Nerea Antonia López Martín

IDS PER UN SISTEMA ENCASTAT

TREBALL DE FI DE GRAU

dirigit per David Gamez Alari

Grau d'Enginyeria Informàtica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2021

Resum.

Els sistemes encastats són àmpliament utilitzats en àmbits que van des del món industrial fins l'ús quotidià, on aquest últim cas veu un creixement amb la internet de les coses. Donat el seu context resulten ser un objectiu per atacants amb pretensions de treure profit de la situació, destacant la extracció d'informació privada i l'atemptat contra aplicacions crítiques. Pel desenvolupament de nous mecanismes de defensa és important l'estudi continuat de les seves vulnerabilitats. Pràctiques com l'anàlisi forense resulten claus per aquest propòsit.

Amb objectiu de recol·lectar dades per aquesta finalitat, es proposa el disseny i desenvolupament d'un Sistema de Detecció d'Intrusions. En particular s'ha creat un Host IDS amb un paradigma de cerca de patrons per la placa NUCLEO-F767ZI. Dita implementació consisteix en la creació d'una llibreria en C++ per Mbed OS, un sistema operatiu de codi obert comú en entorns encastats. La llibreria inclou l'emmagatzematge d'informació dels atacs de manera eficient i la seva detecció amb multithreading. Per provar el seu funcionament s'ha creat un sensor d'atacs de denegació de servei per el port ethernet.

Aquest sistema s'ha avaluat mitjançant l'ús de jocs de proves individuals per cada funcionalitat i la simulació d'un atac de denegació de servei. Totes les proves definides han sigut completades satisfactòriament, assolint els objectius establerts dintre de les limitacions.

Resumen.

Los sistemas embebidos son ampliamente utilizados en ámbitos que van des de el mundo industrial hasta el uso cotidiano, dónde este último caso ve un crecimineto con auge del internet de las cosas. Dado su contexto tienden a ser un objectivo para atacantes con pretensiones de sacar provecho de la situación, destacando la extracción de información privada y el atentado contra aplicaciones críticas. Para desarrollar nuevos mecanísmos de defensa es importante el estudio continuado de sus vulnerabilidades. Prácticas como el análisis forense resultan clave para dicho propósito.

Con objectivo de recolectar datos para este fin, se propone el diseño y desarrollo de un Sistema de Detección de Intrusiones. En particular un Host IDS con un paradigma de búsqueda de patrones para la placa NUCLEO-F767ZI. Dicha implementación consiste en la creación de una libreria de C++ para Mbed OS, un sistema operativo de código abierto común en entornos embebidos. La librería incluye almacenamiento de información de ataques de forma eficiente y su detección mediante multithreading. Para probar su funcionamiento se ha creado un sensor de ataques de denegación de servicio para el puerto ethernet.

Este sistema se ha evaluado mediante el uso de juegos de pruebas individuales para cada funcionalidad y la simulación de un ataque de denegación de servicio. Todas las pruebas definidas han sido completadas satisfactoriamente, cumpliendo los objetivos establecidos dentro de las limitaciones.

Abstract.

Embeded Systems are widely used in fields that range from industrial to everyday tools, where the second case has seen an important growth with the internet of things. Given their context they are a target for attackers seeking to take advantage of the situation, especially private data extraction and critical application attacks. In order to develop new defense mechanisms, it is important to continuously study their vulnerabilities. Practices such as forensic analysis are key to this purpose.

For collecting data with this purpose, the design and development of an Intrusion Detection System is proposed. In particular, a Host IDS has been created using pattern matching for the NUCLEO-F767ZI board. This implementation consists in the creation of a C++ library for Mbed OS, a open source operating system common in embedded environments. The library includes an efficient attack data storage and detection by multithreading. To test its performance, a Denial-Of-Service attack sensor has been created for the Ethernet port.

This system has been evaluated through the use of individual test sets for each functionality and the simulation of a DoS attack. All the defined tests have been satisfactorily completed, meeting the established targets within the constraints.

Índex

Índex de taules

Índex de figures

1 Introducció

Els sistemes encastats són una opció comuna a l'hora de buscar una solució específica a un problema, usualment de temps real. Són àmpliament utilitzats en aplicacions industrials, automobilístiques, de telecomunicacions, mèdiques o d'electrodomèstics entre d'altres. A més amb el creixement de l'internet de les coses i l'addició de tecnologia per dur a terme tasques quotidianes, cada cop hi ha més dispositius que requereixen dissenys simples.

Per tant s'obre la necessitat d'establir mecanismes de protecció contra possibles atacs per diversitat de motius, d'una banda per tenir cura del bon funcionament de sistemes crítics com s'hi poden trobar al sector mèdic o d'automoció on es controlen paràmetres com el nivell d'oxigen a subministrar a un pacient o la velocitat del motor, i d'altra per protegir la privadesa de dades personals dels possibles usuaris d'aquests sistemes a la vida diària, d'on es poden extreure paràmetres com el ritme cardíac o el temps que passa la persona a la seva llar.

No hi ha una manera absoluta de protegir un sistema de usos malintencionats donat que la complexitat tendeix a ser elevada i hi ha la probabilitat que un atacant trobi noves maneres de traspassar les mesures de protecció, per tant l'estudi de nous possibles atacs es una prioritat. El camp enfocat en entendre incidències de seguretat quan ja han succeït s'anomena anàlisi forense, per recollir dades que permetin entendre els successos es fa ús d'eines de detecció i això ens porta a l'objectiu d'aquest projecte.

En aquest treball es proposa la creació d'un sistema de detecció d'intrusos per un sistema encastat. Això suposa dur a terme el disseny i desenvolupament de l'IDS per protegir una placa específica fent una anàlisis de riscos prèvia.

D'aquesta manera s'aporta una opció més al ventall per la seguretat d'aquest tipus de sistemes mentre que des del punt de vista formatiu suposa un primer contacte amb una àrea d'interès.

Cal remarcar que el sistema que es proposa s'encarrega de la detecció d'atacs i per una protecció complerta cal la implementació de mesures auxiliars, però això queda fora de l'abast d'aquest projecte.

1.1 Estructura del document

La organització d'aquest document consta dels apartats següents. Primer s'expliquen els conceptes bàsics que es necessiten per construir-se una idea de què i com s'ha realitzat al projecte. Seguidament ve l'apartat de disseny on es defineix quin tipus d'IDS es vol implementar, s'estudia el sistema a protegir entenent les seves característiques i fent una anàlisi de riscos des d'una perspectiva adversària per plantejar que podria detectar l'IDS i es defineix una estructura modular la posterior implementació. Després es descriu com s'ha portat a terme el disseny explicant les estructures i llibreries realitzades. A l'apartat d'avaluació es mostren els jocs de proves realitzats explicant i els resultats obtinguts. Finalment s'arriba a l'apartat de conclusions on es planteja la qüestió de si s'han assolit els objectius del projecte i es plantegen possibles millores o idees de cara al futur que no s'han fet per disposar d'un temps finit.

Aquest document també es fa amb la intenció de deixar latent el procés de creació d'aquest sistema.

2 Conceptes previs

2.1 IDS

Un Sistema de Detecció d'Intrusos es un dispositiu o aplicació per monitoritzar l'activitat d'una xarxa o dispositiu. S'encarrega de la detecció d'accessos no autoritzats i es composa principalment per un nucli que gestiona les amenaces i sensors que les detecten. La seva funció principal es avisar dels atacs i mantenir constància dels successos.

Pel que fa a la detecció hi ha diverses casuístiques si tot va bé l'IDS detectarà atacs quan hi hagi, però si no és així es poden donar dos caos, que detecti atacs quan no hi ha, el qual implica pèrdua de temps, o que no detecti atacs que estan succeint, aquest últim és el pitjor cas.

	Atac	No atac
Detecció	Cert Positiu	Fals Positiu
No detecció	Fals Negatiu	Cert Negatiu

Taula 1. Casos possibles de detecció d'atacs

En funció del mecanisme de detecció d'atacs es distingeix entre el següent:

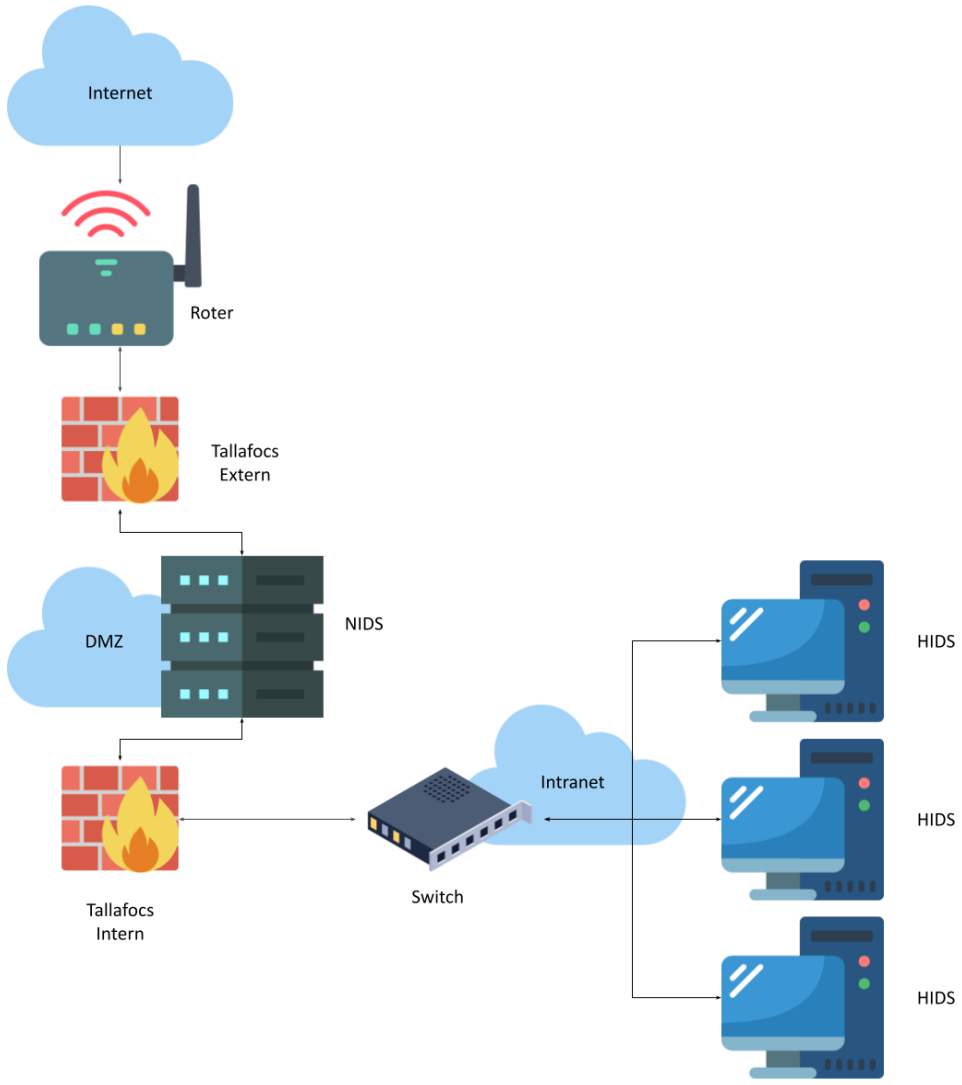
- Patrons/ Firmes: coneixent patrons de xarxa d'atacs coneguts es comparen amb la situació trobada i s'avalua si hi ha o no atac. Comporta que els atacs desconeguts no es detecten.
- Heurístics: Al contrari que en els sensors per patrons el que es coneix és el comportament estàndard del sistema i quan hi ha anomalies s'avisa a l'administrador.

En funció de la seva ubicació i d'on obtenen la informació:

- NIDS (Network IDS): sistema independent que analitza el tràfic de la xarxa, acostumen a estar a la perifèria de la xarxa o a la DMZ.
- HIDS (Host IDS): sistema instal·lat al host que es pretén protegir. Pot detectar crides al sistema, edicions a sistemes de fitxers i logs maliciosos, evita troians i events que un NIDS no pot.

També hi ha altres opcions com per exemple la de fer servir una màquina virtual remota.

A continuació es mostra una imatge exemplificant un HIDS i un NIDS, el NIDS s'ha situat a la DMZ, però podria situar-se dins la xarxa interna o a l'exterior.



Il·lustració 1. Exemple gràfic de HIDS i NIDS

Poden ser estructuralment centralitzats o descentralitzats i l'anàlisi el poden fer en temps real o no.

Un cop detectat l'atac, l'IDS pot emmagatzemar les dades i hi ha l'ampliació de bloquejar l'activitat que s'ha detectat o de respondre per evitar-la.

3 Disseny

Prenent com a base els apartats previs on hem vist quins són els objectius d'aquest projecte i s'han dit els conceptes necessaris per entendre'l, en aquesta secció es procedirà a fer una explicació més detallada i concisa de què s'ha realitzat, com s'ha fet i de quines decisions s'han pres.

3.1 Concepte general

Els sistemes encastats i de temps real poden portar a terme funcions crítiques així que és interessant tenir l'opció de controlar amb el detall més gran possible qualsevol mena d'amenaça.

S'ha optat per fer un HIDS (Host Based Intrusion Detection System). Aquesta decisió es deu al fet que amb aquesta arquitectura l'IDS podrà recopilar dades més específiques tenint un major control del que passa a la placa.

Les funcions que realitzarà aquest sistema seran de detecció d'amenaces, classificació, avaluació, report i emmagatzemament d'informació d'una manera que faciliti la posterior anàlisi per millorar en cadascuna de les parts del procés.

Per tal de detectar les amenaces es disposarà de sensors software enfocats en cada tipus d'atac que es pugui fer. Aquest mecanisme de detecció d'atacs es coneix com a detecció per patrons o firmes, buscant els patrons específics de cada tipus d'atac. Una altra opció seria la d'utilitzar una heurística de com funciona el sistema generalment, però està fora de l'abast d'aquest projecte. El problema que té l'opció escollida és que només es detecten els atacs que es coneixen, si hi ha un atac desconegut, el sistema no el tindrà en compte, però l'avantatge que ofereix és el de donar una informació detallada del tipus d'atac que té lloc, amb un sistema heurístic podries saber que el comportament és diferent del normal però no per què.

Un cop es detecta l'amenaça s'emmagatzema a memòria per facilitar la posterior anàlisi forense.

Per tant, hi distingim tres apartats generals a desenvolupar de l'IDS:

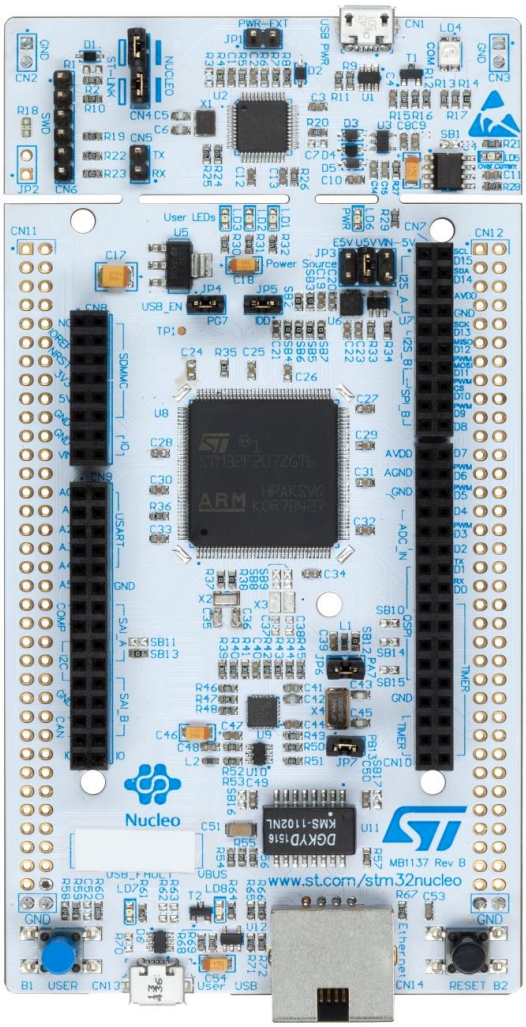
- Memòria de logs: És una zona de memòria on s'emmagatzema la informació que els sensors detecten en forma de logs.
- Manager: És el codi que s'encarrega gestionar les operacions de l'IDS, des dels guardats i consultes a memòria fins a la creació de sensors.
- Sensors: Són els elements software que s'encarreguen de detectar els atacs.

3.2 Entorn cobert per l'IDS

Abans d'entrar en matèria de com és la realització de l'IDS cal entendre l'entorn que ha de defensar. Aquest apartat pretén donar una idea de com és la placa, quines vulnerabilitats té i finalment explicar quines detectarà l'IDS.

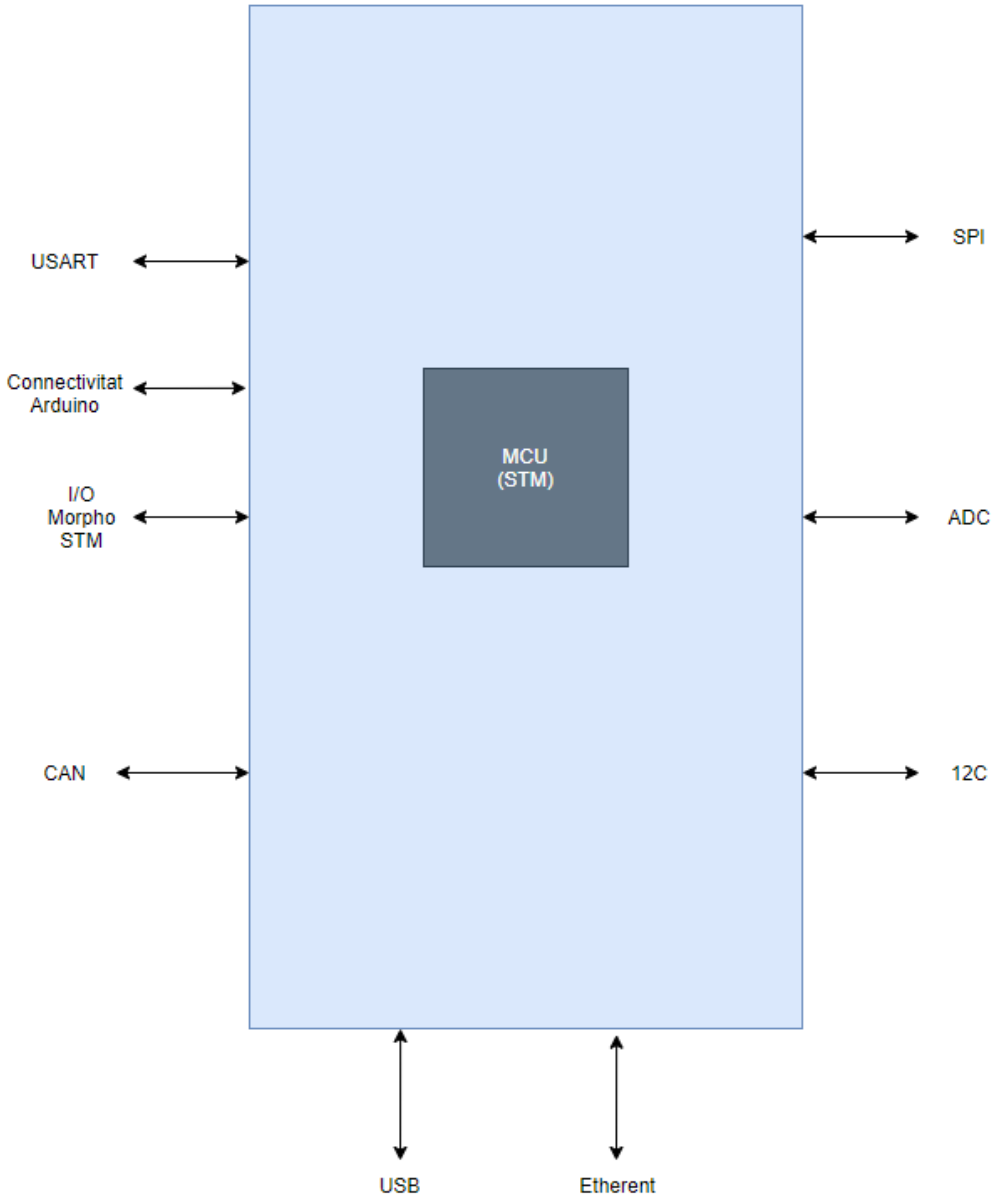
3.2.1 Placa NUCLEO-F767ZI

La placa en qüestió és la NUCLEO-F767ZI, conté un microcontrolador STM32F767ZIT6. A continuació es mostra una imatge del model.



Il·lustració 2. Imatge de placa NUCLEO-F767ZI

S'ha fet un diagrama de blocs de la placa per facilitar el coneixement dels components que la conformen i les interfícies que té.



Il·lustració 3. Diagrama d'entrades i sortides de la placa

En aquest diagrama es pot veure que tenim un microcontrolador d'arquitectura ARM STM32F767ZIT6 amb les següents característiques:

- CPU: ARM32-bit Cortex-M7 + DPFPU + Chrom-ART Accelerator
  - Freqüència màxima: 216 MHz
  - Voltatge: 1,7 V - 3,6 V
  - Random Number Generator (TRNG for HW entropy)
- Memòria Flash: 2 MB
- Memòria SRAM: 512 KB

El microcontrolador té compatibilitat amb més interfícies, però com que la placa no en té, queden excloses.

Hi trobem les interfícies següents:

Interfícies	Descripció
USART/UART	Universal asynchronous receiver-transmitter, mitjà amb què el format de dades i la velocitat de transmissió són configurables.
SPI	Estàndard de comunicació utilitzat per transmetre informació entre circuits integrats.
12C	Bus de comunicació serial síncron multimaster, multi slave.
ADC	Analog to Digital Converters, pins de senyal analògics.
CAN	Controller Area Network, bus usat a l'automoció per comunicar unitats electròniques minimitzant el nombre de cables.
Ethernet	Estàndard de xarxes d'àrea local. En aquest cas de 10/100Mbps.
USB	Bus estàndard que permet connectar, comunicar i alimentar elèctricament dispositius.
Arduino UNO Revisió 3	Conjunt de pins d'entrada sortida analògics i digitals.
Morpho STM	Pins d'accés al processador.

Taula 2. Descripció d'interfícies de la placa

Les interfícies vistes són externes, tot i que en molts casos donen accés directe al processador. Les internes no s'han tingut en compte.

3.2.2 Possibles atacs

En aquest apartat es vol comprendre les diferents maneres que hi ha per atacar la placa basant-se en les especificacions vistes prèviament.

L'escenari que es planteja és de black box, donat que no hi ha informació pública del funcionament de la placa a nivell intern. Per tant, no es tindrà en compte cap vulnerabilitat relacionada amb bugs i errors que pugui tenir.

En termes generals, per atacar la placa es pot comprometre la confidencialitat, la integritat, l'autenticitat i la disponibilitat de cadascun dels components que la formen. Les vies per fer-ho són les interfícies d'entrada i sortida.

A la taula següent es mostren els atacs principals que es poden fer a la placa amb una descripció i quina propietat queda compromesa.

Atac	Descripció	Propietat compromesa
Lectura de memòria	Amb aquest atac s'obté un accés indegut a la lectura d'algunes zones de memòria.	Confidencialitat
Sniffing	Un atacant intercepta la comunicació i té accés al seu contingut.	Confidencialitat
Spoofing	Un atacant falseja la seva identitat i intercepta la comunicació de la placa amb altre element, tenint control sobre aquesta sense que es noti.	Confidencialitat, Integritat, Autenticitat
Escriptura de memòria	Atac en què s'escriu en una zona de memòria indegudament.	Integritat
Buidatge de memòria	L'atacant buida el contingut de la memòria, això pot implicar pèrdua de codi o dades importants.	Integritat
Injecció de codi	L'atacant sobreescriu la memòria afegint el seu codi maliciós. Pot ser un virus, cuc, troià, una bomba lògica, un rootkit, etc. i les finalitats són diverses pot voler espiar, prendre el control de la placa, fer-la servir per atacar altres elements que estiguin connectats amb aquesta, etc.	Integritat
Buffer overflow	Si hi ha massa dades acceptades com a input d'un procés es pot fer servir per executar codi maliciós.	No aplica
Denegació de servei	L'atacant pretén imposibilitar als usuaris l'accés a un element.	Disponibilitat

Taula 3. Possibles atacs

La informació es presenta com a coneixement general de què es pot fer però sense concretar com. Hi ha diverses maneres de realitzar cadascun dels atacs proposats. Per cada tipus d'interfície hi ha uns protocols de comunicació i en funció de cadascun d'aquests es podran dur a terme els atacs d'una manera o altra.

3.2.3 Scope

Hi ha moltes possibilitats diferents per efectuar atacs, tant per la varietat d'objectius que pot tenir l'atacant com per les diferents interfícies i protocols que pot fer servir, en aquest treball s'ha escollit detectar atacs d'Ethernet. Concretament, detectar una denegació de servei.

3.3 Mòduls de l'IDS

L'IDS es composa d'un mòdul de memòria on s'emmagatzemen els logs, d'un programa principal o manager que gestiona la creació dels sensors i dels sensors en si mateixos els quals detecten les intrusions.

3.3.1 Memòria de logs

L'objectiu d'aquesta memòria és facilitar l'anàlisi forense posterior a la detecció d'atacs, per tal de donar tota la informació possible dels successos que han tingut lloc i fer-ho de manera estructurada, facilitant la investigació.

Per tant, es requereix l'ús de logs. Un log és la unitat mínima d'informació del sistema, on es recullen dades d'un atac específic, la memòria conté estructures de dades per emmagatzemar aquests logs.

S'han tingut en compte els següents requeriments per dissenyar la memòria:

- Ha de ser estructurada per tal de permetre i afavorir l'anàlisi de la informació.
- Ha de tenir integritat, ja que hi ha la possibilitat que l'atacant la modifiqui per encobrir l'atac. No cal que sigui confidencial donat que no es tracta d'informació personal.

Per tant, es reservarà una zona estàtica de la memòria Flash de la placa. A continuació es mostra el disseny de les estructures de dades utilitzades per emmagatzemar els logs i la metodologia emprada per assegurar integritat.

3.3.1.1 Estructura de dades

Independentment del tipus d'atac els logs tenen data i tipus, pel que fa als camps que són específics d'un tipus d'atac es fa servir un string amb separadors.

- Data i hora
- Tipus d'atac
- Atributs extres

És interessant tenir constància del moment en què s'ha produït l'atac i quin ha estat, a més això permet veure si s'han produït atacs que estiguin relacionats, la qual cosa pot oferir informació de les intencions de l'atacant. Per cada tipus d'atac diferent s'explicarà posteriorment quins camps es guarden com atributs extres.

Per emmagatzemar els logs es volia tenir una estructura de dades ordenada que permetis conèixer l'ordre dels successos. Interessava que les dades fossin tan variades com fos possible per fer una anàlisi completa. A més es volia que aquesta estructura anés situada a una zona de memòria específica, per tant, havia de ser memòria contigua i que permetés fer les següents operacions:

- Afegir
- Consultar
- Eliminar

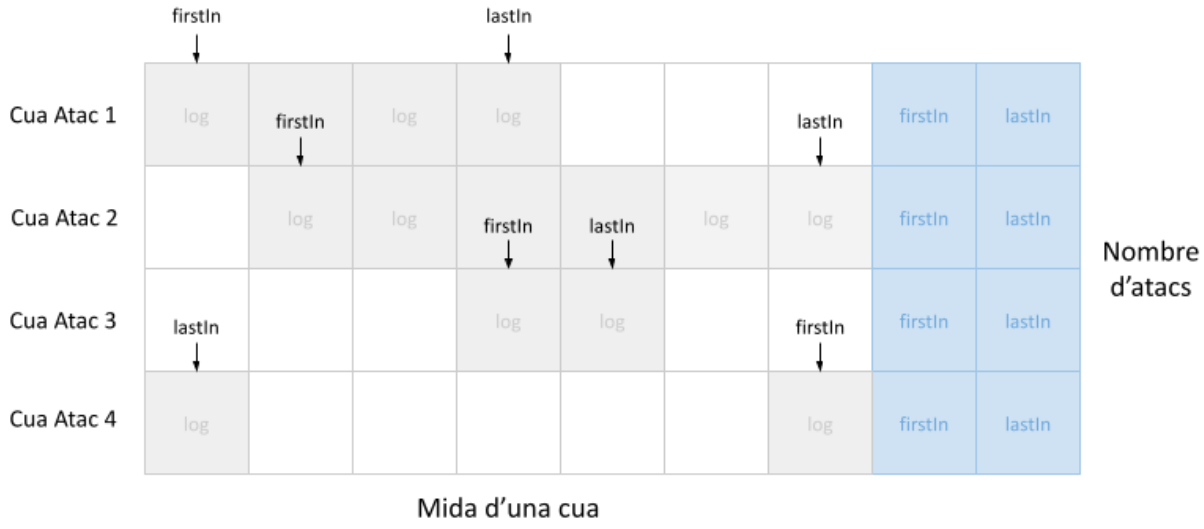
Per aconseguir el que es proposa es disposa d'una cua circular per a cada tipus d'atac.

L'ús de l'estructura de cua permet tenir les dades endreçades per ordre d'arribada i així quan cal esborrar un element s'agafa el més antic.

A més en ser circular s'estalvia cost d'eliminació, ja que no cal reordenar les dades cada cop que s'esborra un element, sinó que es disposa de punters indicant l'inici i el fi de la cua, els quals varien quan es produeix un moviment.

El fet de tenir-ne una per cada tipus d'atac impedeix la pèrdua d'informació dels atacs menys usuals, això es deu al fet que quan hi ha poc espai i s'esborren elements antics per afegir-ne de nous els elements eliminats són de la cua del tipus que s'afegeix. Si només hi hagués una cua, caldria tenir variables per controlar que hi hagués un mínim de logs per cada atac i quan s'hagués d'esborrar caldria fer una reordenació dels elements de la cua, perdent així el benefici de què sigui circular. A més l'operació d'esborrar pot ser cridada per la d'afegir si no hi ha espai a la cua i, per tant, pot ser bastant comuna, el cost temporal de la reordenació no interessava. Com es vol treballar amb memòria contigua es fa servir una array de cues.

L'estructura de dades quedaria com es mostra a continuació.



Il·lustració 4. Memòria de logs

3.3.1.2 Seguretat

La memòria de logs és un objectiu clau pels atacants, donat que si la modifiquen poden falsejar l'historial de logs i, per tant, encobrir altres atacs que hagin fet.

Els principals objectius de protecció de la memòria són la integritat i l'autenticitat. Es vol tenir coneixement de si la memòria ha estat modificada indegudament, la solució proposada és una signatura digital.

La proposta consisteix a signar cada cua de logs de memòria que es vol protegir. Cada cop que es fa una escriptura per l'afegit o esborrat d'un log s'actualitza la firma, la qual va en un vector de firmes. Quan es vol fer una operació de lectura es calcula la signatura i s'avalua si coincideix o no amb la que hi ha concatenada a la memòria, en cas afirmatiu la memòria no ha estat modificada i en cas contrari sí, s'esborrarà perquè la informació ja no és fiable. Tenir una signatura per cada cua en comptes d'una per tot el bloc de memòria permet no perdre la informació de les cues que no estiguin compromeses.

Per fer la signatura es farà servir l'AES 128.CMAC (OpenSSL/TLS). Aquest algorisme fa servir una clau simètrica, això vol dir que si algú compromet la clau podria signar la memòria i passar inadvertit. S'ha de desenvolupar un mètode per protegir-la, això es veurà en l'apartat d'implementació.

3.3.2 Manager

El manager és el fil principal de l'IDS. S'encarrega de la creació de sensors i l'emmagatzematge en memòria.

Tal com s'ha explicat prèviament aquest IDS es troba dins el propi host al qual vol protegir. Per tant, el manager no seria el procés principal de la placa, sinó un dimoni (procés de background) que s'engegaria en iniciar la placa i s'atura en apagar-la.

El codi consistiria en el següent:

1. La creació d'un fil per cada sensor, els quals executen un bucle infinit on hi ha el codi per detectar els diferents patrons.
2. Quan es vol apagar el sistema acabar amb els fils.

Cada fil en termes generals consistirà en un bucle on es farà el següent:

1. Avaluar si hi ha anomalies, el criteri dependrà de l'atac específic. En cas afirmatiu:
    a. Crear log amb dades de l'atac
    b. Guardar log a l'estructura de memòria dins la Flash
    c. Renovar la signatura de la memòria

Com que la memòria és un recurs compartit entre diversos fils d'execució, cal implementar mecanismes per assegurar una lectura i escriptura ordenada.

Tot i no estar inclòs en els objectius d'aquest treball interessa que la seva estructura permeti que el programa principal efectuarà lectures quan l'administrador faci consultes dels logs, la lectura hauria de obtenir una informació actualitzada. D'altra banda, els fils faran escriptures introduint els logs dels atacs detectats. Com cada fil detecta un tipus d'atac diferent no hi haurà accessos a la mateixa zona de memòria, però quan un fil detecta un atac les accions que farà s'han d'executar de forma atòmica.

L'emmagatzematge a memòria i la signatura s'haurien de fer dins la mateixa secció crítica, en cas contrari podria donar-se el cas que no hi hagués la signatura de l'atac actualitzada i s'esborrés el contingut de la cua de l'atac, perdent així informació que és vàlida. Per crear la secció crítica es fa ús de semàfors i la implementació s'explicarà posteriorment.

3.3.3 Sensors d'atacs

En aquest apartat es veu quins atacs es detectaran per l'IDS. Per entendre com detectar-los es veurà com es fa l'atac i com s'emmagatzema la informació de l'atac als logs.

En termes generals cada sensor tindrà una estructura similar, executarà un bucle infinit avaluant si hi ha anomalies i en cas afirmatiu crearà un log amb la informació pertinent, l'emmagatzemarà a memòria i actualitzarà la firma de la memòria.

3.3.3.1 Denegació de Servei per Ethernet

3.3.3.1.1 Descripció de l'atac

Per començar cal entendre què és una denegació de servei i com es pot fer.

Una Denegació de Servei és un tipus d'atac que consisteix a impossibilitar l'ús d'un recurs als usuaris d'aquest, això es pot aconseguir de diverses maneres depenent del cas específic, però en termes generals es pot distingir entre els atacs següents:

- Aquells que consisteixen a fer col·lapsar el sistema basant-se a inundar-lo de trames aleatòries. Es pot fer des d'un host o de manera distribuïda, en el segon cas és més difícil de detectar.
- Aprofitar vulnerabilitats del sistema per incapacitar-lo.

Com que estem treballant sobre una black box l'atac que s'ha fet és del primer tipus, es busca inundar la xarxa amb trames Ethernet enviades des d'un host.

L'atac consisteix en un bucle infinit en el qual s'envien trames a la placa.

3.3.3.1.2 Informació d'un log

Cada log contindrà el següent:

Log de DoS amb Ethernet
Data i hora
Tipus d'atac: DoS
Nombre de missatges rebuts.

Taula 4. Atributs de log

A banda dels camps comuns en aquest atac s'afegeix el nombre de missatges rebuts en un període definit de temps.

3.3.3.1.3 Detecció de l'atac

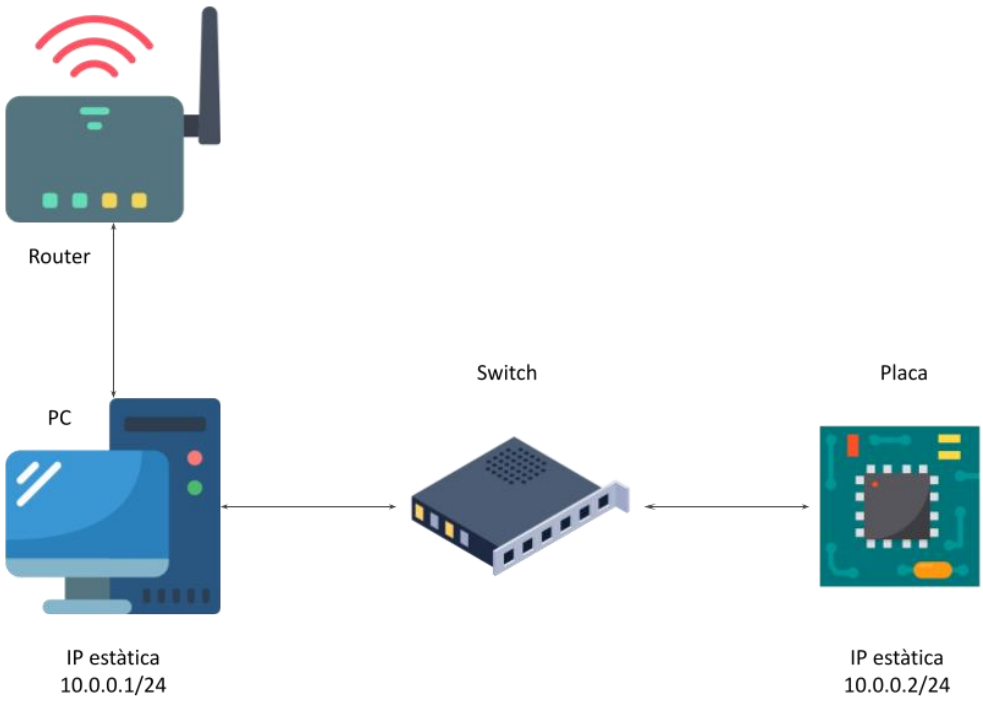
El que fa el sensor és el següent:

- Avaluar la quantitat de missatges rebuts en un període de temps.
- Si hi ha més logs que la quantitat establerta com a màxim es crea el log i en una secció crítica es fa el següent:
    - Afegir a memòria el log de l'atac
    - Signar la memòria

3.4 Entorn de desenvolupament

En aquesta subsecció es definirà l'entorn emprat per desenvolupar i provar el funcionament de l'IDS, focalitzant-se en l'arquitectura de xarxa i explicant el programari utilitzat.

Respecte a l'arquitectura de xarxa, la placa obté accés a internet mitjançant un commutador i un ordinador que està connectat a un router. L'arquitectura de xarxa és la següent, hi ha dues xarxes, una és la 192.168.1.0/24 (tot i que aquesta adreça IP depèn del router) i l'altra és la 10.0.0.0/24. En aquesta segona s'han assignat les adreces IP de manera estàtica tal com es mostren a la imatge següent:



Il·lustració 5. Arquitectura de xarxa

Per fer tot l'IDS es fa servir Mbed Studio, un IDE que permet treballar amb Mbed OS, un sistema de codi obert compatible amb dispositius Cortex-M. Dels sistemes que es plantejaven aquest és del que hi ha més informació i més comunitat, facilitant així el desenvolupament. A més té els avantatges de ser un sistema modular que inclou llibreries de seguretat TLS/SSL.

Per dur a terme l'anàlisi de la xarxa es fa servir Wireshark, és un software que permet analitzar el tràfic en temps real i serveix per veure com es realitzen els atacs i comparar les trames observades amb les deteccions de l'IDS.

La implementació del codi es fa amb el llenguatge de programació C++ i per escriure els scripts dels atacs es fa servir Python.

Tot i fer servir C++, com que estem en el sistema Mbed OS, no es pot fer ús d'excepcions donat que afegeixen un overhead considerat excessiu per la comunitat que el fa servir, tampoc es pot fer ús de classes genèriques.

4 Implementació

4.1 Entorn de desenvolupament

A l'apartat de disseny s'ha vist l'arquitectura de xarxa i la tecnologia utilitzada pel desenvolupament d'aquest treball, a continuació es fa una explicació de com s'ha configurat.

Per dur a terme aquesta tasca s'ha fet ús un conjunt de scripts. D'una banda, per instal·lar l'IDE de Mbed Studio s'ha agafat l'script ofert a la seva pàgina web. Després per fer la configuració de la xarxa s'ha creat el conjunt de scripts següent:

- setup.sh
- network_setup.sh
- nat.sh

Es pretén connectar la placa amb l'ordinador i donar accés a internet per imitar una situació que podria donar-se en un cas d'una aplicació real. L'accés a internet s'obté fent que l'ordinador faci forward del tràfic de la placa i el router.

4.1.1 Script general: setup.sh

Aquest script substitueix el fitxer de configuració de xarxa d'Ubuntu situat al /etc/netplan per un fitxer idèntic on queda afegida la configuració de la xarxa 10.0.0.0/24.

Després per activar el servei de configuració de la xarxa, còpia els fitxers network_setup.sh i network_setup.service als paths /usr/bin/ i /etc/systemd/system/ respectivament, i engega el servei en qüestió.

Activa el forwarding i crida l'script nat.sh on s'introdueix la configuració de les Iptables necessària per permetre que la placa pugui tenir accés a internet.

4.1.2 Servei de configuració de xarxa

Aquest servei es fa necessari donat que, tot i que les IPs de la nostra xarxa siguin estàtiques, el gateway és una IP dinàmica assignada pel DHCP del router a l'ordinador i, per tant, s'ha de modificar, aquest canvi és el que fa el servei.

S'executa al boot de l'ordinador, després d'haver-se engegat els serveis de networking per tal de consultar la IP actual.

L'script substitueix el gateway del fitxer de configuració per aquesta nova IP. Si quan s'engega l'ordinador la placa no està connectada o el switch està apagat, quan el servei s'executi no podrà ficar cap IP al gateway encara que es connecti tot a posteriori, per reconfigurar la xarxa correctament s'hauria de tornar a executar l'script de setup.sh.

4.1.3 Connectivitat a internet

Per facilitar la connectivitat de la placa s'activa el bit de forwarding i s'afegeixen regles d'Iptables amb l'script nat.sh.

Com que la placa té una IP de la xarxa 10.0.0.0/24 el router no la reconeix i per accedir a internet es fa nat traduint aquesta direcció per la de l'ordinador a la xarxa del router.

Per veure que l'entorn funciona i hi ha connectivitat s'han fet pings i s'ha fet un programa anomenat connectivity_test on s'assigna la IP 10.0.0.2 a la placa, s'obre un socket TCP i es crea una connexió a mbed.org.

4.2 IDS

4.2.1 Memòria de logs

El desenvolupament de la memòria de logs s'ha organitzat dividint-se en tres iteracions. La primera consisteix en fer les estructures de dades per emmagatzemar logs a memòria RAM, la segona en calcular una signatura per protegir contra integritat quan s'afegeixen logs i la tercera en fer que això es guardi a memòria Flash per tal que no es perdi la informació entre encesos i apagats.

4.2.1.1 Estructures de dades

Tal com s'ha vist al disseny l'estructura de memòria consisteix en un array que conté cues circulars de logs, una cua diferent per cada tipus d'atac. Això es deu a que interessa que la memòria sigui estàtica donat que es vol signar i en una futura iteració afegir a la Flash, a més de ser un disseny més minimalista i simple la qual cosa dificulta deixar portes de darrere.

Per tal de crear dita estructura es fa ús de la classe LogMem, la qual té la funció de inicialitzar la memòria i aportar operacions d'afegir i consultar logs. Aquesta classe fa la reserva de memòria de l'array de cues.

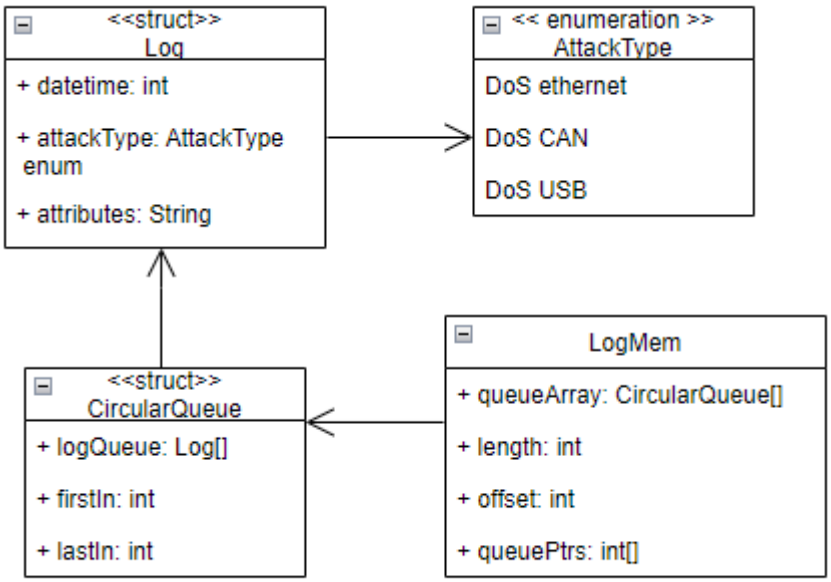
La gestió de les cues es duu a terme a la llibreria circularQueue.h on s'hi troba l'struct del tipus cua circular i les funcions necessàries per l'ús de la cua.

Finalment s'ha creat la llibreria log.h, on es defineix l'enumeració de tipus d'atacs, l'struct del tipus log i les funcions per tractar logs en un sentit més bàsic.

S'ha escollit fer llibreries amb structs i funcions en comptes de classes per poder tenir un control total a l'hora de gestionar la memòria dins la classe LogMem. Altrament hi ha errors en la reserva fruit de variacions en l'espai que ocupa una classe. Per aquest mateix motiu a l'hora d'implementar els logs no s'ha fet servir herència, la qual era una idea inicial ja que es podria haver creat la classe pare log i les classes filles amb el tipus de log específic per cada atac. De fet sense haver de recórrer a l'ús de classes es podria haver adaptat als structs, creant-ne un de diferent per cada log específic que contingues el tipus log i els atributs a afegir, però això té la implicació que cada cua tingués un offset i una mida diferent complicant el tractament de les dades i per això s'ha optat per fer servir un string.

Per motius similars s'ha implementat la pròpia classe cua, donat que tot i haver classes públiques de cues, no n'hi havia cap de circular, la qual cosa suposaria un cost de reordenació dels elements a cada crida a la operació de desencuar. A més no es va trobar cap classe cua genèrica i encara que n'hi hagués per c++, no es podria fer servir en Mbed OS donades les limitacions expressades en l'apartat de entorn de desenvolupament.

Les estructures emprades es mostren en al diagrama de classes següent.



Il·lustració 6. Diagrama de classes de memòria

A continuació s'explicarà el contingut de cada llibreria.

4.2.1.2 Llibreria de logs

Per començar tenim l'struct Log, aquest conté tota la informació necessària per conèixer les circumstàncies de l'atac. Consta del temps, del tipus d'atac i d'un string amb els atributs extra donat que cada atac pot tenir dades diferents a guardar.

L' emmagatzematge del temps s'ha dut a terme amb la llibreria de C ctime, el tipus utilitzat es el de time_t que guarda el nombre de segons que han transcorregut des de la primera epoch de unix, això permet fer operacions i comparacions fàcilment. En aquesta iteració es fa servir el temps en segons des que s'engega la placa, donat que les primeres probes es fan en RAM i no cal tenir en compte el temps real per operar o fer comparacions.

El tipus d'atac es defineix amb un enumerador. Això aporta llegibilitat al codi i a la vegada el tipus d'atac es pot tractar com un nombre enter, facilitant el seu ús per indexar la cua específica dins l'array que es fa servir per donar estructura a la memòria. Els atacs que hi ha al diagrama són d'exemple.

El string d'atributs dona flexibilitat a l'hora de tenir-ne tants com es vulgui, la mida de l'string va definida per la mida que necessiti el log amb més atributs o amb atributs més llargs, aquests es distingeixen amb un separador.

És fa ús de constants per control·lar el nombre d'atacs que detecta l'IDS i la mida de l'string d'atributs.

Tenim les funcions següents, com que es tracta d'una llibreria i no d'una classe les funcions reben per paràmetre el log que es vol tractar.

- createLog:
  - Entrada: Tipus d'atac i punter indicant on es vol guardar el log creat.
  - Sortida: cap.
  - Descripció: aquesta funció inicialitza les dades de l'struct del log rebut per paràmetre. Primer calcula el temps actual amb la funció time(0) de c, es el primer que es fa donat que es vol tenir l'instant més concís que es pugui de quan s'ha donat a terme l'atac i les assignacions d'altres atributs son cicles de rellotges que es perden. Això no te perquè ser rellevant ja que en aquest cas la memòria es compartida, però si es tractés d'un IDS amb paral·lelisme, seria interesant veure exactament en quin ordre succeeixen els atacs i a on. Després s'assigna el tipus d'atac i s'inicialitza el string d'atributs amb els valors pertinents a cada cas.

- copyLog:
  - Entrada: log que es vol copiar i punter a log on es vol guardar la còpia.
  - Sortida: cap.
  - Descripció: s'assignen els camps del log còpia amb els del log a copiar, pels atributs extra es fa servir un strcpy de la llibreria de strings de c.

- toStringLog:
  - Entrada: log a imprimir.
  - Sortida: string amb les dades del log en el format "Nom del camp: " + camp, per cada atribut del log.
  - Descripció: es creen strings amb el contingut dels camps d'un log. Per el temps es fa us de la funció localtime de la llibreria ctime, aquesta retorna un struct anomenat tm on cada camp representa una unitat de temps (segons, minuts, hores, dies, etc), després es fa ús de la funció strftime per passar el struct a string i així s'obté una lectura còmoda del temps. Per el tipus d'atac es crida a la funció toStringAttackType i els atributs ja són un string.

- toStringAttackType:
  - Entrada: objecte del tipus AttackType.
  - Sortida: string amb el tipus d'atac.
  - Descripció: consisteix en un switch on per cada tipus d'atac, en tractar-se d'un enum consisteix en un int, s'assigna el seu string pertinent amb el nom de l'atac.

4.2.1.3 Llibreria per tractar amb cues circulars

En aquesta llibreria tenim el struct amb les dades necessàries per treballar amb una cua i les funcions per operar amb aquesta. La mida màxima de la cua va definida per una constant.

El struct consta de una array de logs amb la mida màxima definida i de dos punters, aquests dos punters són dos nombres enters que contenen la posició del primer element afegit a la cua i de l'últim.

És disposa de les funcions següents, com que es tracta d'una llibreria i no d'una classe les funcions reben per paràmetre la cua que s'està tractant.

- initQueue:
  - Entrada: punter a cua circular.
  - Sortida: res.
  - Descripció: Els dos punters lastIn i firstIn s'inicialitzen a -1 indicant que la cua és buida.

- enqueue:
  - Entrada: punter a cua circular i log a afegir.
  - Sortida: booleà indicant si s'ha esborrat un element de la cua.
  - Descripció: es comprova si la cua és plena, en aquest cas es crida a la operació desencuar, després si es buida el punter firstIn s'inicialitza a 0, s'incrementa lastIn amb el mòdul (per ser circular) i es fa ús de la funció copyLog per guardar el log passat per paràmetre a la cua dins la posició lastIn.

- dequeue:
  - Entrada: punter a cua circular i punter a log on es vol guardar el log que es tregui de la cua.
  - Sortida: booleà indicant si s'ha desencuat algun element.
  - Descripció: aquest mètode en cas que la cua no sigui buida, còpia el log apuntat per firstIn al punter passat per paràmetre i incrementa firstIn en una posició. En cas que sigui l'últim log fica els dos punters a -1 indicant que la cua és buida.

- getQueueLength:
  - Entrada: cua circular.
  - Sortida: enter indicant el nombre de posicions ocupades de la cua.
  - Descripció: si la cua es buida retorna 0. Si el valor lastIn es >= que el firstIn (ordre típic d'una cua), es retorna lastIn – firstIn + 1 (+1 perquè quan hi ha un element a la cua els dos punters l'apunten i per tant la resta dona 0), donant la mida. Finalment si els punters estan creuats, es retorna el llarg total de l'array – firstIn + lastIn + 1, és a dir, la suma de els elements ocupats des l'inici de la cua fins a lastIn i els elements ocupats des de firstIn fins al final de la cua.

- isEmpty
  - Entrada: cua circular.
  - Sortida: booleà indicant que la cua és buida.
  - Descripció: es compara el valor de firstIn amb -1, lastIn també serviria però no es necessari.

- isFull

- Entrada: cua circular.
- Sortida: booleà indicant que la cua es plena.
- Descripció: és mira que lastIn + 1 sigui = a firstIn, això amb el mòdul indica que la cua és plena.

- queryLogsByPos
  - Entrada: punter a cua circular, enter amb posició a consultar(posició de la cua no de l'array, es a dir que es compta a partir de firstIn), punter a log on es vol guardar el resultat.
  - Sortida: booleà indicant si hi havia log a la posició.
  - Descripció: si la cua no és buida i la posició es menor o igual que la última posició omplerta es fa una còpia del log de dita posició dins el log passat per paràmetre.

4.2.1.4 Llibreria de gestió de memòria

Aquesta llibreria consta de la classe LogMem, on tenim els següents atributs. Un punter a la zona de memòria a reservar, una llargària, un offset i un array de punters.

La zona de memòria consisteix a una array de diferents cues, per tant la llargària es la de l'array i l'offset es la mida d'una posició de l'array. Finalment tenim els punters a cada direcció de memòria de cada posició de l'array.

Tant la llargària com l'offset son variables estàtiques, la llargària és el nombre d'atacs i l'offset es la mida d'una cua.

Tenim els mètodes públics següents:

- LogMem:
  - Entrada: cap.
  - Sortida: cap.
  - Descripció: és el constructor de la classe. Aquí es fa la reserva de memòria de l'array de cues fent un malloc(length * offset), després es fa un recorregut per cada posició de l'array assignant la seva direcció al vector de punters i es crida a la funció initQueue.

- add:
  - Entrada: log a afegir
  - Sortida: booleà indicant si s'ha eliminat algun log per afegir el nou.
  - Descripció: es crida la funció enqueue de la manera següent: enqueue(queuePtrs[log.attackType], log), on log.attackType coincideix amb l'índex de la cua que ens interessa i log es el log passat per paràmetre per afegir.

- query:
  - Entrada: una taula amb els atacs a consultar, el nombre d'atacs de la taula, dos variables de tipus time_t que indiquen el rang de temps que es vol consultar, el punter al nombre de logs que s'indicarà a la sortida.
  - Sortida: punter a l'array de logs que s'han consultat.
  - Descripció: s'inicialitza el numero de logs trobats a 0, es calcula el màxim nombre de logs a trobar (atacs rebuts per paràmetre * mida

d'una cua) i es reserva una array de logs d'aquesta mida. A continuació es fa un recorregut per els atacs passats per paràmetre i es crida a funció auxiliar timeFilter, aquesta retorna els logs resultat de la consulta. Com a tal al mètode query es filtra per tipus d'atac i al mètode timeFilter es filtren els logs d'un tipus específic per temps. L'array que retorna està ordenada per tipus d'atac i després per temps, una possible millora es fer que ordeni per temps.

I el mètode auxiliar i privat següent:

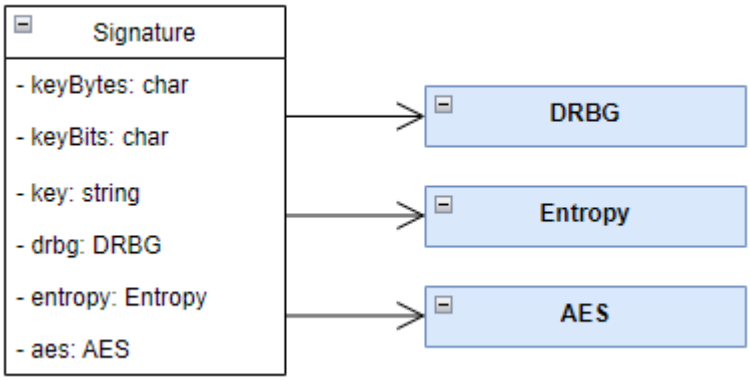
- timeFilter:
  - Entrada: punter a la cua que es filtra, rang de temps a consultar igual al del mètode query, punter a array de logs resultants, numero de logs fins al moment.
  - Sortida: numero de logs actualitzats.
  - Descripció: es recorre la cua rebuda fins l'última posició omplerta, es consulta el log de cada posició i es mira si el temps del log es dins el rang indicat amb la funció difftime de la llibreria de c, en cas afirmatiu es guarda una còpia a l'array resultant.

4.2.1.5 Protecció d'integritat

La protecció d'integritat de les dades es fa mitjançant una signatura AES CMAC amb una clau simètrica de 128 bits.

Per dur a terme aquesta tasca s'ha fet servir la llibreria TLS adaptada a Mbed OS. Aquesta llibreria consta de les eines necessàries per signar, xifrar i per la creació de les claus que s'utilitzen a les operacions esmentades. Conté les classes DRBG i Entropy per la creació de claus i la classe AES per operar amb aquest mètode criptogràfic.

L'ús dins el codi es fa a través d'una classe anomenada Signature.



Il·lustració 7. Diagrama de classes de la signatura

Aquesta classe té el constructor i dos mètodes.

- Constructor:
  - Entrada: cap.
  - Sortida: cap.
  - Descripció: crida a les funcions init de la llibreria Entropy i de la DRBG.

- setKey:
    - Entrada: cap.
    - Sortida: codi d'error.
    - Descripció: Inicialitza la llavor del generador de nombres aleatoris en base als objectes de la funció d'entropia i d'un string que ha de ser únic per cada clau. Després crida a la funció generativa de nombres aleatoris indicant la mida que ha de tenir. Finalment estableix la clau per el seu ús al fer servir la classe AES.
- getSignature:
    - Entrada: el text a signar en format string i la llargària d'aquest.
    - Sortida: un punter a el string amb el contingut de la signatura calculada.
    - Descripció: es crida a la funció AES CMAC indicant la clau, el nombre de bits que té, el text d'entrada, la llargària del text i el punter a on ha de deixar la signatura.

A la memòria de logs s'ha afegit una instància d'aquesta classe per fer servir les seves operacions i s'ha creat un array de signatures on hi haurà una signatura per cada cua de logs.

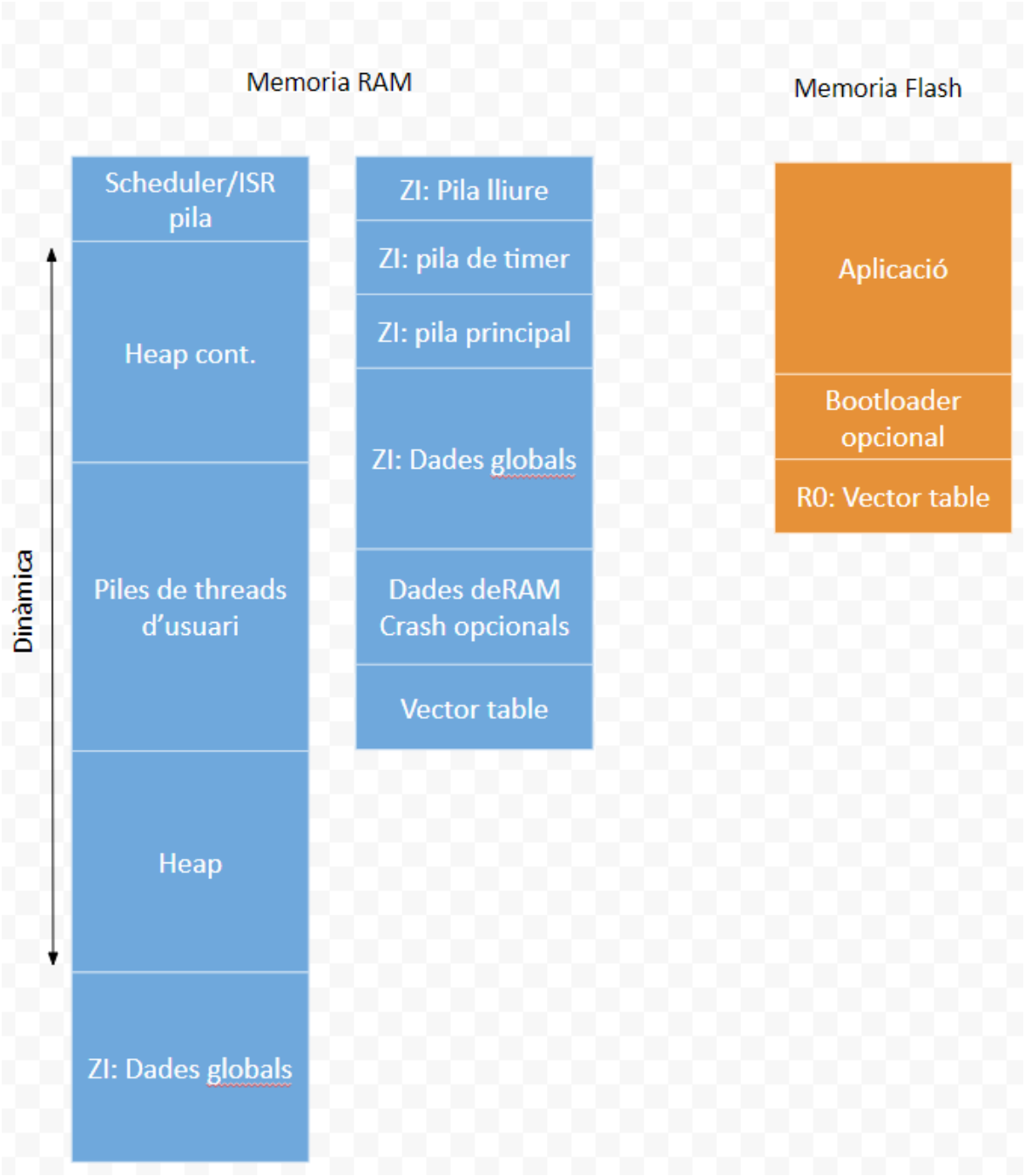
Per mantenir la integritat de memòria, quan s'afegeix un log s'actualitza la signatura de la cua que es modifica. Quan es consulta es torna a calcular la signatura i es compara amb la de l'array, en cas de no coincidir no hi ha integritat i es crida a la funció initQueue per buidar la cua ara que la informació que conté no és vàlida.

La protecció contra integritat resideix en la confidencialitat de la clau, per protegir-la s'acostuma a guardar en un mòdul que tenen els microprocessadors, l'HSM (Hardware Security Module). Aquest mòdul consisteix a una memòria petita amb el propòsit d'emmagatzemar les dades com claus, identificadors, que requereixen de privadesa, donat que té una protecció hardware que impedeix l'accés.

La família de microprocessadors ARM-STM32 té aquest mòdul, però ha de venir habilitat de sèrie i aquest no es el nostre cas.

4.2.1.6 Guardat en memòria Flash

La estructura de la memòria de la placa és la següent.



Il·lustració 8. Estructura de memòries de la placa

A la imatge podem observar la organització interna de les memòries RAM i Flash de la placa amb el sistema Mbed OS. En aquest apartat ens centrarem en la Flash, donat que es la única que permet la persistència de dades.

Aquesta està dividida en una vector table, un espai per afegir un bootloader (o varis) opcionals i l'espai per l'aplicació. Sota aquesta premissa es teoritza que la única manera de

reservar espai es tenir en compte la mida que ocupa la aplicació deixant una part lliure. Amb aquesta idea en ment es va fer un programa de proba.

Mbed ofereix la classe FlashIAP, que permet escriure a la Flash interna del microprocessador tal i com interessa.

Tenint en compte que la mida de l'aplicació és desconeguda, la proba consisteix en esborrar i escriure l'últim sector de la Flash. Malauradament obtenint com a resultat un Segmentation Fault. Raó d'això és que per la placa NUCLEO-F767ZI, la memòria Flash està completament situada en una Read - Only Memory.

Per tal de tenir persistència caldria l'ús d'un xip extern de Flash i per tant no està dintre de l'abast d'aquest treball donat que pretén ser un IDS específic per aquesta placa.

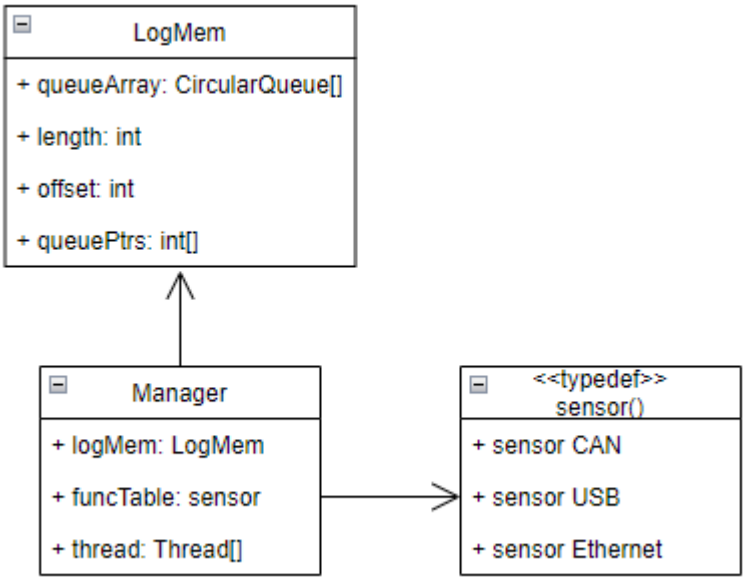
Cal esmentar que la placa no consta de cap bateria per mantenir un rellotge funcionat i així guardar el temps real, la manera de tenir aquest temps es enviar-lo des de un dispositiu extern en el boot i inicialitzar-lo dins la placa cada cop que es torni a encendre. Tenir el temps real era un requeriment quan es volia fer ús de la Flash, però en aquestes circumstàncies s'ha optat per guardar el temps transcorregut des del boot.

4.2.2 Manager

El manager es el programa principal de l'IDS, es el codi encarregat de crear threads per cada sensor, gestionar el sincronisme i la memòria.

En un inici la idea era fer que l'IDS funcionés com a servei en background però investigant de Mbed OS s'ha trobat que no hi ha la opció de fer-ho, només disposa d'API per serveis Bluetooth, a diferencia d'altres sistemes operatius com Linux on es pot afegir a la configuració de la rutina de init quins processos es volen crear. Finalment s'ha optat per crear una classe amb funcions per inicialitzar i finalitzar els threads de l'IDS, per tant quan es crea una aplicació per Mbed s'han de importar les llibreries que es mostren en aquest projecte i cridar a les funcions esmentades per fer servir l'IDS.

Per implementar el manager es crea altre projecte on s'importa la llibreria de memòria i es creen dues de noves, una per el manager i altre per els sensors. Les classes queden com es pot veure a continuació.



Il·lustració 9. Diagrama de classes del manager

La llibreria manager crea una instància de memòria de logs, al constructor de LogMem s'inicialitzen les estructures de dades necessàries.

A la llibreria de sensors es crea una funció per detectar cada tipus d'atac i es declara el tipus funció sensor, establint un format específic. Llavors a la llibreria manager es crea un vector de funcions del tipus sensor amb la mida del nombre d'atacs. A cada posició s'introdueix el nom de la funció de cada atac.

Es crea una taula de threads on cada fil correspondrà a un sensor, la llibreria utilitzada es la Thread de la API de Mbed OS.

4.2.2.1 Classe manager

Aquesta classe té un constructor i un mètode tancar.

- Constructor:
  - Entrada: cap.
  - Sortida: cap.
  - Descripció: recorregut de la taula de threads cridant a la funció start de la llibreria. Aquesta rep per paràmetre la funció callback que permet passar els paràmetres d'entrada al thread creat junt amb el nom de la funció que s'executarà. Es passa la posició corresponent de la taula de funcions.
  - s i una referència a la classe LogMem.
- Close:
  - Entrada: cap.
  - Sortida: cap.
  - Descripció: Per cada thread creat es crida a fa un join.

4.2.2.2 Llibreria de sensors

Aquesta llibreria fa ús de la classe Mutex de Mbed OS per el sincronisme i de la DigitalOut per l'ús de LEDs de la placa, els quals són útils per fer test.

Per cada tipus d'atac es crea una funció encarregada de la seva detecció, tal com s'ha dit, s'ha de afegir a la taula de funcions.

El format es el següent:

- Sensor estàndard:
  - Entrada: punter a classe LogMem.
  - Sortida: cap.
  - Descripció: es fa un bucle infinit en el qual el cos consisteix a avaluar si hi ha atac, el mètode de detecció difereix per cada situació, en cas afirmatiu es crea un log del tipus indicat i dins d'una secció crítica, s'afegeix a memòria el log i aquesta es signa dins la pròpia funció d'afegir. Quan s'afegeix un log a memòria s'encén un LED per tal de veure que s'ha detectat un atac, això es fa com a petit joc de proves però no es escalable, donat que tan sols hi ha 3 LEDs. Aquesta manera de definir seccions critiques comporta que en cas que es faci una consulta a memòria després que es detecti un atac o es creï un log però abans que s'afegeixi a memòria el resultat seria un fals negatiu, donat que no apareixeria cap atac recent tot i haver-hi un just en el moment. Altre possible implementació seria incloure la creació del log dins la secció crítica, però en aquest cas si altre thread detectés un atac i volgués crear el log hauria d'esperar i la data de creació del log seria més tard que la real. Com que la premissa inicial de l'IDS es per fer anàlisi forense més que per rebre un avís en temps real, el fals negatiu es menys problemàtic que tenir dates incorrectes dels atacs.

4.2.3 Sensors

4.2.3.1 Denegació de Servei per Ethernet

Per seguir un ordre coherent es procedirà a explicar la realització de l'atac per després veure la detecció.

La implementació dels scripts ha fet ús de la llibreria Scapy, es un programa de tractament de paquets que permet la seva creació, captura, replicació amb un ampli nombre de protocols, permetent dur a terme operacions com escanejar, veure quina ruta segueixen, descobrir xarxes i realitzar atacs.

La denegació de servei es pot fer de diverses maneres en funció de si fas servir una o varies IPs o ports.

Per començar s'ha implementat un script en el qual es fa un bucle infinit on s'envien paquets TCP/IP al port 80 de la placa indicant com a IP d'origen la del propi PC i com a IP destí la de la placa. Les quals s'inicialitzen als scripts de setup.

Una següent versió afegeix un rang de ports d'origen i envia les trames a partir d'un port escollit aleatòriament

La llibreria Scapy permet enviar paquets de manera manual i la IP d'origen es pot escollir lliurement, per tant també s'ha fet que enviï paquets amb IPs aleatòries, dins el rang possible a la xarxa 10.0.0.0\24. Així es simula un atac DoS distribuït dificultant la seva detecció i essent així un context més realista. Això es fa enviant trames al port 80 i aleatòriament.

Tenint en compte com s'ha realitzat l'atac hi ha varietat d'estratègies per la seva detecció. El concepte més general es el d'avaluar si en un lapse de temps curt hi ha un flux molt gran de trames d'entrada, tot i que si no es te en compte cap paràmetre més podria haver-hi un fals positiu si hi ha molta càrrega. Per evitar aquest cas es pot tenir en compte la IP d'origen o el port, així assegures que la detecció sigui cartera, tot i que a canvi una denegació de servei distribuïda passaria inadvertida, a aquest cas se'l coneix com a fals negatiu.

Si hi ha un fals positiu es fa un esforç per mirar de solucionar un atac que no existeix perdent temps, però un fals negatiu continua essent més perillós així que s'ha pres la decisió de avaluar tenint en compte el flux de trames d'entrada, independentment de la IP. A més això té l'avantatge de ocupar menys espai a l'hora d'afegir atributs extra per cada log.

Per detectar les trames s'ha fet servir la llibreria que ofereix Mbed OS anomenada EthernetInterface.h.

5 Avaluació

En aquest apartat es mostren les proves realitzades per comprovar si el sistema funciona correctament.

Primer es mostren els tests individuals que s'han fet per veure com funciona cada mòdul. Això es fa de manera acumulativa, primer revisant que cada classe emprada per fer la llibreria de memòria funcioni. Aquestes s'utilitzen al manager per guardar els atacs detectats i per tant la següent proba es del manager i les seves llibreries en base a la memòria de logs. Finalment es fan proves de l'atac de denegació de servei i del seu sensor, el qual es creat pel manager.

Per fer els tests s'ha fet servir el debugger de Mbed Studio. Aquest debugger permet veure els diferent threads que corren al sistema, la pila de crides, les variables locals i globals, es poden ficar expressions per veure si es compleixen certes condicions en parts del codi i finalment l'ús de breakpoints.

5.1 Memòria de logs

Seguint l'estructura establerta a l'apartat d'implementació es mostraran les proves fetes per cada llibreria o classe.

5.1.1 Llibreria de logs

Per avaluar que la llibreria funciona correctament s'han creat logs de diversos tipus en bucle afegint un delay entre creacions per veure si el temps assignat és correcte, després s'han fet còpies d'aquests logs i finalment s'ha cridat al mètode toStringLog, el qual internament crida al mètode toStringAttackType, verificant així tots els mètodes de la llibreria.

En aquesta iteració el string d'atributs queda buit ja que és depenent del tipus de sensor i per tant es testejarà en l'apartat dels sensors.

Els resultats observats amb el debugger han estat els següents:

Log	Temps	Tipus d'atac	Validesa
1	81	DoS_CAN	Sí
2	81	DoS_USB	Sí
3	81	DoS_Ethernet	Sí
4	82	DoS_CAN	Sí
5	82	DoS_USB	Sí
6	82	DoS_Ethernet	Sí
7	83	DoS_CAN	Sí
8	83	DoS_USB	Sí
9	83	DoS_Ethernet	Sí

Taula 5. Test de llibreria de logs

Les funcions de toString també han funcionat correctament tal com es mostra a la línia següent: "Datetime: Thu Jan 1 00:01:21 1970 Attack Type: DoS CAN Other attributes: ".

5.1.2 Llibreria per tractar amb cues circulars

Aquesta llibreria s'ha provat amb tres tests diferents, en els tres s'ha definit com a llargària de la cua 3 elements per poder veure en detall si el funcionament es correcte.

El primer consisteix a inicialitzar la cua i executar un bucle infinit on es crea un log i s'encua.

Nombre d'elements dins la cua	Temps	Tipus d'atac	firstIn	lastIn	Validesa
0	-	-	-1	-1	Sí
1	203	DoS_CAN	0	0	Sí
2	204	DoS_CAN	0	1	Sí
3	205	DoS_CAN	0	2	Sí
3	206	DoS_CAN	1	0	Sí
3	207	DoS_CAN	2	1	Sí
3	208	DoS_CAN	0	2	Sí
3	209	DoS_CAN	1	0	Sí
3	210	DoS_CAN	2	1	Sí
3	211	DoS_CAN	0	2	Sí

Taula 6. Test de cues circulars

Es pot observar com s'afegeix un log per segon a la cua del tipus d'atac CAN i passa el següent; els punters s'han inicialitzat correctament amb el valor -1, quan s'afegeix el primer element els dos l'apunten tenint el mateix valor, segons s'afegeixen més elements el punter lastIn s'incrementa i finalment quan la cua és plena i es continuen encuant elements, cada cop que s'afegeix, es desencua un element i això es veu en que el punter firstIn s'incrementa i el lastIn passa a tenir el valor de la primera posició, s'han creuat, segons s'afegeixen més logs continuen incrementant-se fins que tornen a creuar-se i això es fa en bucle.

El següent test es per veure si la funció de consultar logs per posició funciona correctament, s'omple una cua i es crida a la funció per cada element de la cua. Els resultats obtinguts són favorables donat que el mètode retorna una copia del log corresponent a la posició donada.

Per acabar s'ha realitzat un test per veure si els mètodes isEmpty, isFull i dequeue funcionen correctament.

Operació	Nombre d'elements dins la cua	isEmpty	isFull	firstIn	lastIn	Validesa
initQueue	0	True	False	-1	-1	Sí
enqueue	1	False	False	0	0	Sí
enqueue	2	False	False	0	1	Sí
enqueue	3	False	True	0	2	Sí
dequeue	2	False	False	1	2	Sí
dequeue	1	False	False	2	2	Sí
dequeue	0	True	False	-1	-1	Sí
dequeue	0	True	False	-1	-1	Sí

Taula 7. Test de cues circulars desencuant

El resultat obtingut es correcte. Observem que el mètode desencuar no modifica indegudament els punters de fisrtIn i lastIn, en cas de no haver elements en retorna cap. Es pot veure com al desencuar el punter firstIn s'incrementa, ja que s'esborra l'element més vell, fins ser igual que lastIn quan només queda un element i finalment tornant a tenir els dos punter a -1 quan la quan és buida.

5.1.3 Llibreria de gestió de memòria

Aquesta llibreria consta de constructor i de dos operacions importants, que són la de afegir logs i la de consultar-los. S'ha fet un test per cadascuna d'aquestes.

El test de l'operació afegir es un bucle infinit on es crea un log de cada tipus d'atac i es crida a l'operació afegir, amb el debugger es pot veure com els logs s'afegeixen a la cua corresponent amb el seu tipus d'atac.

Per veure si les consultes es fan bé s'omplen les tres cues i es crida al mètode de consulta amb diferent paràmetres.

Primer es fa una consulta de tots els logs que hi ha, el mètode rep una array amb tots els tipus d'atac possibles i el rang de temps des del moment 0 fins el temps actual, calculat amb la funció de c time(0). El resultat obtingut es favorable, una array de logs amb tots el que hi ha. A continuació es fa una consulta dels logs d'un tipus d'atac guardats en tot el temps i el resultat es favorable. Finalment es fa una consulta dels logs d'un tipus d'atac en un rang de temps concret i es veu com no retorna tots els logs del tipus d'atac, tal i com s'esperava.

5.2 Manager

Les funcionalitats que el manager ha de oferir inclouen la gestió de memòria i la de sensors. L'avaluació de la memòria s'ha vist prèviament, així que aquest apartat es centra en la gestió de sensors.

Cada sensor funciona mitjançant un fil d'execució independent, això suposa que el manager ha de crear i tancar aquests fils i durant el seu temps de vida, que des de la creació de l'IDS equival al temps en que la placa estigui encesa, fer ús de mecanismes de sincronització per l'accés a la memòria. Llavors el test queda dividit en dues parts, una primera de creació i tancament correcte de fils i una segona de veure que el sincronisme sigui correcte.

En ambdós tests s'han creat tres fils tal i com s'ha anat fent en totes les altres proves, ja que tot i que en el projecte només s'ha implementat un tipus d'atac, cal verificar que les estructures de dades i les funcionalitats siguin vàlides per múltiples threads

5.2.1 Creació de sensors

Per avaluar que tots els fils es creen i s'executen de manera concurrent s'ha fet una prova en la qual s'assigna un LED de la placa a cada fil i aquest s'encén al crear-se. Durant la execució cada fil canvia l'estat del LED i així es pot veure parpellejar el LED si està viu. El resultat obtingut es favorable.

5.2.2 Sincronisme de sensors

Feta la prova de creació es passa a avaluar el sincronisme. Com ja s'ha explicat cada sensor avalua si hi ha atac, crea un log i per guardar-lo i signar-lo, entra en una secció crítica i això es fa cíclicament. Com que volem fer una prova amb fils de sensors teòrics però dels quals no hi ha una implementació en comptes d'avaluar si hi ha un atac, a cada iteració es crea un log i s'afegeix a memòria, així es fan accessos a la secció protegida i es pot veure si el sincronisme funciona.

Donat que la realització d'un programa de consultes de memòria quedava fora de l'abast d'aquest projecte aquesta no es pot veure des de fora i consultar-la amb el debugger no seria una bona estratègia per comprovar sincronisme. Degut a que la memòria s'ha testejat prèviament tampoc suposa un conflicte ja que sabem que els logs s'afegeixen correctament i el que ens interessa es el flux de treball, per aquesta tasca es fan servir LEDs. Amb un concepte similar al del test de creació cada fil té assignat un LED i en aquest cas el canvi d'estat del LED s'introdueix a la secció crítica. El resultat obtingut difereix del test anterior en que ara els LEDs no parpellegen a la vegada sinó que ho fan esglaonadament, deixant constància del sincronisme.

5.3 Sensors

Per avaluar el funcionament dels sensors es llencen atacs fets en Python i es comparen els resultats que dona la placa amb els que hi ha a l'analitzador de xarxa emprat, Wireshark.

Es torna a fer servir l'estratègia dels LEDs, en aquest cas romanent apagats excepte quan es detecta l'atac, llavors s'encén un LED en forma d'avís emulant el funcionament d'un IDS en temps real. Això només es una mesura de test ja que al haver una quantitat molt limitada de LEDs no es podria escalar a la quantitat d'atacs que es pot arribar a detectar, però resulta suficient per avaluar si es detecten els atacs quan es llencen.

5.3.1 Denegació de Servei per Ethernet

S'han fet tres proves amb diferents complexitats per veure el nivell d'assoliment del sensor, seguint els atacs definits en l'apartat 4.2.3.1. Els resultats han estat els següents.

IP origen	IP destí	Port	Atac detectat
10.0.0.1	10.0.0.2	80	Sí
10.0.0.1	10.0.0.2	Aleatori	Sí
Aleatòria	10.0.0.2	80	Sí
Aleatòria	10.0.0.2	Aleatori	Sí

Taula 8. Test sensor atac DoS

El resultat era d'esperar donat que el disseny del sensor s'ha fet en funció de la pròpia implementació dels atacs deixant de banda la IP d'origen.

Per estudiar els falsos positius s'ha fet una versió del programa de connectivitat descrit a l'apartat 4.1.3. En aquest programa es creava una connexió a mbed.org, la diferencia que s'afegeix per aquest test es la de fer-ho amb un bucle infinit, això implica enviar i rebre trames TCP/IP. Configurant aquest programa amb un temps d'espera adequat entre comunicacions el sensor no detecta cap atac.

6 Conclusions

En aquest treball s'ha dissenyat i desenvolupat un software de detecció d'intrusions per a un sistema encastat. L'entorn de treball ha sigut la placa NUCLEO-F767ZI amb el sistema operatiu Mbed OS.

L'IDS implementat consta de tres mòduls diferents. El primer gestiona la memòria, on s'hi troben les estructures de dades per emmagatzemar logs dels atacs que els altres mòduls detecten i consta de protecció contra ofensives a la integritat de dades. El segon és el manager, el qual s'encarrega de les estructures necessàries per la creació, sincronització i finalització dels fils que és requereixen per a cada sensor d'un atac. Finalment es disposa del conjunt de sensors d'atacs, els quals s'integren al manager. En aquest treball s'ha implementat un d'exemple que detecta denegacions de servei distribuïdes.

En termes generals es considera que els objectius del projecte han estat assolits, donat que s'ha definit una arquitectura i s'ha creat un conjunt de llibreries que permeten detectar i emmagatzemar informació d'atacs. S'ha implementat un únic sensor però s'ha demostrat que el sistema pot funcionar per detectar múltiples atacs sempre que la mida de la memòria de logs no superi l'espai de la placa. Malauradament hi ha parts del disseny que no s'han pogut implementar per limitacions del sistema, com succeeix amb la persistència de dades entre reinicis ja que la Flash es una Read-Only Memory. També hi ha altres casos en que la idea inicial s'ha pogut implementar però requerint de canvis d'estratègia, com es pot veure en el manager. El planejament inicial va ser executar-ho com un servei en background, però Mbed OS no oferia aquesta funcionalitat. Finalment, es va optar per fer una llibreria, donant la possibilitat de fer ús de l'IDS cridant les seves funcions.

Tenint en compte el context en que ens trobem i l'eina proposada es poden treure diverses conclusions en quant als usos del sistema. Es tracta d'un Host IDS, els atacs que detecta són a la pròpia placa i això dóna lloc a dos usos. En un s'executa junt a una aplicació específica que es vol protegir, per tant aquesta ha de compartir els recursos de la placa amb l'IDS, hi haurà casos en el quals aquesta opció serà rentable i altres on no es podrà assumir. A més, donat que no emet avisos en temps real aquest cas no es veu del tot beneficiat. Altre possibilitat és fer-lo servir com a honey pot, tenint l'IDS dins una xarxa, recollint dades dels atacs i posteriorment estudiar-los per poder desenvolupar millors estratègies de protecció per aquests tipus de placa. Això es veuria beneficiat de tenir un paradigma de detecció per heurístiques, detectant comportaments desconeguts. L'enfoc actual és de detecció de patrons però el sistema dóna l'opció de crear un sensor heurístic.

De cara al futur es pot realitzar un conjunt de millores. Per començar hi ha la possibilitat d'afegir sensors per més atacs, a l'apartat de disseny es fa un anàlisi de la placa i es mostren possibles camins a prendre. Es poden afegir al sistema avisos en temps real dels atacs, als jocs de proves de l'apartat d'avaluació es pot intuir una idea de com es podria fer. També és possible fer que el sistema a més de detectar els atacs, respongui per intentar neutralitzar-los, creant un Intrusion Prevention System (IPS). Per consultar la memòria es podria crear una aplicació que demani informació des del terminal de l'administrador de la xarxa. Això requeriria de mecanismes d'autenticació però facilitaria el treball amb l'IDS. Addicionalment, es poden buscar maneres de fer que la memòria sigui persistent, requerint l'adquisició d'una memòria Flash externa.

A la introducció s'ha explicat que la motivació per fer un IDS és en part que sempre es poden trobar maneres diferents d'atacar un sistema, essent important el seu estudi. En aquest treball s'ha tractat una petita fracció dels atacs possibles, però ara existeix una eina més a partir de la qual detectar molts més.

7 Referències

[1] https://os.mbed.com/platforms/ST-Nucleo-F767ZI/ [Placa NUCLEO-F727ZI] 1/07/2021

[2] https://www.st.com/en/microcontrollers-microprocessors/stm32f767zi.html.[MCU] 1/07/2021

[3] https://www.st.com/en/evaluation-tools/nucleo-f767zi.html.[Imatge placa] 1/07/2021

[4] https://developer.arm.com/ip-products/processors/cortex-m/cortex-m7.[MCU] 1/07/2021

[5] https://os.mbed.com/mbed-os/ .[Mbed SO] 1/07/2021

[6] https://os.mbed.com/docs/mbed-os/v6.15/introduction/index.html .[Documentació Mbed] 1/07/2021

[7] https://www.w3schools.com/cpp/default.asp .[Tutorials C++] 1/07/2021

[8] https://www.cplusplus.com/reference/ctime/ .[ctime lib] 1/07/2021

[9] https://forums.mbed.com/t/save-read-files-in-nucleo-f767zi-flash/10432/4 .[Flash Mem] 20/12/2021

[10] https://os.mbed.com/docs/mbed-os/v6.15/apis/flash-iap.html. [Flash Mem] 20/012/2021

[11] https://os.mbed.com/questions/81819/SAVING-DATA-50-KB-OF-DATA-ON-FLASH-MEMOR/l. [Flash Mem] 20/12/2021

[12] https://tls.mbed.org/kb/how-to/generate-an-aes-key .[Signatura] 28/12/2021

[13] https://tls.mbed.org/kb/how-to/encrypt-with-aes-cbc.[Signatura] 28/12/2021

[14] https://en.wikipedia.org/wiki/Embedded_system. [Sistema Encastat] 01/07/2021

[15] https://blog.alicegoldfuss.com/function-dispatch-tables/. [Taula de funcions] 23/12/2021

[16] https://scapy.net/. [Llibreria Atacs Pyhon] 3/01/2022

[17] https://www.st.com/resource/en/application_note/dm00272912-managing-memory-protection-unit-in-stm32-mcus-stmicroelectronics.pdf .[Protecció memòria] 14/11/2021