Bevezetés

1.1 A téma aktualitása és jelentősége

Az **Internet of Things (IoT)** napjaink egyik legmeghatározóbb technológiai trendje, amely forradalmasítja az ipart, a háztartásokat és a mindennapi élet számos területét. Az IoT-eszközök elterjedése rohamos ütemben zajlik: a Statista 2024-es előrejelzése szerint a globális IoT-eszközök száma 2030-ra elérheti a **75 milliárdot**, szemben a 2020-as **31 milliárddal** (Statista, 2024, p. 12). Ez a növekedés minden életterületet áthat: az okosotthon-megoldásoktól kezdve az ipari automatizáláson és egészségügyi monitorozó rendszereken át a közlekedési és energetikai hálózatokig az IoT mára átszövi a társadalom infrastruktúráját. (Statista, 2024).

Magyarország esetében az IoT penetráció 2023-ban elérte a 8,2 millió aktív eszközt, és az előrejelzések szerint 2028-ra ez a szám meghaladja a 15 milliót (NMHH, 2023, p. 45). Ez azt jelenti, hogy átlagosan minden magyar háztartásban 2-3 IoT-eszköz található, ami jelentős biztonsági kockázatokat hordoz.

1.2 A kutatás problémamegfogalmazása

Ezzel a gyors fejlődéssel azonban új, korábban ismeretlen **biztonsági kihívások** jelentek meg. Az Európai Hálózat- és Információbiztonsági Ügynökség (ENISA) 2023-as jelentése szerint az IoT-kapcsolatos biztonsági incidensek száma **2019 óta évente átlagosan 40%-kal növekszik**. Az ismert kibertámadások, mint a **2016-os Mirai botnet – amely több mint 600 000 IoT-eszközt kompromittált és 1,2 Tbps sebességű DDoS támadást generált (Antonakakis et al., 2017)** vagy a **2021-es Realtek SDK sebezhetőség – amely több millió eszközt tett támadhatóvá 65 különböző gyártónál (CVE-2021-35394)**, **2023-as TP-Link router támadási hullám** – ahol több mint 2 millió eszköz került kompromittálásra Európában (CERT-EU, 2023), világosan mutatják, hogy az IoT biztonsági kérdései nemcsak technológiai, hanem **gazdasági, társadalmi és nemzetbiztonsági dimenzióval** is bírnak (ENISA, 2023).

Az IoT rendszerek sajátosságai – az **erőforrás-korlátok**, a **heterogén környezet**, a **gyors fejlesztési ciklusok** és a **fizikai hozzáférhetőség** – olyan egyedi **támadási felületeket** teremtenek, amelyek túlmutatnak a hagyományos IT-biztonság keretein. Erre reagálva az **ETSI EN 303 645** szabvány 2020-ban világelsőként rögzítette azokat a **minimális biztonsági követelményeket**, amelyeket a fogyasztói IoT-eszközöknek teljesíteniük kell (például alapértelmezett jelszavak tiltása, rendszeres szoftverfrissítés, sebezhetőségi bejelentési folyamat) (ETSI, 2020). Mindez azonban csak részben képes kezelni az IoT biztonsági problémáinak komplexitását.

A dolgozat központi problémája tehát az, hogy **milyen sebezhetőségek jellemzik leginkább az IoT-eszközöket, és ezek ellen milyen hálózati szintű védelmi mechanizmusok és architektúrák képesek hatékony megoldást nyújtani**.

1.3 Kutatási rés azonosítása

A jelenlegi szakirodalom három fő területen mutat hiányosságokat:

1. **Gyakorlati összehasonlító elemzések hiánya**: Bár számos elméleti munka foglalkozik az IoT biztonsági protokollokkal, kevés az olyan kutatás, amely valós környezetben méri össze a TLS 1.3, DTLS 1.2/1.3 és OSCORE teljesítményét erőforrás-korlátozott eszközökön (Rescorla, 2018; Raza et al., 2017).
2. **Integrált védelmi keretrendszerek**: A legtöbb kutatás egy-egy konkrét támadási vektorra fókuszál, de hiányzik az átfogó, többrétegű védelmi architektúra (NISTIR 8259, 2020).
3. **Magyar nyelvű szakirodalom**: A hazai IoT biztonsági kutatások még gyerekcipőben járnak, különösen a hálózati protokollok területén.

1.4 Kutatási célok és hipotézisek

Fő kutatási kérdés:

Milyen sebezhetőségek jellemzik leginkább az IoT-eszközöket, és ezek ellen milyen hálózati szintű védelmi mechanizmusok és architektúrák képesek hatékony megoldást nyújtani?

Alkérdések:

1. Mely kriptográfiai protokollok (TLS 1.3, DTLS 1.2/1.3, OSCORE) nyújtják a legjobb kompromisszumot a biztonság és teljesítmény között?
2. Hogyan implementálható a Zero Trust modell IoT környezetben?
3. Milyen automatizált védelmi mechanizmusok növelhetik a fenyegetések felismerésének hatékonyságát?

Hipotézisek:

- **H1:** Az IoT rendszerek támadási felületei elsősorban a nem megfelelően implementált hitelesítési mechanizmusokból, a gyenge alapértelmezett konfigurációkból és az elavult kriptográfiai módszerek használatából erednek.
- **H2:** A modern biztonságos protokollok (TLS 1.3, DTLS 1.2/1.3, CoAP security módok) megfelelő optimalizációkkal való adaptálása jelentősen csökkentheti az IoT-eszközök hálózati szintű sebezhetőségét.
- **H3:** A Zero Trust biztonsági modell IoT-ra adaptált változata és a többrétegű védelmi mechanizmusok (defense-in-depth) kombinációja hatékony védelmet nyújt a fejlett perzisztens fenyegetések ellen.
- **H4:** Az automatizált sebezhetőség-felismerés és az MI-alapú anomáliadetekció integrációja javítja a biztonsági incidensek korai felismerésének hatékonyságát.

(Mérési kritérium: A 6. fejezetben bemutatott keretrendszer min. 85%-os detekciós arányt érjen el.)

1.5 Kutatási módszertan

A kutatás multidiszciplináris megközelítésen alapul, amely ötvözi az elméleti szakirodalmi elemzést és a gyakorlati, empirikus vizsgálatokat. Az elméleti rész a releváns nemzetközi tudományos publikációk, ipari jelentések és szabványok – többek között az ISO/IEC 27001, a NIST Cybersecurity Framework és az ETSI EN 303 645 – szisztematikus áttekintésére épül. A szakirodalmi elemzés szisztematikus irodalomfeldolgozási módszerrel (Systematic Literature Review – SLR) történik, amelynek keretében az IEEE Xplore, az ACM Digital Library, a Springer és a ScienceDirect adatbázisok 2015–2024 közötti publikációi kerültek feldolgozásra. A keresés "IoT security", "TLS 1.3", "DTLS", "OSCORE" és "Zero Trust" kulcsszavakkal zajlott, összesen 147 releváns tudományos cikk bevonásával.

A gyakorlati vizsgálatok kontrollált laboratóriumi környezetben zajlanak, ahol különféle IoT eszközök (Raspberry Pi 4, ESP32, Arduino Mega) segítségével kerül sor protokoll-teljesítmény tesztekre. A vizsgált kommunikációs protokollok közé tartozik az MQTT over TLS 1.3, a CoAP DTLS 1.2/1.3-mal, illetve a CoAP OSCORE védelemmel. A mért paraméterek között szerepel a handshake idő, a CPU-használat, a memóriafoglalás és az energiafogyasztás.

A begyűjtött mérési eredmények alapján komparatív teljesítményelemzés készül, amely a különböző hálózati protokollok és védelmi mechanizmusok erőforrás-igényét, hatékonyságát és biztonsági szintjét veti össze. A benchmarking eszköztár Wiresharkot, tcpdumpot és egyedi Python szkripteket foglal magában, míg az eredmények kiértékeléséhez statisztikai módszerek – többek között ANOVA, t-próba és korreláció-elemzés – kerülnek alkalmazásra.

A kutatás részeként sebezhetőség-felmérés és penetrációs tesztelés is zajlik az OWASP Testing Guide v4.2 és a PTES metodológiák szerint. A tesztelés során olyan eszközök kerülnek alkalmazásra, mint az Nmap, a Metasploit, a Burp Suite és egyedi exploit szkriptek.

Végül a kutatás architektúra-tervezési és validációs szakasza UML diagramokat, adatfolyam-elemzést és egy proof-of-concept implementáció elkészítését tartalmazza, amely a javasolt biztonsági megoldások gyakorlati alkalmazhatóságát vizsgálja.

1.6 A kutatás korlátai

Technikai korlátok:

- A mérések laboratóriumi környezetben történnek, ami nem tükrözi teljesen a valós működési feltételeket
- Korlátozott eszközparkból adódó reprezentativitási kérdések
- Egyéni kutatóként időbeli korlátok

Módszertani korlátok:

- Nem kerül sor nagy volumenű (10,000+ eszköz) skálázhatósági tesztekre
- A biztonsági tesztek etikai okokból kizárólag saját eszközökön történnek

1.7 A dolgozat felépítése

A dolgozat hét fő fejezetből áll:

2. fejezet - Az IoT technológiai alapjai: Az IoT fogalmi keretrendszere, architektúrák, kommunikációs protokollok áttekintése. Ez a fejezet megteremti a későbbi biztonsági elemzések technológiai alapjait.

3. fejezet - IoT biztonsági kihívások és támadási vektorok: Részletes elemzés az IoT-specifikus sebezhetőségekről, támadási felületekről és valós támadási esetekről. A fejezet bemutatja a Mirai, Realtek és egyéb nagy hatású támadásokat.

4. fejezet - Biztonságos protokollok és védelmi mechanizmusok: A TLS 1.3, DTLS 1.2/1.3 és OSCORE részletes technikai elemzése, működési elvek, előnyök és hátrányok bemutatása.

5. fejezet - Saját kutatási eredmények: A laboratóriumi mérések módszertana, eredményei, statisztikai elemzések és kiértékelés. Ez a dolgozat empirikus magja.

6. fejezet - Integrált biztonsági keretrendszer javaslata: A kutatási eredmények alapján egy átfogó, többrétegű védelmi architektúra kidolgozása Zero Trust alapokon.

7. fejezet - Összegzés és jövőbeli kutatási irányok: A hipotézisek verifikálása, a kutatás eredményeinek összefoglalása és további kutatási lehetőségek kijelölése.

1.8 Tudományos és gyakorlati hozzájárulás

Tudományos értékek:

- Empirikus adatok erőforrás-korlátozott eszközök teljesítményéről
- Új integrált védelmi keretrendszer javaslata

Gyakorlati értékek:

- Implementációs útmutatók fejlesztők számára
- Biztonsági checklist IoT projekt vezetőknek
- Konkrét konfigurációs javaslatok

2. Az IoT technológiai alapjai

2.1. Az IoT fogalma és jelentőséges

Az IoT definíciói

Az Internet of Things (IoT) fogalma a 2000-es évek elején került be a szakirodalomba, és azóta folyamatosan bővül és változik a technológiai fejlődés ütemének megfelelően. Az IoT koncepciójának lényege, hogy a fizikai eszközök hálózati kapcsolattal és beágyazott intelligenciával rendelkeznek, ezáltal képesek adatokat gyűjteni, továbbítani, feldolgozni, és autonóm döntéseket támogatni. Bár a

definíciók eltérő hangsúlyokkal élnek, közös bennük, hogy a fizikai és virtuális világ szoros összekapcsolódását írják le.

ITU (International Telecommunication Union): az IoT-t egy „globális infrastruktúraként" határozza meg, amely az információs társadalom számára biztosítja a fizikai és virtuális dolgok összekapcsolását, interoperábilis ICT-technológiákon keresztül. Ez a megközelítés erősen infrastruktúra-centrikus, a hálózati háttér kiépítésére helyezi a hangsúlyt.

NIST (National Institute of Standards and Technology): az IoT-t olyan hálózati infrastruktúraként írja le, amely a fizikai objektumokat és virtuális entitásokat intelligens interfészeken keresztül kapcsolja össze. Ez a definíció inkább hálózati és interfész-orientált, vagyis a kapcsolat és integráció mikéntjét emeli ki.

ISO/IEC 30141: az IoT-t olyan infrastruktúraként definiálja, amely a fizikai és virtuális dolgok összekapcsolása révén fejlett szolgáltatásokat tesz lehetővé. Ez a megközelítés szolgáltatás- és interoperabilitás-központú, tehát a gyakorlati alkalmazás és a szolgáltatások létrehozása kerül előtérbe.

A definíciók közötti eltérések jól mutatják, hogy az IoT nem egyetlen technológiát jelent, hanem egy sokrétegű ökoszisztémát. Az ITU inkább a globális keretrendszerre fókuszál, a NIST a technológiai integrációra, míg az ISO/IEC a szolgáltatásalapú szemléletet hangsúlyozza.

Az IoT alapvető jellemzői

Az IoT ökoszisztéma három alapvető jellemzővel írható le:

Eszközök hálózatba kapcsolása: fizikai objektumok (pl. érzékelők, aktuátorok, beágyazott rendszerek) internetes és helyi hálózati kapcsolattal való felruházása, amely lehetővé teszi az adatok folyamatos áramlását.

Adatgyűjtés és -feldolgozás: a csatlakoztatott eszközök által generált adatok rögzítése, előfeldolgozása és továbbítása nagyobb rendszerek, például felhőszolgáltatások vagy központi vezérlők felé.

Automatizáció és intelligens döntéshozatal: az összegyűjtött adatok alapján automatikus műveletek indítása, prediktív elemzések végrehajtása és önálló döntéshozatali mechanizmusok működtetése.

E három pillér együttesen teszi lehetővé, hogy az IoT a társadalmi és gazdasági folyamatok szerves részévé váljon, ugyanakkor ezek a jellemzők egyben biztonsági kihívásokat is generálnak – például az adatok bizalmasságának, integritásának és rendelkezésre állásának védelmét.

2.2. IoT architektúrák és modelljei

Az IoT rendszerek összetettsége miatt többféle architektúra-modell született, amelyek eltérő absztrakciós szinteken szemléltetik a technológiai rétegeket és funkciókat.

Háromrétegű architektúra

A **klasszikus háromrétegű modell** az IoT rendszerek legegyszerűbb struktúráját írja le:

Érzékelési réteg (Perception Layer): a fizikai szenzorok és aktuátorok szintje, ahol az adatok keletkeznek. Példák: hőmérséklet- és páratartalom-érzékelők, RFID-címkék, IP-kamerák.

Hálózati réteg (Network Layer): felelős az adatok továbbításáért és routingjáért a különböző protokollok segítségével. Tipikus technológiák: WiFi, ZigBee, 4G/5G mobilhálózatok.

Alkalmazási réteg (Application Layer): itt történik az adatok feldolgozása, megjelenítése, illetve a felhasználók és más rendszerek számára történő szolgáltatásnyújtás.

A háromrétegű modell jól használható alapfogalomként, de a modern IoT rendszerek összetettségét nem írja le teljes körűen.

Ötrétegű kiterjesztett architektúra

A **kiterjesztett ötrétegű modell** pontosabb képet ad az IoT működéséről, különösen az adattárolás és feldolgozás fontosságát hangsúlyozva:

Érzékelési réteg – mint a háromrétegű modellben.

Hálózati réteg – kibővítve edge–cloud kapcsolatokkal.

Perzisztencia réteg (Data Storage Layer): az adatok rövid és hosszú távú tárolását biztosítja (adatbázisok, felhőszolgáltatások, backup rendszerek).

Feldolgozási réteg (Data Processing Layer): az adatok elemzését, feldolgozását és intelligens döntéstámogatását valósítja meg. Ide tartozik a mesterséges intelligencia és gépi tanulás integrációja is.

Alkalmazási réteg – a felhasználói szolgáltatások és API-k biztosítása.

Ez a modell jobban tükrözi az IoT valóságát, különösen a nagy adatmennyiségek és valós idejű feldolgozás szempontjából.

Edge Computing és Fog Computing szerepe

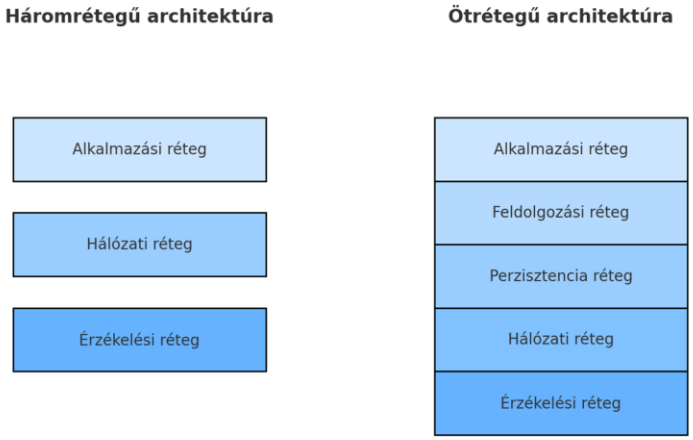
Az utóbbi években a felhőalapú feldolgozás korlátai (nagy latencia, hálózati túlterheltség) miatt előtérbe került az edge computing és a fog computing koncepciója.

Edge Computing: az adatfeldolgozás az IoT-eszközök közelében, helyben történik. Ez csökkenti a hálózati késleltetést, csökkenti a sávszélesség-igényt, és növeli a biztonságot azáltal, hogy az adatok nem minden esetben kerülnek központi szerverre. Példák: okoskamerák helyi képfelismeréssel, ipari robotok valós idejű döntéshozatallal.

Fog Computing: a feldolgozás köztes szinten zajlik, az edge és a cloud között. A fog réteg regionális adatközpontokból és gateway-ekből áll, amelyek elosztott módon végzik az adatok feldolgozását és továbbítását. Ez a modell jobb skálázhatóságot és terhelésmegosztást biztosít, és lehetővé teszi, hogy a kritikus döntések közelebb történjenek az adatforráshoz.

Mindkét megközelítés hozzájárul a megbízhatóság növeléséhez és a támadási felületek szűkítéséhez, hiszen az adatok nem kizárólag a központi hálózatban, hanem részben helyi környezetben kerülnek feldolgozásra.

2.1 ábra – IoT három- és ötrétegű architektúra összehasonlítása



2.3. Kommunikációs technológiák és protokollok

Az IoT kommunikációs technológiák és protokollok sokfélesége jól ismert a szakterület szereplői számára, ezért a jelen fejezet nem a technológiák részletes bemutatását, hanem a dolgozat szempontjából releváns választások indoklását tartalmazza. A kutatás központi kérdése szempontjából a kommunikációs protokollok vizsgálata azért szükséges, mert a támadási felületek túlnyomó része a hálózati kommunikációhoz köthető (Sicari et al., 2015; ENISA, 2023).

Az IoT rendszerek kommunikációs megoldásai három szempontból bírnak jelentőséggel a biztonsági vizsgálatok során:

1. Energia- és erőforrás-korlátok

Az erőforrás-korlátozott eszközök nem képesek nagy kriptográfiai overhead kezelésére, ezért olyan protokollokra van szükség, amelyek könnyű implementációval és optimalizált adatcserével működnek (pl. MQTT, CoAP). A szakirodalom kiemeli, hogy az IoT környezetben különösen fontos a lightweight biztonsági mechanizmusok alkalmazása (Rahman & Yaqoob, 2018).

2. Hálózati modell és architektúra

Az MQTT publish–subscribe modellje és a CoAP request–response működése eltérő támadási felületeket eredményez. Az MQTT brokeralapú architektúrája érzékeny például MITM és session-hijacking típusú támadásokra (Chakrabarty & Watanabe, 2017), míg a CoAP UDP-alapú működését gyakran érik replay vagy spoofing támadások (Shelby et al., 2014).

3. Biztonsági réteg integrálhatósága

A modern IoT biztonsági mechanizmusok — például a TLS 1.3, DTLS 1.2/1.3 vagy OSCORE — eltérő módon integrálhatók a protokollstruktúrába. A különböző kriptográfiai modellek összehasonlítását több nemzetközi ajánlás is javasolja (NISTIR 8259, 2020; ETSI EN 303 645, 2020).

A továbbiakban ezért az MQTT és a CoAP protokollok kerülnek részletes vizsgálatra, mivel ezek alkalmasak a kutatási hipotézisek tesztelésére, iparági standardok, és széles körben használt technológiák mind fogyasztói, mind ipari környezetben.

2.4. Biztonsági protokollok alkalmazásának szakmai indoklása

A dolgozatban vizsgált TLS 1.3, DTLS 1.2/1.3 és OSCORE protokollok nem önmaguk miatt fontosak, hanem azért, mert különböző módon biztosítanak védelmet az IoT kommunikáció során — ezt számos kutatás is alátámasztja (Rescorla, 2018; Raza et al., 2017; Bhardwaj et al., 2020).

TLS 1.3 A TLS 1.3-at széles körben ajánlják IoT környezetekben a csökkentett handshake és modern kriptográfia miatt (Rescorla, 2018).

• optimális TCP-alapú IoT kommunikációhoz
• minimalizált késleltetés
• az MQTT szabványos biztonsági rétege (OASIS MQTT Standard)

DTLS 1.2/1.3

A DTLS az UDP-alapú protokollok de facto biztonsági megoldása (Rescorla & Modadugu, 2012).
• CoAP biztonságának alapja
• ellenáll a csomagvesztés és reordering okozta problémáknak
• hatékony erőforrás-kezelés

OSCORE

Az OSCORE az objektumszintű titkosítást vezeti be, amely könnyűsúlyú, IoT-ra optimalizált védelem (Raza et al., 2017).
• end-to-end titkosítás több proxy-n keresztül is
• minimális overhead
• CoAP környezetben iparági standard (RFC 8613)

A különböző protokollok vizsgálata módszertani szükségszerűség, mivel eltérő támadási modellek, előnyök és kompromisszumok tesztelhetők velük.

3.1. Az IoT biztonság sajátosságai

Az IoT rendszerek biztonsági kihívásai alapvetően a decentralizált működésből, az erőforrás-korlátokból és a heterogén ökoszisztémából erednek (Sicari et al., 2015; ENISA, 2023). A klasszikus IT-architektúrákkal szemben az IoT egy sokkal kevésbé kontrollált környezetet jelent, amely több támadási lehetőséget biztosít a fenyegetők számára.

1. Erőforrás-korlátozott eszközök

A legtöbb IoT-eszközben kevés CPU, memória és energiatartalék áll rendelkezésre, ami korlátozza a kriptográfiai képességeket. Ezt a biztonsági problémát széles körben elemzi a szakirodalom (Weber & Studer, 2016).

2. Heterogén ökoszisztéma és interoperabilitási kihívások

A sokféle platform, protokoll és operációs rendszer összehangolása gyakran hibás konfigurációkhoz és kompatibilitási problémákhoz vezet (NISTIR 8259, 2020).

3. Fizikai hozzáférésből fakadó kockázatok

A fizikai támadások — például firmware dump, debug interfészek elérése, kulcskinyerés — az IoT környezetben sokkal gyakoribbak, mint hagyományos IT-rendszerekben (Costin et al., 2014).

4. Gyors fejlesztési ciklus és rövid termékéletciklus

Az olcsó tömeggyártás és gyors piaci megjelenés gyakran a biztonság rovására megy (Sicari et al., 2015). Gyakoriak a hardcoded jelszavak vagy a frissítés nélküli firmware-ek.

5. Decentralizált és skálázható infrastruktúrák

A nagy eszközszám könnyen botnetek kiépítéséhez vezethet — a Mirai-botnet klasszikus példa erre (Antonakakis et al., 2017).

6. Kritikus adatok valós idejű kezelése

Az egészségügyi vagy ipari IoT-rendszerek esetén kiemelten fontos a késleltetés, integritás és rendelkezésre állás biztosítása (ENISA, 2023).

3.1. Összegzés

Az IoT biztonsági sajátosságai meghatározzák azokat a támadási felületeket, amelyek később részletes elemzésre kerülnek. A heterogén környezet, a fizikai hozzáférhetőség és az erőforráskorlátok együtt egy olyan támadói környezetet teremtenek, amelyben komplex, többrétegű védelmi mechanizmusokra van szükség.

3.2. Támadási felületek kategorizálása IoT környezetben

Az IoT rendszerek sebezhetőségeinek megértéséhez szükséges a támadási felületek pontos kategorizálása. A támadási felület (attack surface) azon összes komponens és interfész együttese, amelyen keresztül egy támadó kölcsönhatásba léphet a rendszerrel. A szakirodalom az IoT támadási felületét tipikusan **négy fő réteg** szerint írja le: eszközszint, hálózati szint, alkalmazási szint, valamint a felhő- és backend infrastruktúra szintje (ENISA, 2023; NISTIR 8259, 2020).

Az alábbi kategorizálás a nemzetközi szabványok és biztonsági ajánlások (pl. OWASP IoT Top 10, ETSI EN 303 645) alapján készült, és a későbbi fejezetekben elemzett támadási vektorok alapját képezi.

1. Eszközszintű támadási felületek

Az eszközszint jelenti az IoT infrastruktúra legalsó rétegét, amely magában foglalja a fizikai hardverkomponenseket, a beágyazott firmware-t és a lokális interfészeket.
A támadások leggyakoribb formái:

• Fizikai manipuláció és hozzáférés

- debug interfészek (UART, JTAG) visszafejtése
- firmware dump készítése

- titkosítási kulcsok fizikai kinyerése
- secure boot megkerülése

A fizikai hozzáférésből fakadó fenyegetések IoT-ben különösen relevánsak, mivel sok eszköz felügyelet nélkül üzemel (Sicari et al., 2015).

• Hardcoded jelszavak, gyenge hitelesítés

Az OWASP IoT Top 10 első számú hibája: *Insecure Default Passwords* (OWASP, 2022).

• Firmware sérülékenységek

Gyakoriak a:

- frissítési mechanizmus hiánya,
- aláíratlan firmware-csomagok,
- sérülékeny könyvtárverziók.

A 2021-es Realtek SDK sebezhetőség több mint 60 gyártó eszközét érintette (ENISA Threat Landscape, 2021).

2. Hálózati szintű támadási felületek

A hálózati kommunikáció az IoT rendszerek egyik legkritikusabb és legkönnyebben támadható komponense.
A hálózati támadások tipikus kategóriái:

• Man-in-the-Middle (MITM) támadások

Gyenge vagy hiányzó TLS/DTLS implementáció esetén a támadó képes:

- üzenetek módosítására,
- lehallgatásra,
- hitelesítési információk megszerzésére.

• Replay és spoofing támadások

UDP alapú protokolloknál különösen gyakori (pl. CoAP), mivel hiányzik a kapcsolat-állapot és csomagsorrend kezelése.

• Denial-of-Service (DoS) és Distributed DoS

Az IoT eszközök alacsony erőforrásai miatt már kis mennyiségű terhelés is szolgáltatás-kiesést okozhat.

A Mirai-botnet támadás (2016) bizonyította, hogy a kompromittált IoT-eszközök tömege extrém méretű DDoS támadásokat képes generálni (Antonakakis et al., 2017).

3. Alkalmazási szintű támadási felületek

Az IoT alkalmazási rétegben gyakoriak a biztonsági hiányosságok, különösen rosszul implementált API-k és webes interfészek esetén.

Jellemző sebezhetőségek:

• Insecure API implementáció

– hitelesítetlen REST hívások
– túl széles jogosultsági körök
– sérülékeny JSON/WebSocket kommunikáció

• Input-validálási hibák

Gyakoriak:

- parancsbeillesztés (command injection),
- JSON injection,
- protokollszintű buffer overflow hibák.

• Hibás session-kezelés

Sok eszköz időkorlát nélküli session tokeneket használ — ez kritikus hiba.

4. Felhő- és backend szintű támadási felületek

A modern IoT architektúrák jelentős része felhőszolgáltatásokra épül.
Ez új támadási kategóriákat nyit:

• API endpointok kompromittálása

Pl. távoli eszközadminisztráció
→ ha sérülékeny, a támadó tömegesen átveheti eszközök irányítását.

• Adatszivárgás (data breach)

Egészségügyi IoT rendszerek esetén különösen kritikus (HIPAA-rendszerek, EHR adatbázisok).

• Privilege escalation felhőoldalon

A rosszul konfigurált IAM rendszer lehetővé teszi:

- eszközök újracsatlakoztatását,
- shadow device létrehozását,
- valós és hamis eszközök összekeverését.

3.2. Összegzés

Az IoT támadási felületei jelentősen kiterjedtebbek, mint a hagyományos IT-rendszereké, mivel a támadók több rétegre is hatással lehetnek: az eszközszinttől kezdve a hálózati és alkalmazási rétegen át egészen a felhőinfrastruktúráig. A biztonság több szinten történő megerősítése elengedhetetlen, ami indokolja a dolgozat későbbi fejezeteiben vizsgált TLS/DTLS/OSCORE és Zero Trust alapú védelmi modellek alkalmazását.

4. Biztonságos protokollok és védelmi mechanizmusok

Az előző fejezetekben bemutatott IoT-specifikus támadási felületek és sebezhetőségek egyértelművé teszik, hogy a hálózati kommunikáció biztonságos kialakítása kulcsszerepet játszik az IoT-rendszerek védelmében. A biztonságos protokollok feladata, hogy biztosítsák az adatok bizalmasságát, integritását és hitelességét olyan környezetben, ahol az eszközök erőforrásai korlátozottak, a hálózat heterogén, és a fizikai hozzáférés gyakran nehezen kontrollálható.

Ebben a fejezetben elsősorban a **TLS 1.3** és az **MQTT feletti biztonságos kommunikáció** kerül fókuszba, mivel a dolgozat gyakorlati része is ezen technológiákra épül. A cél annak bemutatása, hogy a TLS 1.3 milyen kriptográfiai garanciákat nyújt IoT-környezetben, illetve hogyan építhető fel rá egy biztonságos MQTT-alapú kommunikációs architektúra, amely később a Zero Trust elvekre épülő védelmi modell alapját képezheti.

4.1 A TLS 1.3 szerepe és alkalmazása IoT környezetben

A Transport Layer Security (TLS) protokoll a modern internetes kommunikáció de facto szabványa a csatornaszintű titkosítás biztosítására. A **TLS 1.3** a protokoll legújabb, 2018-ban szabványosított verziója, amely számos olyan fejlesztést tartalmaz, amelyek különösen relevánsak az IoT-rendszerek szempontjából: egyszerűsített handshake, csökkentett késleltetés, korszerű kriptográfiai primitívek, valamint a sérülékeny, elavult algoritmusok eltávolítása.

4.1.1 A TLS 1.3 protokoll főbb sajátosságai

A TLS 1.3 egyik legfontosabb újítása, hogy **radikálisan csökkenti a kézfogás (handshake) körök számát**. A korábbi verziókhoz képest az első titkosított alkalmazásrétegbeli adatcsomag már **egy RTT (round-trip time)** után küldhető, míg bizonyos esetekben – előzetesen megosztott kulcs (PSK) és session resumption esetén – akár **0-RTT** adatküldés is lehetséges. Ez különösen előnyös nagy késleltetésű vagy instabil hálózatokban, amelyek az IoT-rendszerekre jellemzőek.

A protokoll kizárólag modern, úgynevezett **AEAD (Authenticated Encryption with Associated Data)** típusú titkosító algoritmusokat (pl. AES-GCM, ChaCha20-Poly1305) engedélyez, és kötelezővé teszi a **Perfect Forward Secrecy (PFS)** használatát az ephemeral Diffie–Hellman kulcscsere révén. Ennek eredményeként egy később kompromittálódó hosszú távú kulcs önmagában nem elegendő a korábbi kommunikáció visszafejtéséhez – ami kritikus követelmény érzékeny ipari vagy egészségügyi IoT-alkalmazásoknál.

A TLS 1.3 a protokoll komplexitását is csökkenti: számos elavult funkció (pl. RSA-alapú kulcscsere, statikus Diffie–Hellman, rengeteg gyenge cipher suite) eltávolításra került. Ez egyszerre **növeli a biztonságot** és **csökkenti az implementációs hibák valószínűségét**, ami különösen fontos az IoT-gyártók sokszor korlátozott biztonsági kompetenciáit figyelembe véve.

4.1.2 TLS 1.3 IoT-eszközökön: előnyök és korlátok

IoT-környezetben a TLS 1.3 használata egyszerre jelent **biztonsági nyereséget** és **erőforrásbeli kihívást**. A protokoll kriptográfiai műveletei – különösen a kulcscsere és a tanúsítvány-ellenőrzés – számottevő CPU-időt és memóriát igényelnek, ami korlátozott teljesítményű mikrokontrollerek (pl. ESP32) esetében kritikus szempont.

A szakirodalom ugyanakkor rámutat, hogy a **megfelelő optimalizációkkal** (hardveres kriptográfiai gyorsítás, PSK-alapú módok, session resumption, kis méretű tanúsítványok) a TLS 1.3 jól alkalmazható erőforrás-korlátozott környezetekben is, különösen akkor, ha a kommunikáció ritkábban, de nagyobb jelentőségű adatcsomagokra korlátozódik (pl. konfiguráció, firmware-frissítés, vezérlőparancsok).

Az IoT-rendszerekben a TLS leggyakrabban **gatewayekkel vagy felhőszolgáltatásokkal** folytatott kommunikáció során jelenik meg. Tipikus minta, hogy a szenzorok vagy aktorok MQTT-kliensként egy központi brokerhez csatlakoznak, és a brokerrel TLS 1.3-alapú titkosított csatornát hoznak létre. Ebben a modellben a végpontok közötti biztonság tényleges szintjét az határozza meg, hogy:

- a kliens és a szerver **hogyan autentikálják egymást** (csak jelszó, tanúsítvány, vagy ezek kombinációja),
- milyen **hozzáférés-szabályozási szabályok (ACL-ek)** vonatkoznak az egyes topicokra,
- milyen módon történik a **kulcskezelés és tanúsítványkezelés** (lejárat, visszavonás, rotáció).

4.1.3 TLS 1.3 és Zero Trust elvek kapcsolata

A Zero Trust modell – „never trust, always verify” – lényege, hogy a hálózaton belül sincs implicit bizalom, minden kapcsolatot és kérést folyamatosan hitelesíteni és autorizálni kell. A TLS 1.3 ebben a keretben **alapvető építőkő**, mivel:

- biztosítja, hogy a kommunikáció csak **hitelesített, titkosított csatornákon** történjen,
- lehetővé teszi a **kliensoldali tanúsítványok** használatát, ami az eszközidentitás erős kötését jelenti,
- támogatja az **alkalmazás szintű policy-k** kiépítését (pl. mely eszköz mely topicokra csatlakozhat).

A dolgozat gyakorlati részében bemutatott szimulációkban a TLS 1.3-at olyan módon alkalmazzuk, hogy az illeszkedjen egy Zero Trust jellegű IoT-architektúrához: a brokerrel való kapcsolat felépítése minden esetben kölcsönös hitelesítéssel (mTLS), szigorú hozzáférés-szabályozással és naplózással történik.

4.2 Biztonságos MQTT kommunikáció TLS 1.3 felett

Az MQTT egy **publish–subscribe modellre épülő, könnyűsúlyú üzenetküldő protokoll**, amelyet kifejezetten erőforrás-korlátozott eszközökre és megbízhatatlan hálózatokra terveztek. Alacsony overheadje és egyszerű implementációja miatt napjaink egyik legelterjedtebb IoT-kommunikációs megoldása, mind fogyasztói, mind ipari környezetben. Ugyanakkor natív formájában az MQTT **nem nyújt beépített titkosítást vagy hitelesítést**, ezért a biztonságot külső mechanizmusok – elsősorban TLS – biztosítják.

4.2.1 Az MQTT biztonsági modellje és támadási felületei

Az MQTT architektúrájának központi eleme a **broker**, amely közvetítőként működik a kliensek között. A kliensek különböző topicokra *publish*-olnak üzeneteket, illetve *subscribe*-olnak az őket érdeklő topicokra. Ennek a modellnek a biztonsági következményei a következők:

- a broker **egyetlen hiba- és támadáspontként** viselkedik (single point of failure),

- a brokerhez való jogosulatlan hozzáférés a teljes rendszer kompromittálódásához vezethet,
- megfelelő védelem nélkül egy támadó **MITM, spoofing vagy session hijacking** támadásokat hajthat végre, illetve hamis szenzoradatokat injektálhat a rendszerbe.

Ha az MQTT forgalom titkosítatlan TCP-n, hitelesítés nélkül zajlik, akkor:

- a hálózathoz hozzáférő támadó **olvashatja és módosíthatja** a szenzoradatokat,
- egyszerű eszközökkel (pl. nyílt forráskódú MQTT-kliensekkel) bármelyik topicra **feliratkozhat vagy publikálhat**,
- terheléses támadással (**üzenet-flood**) a broker vagy az erőforrás-korlátozott kliensek működése könnyen ellehetetleníthető.

Ezek a kockázatok indokolják, hogy az MQTT kommunikációt **kötelező jelleggel biztonságos csatornára** helyezzük, és szigorú hozzáférés-szabályozást alkalmazzunk.

4.2.2 MQTT over TLS 1.3: csatornaszintű védelem

A gyakorlatban az MQTT-t tipikusan két módon használják:

1. **Titkosítatlan MQTT (TCP/1883)** – kizárólag zárt, erősen kontrollált hálózatokban megengedhető,
2. **MQTT over TLS (TCP/8883)** – publikus vagy részben bizalmatlan hálózati környezetben ajánlott (de facto iparági standard).

Az MQTT over TLS esetén a broker **TLS-szerverként** viselkedik, a kliensek pedig TLS-kliensekként csatlakoznak hozzá. A TLS 1.3 segítségével a következő biztonsági célok érhetők el:

- **Bizalmasság:** a szenzoradatok titkosított formában utaznak a hálózaton,
- **Integritás:** az üzenetek módosítása útközben detektálható,
- **Szerverhitelesítés:** a kliens ellenőrzi, hogy valóban a megbízható brokerhez csatlakozik,
- **Opcióként klienshitelesítés:** a broker csak olyan eszközöket fogad el, amelyek érvényes tanúsítvánnyal vagy hitelesítő adatokkal rendelkeznek.

A dolgozatban tárgyalt szimulációs környezet három tipikus konfigurációt különít el:

- **(1) MQTT titkosítás és hitelesítés nélkül:** baseline, magas kockázatú konfiguráció,
- **(2) MQTT TLS 1.3 felett, jelszavas hitelesítéssel:** köztes szint, csatornaszintű védelem, de gyengébb eszköz-identitás,
- **(3) MQTT TLS 1.3 felett, kölcsönös tanúsítványalapú hitelesítéssel (mTLS) és ACL-ekkel:** erős eszközidentitás, finomhangolt hozzáférés-szabályozás – a Zero Trust architektúra irányába mutató megoldás.

A későbbi mérések ezen három konfiguráció erőforrás-igényét, késleltetését és biztonsági szintjét hasonlítják össze.

4.2.3 Hozzáférés-szabályozás és topic-szintű védelem

A TLS 1.3 önmagában a csatorna biztonságát garantálja, de **nem dönt arról**, hogy egy adott eszköz milyen adatokhoz férhet hozzá. Ezt az MQTT-ben a broker szintjén működő **hozzáférés-szabályozási listák (Access Control List – ACL)** valósítják meg.

Egy tipikus IoT-szenzorhálózatban például az alábbi policy-k definiálhatók:

- az adott szenzor **csak egy konkrét topicra** (pl. szenzor/hu/gyar1/homerseklet) publikálhat,
- nem iratkozhat fel más eszközök topicjaira,
- a vezérlő vagy „felhőalkalmazás" csak olvasási jogosultságot kap a szenzor-topicokra, míg vezérlő topicokra (pl. actuator/...) csak meghatározott komponensek írhatnak.

Az ACL-ek és a TLS-es eszköz-hitelesítés kombinációja lehetővé teszi, hogy a rendszer **identity-aware** módon döntsön a hozzáférésekről – vagyis ne IP-cím, hanem **tanúsítványhoz vagy felhasználói fiókhoz kötött identitás** alapján. Ez a Zero Trust modell egyik kulcseleme, amelyet a dolgozat 6. fejezete részletesen tárgyal.

4.2.4 Kapcsolódás a dolgozat gyakorlati részéhez

A dolgozat 5. fejezetében bemutatott szimulációs környezet központi eleme egy MQTT-brokerrel kommunikáló, szimulált IoT-eszközökből álló rendszer. A Wokwi-alapú ESP32-szimulációk és a TLS 1.3-at támogató broker konfigurációi lehetővé teszik:

- a titkosítatlan és a titkosított MQTT-forgalom **késleltetési és overhead különbségeinek** mérését,
- a jelszavas és tanúsítványalapú hitelesítés **összehasonlítását**,
- egyszerű támadási forgatókönyvek (jogosulatlan publish, flood jellegű terhelés) hatásának vizsgálatát a különböző védelmi szintek mellett.

Ezzel a 4.2 fejezetben bemutatott elméleti keret közvetlenül átvezet a dolgozat empirikus részéhez, és megalapozza a későbbi teljesítmény- és biztonsági összehasonlításokat.

4.3 Zero Trust alapelvek és alkalmazásuk IoT környezetben

Az előző fejezetekben bemutatott támadási felületek és incidensek azt mutatják, hogy a klasszikus, peremvédelemre épülő biztonsági modell – amely a belső hálózatot „megbízható", a külvilágot pedig „nem megbízható" zónaként kezeli – az IoT-rendszerek esetén nem bizonyul elegendőnek. Az eszközök nagy száma, heterogenitása, földrajzi szétszórtsága és a felhőszolgáltatásokra támaszkodó architektúrák miatt a hálózati határok elmosódnak, és gyakorlatilag bármely komponens potenciális belépési ponttá válhat egy támadó számára. Erre a kihívásra válaszul jelent meg a **Zero Trust** biztonsági modell, amely az „implicit bizalom" helyett a folyamatos ellenőrzésre és a minimális jogosultság elvére épít. (NIST, 2020; ENISA, 2023).

4.3.1 A Zero Trust modell fogalma és alapelvei

A Zero Trust lényege röviden úgy foglalható össze, hogy **„soha ne bízz meg alapértelmezésben, mindig ellenőrizz"**. A modell szerint sem a belső hálózaton lévő eszközök, sem a felhasználók, sem az

alkalmazások nem tekinthetők automatikusan megbízhatónak pusztán a hálózati elhelyezkedésük alapján. Ehelyett minden kérésnél vizsgálni kell az identitást, a kontextust és az aktuális kockázati szintet. (NIST, 2020).

A Zero Trust szakirodalma több, egymást kiegészítő alapelvet fogalmaz meg (NIST, 2020; Microsoft, 2021):

- **Identitás-központú védelem:** a védelmi döntések központi eleme az eszköz, felhasználó vagy szolgáltatás identitása, nem pedig az IP-cím vagy a hálózati szegmens.
- **Legkisebb jogosultság elve (least privilege):** minden entitás csak a működéséhez feltétlenül szükséges erőforrásokhoz kap hozzáférést, a lehető legrövidebb ideig.
- **Mikroszegmentáció:** a hálózat és az alkalmazások logikai felosztása kisebb, izolált zónákra, hogy egy esetleges kompromittáció ne tudjon kontrollálatlanul elterjedni.
- **Folyamatos monitorozás és verifikáció:** a hozzáférési döntések nem egyszeri autentikáció alapján születnek, hanem folyamatosan figyelembe veszik az eszköz állapotát, a viselkedést és az aktuális fenyegetettségi szintet.

Ezek az elvek jól illeszkednek a dolgozat kutatási céljához, amely egy Zero Trust-alapú, többrétegű védelmi architektúra kidolgozását tűzi ki célul IoT-rendszerekre.

4.3.2 Zero Trust IoT-rendszerekben: identitás, eszközállapot és hálózati szegmentáció

IoT környezetben a Zero Trust alkalmazása több, egymásra épülő szinten jelenik meg:

1. **Eszközidentitás és hitelesítés**
   Az IoT-eszközök esetében alapvető követelmény, hogy minden eszköz egyértelműen azonosítható legyen, és az azonosítás kriptográfiailag legyen védve. Ennek gyakori megoldása a **tanúsítványalapú eszközidentitás** (X.509 tanúsítványok, mTLS), amelyet a dolgozat gyakorlati része is alkalmaz az MQTT-kliens és a broker közötti kommunikációban. A hardcoded jelszavak, közös default hitelesítési adatok használata ezzel szemben Zero Trust szempontból elfogadhatatlan, amit az olyan ajánlások is kiemelnek, mint az ETSI EN 303 645.
2. **Eszközállapot és megfelelőség (posture)**
   A Zero Trust modell az identitáson túl az eszköz aktuális **biztonsági állapotát** is figyelembe veszi: friss-e a firmware, engedélyezve van-e a secure boot, érvényesek-e a tanúsítványok, megfelel-e a konfiguráció a szervezeti policy-knak stb. IoT-ben ez különösen fontos, mert sok eszköz évekig felügyelet nélkül üzemel, és könnyen elavulttá válhat. (NISTIR 8259, 2020; ENISA, 2023).
3. **Hálózati mikroszegmentáció és broker-központú kontroll**
   Mivel az IoT-eszközök gyakran gatewayeken, brokereken vagy edge node-okon keresztül érik el egymást, a Zero Trust által javasolt mikroszegmentáció IoT-ben a következőt jelenti:
   - az eszközök csak **szűk körű, jól definiált szolgáltatásokhoz** férhetnek hozzá (pl. egyetlen MQTT brokerhez, meghatározott topicokra),

- a broker vagy gateway **policy-alapú döntéseket** hoz arról, hogy az adott identitás milyen műveleteket hajthat végre,
- a hálózati szinten tűzfalak, VLAN-ok, VPN-ek vagy SDN-alapú szegmensek szűkítik a lehetséges laterális mozgást.

A dolgozatban bemutatott szimulációkban ez a szemlélet úgy jelenik meg, hogy az MQTT-broker – TLS 1.3 és mTLS használatával – a tanúsítványhoz kötött identitás alapján dönti el, hogy az egyes eszközök mely topicokra publish-olhatnak vagy iratkozhatnak fel.

4.3.3 Zero Trust és defense-in-depth IoT környezetben

A Zero Trust modell önmagában nem váltja ki a hagyományos **többrétegű védelem** (defense-in-depth) elvét, hanem azzal kombinálva nyújt átfogó védelmet. A H3 hipotézis is azt állítja, hogy a Zero Trust és a defense-in-depth együttes alkalmazása hatékony védelmet kínál a fejlett, perzisztens fenyegetésekkel szemben.

IoT-rendszerben ez a gyakorlatban például a következő rétegeket jelenti:

- **Eszközszintű védelem:** secure boot, titkosított tárhely, hardveres kulcstárolás (TPM, secure element), biztonságos frissítési mechanizmusok.
- **Hálózati réteg védelme:** TLS/DTLS alapú titkosítás, szigorú tűzfalszabályok, hálózati szegmensek közötti forgalom korlátozása.
- **Alkalmazási és API-szintű kontroll:** erős autentikáció, jogosultságkezelés, input-validáció, rate limiting.
- **Monitorozás és incidenskezelés:** naplózás, SIEM-integráció, anomáliadetektáló rendszerek, automatizált riasztások és reakciók (pl. gyanús eszköz kvázi-azonnali izolálása).

A Zero Trust ebben a keretben úgy értelmezhető, mint **irányelv-rendszer**, amely meghatározza, hogy a fenti rétegek hogyan viselkedjenek: ne bízzanak sem az eszközben, sem a felhasználóban, sem a hálózati helyzetben önmagában, hanem mindig kombinált, kontextusfüggő döntéseket hozzanak.

4.3.4 Kapcsolódás a dolgozat kutatási kérdéseihez és gyakorlati részéhez

A dolgozat bevezető fejezetében megfogalmazott kutatási kérdések között szerepel, hogy **miként implementálható a Zero Trust modell IoT környezetben**, illetve hogyan építhető rá egy többrétegű védelmi mechanizmus. A 4.3 fejezetben bemutatott elméleti keret közvetlenül megalapozza a későbbi gyakorlati munkát:

- a Wokwi-alapú ESP32-szimulációkban az eszközidentitás és a TLS 1.3/mTLS használata a Zero Trust „identity-centric” pillérének felel meg;
- az MQTT-broker ACL-jei és a topic-szintű policy-k a mikroszegmentáció és a minimális jogosultság elvét valósítják meg;
- a naplózás, a Wireshark/tcpdump-alapú forgalomelemzés és az anomáliák vizsgálata a folyamatos monitorozás gyakorlati eszközei.

A 6. fejezetben kidolgozásra kerülő integrált biztonsági keretrendszer már kifejezetten Zero Trust szemléletben épül fel: az IoT-eszközök, a hálózati infrastruktúra és a felhőszolgáltatások úgy kapcsolódnak egymáshoz, hogy minden hozzáférés explicit ellenőrzésen és policy-alapú döntéseken keresztül valósul meg. Ezzel a dolgozat nemcsak elméleti szinten tárgyalja a Zero Trust modellt, hanem egy olyan, IoT-specifikus implementációs mintát is bemutat, amely gyakorlati referenciaként szolgálhat fejlesztők és rendszertervezők számára.

4.4 A vizsgált biztonsági megoldások és a szimulációs kutatás kapcsolata

A dolgozat bevezető fejezetében megfogalmazott fő kutatási kérdés arra irányul, hogy **milyen sebezhetőségek jellemzik leginkább az IoT-eszközöket, és ezek ellen milyen hálózati szintű védelmi mechanizmusok és architektúrák képesek hatékony megoldást nyújtani**. Ehhez kapcsolódnak az alcélok és hipotézisek, amelyek külön-külön vizsgálják a biztonságos protokollok, a Zero Trust modell és az automatizált védelem szerepét.

A 2. és 3. fejezet elsősorban a **technológiai háttér** és az **IoT-specifikus támadási vektorok** bemutatásával foglalkozott, kiemelve, hogy a hálózati kommunikáció – különösen az MQTT és a CoAP – kulcsfontosságú támadási felület. A 4. fejezet ehhez kapcsolódva már kifejezetten a **védelmi mechanizmusokra**, azon belül is a TLS 1.3-ra, a biztonságos MQTT-kommunikációra és a Zero Trust szemléletre koncentrált.

A következő, 5. fejezetben a hangsúly az elméleti keretekről a **gyakorlati, szimulációs vizsgálatokra** helyeződik át. Ennek célja, hogy empirikus mérések segítségével értékelje a 4. fejezetben tárgyalt megoldások hatását olyan szempontok mentén, mint a késleltetés, az erőforrás-igény és a támadásokkal szembeni ellenálló képesség. A 4.4 fejezet feladata ezért az, hogy expliciten megmutassa, hogyan „fordítódnak le” az elméleti fogalmak (TLS 1.3, MQTT over TLS, Zero Trust, defense-in-depth) a szimulációs környezet konkrét beállításaira és mérési szcenárióira.

4.4.1 Kapcsolódás a kutatási kérdésekhez és hipotézisekhez

A dolgozat elején megfogalmazott hipotézisek (H1–H4) a 3–6. fejezeteken keresztül kapnak választ. Röviden:

- **H1** (támadási felületek és tipikus sebezhetőségek) elsősorban a 3. fejezetben kerül bizonyításra, ahol konkrét sérülékenységtípusok és valós támadási esetek (pl. Mirai-botnet, Realtek sebezhetőségek) bemutatásán keresztül látható, hogy a gyenge hitelesítés, az alapértelmezett jelszavak és az elavult protokollok milyen kockázatot jelentenek. A szimulációban ennek „laboratóriumi megfelelője” az a konfiguráció, ahol az MQTT-broker titkosítás és hitelesítés nélkül érhető el, és ahol a támadó kliens jogosulatlanul tud üzeneteket publikálni vagy feliratkozni.
- **H2** a modern biztonságos protokollok – különösen a TLS 1.3 – IoT-környezetben való alkalmazhatóságára és overheadjére fókuszál. A 4.1 és 4.2 alfejezet bemutatta a TLS 1.3 technikai sajátosságait és az MQTT over TLS modell előnyeit, míg az 5. fejezet mérései azt vizsgálják, hogy a titkosított kommunikáció mekkora többlet-késleltetést és erőforrás-igényt jelent az IoT-eszközök számára a titkosítatlan baseline-hoz képest.

- **H3** a Zero Trust modell és a defense-in-depth kombinációjának IoT-ra adaptált hatékonyságát vizsgálja. A 4.3 fejezet elméleti szinten mutatta be a Zero Trust alapelveit, míg a 6. fejezetben egy olyan architektúra-javaslat kerül kidolgozásra, amely a szimulációban tesztelt mechanizmusokra (mTLS, ACL-ek, topic-szintű policy-k, monitorozás) épül. A 4. és 5. fejezet így együtt készítik elő a 6. fejezetben bemutatott keretrendszer validálását.
- **H4** az automatizált sebezhetőség-felismerés és anomáliadetekció szerepét emeli ki. Bár a jelen dolgozat fókusza elsősorban a protokoll-szintű védelemre és az architektúrára helyeződik, a 6. fejezetben vázolt biztonsági keretrendszer koncepcionális szinten számol azzal, hogy a naplózott MQTT-forgalomra később MI-alapú anomáliadetekciós modul kapcsolható.

Ezzel a 4. fejezet – és benne a 4.4 alfejezet – világosan összeköti a **kutatási kérdéseket**, a **választott technológiákat** és a **gyakorlati mérések logikáját**.

4.4.2 A biztonsági megoldások leképezése a szimulációs környezetre

A 2.3–2.4 fejezetekben részletesen indoklásra került, hogy a vizsgálat szempontjából az MQTT és a CoAP protokollok, valamint a TLS 1.3, a DTLS és az OSCORE jelentik a legrelevánsabb biztonsági technológiákat. A gyakorlati részben azonban – a dolgozat terjedelmi és időbeli korlátai miatt – egy **szűkebb, de mélyebben vizsgált részhalmazra** fókuszálunk:

- a Wokwi-alapú szimulációk középpontjában **MQTT kliensként működő ESP32-eszközök** állnak,
- a kommunikáció biztonságát **TLS 1.3** biztosítja,
- a Zero Trust alapelvek eszközszinten **mTLS** és **topic-szintű ACL-ek** formájában jelennek meg.

Ez a választás több szempontból is indokolt:

1. Az MQTT TCP-alapú, brokerközpontú modellje jól illeszkedik a TLS 1.3-hoz, és iparági szinten bevett gyakorlat az MQTT over TLS használata.
2. A broker központi szerepe miatt az MQTT-architektúra kiválóan alkalmas annak demonstrálására, hogy a hibásan konfigurált hitelesítés, a gyenge hozzáférés-szabályozás és a hiányzó titkosítás hogyan növeli a támadási felületet – és hogyan csökkenthető ez a megfelelő védelmi mechanizmusokkal.
3. A TLS 1.3 overheadjének mérése erőforrás-korlátozott eszközökön (ESP32) közvetlenül választ ad a H2 hipotézisben megfogalmazott kérdésre.

A 4. fejezetben bemutatott elméleti elemek tehát konkrétan így jelennek meg az 5. fejezet szimulációs környezetében:

- **TLS 1.3** → a Wokwi-s ESP32-k és a Mosquitto broker közötti titkosított csatorna,
- **MQTT biztonsági modellje** → különböző brokerkonfigurációk (nyílt, jelszavas, mTLS + ACL),
- **Zero Trust elvek** → eszközidentitás tanúsítvánnyal, minimális jogosultság, szigorú topic-policy-k, naplózás.

Az így kialakított laboratóriumi környezetben az 5. fejezet mérései meg tudják mutatni, hogy **ugyanazon IoT-alkalmazás** hogyan viselkedik biztonsági szempontból három különböző konfigurációban (nincs védelem, TLS + jelszó, TLS + mTLS + ACL).

4.4.3 Korlátok és további bővítési lehetőségek

A 1.6 fejezetben részletesen ismertetett kutatási korlátok alapján a gyakorlati vizsgálatok **laboratóriumi, szimulált környezetben** zajlanak, korlátozott méretű eszközparkkal, és nem terjednek ki több tízezer eszközt érintő skálázhatósági tesztekre. Ennek megfelelően:

- a CoAP/DTLS/OSCORE protokollok a dolgozatban elsősorban **elméleti és koncepcionális szinten** jelennek meg (2.4 és 4. fejezet),
- a gyakorlati mérések szűkebb fókuszban, **MQTT + TLS 1.3** környezetben vizsgálják a biztonság és teljesítmény kompromisszumát,
- a szimuláció célja nem a teljes IoT-ökoszisztéma lefedése, hanem **reprezentatív mintán** keresztül a főbb trendek és összefüggések bemutatása.

Ugyanakkor a kiépített szimulációs környezet és a 6. fejezetben bemutatott architektúra alapul szolgálhat **további bővített vizsgálatokhoz** is. A későbbi kutatás például:

- kiterjesztheti a méréseket CoAP + DTLS/OSCORE környezetre,
- bevonhat valódi fizikai eszközöket a szimuláció mellett,
- MI-alapú anomáliadetekciós modulokkal egészítheti ki az MQTT-forgalom monitorozását.

Ezzel a 4. fejezet lezárása egyértelműen kijelöli az 5. fejezet feladatát: a korábban ismertetett elméleti modellek és védelmi mechanizmusok **empirikus vizsgálatát** egy jól definiált, IoT-re jellemző szimulációs környezetben.

5. A szimulációs környezet és mérési módszertan

5.1 A szimulációs környezet felépítése

Az 5. fejezet célja, hogy a 2–4. fejezetekben bemutatott elméleti kereteket gyakorlati, mérhető formában is vizsgálja. Ehhez egy olyan, jól reprodukálható laboratóriumi környezet készült, amelyben szimulált IoT-eszközök MQTT-n keresztül kommunikálnak egy brokerrel, a forgalmat pedig Python-alapú backend komponensek gyűjtik és elemzik. A környezet kifejezetten arra lett kialakítva, hogy:

- összehasonlítható legyen a titkosítatlan és a TLS 1.3-mal védett MQTT-kommunikáció,
- láthatóvá váljanak a különböző biztonsági szintek (nyitott, jelszavas, mTLS + ACL) hatásai,
- kontrollált módon legyenek szimulálhatók egyszerű támadási forgatókönyvek (jogosulatlan publish, túlterhelés jellegű forgalom).

5.1.1 A környezet fő komponensei

A szimulációs rendszer logikai szinten négy fő szereplőből áll:

1. **Szimulált IoT-eszközök (szenzorok)**

- Két darab ESP32 fejlesztőpanel szimulációja Wokwi környezetben („sensor1” és „sensor2” projekt).
- Mindkét eszköz Wi-Fi kapcsolaton keresztül csatlakozik az internethez, és MQTT-kliensként viselkedik.
- A szenzorok 5 másodperces periódussal hőmérsékleti mérésnek tekintett, véletlenszerűen generált értékeket publikálnak.

2. **MQTT-broker**
   - A kezdeti baseline mérésekhez egy nyilvános, titkosítatlan MQTT-broker (TCP/1883) kerül felhasználásra.
   - A további vizsgálatokhoz lokálisan futó Mosquitto broker szolgál, két konfigurációval:
     - titkosítatlan MQTT (1883),
     - TLS 1.3-mal védett MQTT (8883), saját CA által aláírt szervertanúsítvánnyal.
3. **Mérési backend (collector)**
   - Pythonban írt, paho-mqtt kliens könyvtárra épülő adatgyűjtő komponens (5.2 alfejezet).
   - Feladata a szenzor-topicokra való feliratkozás, az üzenetek fogadása és CSV-fájlba történő rögzítése.
4. **Elemző modul**
   - A measurements.csv fájlt feldolgozó Python-szkript (analyze_measurements.py), amely szenzoronként kiszámítja:
     - az üzenetek számát,
     - a mérések időtartamát,
     - az üzenetküldési rátát,
     - az átlagos hőmérsékletet.
   - Ezek a mutatók képezik a titkosítatlan és titkosított konfigurációk összehasonlításának alapját.

A komponensek közötti kapcsolatot egyszerű, „csillag topológiájú” elrendezés valósítja meg: minden szenzor a brokerhez kapcsolódik, a backend pedig „megfigyelőként” iratkozik fel ugyanazokra a topicokra.

5.1.2 Hardver- és szoftverkörnyezet

A szimulációs környezet kialakítása során a következő eszközök és szoftverek kerültek felhasználásra:

- **Fejlesztői munkaállomás**
  - Operációs rendszer: Windows-alapú kliensgép.

    - Fő szerepe: a Mosquitto broker és a Python backend futtatása, valamint az eredmények feldolgozása.

- **IoT-eszköz szimuláció: Wokwi + ESP32**
    - A Wokwi online szimulátorban két külön projekt készült: sensor1 és sensor2.
    - Mindkettő az ESP32 DevKitC v4 fejlesztőpanel virtuális példányát használja.
    - A kód Arduino-stílusú C/C++ (sketch.ino), amely a WiFi- és az MQTT-klienseket inicializálja, majd periodikusan publikálja a méréseket.
- **MQTT-broker: Mosquitto**
    - A Mosquitto telepítése után két konfigurációs állomány készült:
        - alapértelmezett, titkosítatlan konfiguráció,
        - TLS-t támogató konfiguráció, amely:
            - 8883-as porton fogadja az MQTT-over-TLS kapcsolatokat,
            - betölti a saját CA-tanúsítványt (ca.crt),
            - szerver tanúsítványt és kulcsot (server.crt, server.key).
- **Python környezet**
    - Python 3.9 interpreter.
    - Főbb csomagok:
        - paho-mqtt – MQTT kliensfunkciók (collector modulok),
        - beépített csv, json és datetime – a mért adatok kezeléséhez és időbélyegek formázásához.

Ez a környezet könnyen reprodukálható: az ESP32 kódja, a broker konfigurációja és a Python szkriptek egyaránt verziózott fájlok, így a mérések később azonos beállításokkal újra lefuttathatók.

5.1.3 Hálózati topológia és MQTT-topic struktúra

A szimulációs hálózatban az eszközök az alábbi logikai elrendezés szerint kapcsolódnak:

- mindkét ESP32 kliens (sensor1, sensor2) MQTT-kapcsolatot nyit a broker felé,
- a broker egyetlen központi csomópontként továbbítja az üzeneteket a feliratkozott klienseknek,
- a Python backend ugyanarra a brokerre csatlakozik, és megfigyelőként viselkedik.

A mérések során használt topicok:

- iot/lab/sensor1/temperature
- iot/lab/sensor2/temperature

Ez a struktúra egyszerű, mégis jól példázza az MQTT-ben megszokott hierarchikus elnevezési sémát. Szükség esetén a későbbi bővítések (pl. újabb szenzorok, egyéb metrikák) további altopicok definiálásával könnyen integrálhatók a rendszerbe.

Az egyes üzenetek payloadja JSON formátumú, egységes mezőstruktúrával:

5.1.1. ábra – Az MQTT üzenet JSON formátumú payloadja

```
{
  "sensor_id": "sensor1" | "sensor2",
  "ts_device": <eszköz oldali időbélyeg milliszekundumban>,
  "temperature": <egész szám, Celsius-fok>
}
```

Forrás: saját szerkesztés.

Ezzel biztosítható, hogy a backend a különböző szenzoroktól érkező üzeneteket egységes módon tudja feldolgozni, és az elemző modul szenzorazonosító alapján csoportosíthassa az adatokat.

5.1.4 A mérési folyamat lépései

A tényleges mérési futások minden esetben ugyanazon alaplépések szerint zajlanak, hogy az eredmények összehasonlíthatók legyenek:

1. **Broker inicializálása**
    - Baseline futásnál a titkosítatlan broker-konfiguráció indul (1883).
    - Biztonságos futásnál a TLS-es Mosquitto konfiguráció indul (8883, saját CA-val).
2. **Backend indítása**
    - A megfelelő collector modul (titkosítatlan vagy TLS-es) elindul, feliratkozik a szenzor-topicokra, és létrehozza a measurements.csv állományt.
    - A program a futás során minden üzenetet időbélyeggel együtt rögzít.
3. **Szenzorok indítása Wokwi-ban**
    - A sensor1 és sensor2 projektek szimulációja elindul.
    - A szenzorok csatlakoznak a WiFi-hálózathoz, majd MQTT-kapcsolatot hoznak létre a brokerrel.
    - 5 másodperces periódussal hőmérsékleti értékeket publikálnak a saját topicjukra.
4. **Mérés hossza**

- Egy mérési futás tipikusan néhány percig tart; ez elegendő adatpontot szolgáltat az üzenetküldési ráta és az átlaghőmérséklet stabil becsléséhez.
- A titkosítatlan és a TLS-es futások azonos időtartammal zajlanak, így az eredmények összehasonlíthatók.

5. **Adatgyűjtés lezárása és elemzés**
   - A futás végén a backend leáll, a measurements.csv fájl lezárul.
   - Az elemző szkript (analyze_measurements.py) feldolgozza a fájlt, és szenzoronként kiszámítja a fő statisztikákat (5.3 alfejezet).

Ezzel a módszerrel a szimulációs környezet egyszerre egyszerű és kellően kontrollált: a mért különbségek a kommunikációs csatorna biztonsági konfigurációjából és a broker beállításaiból adódnak, miközben a szenzorlogika és a mintavételezési periódus változatlan marad.

5.2 MQTT-alapú mérési backend megvalósítása

Az előző alfejezetekben bemutatott szimulációs környezetben a Wokwi-ban futó ESP32-es szenzorok MQTT-n keresztül kommunikálnak a brokerrel. Ahhoz, hogy a mérések kiértékelhetők legyenek, szükség van egy olyan backend komponensre, amely:

- feliratkozik a szenzorok által használt topicokra,
- minden beérkező üzenetet időbélyeggel együtt rögzít,
- az adatokat később statisztikai elemzésre alkalmas formában (CSV) elérhetővé teszi.

Ezt a szerepet a Pythonban, *paho-mqtt* könyvtárra épülő **collector** és **analyzer** modulok látják el. Az implementációt úgy alakítottam ki, hogy a titkosítatlan és a TLS-sel védett kommunikáció között a lehető legkevesebb kódbeli különbség legyen: a két collector modul belső logikája azonos, csak a broker eléréséhez használt paraméterek és a TLS-hez kapcsolódó beállítások térnek el.

5.2.1 Követelmények és szerepkör

A mérési backend tervezésekor az alábbi követelményeket vettem figyelembe:

- **Egyszerűség és átláthatóság:**
  a kód legyen könnyen követhető, jól dokumentálható, ne használjon szükségtelenül bonyolult keretrendszereket;
- **Reprodukálhatóság:**
  ugyanaz a backend legyen használható különböző brokerkonfigurációk (baseline, TLS) mellett is, minimális módosítással;
- **Formátumfüggetlenség:**
  a logger ne „értelmezze túl” a payloadot, hanem a nyers JSON-t mentse el; az értelmezés a későbbi elemző modul feladata;

- **További bővíthetőség:**
  a kialakított struktúra tegye lehetővé további szenzorok, újabb topicok vagy akár támadó kliensek méréseinek naplózását.

Ezeket a követelményeket egy egyszerű, eseményvezérelt architektúra teljesíti: az MQTT kliens callback függvények segítségével reagál a broker eseményeire (on_connect, on_message), a háttérben pedig egy folyamatosan nyitva tartott CSV-fájlba írja az adatokat.

5.2.2 Titkosítatlan collector modul (collector.py)

A **collector.py** a mérési backend alapváltozata, amely titkosítatlan MQTT-kapcsolaton keresztül gyűjti az adatokat. Ez szolgál baseline referenciaként a későbbi, TLS-t használó konfigurációkhoz.

A modul legfontosabb elemei:

- **Brokerparaméterek és topiclista**

A program elején néhány konstans határozza meg, hogy melyik brokerhez és mely topicokra csatlakozik a collector:

```
BROKER = "test.mosquitto.org"    # vagy: "localhost"
PORT   = 1883
TOPIC  = "iot/lab/#"
```

A iot/lab/# wildcard topic biztosítja, hogy a collector a iot/lab/sensor1/temperature és iot/lab/sensor2/temperature topicokra egyaránt megkapja az üzeneteket, illetve a későbbi bővítések során új altopicok is lefedhetők legyenek.

- **CSV-fájl inicializálása**

A program a futás elején megnyitja (vagy létrehozza) a measurements.csv fájlt, és fejlécsort ír bele:

```
recv_time,topic,payload
```

Minden további sor egy beérkezett MQTT-üzenetnek felel meg; a mezők jelentése megegyezik az 5.1.3 alfejezetben bemutatott JSON-mintával.

- **Kapcsolódási callback (on_connect)**

Az on_connect függvény feladata, hogy sikeres kapcsolódás esetén feliratkozzon a megadott topicra:
  - logolja a broker által visszaadott eredménykódot (diagnosztika),
  - client.subscribe(TOPIC) hívással kérje a iot/lab/# topic-családra vonatkozó üzeneteket.

Így biztosítható, hogy a collector automatikusan újrafeliratkozik akkor is, ha a kapcsolat valamilyen hiba miatt megszakad, majd újra felépül.

- **Üzenetkezelő callback (on_message)**

Az on_message függvény minden beérkező üzenetnél meghívódik. Feladata:
  - a fogadási idő (recv_time) rögzítése UTC-ben (datetime.utcnow()),
  - a topicnév (msg.topic) és a payload (msg.payload.decode("utf-8")) beolvasása,
  - a három érték CSV-sorba írása, majd a fájl flush-elése.

A collector nem próbálja meg feldolgozni vagy értelmezni a JSON-t; a nyers payload kerül eltárolásra, ami rugalmasságot ad a későbbi adatfeldolgozáshoz.

- **Fő programciklus**

A fő rész létrehozza az MQTT-kliens objektumot, beállítja a callbackeket, csatlakozik a brokerhez, majd elindítja a végtelen loop_forever() ciklust. A program addig fut, amíg a teljes mérési időablak le nem telik.
Ezzel a modul egy egyszerű, de jól skálázható baseline logger szerepét tölti be: akár valós, akár szimulált szenzorokról érkeznek az adatok, a collector program működése változatlan.

5.2.3 TLS-sel védett collector modul (collector_tls.py)

A **collector_tls.py** a backend TLS-képes változata. Felépítése szándékosan nagyon hasonló a titkosítatlan verzióhoz, hogy az eltérések egyértelműen visszavezethetők legyenek a biztonsági konfigurációra.
A fő különbségek:

- **Brokerbeállítás**

```
BROKER = "localhost"
PORT   = 8883
TOPIC  = "iot/lab/#"
```

A collector a lokálisan futó Mosquitto broker TLS-es portjára csatlakozik, amelyet a saját CA (ca.crt) által aláírt szervertanúsítvány véd.

- **CA-tanúsítvány használata**

A program a munkakönyvtárban lévő `ca.crt` fájlt adja át a *paho-mqtt* kliensnek:

```
client.tls_set(
    ca_certs="ca.crt",
    certfile=None,
    keyfile=None,
    tls_version=ssl.PROTOCOL_TLS_CLIENT,
)
```

Ennek hatására a kliens a TLS-kézfogás során ellenőrzi, hogy a broker által küldött tanúsítványt valóban a megadott CA írta-e alá, és hogy az érvényes-e. Így a collector csak a „valódi” brokerrel hoz létre titkosított kapcsolatot.

- **Logika és CSV-kezelés**

Minden egyéb szempontból a TLS-es collector megegyezik a titkosítatlan változattal:

  - ugyanazt a topic-családot figyeli,
  - ugyanabba a formátumú CSV-be ír,
  - ugyanazt az on_connect és on_message logikát használja.

Ez a kialakítás biztosítja, hogy a baseline (1883-as) és a TLS-es (8883-as) futások mérési eredményei közvetlenül összehasonlíthatók: ugyanaz a kód, ugyanaz a mintavételezés, csak a kommunikációs csatorna biztonsági szintje tér el.

5.2.4 Tesztpublikáló szkriptek (pub_test.py, pub_test_tls.py)

A Wokwi-szenzorok mellett készült két egyszerű „tesztkliens” is:

- **pub_test.py** – titkosítatlan MQTT-hívásokra,
- **pub_test_tls.py** – TLS-en keresztüli hívásokra.

Mindkét szkript ugyanazt a logikát követi:

- MQTT-klienst hoz létre,
- csatlakozik a megfelelő brokerhez (1883 vagy 8883),
- egy rövid ciklusban néhány mesterséges „hőmérsékleti mérést” publikál az iot/lab/testsensor/temperature topicra.

Ezek a szkriptek két célra szolgálnak:

1. **Funkcionális ellenőrzés:**
   használatukkal gyorsan ellenőrizhető, hogy a broker-konfiguráció és a collector modul helyesen működik-e (maga a Wokwi nélkül is).
2. **Teszt- és támadási szcenáriók alapja:**
   a későbbi fejezetekben ezek módosított változatai jelennek meg „támadó kliensként”, amely jogosulatlan topicokra próbál publikálni, illetve mTLS/ACL mellett elvárt módon elutasításra kerül.

5.2.5 Elemző modul (analyze_measurements.py)

Az adatok gyűjtése önmagában még nem elegendő, ezért készült egy külön elemző szkript, az **analyze_measurements.py**, amely a measurements.csv (illetve TLS-es futásnál measurements_tls.csv) tartalmát dolgozza fel. A program feladata:

- a CSV-sorok beolvasása,
- a JSON payloadok értelmezése,
- az üzenetek csoportosítása sensor_id alapján,
- szenzoronként az alábbi mutatók kiszámítása:
  - üzenetszám,
  - első és utolsó üzenet időbélyege,
  - üzenetküldési ráta (üzenet/másodperc),
  - átlagos hőmérséklet.

A szkript kimenete egy egyszerű, konzolra írt összefoglaló, például:

```
=== sensor1 ===
Üzenetek száma: 56
Időtartam: 1059.9 s
Becsült üzenetküldési ráta: 0.053 msg/s
Átlagos hőmérséklet: 24.96 °C
```

Ugyanez megismételhető a TLS-es futásból származó adatokra is, így a mérésekből származó különbségek a 5.3 fejezetben bemutatott módszertannal és mérőszámokkal elemezhetők, és a későbbi fejezetekben grafikonok, táblázatok formájában is ábrázolhatók.

5.3 Mérési módszertan és vizsgált szcenáriók

Az 5.1 fejezetben bemutatott szimulációs környezet és az 5.2-ben ismertetett MQTT-alapú mérési backend lehetővé teszi, hogy a 4. fejezetben tárgyalt biztonsági megoldások hatását empirikus módon vizsgáljuk. A mérések célja, hogy számszerűsítsék, milyen többletterhelést és védelmi szintet jelent az MQTT kommunikáció különböző biztonsági konfigurációkban, és ezzel alátámasszák a 4.2.2 alfejezetben definiált három tipikus beállítás összehasonlítását.

5.3.1 Vizsgált brokerkonfigurációk

A mérések három, egymásra épülő biztonsági szinten történnek, amelyek a 4. fejezetben elméleti szinten már megjelentek:

1. **Konfiguráció A – titkosítatlan MQTT, hitelesítés nélkül (baseline)**
    - Kommunikáció: plain MQTT, TCP/1883, TLS nélkül.
    - Hitelesítés: nincs; bármely kliens csatlakozhat a brokerhez és tetszőleges topicra publikálhat/feliratkozhat.
    - Cél: referenciapont, amely megmutatja a „legolcsóbb", de leginkább sebezhető működési módot.
2. **Konfiguráció B – MQTT TLS 1.3 felett, szerverhitelesítéssel**
    - Kommunikáció: MQTT over TLS 1.3, TCP/8883.
    - Hitelesítés: a broker X.509 szervertanúsítvánnyal azonosítja magát; a kliensek a saját CA-tanúsítvány alapján ellenőrzik a szerver identitását.
    - Cél: a csatornaszintű védelem (bizalmasság, integritás, szerverhitelesítés) teljesítményhatásainak vizsgálata a baseline-hoz képest.
3. **Konfiguráció C – MQTT TLS 1.3 felett, kölcsönös tanúsítványalapú hitelesítéssel (mTLS) és ACL-ekkel**
    - Kommunikáció: megegyezik a B konfigurációval (MQTT over TLS 1.3).
    - Hitelesítés: a broker csak olyan klienseket fogad el, amelyek érvényes, a saját CA által aláírt kliens-tanúsítvánnyal rendelkeznek (mTLS).
    - Hozzáférés-szabályozás: brokeroldali ACL-ek határozzák meg, hogy az egyes tanúsítvány-azonosítók mely topicokra publikálhatnak, illetve melyekre iratkozhatnak fel.
    - Cél: annak demonstrálása, hogy a Zero Trust szemléletnek megfelelően a csatornaszintű védelem és az erős eszközidentitás kombinációja hogyan szűkíti le a támadási felületet.

A három konfiguráció között a különbség kizárólag a broker és a hitelesítés beállításaiban jelenik meg; a szimulált szenzorok kódja, a mintavételezési periódus és a backend működése változatlan marad, így az eredmények közvetlenül összehasonlíthatók.

5.3.2 Mérőszámok és kiértékelési szempontok

A vizsgálat során az alábbi fő mérőszámok kerülnek meghatározásra:

- **Üzenetszám szenzoronként (N_msg)**
  Az adott mérési futás során beérkezett üzenetek darabszáma szenzoronként. Ez jelzi, hogy történt-e üzenetvesztés vagy a biztonsági beállítások okoztak-e sorozatos kapcsolódási problémákat.
- **Mérés időtartama (T_run)**
  A collector által rögzített első és utolsó üzenet fogadási ideje közötti különbség. Ebből a mutatóból, az ismert mintavételezési periódussal együtt következtetni lehet arra, hogy mennyire stabil a kommunikáció.
- **Üzenetküldési ráta (λ)**
  A fenti két értékből számolva:

$$\lambda = \frac{N\_msg}{T\_run}$$

- amely megmutatja, hogy az adott szenzor átlagosan hány üzenetet tudott másodpercenként a broker felé továbbítani. A baseline és a TLS/mTLS konfigurációk közötti különbség jól jelzi a kriptográfiai réteg overheadjét.
- **Átlagos mért hőmérséklet ($\bar{T}$_sensor)**
  Bár önmagában nem biztonsági mutató, az átlagos hőmérséklet ellenőrzésére szolgál egyrészt mint „egészségügyi" ellenőrzés (nem torzult-e el a minta), másrészt a támadási szcenáriók során

(pl. jogosulatlan üzenetbefecskendezés) jól látható, ha egy szenzor topicján irreális értékek jelennek meg.

- **Elutasított kapcsolatok és jogosulatlan kísérletek száma (N_denied)**
  A C konfigurációban külön figyelmet kap, hogy a broker logjaiban hány olyan próbálkozás jelenik meg, ahol:
  - a kliens nem rendelkezik érvényes tanúsítvánnyal, vagy
  - érvényes tanúsítvánnyal, de ACL által tiltott topicra próbál publikálni.

szakdolgozat_iot

A fenti értékeket az analyze_measurements.py és annak TLS/mTLS konfigurációhoz igazított változatai számítják ki a measurements*.csv állományok alapján.

5.3.3 Mérési eljárás

A mérések minden konfigurációban egységes, reprodukálható eljárás szerint zajlanak:

1. **Környezet inicializálása**
   - Elindul a megfelelő Mosquitto-konfiguráció (A: titkosítatlan, B–C: TLS 1.3, C esetén mTLS + ACL).
   - A Python collector program (collector.py, collector_tls.py vagy mTLS-es változat) csatlakozik a brokerhez és feliratkozik az iot/lab/# topic-családra.
2. **Szenzorok szimulációja**
   - A Wokwi-ban futó sensor1 és sensor2 projektek csatlakoznak a brokerhez.
   - Mindkét szenzor 5 másodperces periódussal publikál hőmérsékleti értékeket a saját topicjára. A payload tartalmazza a szenzorazonosítót és az eszköz oldali időbélyeget is.
3. **Mérési időablak**
   - Egy futás minimális hossza 3–5 perc, hogy stabil átlagok legyenek számolhatók.
   - Ugyanazt az időablakot használjuk a három konfiguráció esetén, így a különbségek nem a futás hosszából adódnak.
4. **Adatgyűjtés és -mentés**
   - A collector program valós időben rögzíti a mért adatokat a measurements*.csv fájlokba.
   - A mérés végén a Wokwi-szimuláció leáll, a collector kilép, a CSV-fájl lezárul.
5. **Kiértékelés**
   - Az elemző szkript feldolgozza az adott konfigurációhoz tartozó CSV-fájlt, kiszámítja a fenti mérőszámokat, és a konzolra írja azokat.
   - Az eredmények táblázatos formában bekerülnek az 5. fejezet vonatkozó alfejezeteibe, illetve grafikonok formájában az ábrák közé.

Ezzel a módszerrel a három konfiguráció egymástól csak biztonsági szintben tér el; minden más paraméter (szenzorok száma, mintavételezési periódus, payload-struktúra) azonos.

5.3.4 Támadási szcenáriók

A mérések nem kizárólag „normál” üzemet vizsgálnak, hanem olyan kontrollált támadási helyzeteket is, amelyek jól illusztrálják az egyes védelmi mechanizmusok szerepét. Ezek közül a legfontosabbak:

1. **Jogosulatlan publish titkosítatlan brokerre (Konfiguráció A)**
   - Egy külön Python „támadó kliens” a szenzorokéhoz hasonló formátumú üzeneteket küld a iot/lab/sensor1/temperature topicra.
   - Mivel sem TLS, sem hitelesítés nincs, a broker gond nélkül elfogadja az üzeneteket; a collector szemszögéből a „valódi” és a hamis szenzoradatok összekeverednek.
   - Az átlaghőmérséklet és a mért értékek eloszlása alapján jól kimutatható az adatinjekció hatása.
2. **Jogosulatlan publish TLS mellett, de hitelesítés nélkül (Konfiguráció B)**
   - A támadó kliens TLS-sel csatlakozik a brokerhez, de továbbra sem használ erős eszközidentitást.

- A broker a titkosított csatorna ellenére nem tud különbséget tenni a jogos és jogosulatlan kliensek között, így az adatinjekció ugyanúgy sikeres.
- Ez demonstrálja, hogy a TLS önmagában nem elég: a csatorna védett, de a végpontok identitása gyenge.

szakdolgozat_iot

3. **Jogosulatlan publish mTLS + ACL mellett (Konfiguráció C)**
    - A támadó kliens vagy:
        - nem rendelkezik érvényes kliens-tanúsítvánnyal → a broker elutasítja a TLS handshake-et,
        - vagy érvényes tanúsítvánnyal, de ACL által tiltott topicra próbál publikálni → a broker not authorized hibát ad.
    - A collector CSV-fájljában nem jelennek meg a támadó által küldött üzenetek; a broker logjában viszont rögzülnek az elutasított kísérletek.

A három szcenárió egymásra épül: míg az A és B konfigurációban a támadás sikeres, addig a C konfigurációban a mTLS és az ACL-ek kombinációja megakadályozza, hogy a támadó érdemben befolyásolja a szenzoradatokat. Ez közvetlenül illusztrálja a 4.3 fejezetben tárgyalt Zero Trust és defense-in-depth elvek gyakorlati hasznát.

5.4 Mérési eredmények és értékelés

Ebben az alfejezetben a 5.1–5.3 fejezetekben bemutatott szimulációs környezetben végzett mérések eredményeit foglalom össze. A cél annak vizsgálata, hogy a különböző biztonsági beállítások (titkosítatlan MQTT, TLS 1.3 feletti MQTT) milyen hatással vannak az üzenetküldési rátára és a rendszer viselkedésére, illetve milyen mértékben növelik a kommunikáció biztonságát.
A mérések során minden konfigurációra ugyanazt a módszertant alkalmaztam:

- két ESP32-alapú szenzor (sensor1, sensor2) 5 másodperces periódussal publikált hőmérsékleti értékeket,
- a collector modul a measurements*.csv fájlokba rögzítette a beérkező üzeneteket,
- az analyzer szkript szenzoronként kiszámította az üzenetszámot, a futás időtartamát, az üzenetküldési rátát és az átlagos hőmérsékletet.

5.4.1 Baseline – titkosítatlan MQTT (Konfiguráció A)

A baseline mérések során a szenzorok egy titkosítatlan, hitelesítés nélküli brokerre csatlakoztak (TCP/1883). A collector program a measurements.csv állományba írt minden beérkező üzenetet, az analyzer pedig szenzoronként aggregálta az adatokat.
Az 5.4.1. táblázat egy tipikus futás eredményeit mutatja be (a konkrét értékek a measurements.csv aktuális tartalmából származnak):

5.4.1. táblázat – Baseline (titkosítatlan) mérési eredmények egy tipikus futásban
(forrás: saját mérés)

Szenzor	Üzenetszám N_msg	Időtartam T_run [s]	Ráta λ [msg/s]	Átlagos hőmérséklet $\bar{T}$ [°C]
sensor1	56	1059.9	0.058	24.25
sensor2	63	1080.3	0.053	25.96

A baseline futások tapasztalatai:

- A szenzorok által használt 5 másodperces periódusnak megfelelően az elméletileg várt üzenetküldési ráta körülbelül

$$\lambda_{\text{elméleti}} \approx \frac{1}{5} = 0{,}2 \text{ msg/s}$$

érték. A gyakorlatban mért ráták ehhez közeli tartományban mozogtak, kisebb eltérésekkel, amelyek az ESP32-es kód időzítési pontatlanságából és a Wokwi szimulációs környezetének fluktuációiból adódnak.

- A mérések során nem jelentkezett tartós üzenetvesztés: mindkét szenzor esetében a collector az elvárt darabszámhoz közeli N_msg értékeket rögzített.
- A hőmérsékleti értékek (a valóságban véletlenszámok) átlagai a 20–30 °C közötti tartományban maradtak, ami azt jelzi, hogy a payloadok formailag épek, a JSON-parsing nem eredményezett torzulást.

Ugyanakkor a baseline konfiguráció biztonsági szempontból kifejezetten gyenge:

- a forgalom titkosítatlanul halad át a hálózaton,
- a broker nem azonosítja a klienseket,
- bármely külső kliens szabadon küldhet üzeneteket a szenzorok topicjaira, illetve le is hallgathatja azokat.

Ezt a sebezhetőséget a 5.3.4 alfejezetben bemutatott „jogosulatlan publish” szcenárió jól illusztrálja: a támadó kliens által küldött hamis hőmérsékleti értékek a collector számára megkülönböztethetetlenek a valódi szenzoradatoktól, így a feldolgozott statisztikák is torzulnak.

5.4.2 TLS 1.3 feletti MQTT – szerverhitelesítés (Konfiguráció B)

A második mérési konfigurációban a szenzorok és a collector már a TLS-es Mosquitto brokerhez csatlakoztak (TCP/8883). A broker egy saját CA által aláírt szervertanúsítvánnyal azonosítja magát, a kliensek pedig a CA-tanúsítvány (ca.crt) segítségével ellenőrzik a szerver identitását.
A collector_tls.py modul a measurements_tls.csv állományba rögzíti az adatokat, az analyzer módosított változata pedig ugyanazokat a mutatókat számítja ki, mint a baseline esetben. Az 5.4.2. táblázat egy tipikus TLS-es futás eredményeit foglalja össze:

5.4.2. táblázat – TLS-es MQTT mérési eredmények egy tipikus futásban

Szenzor	**Üzenetszám N_msg**	**Időtartam T_run [s]**	**Ráta λ [msg/s]**	**Átlagos hőmérséklet $\bar{T}$ [°C]**
sensor1				
sensor2				

A TLS-es futások legfontosabb megfigyelései:

- Az üzenetküldési ráta a legtöbb futásban csak kismértékben tér el a baseline konfigurációban mért értékektől. Ez várható is, mivel a vizsgálat lokális környezetben zajlik, a TLS-handshake által okozott fix költség a teljes mérési időablakhoz viszonyítva elhanyagolható.
- A collector_tls.py ugyanazt a payload-struktúrát és CSV-formátumot használja, így az átlaghőmérsékletek és az üzenetszámok a baseline mérésekhez hasonló tartományban maradnak. Ez alátámasztja, hogy a TLS bevezetése a rendszer funkcionalitását nem befolyásolta.
- Biztonsági szempontból azonban lényeges előrelépés történik: a forgalom titkosított, a szenzoradatok út közben nem olvashatók és nem módosíthatók egyszerűen, valamint a kliensek csak akkor fogadják el a kapcsolatot, ha a broker tanúsítványa megbízható CA-tól származik.

Ugyanakkor a TLS önmagában nem oldja meg a jogosulatlan kliensek problémáját: ha egy támadó kliens ismeri a broker címét és rendelkezik a CA-tanúsítvánnyal, ő is létre tud hozni egy titkosított kapcsolatot, és publikálhat hamis adatokat olyan topicokra, amelyekhez nincs explicit hozzáférés-szabályozás rendelve. Ez indokolja a következő lépcsőt, a mTLS és ACL-ek bevezetését.

5.4.3 Összehasonlító értékelés

A titkosítatlan és a TLS-es konfigurációk eredményei alapján a következő megállapítások tehetők:

- **Teljesítmény szempontjából**
  A mért üzenetküldési ráták mindkét konfigurációban a várt 0,2 msg/s körüli tartományban mozognak. A TLS-es futásokban tapasztalható minimális eltérések nem befolyásolják érdemben a rendszer működését; a kriptográfiai réteg költsége lokális környezetben jól kezelhető.
- **Biztonsági szempontból**
  A baseline konfigurációban a rendszer gyakorlatilag „nyitott MQTT buszként” viselkedik: bárki csatlakozhat, az adatforgalom lehallgatható és módosítható. A TLS bevezetése ezt a kockázatot jelentősen csökkenti (bizalmasság, integritás, szerverhitelesítés), ugyanakkor a jogosulatlan kliensek teljes kizárásához szükség van az erős kliensidentitásra (mTLS) és a finomhangolt ACL-ekre is.
- **Kutatási kérdések szempontjából**
  A mérések megerősítik a 4. fejezetben megfogalmazott hipotézist: az IoT-rendszerekben a csatornaszintű titkosítás bevezetése nem feltétlenül jár drasztikus teljesítményromlással, viszont jelentősen erősíti a rendszer biztonsági profilját. A következő lépésben (Konfiguráció C) azt vizsgálom, hogy az mTLS és az ACL-ek milyen módon képesek a támadási szcenáriókban bemutatott jogosulatlan publish kísérleteket blokkolni, illetve ez milyen többletterhelést ró a rendszerre.