Softwareprojekt

Inhaltsverzeichnis

1 Einleitung

Die Sicherheit von eingebetteten System und dem Internet of Things gewinnt immer weiter an Bedeutung. In naher Zukunft sollen Milliarden von Geräten ans Netz gehen und damit neue Sicherheitsstandards fordern.
Dieses Softwareprojekt beschäftigt sich mit der PSA-Initiative und der aktuellen Trusted-Firmware-Version. Die Firmware wird dabei auf dem *STM NUCLEO L552ZE-Q -Board* gespielt um verschiedene Angriffe und Sicherheitsfunktionalitäten zu testen.
Ziel ist die Lücken der Funktionalitäten ausnutzen und so beispielsweise bestimmte Sicherheitsfeature zu deaktivieren, die Angriffe nachweislich ermöglichen. Darüber hinaus kann gezeigt werden, wie die Sicherheitsmechanismen diese Angriffe bei fehlender Deaktivierung abwehren können.

1.1 Verwendung von Linux

Im Laufe des Projektes haben sich die Projektmitglieder auf das Betriebssystem Linux geeignet und Ubuntu als

2 Verwendung des Boards

Das STM32 Nucleo-144 nutzen wir als Board für Testen und Ausführen von Sicherungsfunktionalitäten. Die Nuclep Reihe dient dem Testen neuer Ideen und der schnellen Entwicklung von Prototypen.

2.1 Herstellung der Verbindung zwischen Board und Computer

Der erste Schritt der Installation war die physisch Beschaffung und Verbindung des Boards mit dem Computer. Dabei wird die Schnittstelle für den USB Micro B Port verwendet. Dabei ist zu beachten, dass ein Kabel zur Datenübertragung verwendet wird um zum Beispiel Software und Firmware auf das Gerät zu spielen.

Bild 1: USB Anschluss

2.2 Integrierten Entwicklungsumgebung - STM32CubeIDE

Die Wahl der integrierte Entwicklungsumgebung fiel auf das Programm STM32CubeIDE. Dieses Programm basiert auf Eclipse und bietet den nötigen Funktionsumfang für das Softwareprojekt.

Link: STM32CubeIDE

Die STM32CubeIDE bietet folgende Eigenschaften und Funktionalitäten:

- Fortgeschrittene C/C++ Entwicklungsplattform
- Peripherie ist konfigurierbar per grafischer Oberfläche (CubeMX)
  - Pinout, Clock, Peripherie und Middleware sind direkt konfigurierbar
- Quellcode lässt sich kompilieren und debuggen
- Spezialisiert auf STM32 Mikrocontroller und Mikroprozessor

Die Installation der Entwicklungsumgebung begann mit dem Download der Software über das Portal von STM und entpacken der Dateien.

2.2.1 Einrichtung der IDE und Verbindung zum Board

Für die Ausführung beziehungsweise Bash-Skripting werden folgende Befehle genutzt:

```
$ chmod +x /PATH/st-stm32cubeide_1.3.0_5720_20200220_1053_amd64.sh
$ ./st-stm32cubeide_1.3.0_5720_20200220_1053_amd64.sh
```

Zur Ausführung der STM32CubeIDE wurde eine ältere Javaversion benötigt. Vor dieser Einstellung konnte die Entwicklungsumgebung nicht gestartet werden.

```
$ sudo apt-mark hold openjfx libopenjfx-jni libopenjfx-java
$ sudo apt install openjfx=8u161-b12-1ubuntu2
$ libopenjfx-jni=8u161-b12-1ubuntu2
$ libopenjfx-java=8u161-b12-1ubuntu2
$ sudo apt purge openjfx
$ sudo apt install openjdk-8-jre-headless
$ sudo update-alternatives --config java
```

Um die Verbindung zwischen Board und Computer zu ermöglichen wurde ein weitere Applikation benötigt. Der sogenannte *STM32CubeProgrammer* ist ein Werkzeug um STM32-Produkte zu programmieren. Der Funktionsumfang beinhaltet vor allem die Möglichkeit den Speicher direkt zu konfigurieren.

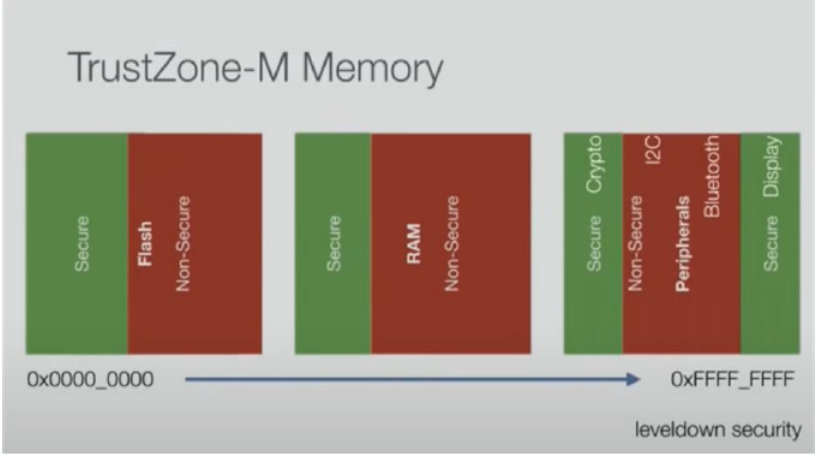
Die STM32CubeProgrammer bietet folgende Eigenschaften und Funktionalitäten:

3 TrustZone_M

3.1 Was ist Trustzone M?

Trustzone Cortex M ist eine kleinere Version von Trustzone Cortex A. Cortex A wird verwendet für Smartphone-Prozessoren und Cortex M wird für IoT-Geräte eingesetzt. Der Prozessor wird in einen sicheren und unsicheren Modus geteilt. Darüber hinaus kann Flash, RAM oder Peripherie in sicheren und unsicheren Teile partitioniert werden. Die Unterteilung erfolgt über den Einsatz von Adressen, die Speicherbereiche in *Secure* und *Non-Secure* unterteilen

Bild 2: Aufteilung der Speicher und Peripherie



Ein Beispiel für den Einsatz von dieser Art von Unterteilung findet sich beim Einsatz von BitCoin-Key-Wallets.
Die TrustedZone wird in Hardware-Wallets verwendet. Die Hardware-Wallets dienen als sichere, physische Speichermöglichkeit für *private Keys* des Nutzer. In diesen System befindet sich ein sicherer Ort oder Partition, auf die man von außen nicht zugreifen kann. Dieser abgeschlossene und getrennte Bereich wird als TrustedZone bezeichnet.
Auf Grund dieser Sicherheiten lassen sich beispielsweise alle kryptographische Schlüsseln in einen gesicherten Bereichen speichern. In diesem Zusammenhang werden auch Betriebssysteme unterteilt. Für bestimmte Anwendungen wird ein sogenanntes rich operating system (ROS) benötigt, welche aufwendigere Software ermöglicht. Bei der TrustZone befindet sich dann ein weiteres Betriebssystem, welches separiert wird. Diese Betriebssysteme werden vom selben Kern betrieben.

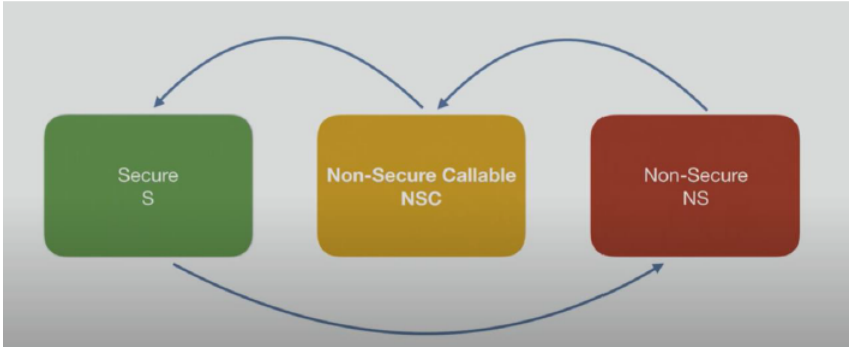
Bild 3: Hardware-Wallet und Unterteilung

3.2 Aufteilung in Secure, NonSecure & NonSecureCallable

Der Speicherbereich wird in drei Bereiche unterteilt. Dabei wird zu erst eine Unterteilung zwischen gesicherten Speicherbereich und ungesicherten Speicherbereich vorgenommen. Der gesicherte Speicherbereich wird nochmal in den Non-secure Callable- und Non-Secure-Bereich aufgetrennt. In der Regel sind wir nur in der Lage vom gesicherten Bereich auf den ungesicherten Bereich zuzugreifen, aber mit der Hilfe von Callable-Speicherbereichen lassen sich Ausnahmen definieren.

- Secure-Modus (S)
  - Gesicherter Bereich
    Kein externer Zugriff möglich und Funktionen können nur innerhalb dieses Bereiches aufgerufen werden. Ausnahme dazu sind die NonSecureCallable, welche einen externen Zugriff ermöglichen.
- NonSecureCallable-Modus (NSC)
  - Verbindung zwischen gesichertem und ungesichertem Bereich
    Funktionen, die aus dem Non-Secure aufgerufen werden sollen, müssen in diesem Bereich definiert werden.
- NonSecure-Modus (NS)
  - Freie Zugänglichkeit
    Kompletter Zugriff für jegliche Software innerhalb des eingebetteten Gerätes. Alle Funktionen lassen sich aus anderen Bereichen aufrufen.

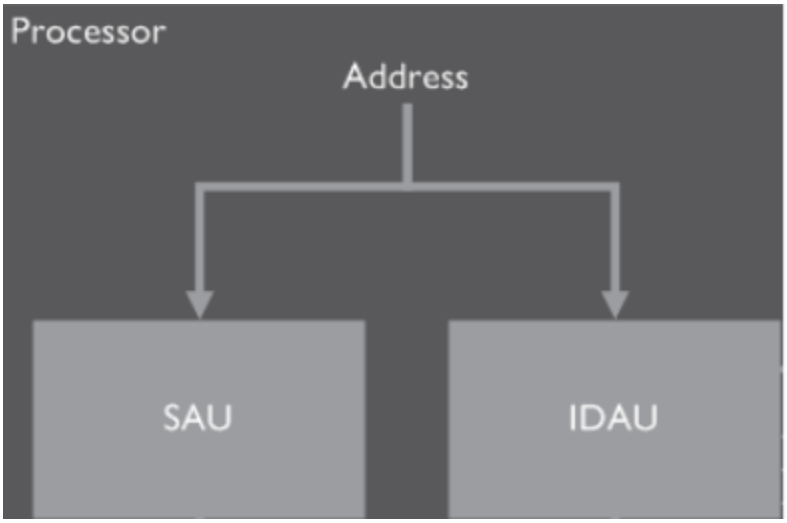
Bild 4: Aufrufkette - S, NS & NSC



3.3 Attribution Unit - Adressspeicheraufteilung

Mit Hilfe der Attribution Unit kann die Speicheraufteilung und Kommunikation zwischen den S-, NS- und NSC-Callable verwalten werden. Wenn die AU umgangen, deaktiviert oder außer Kraft gesetzt wird, kann das gesamte Gerät ungesichert werden.
Der Sicherheitsstatus eines Speicherbereichs wird definiert durch eine Kombination aus der internen Secure Attribution Unit (SAU) oder einer externen Implementation der Defined Attribution Unit (IDAU). Die SAU ist programmierbar und stellt dynamische Adresspartitionierung bereit, während die IDAU nicht programmierbar ist und durch statische Adresspartitionierung bereitstellt.
Wenn die SAU deaktiviert ist, keine SAU-Region definiert ist und keine IDAU im System enthalten ist, wird der gesamte Speicheradressraum als sicher definiert und der Prozessor hat keine Unterteilung mehr. Jeder Versuch, in den nicht sicheren Zustand zu wechseln, führt zu einem Fehler. Dies ist der Standardstatus des Prozessors ohne Nutzung der AU.

Bild 5: AU



Wenn SAU aktiviert und korrekt konfiguriert ist, sind bestimmte Bereiche im Speicher nicht sicher, da eine Aufteilung in gesichert und ungesichert vorgenommen wird.

3.3.1 Security Attribution Unit und Definition eines nicht sicheren Bereiches

Die Security Attribution Unit Control Register dient der Aktivierung der Security Attribution Unit. Das Register SAU_CTRL.ENABLE aktiviert die SAU bei entsprechender Belegung.
SAU_TYPE bestimmt die Nummer von Regionen, die von der Security Attribution Unit implementiert werden.
SAU_RNR (Region Number Register) selektiert die Regionen auf die SAU_RBAR und SAU_RLAR aktuell Zugriff haben. SAU_RBAR (Region Base Address Register) ermöglicht indirekte Schreib- und Leserechte für die "base address". Dieses Basisadresse dient als Referenz beziehungsweise Beginn für unseren ungesicherten Bereich. Die SAU_RLAR ermöglicht indirekte Schreib- und Leserechte für die "limit address". Damit dienen SAU_RBAR und SAU_RLAR als Begrenzungen für den nicht gesicherten Bereich. Dies Basisadresse dient als Startreferenz, während die Limitierungsadresse als Endpunkt fungiert.

Bild 6: Register der Security Attribution Unit

SAU_CTRL	SAU Control Register
SAU_TYPE	Number of supported regions
SAU_RNR	Region Number Register
SAU_RBAR	Region Base Address
SAU_RLAR	Region Limit Address

Ein Beispiel für die Adressaufteilung findet sich im folgenden Beispiel. In diesem Beispiel wird ein Adressraum als Non-secure definiert.

1.) Selektion einer der Region — SAU_RNR = 0x0

2.) Definition Startpunkt — SAU_PBAR = 0X1000

3.) Definition Endpunkt — SAU_RLAR = 0x1FFF

4.) Aktivierung SAU — SAU_CTRL = 0x1

Bei obigen Eingaben werden alle Adressen zwischen 0x1000 und 0x1FFF als Non-secure definiert. Bei der Auswahl der Regionen müssen noch weitere Regionen definiert werden, aber für die Übersichtlichkeit wird nur eine Regionsdefiniton im Beispiel dargestellt.
Im Softwareprojekt ist das in Partition_stm32l552xx.h definiert und kann für Testzwecke angepasst werden.

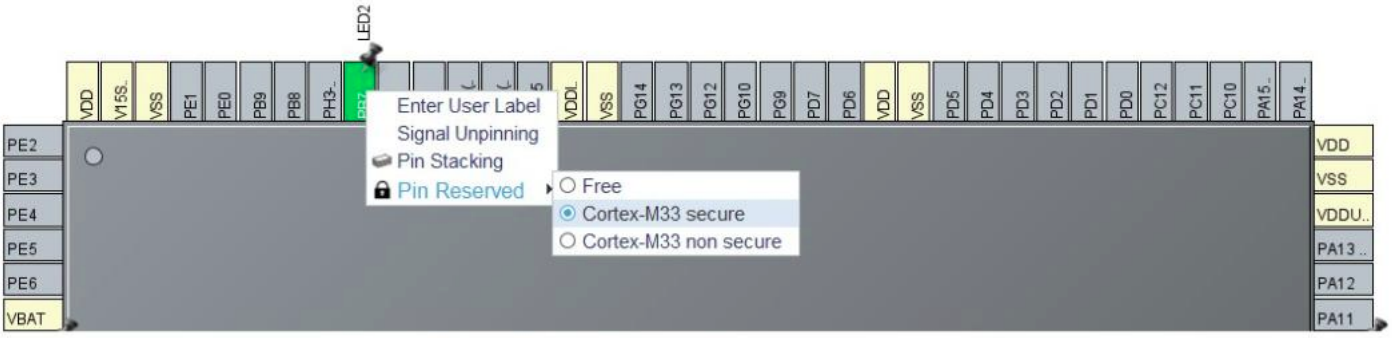
3.4 Beispiel Projekt:

Für das erste Beispiel wurde ein Toggeling durchgeführt, welches eine Verbindung zwischen Non-Secure-Bereich und Secure-Bereich herstellt. Auf dem Board befindet sich auf dem PIN PB7 eine LED und auf dem Pin PC13 eine Taste. Die Abfrage zur Betätigung der Taste wird im Non-Secure-Bereich durchgeführt, während aber das LED Toggling (ein- und ausschalten der LED) im Secure-Bereich aufgerufen wird. Dazu wird die Funktion in der Datei secure_nsc.c benötigt. Die Datei nsc.c ist eine NonSecureCallable, die also als Aufruf vom ungesicherten Bereich in den gesicherten Bereich dient. Die Funktionsdefinition befindet sich innerhalb secure_nsc.h.

3.4.1 Schrittweise Durchführung und Ablauf des ersten Projektes

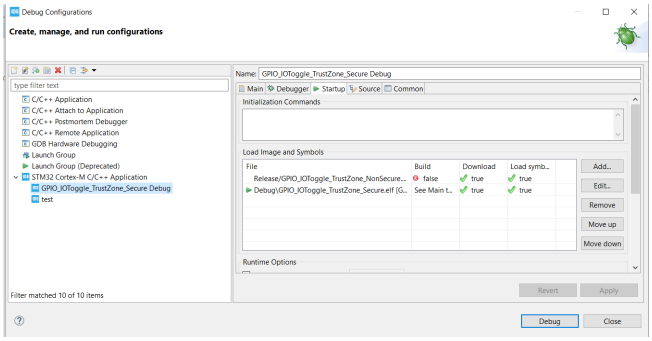
1. Neues Projekt STM32 erstellen
2. Das Board Nucleo-l552ze-q aus der Liste auswählen
3. Projektname eingeben und TrustZone aktivieren
4. Auf dem PB13-Pin per Linksklick GPIO_input auswählen. Darauf folgend per Rechtsklick für den Cortex M33 unsecure auswählen und den Pin auf "Taste" umbenennen.
5. Auf dem PB7-Pin per Linksklick GPIO_output auswählen. Anschließend per Rechtsklick Cortex M33 secure auswählen und den Pin auf "LED2" umbenennen.

Bild 7: Belegung eines Pins



6. Im Project Explorer die Eigenschaften über die Project references der Datei ändern, sodass zu erst die Non-Secure gebuildet wird, da Secure automatisch gebuildet wird, sobald Non-Secure gebuildet wurde.
7. Das "Perform BUILD" muss über die Debug Configuration bei TRUSTZONE deaktiviert werden, da das Builden bereits vorgenommen wurde.
8. Die Reihenfolge kann im Startup so angepasst, dass Non-Secure vor Secure eingereiht wird.

Bild 8: Ausführungsreihenfolge



Die Definition im Header im Secure-Bereich:
secure_nsc.h

```
1                    ...
2                     void my_toggle(void);
3                    ...
```

Source code 1: Definition Togglefunktion - Ausschnitt

Funktion: **secure_nsc.c**

```
1               CMSE_NS_ENTRY void my_toggle(void){
2               HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
3                    ...
```

Source code 2: Togglefunktion - Ausschnitt

Main-Funktion innerhalb des Secure-Bereiches:
main.c

```
1   ...
2   while (1){
3         /* USER CODE END WHILE */
4
5         /* USER CODE BEGIN 3 */
6         if (HAL_GPIO_ReadPin(TasteNS_GPIO_Port, TasteNS_Pin)==GPIO_PIN_SET){
7             my_toggle();
8         }
9         HAL_Delay(1000);
10  ...
```

Source code 3: Main mit endloser Schleife - Ausschnitt

Den Quellcode kann kompiliert und auf das Board geladen werden. Die neue Funktionalität ermöglicht bei jeder Betätigung der Taste eine blinkende LED.

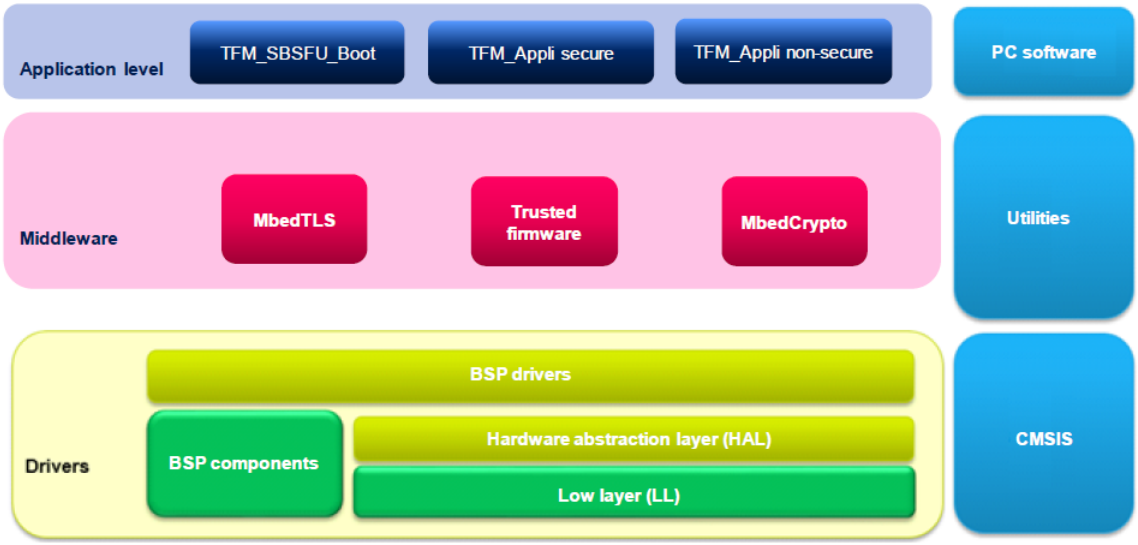
4 Trusted_Firmware_M

4.1 Einführung

Trusted Firmware bietet eine Referenzimplementierung von Secure-Software für Armv8-A und Armv8-M. Es bietet "Software-on-Chip" Entwicklern eine vertrauenswürdige Referenzcodebasis, die den relevanten Arm-Spezifikationen entspricht. PSA(Platform Security Architecture) bietet eine gemeinsame Sicherheitsgrundlage für das gesamte IoT-Ökosystem. Es enthält viele Elemente, einschließlich Architekturspezifikationen und Bedrohungsmodelle. Ein wichtiger Teil von PSA ist die Open Source-Firmware. Dies ist in Form von Trusted Firmware-M für Arm Cortex-M23- und Arm Cortex-M33-Prozessoren verfügbar, die die Arm TrustZone-Technologie verwenden.

4.2 Architektur

Bild 9: Die Architektur von TFM



4.2.1 Boad Support Package

Diese Schicht bietet die APIs in Bezug auf die Hardwarekomponenten in den Hardwarekarten

- Component
  - Dies ist der Treiber relativ zum externen Gerät auf der Karte und nicht zum STM32

- Driver
  - Es ermöglicht die Verknüpfung des Komponententreibers mit einer bestimmten Karte.(zB BSP_LED_Init())

4.2.2 Hardware abstraction layer (HAL) and low-layer (LL)

Die LL-Treiber bieten eine schnelle, leichtgewichtige Schicht, die näher an der Hardware liegt als die HAL. Die HAL-Treiber offer high-level function-oriented highly-portable APIs.

4.2.3 MbedTLS and mbed-crypto libraries

- MbedTLS
  - Kryptografische Dienste, die von der Anwendung TFM_SBSFU_Boot während der sicheren Startphase verwendet werden
- Mbed-crypto
  - Kryptografische Dienste, die von TFM Secure Application während Laufzeit verwendet werden.(enthält PSA APIs)

4.2.4 Trusted firmware middleware (TF-M)

Es enthält TFM Secure Boot, Die TF-M-Core Services bei der Laufzeit und Die TF-M Secure-Services bei der Laufzeit(basierend auf Mbed-Crypto).

4.2.5 TFM_SBSFU_Boot application

Diese Anwendung verwaltet die TF-M-Dienste für den sicheren Start und die sichere Firmware-Aktualisierung.

4.2.6 TFM_Appli secure application

Diese Anwendung verwaltet die Dienste, die im Secure-Modus stehen und vom nonsecure-Modus aufgerufen werden.

4.2.7 TFM_Appli non-secure application

Diese Anwendung ist ein Beispielcode für eine nicht sichere Benutzeranwendung, der die Verwendung sicherer TF-M-Dienste demonstriert.

References