https://doi.org/1071066/VARECZA.2025.1.05

Vegera József

IoT ESZKÖZÖK SKÁLÁZHATÓ EMULÁCIÓJA KONTÉNERALAPÚ KÖRNYEZETEKBEN SÉRÜLÉKENYSÉGVIZSGÁLATI CÉLOKRA

SCALABLE EMULATION OF IoT DEVICES IN CONTAINER-BASED ENVIRONMENTS FOR VULNERABILITY TESTING PURPOSES

ÖSSZEFOGLALÓ

A Dolgok Internete (IoT) biztonsági vizsgálata során a kutatók alapvető kihívása a „skálázhatóság-hűség szakadék” (Scalability-Fidelity Gap) áthidalása: míg a fizikai tesztkörnyezetek magas hűséget, de alacsony skálázhatóságot kínálnak, addig a szimulációk skálázhatók, de hiányzik belőlük a hardveres pontosság. Jelen tanulmány egy hibrid megközelítést javasol, amely a konténerizáció (Docker) és a hardveremuláció (QEMU) kombinációjára épít a hatékony sérülékenységvizsgálat érdekében. A dolgozat részletezi az ARM és MIPS alapú firmware-ek x86-os környezetben történő futtatásának technikai megoldásait (User-Mode és System-Mode emuláció), valamint bemutatja a Kubernetes és a KubeVirt szerepét a skálázható, hardveresen izolált tesztkörnyezetek orkesztrációjában. A tanulmány továbbá elemzi az IP-alapú (MQTT) protokollok szimulációját, valamint a rádiós (Zigbee) protokollok vizsgálatához szükséges Hardware-in-the-Loop (HIL) módszertan integrációját konténerizált környezetekbe.

Kulcsszavak: IoT biztonság, konténerizáció, emuláció, Docker, Kubernetes, QEMU, KubeVirt, firmware elemzés, Hardware-in-the-Loop (HIL)

ABSTRACT

In the security testing of the Internet of Things (IoT), a fundamental challenge for researchers is bridging the "Scalability-Fidelity Gap": while physical testbeds offer high fidelity but low scalability, simulations are scalable but lack hardware accuracy. This study proposes a hybrid approach leveraging the combination of containerization (Docker) and hardware emulation (QEMU) for effective vulnerability assessment. The paper details technical solutions for running ARM and MIPS-based firmware in x86 environments (User-Mode and System-Mode emulation) and demonstrates the role of Kubernetes and KubeVirt in orchestrating scalable, hardware-isolated testbeds. Furthermore, the study analyzes the simulation of IP-based protocols (MQTT) and the integration of Hardware-in-the-Loop (HIL) methodologies required for testing radio protocols (Zigbee) within containerized environments.

Keywords: IoT security, containerization, emulation, Docker, Kubernetes, QEMU, KubeVirt, firmware analysis, Hardware-in-the-Loop (HIL)

Bevezetés az IoT sérülékenységvizsgálati környezetekbe

A Dolgok Internete (Internet of Things, IoT) alapvetően különbözik a hagyományos IT-rendszerektől. Nem csupán szoftverekről van szó, hanem komplex, többrétegű kiber-fizikai rendszerekről (Cyber-Physical Systems, CPS), amelyek magukba foglalják a fizikai érzékelőket, a beágyazott rendszerek firmware-jét, a heterogén hálózati protokollokat (a rádiófrekvenciástól az IP-alapúig) és a globális felhő-infrastruktúrákat. Ez a komplexitás egy egyedi és kiterjedt sebezhetőségi mátrixot hoz létre.

Célkitűzés:

A tanulmány célja egy olyan hibrid IoT sérülékenységvizsgálati architektúra bemutatása, amely egyesíti a konténeralapú skálázhatóságot és a hardveremuláció (QEMU) nyújtotta magas

hűséget. A megközelítés feladata, hogy csökkentse a „Scalability–Fidelity Gap” hatását, és megteremtse a nagyméretű, determinisztikus és biztonságos firmware-elemzési tesztkörnyezetek kialakításának lehetőségét.

Kutatási kérdés:
Vajon képes-e a konténerizáció és a hardveremuláció kombinációja — Kubernetes- és KubeVirt-alapú orkesztrációval kiegészítve — olyan IoT tesztkörnyezetet biztosítani, amely egyszerre nyújt magas hűséget és ipari szintű skálázhatóságot a firmware-sérülékenységek dinamikus vizsgálatához?

Az IoT-ökoszisztéma sebezhetőségi mátrixa

A biztonsági kockázatok az ökoszisztéma minden szintjén megjelennek. Ide tartoznak a hardveres támadások, a firmware-ek elégtelen tesztelése és a frissítési mechanizmusok hiányosságai, a hálózati protokollok (mint az MQTT vagy CoAP) implementációs gyengeségei, valamint a háttérrendszerek és felhőkomponensek sebezhetőségei. A támadási vektorok skálája az eszközök fizikai eltulajdonításától (device hijacking) és a hálózati behatolástól (network intrusion) a szintetikus identitások létrehozásáig terjed.

A problémát súlyosbítja, hogy az IoT-piacon uralkodó gyors innovációs kényszer és az erős funkcionális követelmények miatt a biztonsági szempontokat (security-by-design) gyakran „későbbre hagyják” a fejlesztési ciklusban. Ez a hiányosság teszi elengedhetetlenné a robusztus, külső tesztelési és validációs módszertanokat.

A fizikai tesztkörnyezetek (Physical Testbeds) korlátai

A sérülékenységvizsgálat legmagasabb hűségű (high-fidelity) formája a fizikai tesztkörnyezeteken, azaz valódi eszközök laboratóriumán történő tesztelés. Ez a megközelítés azonban súlyos korlátokkal küzd:

- **Skálázhatóság és költség:** Egy valósághű „smart city” vagy ipari IoT (IIoT) telepítés szimulálása több ezer, vagy akár több tízezer fizikai eszköz beszerzését, telepítését és menedzselését igényelné, ami irreális költségekkel jár.
- **Reprodukálhatóság:** A fizikai környezetekben fellépő változók (pl. rádiófrekvenciás interferencia, hálózati ingadozások) miatt a tesztek pontos megismétlése (reprodukálhatósága) és a környezet determinisztikus viselkedése nem garantálható.
- **Menedzselhetőség:** Nagy számú eszköz firmware-frissítése, tesztelés utáni alaphelyzetbe állítása (rollback) és konfigurációja logisztikai és időbeli rémálommá válik.

A szimuláció és emuláció definíciós spektruma: a „Scalability-Fidelity Gap” áthidalása

A fizikai tesztelés korlátai miatt a kutatók és biztonsági szakemberek a virtualizáció felé fordultak, amely egy spektrumon mozog:

1. **Szimuláció (Simulation):** Az eszközök *viselkedésének* modellezése, gyakran magas absztrakciós szinten. Egy szimulátor (pl. IoTNetSim) képes lehet több ezer eszköz hálózati forgalmát generálni, de nem a *valódi* firmware-kódot futtatja. Előnye a rendkívüli skálázhatóság, hátránya az alacsony hűség.
2. **Emuláció (Emulation):** Az eszköz *hardverének* és *szoftverének* (pl. a teljes firmware bináris) valós időben történő futtatása egy másik rendszeren (pl. QEMU emulátorral).

Ez biztosítja a legmagasabb szoftveres hűséget, mivel a *valódi, potenciálisan sebezhető kódot* teszteljük.

A sérülékenységvizsgálatokkal kapcsolatosan létezik egy alapvető probléma, amely **„Scalability-Fidelity Gap”** (skálázhatóság-hűség szakadék) néven ismert. A fizikai tesztelés maximális valósághűséget, de minimális skálázhatóságot kínál. A tiszta szimuláció maximális skálázhatóságot, de minimális hűséget nyújt. A hatékony sérülékenységvizsgálat célja azonban a *valódi kód* (magas hűség) tesztelése *nagyszámú, interakcióban lévő eszköz* kontextusában (magas skálázhatóság).

A konténerizált megközelítés tézise

A konténerizáció (pl. Docker) önmagában is jelentős előnyöket kínál az IoT-eszközökön való alkalmazástelepítésben, mint például a hordozhatóság a heterogén hardverek között, az erőforrás-hatékonyság (a megosztott kernel révén), a gyors telepítés és az alkalmazások közötti biztonsági izoláció.

Ez a tanulmány azt a tézist állítja fel, hogy a konténereket nem csupán az *eszközökön* (edge deployment), hanem a *tesztelési infrastruktúrában* is alkalmazni kell. A konténerek ebben a kontextusban a magas hűségű emulátorok (mint a QEMU) skálázható, hordozható és hálózati szempontból izolálható „csomagolóanyagaként” (wrapper) funkcionálnak. Ez a hibrid megközelítés lehetővé teszi egy teljes, komplex IoT-ökoszisztéma – beleértve magukat az emulált eszközöket, az átjárókat (gateways), a felhővégpontokat, sőt, a támadó infrastruktúrákat (pl. botnet szerverek) – determinisztikus, izolált és skálázható tesztelését.

Architekturális alapok: konténerek és virtuális gépek összevetése biztonsági kontextusban

A konténerizált emulációs stratégia megértéséhez elengedhetetlen a virtualizációs technológiák közötti alapvető architekturális és biztonsági különbségek tisztázása.

Az izoláció anatómiája: Hypervisor vs. megosztott kernel

A virtualizáció két domináns formája alapvetően eltérő módon éri el az izolációt:

- **Virtuális gépek (VMs):** A VM-ek egy hypervisor (legyen az Type 1 „bare-metal”, vagy Type 2 „hosted”) által felügyelt teljes hardverabsztrakciót valósítanak meg. Minden egyes VM saját, teljes vendég operációs rendszert (Guest OS) futtat, beleértve a *saját, dedikált kernelét* is. Az izoláció hardveresen szimulált szinten történik.
- **Konténerek (pl. Docker):** A konténerek OS-szintű virtualizációt alkalmaznak. Nem tartalmaznak vendég operációs rendszert vagy kernelt, ehelyett *megosztják a gazdagép (Host OS) kernelét*. Egy konténer-image csupán az alkalmazást és annak bináris/könyvtári függőségeit (libraries, binaries) tartalmazza.

Biztonsági kompromisszumok: a „Hard” vs. „Lightweight” izoláció

A két megközelítés közötti legfontosabb különbség a biztonsági tesztelés szempontjából az izoláció erőssége.

- **VM-ek előnye (erős izoláció):** A VM-ek „Hard Security Boundaries”-t (erős biztonsági határokat) biztosítanak. Mivel minden VM saját kernelt futtat, a vendég OS-ek teljesen elkülönülnek egymástól és a gazdagéptől. Egy VM kompromittálódása (pl. egy kernel-

exploit révén) általában nem teszi lehetővé a támadó számára a gazdagép vagy más VM-ek elérését. Ez az erős szegregáció ideálissá teszi őket megbízhatatlan, potenciálisan rosszindulatú kód – mint például egy ismeretlen IoT firmware – futtatására.

- **Konténerek kockázata (könnyűsúlyú izoláció):** A konténerek „könnyűsúlyú izolációt" nyújtanak. A közös kernel egyben közös támadási felületet is jelent. Egy, a gazdagép kernelében található sebezhetőség (mint a „Dirty COW" vagy a „Leaky Vessels") kihasználása egy konténerből „container escape"-hez vezethet. Ez lehetővé teszi a támadó számára, hogy kitörjön a konténerből, és teljes (gyakran root) hozzáférést szerezzen a gazdagéphez és annak összes többi konténeréhez.

Teljesítmény, erőforrás-hatékonyság és skálázhatóság

Míg biztonsági izolációban a VM-ek jobbak, addig teljesítményben és erőforrás-hatékonyságban a konténerek felülmúlhatatlanok.

- **Méret és indítási idő:** A konténerek image-mérete jellemzően MB-os nagyságrendű, és másodpercek alatt indulnak. Ezzel szemben a VM-ek (melyek egy teljes OS-t tartalmaznak) GB-os méretűek és percekig tartó bootolási idővel rendelkeznek.
- **Teljesítmény és overhead:** A konténerek natívhoz közeli teljesítményt nyújtanak, mivel nincs hypervisor és vendég OS overhead. Az erőforrás-kihasználásuk sokkal hatékonyabb, így sűrűbben telepíthetők ugyanarra a hardverre, ami közvetlen költségmegtakarítást jelent.
- **Skálázhatóság:** A konténerek a microservice architektúrák alapkövei. Lehetővé teszik az alkalmazások egyes komponenseinek granuláris, egymástól független skálázását, amit az olyan orkesztrációs platformok, mint a Kubernetes, automatizálnak.

Az IoT-tesztelés alapvető architekturális ellentmondása

A fenti összehasonlítás egy kritikus architekturális ellentmondást tár fel az IoT-tesztelés kontextusában:

1. A biztonsági tesztelő egy IoT firmware-t szeretne futtatni, amely jellemzően ARM vagy MIPS architektúrára épül.
2. Ez az IoT firmware egyedi, beágyazott Linux kernelt (vagy RTOS-t) igényel a futáshoz.
3. A konténerek definíció szerint *megosztják* a gazdagép kernelét, amely a tesztkörnyezetben szinte mindig x86-alapú.
4. **Konklúzió**: a Docker önmagában *képtelen* egy ARM-alapú IoT-eszköz teljes rendszerének (és kernelének) futtatására.

A „konténerizált szimuláció" tehát ebben a kontextusban nem jelentheti azt, hogy az IoT firmware „konténerként" fut. **A megoldás egy hibrid architektúra:** egy *teljes rendszert emuláló szoftver (QEMU System-Mode) fut magán a konténeren belül*. A konténer itt a skálázható, izolált *környezetet* (hálózat, függőségek) biztosítja, míg a benne futó QEMU adja a hardveres *hűséget*.

Egy lehetséges megoldás: KubeVirt – a VM-izoláció és a konténer-orkesztráció szintézise

A fenti hibrid modell (QEMU egy Docker-konténerben) működőképes, de örökli a konténerek biztonsági gyengeségét: ha a firmware-ben lévő exploit képes kitörni a QEMU-ból *és* a konténerből (pl. egy kernel-sebezhetőségen keresztül), a gazdagép kompromittálódik.

Az ideális megoldás egyesíti a két szemlélet előnyeit. A **KubeVirt** egy Cloud Native Computing Foundation (CNCF) projekt, amely lehetővé teszi *teljes virtuális gépek* (QEMU/KVM-alapú) futtatását és menedzselését Kubernetes-podokon belül. A KubeVirt segítségével a tesztelő megkapja:

- A *VM-ek hardveres izolációját* (a KVM hypervisor révén), ami maximális biztonságot nyújt a gazdagépre nézve a tesztelt firmware-kóddal szemben.
- A *Kubernetes teljes orkesztrációs képességeit* (skálázás, hálózatkezelés, automatizálás).

A KubeVirt így a legfejlettebb megoldást képviseli a skálázható és biztonságos IoT firmware-sérülékenységvizsgálati tesztkörnyezetek felépítésében.

1. Táblázat: Virtualizációs technológiák összehasonlítása IoT tesztelési kontextusban

Jellemző	**Tiszta VM (pl. KVM)**	**Tiszta Konténer (Docker)**	**Hibrid Emuláció (Docker + QEMU-User)**	**Hibrid Emuláció (Docker + QEMU-System)**	**Orkesztrált VM (KubeVirt)**
Izolációs Szint	Hardveres (Hypervisor)	Megosztott Kernel (OS-szintű)	Megosztott Kernel (OS-szintű)	Megosztott Kernel (OS-szintű)	Hardveres (Hypervisor)
Emulációs Hűség	Magas (Teljes OS)	Nincs (Csak natív binárisok)	Alacsony (Csak syscall fordítás)	Magas (Teljes hardver emuláció)	Magas (Teljes OS)
Teljesítmény	Közepes (Overhead)	Nagyon Magas (Natív közeli)	Magas (Gyors)	Nagyon Alacsony (Lassú)	Közepes (Overhead)
Indítási Idő	Percek	Másodper-cek	Másodper-cek	Percek (OS boot)	Percek (OS boot)

Erőforrás-sűrűség	Alacsony	Magas	Magas	Közepes	Közepes
Skálázhatóság	Nehézkes (Manuális)	Magas (Orkesztrálható)	Magas (Orkesztrálható)	Magas (Orkesztrálható)	Nagyon Magas (K8s-natív)
Fő Alkalmazás	Manuális, mély elemzés	IP-protokollok tesztelése (MQTT)	Binárisok fuzzingja (AFL)	Teljes firmware futtatása (FirmAE)	Skálázott, biztonságos rendszer-pentest

A konténerizált emuláció magja: a QEMU és a Docker házasítása

A konténeres IoT-tesztelés technikai magja a kereszt-architektúrás (cross-architecture) probléma megoldása: hogyan futtassunk ARM vagy MIPS alapú IoT-kódot egy x86-os Docker-gazdagépen.

A kereszt-architektúra probléma definiálása

Az IoT-eszközök döntő többsége alacsony fogyasztású ARM vagy MIPS processzorokra épül. A Docker-konténerek azonban, mivel a gazdagép kernelét osztják meg, alapértelmezetten csak a gazdagép architektúrájának megfelelő (pl. x86-64) binárisokat képesek futtatni. Egy ARM-alapú image (pl. arm64v8/ubuntu) futtatásának kísérlete egy x86-os hoszton emuláció nélkül azonnali exec format error hibát eredményez. A megoldást a QEMU (Quick EMUlator) biztosítja.

A QEMU két üzemmódja: User-Mode vs. System-Mode

A QEMU két, alapvetően eltérő módban képes működni, amelyek eltérő tesztelési célokat szolgálnak:

1. QEMU User-Mode (qemu-user-static)

A felhasználói módú emuláció célja egyedi, nem natív architektúrájú binárisok futtatása a gazdagép operációs rendszerén.

- **Működése:** Nem emulál teljes hardvert (pl. alaplapot, eszközöket). Ehelyett dinamikusan lefordítja a célarchitektúra (pl. ARM) utasításait a gazda-architektúra (x86) utasításaira, és ami kulcsfontosságú, „elfogja” a bináris által indított rendszerhívásokat (syscalls), és átirányítja azokat a gazdagép kerneléhez.
- **Előnyök:** Rendkívül gyors, közel 10-szer gyorsabb, mint a teljes rendszeremuláció. Ideális egyedi alkalmazások (pl. egy webszerver bináris, egy titkosító rutin) izolált tesztelésére és fuzzingolására.
- **Hátrányok:** A hűsége alacsony. Azonnal meghiúsul, ha a program hardverspecifikus műveletet kísérel meg (pl. közvetlen memóriahozzáférés, speciális ioctl hívások), vagy

olyan rendszerhívást használ, amelyet a QEMU nem tud lefordítani. Nem képes egy operációs rendszer kernelt bootolni.

2. QEMU System-Mode (qemu-system-)*

A rendszermódú emuláció célja egy *teljes számítógép* hardverének szimulálása.

- **Működése:** A CPU mellett emulálja az alaplapot (az -M kapcsolóval megadva), a memóriát, a hálózati kártyákat és egyéb perifériákat. Ehhez egy bootolható kernel image-re és egy lemezképre (pl. a firmware fájlrendszerét tartalmazó qcow2 fájlra) van szüksége.
- **Előnyök:** Magas hűséget biztosít. Képes a teljes, módosítatlan IoT firmware (pl. egy OpenWRT-alapú router) bootolási folyamatának és futásának emulálására. Lehetővé teszi a kernel-interakciók és a hardverfüggő szolgáltatások elemzését.
- **Hátrányok:** Rendkívül lassú (a SoftMMU overhead miatt) és közismerten nehéz beállítani. Az ARM ökoszisztéma rendkívüli fragmentáltsága miatt egy adott, valós eszközre (pl. Netgear router) készült firmware-image szinte soha nem fog elindulni egy generikus QEMU „board” (-M) definíción. Egy új eszköz beállítása hetekig is tarthat.

Sebezhetőségvizsgálati módszertanok konténerizált környezetben

Egy sikeresen felépített, konténerben futó emulált IoT-eszköz (legyen az User-Mode vagy System-Mode alapú) lehetővé teszi a sérülékenységvizsgálati módszertanok széles skálájának alkalmazását.

A pentest folyamata emulált környezetben

A folyamat lépései megegyeznek egy fizikai eszköz tesztelésével, de a műveletek a konténerhálózaton keresztül, a virtuális célpont ellen irányulnak:

1. **Felderítés (Reconnaissance):** Az emulált eszköz konténerének IP-címét nmap segítségével szkenneljük a nyitott portok és futó szolgáltatások (pl. webkiszolgáló, Telnet, MQTT broker) azonosítására.
2. **Sebezhetőségelemzés (Vulnerability Analysis):** Automatizált szkennerek, mint a Trivy, futtathatók magán a Docker-image-en a build fázisban, vagy a futó rendszeren a szoftverkomponensek ismert CVE-inek azonosítására.
3. **Kihasználás (Exploitation):** Olyan keretrendszerek, mint a Metasploit, vagy egyedi szkriptek használhatók a talált hibák (pl. gyenge alapértelmezett jelszavak, parancsinjekciós sebezhetőségek a webes felületen) kihasználására.
4. **Forgalomelemzés (Traffic Analysis):** A tcpdump vagy Wireshark eszközökkel rögzíthetjük a konténer hálózati interfészének (vagy a Docker bridge-nek) forgalmát, keresve titkosítatlanul küldött érzékeny adatokat (pl. jelszavak) vagy protokoll-anomáliákat.

IP-alapú protokollok támadása (MQTT & CoAP)

Az MQTT és a CoAP tisztán IP-alapú, alkalmazásrétegbeli (L7) protokollok. Mivel nem kötődnek specifikus hardverhez (mint a rádiós protokollok), *tökéletesen* szimulálhatók tiszta konténeres környezetben, QEMU emuláció nélkül is.

Egy tipikus MQTT tesztkörnyezet felállítása:

- Egy „broker” konténer indítása (pl. a hivatalos eclipse-mosquitto image-ből).
- Egy vagy több „publisher” (szimulált eszköz) konténer indítása, amelyek adatokat küldenek.
- Egy „subscriber” (támadó) konténer indítása, amely megkísérli lehallgatni a forgalmat vagy támadni a brokert.

Gyakori MQTT támadási vektorok:

- **Anonim hozzáférés:** Ha a broker rosszul van konfigurálva (allow_anonymous true), a támadó egy egyszerű mosquitto_sub -t '#' -h <broker_címe> paranccsal feliratkozhat az összes (#) témára, és lehallgathat minden, potenciálisan érzékeny adatot.
- **Szótáras támadás (Brute-Force):** Ha a broker használ hitelesítést, de nincs „rate limiting” (próbálkozási korlát), a Metasploit auxiliary/scanner/mqtt/connect modulja vagy más eszközök használhatók a felhasználónevek és jelszavak kitalálására.
- **Denial of Service (DoS):** Ismert CVE-k (pl. CVE-2017-7651) kihasználásával a broker összeomlasztható speciálisan formázott topic-nevek (pl. $TEST) vagy túlméretezett payload-ok küldésével.

Rádiós protokollok (Zigbee) támadása és a Hardware-in-the-Loop (HIL)

Míg az MQTT (L7) tisztán szimulálható, addig az olyan protokollok, mint a Zigbee vagy a Bluetooth, az IEEE 802.15.4 szabványra épülnek és az alacsonyabb fizikai (L1) és MAC (L2) rétegeken működnek. Ezek *nem* IP-alapúak, így nem szimulálhatók egy Docker-hálózaton. Bár léteznek Zigbee szimulátorok, azok általában csak a kriptográfiai protokoll logikáját (pl. AES kulcscsere) vizsgálják, nem pedig a *valós rádiós implementáció* sebezhetőségeit.

A Zigbee teszteléséhez *valódi* rádiós hardverre van szükség. Ez a **„Hardware-in-the-Loop” (HIL)** megközelítés, ahol a tesztelő szoftver (a konténerben) egy fizikai hardveren keresztül lép interakcióba a valódi, fizikai célponttal.

Gyakorlati útmutató HIL teszteléshez:

- **Eszköz:** A KillerBee a Zigbee pentestelés de facto eszközkészlete, amely kompatibilis USB-s rádiós hardvert (dongle) igényel.
- **Konténerizálás:** A KillerBee szoftver futtatható egy Docker konténerben (pl. sdcampbell/killerbee vagy dockerBee).
- **Eszközáteresztés (Device Passthrough):** A konténer indításakor át kell adni a gazdagéphez csatlakoztatott fizikai USB eszközt a konténernek. Ez megtehető a --privileged és --device kapcsolókkal (docker run --privileged --device=/dev/ttyUSB0...), vagy biztonságosabban, a docker-compose.yml fájl devices szekciójában, ideális esetben az eszköz stabil /dev/serial/by-id/ elérési útját használva.

2. Táblázat: IoT sebezhetőségvizsgálati technikák és eszközök konténeres környezetben

Támadási vektor / technika	Célprotokoll / Réteg	Javasolt eszköz	Konténerizált megvalósítás (és főbb parancsok)
Hálózati felderítés	L3/L4 (IP/TCP/UDP)	nmap	docker run --rm -it nmap <cél-IP>
Protokoll Pentest (MQTT)	L7 (Alkalmazás)	mosquitto_clients, Metasploit	docker run --rm -it eclipse-mosquitto mosquitto_sub -t '#' -h <cél-IP>
Protokoll Pentest (Zigbee)	L1/L2 (Rádió)	KillerBee	(HIL) docker run -it --rm --privileged --device=/dev/ttyUSB0 docker-killerbee
Hálózati forgalom elemzés	L2-L7	Wireshark, tcpdump	docker run --rm --net=container:<cél-konténer-ID> tcpdump -i eth0

Skálázható tesztkörnyezetek és hibrid megközelítések

Míg az egyedi firmware-ek elemzése Docker-konténerekben is megoldható, a valósághű, nagyméretű (pl. botnet) támadások szimulálásához orkesztrációra van szükség.

Az orkesztráció választása: Docker Compose vs. Kubernetes

- **Docker Compose:** Ideális eszköz helyi fejlesztéshez, prototípusok készítéséhez és egyszerű, egy-gazdagépes (single-host) tesztkörnyezetek (pl. egy broker, egy eszköz, egy támadó) gyors definiálására és indítására. Előnye az egyszerűsége.
- **Kubernetes (K8s):** Az iparági standard a skálázható, elosztott, több-gazdagépes (multi-host) rendszerek orkesztrálására. Olyan képességei, mint az automatikus skálázás (auto-scaling), öngyógyítás (self-healing) és terheléselosztás (load balancing), elengedhetetlenek egy nagyméretű (pl. 100+ eszközt szimuláló) tesztkörnyezet futtatásához. Erőforrás-szűkös „edge” környezetekben (pl. Raspberry Pi klasztereken) a K3s, egy könnyűsúlyú K8s disztribúció, az ideális választás.

Az ökoszisztéma szimulálása: Kubernetes Network Policies

Egy IoT-tesztkörnyezet hűsége nemcsak az eszközök emulálásán, hanem a *hálózati topológia* szimulálásán is múlik. Egy valós telepítésben az eszközök szigorúan szegmentáltak (pl. egy kamera csak a NVR-rel kommunikálhat).

A Kubernetesben a pod-ok (szimulált eszközök) alapértelmezetten egy „lapos" hálózaton vannak, ahol mindenki kommunikálhat mindenkivel. A **Kubernetes Network Policies** (hálózati irányelvek) lehetővé teszik a pod-ok közötti L3/L4 (IP/Port) szintű forgalom finomhangolt szabályozását. Ennek segítségével pontosan leképezhetjük egy valós IoT-telepítés tűzfalszabályait és hálózati szegmentációját. Így tesztelhetővé válik, hogy egy kompromittált „eszköz" (pod) képes-e kitörni a szegmentációból és laterális mozgást (lateral movement) végrehajtani a szimulált hálózaton.

Összefoglalás

A tanulmány áttekintette az IoT sérülékenységvizsgálat jelenlegi módszertani kihívásait, különös tekintettel a fizikai tesztkörnyezetek hűségelőnyeire és súlyos skálázhatósági korlátaira, illetve a tisztán szimulált környezetek ellentétes tulajdonságaira. Rámutattam, hogy a „Scalability–Fidelity Gap" áthidalása nem érhető el kizárólag konténertechnológiákkal vagy tiszta hardveremulációval. A bemutatott hibrid megközelítés — amely konténerizált környezetben futó QEMU-emulációra épül, Kubernetes-alapú skálázhatósággal és KubeVirt-alapú biztonsági izolációval kiegészítve — képes egy komplex IoT-ökoszisztéma valósághű, determinisztikus és skálázható reprodukciójára.

A modell kiterjed mind az IP-alapú protokollok (pl. MQTT), mind a rádiós technológiák (pl. Zigbee) támadási vektorainak vizsgálatára, utóbbi esetben Hardware-in-the-Loop integrációval. Az eredmények alapján megállapítható, hogy a hibrid konténer–emulációs architektúra nemcsak képes nagyméretű firmware-dinamikus elemzési környezetek létrehozására, hanem csökkenti az eszköz- és infrastruktúraigényt, növeli a reprodukálhatóságot és a vizsgálati folyamat automatizálhatóságát is.

Következtetések és ajánlások, a tesztkörnyezet biztonsága (Testbed Hardening)

A konténeralapú technológiák és a hardveremulátorok kombinációja forradalmasítja az IoT sérülékenységvizsgálatot, lehetővé téve a korábban elérhetetlen skálázhatóságot és automatizálást.

Kritikus fontosságú felismerni, hogy a sérülékenységvizsgálati környezet *definíció szerint* sebezhető kódot futtat, miközben a felépítéséhez gyakran veszélyes, emelt szintű jogosultságokra van szükség (pl. `docker run --privileged` a `binfmt_misc` vagy a HIL beállításához). Egy rosszul konfigurált környezet (pl. a Docker socket átadása egy konténernek) vagy egy „container escape" sebezhetőség kihasználása a teljes tesztelési infrastruktúra kompromittálódásához vezethet.

Ezért a tesztkörnyezetre *még szigorúbb* biztonsági szabályokat kell alkalmazni, mint egy éles rendszerre:

- A konténereket soha ne futtassuk root felhasználóként.
- Kerüljük a Docker socket (`docker.sock`) megosztását.
- Használjunk szigorúan izolált Docker-hálózatokat.
- Tartsuk naprakészen a gazdagép kernelét és a Docker motort a kernel-szintű sebezhetőségek elkerülése érdekében.

- Ahol a biztonságos izoláció kritikus, használjunk KubeVirt-et, amely hardveres izolációt biztosít a futtatott firmware és a gazdagép között, minimalizálva a „testbed escape” kockázatát.

Jövőbeli irányok

A terület fejlődését két fő irány határozza meg:

1. **Orkesztrált hibrid tesztelés:** A KubeVirt (a biztonságos, hardveresen izolált emulációért) és az Akri (a skálázható HIL-képességért) kombinációja válik a nagyméretű, professzionális tesztkörnyezetek alapjává.
2. **Mesterséges intelligencia (AI/ML):** Az AI-alapú futásidejű biztonsági (runtime security) eszközök és anomália-detekciós rendszerek alkalmazása a szimulált környezetekben generált hatalmas adatmennyiség valós idejű elemzésére.

Felhasznált szakirodalom

- Bevywise. (n.d.). *MQTT security best practices & implementation guide*. Retrieved November 7, 2025, from https://www.bevywise.com/blog/mqtt-security-best-practices/
- Department of Homeland Security. (n.d.). *CPSSEC — Cyber physical systems security*. Science and Technology Directorate. Retrieved November 7, 2025, from https://www.dhs.gov/archive/science-and-technology/cpssec
- eInfochips. (n.d.). *Containerized data processing for IoT: Orchestrating microservices at the edge*. Retrieved November 7, 2025, from https://www.einfochips.com/blog/containerized-data-processing-for-iot-orchestrating-microservices-at-the-edge/
- Fortinet. (n.d.). *Top IoT device vulnerabilities: How to secure IoT devices*. Retrieved November 7, 2025, from https://www.fortinet.com/resources/cyberglossary/iot-device-vulnerabilities
- IBM. (n.d.). *Containers versus virtual machines (VMs): What's the difference?* Retrieved November 7, 2025, from https://www.ibm.com/think/topics/containers-vs-vms
- InGuardians. (n.d.). *KillerBee: Practical ZigBee exploitation framework*. Retrieved November 7, 2025, from https://www.inguardians.com/presentations/killerbee-practical-zigbee-exploitation-framework/
- Kaspersky Lab. (2020). *Dynamic analysis of firmware components in IoT devices*. Securelist. Retrieved November 7, 2025, from https://securelist.com/dynamic-analysis-of-firmware-components-in-iot-devices/106901/
- Keysight. (n.d.). *IoT security assessment*. Retrieved November 7, 2025, from https://www.keysight.com/us/en/products/network-security/iot-security-assessment.html
- Kim, M., Jung, D., Choi, H., & Cha, S. (2020). *FirmAE: Towards large-scale emulation of IoT firmware for dynamic analysis*. Annual Computer Security Applications Conference

(ACSAC). https://0xdkay.me/pub/2020/kim-acsac2020-slides.pdf https://doi.org/10.1145/3427228.3427294

- KubeVirt. (n.d.). *KubeVirt: Building a virtualization API for Kubernetes*. Retrieved November 7, 2025, from https://kubevirt.io/
- Kubernetes. (n.d.). *Device plugins*. Kubernetes Documentation. Retrieved November 7, 2025, from https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins/
- Kubernetes. (n.d.). *Network policies*. Kubernetes Documentation. Retrieved November 7, 2025, from https://kubernetes.io/docs/concepts/services-networking/network-policies/
- Meegle. (n.d.). *Containerization in IoT*. Retrieved November 7, 2025, from https://www.meegle.com/en_us/topics/containerization/containerization-in-iot
- National Center for Biotechnology Information. (n.d.). *A comprehensive analysis of security challenges in ZigBee 3.0 networks*. PMC. Retrieved November 7, 2025, from https://pmc.ncbi.nlm.nih.gov/articles/PMC12349651/
- QEMU Project. (nnd.). *Emulation*. QEMU Documentation. Retrieved November 7, 2025, from https://www.qemu.org/docs/master/about/emulation.html
- Red Hat. (n.d.). *Containers vs VMs*. Retrieved November 7, 2025, from https://www.redhat.com/en/topics/containers/containers-vs-vms
- Stereolabs. (n.d.). *Running and building ARM Docker containers on x86*. Retrieved November 7, 2025, from https://www.stereolabs.com/docs/docker/building-arm-container-on-x86
- Zero Day Initiative. (2020, May 27). *MindShaRE: How to "Just emulate it with QEMU"*. Retrieved November 7, 2025, from https://www.thezdi.com/blog/2020/5/27/mindshare-how-to-just-emulate-it-with-qemu
- Zheng, Y., Zhang, A., & Feng, D. (2019). *FIRM-AFL: High-throughput greybox fuzzing of IoT firmware via augmented process emulation*. USENIX Security Symposium. https://wcventure.github.io/FuzzingPaper/Paper/USENIX19_FirmAFL.pdf

SZERZŐI ADATOK

Vegera József
Mesteroktató
Nyíregyházi Egyetem
Matematika és Informatika Intézet
vegera.jozsef@nye.hu