MALLOC_BUILTINS	組み込みの malloc 関数を無効にする

[1] -Wall -Wextra -Werror -Wformat=2 -Winit-self -Wmissing-include-dirs -Wunknown-pragmas -Wshadow -Wpointer-arith -Wcast-align -Wwrite-strings -Wconversion -Wstrict-prototypes -Wold-style-definition -Wmissing-declarations -Wmissing-format-attribute -Wpointer-arith -Woverlength-strings -Wredundant-decls -Winline -Winvalid-pch -Wdouble-promotion -Wvla -Wno-missing-field-initializers -Wno -bad-function-cast -Wno-type-limits および GCC が使用されている場合は、-Wjump-misses-init -Wlogical-op の追加フラグが使用されます。

[2] VERSION として定義:= $(shell git rev-parse --short=8 HEAD 2>/dev/null || echo xxxxxxxx)$(shell git diff --quiet 2>/dev/null | | [$$? -ne 1] || echo "-dirty")

プリプロセッサ マクロ	説明
WOLFSENTRY_HAVE_NONGNU_ATOMICS	アトミックは非 GNU です (SINGLETHREADED が設定されている場合は無視されます)
WOLFSENTRY_NO_CLOCK_BUILTIN	Bulitin 時間関数を使用しないでください
WOLFSENTRY_LWIP	wolfSentry は BSD ソケットではなく LWIP に対して構築されています
FREERTOS	FreeRTOS サポートでビルド

1.4.2 ビルドオプションの例

別のビルド場所から非標準の場所にインストールします。

```
$make BUILD_TOP=./build INSTALL_DIR=/usr INSTALL_LIBDIR=/usr/lib64 インストール
```

libwolfsentry.a をビルドし、さまざまなアナライザー (メモリとスレッド valgrind およびサニタイザー テストのフル バッテリー下でのテスト):

```
$make -j check
```

マルチスレッドをサポートせずに libwolfsentry.a をビルドしてテストします。

```
$make -j SINGLETHREADED=1 test
```

その他の使用可能な make フラグは、「STATIC=1」、「STRIPPED=1」、「NO_JSON=1」、および「NO_JSON_DOM=1」、および「DEBUG」、「OPTIM」、および「C_WARNFLAGS」のデフォルト値 便利にオーバーライドすることもできます。

ユーザー提供の makefile プリアンブルを使用してビルドし、デフォルトをオーバーライドします。

```
$make -j USER_MAKE_CONF=Makefile.settings
```

(Makefile.settings には、OPTIM := -Os のような単純な設定を含めることができます。追加の規則と依存メカニズムを含む、複雑な makefile コード)。

可能な限り最小で最も単純なライブラリを構築します。

```
make -j SINGLETHREADED=1 NO_STDIO=1 DEBUG= OPTIM=-Os
 ↪  EXTRA_CFLAGS='-DWOLFSENTRY_NO_CLOCK_BUILTIN -DWOLFSENTRY_NO_MALLOC_BUILTIN
 ↪  -DWOLFSENTRY_NO_ERROR_STRINGS -Wno-error=inline -Wno-inline'
```

ユーザー設定を使用してビルドおよびテストします。

```
$make -j USER_SETT
```

```
## wolfSSL と統合したサンプルプログラム

[the wolfSSL repository](https://github.com/wolfSSL/wolfssl) で、
 ↪  `WOLFSSL_WOLFSENTRY_HOOKS` でゲートされた `wolfsentry/test.h` のコードを参照して
 ↪  ください。
`wolfsentry_store_endpoints()`、`wolfSentry_NetworkFilterCallback()`、
 ↪  `wolfsentry_setup()`、および`tcp_connect_with_wolfSentry()`。次のコードも参照し
 ↪  てください。
`examples/server/server.c` と `examples/client/client.c` は
 ↪  `WOLFSSL_WOLFSENTRY_HOOKS` でゲートされています。 `configure
 ↪  --enable-wolfsentry` を使用してビルドします。
wolfSentry 統合、および wolfSentry が非標準の場所にインストールされている場合は、
 ↪  `--with-wolfsentry=/the/install/path` を使用します。 wolfSSL テスト
クライアント/サーバーは、ユーザー提供の wolfSentry JSON 構成でロードできます。
```

コマンドラインから、`--wolfsentry-config <file>` を使用して。

```sh
$ ./examples/server/server -b -i
wolfSentry got network filter callback: family=2 proto=6 rport=52666
 ↪  lport=11111 raddr=127.0.0.1 laddr=127.0.0.1 interface=0; decision=1
 ↪  (ACCEPT)
SSL version is TLSv1.2
SSL cipher suite is TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
SSL curve name is SECP256R1
Client message: hello wolfssl!

$ ./examples/client/client
wolfSentry callin from tcp_connect_with_wolfSentry: family=2 proto=6
 ↪  rport=11111 lport=0 raddr=127.0.0.1 laddr=0.0.0.0 interface=0; decision=1
 ↪  (ACCEPT)
wolfSentry got network filter callback: family=2 proto=6 rport=11111 lport=0
 ↪  raddr=127.0.0.1 laddr=0.0.0.0 interface=0; decision=1 (ACCEPT)
SSL version is TLSv1.2
SSL cipher suite is TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
SSL curve name is SECP256R1
I hear you fa shizzle!
```

1.5 wolfSentry のサンプルプログラム

サンプルプログラムは wolfSentry rero、<wolfSentry root>/examples にあります。

1.5.1 wolfSentry lwIP Echo テスト

これは非常に基本的なデモ アプリケーションであり、Docker 上の PCAP を介して lwIP スタックで wolfSentry を使用し、送信されたものをすべてエコーします。wolfSentry は、テストで生成された 3 つの Docker ノードを使用してテストできるさまざまなタイプのトラフィックをフィルタリングします。

1.5.1.1 前提条件 このテストは Linux または macOS で実行するように設計されていますが、Windows でも動作するはずです。コンピューターに次のものがインストールされている必要があります。

- Docker - https://docs.docker.com/get-docker/
- docker-compose - https://docs.docker.com/compose/install/

1.5.1.2 実行中の サーバー 次のコマンドは、テスト Echo サーバーを構築し、3 つのテスト ノードと共にこれを起動します。

```
sudo docker-compose -f docker-compose.yml up --build -d
```

以下を使用して、Echo サーバーのログを追跡できます。

```
sudo docker-compose -f docker-compose.yml log -f
```

テストの実行中は、ログを実行したままにしておくことをお勧めします。

1.5.1.3 テスト

1.5.1.3.1 ノードへのアクセス 操作する 3 つのユーザー テスト ノードがあります。それらは linux-lwip-tester?-1 という名前で、? は 1、2、または 3 です。サンプルプログラムとして tester2 にログインするには:

```
sudo docker exec -it linux-lwip-tester2-1 /bin/sh
```

1.5.1.3.2 Ping テスト 以下を使用して、任意のノードから ping を実行できます。

```
ping 127.20.20.5
```

テスター ノード 1 は動作し、テスター 2 は ICMP ping で拒否され、テスター 3 は MAC アドレスで拒否されます。これはログ出力に反映されます。

1.5.1.3.3 Echo テスト 以下を使用して、任意のノードから接続できます。

```
nc -v 172.20.20.5 11111
```

テスト済みのノード 2 は動作し、netcat ターミナルに入力した内容はすべてサーバー ログに記録されます。テスター 1 は TCP 接続で拒否され、テスター 3 は MAC アドレスで拒否されます。

1.5.1.4 ノードの詳細

1.5.1.4.1 エコーサーバー

- IP アドレス: 172.20.20.3 (ノード) 172.20.20.5 (エコー プロセス)
- MAC アドレス: de:c0:de:01:02:03

エコー テスト プロセスはこのノードから実行され、PCAP と lwIP を使用して、実際のテスト用に 127.20.20.5 の静的 IP を作成します。

1.5.1.4.2 テスター 1 ※IP アドレス：172.20.20.10 * MAC アドレス: de:c0:de:03:02:01

セントリー テストは、このノードが echoserver ノードに ping を実行できるように構成されていますが、ハンドシェイク中に TCP 接続が受け入れられません。

linux-lwip-tester1-1 ログの例:

```
$ sudo docker exec -it linux-lwip-tester1-1 /bin/sh
/ # ping 172.20.20.5 -c 3
PING 172.20.20.5 (172.20.20.5): 56 data bytes
64 bytes from 172.20.20.5: seq=0 ttl=255 time=193.580 ms
64 bytes from 172.20.20.5: seq=1 ttl=255 time=25.759 ms
64 bytes from 172.20.20.5: seq=2 ttl=255 time=62.264 ms

--- 172.20.20.5 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 25.759/93.867/193.580 ms
/ # nc -v 172.20.20.5 11111
^Cpunt!

/ #
```

linux-lwip-echoserver-1 ログの例:

```
...
// Accept PING
linux-lwip-echoserver-1  | Ping accepted from 172.20.20.10
```



```
linux-lwip-echoserver-1  | MAC Sentry action returned code 0 (OK, operation
 ↪  succeeded), src 4 (routes.c), line 1107
linux-lwip-echoserver-1  | Sentry accepted MAC address DE:C0:DE:03:02:01
linux-lwip-echoserver-1  | PING Sentry action returned code 0 (OK, operation
 ↪  succeeded), src 4 (routes.c), line 1107
linux-lwip-echoserver-1  | Ping accepted from 172.20.20.10
linux-lwip-echoserver-1  | MAC Sentry action returned code 0 (OK, operation
 ↪  succeeded), src 4 (routes.c), line 1107
linux-lwip-echoserver-1  | Sentry accepted MAC address DE:C0:DE:03:02:01

....
// Reject TCP
linux-lwip-echoserver-1  | Sentry rejected connection from: 172.20.20.10
linux-lwip-echoserver-1  | MAC Sentry action returned code 0 (OK, operation
 ↪  succeeded), src 4 (routes.c), line 1107
linux-lwip-echoserver-1  | Sentry accepted MAC address DE:C0:DE:03:02:01
linux-lwip-echoserver-1  | Incomming connection from: 172.20.20.10
linux-lwip-echoserver-1  | TCP Sentry action returned code 0 (OK, operation
 ↪  succeeded), src 4 (routes.c), line 1064
linux-lwip-echoserver-1  | Sentry rejected connection from: 172.20.20.10
```

1.5.1.4.3 テスター 2 ※IP アドレス：172.20.20.20 * MAC アドレス: de:c0:de:03:02:02

セントリー テストは、このノードがエコー サーバー ノードに ping を送信するのをブロックするように構成されていますが、TCP 接続はハンドシェイク中に受け入れられます。

linux-lwip-tester2-1 ログの例:

```
$ sudo docker exec -it linux-lwip-tester2-1 /bin/sh
/ # ping 172.20.20.3 -c 3
PING 172.20.20.3 (172.20.20.3): 56 data bytes
64 bytes from 172.20.20.3: seq=0 ttl=64 time=0.230 ms
64 bytes from 172.20.20.3: seq=1 ttl=64 time=0.608 ms
64 bytes from 172.20.20.3: seq=2 ttl=64 time=0.323 ms


--- 172.20.20.3 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.230/0.387/0.608 ms
/ # ping 172.20.20.5 -c 3
PING 172.20.20.5 (172.20.20.5): 56 data bytes
^C
--- 172.20.20.5 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
/ # nc -v 172.20.20.5 11111
172.20.20.5 (172.20.20.5:11111) open
hello
^C

/ #
```

linux-lwip-echoserver-1 ログの例:

```
// Accept PING
linux-lwip-echoserver-1  | Ping accepted from 172.20.20.10
```

```
linux-lwip-echoserver-1  | MAC Sentry action returned code 0 (OK, operation
 ↪ succeeded), src 4 (routes.c), line 1107
linux-lwip-echoserver-1  | Sentry accepted MAC address DE:C0:DE:03:02:01
linux-lwip-echoserver-1  | PING Sentry action returned code 0 (OK, operation
 ↪ succeeded), src 4 (routes.c), line 1107
linux-lwip-echoserver-1  | Ping accepted from 172.20.20.10
linux-lwip-echoserver-1  | MAC Sentry action returned code 0 (OK, operation
 ↪ succeeded), src 4 (routes.c), line 1107
linux-lwip-echoserver-1  | Sentry accepted MAC address DE:C0:DE:03:02:01

....
// Reject TCP
linux-lwip-echoserver-1  | Sentry rejected connection from: 172.20.20.10
linux-lwip-echoserver-1  | MAC Sentry action returned code 0 (OK, operation
 ↪ succeeded), src 4 (routes.c), line 1107
linux-lwip-echoserver-1  | Sentry accepted MAC address DE:C0:DE:03:02:01
linux-lwip-echoserver-1  | Incomming connection from: 172.20.20.10
linux-lwip-echoserver-1  | TCP Sentry action returned code 0 (OK, operation
 ↪ succeeded), src 4 (routes.c), line 1064
linux-lwip-echoserver-1  | Sentry rejected connection from: 172.20.20.10
```

1.5.1.4.4　テスター 3　※IP アドレス：172.20.20.30 * MAC アドレス: `de:c0:de:03:03:01`

セントリー テストは、この MAC アドレスからのトラフィックを拒否するように構成されています。

linux-lwip-tester3-1 ログの例:

```
$ sudo docker exec -it linux-lwip-tester3-1 /bin/sh
/ # ping 172.20.20.5 -c 3
PING 172.20.20.5 (172.20.20.5): 56 data bytes
^C
--- 172.20.20.5 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
/ # nc -v 172.20.20.5 11111
nc: 172.20.20.5 (172.20.20.5:11111): Host is unreachable
```

linux-lwip-echoserver-1 ログの例:

```
// Reject PING
linux-lwip-echoserver-1  | MAC Sentry action returned code 0 (OK, operation
 ↪ succeeded), src 4 (routes.c), line 1116
linux-lwip-echoserver-1  | Sentry rejected MAC address DE:C0:DE:03:03:01
linux-lwip-echoserver-1  | MAC Sentry action returned code 0 (OK, operation
 ↪ succeeded), src 4 (routes.c), line 1116
linux-lwip-echoserver-1  | Sentry rejected MAC address DE:C0:DE:03:03:01
linux-lwip-echoserver-1  | MAC Sentry action returned code 0 (OK, operation
 ↪ succeeded), src 4 (routes.c), line 1116
linux-lwip-echoserver-1  | Sentry rejected MAC address DE:C0:DE:03:03:01

// Rject TCP
linux-lwip-echoserver-1  | MAC Sentry action returned code 0 (OK, operation
 ↪ succeeded), src 4 (routes.c), line 1116
linux-lwip-echoserver-1  | Sentry rejected MAC address DE:C0:DE:03:03:01
linux-lwip-echoserver-1  | MAC Sentry action returned code 0 (OK, operation
 ↪ succeeded), src 4 (routes.c), line 1116
linux-lwip-echoserver-1  | Sentry rejected MAC address DE:C0:DE:03:03:01
```

```
linux-lwip-echoserver-1  | MAC Sentry action returned code 0 (OK, operation
 ↪  succeeded), src 4 (routes.c), line 1116
linux-lwip-echoserver-1  | Sentry rejected MAC address DE:C0:DE:03:03:01
```

1.5.1.5 シャットダウン 以下を実行して、ノードを停止してクリーンアップできます。これにより、仮想ネットワークも削除されます。

```
sudo docker-compose -f docker-compose.yml down
```

ノート

- lwip-include/arch ディレクトリは、contrib/ports/unix/port/include/arch の lwIP ディレクトリのコピーです。

1.6 wolfSentry STM32 LWIP のサンプルプログラム

これは、このコードベースの Linux の例と非常によく似たデモ アプリケーションです。TCP エコー、ICMP ping、および MAC アドレスでフィルタリングできます。OS には FreeRTOS、ネットワークスタックには LWIP を備えた STM32 を使用しています。

このサンプルプログラムは、STM32CubeMX および STM32CubeIDE で使用するように設計されています。

1.6.1 STM32CubeMX のセットアップ

STM32CubeMX でプロジェクトを作成して FREERTOS と LWIP を有効にするには、ハードウェアに応じて「ETH」接続オプションを構成する必要もあります。ICMP ping フィルタリングを行う場合は、[キー オプション] に移動し、[高度なパラメータを表示] を選択して、LWIP_RAW を有効にします。

ノート： 1. 特に LWIP で DHCP を実行している場合は、FREERTOS の「タスクとキュー」で「defaultTask」スタック サイズを増やすことをお勧めします。8KB で問題なく動作するはずです。

2. CMSISv1 を使用し、STM32Cube のバージョンの LWIP ではマルチスレッド サポートを使用しないことをお勧めします。これにより、安定性の問題が発生する可能性があります。

1.6.2 ファイルのコピー

1. プロジェクトのベースに wolfsentry git checkout をコピー (または新しいクローンを作成) します。
2. この README の隣にある Src ディレクトリのファイルをプロジェクトの Core/Src ディレクトリにコピーします。
3. この README の隣にある Inc ディレクトリのファイルをプロジェクトの Core/Inc ディレクトリにコピーします。

1.6.3 STM32CubeIDE のセットアップ

STM32CubeIDE で、[プロジェクト] -> [プロパティ] をクリックします。ここから「C/C++ General」「Paths and Symbols」へ。[ソースの場所] タブをクリックし、[フォルダーの追加] をクリックします。wolfSentry をビルド チェーンに追加する “wolfsentry” を選択できます。追加したら、[フィルターの編集] をクリックし、[追加...] をクリックして、次を追加します。

```
**/unittests.c
**/examples
```

これは、単体テストがアプリケーションの一部としてビルドされないようにするためであり、main() で競合が発生します。

1.6.3.1 コードの変更

1.6.3.2 wolfsentry_options.h 「wolfsentry/wolfsentry」で、「wolfsentry_options.h」ファイルを作成します。

‘

```
#ifndef WOLFSENTRY_OPTIONS_H
#define WOLFSENTRY_OPTIONS_H

#define FREERTOS
//#define WOLFSENTRY_SINGLETHREADED
#define WOLFSENTRY_LWIP
#define WOLFSENTRY_NO_PROTOCOL_NAMES
#define WOLFSENTRY_NO_POSIX_MEMALIGN
#endif /* WOLFSENTRY_OPTIONS_H */
```

シングル スレッドを使用している場合は、WOLFSENTRY_SINGLETHREADED のコメントを外します。これにより、セマフォが不要になります。

1.6.3.3 main.c main() で、必要に応じて以下を追加し、コメント ブロックを使用してコードを配置する場所を見つけます。

ファイルの上部付近:

```
/* USER CODE BEGIN インクルード */

#include "echo.h"
#include "ping.h"
```

StartDefaultTask() で:

```
  /* USER CODE BEGIN 5 */
  printf("Sentry init\n");
  sentry_init();
  printf("Echo init\n");
  echo_init();
  printf("Ping init\n");
  ping_init();
```

1.6.3.4 sentry.c セントリー構成はファイルの先頭にあります。ping、TCP、および MAC アドレスの構成を確認できます。必要に応じて、これらのアドレス/プレフィックス範囲を編集します。

1.6.3.5 ethernetif.c このファイルへの変更は、MAC アドレス フィルタリングのために必要です。MAC アドレス フィルタリングを行いたくない場合は、これらの変更を行う必要はありません。

このファイルは「LWIP/Target」にあります。コードの上部近くに次を追加します。

```
/* ユーザーコード BEGIN 0 */
#include "sentry.h"
/* MAC アドレス フィルタリングに使用される raw 入力パケット コールバック */
static err_t filter_input(struct pbuf *p, struct netif *inp)
{
     /* ペイロードの先頭にはイーサネット ヘッダーが含まれます */
     struct eth_hdr *ethhdr = (struct eth_hdr *)p-> ペイロード;
     struct eth_addr *ethaddr = &ethhdr->src;

     /* 「src」には、パケットからの送信元ハードウェア アドレスが含まれます */
     if (sentry_action_mac(&ethhdr->src) != 0)
```

```
    {
        /* printf("Sentry は MAC アドレス %02X:%02X:%02X:%02X:%02X:%02X を拒否しま
         ↪  した\n",
                ethaddr->addr[0]、ethaddr->addr[1]、ethaddr->addr[2]、
                ethaddr->addr[3]、ethaddr->addr[4]、ethaddr->addr[5]); */

        /* 基本的にパケットをドロップします */
        ERR_ABRT を返します。
    }

    printf("Sentry は MAC アドレス %02X:%02X:%02X:%02X:%02X:%02X を受け入れまし
↪   た\n",
        ethaddr->addr[0]、ethaddr->addr[1]、ethaddr->addr[2]、
        ethaddr->addr[3]、ethaddr->addr[4]、ethaddr->addr[5]);

    /* MAC フィルタを通過したので、パケットを通常の内部に渡します
     * lwIP 入力コールバック */
    return netif_input(p, inp);
}
```

関数 ethernetif_init() を見つけて、#if LWIP_ARP の下に以下を追加します。

```
    netif->input = filter_input;
```

STM32CubeMX を使用してコードを再生成すると、上記の最後の編集が消去されることに注意してください。

最後に、同じディレクトリにある lwipopts.h を開き、USER CODE BEGIN 0 セクション内に次を追加します。

```
#include "echo.h"
#define LWIP_HOOK_TCP_INPACKET_PCB sentry_tcp_inpkt
```

1.6.3.6 アプリケーションの使用

実行すると、ポート 11111 で TCP 接続できるようになり、そこに送信されたものはすべて printf() を介してエコーされます (ブロックされていない場合)。フィルタリングは ping にも適用され、着信パケットは MAC アドレスに基づいてフィルタリングされます。

1.7 wolfSentry STM32 LWIP wolfSSL のサンプルプログラム

これは、非常に基本的な HTTPS サーバーを起動するデモ アプリケーションです。特定の IP からポート 8080 で HTTPS 接続を受け入れるように設計されています。他の IP からの接続はブロックされます。wolfSSL 側では、データの送受信にネイティブ LWIP コードが使用されます。

この例は、STM32CubeIDE で使用するように設計されています。

1.7.1 STM32CubeMX のセットアップ

STM32CubeMX でプロジェクトを作成して FREERTOS と LWIP を有効にするには、ハードウェアに応じて「ETH」接続オプションを構成する必要もあります。

ノート： 1. 特に LWIP で DHCP を実行している場合は、FREERTOS の「タスクとキュー」で「defaultTask」スタック サイズを増やすことをお勧めします。8KB で問題なく動作するはずです。

2. CMSISv1 を使用し、STM32Cube のバージョンの LWIP ではマルチスレッド サポートを使用しないことをお勧めします。別の構成では、安定性の問題が発生する可能性があります。

1.7.2 ファイルのコピー

1. プロジェクトのベースに wolfsentry git checkout をコピー (または新しいクローンを作成) します。
2. この README の隣にある Src ディレクトリのファイルをプロジェクトの Core/Src ディレクトリにコピーします。
3. この README の隣にある Inc ディレクトリのファイルをプロジェクトの Core/Inc ディレクトリにコピーします。

1.7.3 STM32CubeIDE のセットアップ

STM32CubeIDE で、[プロジェクト] -> [プロパティ] をクリックします。ここから「C/C++ General」「Paths and Symbols」へ。[ソースの場所] タブをクリックし、[フォルダーの追加] をクリックします。wolfSentry をビルド チェーンに追加する “wolfsentry” を選択できます。追加したら、[フィルターの編集] をクリックし、[追加...] をクリックして、次を追加します。

```
**/unittests.c
**/examples
```

これは、単体テストがアプリケーションの一部としてビルドされないようにするためであり、main() で競合が発生します。

1.7.4 コードの変更

1.7.4.1 wolfsentry_options.h 「wolfsentry/wolfsentry」で、「wolfsentry_options.h」ファイルを作成します。

```
#ifndef WOLFSENTRY_OPTIONS_H
#define WOLFSENTRY_OPTIONS_H

#define FREERTOS
#define WOLFSENTRY_SINGLETHREADED
#define WOLFSENTRY_LWIP
#define WOLFSENTRY_NO_PROTOCOL_NAMES
#define WOLFSENTRY_NO_POSIX_MEMALIGN
#endif /* WOLFSENTRY_OPTIONS_H */
```

1.7.4.2 main.c main() で、必要に応じて以下を追加し、コメント ブロックを使用してコードを配置する場所を見つけます。

ファイルの上部付近:

```
/* USER CODE BEGIN インクルード */

#include "echo.h"
#include "sentry.h"
```

StartDefaultTask() で:

```
/* USER CODE BEGIN Includes */

#include "echo.h"
#include "sentry.h"
```

In StartDefaultTask():

```
  /* USER CODE BEGIN 5 */
  printf("Start!\r\n");
  printf("Sentry init\n");
```

```
  sentry_init();
  printf("Echo init\n");
  echo_ssl();

  connQueue = xQueueCreate( 10, sizeof( struct thread_data* ) );
  echo_init();

  /* Infinite loop */
  for(;;)
  {
        BaseType_t qRet = pdFALSE;
        struct thread_data tdata;
        while (qRet != pdTRUE) {
          qRet = xQueueReceive( connQueue, &( tdata ), ( TickType_t ) 10 );
        }
        char buff[256];
        int ret;
        int retry = 10;
        struct tcp_pcb *pcb = tdata.pcb;
        WOLFSSL *ssl = tdata.ssl;

        fprintf(stderr, "Queue item running\r\n");
        do {
            if (pcb->state == CLOSE_WAIT) {
                fprintf(stderr, "Client immediately hung-up\n");
                goto close_wait;
            }
            ret = wolfSSL_accept(ssl);
            if ((wolfSSL_want_read(ssl) || wolfSSL_want_write(ssl))) {
                osDelay(500);
                retry--;
            } else {
                retry = 0;
            }
        } while (retry);
        if (ret != WOLFSSL_SUCCESS) {
            fprintf(stderr, "wolfSSL_accept ret = %d, error = %d\n",
                ret, wolfSSL_get_error(ssl, ret));
            goto ssl_shutdown;
        } else {
            fprintf(stderr, "Handshake done!\n");
        }

        memset(buff, 0, sizeof(buff));
        if (ret == WOLFSSL_SUCCESS) {
            retry = 10;
            do {
                ret = wolfSSL_read(ssl, buff, sizeof(buff));
                if ((wolfSSL_want_read(ssl) || wolfSSL_want_write(ssl))) {
                    osDelay(500);
                    retry--;
                } else {
                    retry = 0;
                }
```

```
            } while (retry);
            if (ret == -1) {
                fprintf(stderr, "ERROR: failed to read\n");
                goto ssl_shutdown;
            }
            else
            {
                fprintf(stderr, "Sending response\n");
                if ((ret = wolfSSL_write(ssl, response, strlen(response))) !=
                 ↪  strlen(response)) {
                    fprintf(stderr, "ERROR: failed to write\n");
                }
            }
        }

ssl_shutdown:
        retry = 10;
        do {
            ret = wolfSSL_shutdown(ssl);
            if (ret == SSL_SHUTDOWN_NOT_DONE) {
                osDelay(500);
                retry--;
            } else {
                break;
            }
        } while (retry);

close_wait:
        fprintf(stderr, "Connection closed\n");
        wolfSSL_free(ssl);
  }
  /* USER CODE END 5 */
```

1.7.4.3 sentry.c セントリー構成はファイルの先頭にあります。必要に応じてアドレス/プレフィックス範囲を編集します。

1.7.4.4 ethernetif.c このファイルへの変更は、MAC アドレス フィルタリングのために必要です。MAC アドレス フィルタリングを行いたくない場合は、これらの変更を行う必要はありません。

このファイルは「LWIP/Target」にあります。コードの上部近くに次を追加します。

```
/* USER CODE BEGIN 0 */
#include "sentry.h"
/* Raw input packet callback used for MAC address filtering */
static err_t filter_input(struct pbuf *p, struct netif *inp)
{
    /* Start of payload will have an Ethernet header */
    struct eth_hdr *ethhdr = (struct eth_hdr *)p->payload;
    struct eth_addr *ethaddr = &ethhdr->src;

    /* "src" contains the source hardware address from the packet */
    if (sentry_action_mac(&ethhdr->src) != 0)
    {
       /* printf("Sentry rejected MAC address %02X:%02X:%02X:%02X:%02X:%02X\n",
```

```
                ethaddr->addr[0], ethaddr->addr[1], ethaddr->addr[2],
                ethaddr->addr[3], ethaddr->addr[4], ethaddr->addr[5]); */

        /* Basically drop the packet */
        return ERR_ABRT;
    }

    printf("Sentry accepted MAC address %02X:%02X:%02X:%02X:%02X:%02X\n",
        ethaddr->addr[0], ethaddr->addr[1], ethaddr->addr[2],
        ethaddr->addr[3], ethaddr->addr[4], ethaddr->addr[5]);

    /* We passed the MAC filter, so pass the packet to the regular internal
     * lwIP input callback */
    return netif_input(p, inp);
}
```

関数 ethernetif_init() を見つけて、#if LWIP_ARP の下に以下を追加します。

```
     netif->input = filter_input;
```

STM32CubeMX を使用してコードを再生成すると、上記の最後の編集が消去されることに注意してください。

最後に、同じディレクトリにある lwipopts.h を開き、USER CODE BEGIN 0 セクション内に次を追加します。

```
#include "echo.h"
#define LWIP_HOOK_TCP_INPACKET_PCB sentry_tcp_inpkt
```

1.7.4.5 アプリケーションの使用 実行すると、HTTP クライアントを使用してポート 8080 で TCP 接続できるようになります。ブロックされている場合はタイムアウトになり、UART に詳細が表示されます。それ以外の場合、SSL ハンドシェイクが発生し、HTTP 応答が提供されます。