Sem vložte zadání Vaší práce.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Detekce anomálií v provozu IoT sítí

Bc. Dominik Soukup

Katedra počítačových systémů
Vedoucí práce: Ing. Tomáš Čejka

29. dubna 2018

Poděkování

Rád bych poděkoval vedoucímu diplomové práce Ing. Tomáši Čejkovi za odborné vedení, cenné rady a čas věnovaný konzultacím. Dále děkuji sdružení CESNET, z.s.p.o. za možnost práce na tomto projektu a poskytnutým prostředkům pro vývoj vytvořeného nástroje.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 29. dubna 2018 .....

Odkaz na tuto práci

Soukup, Dominik. *Detekce anomálií v provozu IoT sítí.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato diplomová práce se zabývá problematikou bezpečnosti v prostředí Internet of Things (IoT). Prvním cílem je analýza aktuálního stavu IoT sítí a identifikace bezpečnostních slabin bezdrátových senzorových protokolů. Druhým cílem je vytvoření nástroje, který v probíhající komunikaci detekuje nalezené bezpečnostní incidenty. V analytické části je čtenář seznámen s principem fog computingu a nově vzniklým modelem komunikace. Zároveň jsou důkladně rozebrány aktuálně používané protokoly včetně jejich bezpečnostních slabin. Součástí práce je návrh architektury, který je připraven na budoucí rozšiřování, což je nezbytné pro rychle se rozvíjející a širokou oblast IoT. Při návrhu byl kladen důraz na nízké hardwarové nároky tak, aby bylo možné provozovat výsledné řešení i na IoT branách s omezenými prostředky.

První část výstupu této práce tvoří rešerše aktuálního stavu IoT, která je obsažena v textu této práce. Druhou částí je modulární systém, který lze konfigurovat a přizpůsobit pro konkrétní topologii. Výsledný nástroj je implementovaný v jazyce C++ a rozšiřuje již existující IoT bránu BeeeOn o možnost detekce anomálií v bezdrátových senzorových protokolech. Výsledkem je nová verze této brány s mechanismem pro detekci útoků.

Klíčová slova IoT, detekce anomálií, BeeeOn, NEMEA

Abstract

This work covers security concerns and issues of the Internet of Things (IoT). The first aim is to analyse the actual situation of IoT and to identify vulnerabilities of the wireless sensor network protocols. The second aim is to develop a tool that is able to detect security incidents in communication traffic. The analytical part familiarizes the reader with the fog computing concept and new communication architecture. Simultaneously, there are thoroughly explored current IoT protocols including their vulnerabilities. This is followed by the tool design that is ready for the future extension, which is necessary for the rapidly growing area like IoT. During designing, low hardware requirements was emphasised so that it would be possible to deploy the created solution event on IoT gateways with restricted resources.

The first part of this work output is research of the current IoT state, which is contained in the text of this work. The second part is a modular system that is possible to configure and customize for target topology. The created tool is implemented in C++ language and extends the already existing IoT gateway BeeeOn by anomaly detection of the wireless sensor network protocols. The result is a new version of the BeeeOn gateway with the mechanism for attacks detection.

Keywords IoT, anomaly detection, BeeeOn, NEMEA

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Koncept internetu existuje již několik desítek let a pro spoustu lidí se stal nedílnou součástí pracovního i osobního života. V poslední době je možné sledovat stále rostoucí počet zařízení, která jsou do něj zapojena. Tento trend by měl pokračovat i do budoucnosti, a dokonce v ještě větším měřítku. Odhadem je přes 30 miliard připojených zařízení do roku 2020 [1]. Důvodem zrychleného růstu je expanze síťového připojení na veškeré elektronické zařízení a senzory, které umožní vzdálené ovládání a monitorování. Pro označení tohoto trendu se používá termín Internet věcí (Internet of Things, IoT).

Cílem IoT je usnadnit, zlepšit a ušetřit lidskou činnost napříč všemi odvětvími. Uplatnění se nachází zejména ve výrobních podnicích, dopravě nebo běžných domácnostech. Příklad konkrétního nasazení do osobního života popisuje článek [2], jehož cílem je měření kvality spánku a odhalení případných poruch. Monitorovací systém lze dále propojit například s ovládáním místnosti, které bude reagovat na aktuální úroveň spánku úpravou světel, oken nebo vzduchu.

Dále je upraven model komunikace, který již nevyžaduje zasílání zpráv centrálnímu serveru (north-south), ale podporuje přímou komunikaci mezi připojenými uzly (east-west). Oblast IoT nezahrnuje jen malá a nevýkonná zařízení, ale jeho součástí jsou i výkonná datová centra a pokročilé algoritmy, které vyhodnocují získané informace.

Hlavní hrozbou Internetu věcí je zejména bezpečnost a ochrana soukromí. Dochází zde k přenosu citlivých dat, která slouží k automatizovanému řízení dalších systémů, monitorování prostředí a zabezpečovacím účelům. Zároveň s masivním rozšířením nově připojených zařízení roste riziko vzniku nových útoků a možnosti způsobení větších škod. Příkladem bezpečnostního incidentu je útok na distribuční síť elektrického proudu na Ukrajině, který měl dopad na 225 000 zákazníků [3].

Pro potlačení vzniku hrozeb musí být součástí každé dnešní IoT sítě sada procesů, které umožní důvěryhodné získávání validních dat, vzdálenou správu a možnost detekce anomálií jako v běžných IP (Internet Protocol) sítích.

Cíl práce

Cílem diplomové práce je analyzovat množinu aktuálně používaných protokolů pro komunikaci v IoT sítích a identifikovat jejich bezpečnostní zranitelnosti. Při analýze bude věnována pozornost zejména bezdrátovým senzorovým protokolům. Na základě získaných znalostí bude navržen, implementován a otestován algoritmus pro monitorování a automatickou detekci anomálního provozu v IoT sítích. Algoritmus bude možné spustit v prostředí nově vznikající open-source brány BeeeOn [4], čímž dojde k rozšíření dostupných bezpečnostních funkcí.

Kapitola 1

Analýza

Kapitola se zabývá analýzou celkové architektury IoT sítí a způsoby pro její zabezpečení. Postupně je prozkoumán komunikační model, možné bezpečnostní hrozby a existující řešení pro obranu. Na základě analýzy jsou uvedeny funkční a nefunkční požadavky, které jsou kladeny na výsledný program. V závěru jsou vybrány konkrétní technologie pro realizaci.

1.1 Architektura IoT sítí

V blízké době se očekává stále větší nárůst zařízení, která jsou připojena k internetu. Dle odhadů by jejich počet měl v roce 2020 překročit 30 miliard [1]. Pro takové množství připojení už není možné, aby každé zařízení komunikovalo přímo se vzdáleným datovým centrem, protože nároky na potřebnou šířku pásma by byly obrovské [5]. Dalším problémem je často velmi omezený výkon připojených prvků, který je nezbytný pro použití bezpečnostních funkcí umožňujících kompletně zabezpečenou komunikaci.

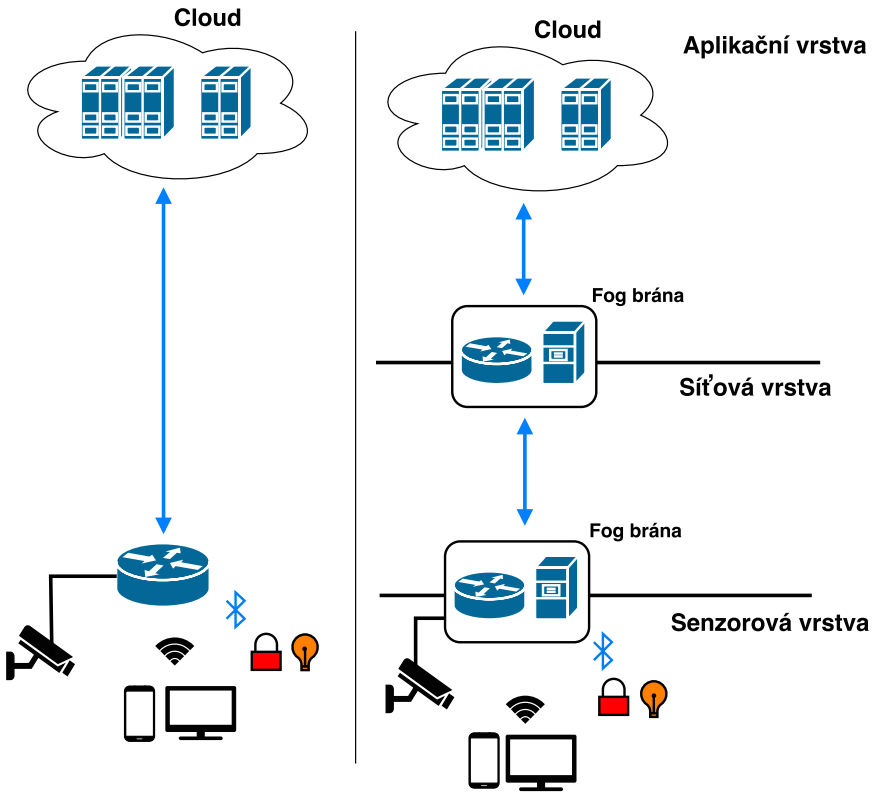
Řešením těchto problémů je do probíhající komunikace přidat několik podvrstev, které umožní přesunout výpočetní výkon blíže ke koncovým zařízením, a tím celý proces zpracování dat provést efektivněji.

V následujících podkapitolách bude popsán nový model komunikace, který přináší IoT sítě.

1.1.1 Fog computing

Fog computing je rozšíření cloud computingu, které spočívá v přesunutí výpočetního výkonu blíže k okraji sítě. Rozšíření je umožněno přidáním síťových zařízení, které kromě běžných funkcionalit poskytují i výpočetní výkon pro běh externích programů. Programy je často možné nasadit pomocí kontejnerů nebo samostatných virtuálních strojů, což velmi usnadňuje jejich vývoj a distribuci [5].

Porovnání klasické a fog architektury z hlediska zpracování dat se nachází na Obrázku 1.1. V reálném nasazení může být použito i více fog vrstev, kde každá provádí určitý stupeň předzpracování a řízení dat. Naopak u klasického modelu tato možnost není a vše zajišťuje až datové centrum. Zavedením



Obrázek 1.1: Porovnání klasické a fog architektury

principů fog computingu vznikají pro síť a zejména zpracování dat následující výhody:

- **Zlepšení bezpečnosti**

  Síťové prvky jsou trvale napájené a připojené k internetu. Podporují pokročilé bezpečnostní funkce, a proto je možné například vytvářet šifrované tunelové spojení pro bezpečný přenos dat [5].

- **Nižší nároky na šířku pásma a latenci**

  Odeslaná data z koncových zařízení jsou zpracovávána a filtrována na okraji sítě. Tím je možné rychleji reagovat na přijaté zprávy a snížit nároky na latenci a šířku pásma. Zároveň krátkodobá data mohou být uložené ve fog vrstvě a centrální datové centrum může být využito pro

dlouhodobé údaje, které se zpracovávají pokročilými algoritmy pro analýzu dat [5].

- **Jednotná správa**

  Při správě sítě už se nemusí přistupovat přímo na koncové prvky, které často komunikují různými protokoly, ale stačí pouze řídit síťová zařízení v jednotlivých fog vrstvách, které odstiňují různorodost protokolů a nabízí standardizovaný přístup. Díky této abstrakci je zároveň zjednodušeno zpracování získaných dat a je umožněno přímé zasílání zpráv mezi koncovými prvky, které používají odlišné komunikační protokoly [5].

  Další výhodou je, že výpočetní i síťové funkce jsou zajištěny jedním hardwarovým zařízením, což značně usnadňuje požadavky pro nasazení.

Nevýhodou naopak může být distribuovaná topologie, která má větší nároky na údržbu.

1.1.2 IoT brána

IoT brána je síťové zařízení, které je umístěno velmi blízko koncových zařízení a představuje vstup do fog vrstvy. Jejím hlavním cílem je získávat data z připojených zařízení a poskytovat je vyšším vrstvám. Pokud je brána reprezentována výkonnějším síťovým prvkem, tak v něm zároveň může probíhat i základní zpracování dat.

Pro IoT sítě je typické, že obsahují velké množství koncových prvků komunikujících různorodými způsoby. Zejména senzory používají protokoly, které nepodporují IP spojení. Důvodem použití této komunikace je často velký důraz na nízkou spotřebu a specifické požadavky na způsob zasílání zpráv. Příkladem protokolů pro senzorové sítě je například: Z-Wave, Bluetooth a Zigbee. Jejich detailní popis se nachází v sekci 1.2. Tato různorodost vyžaduje, aby brána obsahovala dodatečná komunikační rozhraní, která umožní připojení nejrůznějších bezdrátových i drátových koncových prvků.

V současné době existuje mnoho různých bran jejichž parametry se liší dle způsobu nasazení a provozních nároků. Velkým problémem v této oblasti je malý důraz na bezpečnostní funkce, které umožní vzdálené řízení brány, kontrolu provozu a aktualizace programového vybavení. Z těchto důvodů vznikl opensource projekt BeeeOn [4] jehož cílem je vytvořit softwarovou IoT bránu, kterou bude možné spustit na různých hardwarových platformách. BeeeOn brána je navržena modulárně tak, aby byla schopna zpracovávat více rozdílných senzorových protokolů, a tím bude možné provozovat jedno univerzální zařízení namísto několika proprietárních. Zároveň je kladen důraz na bezpečnost, a proto veškeré údaje, které je možné získat o provozu, jsou poskytovány pro analýzu. Nad těmito údaji je postaven detekční algoritmus, který je výsledkem této diplomové práce.

1.1.3 Komunikační model a jeho hrozby

Při použití principů popsaných v předchozích kapitolách lze model komunikace rozdělit do následujících vrstev [6]:

- **Senzorová vrstva**

  Senzorová vrstva obsahuje veškerá koncová zařízení, které získávají informace ze svého okolí nebo vykonávají potřebnou službu [7]. Tato zařízení jsou připojena kabelově nebo bezdrátově k IoT bráně. K jedné bráně může být připojeno několik prvků, které komunikují odlišnými způsoby. Princip komunikace a možnosti topologie se odlišují na základě použité technologie.

  Velkým nebezpečím této vrstvy jsou zejména bezdrátové protokoly, protože při nepoužití zabezpečení může snadno dojít k odposlouchávání nebo úpravám provozu [6]. Dále se zde mohou vyskytovat zařízení, které jsou označeny jako zabezpečené, ale díky starší verzi komunikačního protokolu používají zastaralé bezpečnostní funkce nebo obsahují implementační chyby. Tento případ je velmi nebezpečný, protože vyvolává falešný pocit bezpečí. Útoky se také mohou zaměřovat na prvky s bateriovými zdroji, které mohou být nadměrnou komunikací účelově vybíjeny.

- **Síťová vrstva**

  Po zpracování senzorových dat na bráně je nutné získané informace odeslat dalším službám. K tomuto účelu slouží síťová vrstva. Cílem této vrstvy je také umožnit vzdálenou správu brány [7]. Pro výběr konkrétního protokolu je nutné znát rozhraní aplikační vrstvy. Ve většině případů je spojení vytvořeno pomocí protokolu HTTPS (Hypertext Transfer Protocol Secure) nebo technologie VPN (Virtual Private Network). Nad tímto spojením je postavena další služba pro výměnu zpráv. Příkladem může být: MQTT (Message Queuing Telemetry Transport), CoAP (Constrained Application Protocol) nebo AMQP (Advanced Message Queuing Protocol). Tyto protokoly jsou podrobněji popsány v Sekci 1.2.

  Bezpečnostní hrozby síťové vrstvy jsou stejné jako v klasických sítích. Je potřeba dodržet principy důvěry, integrity a dostupnosti. Tímto přístupem je možné předejít útokům jako: DDoS (Distributed Denial Of Service), MITM (Man In The Middle) a podvržení informací. Zároveň je nutné nezapomenout, že se zde většinou vyskytuje M2M (Machine To Machine) komunikace a je důležité použít vhodná komunikační rozhraní [6]. Častým případem bývá zastaralá verze firmware, jehož napadení může vést k nestandardnímu chování nebo dokonce ke kompletnímu převzetí kontroly.

- **Aplikační vrstva**

  Aplikační vrstva se stará o ukládání dlouhodobých dat a jejich finální zpracování. Zároveň zobrazuje uživateli zpracované informace a umožňuje provádět konfiguraci celé sítě. Z důvodu její možné rozsáhlosti je kritické, aby správa topologie podporovala automatizaci [7].

  Tato vrstva je umístěna vetšinou v datovém centru a umožňuje vzdálený přístup. Její bezpečnostní problémy lze přirovnat k problémům cloud computingu. Dle množství požadovaných funkcí může být různě složitá a s rostoucí složitostí se také liší nároky na úroveň zabezpečení. Příkladem možných útoků může být: Buffer Overflow, SQL Injection nebo DDoS.

Mezi obecné problémy patří způsob nasazení. Koncové IoT prvky mají odlišné parametry než běžná uživatelské zařízení. Zároveň se liší i průběh komunikace, která je v IoT méně heterogenní a často založena na M2M. V případě umístění všech koncových prvků do jednoho společného segmentu se komplikuje nastavení bezpečnostních pravidel a také roste možný dopad provedeného útoku. Při úspěšném napadení jednoho prvku útočník získá možnost rozšíření na další uzly ve stejném segmentu.

1.2 Používané komunikační protokoly

Jedním z hlavních cílů IoT je možnost vzájemného propojení různých komunikačních protokolů, které umožní automatizovanou výměnu zpráv mezi všemi dostupnými zařízeními. Tímto způsobem je následně možné zefektivňovat a usnadňovat lidskou práci.

V následujících podkapitolách bude popsána množina aktuálně často používaných protokolů včetně jejich bezpečnostních funkcí.

1.2.1 MQTT

MQTT je otevřený síťový komunikační protokol typu publish/subscribe, který byl navržen v roce 1999. Již od návrhu byl zaměřen na nízkou náročnost komunikace a jednoduchost implementace. Díky těmto vlastnostem je velmi vhodný pro IoT a M2M systémy [8].

1.2.1.1 Způsob komunikace

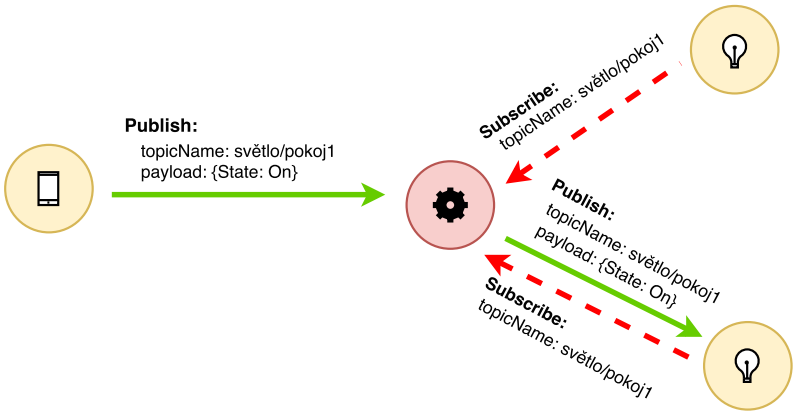
Protokol MQTT je postaven nad transportní vrstvou TCP (Transmission Control Protocol) a využívá model publish/subscribe, který vychází z tradičního způsobu zasílání zpráv typu klient-server. Roli serveru zde plní speciální uzel, který se nazývá broker. Broker je známý všem ostatním klientům, kteří mohou zasílat zprávy pomocí operace *publish* nebo se přihlásit o příjem zpráv díky operaci *subscribe*. Na základě provedených operací broker přijímá zprávy a rozhoduje o jejich přeposlání. Způsob odeslání zprávy závisí na obsažených metadatech.

Nejčastěji o směru odeslání rozhoduje předmět (topic) zprávy. Předmět je tvořen jednoduchým UTF-8 řetězcem s hierarchickou strukturou, ve které jsou jednotlivé vrstvy odděleny lomítkem a každá z nich musí obsahovat minimálně jeden znak (např. domov/přízemí/světloŠatna). V předmětu zprávy mohou být některé vrstvy nahrazeny zástupnými symboly + a #. Symbol + dokáže nahradit pouze jednu úroveň předmětu libovolným řetězcem a symbol # umožňuje zastoupit více úrovní. Díky těmto symbolům mohou klienti odeslat nebo přijímat zprávy z více témat. Schéma topologie a ukázka využití předmětů zpráv se nachází na Obrázku 1.2.

Další užitečnou položkou protokolu MQTT je QoS (Quality of Service), která může nabývat hodnot 0, 1 nebo 2.

- **QoS 0**

  Veškeré zprávy jsou odesílány bez potvrzení a žádným způsobem není zvýšena úroveň spolehlivosti, která je shodná se spolehlivostí protokolu TCP.



Obrázek 1.2: MQTT architektura

- **QoS 1**

  Pomocí potvrzování zajišťuje, že každá zpráva bude příjemci doručena alespoň jednou.

- **QoS 2**

  Umožňuje, aby každá zpráva byla spolehlivě doručena právě jednou.

Hodnota QoS se nastavuje vždy mezi dvěma uzly při navazování spojení. Z pohledu brokeru se může stát, že přijatá a odeslaná zpráva mají jiné QoS. Úrovně 1 a 2 dále umožňují perzistentní ukládání zpráv v případě, že příjemce je nedostupný. Zároveň platí, že s vyšší úrovní roste i režie komunikace [9].

1.2.1.2 Bezpečnost

Zabezpečení protokolu MQTT je možné rozdělit do následujících vrstev:

- **Síťová vrstva**

  Veškerá komunikace je postavena nad TCP/IP, a proto lze probíhající komunikaci zapouzdřit pomocí VPN připojení jako v běžných počítačových sítích. Kvůli větším nárokům na výkon je toto řešení vhodnější pro výkonnější zařízení jako jsou například IoT brány, které mohou s brokerem navázat site-to-site spojení [10].

- **Transportní vrstva**

  Na této úrovni se využívá šifrování provozu pomocí protokolu TLS (Transport Layer Security). Omezením této metody jsou požadavky na výkon, které mohou být poměrně vysoké pokud nastává časté navazování spojení [10].

- **Aplikační vrstva**

  Samotný protokol MQTT nedefinuje žádné šifrovací mechanismy na aplikační úrovni. Zabezpečení dat zde musí zajistit uživatel ještě před jejich zapouzdřením do MQTT zprávy. Ovšem tímto způsobem je možné šifrovat jen tělo zprávy a hlavička zůstává nezměněná.

  Pro autentizaci je možné využít ověření pomocí jména a hesla nebo x.509 certifikátu. Jméno a heslo je přenášeno nešifrovaně, a proto je vhodné tuto metodu doplnit se zabezpečením síťové nebo transportní vrstvy. Autentizaci pomocí certifikátů je možné využít jen v případě použití TLS. Tato metoda je vhodnější pokud všechna zařízení jsou po jednotnou správou a je možné automatizovat distribuci klientských certifikátů.

  Dále je na straně brokeru možné definovat pravidla pro autorizaci. Tato pravidla přiřazují klientů oprávnění pro provedení operací publish a subscribe nad příslušnými tématy [10].

1.2.2 CoAP

CoAP je otevřený přenosový protokol určený pro komunikaci síťových zařízení s velmi omezeným výkonem. Návrh vychází z RESTful (Representational State Transfer) principů, tudíž jeho použití je velmi vhodné v prostředích s již existujícím webovým rozhraním, do kterého se snadno integruje. [11]

1.2.2.1 Způsob komunikace

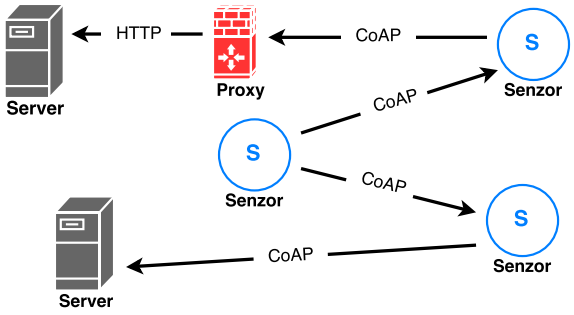
Komunikace je postavena nad protokolem UDP (User Datagram Protocol) a vychází z modelu request/response, který se využívá u protokolu HTTP (Hypertext Transfer Protocol). Oproti protokolu HTTP je zde omezen počet možných operací a komunikace probíhá asynchronně. Dle důležitosti odesílané zprávy je možné určit, zda se má zasílat potvrzení či nikoli. Protože veškerá komunikace probíhá nad protokolem UDP, tak přenášené zprávy obsahují položku Message ID, která dokáže ošetřit duplikaci přijatých dat.

Pro zajištění integrace s běžnými webovými službami a zachování nízkých nároků se v CoAP topologii velmi často vyskytují proxy servery. Tyto servery mohou plnit funkci běžných reverzních a dopředných proxy, mapování mezi CoAP a HTTP protokolem nebo vyvažování zátěže. Způsob komunikace a schéma možné způsoby propojení jsou na Obrázku 1.3. Senzor reprezentuje koncový prvek s omezeným výkonem.

Ve specifikaci protokolu CoAP jsou definovány následující operace:

- **GET**

  Metoda *GET* vrací aktuální stav požadovaného zdroje, který je identifikován pomocí URI (Uniform resource identifier). Tato operace je vždy bezpečná a idempotentní.



Obrázek 1.3: CoAP architektura

- **POST**

  POST požadavek obsahuje ve svém těle novou reprezentaci cílového zdroje a požaduje jeho zpracování. Funkce, která novou reprezentaci přijímá je definovaná na cílovém uzlu s příslušným URI. Výsledkem je vytvoření nového zdroje nebo aktualizace původního. Tato metoda není z pohledu zpracování bezpečná ani idempotentní.

- **PUT**

  Tato metoda specifikuje nový stav cílového zdroje, který specifikován použitým URI. Zdroj se buď vytvoří, nebo v případě jeho existence aktualizuje. Provedení operace není bezpečné, ale je idempotentní.

- **DELETE**

  Operace *DELETE* požaduje smazání zdroje s příslušným URI. Tato metoda není bezpečná, ale je idempotentní.

Vyhledání potřebného zdroje je v topologii protokolu CoAP možné provést pomocí znalosti jeho URI nebo odesláním multicastového požadavku na definovanou skupinu uzlů. Možnost vyhledávání cílových zdrojů je velmi důležitá zejména v M2M prostředích [11].

1.2.2.2 Bezpečnost

Samotný protokol nijak nedefinuje možnost autentizace a autorizace. V případě potřeby je nutné tyto mechanismy implementovat v aplikačním kódu.

Pro zajištění šifrování provozu nabízí CoAP následující režimy:

- **NoSec**

  Tento mód neobsahuje žádnou úroveň zabezpečení a veškeré zprávy jsou zasílány v otevřené podobě.

- **PreSharedKey**

  V tomto případě se naváže spojení pomocí protokolu DTLS (Datagram Transport Layer Security), které využívá symetrické šifrování. Předsdílený klíč musí být známý všem uzlům před zahájením komunikace.

- **RawPublicKey**

  Tento režim také navazuje zabezpečené spojení pomocí protokolu DTLS, ale místo symetrické šifry se používá asymetrická.

- **Certificate**

  Mód *Certificate* rozšiřuje *RawPublicKey* o přidání certifikátu.

Podobně jako u protokolu MQTT je i zde možné ochránit provoz na síťové vrstvě pomocí VPN. Při nasazení libovolného režimu zabezpečení je nutné počítat s většími nároky na výkon, které musí klient splňovat pro navazování a udržování spojení [11].

1.2.3 Z-Wave

Z-Wave je bezdrátový komunikační protokol určený pro senzorové sítě, který vysílá v subgigahercových pásmech ISM (Industrial, Scientific and Medical). Veškeré komunikační prvky jsou certifikovány aliancí Z-Wave, která zároveň poskytuje technickou dokumentaci a licence pro vývoj. V současné době existují certifikace Z-Wave a Z-Wave Plus. Z-Wave Plus zařízení obsahují nový chipset, který vylepšuje komunikační parametry sítě a zároveň je zpětně kompatibilní se staršími modely [12]. Otevřená implementace celého protokolu se nazývá OpenZWave [13], která je vyvíjena komunitou [14].

1.2.3.1 Způsob komunikace

V Z-Wave síti se může maximálně vyskytovat 232 uzlů, mezi kterými se vždy nachází jeden označený jako kontroler. Pro přidání libovolného zařízení do sítě musí být nejprve provedeno přímé spárování s kontrolerem. Během párovacího procesu nový prvek získá vlastní 8 bitový identifikátor (*Node ID*) a unikátní 32 bitový identifikátor sítě (*Home ID*), který má již od výroby kontroler uložen v nepřepisovatelné paměti. Následné zasílání zpráv už nemusí probíhat přímo mezi senzorem a kontrolerem, ale zprávy mohou být přeposílány sousedními prvky, které jsou trvale napájené, čímž se velmi zvyšuje možná rozloha sítě. Pro odebrání libovolného zařízení je opět nutné zajistit přímé spojení s kontrolerem a spustit odstraňovací proces [14].

V rámci specifikace Z-Wave [15] je definován mechanismus pro určení dostupných příkazů. Každý senzor obsahuje svou definici tříd funkcionalit (*Command Class*), které obsahují dostupné příkazy a formát odpovědi. Tyto definice jsou předány kontroleru během procesu párování. Příkladem může být třída *Binary Switch*, která obsahuje příkazy:

- *SET* – odesílá kontroler pro nastavení hodnoty
- *GET* – odesílá kontroler pro získání hodnoty
- *REPORT* – odesílá senzor jako odpověď na dotaz *GET*

Při posílání zpráv je vždy vyžadováno potvrzení. V případě neobdržení potvrzení se vysílání opakuje. Po třetím neúspěšném pokusu je požadavek zahozen.

1.2.3.2 Bezpečnost

Původní verze protokolu Z-Wave umožňovala volitelně využívat šifrování pomocí 128 bitového AES (Advanced Encryption Standard). Výměna symetrického klíče v této verzi probíhá během počátečního párování, kdy je šifrován pomocí výchozího klíče, který je uložen ve firmwaru. Vzhledem k tomuto postupu je dobré provádět počáteční párování na bezpečném místě, kde nemůže dojít k odposlechu [16].

V roce 2016 Z-Wave aliance vydala nový S2 (Security 2) framework, který vylepšuje bezpečnostní funkcionality a od roku 2017 je jeho použití povinné pro všechny nově certifikovaná zařízení.

S2 framework umožňuje zařízení rozdělit do následujících skupin s rozdílnými šifrovacími klíči:

- **Access Control** – nejdůvěryhodnější třída, která obsahuje bezpečnostní prvky jako jsou například zámky, které zároveň podporují autentizaci
- **Authenticated** – skupina určená pro běžné senzory, které podporují autentizaci
- **Unauthenticated** – třída pro ostatní prvky, které nepodporují autentizaci.

Autentizace probíhá pomocí PIN (Personal Identification Number) nebo QR (Quick Response) kódu. Výměna klíče je založena na algoritmu ECDH (Elliptic-curve Diffie–Hellman), který zajišťuje dostatečnou úroveň bezpečnosti během párovacího procesu [14].

1.2.4 BLE (Bluetooth Low Energy)

BLE je bezdrátový protokol určený pro senzorové sítě s důrazem na nízkou spotřebu. Byl představen v roce 2010 jako součást specifikace Bluetooth 4.0 a je nekompatibilní s původními verzemi. Za vývoj a údržbu protokolu je odpovědná skupina Bluetooth SIG (Special Interest Group). V současné době je nejnovější verzí Bluetooth 5 [14].

1.2.4.1 Způsob komunikace

BLE vysílá v bezlicenčním ISM pásmu na frekvencích od 2.4 GHz až do hodnoty 2.4835 GHz. Specifikace protokolu vychází z Bluetooth, ale zejména díky změnám parametrů v rádiové vrstvě jsou navzájem nekompatibilní. Do verze 4 umožňuje navázat pro koncová zařízení pouze jedno spojení, a proto vytváří hvězdicovou topologii s jedním centrálním prvkem. Od verze 5 je možné využívat více připojení a lze vytvořit flexibilnější mesh topologii [14].

Před zahájením komunikace mezi centrálním prvkem a senzory je nutné nejprve provést párování, které probíhá v následujících krocích:

- **Vysílání žádostí**

  Při spuštění párování začne koncový prvek na kanálech určených pro propagaci všesměrově vysílat žádosti o připojení, které obsahují název zařízení, jméno výrobce a podporované služby [17].

- **Přijímání žádostí**

  Pokud je centrální prvek přepnutý do párovacího režimu, tak naslouchá příchozím požadavkům, které zobrazuje uživateli. Po výběru správného zařízení se ukončí mód naslouchání a začne se navazovat spojení [17].

- **Inicializace připojení**

  Během této fáze se obě strany domlouvají na parametrech komunikace [17].

- **Komunikace**

  Po úspěšném navázání spojení je možné na základě dostupných služeb provádět zasílání zpráv [17].

1.2.4.2 Bezpečnost

BLE umožňuje volitelně používat šifrování pomocí AES, jehož šifrovací klíč je vytvořen během párování v části inicializace připojení. Velmi důležité je zabezpečit výměnu sdíleného klíče, která až do verze 4.1 není bezpečná, protože neumožňuje ochranu před odposloucháváním. Vylepšení přichází až od verze 4.2, ve které lze využít ECDH [14].

Vzhledem k různorodosti možných typů zařízení definuje BLE následující kategorie určující způsob výměny klíče:

- **Just Works**

  Nejjednodušší třída, která provádí výměnu automaticky a zároveň má nejmenší nároky na připojované zařízení, které nemusí podporovat autentizaci. Díky chybějící autentizaci je tato metoda zranitelná vůči MITM útokům [14].

- **Out of Band**

  V této kategorii jsou veškeré klíče vyměňovány odlišným komunikačním kanálem např. přes NFC (Near Field Communication). Celková bezpečnost této metody závisí na důvěryhodnosti použitého kanálu.

- **Passkey**

  Tato metoda vylepšuje *Just Works* o autentizaci, která spočívá v uživatelském zadání šestimístného kódu. Pro ochranu před odposlouchávání musí být použit algoritmus ECDH, který je dostupný až od verze 4.2 [14].

- **Numeric Comparison**

  Využití tohoto způsobu párování je možné pouze od verze 4.2. Dochází zde k rozšíření metody *Just Works* o jeden kontrolní krok, který slouží jako ochrana před MITM útokem. Po výměně klíčů každé zařízení vygeneruje šestimístný číselný kód, který následně zobrazí uživateli a čeká na jeho potvrzení [14].

1.3 Bezpečnostní slabiny

Na základě provedené analýzy aktuálně používaných protokolů v Sekci 1.2 budou v této kapitole popsány jejich bezpečnostní slabiny, které budou později využity při návrhu detekčních algoritmů. Jelikož je tato práce zaměřena na senzorové protokoly nekomunikující přes IP, budou v podkapitolách popsány slabiny protokolů Z-Wave a BLE.

1.3.1 Z-Wave

Hlavním problém jsou zařízení certifikovaná před březnem 2017, jelikož nepodporují S2 framework. Tyto prvky nemusí při své komunikaci využívat žádnou formu šifrování a síť se tak stává velmi zranitelnou. V minulosti již bylo provedeno několik útoků, které pomocí Scapy-Radio projektu [18] nebo knihovny OpenZWave byly schopny ovládat jednotlivé senzory.

Při využití šifrování je velmi zranitelná doba během párovacího procesu, jelikož je při výměně šifrovacích klíčů použit výchozí klíč uložený ve firmware zařízení. Zároveň se u jednoho typu zámku podařilo objevit implementační chybu [16], která umožňovala vnutit nový šifrovací klíč a převzít tak kontrolu nad senzorem.

Při využití S2 frameworku dosud nebyly objeveny žádné zranitelnosti, ovšem tento framework je poměrně nový a většina aktuálně používaných i nabízených zařízení byla certifikována ještě před jeho zavedením.

1.3.2 BLE

Velkou hrozbou jsou zařízení s verzí 4.1 nebo nižší, protože nepodporují žádnou ochranu před odposloucháváním a MITM útokům v době párování. Od verze 4.2 je již pro výměnu klíčů použit ECDH algoritmus, který zabraňuje možnému odposlouchávání, ale v případě použití párovací metody, které nepodporuje autentizaci není zajištěna ochrana před MITM útoky [14].

Dalším problémem jsou samotní výrobci, kteří často nevyužívají bezpečnostní funkce protokolu [19] nebo implementují vlastní způsoby zabezpečení na aplikační úrovni. Tímto dochází k nezabezpečení fáze párování a často se vyskytují i implementační chyby, které přinášení další zranitelnosti [17] a umožňují útočníkovi získat kontrolu nad celým provozem [14].

Nevýhodou je velmi dlouhá a komplikovaná specifikace protokolu, která vede k implementačním chybám samotného BLE [20]. Tím vzniká nebezpečí, že i u výrobce, který využívá všech bezpečnostních funkcionalit se mohou vyskytovat zranitelnosti díky chybné implementaci komunikačních vrstev protokolu [14].

1.4 Možnosti detekce

V běžných IP sítích se pro detekci hrozeb nejčastěji používají IDS (Intrusion Detection System) a IPS (Intrustion Prevention System) systémy. Tyto služby rozšiřují koncept klasického firewallu, který blokuje nebo povoluje síťový provoz na základě statických pravidel, o podrobnější sledování datových toků, které jsou zablokovány na základě nestandardního chování.

IDS/IPS může být reprezentováno samostatným hardwarovým zařízením nebo softwarovým programem, který může být dále rozšířen o monitorovací sondy, které se starají pouze o sběr dat a jejich odesílání pro následnou analýzu. Zároveň se může lišit i umístění v síti. Detekční systémy lze provozovat před hraničním směrovačem a detekovat tak kompletní příchozí a odchozí data nebo za hraničním směrovačem, což umožní vyhodnocovat jen vyfiltrovaný provoz. Případně je možné nasadit IDS/IPS přímo na koncové stanice, kde kromě síťových dat lze získávat i informace o běhu systému. Poslední způsob poskytuje nejvíce detailní možnost analýzy, protože se provádí na místě vzniku komunikace a případný útok je možné zastavit již při jeho začátku a zabránit tak případnému rozšíření. Nevýhodou takového nasazení je ztráta celkového pohledu na síť. Při zavedení IDS/IPS systému je z těchto důvodů dobré kombinovat způsoby nasazení a umožnit vzájemné sdílení detekovaných incidentů.

Při zaměření na zpracovávání datových toků lze detekční systémy rozdělit do dvou základních kategorií:

- **Detekce anomálií**

  Tato metoda je založena na statistickém modelování. Nejprve se vytvoří profil běžného chování sítě, který se následně porovnává s aktuálním provozem. Dle použité metody se může profil běžného provozu průběžně aktualizovat. Pokud měřené údaje síťového provozu překročí hodnotu stanoveného profilu nad definovaný limit, tak je detekován incident. Výhodou metody detekce anomálií je její uplatnění i na dosud neznámé útoky. Tento princip zároveň způsobuje větší míru falešných poplachů, a proto je nutné jejich pečlivější ověření. Příkladem detekčních metod je: strojové učení, časové řady nebo stavové automaty [21].

- **Detekce signatur**

  V případě použití této metody je využíváno signatur (profilů) předem známých útoků. Výhodou je, že díky popsaným signaturám je tato metoda poměrně přesná a detekuje málo falešných poplachů. Naopak nevýhodou je, že není možné detekovat nové druhy útoků, pro které není známý profil. Problémem také je, že signatury musí být uloženy na nějakém perzistentním úložišti, které je dostupné lokálně nebo vzdáleně [21].

IoT sítě k běžnému IP provozu přidávají senzorové protokoly, jejichž chování je také dobré monitorovat, protože jsou připojeny do počítačové sítě a mo-

hou být zneužity při útocích. Bohužel detekční metody nejsou pro tyto sítě moc rozšířené, a tím dochází ke zvýšení rizika a dopadu možných útoků. Pro určení incidentů lze využít stejných principů jako v IP sítích, ale liší se způsob získávání dat pro analýzu. Pro sběr informací lze využít následující přístupy:

- **Testbed**

  Tato metoda spočívá ve vytvoření specializovaného prostředí, ve kterém se nachází pouze testované a měřící zařízení. Cílem je ověřit, že nově připojovaný senzor splňuje veškeré bezpečnostní požadavky a neobsahuje žádné známé zranitelnosti. Měřící prvky reprezentují nástroje, které jsou schopné odposlouchávat komunikaci a využívají se také například při automatizovaných penetračních testech.

- **Externí sonda**

  Funkce sondy je stejná jako v IP sítích. Jedná se o samostatné zařízení umístěné v sítí, které umožňuje sledovat probíhající komunikaci a odesílat získané údaje ke zpracování. Výhodou je velké množství různých dat, které lze získat, ale zároveň komplikací je šifrovaný provoz a často i cena kvalitní sondy. Dalším využitím může být honeypot, kdy se sonda tváří jako zranitelný prvek a reportuje veškeré pokusy o útok.

- **Provozní statistiky**

  Posledním způsobem je sběr dat z příslušných rozhraní na IoT bráně. Tento postup nevyžaduje použití žádného dalšího zařízení, ale potřebuje, aby brána umožňovala získávat tyto statistiky. Statistiky nejsou tak podrobné jako u externí sondy, ale výhodou je, že získávání aktuálních dat o provozu je poměrně nenáročné.

Každá z předchozích metod používá jiný styl sběru dat. Z tohoto důvodu je výhodné při detekci možných útoků jejich kombinované nasazení. Například provozní statististiky jsou vhodné k vyhodnocení chování parametrů provozu, ale pro některé konkrétní útoky jako MITM může být vhodnější externí sonda, která má podrobnější přehled o probíhající komunikaci. Naopak metoda testbedu je užitečnější pro technologické společnosti, které mohou konkrétní zařízení otestovat ve specializovaném prostředí a vydávat pro ně doporučení určená koncovým uživatelům.

1.5 Existující řešení

V současnosti se veškerá dostupná řešení zaměřují na detekci útoků v IP protokolech. Mezi současnými IDS/IPS existují signatury pouze pro SCADA (Supervisory Control And Data Acquisition) protokoly. Router Turris Omnia umožňuje nasadit systém Suricata, pro který nabízí rozšíření PaKon [22]. Toto rozšíření zpracovává data ze Suricaty, které následně ukládá v přehledné formě. Díky tomu uživatel získá podrobný přehled o stavu provozu. Získaná data může dále využívat například k vylepšení stávajících bezpečnostních pravidel. Nevýhodou tohoto řešení jsou vyšší hardwarové požadavky, a proto ho nelze provozovat na branách s omezenými prostředky. Další komplikací může být centralizovaná architektura a zaměření pouze na IP provoz.

Na základě provedené rešerše nebylo nalezeno žádné řešení, které umožňuje vyhodnocovat provoz aktuálně používaných IoT protokolů s ohledem na možné omezení hardwarových prostředků.

1.6 Analýza požadavků

Podstatnou částí návrhu výsledného řešení je přesné určení požadavků, které se dělí na funkční a nefunkční. Funkční požadavky specifikují funkcionality kladené přímo na vznikající program, zatímco nefunkční spíše určují omezení vlastností systému a architekturu návrhu. V následující kapitole budou popsány nároky na vznikají detekční nástroj, které vycházejí z obsahu zadání této práce a provedené analýzy.

1.6.1 Funkční požadavky

Tato sekce popisuje funkční požadavky, které jsou od detekčního nástroje očekávány.

- **Sběr informací o provozu**

  Program bude umožňovat sběr dat na IoT bráně o aktuálním provozu z dostupných komunikačních rozhraní. Zároveň bude možné získané informace přeposílat k další analýze.

- **Detekce anomálií**

  Vytvořený detekční algoritmus bude schopen na základě získaných dat odhalit definované anomálie, které reprezentují neočekávané změny v síti.

- **Konfigurace způsobu detekce**

  Detekční modul bude umožňovat nastavení parametrů pro jednotlivé zpracovávané položky pomocí konfiguračního souboru. Tyto parametry budou následně sloužit jako vstup pro vytvořený detektor.

- **Zpracování získaných informací**

  Kromě analýzy dat a hlášení nalezených incidentů bude také možné zpracovaná data pravidelně exportovat do dalších rozšiřujících modulů.

1.6.2 Nefunkční požadavky

V této sekci jsou obsaženy nefunkční požadavky, které jsou kladeny na výsledný nástroj.

- **Rozšiřitelnost**

  Architektura celého řešení bude navržena tak, aby bylo možné rozšíření o nové způsoby detekce anomálií a další typy provozních dat.

- **Flexibilita nasazení**

  Návrh způsobu detekce umožní fyzicky oddělit komponentu zajišťující sběr dat a komponentu vyhodnocující provoz. Díky tomu bude možné

provádět analýzu i na IoT branách s velmi omezeným prostředky, protože tyto brány budou sloužit jako sondy, které budou posílat data do externího detektoru s dostatečnými výkonem k provedení analýzy.

- **Kompatibilita a vývoj**

  Pro zachování kompatibility a usnadnění vývoje bude použit framework NEMEA [23] (Network Measurements Analysis), který umožní snadné propojení jednotlivých detekčních modulů.

- **Operační systém**

  Výsledné řešení bude implementováno a otestováno na operačním systému Ubuntu 16.04, na kterém bude nasazena IoT brána BeeeOn a framework NEMEA. Zároveň budou při návrhu a implementaci vybírány takové technologie, aby bylo možné vytvořený program spustit pod distribucí OpenWrt, která je velmi rozšířena mezi síťovými prvky.

1.7 Zvolené řešení

Obsahem této kapitoly je popis zvoleného řešení, které bylo na základě provedené analýzy určeno pro realizaci výsledného nástroje.

Vytvořený detekční algoritmus bude zaměřen na senzorové IoT protokoly, protože v současné době neexistuje řešení, které by to umožňovalo. Pro analýzu budou použity protokoly Z-Wave a BLE, které jsou v dnešních sítích velmi rozšířené. Algoritmus bude umístěn přímo na IoT bráně, která je ideálním místem, protože se nachází velmi blízko koncových zařízení a má dostatečný výkon k vyhodnocování provozu. Konkrétně bude použita brána BeeeOn, protože v současné době jako jediná umožňuje sběr provozních dat o senzorových protokolech.

Pro účely detekce budou sbírány jen informace dostupné z lokálních rozhraní brány. Je velice pravděpodobné, že z hlediska reálného nasazení bude tato varianta nejčastější, protože nevyžaduje žádná dodatečná monitorovací zařízení. Zároveň bude umožněno rozšíření i pro další způsoby získávání dat.

Vyhodnocování dat bude probíhat metodou detekce anomálií, která je vhodná pro statistickou povahu dat, má menší nároky na množství dostupných prostředků a umožňuje rozpoznat i neznámé útoky. Metoda bude realizovaná pomocí časových řad, jejichž parametry bude možné upravit dle potřeby.

Jako programovací jazyk bude použit C++, protože podporuje objektový přístup a úspornou implementaci na paměť i procesorový čas. Zároveň tento jazyk je použit i v bráně BeeeOn, tudíž bude usnadněna integrace. Pro zajištění flexibility a možnosti dalšího rozšiřování bude využito systému NEMEA, který také podporuje jazyk C++.

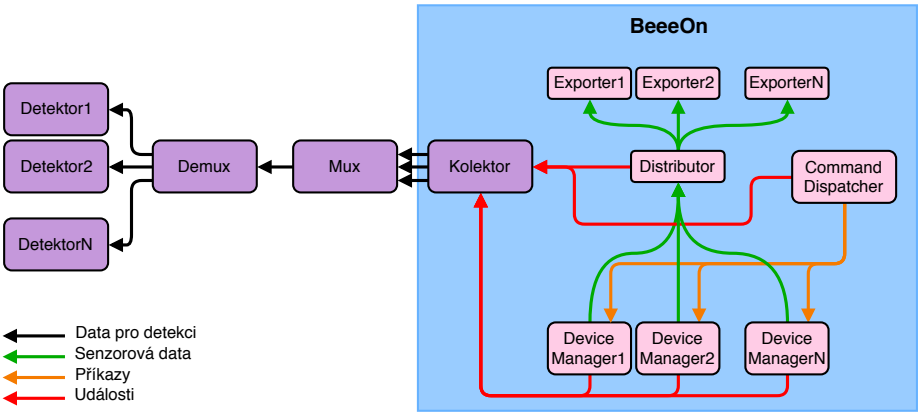
Kapitola 2

Návrh

Kapitola se zabývá návrhem způsobu detekce anomálií v IoT sítích, který s pomocí NEMEA systému rozšiřuje bránu BeeeOn. Nejprve je popsána celková architektura a možné způsoby nasazení. Dále následuje popis jednotlivých komponent detekčního systému. V závěru jsou identifikovány možné scénáře útoků.

2.1 Architektura

Cílem této práce je vytvoření programu, který bude schopen detekovat anomální provoz v IoT sítích. Vzhledem k hardwarovým omezením, která mohou na IoT branách nastat, je architektura navržena tak, aby měla co nejmenší nároky na dostupné prostředky. Schéma nasazení detekčního systému se nachází na Obrázku 2.1



Obrázek 2.1: Architektura detekčního systému

Implementace BeeeOn brány obsahuje pro zpracování senzorových dat následující komponenty:

- **DeviceManager** – komponenta definovaná pro každý senzorový protokol, která implementuje veškerou komunikaci a zpracování dat
- **Distributor** – přijímá data od *DeviceManageru*, která následně předává příslušnému *Exporteru*
- **Exporter** – implementuje protokol, kterým jsou data odesílány z brány
- **CommandDispatcher** – přijímá uživatelské příkazy a distribuuje je cílovým komponentám

Každá z komponent v BeeeOn bráně navíc využívá návrhový vzor *Observer*, pomocí kterého jsou definovány události poskytující informace o každé komponentě. Tímto způsobem bude vytvořený kolektor získávat data o provozu, která převede do formátu UniRec (Unified Record) a pomocí systému NEMEA je zpracuje a nebo odešle k analýze.

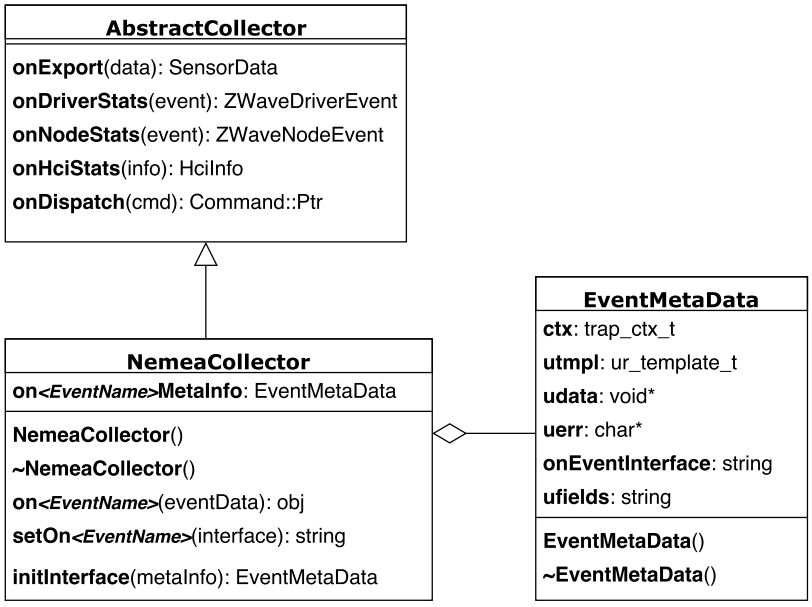
Exportované informace o provozu bude přijímat detektor, který následně provede jejich zpracování a vyhodnocení stavu. Všechny vytvořené komponenty určené pro detekci budou používat rozhraní NEMEA. Díky tomu bude možné komponenty flexibilně provozovat na jednom nebo více různých zařízeních. Oddělené nasazení je důležité zejména pro brány s omezenými prostředky, které mohou provádět pouze export dat a o vyhodnocení se bude starat odlišné zařízení s dostatečným výkonem. V případě provozování více bran lze brány používat jako exportéry a veškerá získaná data zpracovávat centrálně.

Kolektor pro každý typ události vytvoří samostatné výstupní komunikační rozhraní, kterých při reálném provozu může být velké množství. Z tohoto důvodu bude vytvořen modul *Mux*, který všechna výstupní rozhraní spojí do jednoho společného. Pro zpětné rozdělení na jednotlivé typy událostí bude sloužit modul *Demux*. Tyto moduly budou velmi užitečné zejména v případě, kdy kolektor a detektor budou na rozdílných síťových prvcích, protože údaje o provozu budou přenášeny přes počítačovou síť a bude vyžadováno zabezpečení všech odchozích komunikačních rozhraní. S využitím *Mux* a *Demux* modulů bude stačit zabezpečit pouze jedno sjednocené rozhraní.

Výhodou návrhu řešení je flexibilita a modularita. Každá komponenta vždy samostatně pokrývá pouze jednu část detekčního systému a díky NEMEA se každá z nich může nacházet na různých síťových prvcích. Zároveň v případě potřeby lze vytvořit další moduly, které budou rozšiřovat stávající funkcionalitu. Podrobnější popis návrhu nově vytvořených komponent detekčního systému se nachází v následujících sekcích.

2.2 Kolektor

Úkolem kolektoru bude sběr dostupných dat a jejich odeslání k následné analýze. Informace o provozu budou sbírány z jednotlivých komponent BeeeOn brány, které jsou zpřístupňovány pomocí návrhového vzoru *Observer*. Z tohoto důvodu bude kolektor vždy přímou součástí BeeeOn brány. Návrh struktury kolektoru je na Obrázku 2.2



Obrázek 2.2: Návrh kolektoru provozních dat

AbstractCollector, je třída vytvořená v projektu BeeeOn, které shromažďuje všechny dostupné typy událostí. V současné době jsou k dispozici následující události:

- **onExport** – reprezentuje hodnoty dat přicházejících ze senzorů, která jsou odeslána vždy po jejich příjmu
- **onDriverStats**: v pravidelných intervalech generuje statistiky o Z-Wave síti, které jsou dostupné na komunikačním rozhraní
- **onNodeStats** – periodicky získává informace o jednotlivých prvcích umístěných uvnitř jedné Z-Wave sítě, které jsou dostupné pomocí knihovny OpenZWave
- **onHciStats** – v definovaných intervalech získává z komunikačního rozhraní pro BLE statistiky o provozu sítě
- **onDispatch** – umožňuje získávat zadané uživatelské příkazy

Vytvořený kolektor je reprezentován třídou *NemeaCollector*, která je potomkem třídy *AbstractCollector* a bude implementovat všechny její členské funkce pro zachytávání událostí. Pro každou členskou funkci bude vytvořen setter, který při spuštění brány provede počáteční nastavení parametrů pro obsluhu dané události. Vstupním parametrem bude řetězec určující název výstupního rozhraní, který bude specifikovaný v konfiguračním souboru BeeeOn brány. Po nastavení počátečních parametrů jako je název výstupního rozhraní a seznam UniRec polí se zavolá členská funkce *initInterface()*, jejímž cílem bude inicializace veškerých struktur vyžadovaných NEMEA frameworkem. Jako vstupní parametr bude přijímán objekt *EventMetaData* udržující veškeré informace potřebné k odesílání získaných dat.

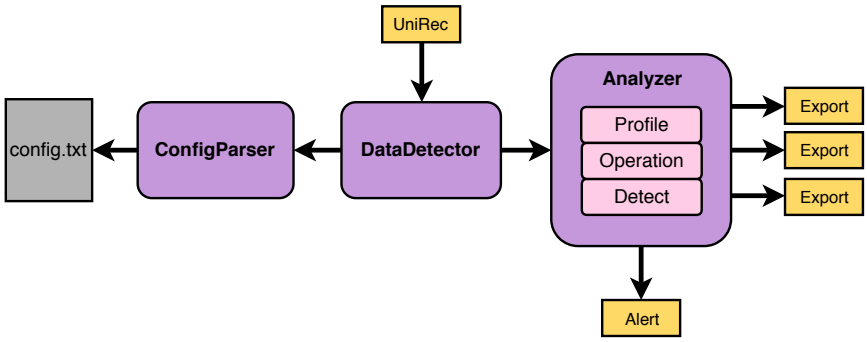
Instance objektu *EventMetaData* bude vytvořena jako členská proměnná třídy *NemeaCollector* pro každou členskou funkci události. Díky tomu bude možné každou událost zpracovávat nezávisle dle definovaných parametrů.

Definované konstruktory a destruktory budou sloužit jen pro vytvoření a destrukci potřebných struktur. Tímto bude zajištěn programovací idiom RAII (Resource Acquisition Is Initialization).

Veškeré odesílané údaje budou rozšířeny o časovou značku a číselný identifikátor, aby mohl detektor přesně určit původ případné anomálie.

2.3 Detektor

Detektor bude hlavní komponentou celého řešení, protože bude vyhodnocovat získaná data. Vzhledem k zadaným požadavkům musí být návrh proveden tak, aby bylo umožněno budoucí rozšiřování. Návrh architektury je zobrazen na Obrázku 2.3



Obrázek 2.3: Architektura detektoru

Komponenta *DataDetector* bude mít jedno vstupní rozhraní, kterým bude přijímat data z kolektoru. V případě potřeby zpracování více událostí z kolektoru lze využít již existující NEMEA modul *Merger*, který dokáže sloučit více vstupů do jednoho konsolidovaného výstupu.

Dále se v rámci třídy *DataDetector* bude volat *ConfigParser*, který bude mít na starosti zpracování konfiguračního souboru. Konfigurační soubor bude obsahovat seznam parametrů pro každé UniRec pole, na základě kterých bude vytvořena instance třídy *Analyzer* pro analýzu dat.

Po provedení inicializace všech modulů bude *DataDetector* pouze přijímat data z kolektoru a předávat je ke zpracování komponentě *Analyzer*. Tato komponenta bude obsahovat tři hlavní části:

- **Profile**

  Bude obsahovat členské funkce, které se budou starat o připravení profilu sítě. Vytvořený profil bude složen z: průměru, klouzavého průměru, klouzavého rozptylu a klouzavého mediánu. Vysvětlení jejich významu se nachází v Sekci 3.4. Tyto hodnoty budou následně použity při porovnávání s aktuálním provozem v rámci detekčních metod. V případě potřeby bude možné profil rozšířit o další hodnoty.

- **Operation**

  Cílem této části bude udržování veškerých použitých struktur. Bude se jednat o struktury pro konfiguraci, zpracovávaná data a odesílání získaných výsledků. Odesílat lze informaci o nalezené hrozbě, pro kterou

se používá jedno společné rozhraní, nebo pravidelně exportovat definované části aktuálního profilu sítě pro účely dalšího zpracování. Obsahem zprávy s nalezeným incidentem bude: ID zdroje dat, časové razítko, hodnota anomálie, hodnota položky vzorového profilu, název položky vzorového profilu, název UniRec klíče a popis anomálie. V rámci pravidelného exportu se budou odesílat jen definované číselné položky z aktuálního profilu.

- **Detect**

  Bude sdružovat všechny členské funkce určené pro detekci anomálií v získaných datech. Každá členská funkce bude reprezentovat jedno kritérium a bude možné budoucí rozšíření o další potřebné metody. V rámci vytvořeného řešení budou dostupná kritéria pro detekci: neočekávaného růstu, překročení vymezených hodnot a porušení periodicity. Popis jednotlivých metod se nachází v Sekci 3.5.

Díky konfiguraci organizované po jednotlivých UniRec polích bude možné nastavit chování výše popsaných částí nezávisle pro každé pole.

2.3.1 Konfigurace modulu

Jedním z funkčních požadavků je možnost konfigurace způsobu detekce, která se tak bude moci přizpůsobit cílovým podmínkám. Každá instance detektoru bude mít jeden konfigurační soubor, který bude popisovat jednotlivá UniRec pole určená pro analýzu.

Konfigurace bude v textové podobě ve formátu klíč a seznam hodnot. Textová podoba je velmi vhodná pro fázi ladění a vývoje nástroje, protože jsou možné manuální úpravy. Cílem ovšem je generovat konfiguraci pomocí externího nástroje, který by mohl být součástí řídící vrstvy IoT brány.

Klíčem bude vždy název UniRec pole, který se bude nacházet v příchozích datech. Za tímto klíčem se může nacházet volitelně číselný identifikátor senzoru, který se musí shodovat s identifikátorem odesílaným z kolektoru. Důsledkem toho bude možnost detekční parametry definovat i pro jednotlivá zařízení pokud se jich pod jedním UniRec názvem bude nacházet více. V případě neuvedení identifikátoru se jako výchozí hodnota použije číslo nula. Takto definovaná hlavička bude oddělena dvojtečkou, za kterou bude sekvence parametrů oddělených středníkem. Každý nový záznam s popisem přicházejících dat se bude nacházet na samostatném řádku. Řádky prázdné a začínající znakem # budou považovány za komentář.

Definované parametry budou rozděleny do následujících hlavních skupin:

- **General**

  Tato skupina bude obsahovat obecné volby pro definování časové řady. Pomocí číselných parametrů bude možné určit velikost řady, délku učící

fáze, během které bude vytvořen profil sítě a její délka musí být stejná nebo delší než velikost časové řady, a rozsah ignorovací části, která bude vhodná k odfiltrování počáteční výměny zpráv při navazování spojení.

Dalším parametrem bude způsob ukládání dat do časové řady. K dispozici budou dva režimy. Prvním z nich bude *simple*, který přijatou hodnotu pouze uloží. Druhý bude *delta*, který bude vkládat rozdíly dvou po sobě jdoucích hodnot. Toto bude vhodné zejména u časových značek nebo počtu odeslaných zpráv, protože bude vhodnější udržovat jednotlivé přírůstky než pouhé absolutní hodnoty.

Poslední volbou budou číselné hodnoty určující rozmezí mezi pravidelnými kontrolami přijetí a změny dat a číselný parametr definující periodu pro exportování vybraných parametrů. Nastavení těchto položek bude volitelné. Jejich nepoužití bude označeno znakem -.

- **Profile**

  Zápis této skupiny bude identifikovaný klíčovým slovem *profile*, který má uvnitř kulatých závorek vyjmenované jednotlivé položky, které budou součástí vytvořeného profilu. Ve vytvořeném řešení budou k dispozici následující volby: *average* (průměr), *moving_variance* (klouzavý rozptyl), *moving_median* (klouzavý medián) a *moving_average* (klouzavý průměr). Popis významu jednotlivých částí je umístěn v Sekci 3.4. Součástí profilu bude vždy alespoň jedna položka. Použité části a jejich počet bude vždy záviset na uživateli.

- **Profile Values**

  Pro každou položku definovanou v rámci profilu musí být uveden seznam jejich detekčních parametrů. Zápis bude probíhat pomocí klíčového slova z profilu, které bude mít jednotlivé hodnoty specifikované uvnitř kulatých závorek. Zde bude možné určit minimální nebo maximální *soft* a *hard* limity, kterých daná položka aktuálního profilu může dosahovat. Pro *soft* limit bude dále možné určit povolenou délku, po kterou může být nastavená hranice překročena (*grace period*).

  Dalšími parametry budou číselné hodnoty určující meze pro minimální a maximální velikost změny sledované položky ve srovnání s vzorovým profilem. Všechny ostatní argumenty nebudou mít žádný význam pro konfiguraci, ale budou sloužit jako proměnné pro ukládání dočasných hodnot.

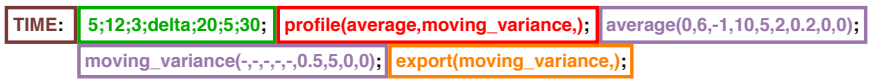
  Zadání veškerých hodnot nebude nutné a tudíž bude záležet na uživateli, které způsoby detekce bude chtít využít. V případě nedefinování některého parametru bude zapsán znak -. Na základě nastavených hodnot se budou volat příslušné detekční metody.

- **Export**

  Poslední nastavitelnou možností bude export vybraných dat z profilu. Zápis bude probíhat pomocí klíčového slova *export*, který v kulatých závorkách bude mít seznam položek z profilu. Zároveň v případě využití této možnosti bude nutné ve skupině *general* určit časovou periodu odesílání. Názvy výstupních *libtrap* rozhraní budou odvozené od klíčů, které se nacházejí před dvojtečkou v konfiguraci.

Příklad záznamu s konfigurací je na Obrázku 2.4. Jednotlivé barvy označují výše definované skupiny.

Nastavení formou souboru bude jedninou volbou pro přizpůsobení detekce. Dále při spuštění programu bude očekáván povinný přepínač *-i*, který je vyžadován knihovnou *libtrap* a bude definovat název vstupního rozhraní pro příchozí data a název výstupního rozhraní pro detekované incidenty. Jako nepovinný přepínač bude možná zadat *-v*, jehož výsledkem bude zapnutí ladících výpisů. Tato volba bude umožňovat až tři úrovně podrobnosti výpisů. Tyto úrovně budou určeny počtem znaků *v*.



Obrázek 2.4: Popis konfiguračních parametrů pro jeden záznam

2.3.2 Použité struktury

Velmi důležitou částí návrhu bude určení formátu použitých struktur, které budou udržovat veškeré údaje nutné pro analýzu. V rámci detektoru bude potřeba udržovat informace pro následující položky:

- **Konfigurační parametry**

  Tato struktura bude hierarchicky ukládat veškeré hodnoty z konfiguračního souboru a také údaje nutné k provedení analýzy. V první úrovni se budou nacházet názvy jednotlivých UniRec polí, které budou fungovat jako klíče. V druhé vrstvě bude seznam identifikátorů odlišující senzory skryté pod stejným klíčem. Třetí úroveň bude vždy obsahovat stejnou sekvenci podklíčů, které budou určovat typ uchovávaných dat. Bude se jednat o jednotlivé skupiny z konfiguračního souboru popsané v Sekci 2.3.1), které budou označeny stejnojmennými identifikátory. Hodnotou těchto skupin budou specifikované parametry detekce dle zapsané konfigurace. Dále zde budou podklíče s názvem *metaProfile* a *metaData*, které budou mít stejné položky. Tyto položky budou obsahovat vypočtené údaje z příslušné časové řady a zároveň budou umožňovat ukládání dat,

která budou nutná k provedení výpočtu a vkládání nových prvků. Rozdíl mezi *metaProfile* a *metaData* bude typ profilu, pro který budou určeny. Podklíč *metaProfile* bude uchovávat údaje pro vzorový profil, který bude vytvořen během učící fáze a bude sloužit pro určení anomálie. Zatímco hodnoty v *metaData* budou vztaženy k aktuálnímu profilu sítě, který se mění s každým nový prvkem v časové řadě.

- **Senzorová data**

  Cílem této struktury bude hierarchicky ukládat přicházející data z kolektoru do časových řad. V první úrovni se jako v předchozím případě budou nacházet názvy jednotlivých UniRec polí. Protože jedno UniRec pole může reprezentovat data z více různých senzorů nebo bran, tak se na další úrovni nachází identifikátor zdroje dat. Předpokládá se, že každá použitá brána a senzor bude mít v rámci jedné instance detektoru unikátní identifikátory. Na třetí úrovni se bude vyskytovat definovaná časová řada, která uloží data podle zadané konfigurace.

- **Nastavení pro export**

  Parametry pro export budou uchovány ve struktuře s formátem klíč a hodnota. Klíčem bude pořadové číslo výstupního komunikačního rozhraní, které bude odpovídat konkrétnímu UniRec poli. Hodnotou bude sekvence názvů určující jednotlivé části profilu, které se budou exportovat. Tyto názvy budou definované v konfiguračním souboru.

2.3.3 Způsob analýzy

Veškerá analýza dat bude prováděna pomocí časových řad, které jsou vhodné pro statistický typ dat a metodu detekce anomálií. Zároveň parametry řad budou nastavitelné, aby bylo možné se přizpůsobit pro danou datovou položku, ale i cílovou bránu, na které může být spuštěný detektor. Jednotlivé fáze zpracování dat jsou na Obrázku 2.5 a jejich popis se nachází v následujících sekcích:

Způsob zpracování dat lze rozdělit do následujících fází:

- **Inicializace**

  V rámci této sekce budou načteny všechny konfigurační položky a inicializovány veškeré struktury nutné pro běh celého řešení. Následně bude program očekávat příchozí data na jediném vstupním rozhraní.

- **Učení**

  Po prvotním přijetí hodnot z kolektoru bude probíhat stanovení vzorového profilu sítě pro položky, které budou popsány v konfiguračním souboru. V rámci této periody se do časové řady budou vkládat přicházející data a s každým novým prvkem bude prováděn výpočet všech

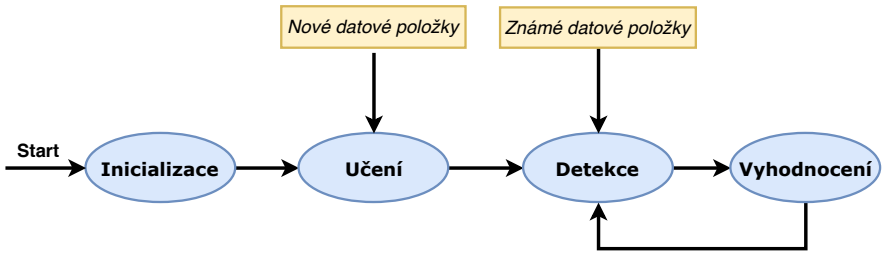
částí profilu, které budou určeny konfiguračním souborem. Pokud bude délka učící fáze delší než velikost časové řady, tak budou všechny prvky posunuty, a tím nejstarší z nich uvolní místo pro nový.

- **Detekce**

  Jakmile bude stanoven vzorový profil sítě, tak se zahájí analýza provozu pro určení anomálií. S každým nový příchozím prvkem bude z příslušné časové řady vypočítáván aktuální profil sítě, který budou používat detekční metody. Dostupné metody jsou popsány v Sekci 3.5.

- **Vyhodnocení**

  Výsledky detekčních metod budou sloužit k určení stavu provozu. Pokud budou data vyhodnocena jako bezpečná, tak nebude provedena žádná akce a přejde se k obsluze další přijaté hodnoty z kolektoru. V případě nalezení anomálie se vytvoří zpráva s informacemi popisující událost, která bude odeslána výstupním komunikačním rozhraním určeném pro detekované hrozby.



Obrázek 2.5: Diagram analýzy dat

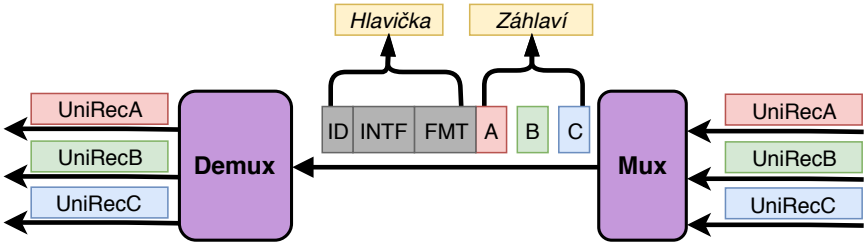
2.4 Multiplexor a demultiplexor

Hlavním cílem těchto dvou modulů bude ulehčit nasazení analýzy dat v případě fyzicky odděleného kolektoru a detektoru. Bez použití těchto modulů bude nutné na kolektoru zabezpečit přenos dat z každého výstupního komunikačního rozhraní, což zvýší nároky na výkon brány a zároveň bude vyžadováno více konfigurace. Díky multiplexoru bude možné spojit několik různých *libtrap* rozhraní do jednoho, které lze demultiplexorem zase zpětně rozdělit na cílovém zařízení. Tímto bude pro export dat z brány nutné zabezpečit pouze jedno rozhraní.

Multiplexor bude vícevláknový NEMEA modul, kde každé vlákno bude naslouchat na specifikovaném vstupním rozhraní. Po přijetí nového formátu dat se nejprve odešle *hello* zpráva, která bude obsahovat popis nového formátu. V případě přijetí zprávy se stejným formátem bude tato zpráva rovnou přeposlána. Oba typy zpráv budou mít stejnou hlavičku s položkami: identifikátor zprávy, číslo vstupního rozhraní a typ formátu. Ostatní data nutná pro přenos budou zapouzdřena v záhlaví. Díky stejnému formátu zprávy bude možné veškerá data zasílat jedním společným rozhraním.

Demultiplexor bude jednovláknový modul, který bude přijímat na jednom vstupním rozhraní zprávy od multiplexoru. Po přijetí dat provede jejich zpracování a na základě identifikátoru provede příslušnou akci. V případě typu *hello* se upraví nový formát pro specifikované komunikační rozhraní. V druhém případě se jen přepošlou přijatá data příslušným výstupem.

Způsob komunikace mezi moduly je zachycen na Obrázku 2.6. Vstupem do modulu *Mux* jsou tři různé UniRec šablony, které jsou postupně zapouzdřeny do společného formátu a odeslány společným komunikačním rozhraním. Následně modul *Demux* provede zpětné rozbalení zpráv na odpovídající komunikační rozhraní.



Obrázek 2.6: Diagram komunikace modulů *Mux* a *Demux*

2.5 Scénáře útoků

Na základě vypracované analýzy a provedených experimentů s komunikačními protokoly Z-Wave a BLE, byly navrženy možné scénáře útoků, které se mohou objevit na senzorové vrstvě. V popsaných scénářích se bude vyskytovat jen nezbytně nutná množina informací, kterou bude nutné sledovat. Popis všech dostupných dat se nachází v příloze D.

S.1 Periodicita dat

Velké množství koncových prvků v senzorové vrstvě se chová periodicky. Kvůli úspoře energie jsou senzory často v režimu spánku a jen v určitých časových okamžicích se probudí, aby odeslaly potřebné údaje. Pravidelně pracuje i BeeeOn brána, která statistiky o jednotlivých komunikačních protokolech exportuje ve stálých časových intervalech.

Tento scénář se bude soustředit na analýzu času příchozích dat, aby určil případné nestandardní chování. Díky tomu bude možné odhalit vzniklé problémy na bráně, které vedou k nepravidelnému odesílání informací o provozu. V případě, že bude použito čidlo s periodickým chodem, bude možné zjistit změnu jeho chování nebo jeho úplné odpojení.

Určení času příchozích dat bude snadné, protože každý přijatý záznam bude obsahovat datovou položku s časovou známkou.

S.2 Množství přenášených zpráv

Velkou hrozbou, kterou představuje nástup IoT jsou nejrůznější varianty DoS útoků. Konkrétně nejobávanějším typem je pravděpodobně zahlcení cílové komunikační linky, protože v síti bude dostupné obrovské množství prvků, které útočník může v případě napadení využít. Zároveň nejlepším místem, kde tyto útoky zastavit, je zdroj jejich vzniku. Tento scénář může také identifikovat připojení dalšího zařízení.

Cílem bude identifikovat neobvyklý nárůst nebo pokles přenášených dat v porovnání s naučeným vzorovým profilem sítě.

Výhodou jsou Z-Wave sítě, protože nabízí statistiky pro celou síť i pro jednotlivé senzory. Položka *SOAFCount* určuje celkový počet detekovaných zpráv ve vytvořené topologii. Při analýze jednotlivých čidel bude možné sledovat čítače *receivedCount* a *sentCount*. První čítač reprezentuje počet zpráv od konkrétního čidla, který byl přijat na bráně. Druhá hodnota určuje počet odeslaných zpráv danému senzoru.

BLE nabízí pouze celkové informace, kterých navíc není tolik. V rámci této detekce bude vhodné použít položky *rxBytes* a *txBytes*, které odpovídají počtu přijatých a odeslaných bajtů.

S.3 Limity senzorových hodnot

Tento případ užití se zaměřuje přímo na získávané hodnoty z čidel. Pro tyto hodnoty se většinou dá určit jejich minimální a maximální hodnota s rychlostí růstu. Pokud se vývoj hodnot nebude shodovat s očekáváním, může to znamenat poruchu čidla nebo neočekávanou událost, jejíž příčinou může být i útok.

Možnost této detekce bude záviset na události *onExport*, která exportuje všechna získaná data nezávisle na použitém protokolu.

S.4 Kvalita přenosového kanálu

Zejména u bezdrátových komunikačních protokolů je velmi důležitá kvalita přenosového kanálu, který může být případným útočníkem zarušen. V situaci, že tento typ útoku nastane, budou detekční metody očekávat zhoršení statistik komunikace.

Knihovna OpenZWave nabízí pro každé čidlo informace o hodnotě RTT (Round-Trip Time). Z pohledu celé sítě bude možné zjistit počet zahozených zpráv (položka *dropped*), počet opakovaných odeslání zpráv (položka *retries*) nebo počet zpráv z poškozeným kontrolním součtem (položka *badChecksum*).

V případě BLE lze využít položky *rxErrors* a *txErrors*, které reprezentují počet přijatých a odeslaných chyb.

S.5 Konektivita

Tento případ navazuje na předchozí **S.4**. Kvalita připojení může být zhoršena natolik, že dojde ke ztrátě konektivity. Případně spojení může být ztraceno pokud je senzor odcizen.

Analýza může probíhat na základě parametrů specifikovaných v předchozím scénáři. Nebo lze ztrátu konektivity se senzorem identifikovat tím, že se provozní statistiky nezměnily vůbec nebo menším způsobem než bylo obvyklé. Pro tento typ detekce mohou být využity stejné položky jako v případu užití **S.2**.

Všechny definované situace budou zároveň sloužit k otestování detekčního systému, který by je měl všechny odhalit.

Kapitola 3

Realizace

Obsahem kapitoly je popis realizace nejzajímavějších částí vytvořeného řešení. Pro zajištění efektivity a snadné přenositelnosti byla veškeré implementace provedena v jazyce C++, ve kterém je zároveň napsána BeeeOn brána. Detekční systém přináší pouze jednu novou závislost a tou je NEMEA framework. Ostatní použité knihovny se shodují s již použitými v BeeeOn bráně.

3.1 Integrace kolektoru

Vytvořený kolektor provozních dat je přímou součástí již existujícího projektu BeeeOn, a proto jej bylo nutné integrovat do spouštěcího procesu brány. Pro zajištění společné kompilace byly do souborů *CMakeLists.txt* přidány cesty ke zdrojovým souborům kolektoru a závislosti na knihovny z NEMEA frameworku. K určení způsobu spouštění jednotlivých komponent používá BeeeOn soubor *factory.xml*. V tomto souboru byla vytvořena nová komponenta s názvem *collector*, která byla následně přidána pod označením *listener* do specifikace ostatních komponent. Toto nastavení umožňuje přijímání definovaných událostí v rámci návrhového vzoru *Observer*. Samotná komponenta *collector* obsahuje ve svém popisu seznam jmen událostí, kterým přiřazuje pojmenování výstupního *libtrap* rozhraní. Formát názvu výstupu je vždy *event-<názevUdálosti>*.

Vytvořené názvy událostí jsou při spouštění programu pomocí C++ reflexe předány třídě kolektoru *NemeaCollector*. Tato třída používá již vytvořené makro *BEEEON_OBJECT_TEXT*, které pro definované typy událostí volá přiřazené členské funkce. Každá událost má členskou funkci s názvem ve formátu *set<názevUdálosti>()* přijímající jeden vstupní parametr typu string s názvem výstupního komunikačního rozhraní. V rámci volání se do instance třídy *EventMetaData* nastaví specifické hodnoty členských atributů o daném typu události. Následuje volání členské funkce *initInterface()*, která přijímá vytvořenou instanci třídy *EventMetaData* a jednotně inicializuje všechny potřebné struktury pro odesílání dat.

Po úspěšném nastavení všech částí se už jen v rámci návrhového vzoru *Observer* volají členské funkce událostí, které jsou definované ve třídě *AbstractCollector* a implementované ve třídě *NemeaCollector*. Získané informace z brány jsou vkládány do UniRec zprávy a odeslány výstupním rozhraním.

3.2 Mux a Demux

Modul *Mux* očekává na vstupu přepínač *-i*, který ve formě řetězce určuje dostupná komunikační rozhraní zajištěná knihovnou *libtrap*. Poslední identifikátor v zadaném řetězci označuje jméno výstupu. Druhým parametrem je *-n*, který odpovídá počtu vstupních komunikačních rozhraní. Při spuštění se pro každý vstup vytvoří samostatné vlákno pomocí knihovny OpenMP. Jelikož je veškerý provoz odesílaný jedním společným výstupem, bylo nutné funkci *trap_ctx_send()* vložit do kritické sekce, protože pracuje se sdílenými strukturami pro všechny vlákna.

Společná spojení mezi vytvořenými moduly *Mux* a *Demux* používá nastavený typ *TRAP_FMT_RAW*, který umožňuje posílat zprávy ve vlastně definovaném formátu. Hlavička vytvořeného formátu obsahuje: identifikátor druhu zprávy, číslo komunikačního rozhraní a typ formátu. Obsah přijatých zpráv je zapouzdřen do záhlaví. *Mux* při každém přijetí dat kontroluje návratový kód funkce *trap_ctx_recv()*, který identifikuje nový formát přijatých zpráv. Pokud dojde k detekování změny, tak se pošle *hello* zpráva s upraveným popisem rozhraní. V ostatních případech se jen přeposílají zapouzdřená data.

Modul *Demux* vyžaduje stejné přepínače jako *Mux*. Jediným rozdílem je, že název společného komunikačního rozhraní, které má *Mux* na posledním místě, musí být zde uveden jako první. Důvodem je, že knihovna *libtrap* zpracovává nejprve vstupní a pak výstupní rozhraní.

3.3 Zpracování zadaných parametrů

Načtení konfiguračního souboru má na starosti třída *ConfigParser*, která pomocí parametrického konstruktoru přijímá název konfiguračního souboru. Následně je možné volat členskou funkci *parseFile()*, která zpracovává jednotlivé konfigurační řádky a plní připravené struktury. Zároveň jsou inicializovány pole pro uchovávání vypočtených profilů. Jejich délka je specifikována preprocesorovou direktivou *#define DYNAMIC*.

Načtení konfiguračního souboru má na starosti třída *ConfigParser*, která neprovádí žádnou kontrolu vstupních dat, protože se již od návrhu předpokládá, že konfigurace bude generována odlišným programem, který zajistí správnost zadaných údajů. Jediný konstruktor třídy *ConfigParser* očekává jako vstupní parametr řetězec s cestou k cílovému souboru. Během vytváření objektu jsou postupně zpracovány jednotlivé řádky zadaného souboru. Záro-

veň jsou inicializovány pole pro uchovávání vypočtených profilů. Jejich délka je specifikována preprocesorovou direktivou *#define DYNAMIC*.

Pokud byly zadány parametry pro pravidelný export dat, tak se při inicializaci potřebných struktur zavolá funkce s názvem *initExportInterfaces()*, která vytvoří příslušná výstupní komunikační rozhraní. Jejich název je vždy vygenerován v následujícím formátu: *u:export-<názevKlíče><idKlíče>*. V opačném případě se tato inicializace přeskočí a žádné výstupní komunikační rozhraní pro export dat není vytvořeno.

3.4 Výpočet profilu

Veškeré přijaté údaje z kolektoru jsou v rámci detektoru ukládány do časových řad. Nad těmito daty jsou prováděny výpočty profilů, které využívají detekční funkce. Jelikož jsou profily přepočítávány s každým nový prvkem v časové řadě, tak je důležité provádět výpočty efektivně, aby nemusela být vždy procházena celá řada při změně jedné hodnoty. Během implementace byly vytvořeny členské funkce pro výpočet následujících částí profilu: průměr, klouzavý průměr, klouzavý rozptyl a klouzavý medián. Tyto členské funkce jsou popsány v níže uvedených sekcích.

- **getMovingMedian()**

  Klouzavý medián značí hodnotu běžného mediánu odpovídající prvkům umístěných v časové řadě. Pro jeho výpočet je nutné udržovat položky aktuálního časového okna v seřazeném pořadí. Při vkládání nového prvku je potřeba najít nejstarší prvek v poli a nahradit ho nově přijatým. Po provedeném nahrazení se pole může nacházet v neseřazeném stavu a je vyžadováno jeho opětovné seřazení. Tato operace není náročná, protože se změní vždy jen jeden prvek, který je nutné přesunou na správné místo. Celková složitost začlenění nového prvku je $O(n)$, protože operace nalezení a seřazení lze provést v lineárním čase. Paměťové nároky vyžadují alokování dodatečného místa, které obsahuje kopii časové řady.

  Dále je potřeba zjistit, zda je časová řada sudé nebo liché délky. Pokud je lichá, tak se jako medián vrátí její prvek umístění ve středu. V opačném případě je výsledkem průměr dvou prostředních hodnot.

  Aby mohly být provedeny veškeré popsané operace, tak členská funkce pro výpočet mediánu potřebuje tyto vstupní parametry:

  - **sensor_it** – iterátor na časovou řadu daného UniRec pole
  - **meta_it** – iterátor na meta informace o daném UniRec poli
  - **ur_field** – řetězec s názvem zpracovávaného UniRec pole
  - **ur_id** – číselný identifikátor přijatého UniRec pole

- **getMovingAverageAndVariance()**

  Tato členská funkce je zaměřená zejména na získání klouzavého rozptylu, ale v rámci jeho výpočtu je možné získat i hodnotu klouzavého průměru, a proto jsou v návratové hodnotě vráceny oba výsledky. Stejně jako v předchozí sekci klouzavý rozptyl i průměr jsou odvozeny z aktuálního časového okna celé řady. Průběh zpracování údajů probíhá s ohledem na vstupní data, která jsou reprezentována stále se posouvajícím časovým oknem.

  Pro umožnění efektivního výpočtu se používá několik pomocných paměťových struktur. První z nich jsou dvě pole *x* a *x2* se shodnou velikostí jako má hlavní časová řada. Do pole *x* jsou ukládána nově příchozí data a *x2* udržuje jejich druhé mocniny. Dále je nutné ukládat součty hodnot v těchto polích, které jsou uchovávány v proměnných *SX* a *SX2*.

  V rámci učící fáze detektoru jsou i zde příchozí prvky postupně vkládány do připravených struktur. Pokud jsou již pomocná pole naplněná, tak se při příchodu nové položky používá funkce *rotate()*, která provede levou rotaci prvků a umístí nejstarší prvek na konec. Tento prvek je poté nahrazen nejnovější hodnotou. Součty jednotlivých polí jsou upraveny podle následujícího předpisu:

  $$součetPole + novýPrvek - nejstaršíPrvek$$

  Po upravení pomocných struktur je možné provést finální výpočet. Průměr je snadno získán vydělením udržovaného součtu počtem prvků v časovém okně (N). Pro určení hodnoty rozptylu se provádí následující operace:

  $$\frac{N * SX2 - SX * SX}{N * (N - 1)}$$

  Pro získání veškerých údajů pro popsané výpočty přijímá členská funkce stejné vstupní parametry jako *getMedian()* a navíc je ještě očekávána hodnota *meta_id*, která určuje, zda bude v rámci metadat použita skupina *metaData* nebo *metaProfile*. Tyto skupiny byly diskutovány v dřívější Sekci 2.3.2.

- **getOverallAverage()**

  Poslední možnou částí vypočítávaného profilu sítě je průměr, který je získán v rámci této členské funkce. Na rozdíl od ostatních metod jsou v tomto případě zahrnuty i položky mimo časové okno. Výsledná hodnota reprezentuje celkový průměr všech dosud přijatých dat pro daný UniRec klíč. Výpočetní operace využívá znalosti aktuálního průměru a celkového počtu prvků (M). Při přijetí nové položky se nová hodnota vytvoří dle následujícího předpisu:

  $$\frac{novýPrvek + (M - 1) * aktuálníPrůměr}{M}$$

Velkou výhodnou je, že nejsou potřeba žádné dodatečné pomocné paměťové struktury a vše je efektivně uloženo do jedné proměnné, která se aktualizuje s přicházejícími prvky. Vstupní parametry jsou totožné s členskou funkcí *getMovingAverageAndVariance()* až na položku ur_field, která zde není potřeba.

3.5 Detekční metody

Na základě vypočtených údajů z přicházejících dat a podle zapsané konfigurace jsou spouštěny členské funkce určené pro detekci incidentů. Popis realizace jednotlivých možností je uveden v následujících sekcích.

- **Kontrola limitů**

  Tato detekce je reprezentována členskou funkcí *dataLimitCheck()*, která vyhodnocuje *soft* a *hard* limity.

  V případě *soft* limitu se s každou nově příchozí hodnotou provádí porovnání aktuálního profilu sítě s definovanými hranicemi v konfiguračním souboru. Pokud dojde k jejich překročení, tak se zvýší hodnota odpovídajícího čítače o jedna. Jakmile čítač překročí nastavenou hranici pro *grace period*, tak se odešle zpráva o detekovaném incidentu. V případě, že během porovnání hodnota aktuálního profilu nepřekračuje nastavené limity, tak se do čítače uloží číslo 0.

  Hard limit provádí stejný způsob porovnání částí aktuálního profilu s nastavenými hodnotami. Jediným rozdílem je, že odeslání incidentu se provede ihned po překročení nakonfigurovaných hranic.

  Mezi *soft* a *hard* limitem není žádná závislost a lze je používat nezávisle. Při konfiguraci libovolného z nich se vždy povinně vyžaduje určení minimální a maximální meze současně.

- **Kontrola změny**

  Neočekávaný nárůst dat je možné detekovat pomocí členské funkce *dataChangeCheck()*. Při každém přijetí nové zprávy se provede kontrola poměru mezi hodnotami vzorového a aktuálního profilu sítě. V případě, že vypočítaný poměr překročí nastavené meze, je odeslána informace o detekované anomálii. Implementace předpokládá, že je vždy nastavena spodní i horní hranice zároveň.

- **Periodické kontroly**

  V rámci vytvořeného detektoru jsou implementovány dvě členské funkce s názvy *periodicCheck()* a *periodicExport()*, které provádí požadovanou akci jednou za periodu, jejíž délka je určená konfiguračním souborem. Toto chování je realizované pomocí vláken, které je možné uspávat a probouzet. Díky tomu je možné provádět kontroly bez ohledu na přijetí dat.

Vytváření vláken má na starosti členská funkce *runThreads()*, která dle nastavených parametrů spouští vlákna pro dostupné UniRec položky.

V členské funkci *periodicCheck()* jsou prováděny dvě kontroly. První porovnává časovou značku posledního přijatého prvku do časové řady s aktuálním časem. Pokud výsledný rozdíl je větší než definovaný limit, tak je odeslána zpráva s popisem incidentu. Druhá kontrola sleduje hodnoty v časové řadě a pokud s příchodem nové zprávy nedojde k její změně, tak je zvýšen čítač o jedna. Jakmile čítač překročí nastavený limit, tak se odešle zpráva o nalezené anomálii. V případě, že jsou sousední položky odlišné, tak se čítač nastaví zpět na hodnotu 0.

Druhá členská funkce s názvem *periodicExport()* zajišťuje pravidelný export položek profilu. Vlákno po probuzení pouze naplní připravenou zprávu pro export a pošle ji příslušným výstupním komunikačním rozhraním.

Kapitola 4

Testování

V této kapitole je popsán postup testování vytvořeného detekčního řešení. Nejprve je představeno testovací prostředí a jsou otestovány obě možné metody nasazení. Dále jsou ověřeny jednotlivé případy hrozeb ze Sekce 2.5 a také jsou prověřeny ostatní funkcionality detektoru. V závěru je provedeno měření chování jednotlivých položek z profilu.

4.1 Testovací prostředí

Veškeré provedené testy proběhly na lokálním počítači s operačním systémem Ubuntu 16.04, na kterém zároveň proběhl vývoj nástroje. Systémové prostředky dostupné pro testovaní byly: 4 jádra CPU, 8 GB RAM a 17 GB SSD disk.

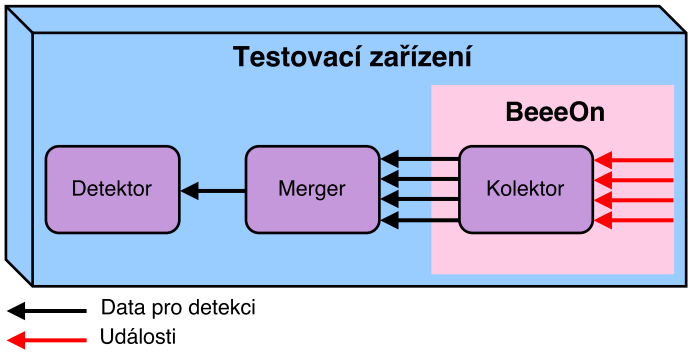
Na počítači byl nasazen systém NEMEA a IoT brána BeeeOn, která byla rozšířena o vytvořený detekční nástroj. Pro generování senzorových dat byly použity následující zařízení: BLE teplotní senzor (BeeWi), Z-Wave zásuvky (POPP a Fibaro) a virtuální senzory, které jsou dostupné pro testování v rámci BeeeOn brány. Senzorová data přijímal testovací počítač, který měl připojený USB Z-Wave kontroler (Aeotec) a integrované BLE rozhraní.

4.2 Způsoby nasazení

Jako první byly testovány možnosti nasazení vytvořeného nástroje, který je možné používat v následujících režimech:

- **Lokální režim**

  V lokální variantě byl použit pouze modul detektoru a kolektoru. Aby se nemusela pro každou událost spouštět nová instance detektoru, byl využit již existující NEMEA modul *Merger*, který dokáže jednotlivé UniRec



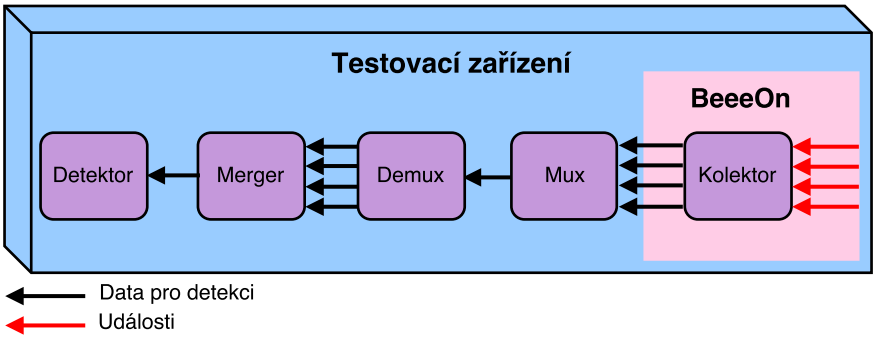
Obrázek 4.1: Lokální nasazení

zprávy spojit do jedné konsolidované. Způsob nasazení je zobrazen na Obrázku 4.1.

Kolektor vždy odesílá dostupné události výstupními komunikačními rozhraními dle konfigurace brány. Exportovaná data přijímá NEMEA modul *Merger*, který je spojuje a předává detektoru.

- **Oddělený režim**

  Druhý způsob nasazení byl také otestován na testovacím počítači, protože lze využít lokálních soketů dostupných přes NEMEA framework. Provedení realizace je na Obrázku 4.2.



Obrázek 4.2: Oddělené nasazení

Složení komponent vychází z prvního případu, který byl rozšířen o moduly *Mux* a *Demux*, které umí spojit a rozdělit přicházející komunikaci.

Cílem testů bylo, aby detektor obdržel všechny odeslané zprávy z kolektoru. Ověření bylo provedeno spuštěním detektoru s přepínačem *-vv*, který

zapne ladící výpisy druhé úrovně pro zobrazení přijatých UniRec polí. Výsledky obou případů byly úspěšné a všechna data byla přijata.

4.3 Ověření scénářů útoků

Velmi důležitou částí bylo otestování definovaných anomálií, které mohou reprezentovat skutečný útok na sít. Tato sekce popisuje testy pro jednotlivé scénáře.

S.1 Periodicita dat

Tento scénář nebylo nutné dělit na jednotlivé protokoly, protože popisuje obecné chování připojených senzorů bez ohledu na technologii. Cílem bylo odhalit provoz, který porušuje očekávaný periodický průběh. V rámci detekce nebyly potřebné naučené profily sítě, ale využívalo se parametrů ze skupiny *general* umožňujících periodické kontroly.

První test byl určený na odhalení nepravidelného přijetí dat. Pro ověření byla použita událost *onExport*, která poskytuje hodnoty získané ze senzorů, a v konfiguračním souboru byla aktivována pravidelná kontrola dat každých 8 sekund.

Výsledek testu byl úspěšný. Pokud nebyla dodržena pravidelnost o více než povolenou odchylku, tak byly odeslány informace o incidentu.

Několik událostí poskytovaných branou BeeeOn je periodických a po uplynutí definovaného času vždy odešlou dostupné statistiky. Druhý test byl zaměřen na odhalení případu, kdy se přijímané datové položky nemění, což může reprezentovat odpojení nebo ztrátu čidla. Pro otestování byla zvolena událost *onHciStats*, která získává informace o provozu BLE sítě. Dále byla v konfiguračním souboru nastavena pravidelná kontrola dat na 7 sekund a maximálně 5 po sobě přijatých hodnot mohlo být stejných.

Výsledek byl pozitivní, protože při přijetí více než pěti stejných po sobě jdoucích hodnot byla detekována anomálie.

S.2 Množství přenášených zpráv

V případě protokolu Z-Wave byly pro nasimulování anomálií použity dvě vzdáleně ovladatelné zásuvky. Obě mají k dispozici ovládací tlačítko pro vypnutí a zapnutí zásuvky. Tímto způsobem byly generovány nové zprávy. Cílem tohoto scénáře bylo odhalit neočekávaný nárůst dat, a proto byla zvolena detekční metoda, která hlídá tyto změny. V rámci testu byly do profilu vloženy všechny dostupné položky. Limitem pro ohlášení incidentu byl pětinásobný nárůst provozu. Délka časové řady byla nastavena na deset prvků a prvních jedenáct přijatých zpráv bylo ignorováno, protože během nich bylo navazováno spojení.

První detekce byla zaměřena na položku *SOAFCount*, která určuje celkový počet detekovaných zpráv. Druhý scénář sledoval hodnotu *receivedCount*, která reprezentuje počet přijatých zpráv od konkrétního čidla.

Pro BLE byla časová řada zkrácena na pět prvků, žádné zprávy nebyly ignorovány a limitem pro určení anomálie byl dvojnásobný nárůst dat. Test byl proveden pro položku *rxBytes*, která určuje množství obdržených zpráv. Údaje byly vyčítány skriptem, který v čase měnil četnost zaslaných požadavků o data.

Výsledky testů byly úspěšné a každá změna provozu byla odhalena. Průběh všech testovaných případů byl velmi podobný a lišil se jen typ události a způsob generování dat. Chování jednotlivých položek profilu velmi ovlivňovala délka časové řady, která určovala paměťové okno. Nejcitlivější položkou na změnu byl rozptyl, který výrazně zvyšoval svou hodnotu při vložení vyššího čísla. K častým výchylkám docházelo i v případě průměru. Méně frekventované změny nastávaly u mediánu, který nejvíce využíval délky časové řady. Nejmenších odlišností dosahoval klouzavý průměr, protože ve své hodnotě obsahuje i data mimo aktuální časové okno.

S.3 Limity senzorových hodnot

Pro tento scénář byla použita data vygenerována virtuálními senzory, které jsou dostupné v rámci brány BeeeOn. Do konfiguračního profilu byly vloženy všechny položky, pro které byly specifikovány parametry pro očekávané *soft* a *hard* limity. Cílem scénáře bylo odhalení změn v aktuálním profilu provozu, které porušují předepsané limity.

Výsledek detekce splnil očekávání a nepovolené změny byly úspěšně detekovány. Chování jednotlivých položek profilu se shodovalo s předchozím scénářem.

S.4 Kvalita přenosového kanálu

Cílem tohoto scénáře bylo detekování změny kvality přenosového kanálu. Pro Z-Wave jsou k tomuto účelu vhodné položky *lastResponseRTT* a *dropped*. Údaj *lastResponseRTT* je součástí události *onNodeStats* a *dropped* je obsažen v *onDriverStats*. Z tohoto důvodu byl při generování dat použit NEMEA modul *Merger*, který spojil dvě různě události do jedné. Zároveň bylo nutné v konfiguračním souboru popsat obě hodnoty. V případě *lastResponseRTT* byla jako anomálie označena hodnota klouzavého průměru časové řady přesahující dvojnásobek vzorového profilu a nebo byla nižší než polovina stanoveného klouzavého průměru. Pro položku *dropped* bylo jako incident považováno libovolné zvětšení klouzavého mediánu. Tuto kontrolu lze nejlépe provést nastavením časového okna na délku jedna, položky *hard* limitu na hodnotu jedna a způsobu ukládání na možnost *delta*.

Proto byla velikost časového okna rovna jedné a hard limitu byl nastaven na jedna.

Jelikož došlo ke spojení dvou různých událostí, kde *onNodeStats* umožňuje získat informace pro jednotlivé senzory a *onDriverStats* pro celou síť, tak výsledkem byla událost poskytující informace pro každý koncový prvek. Z tohoto důvodu musel být v konfiguraci uveden v rámci klíče identifikátor příslušných zařízení určených k analýze.

Výstupy detekce se shodovaly s předpoklady, a proto byl výsledek testu úspěšný. V současné době se zatím nepodařilo připravit zkušební prostředí, ve které by bylo možné otestovat rušení kanálu, a tím ovlivnit hodnotu *lastResponseRTT*. Pro účely otestování detekce však byla postačující ruční úprava dat v souboru se zachyceným provozem.

V případě BLE by se využily položky *rxErrors* nebo *txErrors* a nastavení detekce by bylo shodné s *lastResponseRTT*. Vzhledem k tomu, že by došlo jen ke změně názvu klíče v konfiguračním souboru, tento test nebyl proveden.

S.5 Konektivita

Pro ověření posledního scénáře lze využít poznatků z předchozích testů. Ztráta konektivity může být způsobena neobdržením očekávané události a to lze detekovat pomocí periodicity dat. Druhým způsobem je výrazné zhoršení statistik přenosového kanálu, které byly zpracovány v předchozím případu užití.

Ověření definovaných scénářů umožnilo otestování správné funkcionality vytvořeného detekčního systému a zároveň byla sestavena množina informací, kterou je nutné sledovat k získání přehledu o stavu sítě.

Vygenerovaný provoz, na kterém byly jednotlivé případy užití otestovány je uložen i s nalezenými anomáliemi a konfiguračním souborem na přiloženém CD. V případě testu periodicity dat nebylo možné záznam komunikace uložit, protože použitá detekce využívá aktuální čas. Z tohoto důvodu byl do souboru místo záznamu uložen popis, jak takový provoz vygenerovat.

4.4 Test exportu dat

Identifikované scénáře útoků otestovaly většinu funkcí detekčního systému. Vytvořený nástroj ovšem poskytuje i možnost pravidelného exportu definovaných položek aktuálního profilu, a tím vytváří informace pro další pokročilejší detekce.

Sada testů ověřila, že vypočítané části profilu lze spolehlivě exportovat. Výsledek byl tedy pozitivní.

4.5 Měření položek profilu

Již v Sekci 4.3 bylo možné sledovat odlišné chování jednotlivých částí profilu. V této sekci je provedeno vyhodnocení každé z nich. Veškeré měření probíhalo nad stejným provozem, kde byly zprávy generovány každé tři sekundy a pravidelné události chodily každých pět sekund. Útoky byly reprezentovány nárůstem sledovaného provozu. První útok byl zahájen 3 minuty po zahájení komunikace a trval 90 vteřin. Druhý útok nastal 210 vteřin po prvním a trval 30 vteřin. Tyto hodnoty byly zvoleny tak, aby se projevily vlastnosti jednotlivých částí profilu.

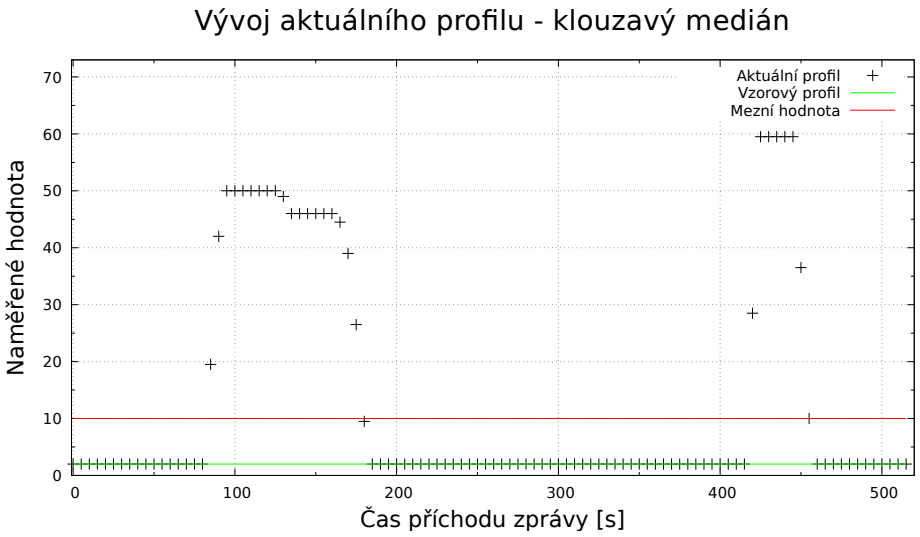
Tabulka 4.1 zobrazuje pro každou položku profilu počet detekovaných anomálií s rostoucí velikostí časové řady. Její rostoucí délka nejvíce stabilizovala klouzavý medián, protože jeho počty incidentů klesly se zvyšující se délkou. Zároveň z jeho grafu 4.3 lze vidět, že začátek anomálií odhalil vždy jako poslední a jejich doba trvání byla nejkratší.

S větší časovou řadou bylo možné uchovat i více hodnot, a tím se detekovalo více událostí v rámci klouzavého průměru a rozptylu. Začátky i konce rozpoznání incidentů byly u obou položek velmi podobné. Jediným rozdílem u klouzavého rozptylu bylo, že při stabilním nárůstu sledované položky může jeho hodnota po uplynutí časového okna opět klesnout do bezpečného pásma. Graf průběhu 4.4 pro větší přehlednost nezobrazuje klouzavý rozptyl, ale klouzavou směrodatnou odchylku.

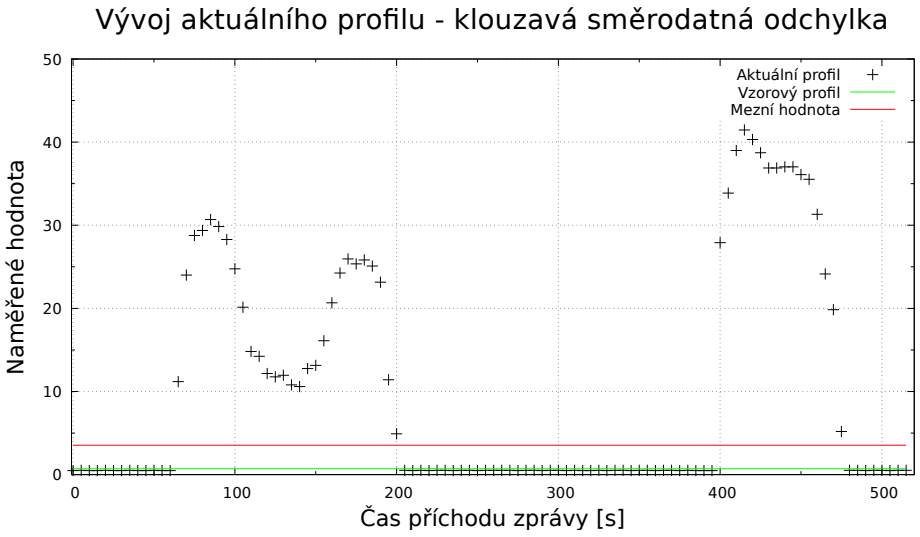
Průměr jako jediný nebyl závislý na časovém okně. Ovšem s rostoucí délkou řady byla delší i doba učení, při které byla vždy stanovena lehce odlišná hodnota vzorového profilu a mezních limitů. Při provedeném měření se vždy s delším paměťovým oknem vzorové hodnoty mírně zvýšily. Na základě těchto předpokladů lze v jeho grafu 4.6 vidět pomalé klesání, díky kterým byly incidenty detekovány nejdelší dobu. Zároveň dle očekávání velikost časového okna nemá vliv na počet detekovaných anomálií.

velikost časového okna	**klouzavý medián**	**klouzavý rozptyl**	**klouzavý průměr**	**průměr**
1	26	0	26	26
3	26	30	30	90
5	26	34	32	89
10	26	44	40	89
15	19	54	48	89
20	19	62	56	89

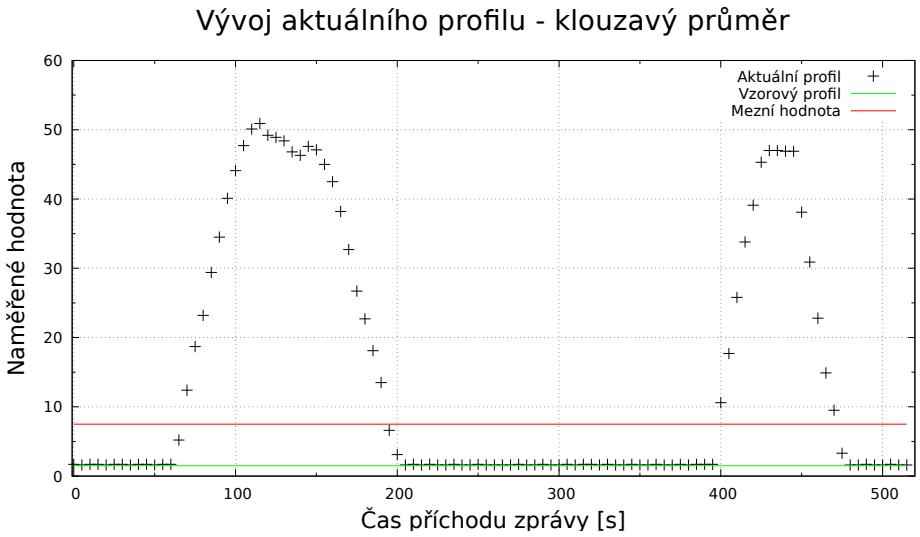
Tabulka 4.1: Tabulka obsahuje počty detekovaných anomálií pro každou část profilu v závislosti na rostoucí délce časového okna



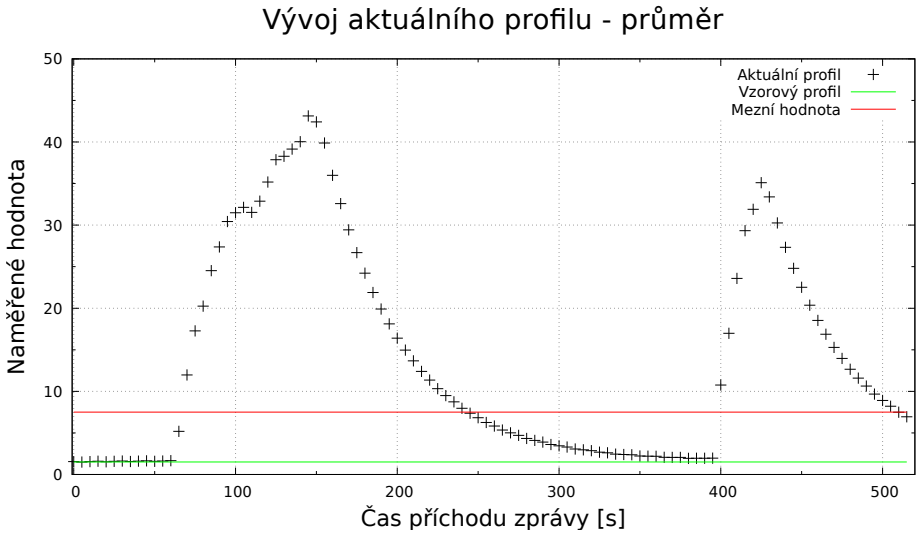
Obrázek 4.3: Graf vývoje klouzavého mediánu s časovou řadou délky 10



Obrázek 4.4: Graf vývoje klouzavé směrodatné odchylky s časovou řadou o délce 10



Obrázek 4.5: Graf vývoje klouzavého průměru s časovou řadou délky 10



Obrázek 4.6: Graf vývoje průměru s časovou řadou délky 10

Závěr

Cílem diplomové práce bylo analyzovat aktuální stav v oblasti IoT sítí a identifikovat bezpečnostní slabiny. Na základě získaných znalostí byl navržen, implementován a otestován nástroj pro monitorování a automatickou detekci anomálního provozu. Výsledný nástroj se skládá z několika komponent, které komunikují pomocí systému NEMEA a rozšiřují funkcionalitu IoT brány BeeeOn o možnost detekce.

V první části této práce byl představen koncept fog computingu, který mění způsob zpracování dat v IoT sítích. Dále byly popsány jednotlivé komunikační vrstvy a detailně rozebrány aktuálně používané protokoly včetně jejich bezpečnostních slabin. Následně byla věnována pozornost možným způsobům detekce útoků a také byly hledány již existující nástroje pro řešení této problematiky, které v současnosti nedostatečné pokrývají možné hrozby. Nejméně pokrytá byla senzorová vrstva, kde jsou nejzranitelnější bezdrátové komunikační protokoly, které jsou zároveň pro svou mobilitu velmi často používané. V závěru jsou uvedeny požadavky na výsledný program a vhodné technologie pro vývoj.

Při návrhu byly splněny všechny kladené požadavky a výsledné řešení bylo navrženo tak, aby jej bylo možné nasadit i na IoT branách s omezenými prostředky nebo v decentralizovaném prostředí. Dále byly popsány scénáře útoků, které se mohou objevit při reálném provozu v senzorové vrstvě. V rámci scénářů byla sestavena množina informací, které je potřeba měřit k získání přehledu o stavu sítě.

Vzhledem k pečlivě provedenému návrhu nebylo nutné během implementace řešit žádné problémy. Naopak díky použitému frameworku NEMEA bylo spoustu věcí usnadněno. Při realizaci byly splněny všechny funkční i nefunkční požadavky, které dokonce přesahují nároky specifikované v zadání práce.

Velký důraz byl kladen na testování, které odhalilo řadu implementačních chyb, ale všechny byly opraveny. V rámci testů bylo ověřeno úspěšné detekování navržených scénářů útoků, byly otestovány způsoby nasazení a také bylo provedeno měření chování položek profilu sítě.

Zpracování této diplomové práce mělo pro mě velký přínos. Dozvěděl jsem se mnoho nových znalostí z oblasti IoT, které jsem si navíc prakticky vyzkoušel. Tato oblast se bude v dalších letech rozšiřovat, a proto se získané zkušenosti budou velmi hodit v dalších projektech.

Budoucí práce

V následujících letech je očekáváno velké rozšíření IoT sítí, které jsou řízeny člověkem a konfigurovány strojem. Z tohoto důvodu má tato práce velký potenciál, protože tuto oblast pokrývá teoretickým i praktickým směrem a může být využita jako základ pro budoucí práce.

Jedním ze způsobů rozšíření je použití dalších detekčních metod, které budou využívat externí sondy pro získávání podrobnějších informací o daném protokolu. Tyto sondy zároveň mohou sloužit jako honeypoty.

Dalším rozšířením může být externí nástroj, který se bude starat o generování konfigurace pro vytvořený detektor. Generátor nastavení bude velmi vhodný pro koncové uživatele, protože si budou moci přehledně vybrat své parametry bez znalosti všech závislostí, které textová varianta ani modul pro zpracování nehlídá. Případně může být zajímavé využití strojového učení pro určení profilu sítě a konfiguračních parametrů.

Vytvořený program zároveň umožňuje pravidelně získávat informace o profilu sítě, které mohou být využity k sestavení datového modelu. Následně při použití analytických metod lze provádět pokročilé způsoby detekce.

Důležité bude také pokrytích další komunikačních protokolů, které zahrnují i protokoly síťové vrstvy. Pro síťové protokoly může být vhodné využít principů nástroje *Joy* [24], který se zaměřuje na vyhodnocování incidentů v šifrovaném provozu.

Literatura

[1] Statista: Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025. Statista. [online], 2016 [cit. 2018-01-14]. Dostupné z: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[2] Milici, S.; Lázaro, A.; Villarino, R.; aj.: Wireless Wearable Magnetometer-Based Sensor for Sleep Quality Monitoring. *IEEE Sensors Journal*, 2018: s. 2145–2152.

[3] Whitehead, D. E.; Owens, K.; Gammel, D.; aj.: Ukraine cyber-induced power outage: Analysis and practical mitigation strategies. In *2017 70th Annual Conference for Protective Relay Engineers (CPRE)*, 2017, s. 1–8.

[4] BeeeOn: Main Page. beeeon.org. [online], 2017 [cit. 2018-02-06]. Dostupné z: https://beeeon.org

[5] Statista: Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. Cisco. [online], 2015 [cit. 2018-01-15]. Dostupné z: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf

[6] Zhao, K.; Ge, L.: A Survey on the Internet of Things Security. In *2013 Ninth International Conference on Computational Intelligence and Security*, 2013, s. 663–667.

[7] Pacheco, J.; Hariri, S.: IoT Security Framework for Smart Cyber Infrastructures. In *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, 2016, s. 242–247.

[8] OASIS: MQTT Version 3.1.1 Plus Errata 01. OASIS. [online], 2015 [cit. 2018-03-08]. Dostupné z: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html

[9] DC-square: MQTT Essentials Wrap-Up. HiveMQ. [online], 2018 [cit. 2018-03-08]. Dostupné z: https://www.hivemq.com/blog/mqtt-essentials-wrap-up

[10] DC-square: MQTT Security Fundamentals. HiveMQ. [online], 2018 [cit. 2018-03-08]. Dostupné z: https://www.hivemq.com/mqtt-security-fundamentals/

[11] Shelby, Z.; Hartke, K.; Bormann, C.: The Constrained Application Protocol (CoAP). RFC7252. [online], 2014 [cit. 2018-03-08]. Dostupné z: https://tools.ietf.org/html/rfc7252

[12] Z-Wave Alliance: Z-Wave Plus Certification. Z-Wave Alliance. [online], 2018 [cit. 2018-03-17]. Dostupné z: https://z-wavealliance.org/z-wave_plus_certification/

[13] OpenZWave: OpenZWave. OpenZWave. [online], 2018 [cit. 2018-03-17]. Dostupné z: http://www.openzwave.net/

[14] Krejčí, R.; Hujňák, O.; Švepeš, M.: Security survey of the IoT wireless protocols. In *2017 25th Telecommunication Forum (TELFOR)*, 2017.

[15] Designs, S.: Introduction to Z-Wave Specifications. ZWave. [online], 2016 [cit. 2018-03-18]. Dostupné z: http://zwavepublic.com/specifications

[16] Fouladi, B.; Ghanoun, S.: Security Evaluation of the Z-Wave Wireless Protocol. In *BlackHat Conference, Las Vegas, NV, USA*, 2013 [cit. 2018-03-18].

[17] Jasek, S.: Gattacking Bluetooth Smart Devices. SecuRing. [online], 2016 [cit. 2018-03-18]. Dostupné z: http://gattack.io/whitepaper.pdf

[18] Fouladi, B.; Ghanoun, S.: EZ-Wave. Github. [online], 2016 [cit. 2018-03-18]. Dostupné z: https://github.com/cureHsu/EZ-Wave

[19] Rose, A.; Ramsey, B.: Picking Bluetooth Low Energy Locks from a Quarter Mile Away. In *DEF CON, Las Vegas, NV, USA*, 2016 [cit. 2018-03-18].

[20] Seri, B.; Vishnepolsky, G.: BlueBorne. Armis. [online], 2017 [cit. 2018-03-18]. Dostupné z: http://go.armis.com/hubfs/BlueBorne%20Technical%20White%20Paper-1.pdf

[21] TechTarget: Do you need an IDS or IPS, or both? TechTarget. [online], 2009 [cit. 2018-03-19]. Dostupné z: http://searchsecurity.techtarget.com/Do-you-need-an-IDS-or-IPS-or-both

[22] CZ.NIC: PaKon - sledování sítě (Parental Control). CZ.NIC. [online], 2018 [cit. 2018-03-20]. Dostupné z: `https://doc.turris.cz/doc/cs/howto/pakon`

[23] Cejka, T.; Bartos, V.; Svepes, M.; aj.: NEMEA: A framework for network traffic analysis. In *2016 12th International Conference on Network and Service Management (CNSM)*, 2016, doi:10.1109/CNSM.2016.7818417.

[24] Systems, C.: Joy. Github. [online], 2016 [cit. 2018-04-14]. Dostupné z: `https://github.com/cisco/joy`

Příloha A

Seznam použitých zkratek

IoT Internet of Things

IP Internet Protocol

HTTPS Hypertext Transfer Protocol Secure

VPN Virtual Private Network

MQTT Message Queuing Telemetry Transport

CoAP Constrained Application Protocol

AMQP Advanced Message Queuing Protocol

DDoS Distributed Denial Of Service

MITM Man In The Middle

M2M Machine To Machine

TCP Transmission Control Protocol

QoS Quality of Service

TLS Transport Layer Security

RESTful Representational State Transfer

UDP User Datagram Protocol

HTTP Hypertext Transfer Protocol

URI Uniform resource identifier

DTLS Datagram Transport Layer Security

ISM Industrial, Scientific and Medical

AES Advanced Encryption Standard

PIN Personal Identification Number

QR Quick Response

ECDH Elliptic-curve Diffie–Hellman

BLE Bluetooth Low Energy

SIG Special Interest Group

NFC Near Field Communication

IDS Intrusion Detection System

IPS Intrustion Prevention System

SCADA Supervisory Control And Data Acquisition

NEMEA Network Measurements Analysis

UniRec Unified Record

RAII Resource Acquisition Is Initialization

RTT Round-Trip Time

Příloha **B**

Instalační příručka

Realizovaný detekční systém pro svůj běh využívá framework NEMEA a projekt BeeeOn. V následujících krocích je popsána instalace těchto dvou nástrojů, způsob nasazení kolektoru a spuštění detekčních modulů.

1. **Instalace projektu BeeeOn**

   Zdrojové kódy tohoto projektu se nachází v repozitáři služby Github. Pro jeho sestavení je nutné vykonat následující příkazy:

   ```
   git clone --recursive https://github.com/BeeeOn/gateway.git
   cd  gateway
   mkdir build
   (cd build && cmake ..)
   make -C build
   ```

2. **Instalace NEMEA frameworku**

   Kompletní zdrojové kódy systému NEMEA jsou také udržovány v repozitářích služby Github. Pro vytvořený nástroj je potřeba nasadit část *NEMEA-Framework*.

   ```
   git clone https://github.com/CESNET/Nemea-Framework.git
   cd Nemea-Framework
   ./bootstrap.sh
   ./configure
   make
   sudo make install
   ```

   Pro spouštění vytvořených testů nebo pro úpravu příchozích a odchozích dat ve formátu UniRec se mohou hodit moduly *Logger*, *Logreplay* a *Merger*, které se nacházejí v částí *NEMEA-Modules*. V tomto repozitáři se zároveň nachází vytvořené moduly *Mux* a *Demux*.

```
git clone https://github.com/CESNET/Nemea-Modules.git
cd Nemea-Modules
./bootstrap.sh
./configure
make
sudo make install
```

3. **Nasazení kolektoru**

   V rámci tohoto kroku je potřeba zdrojové kódy vytvořeného kolektoru přesunout do BeeeOn brány a upravit konfigurační soubory. Postup je popsán v následujících sekcích.

   - **Přesun zdrojových souborů**

     ```
     cp NemeaCollector.* <adresářBeeeOn>/src/core
     cp fields.* <adresářBeeeOn>/src/core
     ```

   - **Úprava souborů pro kompilaci**

     ```
     vim <adresářBeeeOn>/src/CMakeLists.txt
     ```

     Do příslušných částí v souboru je potřeba vložit tyto položky:

     ```
     find_library (LIBTRAP trap)
     find_library (UNIREC unirec)
     find_library (PCAP pcap)

     ${PROJECT_SOURCE_DIR}/core/fields.cpp
     ${PROJECT_SOURCE_DIR}/core/NemeaCollector.cpp

     ${PCAP}
     ${UNIREC}
     ${LIBTRAP}
     ```

     Podobné změny je nutné provést v adresáři *test*

     ```
     vim <adresářBeeeOn>/test/CMakeLists.txt
       find_library (LIBTRAP trap)
       find_library (UNIREC unirec)
       find_library (PCAP pcap)

       ${PCAP}
       ${UNIREC}
       ${LIBTRAP}
     ```

   - **Úprava sestavovací konfigurace brány**

     ```
     vim <adresářBeeeOn>/conf/config.d/factory.xml
     ```

V otevřeném souboru je nutné do odpovídajících elementů vložit tyto parametry:

```
#element instance name="distributor"
<add name="listener" ref="collector"/>

#element instance name="bluetoothAvailability"
<add name="listeners" ref="collector"/>

#element instance name="zwaveDeviceManager"
<add name="listeners" ref="collector"/>

#element instance name="commandDispatcher"
<add name="listeners" ref="collector"/>

#element factory
<instance name="collector"
          class="BeeeOn::NemeaCollector">
  <set name="onExportInterface"
       text="u:event-onExport"/>
  <set name="onHCIStatsInterface"
       text="u:event-onHCIStats"/>
  <set name="onDispatchInterface"
       text="u:event-onDispatch"/>
  <set name="onNodeStatsInterface"
       text="u:event-onNodeStats"/>
  <set name="onDriverStatsInterface"
       text="u:event-onDriverStats"/>
</instance>
```

- **Rekompilace brány**

```
cd <adresářBeeeOn>
(cd build && cmake ..)
make -C build
```

4. **Kompilace detektoru**

   Kompilace komponenty detektoru se provede následujícím příkazem:

```
g++ DataDetector.cpp fields.cpp ConfigParser.cpp \
Analyzer.cpp -o prg -ltrap -lpcap -lunirec --std=c++11 \
-Wno-write-strings -pthread
```

5. **Spuštění detekčního systému**

   Posledním krokem je spuštění celého detekčního systému. Pro účely demonstrace je v ukázce použit oddělený model nasazení popsaný v Sekci 4.2, který zahrnuje více komponent. Při reálném použití stačí spustit pouze kolektor a jednu instanci detektoru pro každý typ události. V uvedené ukázce každá z komponent používá standardní výstup, proto je nutné každý příkaz spustit v samostatném terminálovém okně.

```
cd <adresářBeeeOn>
build/src/beeeon-gateway -c conf/gateway-testing.ini \
   -Dtesting.center.enable=yes
mux "u:event-onDriverStats,u:event-onNodeStasts,u:output" \
   -n 2
demux -i "u:output,u:onDriverStats,u:onNodeStats" -n 2
merger -i "u:onDriverStats,u:onNodeStats,u:merged" -n 2
cd <adresářDetector>
./prg -i "u:merged,u:alert" -v
logger -i "u:alert" -t
logger -i "u:export-dropped0" -t
```

Příloha C

Nástroje pro testování

V následující kapitole přílohy se nachází popis způsobu použití nástrojů, které byly uplatněny v rámci testování.

- **Logger**

  Tento NEMEA modul slouží k ukládání přijatých UniRec zpráv. Při spuštění je nutné zadat přepínač *-i* s popisem vstupního komunikačního rozhraní. Dále je dobré použít přepínač *-w <názevSouboru>* pro ukládání přijatých dat do souboru, *-t* pro uložení datové hlavičky příchozích dat a *-T*, který ukládá i časové značky.

  Příklad spuštění: *logger -t -T -i u:alert -w alert-data.log*

- **Logreplay**

  Modul *Logreplay* lze využít pro přehrání zachyceného provozu pomocí modulu *Logger*. Mezi povinné přepínače patří *-i*, za kterým následuje popis výstupního komunikačního rozhraní, a *-f <názevSoboru>* pro načtení uloženého provozu. Vhodným přepínačem může být *-n*, který neposílá po přehrání EOF (End Of File) zprávu. Pokud soubor se zachycenými daty obsahuje časové značky každého záznamu, tak jsou jednotlivé zprávy odesílány dle těchto značek.

  Ukázka použití: *logreplay -i u:zwave -f z-wave-connection.log -n*

- **Merger**

  Poslední uvedený NEMEA modul se používá pro slučování několika různých vstupních UniRec záznamů do jednoho společného, který je odesílán výstupním rozhraním. Očekávanými parametry jsou *-i* s popisem komunikačního rozhraní a *-n* určující počet vstupních rozhraní, který odpovídá zadanému popisu. Název výstupu je vždy uveden jako poslední položka v rámci přepínače *-i*.

  Možné zavedení: *merger u:DriverStasts,u:NodeStats,u:merged -n 2*

Příloha D

Popis množiny senzorových informací

Tato část přílohy obsahuje kompletní popis množiny informací, kterou lze v současné době získat o provozu senzorových protokolů v rámci BeeeOn brány. Při popisu byla množina informací rozdělena do příslušných událostí, které jsou reprezentovány samostatnou sekcí.

D.1 onDriverStats

SOAFCount: počet přijatých Start Of Frame bajtů

ACKWaiting: počet nevyžádaných zpráv při čekání na potvrzení

readAborts: počet nedokončených operacích čtení z důvodu překročení časového limitu

badChecksum: počet zpráv se špatných kontrolním součtem

readCount: počet úspěšně přijatých zpráv

writeCount: počet úspěšně odeslaných zpráv

CANCount: počet přijatých CAN bajtů

NAKCount: počet přijatých negativních potvrzení

ACKCount: počet přijatých potvrzení

OOFCount: počet přijatých Out Of Frame bajtů

dropped: počet zahozených zpráv

retries: počet znovu odeslaných zpráv

callbacks: počet neočekávaných callbacků

badroutes počet neodeslaných zpráv kvůli směrování

noACK: počet neobdržených potvrzení

netBusy: počet zpráv s chybovým stavem

notIdle: počet zpráv se stavem not idle

nonDelivery:počet nedoručených zpráv

routedBusy: počet přijatých zpráv se stavem routed busy

broadcastReadCount: počet přijatých všesměrových zpráv

broadcastWriteCount:počet odeslaných všesměrových zpráv

D.2 onNodeStats

sentCount: počet odeslaných zpráv

sentFailed: počet zpráv, které se nepodařilo odeslat

receivedCount: počet přijatých zpráv

receivedDuplications: počet přijatých duplikovaných zpráv

receivedUnsoliced: počet přijatých nevyžádaných zpráv

lastRequestRTT: RTT poslední odeslané zprávy

lastResponseRTT: RTT poslední přijaté odpovědi

averageRequestRTT: průměr z lastRequestRTT

averageResponseRTT: průměr z lastResponseRTT

quality: kvalita připojení

nodeID: identifikátor senzoru

D.3 onHCIStats

address: adresa BLE rozhraní

aclMtu: velikost MTU pro ACL zprávy

aclPackets: velikost zásobníku pro ACL zprávy

scoMtu: velikost MTU pro SCO zprávy

scoPackets: velikost zásobníku pro SCO zprávy

rxErrors: počet přijatých chybových zpráv

txErrors: počet odeslaných chybových zpráv

rxEvents: počet přijatých BLE událostí

txCmds: počet odeslaných BLE příkazů

rxAcls: počet přijatých ACL zpráv

txAcls: počet odeslaných ACL zpráv

rxScos: počet přijatých SCO zpráv

txScos: počet odeslaných SCO zpráv

rxBytes: počet přijatých bytů

txBytes: počet odeslaných bytů

D.4 onExport

value: senzorová hodnota

deviceID: identifikátor senzoru

Příloha **E**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src` ..... zdrojové soubory
  - `impl` ..... zdrojové kódy implementace
    - `collector` ..... zdrojové kódy kolektoru
    - `detector` ..... zdrojové kódy detektoru
    - `mux` ..... zdrojové kódy multiplexoru
    - `demux` ..... zdrojové kódy demultiplexoru
    - `test` ..... soubory pro testování
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Soukup_Dominik.pdf` ..... text práce ve formátu PDF
  - `ZadaniDP_Soukup_Dominik.pdf` ..... zadání práce ve formátu PDF