Cloud-infrastruktúra és a rajta futó telekom-applikációk aktuális biztonsági kihívásai

CSORDÁS GÁBOR
Nokia
gabor.csordas@nokia.com

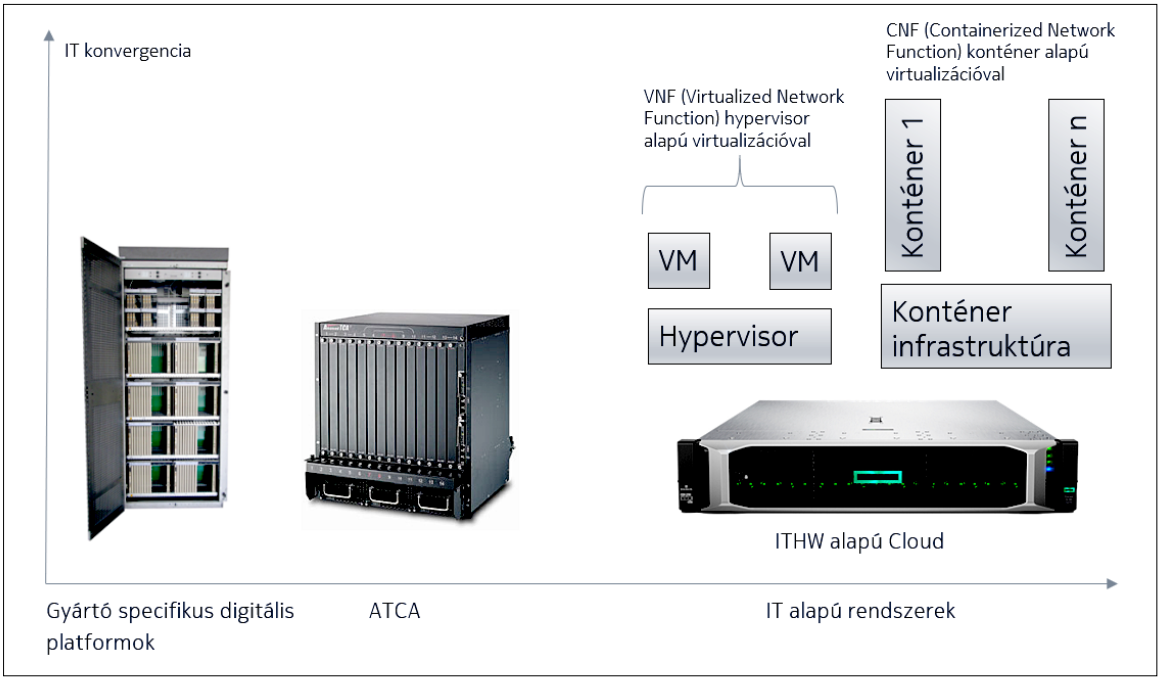
Kulcsszavak: cloud-infrastruktúra, nyílt forráskód, biztonság, sérülékenység, CVE

A távközlési iparágban használt infrastruktúra- és platform-megoldások egyre inkább konvergálnak az egyéb IT-megoldások által használt platformok irányába. Ez rengeteg előnnyel jár technikai és üzleti szempontból is, ugyanakkor olyan fenyegetések jelentek meg a távközlés területén is, amik eddig csak mérsékelten, vagy egyáltalán nem voltak jellemzőek. Mivel a telekommunikációs hálózatok kritikus rendszereknek tekintendők, különös figyelemmel kell kísérni ezen klasszikusan IT-eredetű, de immár a telekom-rendszereket is érintő fenyegetéseket. A teljesség igénye nélkül ide tartoznak a privát és publikus cloud-rendszerek általános sebezhetőségével kapcsolatos kérdések, a szabad vagy nyílt forráskódú szoftverek beépítése az infrastruktúrába és a telekom-applikációkba, illetve a privát cloudból a publikus cloud-rendszerek felé elmozdulás miatti koncepcionális változások.

1. Bevezetés

Elsődleges témánk a telekommunikációs hálózatok által használt infrastruktúrák bemutatása biztonsági szempontból, kitérve a legfőbb fenyegetési típusokra, amik manapság ezekre a platformokra leselkednek. Bár a cikk főleg technikai részletekre fókuszál, de a teljes kép kialakításához szükséges a technikai részletek mellett üzletmenetet érintő, folyamatirányítási és felelősséggel kapcsolatos kérdések tárgyalása is. Emellett érintőlegesen foglalkozni kell a telekommunikációs infrastruktúra részét nem képző, de annak biztonságát befolyásoló tényezőkkel is: magukkal a telekommunikációs applikációkkal, illetve a telekommunikációs szolgáltatásokat igénybe vevő eszközökkel.

Az írás első szakaszában a telekommunikációban használt infrastruktúra fejlődését tekintjük át röviden a biztonsági kihívások szempontjából, majd foglalkozunk az infrastruktúrát, illetve annak biztonságát érintő egyéb tényezőkkel. Ezt követően néhány komplex támadási típus bemutatására kerül sor: direkt az infrastruktúra elleni, vagy a megtámadott infrastruktúra felhasználásával további, tipikusan a telekommunikációs applikációk elleni támadásokhoz. Egy speciális veszélyforrás is részletesebben bemutatásra kerül, kifejtve miért jár veszélyekkel az OSS (Open Source Software) használata a telekommunikációs infrastruktúrában és applikációkban. Végül néhány, az infrastruktúra részét nem képző, de a hálózat biztonságát ettől még befolyásoló tényezőről is szükséges szót ejteni.



1. ábra A telekommunikációban használt infrastruktúra fejlődése.

2. A telekommunikációs infrastruktúra technikai fejlődésének történelmi áttekintése

A napjainkban a telekommunikációs infrastruktúrára leselkedő veszélyek megértéséhez szükséges áttekinteni, hogyan is jutottunk el a ma használt infrastruktúrákhoz, hogyan változott a fenyegetés jellege és mértéke ezen evolúció alatt. A későbbi szakaszok az ebben a szakaszban ismertetett fejlődési szakaszokra fognak hivatkozni a fenyegetettségek tárgyalásánál.

2.1. Kezdeti digitális megoldások

A telekommunikációs alkalmazások elterjedése digitális platformon a nyolcvanas évektől vált jelentőssé, amikor azok tipikusan valamilyen alkalmazás-specifikus HW- és SW-alapon futottak. Ahol lehetőség adódott rá, a gyártók igyekeztek valamilyen x86-alapú HW-t készíteni, de ezek attól még gyártó- és iparág-specifikusak maradtak. Az akkori teljesítménye az x86 architektúrának sok telekommunikációs alkalmazás esetén nem is tette lehetővé az arra való migrálást, ezért mindig is voltak olyan alkalmazások, amelyek inkább célorientált HW-en futottak, például valamilyen DSP-platformon. Ilyen DSP-n futó alkalmazások a mai napig léteznek, bár ezeknek a DSP-n tartása inkább üzleti döntésekre vezethető vissza. (A kifutó technológiákhoz tartozó alkalmazásokat tipikusan nem éri már meg migrálni a modernebb környezetbe.)

Az alkalmazások megírásához használt programnyelv is valamilyen az adott környezet és telekomspecifikus variáns volt (pl. gyártóspecifikus SDL-variánsok [1]), saját operációs rendszert használva.

Ezek a kezdetleges digitális platformok többszörös ráncfelvarráson estek keresztül, gyártói stratégiák függvényében 3G-alkalmazásoknál is még megtalálhatóak voltak a modernizált verzióik, de a ma futó telekommunikációs szolgáltatásokat nyújtó eszközök közül már teljesen eltűntek, a rajtuk futó releváns alkalmazásokat valamilyen modernebb platformra ültették át. Egy ilyen gyártóspecifikus digitális platformra jó példa a Nokia DX 200 [2].

2.2. x86-konvergencia, kezdetleges virtualizáció

Az x86-architektúra terjedésével megjelentek a legalább telekomon belüli platform-szabványosítási törekvések. Erre jó példa az ATCA-platform [3], sokszor itt már valamilyen Linux-alapú, de telekom-igényekre optimalizált operációs rendszert és kezdetleges hypervisor-alapú virtualizációs technológiákat használva. Ezek a megoldások már tekinthetőek a cloud-alapú infrastruktúra előfutárainak, de attól még jelentősen különböztek az alábbi pontokban:

• A használt ATCA HW ugyan a szabvány szerinti volt, de a legtöbb gyártó alkalmazásait csak saját ATCA HW-en támogatta. Tehát volt ugyan egy elvileg közös szabvány, amit a gyártók közösen próbáltak követni, de a szolgáltató szempontjából továbbra is összeforrott egység formájában történt a HW és rajta futó alkalmazás kiszállítása, a különböző gyártók elvileg például ATCA-specifikus komponenseit nem lehetett keverni, az egyik gyártó alkalmazását nem lehetett egy másik gyártó ATCA HW-környezetben futtatni a gyártói támogatás megtartása mellett.

• Amennyiben a használt operációs rendszer valamilyen Linux-alapra épült, rengeteg módosítást tartalmazott a telekom által megkövetelt speciális rendelkezésre állás, redundancia és valós idejű feldolgozási igények miatt. Az operációs rendszer, ide értve az esetleges hypervisort is, és a rajta futó alkalmazás elválaszthatatlan egységet képzett.

• A hypervisor-alapú virtualizáció itt még igen kezdetleges volt, mivel az többnyire statikus módon történt, az applikációt felépítő virtuális gépek leosztása többnyire előre definiált (az adott operátorra vonatkozóan), sokszor speciális igényű virtuális gépekkel, ami miatt a platform nem is mindig volt teljesen homogén.

2.3. IT-konvergencia

A következő logikus lépés a „telekom siló” felszámolása lett, cél az IT-területen már széles körűen és költséghatékonyan használt HW- és SW-komponensek integrálása:

– blade/rackmount alapú szabványos HW, ami már nem feltétlenül a telekom-beszállítótól kell származzon (IBM, Dell, HPE, ...);
– már széles körűen elterjedt operációs rendszerek használata, vagy legalább azok alapul vétele, amire aztán a beszállító elkészíttette a szükséges módosításokat (Ubuntu, Red Hat, ...);
– teljes hypervisor-alapú virtualizáció, a hozzá tartozó management- és kiegészítő funkciókkal (VMware NFV, OpenStack...).

A telekom-applikációknak azonban ezen a már szinte teljesen IT-alapú környezeten is szükségük van egy speciális VNF- (Virtualized Network Function) managementszintre, a legtöbb gyártó az ETSI MANO-t [4] követte ezen a területen. Mindezek mellett a speciális rendelkezésreállási és valósidejű feldolgozási igények miatt az operációs rendszer és a cloud SW szintjén továbbra is szükség volt apróbb módosításokra, amit vagy a beszállító saját maga végzett el, vagy partneri kapcsolatban a cloud SW beszállítóival közösen vittek végbe.

2.4. Cloud native infrastruktúra

A konténeralapú virtualizáció megjelenésével a telekommunikációs alkalmazások még közelebb kerültek az infrastruktúra felhasználása szempontjából az IT-iparághoz:

– szabványos IT HW,
– széles körben használt operációs rendszerek és cloud-infrastruktúra.

Az 5G-alkalmazások jelentős része már ebbe a környezetbe született bele, ezért sokkal könnyebb volt azoknak az implementálása ilyen módon úgy, hogy a cloud-infrastruktúra felé ne kelljen speciális telekom-jellegű igényeket támasztani, amiket azok a megfelelő tervezéssel és méretezéssel ne tudnának kielégíteni.

A beszállítók termékei között megtalálhatóak ugyan saját operációs rendszer és cloud SW-termékek is (ezek többnyire valamelyik, a piacon elérhető megoldás továbbgondolásai), de a fő hangsúly a telekom-applikáción van, a gyártók portfoliójában egyre kisebb szeletet képvisel a HW- és a SW-infrastruktúra. Amennyiben a beszállító portfoliójában vannak ilyen termékek, azokat már tipikusan nem csak a saját applikációihoz kínálja, hanem akár önmagában is, és mint HW- vagy teljes cloud-infrastruktúra megoldást árulja, ebben az esetben a klasszikus IT-beszállítók konkurenciája szerepkörben.

3. Üzleti folyamatok és percepciók változása, a fenyegetettség szintjének és jellegének változása az infrastruktúra fejlődése során

A technológia evolúciója mellett vagy részben annak a hatására is jelentős változások történtek a telekom-applikációk üzemeltetése terén is.

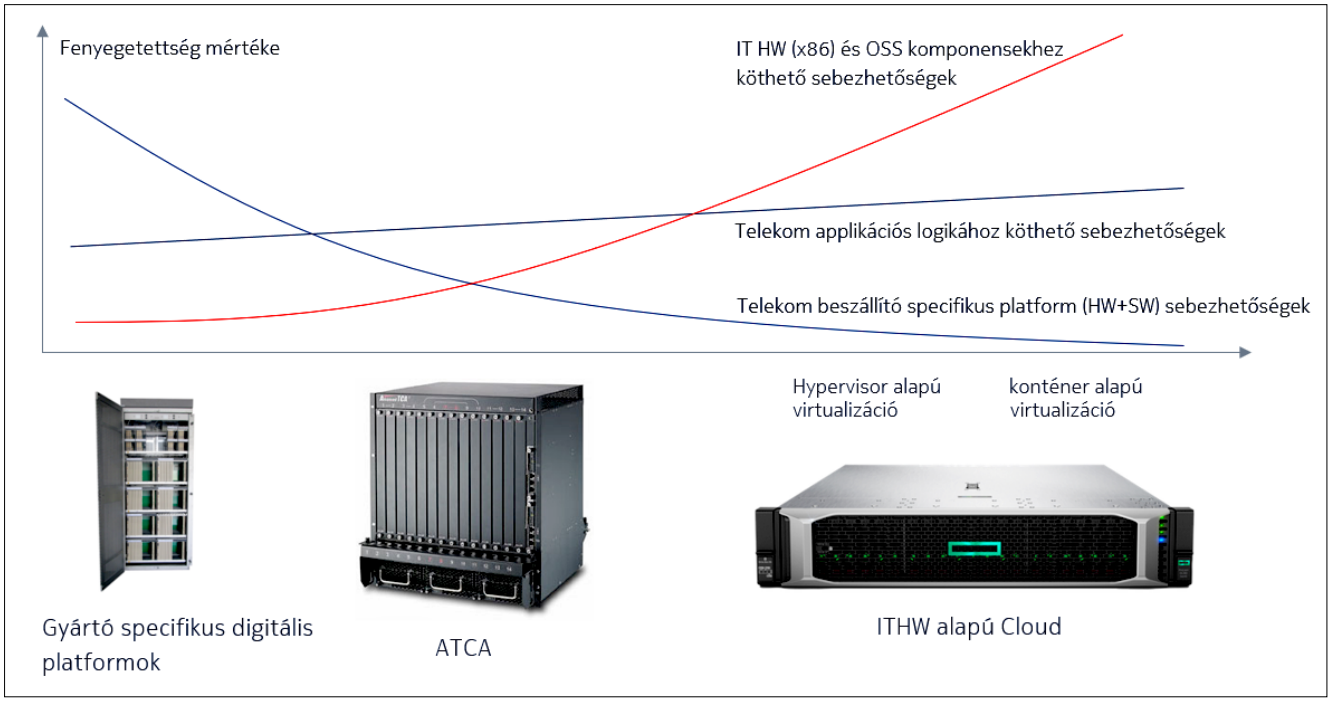
A korai időszakban az adott 2G-, 3G- és részben még a 4G-megoldásokat felépítő funkciók, vagy legalább a különböző alrendszerek egy beszállítótól származtak (pl. rádiós hálózat, core network, hálózatfelügyelet). Ennek köszönhetően üzemeltetési szempontból is azokra egy egységként tekintett mindenki, miszerint elég annak a határait megfelelően levédeni, azon belül egy DMZ-t (demilitarized zone) létrehozni, ahol már enyhébb követelmények vonatkoznak a biztonságra. Ne feledjük, hogy sok biztonsági követelménynek nagyon komoly kihatása lehet a teljesítményre is, tehát egy ilyen DMZ használata nem tekintendő könnyelműségnek, egyszerűen csak ez volt a beszállító és az üzemeltető közös érdeke az adott rendszer teljesítményének maximalizálása érdekében.

Az egyre komolyabb IT-irányú konvergenciának, egyre nyíltabb telekom szabványoknak és a növekvő komplexitásnak köszönhetően napjainkra ez a fajta gyártóhomogén környezet eltűnőben van. A DMZ határvonalai elmosódnak, egyre több gyártó egyre több terméke szükséges egy telekommunikációs hálózat megvalósításához. (Érdemes összevetni hány hálózati elem volt szükséges egy 2G-hálózat felépítéséhez és hány elemből épül fel egy 5G-hálózat.)

A határok védelme már nem elégséges, minden egyes funkciót vagy elemet külön, önmagában is fel kell ruházni az összes védelmi funkcióval. Fontos megjegyezni, hogy ebben a kontextusban a DMZ nem csak az elemek közötti kommunikációra vonatkozik, hanem a SW készítésének módjára is. Tehát nem csak arról van szó, hogy például IP szinten minden egyes elemet már önmagában is meg kell védeni, hanem azt is kontrollálni kell, hogy az adott elem fejlesztése, kiszállítása, telepítése és üzemeltetése során azzal mi történik: milyen beszállítói láncon keresztül milyen idegen komponensek kerülhetnek bele (lásd később az OSS-el foglalkozó szakaszt).

Fontos szerepe van az új technológiákhoz kapcsolódó percepcióknak is. Egy új technológia, mint például a konténeralapú virtualizáció megjelenésekor nem mindig annak a biztonságos üzemeltethetősége a legelső fő szempont. E cikk nem tér ki a hypervisor- és konténeralapú virtualizáció összehasonlítására, de általánosan elmondható, hogy a döntéshozók szemszögéből a konténeralapú virtualizáció jelenleg kevésbé tekintett biztonságosnak, mint a hypervisor-alapú, ezt több országspecifikus szabványban is nyomatékosították. Amikor egy infrastruktúra biztonságosságáról beszélünk, akkor az ilyen percepciókkal is foglalkozni kell, hiszen okkal vagy ok nélkül, de ezek is alakítják a biztonsággal kapcsolatos elvárásokat és irányelveket.

2. ábra A fenyegetettség változása a különböző infrastruktúrák esetében.



A fenti áttekintés után a technológia evolúció, üzletmenet és percepciók terén el is érkeztünk ahhoz a ponthoz, amikor ezek függvényében vizsgálható a rendszerre ható fenyegetettségi szint *(lásd a 2. ábrát)*.

Az első fontos észrevétel, miszerint a silóalapú kezdetleges, IT-tól távol álló infrastruktúra kevesebb, de legalábbis más jellegű veszélynek volt kitéve. Az IT felé történő migrációval a telekommunikációs iparág megörökölte az összes IT-sebezhetőséget is minden logikai szinten. Egy DSP-alapú platformra aránylag kevés támadási módszer ismert annak egyedisége és elszigeteltsége miatt, míg egy általános cloud-alapú infrastruktúra rengeteg módon támadható, jól dokumentált módszerekkel. Az IT-alapú rendszerekre már magasabb az infrastruktúrára leselkedő fenyegetési szint, ugyanakkor ezek jellege is megváltozott:

– a beszállító-specifikus HW-rel együtt az ahhoz kapcsolódó specifikus veszélyforrások is eltűnnek,
– az IT-alapú HW magával hozta az általános HW-sérülékenységeket (lásd a Spectre példáját az 5. szakaszban),
– a nagyfokú OSS- (Open Source Software) használat miatt azoknak a karbantartása a telekommunikációs alkalmazásokban egy soha véget nem érő küzdelemmé vált (lásd 6. szakasz).

Meg kell említeni a telekommunikációs applikációs logikát érintő fenyegetéseket is, amik a használt platformtól függetlenül jelen vannak. Mivel a telekommunikációs applikációk komplexitása növekszik, ezért ez a fenyegetettség is enyhén növekvő szinttel van jelölve a diagramon. (Például signaling protokollok sebezhetősége, részletesebben kifejtve az 5. szakaszban.)

(A diagramon a fenyegetési típusok trendje a fontos, nem feltétlenül azok egymáshoz viszonyított nagysága.)

4. Infrastruktúrát és az applikációs szintet érintő sérülékenységek

Mielőtt tovább elemeznénk az infrastruktúra támadhatóságát és annak főbb okait, szükséges jobban elkülöníteni az infrastruktúrára és az applikációra leselkedő veszélyeket, összehasonlítani és foglalkozni azzal, hogyan épül fel egy hatékony támadás, azt néhány példával illusztrálva. Mivel az infrastruktúra megörökölte az összes IT-sebezhetőséget, nem meglepő, hogy az ismert támadási metódusok jelentős része az infrastruktúrán keresztül indul (később esetlegesen behatolva az applikációs szintre is). Ennek főbb okai:

- Hatalmas installált bázis nagyon hasonló vagy teljesen egyforma HW- és SW-komponensekből felépítve.
- Jól dokumentált gyengeségek és támadási metódusok (pl. CVE [5] adatbázis).
- Toolkitek elérhetőek az ismert infrastruktúra sérülékenységek kihasználására, estenként ezeknek a kiaknázása még különösebben mély ismereteket sem igényel. Többször is előfordult már, hogy CVE formájában már elérhető volt egy sérülékenység leírása, annak a kihasználására már az automatikus eszközök is megjelentek azonnal, miközben az érintett sérülékeny komponens beszállítója még dolgozott a javításokon. Ez tulajdonképpen nyílt felhívás volt ezeknek a kihasználására ezen átmeneti időszak alatt, amíg különböző szinten átmeneti intézkedéseket kellett hozni a védelem érdekében.
- A később felhozott példák kapcsán látszani fog, hogy egy sikeres átfogó támadás esetében az infrastruktúra megsebzése (legyen az a CIA-hármas [6] bármelyik aspektusa) csak az első lépés. A tényleges applikációs logikát érintő támadások sokszor valamilyen infrastruktúrát érintő támadásra épülnek.

Az applikációs szint megtámadása már tipikusan jóval komplexebb feladat, sokkal mélyebb és specifikusabb tudást és eszközöket igényel:

- A telekommunikációs applikációk logikáját érintő sérülékenységek nincsenek annyira széleskörűen és nyíltan dokumentálva, mint a széles körben használt IT-rendszerek infrastruktúra-szintű sérülékenységei.
- A telekom-applikációk sikeres megtámadásához (ismét legyen az a CIA-hármas bármelyik aspektusa) sokkal részletesebb ismerete szükséges a távközlési szabványoknak, beszállító-specifikus megoldásoknak. (Signalling protokollok, ETSI-szabványok [7], architektúrák stb.)

5. Példák a támadások összetettségére

Ez a fejezet példákon keresztül mutatja be, hogyan is néz ki néhány tipikus támadási módszer. Nem foglalkozunk az olyan „egyszerű” módszerekkel, mint root-jogosultságok szerzése a jelszó vagy privát kulcs megszerzésével. A cél néhány ezeknél jóval összetettebb példa rövid bemutatása és annak illusztrációja, hogy egy sikeres támadáshoz többnyire nem elég egy adott sérülékenységet kihasználni, hanem arra valamilyen workflow-t kell építeni a kívánt cél elérése érdekében (attól függően, a CIA-hármas melyik eleme a cél).

5.1. HW-sérülékenység kihasználása

Egy HW-sérülékenység kihasználása önmagában még csak az első lépés szokott lenni. Jó példa erre a Spectre&Meltdown [8], illetve az L1TF [9] sérülékenység. Ezekkel a processzor regisztereiből, illetve cache-ből szedegethető össze információtöredék, amit felhasználva lehet folyatni a támadást. A Spectre esetében azonban ez egy rendkívül komplex folyamat.

A Spectre alapvetően az arra érzékeny processzorgenerációkban használta ki a hyperthreading korlátait, miszerint a HT esetében a processzorgyártók nem minden regisztert dupláztak meg. Szerencsés együttállás esetében az egyik félprocesszoron futó alkalmazás beleláthat a másik félprocesszoron futó alkalmazás adatainak egy részébe.

Ez azonban rögtön két kihívást is támaszt a Spectre sérülékenységét kihasználni szándékozó támadó számára:

1. Gondoskodni kell arról, hogy a kártékony alkalmazás pont a megfelelő időben, pont a megfelelő támadandó applikációval egy időben ütemeződjön egy cloud-infrastruktúra megfelelő szerverének megfelelő processzorára. Ehhez először el kell érni, hogy a kártékony alkalmazás egyáltalán felkerüljön az adott infrastruktúrára, és utána el kell érni, hogy a hypervisor pont a megfelelő módon ütemezze azt. Nem lehetetlen, de igen komplex kihívás. Ráadásul a kezdetleges védelmi megoldás igen hamar megérkezett: tiltani kell a hyperthreading használatát például BIOS-beállításokban, vagy a kezdetleges hypervisor-javítócsomagok ugyanezt tették titkoltan. Nem véletlen, hogy az első, kezdetleges átmeneti javítások 20-40%-os teljesítménycsökkenést jósoltak, ez a pont annyi, mint amit a hyperthreading használata általában adna pluszban. A hypervisor patch gyakorlatilag letiltotta a hyperthreading használatát. A későbbi javítások természetesen sokkal kifinomultabbak lettek, biztosítva azt, hogy az ütemezés során egy processzor például két hyperthread felére csak azonos virtuális géphez tartozó processzek ütemeződjenek (Sibling Scheduler).

2. Ha valamilyen csodával határos együttállásnak köszönhetően mégis sikerül a Spectre segítségével részadatokat gyűjteni egy másik applikációból, azokból valami ténylegesen használható konzisztens információt generálni szintén komplex feladat, ami a megtámadott applikáció mély ismeretét igényli.

Mivel a Spectre estében már a sebezhetőség kihasználása is rettentően bonyolult, és a segítségével szerzett adatok használhatóvá tétele még bonyolultabb, nem csoda, hogy bár ezen sérülékenységek néhány éve nagy publicitást kaptak, általuk végrehajtott valóban sikeres támadásokról nem található információ a cikk írásának idején.

5.2. SS7 signalling és más protokoll-szintű támadások

A napjainkban használt telekom- és internetprotokollok jelentős része a 80-as évekből származik. Ugyan átestek különböző frissítéseken, új verzióik jeletek meg, de az alaplogika többnyire nem változott jelentősen az elmúlt évtizedekben. A kezdeti verziókba nem ritkán mai szemmel nézve csak korlátozottnak nevezhető védelmi mechanizmusok voltak beépítve, ugyanakkor ezek nagyon jól szeparált átviteli hálózatokon futottak (akkoriban sokszor nem is IP-transzporton). Később, amikor ezek a protokollok átkerültek IP-transzportra, vagy a már amúgy is IP-alapú protokoll publikus(abb) IP-hálózatra került, a transzport védelme IP-szinten ugyan megvédte a protokoll átvitel alatti integritását, de a protokoll logikájának alapvető sérülékenységét nem szüntette meg, a „poisoning” és hasonló módszerek továbbra is használatosak [10].

Erre egy példa a Syniverse feltörése [11]. Ez a cég többek között szöveges üzenetek továbbításával foglalkozik. A publikus információk között ugyan nem lehet arra konkrétumukat találni, hogy a megtámadott platformon a szöveges üzenetekhez milyen szinten férték hozzá, de mivel alapvetően az SMS is MAP-protokollra épül, gyanítható, hogy itt is valamilyen SS7/MAP-protokollgyengeséget használtak ki, vagy a megtámadott platformon esetleg a szöveges üzenetek továbbküldése alatt hozzáférhettek a tartalomhoz is. Az irodalomjegyzékben bővebb információ is található az SS7 protokollok támadási módszereiről [10].

Ez a példa arra is felhívja a figyelmet, hogy egy összetett támadás során lehetséges, hogy az infrastruktúra és az applikáció megtámadása is csak egy eszköz, a valódi cél egy a telekomot használó réteg támadása, mint például SMS-t használó banki multifaktoros hitelesítő rendszerek az SMS-ek lehallgatása vagy eltérítése által.

Egy másik példa a tavaly történt Facebook-leállás [12]. Ebben az esetben ugyan semmilyen jel vagy nyilatkozat nem utal arra, hogy ez valóban szándékos támadás lett volna, viszont a leállás oka szintén visszavezethető a BGP-protokoll gyengeségére. Az elérhető információk alapján ebben az esetben egy félrekonfigurálás történt, de a BGP-protokoll működéséből adódóan a helyreállás jelentős időt vett igénybe. Egy sikeres infrastruktúraszintű támadás második lépéseként a megfelelő jogosultságok megszerzése után ez egy remek szándékos támadási kísérlet eredménye is lehetett volna a rendelkezésre állás ellen.

6. OSS (Open Source Software) és egyéb „3rd party”-komponensek használata

Az előző fejezetek alapján látható, hogy a technológiai és üzletmenetbeli fejlődések hatására hogyan nőtt a telekom-infrastruktúrák sebezhetősége, ugyanakkor milyen komplex folyamat tud lenni egy támadás. Röviden azt is érintettük, hogy a sérülékenység elleni védelem már a teljes szoftverfejlesztési életciklust át kell járja. Ez a fejezet a szoftverfejlesztési trendekben rejlő biztonsági problémákra hívja fel a figyelmet.

Az IT irányába mutató konvergencia nem csak a HW- és infrastruktúra szintű SW-használatban figyelhető meg, hanem az applikáció fejlesztési módszerek területén is. Az egyik ilyen legfontosabb trend az OSS-komponensek elterjedt használata minden szinten.

Infrastruktúra esetében:

– Host OS.
– Cloud infra SW (hypervisor és/vagy container alapú virtualizációhoz).
– Néhány BIOS- és FW-jellegű komponensben. Ezekben ugyan nem olyan elterjedt az OSS-komponensnek használata, de ha mégis egy ilyen szintű HW-közeli modulban jelenik meg OSS által okozott sérülékenység, annak a következménye még súlyosabb.

Applikációs SW szinten: az applikációs réteg elvileg kizárólag az adott telekomfunkcióra kellene koncentráljon, de technikai okokból itt is szükség van olyan rétegekre, ahol az OSS- komponensek elterjedt használata jellemző:

– guest OS hypervisor alapú virtualizáció esetén,

– micro OS konténeralapú virtualizáció esetén,
– különböző, már az applikáció által direktben használt megoldásoknál (adatbázisok, webszerverek, kommunikációs modulok stb.).

Az OSS-komponensek használatának vitathatatlan előnyei vannak a szoftverfejlesztés során [13]. Az elterjedten használt funkciók, amik szinte minden applikációban felbukkannak (adatbázis-kezelés, webszerver, kommunikációs interfészek, AAA stb.) többnyire elérhetőek valamilyen OSS-komponens implementációban. Logikus lépés ezeket felhasználni, így a fejlesztés felgyorsul, olcsóbb lehet, konvergensebb lesz, és ami a legfontosabb, hogy a beszállító koncentrálhat az applikáció által képviselt tényleges funkciók implementálására. (A cikk nem foglalkozik az OSS-komponensek különböző felhasználási, jogi feltételeivel.)

Biztonsági szempontból azonban az OSS-komponensek használata jelentős kockázati forrást jelent, amivel a gyártónak és az üzemeltetőnek is foglalkoznia kell. Amennyiben bármelyik felhasznált OSS-komponens sérülékenynek bizonyul, az potenciálisan veszélyezteti az összes applikációt is, amelyekben az adott OSS-komponens adott verziója került beépítse.

Egy átlagos telekom-applikációban az OSS-komponensek százasával, nem ritkán ezresével vannak beépítve, ezért a legelső követelmény, hogy a gyártó (és később az üzemeltető is) tisztában legyen azzal, pontosan milyen OSS-komponenseket is épített be. Ez az egyszerűnek hangzó követelmény a valóságban nem mindig ilyen egyszerű, gondoljunk csak a különböző tranzitív függőségekre [14], vagy az adott gyártó saját komponensei közötti függőségekre. A felhasznált komponensek listázására különböző scannerek állnak rendelkezésre, amik vagy a szoftverfejlesztés során vagy akár a már kész termék (pl. cloud image vagy konténer) szinten képesek listázni az összes beépült OSS-komponensverziót. Ezeket elterjedten használják a beszállítók és az üzemeltetők is, az automatikus kiszállítási és integrálási folyamatoknak (CI-CD) [15] ezek az eszközök is már tipikusan szerves részei.

A fent említett eszközök nem csak listázni tudják a felhasznált OS-komponenseket, hanem azt is meg tudják nézni, hogy kapcsolódik-e az adott verzióhoz valamilyen addig ki nem javított sérülékenység. Ez az ellenőrzés tulajdonképpen egy adatbázis-alapú ellenőrzés, az adott OSS-komponens verzió alapján egy keresés például CVE-jellegű adatbázisban, de a különböző scannereket gyártó cégeknek saját adatbázisaik is vannak.

Fontos megjegyezni:

• Az adott termékre az adott OSS-komponensekre a scanner által talált sérülékenységek csak potenciális sérülékenységekként kezelendők (original score a CVSS alapján [16]). A valóságban ugyanazon OSS-komponens sérülékenységi hatása teljesen eltérő lehet attól függően, hogy az adott OSS-komponens milyen applikációba van beépítve, pontosan milyen funkcióit használják annak. (Expert score, a beszállító által végzett vizsgálat alapján, mivel egy sérülékenység vizsgálata során annak hatásláncát a valódi kontextusban szükséges elemezni). Tehát lehet például az automatikus integrációt blokkolni, amennyiben egy scanner valamilyen potenciális sérülékenyéget talál a vizsgált applikációban, ahova az OSS-komponens be van építve, de a valós fenyegetettség megállapításához mindig további vizsgálatok szükségesek.

• A tradícionális scannerek által végzett vizsgálat többnyire csak adatbázis-alapú ellenőrzést tartalmaz. Nem próbálják meg valóban leteszteltni vagy kihasználni az adott OSS-komponens sebezhetőségét. Gondoljunk csak bele, mennyire komplex feladat lenne például egy Spectre sebezhetőségi vizsgálata. Ehelyett egyszerűbb megnézni, hogy a felhasznált OSS-komponensverzió a CVE-adatbázis információi alapján már javítva lett-e, vagy még mindig sebezhető a Spectre által, vagy van e rá más nyitott sebezhetőség.

• Az OSS-komponensek keresése a scannerek által többnyire statikus, offline scannelés, nem igényli az applikáció futtatását, elég az image-eket scannelni. Nem összekeverendő a system hardening-el [17], amikor is egy futó alkalmazáson, futásidőben végezünk el bizonyos biztonsági ellenőrzéseket. A piacon elérhető scannerek nagy része kombinálja ezt a két funkciót: statikus analízis az applikáció image-re és dinamikus analízis a futó alkalmazásra.

Nézzünk egy konkrét példát, mi történik, amikor egy OSS-komponensre új sebezhetőséget jelentenek be. 2021 során az egyik ilyen legkomolyabb eset az év végén felfedezett log4j sérülékenység volt [18]. A nagy publicitásnak köszönhetően a bejelentés futótűzként terjedt el az üzemeltetői és beszállítói oldalon egyaránt. A beszállítók fel vannak készülve az ilyen esetekre és az új sérülékenységet a megfelelő processzeik alapján kezelik:

1. Még mielőtt egy új sérülékenység megjelenne, bármely időpillanatban szükséges egy friss belső adatbázis megléte, pontosan listázva, melyik termék milyen OSS-komponenseket használ. Emellett szükséges egy adatbázis a korábbi analízisek eredményeinek nyilvántartására (original score vs. expert score).
2. Amikor egy új sérülékenység megjelenik (pl. a lpg4j 2021 decemberében), le kell ellenőrizni, hogy a sérülékenynek ítélt OSS-komponens be van-e építve valamelyik termékbe.
3. Amennyiben a potenciálisan sérülékeny OSS-komponens jelen van, részletesebb elemzés szükséges, hogy a potenciális sérülékenység milyen hatással lehet a termékre, szükséges-e valamilyen javítás, kell-e frissíteni az OSS-komponens verzióját. Vizsgálni kell, hogy szükséges-e valamilyen átmeneti biztonsági intézkedés foganatosítása, amíg az új, javított OSS-komponensverzió nem áll rendelkezésre, illetve az frissítésre és kiszállításra kerül a termék egy javított verziójában, amennyiben a sérülékenyég valódi fenyegetések bizonyul a termékre nézve.
4. Dokumentálás, kiszállítás.

A példában említett log4j komponens egy nagyon széles körűen használt OSS-modul, ezért szinte minden te-

lekom-beszállító valamilyen applikációjában jelen van valamilyen szinten. Ez lehet direkt használat, de akár tranzitív függőség miatt is. A sérülékenynek minősítésnek köszönhetően minden beszállító rögtön megkezdte az analízist, a szükséges döntések nagy része még december folyamán megszületett (jelent-e valós veszélyt egy adott applikációra, ha igen szükséges-e új verzió kiszállítása, vagy van egyszerűbb megoldás is a sérülékenység elhárítására), ennek megfelelően amennyiben új verzió kiszállítása szükséges, az a cikk írásakor folyamatban van. A helyzetet bonyolította, hogy a log4j (2.x) ugyanazon sérülékenységéhez kapcsolódóan csak december folyamán négy CVE keletkezett, ami négy egymást követő log4j verziót eredményezett (2.15–2.17.1 a cikk írásának idején), tovább bonyolítva az integrálási és kiszállítási terveket.

Az OSS-sérülékenységek részletesen dokumentálva vannak (pl. CVE). Ez jó hír az elhárítás szempontjából, könnyebb eldönteni a fenyegetés mértékét a felhasználó applikáció szempontjából, könnyebb követni az OSS-hez kapcsolódó korrekciók állapotát, és úgy általában könnyebb belső folyamatokat építeni az új sérülékenységek kezelésére is.

Ugyanakkor ez a nyílt és rapid kommunikáció extra veszélyforrást is jelent, mivel – szintén a log4j példánál maradva –, a sérülékenységet kihasználók is részletes információkat kapnak nagyon gyorsan, a leírás alapján nem nehéz megfejteni, hogy milyen alkalmazásokban és milyen területeken használják a sérülékeny komponenst (log4j-t szinte mindenhol), és rögtön támadások indíthatók, amik megpróbálják kihasználni az átmeneti helyzetet, amíg a sérülékenységet nem foltozzák be, vagy átmeneti biztonsági intézkedéseket nem hoznak. Nagyon sok új sérülékenység esetén ráadásul még a korrekciók megjelenése előtt elérhetővé válnak automatizált eszközök a sérülékenység kihasználására, így különösebb előképzettség nélkül, nagyszámú rendszeren lehet próbálkozni, hátha valamelyikre nem jutott még el a szükséges adott korrekció.

Az OSS-komponensekkel kapcsolatos sérülékenységek kapcsán érdemes azt is elemezni, hogyan alakul ezek száma az applikációs SW életciklusa alatt (amibe az OSS-komponensnek be vannak építve).

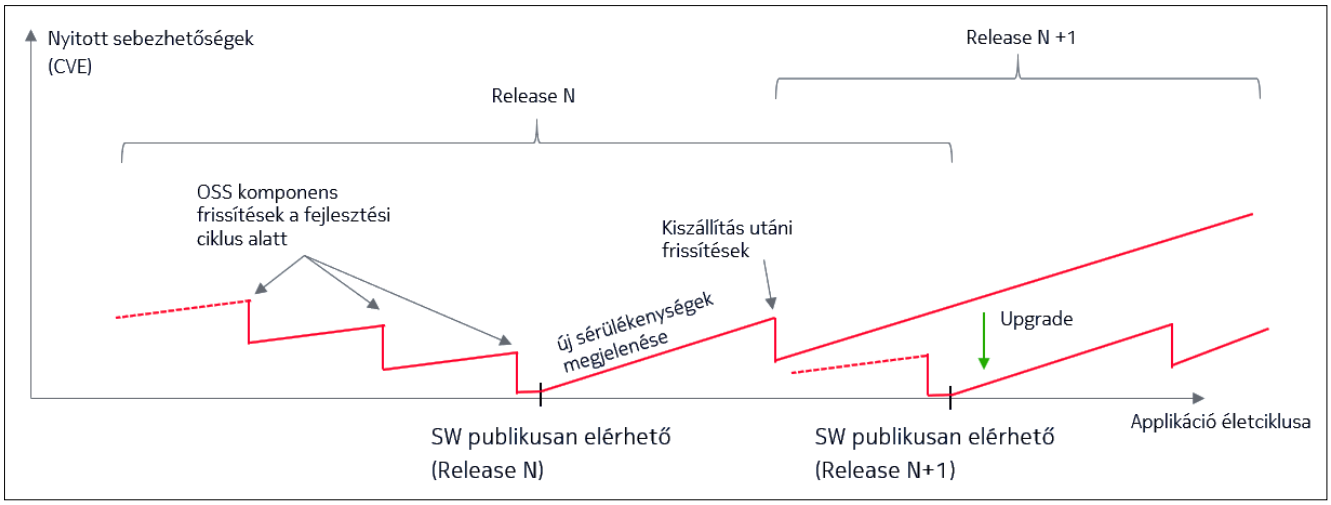
Mivel a SW életciklusa alatt a felhasznált OSS-komponensekre folyamatosan válnak ismertté új sebezhetőségek, azok száma a fejlesztési ciklus alatt és a kiszállítás után is folyamatosan növekszik. A *3. ábrán* ez a növekedés lineárisnak van feltüntetve, de a valóságban ez bármilyen jellegű növekedés lehet. A „nulla” sebezhetőséghez OSS-szempontból a legközelebb a publikusan elérhetőség kezdetén áll az applikáció. Ez a valóságban nem mindig szokott nulla lenni, különböző okok miatt, mint például:

– Lehetséges, hogy egy új CVE kapcsán még nem áll rendelkezésre a javított OSS-verzió. Ilyen esetben a beszállítónak el kell döntenie, tovább engedi-e a szoftvert, és majd később szállítja ki a korrekciót egy frissítés formájában, amint az OSS korrekció elérhetővé válik, vagy késlelteti az applikáció kiszállítását.
– A fentiekben szót ejtettünk arról, hogy a scannerek által talált sérülékenységeket potenciális sérülékenységként kell kezelni, és csak az applikációspecifikus analízis után dönthető el annak valós mértéke az applikációra nézve. Ezért a SW életciklusa alatt bármelyik időpillanatban előállhat az a helyzet, hogy a scannerek ugyan találnak valamilyen sérülékenységet az applikáció tisztán statikus, metaadatok felhasználása alapján történő átvizsgálása alapján, de a részletes analízis alapján azok a sérülékenységek nem jelentenek valós veszélyt, vagy a sérülékenységhez rendelt CVSS-pontszám [19] csökkenthető.

A nyitott OSS-sebezhetőségek száma az applikációban tipikusan az OSS-komponensek verzióinak frissítésével csökkenthető. A SW kiadása előtt ez a fejlesztési fázis szerves része, célul tűzve ki a kiadás pillanatában minimalizálni az applikációban lévő sérülékenységeket, a kiszállítás után pedig javítócsomagok formájában.

Amikor az adott applikáció új verziója elérhetővé válik, az természeténél fogva már kevesebb nyitott sérülékenységet fog tartalmazni a kiadás pillanatában, illetve

3. ábra OSS-sérülékenységek alakulása a SW életciklusa során.



a beszállítók is leginkább az aktuális vagy néhány legutolsó applikáció verzióra teszik elérhetővé a frissítéseket, ezért az OSS-hez kapcsolódó sérülékenységek minimalizálása céljából is érdemes az applikáció újabb verziójára váltani, általában a karbantartási szerződések is erre terelik a telekommunikációs applikációk üzemeltetőit.

OSS-felhasználási szempontból tehát a konklúzió az, hogy az egyértelmű előnyök mellett nem szabad elfeledkezni az általuk jelentett potenciális biztonsági fenyegetettségről sem, mivel manapság a telekom-termékeket érintő valós fenyegetettségek jelentős része valamilyen OSS-komponens által előidézett sérülékenységre vezethető vissza. (Nem számolva ide az alapvető biztonsággal kapcsolatos irányelvek be nem tartását a fejlesztés-kiszállítás és üzemeltetés során.)

Amennyiben egy beszállító OSS-komponensek felhasználása mellett dönt, elengedhetetlen az ehhez szükséges háttér folyamatos biztosítása is a biztonság megőrzése érdekében, mivel az OSS-t tartalmazó applikáció sebezhetetlensége folyamatosan inflálódik az életciklusa során. Ennek főbb követelményei:

– megbízható adatbázisok a felhasznált OSS-komponensekről és azokhoz tartozó korábbi elemzések eredményeiről,
– magas fokú automatizáció a gyors reagálás érdekében új sérülékenység megjelenése esetén,
– scannerek és más biztonsággal kapcsolatos eszközök integrációja a fejlesztési-kiszállítási láncba (CI-CD),
– megfelelő, transzparens kommunikáció a beszállító és az üzemeltetők között,
– szükséges erőforrások tervezése, a már kiszállított termékek teljes élettartama alatt a felhasznált OSS-szoftverkomponensekre jövőben felfedezett sérülékenységek foltozására.

A fentiekben ugyan az OSS-komponensek által jelentett kockázatokkal foglalkoztunk, de hasonló függőség fennállhat bármilyen, a beszállító által más gyártótól vásárolt szoftverkomponens esetében is. Ebben az esetben a jelentős különbség az SLA (Service Level Agreement) megléte, szemben az OSS-jellegű felhasználással.

7. Infrastruktúrán kívüli eszközök

Az előző szakaszok kizárólag telekom-infrastruktúrára koncentrálnak. A kezdetleges digitális rendszereknél a biztonsági kihívások nagy része valóban az infrastruktúrához kapcsolódott inkább, ide nem értve a terminál (mobiltelefon) fizikai megszerzését, vagy az abban tárolt adatok megszerzését valamilyen egyszerű klónozási módszerrel, amikor a fizikai eszköz megszerezhető. Ez főleg annak volt köszönhető, hogy ezek a végfelhasználói eszközök túlságosan buták voltak, üzemeltetési szempontból többnyire elég volt a köztük lévő szabványos interfészek megfelelő védelme.

Napjainkra azonban a végfelhasználói eszközök is hatalmas fejlődésen mentek keresztül, az IT-eszközök felé történő konvergencia itt is megfigyelhető, a gyártóspecifikus operációs rendszert használó kevésbé okos eszközök jelentős része a telefonok közül eltűnt, gyakorlatilag két fő opció maradt meg: Android és iOS.

Az infrastruktúrára leselkedő veszélyek nagy része ezekre is érvényesek, hiszen ezek is nagy mértékben használnak OSS-komponenseket, a nagyfokú hasonlóság miatt egy esetleges sérülékenység egyszerre túl nagy részét érintheti a használatos eszközöknek.

Az infrastruktúra beszállítója és a telekommunikációs hálózat beszállítója néhány kivételes esettől eltekintve (blokkolás, terminálmenedzsment-rendszerek használata) nem tehető felelőssé ezeknek az eszközöknek a biztonsági szempontból napra készen tartásáért, az nagy részben a felhasználó és az eszköz gyártójának a felelőssége.

Ezen cikk elsődlegesen ugyan a telekommunikációs hálózatokkal foglalkozik, de amikor ezen hálózatok sebezhetőségét tárgyaljuk, nem hagyhatjuk figyelmen kívül, hogy a telekom-hálózatok elleni támadások jelentős része a valóságban nem is a hálózatot támadja, hanem magát a végfelhasználói eszközt, gondoljunk csak a nagy figyelmet kapott NSO „zéro click”-megoldásaira az elmúlt évből [20]. Az ilyen és hasonló megoldásokkal kifejezetten a végfelhasználói eszközöket támadják, kihasználva azoknak a nagyfokú hasonlóságát és moduláris, sokszor OSS-komponenseket is tartalmazó felépítését.

Az 5G elterjedésével ez a probléma még több figyelmet kell kapjon. Az 5G egyik jelentős felhasználási területe az IoT-kommunikáció, ahol 5G-modemeket használnak. Tehát végfelhasználói eszközök alatt nem csak a tipikusan Android/iOS-alapú okostelefonokat értjük, hanem az 5G-modemeket is, amik a sima okostelefonokra leselkedő veszélyek mellett még több problémával szembesülnek. Biztonsági szempontból pont ezek a veszélyesebbek mert:

– nagyon sok típus létezik, sokszor aránylag kevesebb példányszámban gyártva a nagy mennyiségben gyártott mobiltelefonokhoz képest;
– gyártói oldalról rövidebb idejű támogatás (az alacsonyabb példányszámok vagy egyéb végbemenő folyamatok miatt, mint pl. portfoliótisztítás, merger és felvásárlások miatt);
– ha rendelkezésre is áll a biztonsági javításokat tartalmazó FW, ritkább frissítés nemtörődömség vagy technikai limitációk miatt (pl. távoli FW-frissítés nem mindig támogatott vagy a nem hivatalos FW-en futó eszközök).

Amikor biztonsági szempontból beszélünk elsődlegesen adatátviteli eszközökről, például modemek esetében, akkor érdemes felhasználni a Wi-Fi-modemekkel kapcsolatban már megtanultakat. Bár a rádiós technológia különböző, a kihívások hasonlóak. Éppen ezért ami kihívásokat a mostani Wi-Fi-eszközök támasztanak, azokkal lehet számolni az egyre nagyobb számban elterjedő 5G-modemek eseteben is, például az IoT-alkalmazásokban:

• „Don’t upgrade if not broken” – egy átlagos felhasználó milyen sűrűn frissíti például a mosógépében vagy

klímájában lévő modem SW-verzióját? Egy ipari környezetben elvárható, hogy egy autógyár gyártósorán a felhasznált adatátviteli eszközök az IoT-láncolatban naprakész biztonsági frissítéseket kapjanak, de egy átlagos felhasználó estében ez már nem biztosított, amíg valami konkrét hiba nem lép fel, általában nem szokták az ilyen eszközök SW-ét frissíteni.

• Mi a célja a támadásnak? (Ez igaz a telefonokra is, nem csak a modem-jellegű eszközökre). Az adott eszközön (pl. egy telefonon vagy ipari kamerán) megtalálható-átfutó információ megszerzése, vagy a megtámadott eszköz csak azért fontos, mert annak van közvetlenebb hozzáférése a valóban megtámadni kívánt hálózathoz? (Például egy színes LED-szalagot vezérlő szerkezet Wi-Fi-kontrollal önmagában nem különösen csábító célpont, de ha ez az eszköz közben hozzáfér egy belső hálózaton folyó kommunikációhoz, akkor már jó ötletnek tűnhet azt snifferként vagy poisoning célokra használni, vagyis a megtámadott eszköz nem a cél, csak egy belépési pont.)

A fentiek alapján a végfelhasználói eszközük sebezhetősége ugyan elsődlegesen a felhasználók problémájának tűnhet (és többnyire az is), de a valóságban a megtámadott végfelhasználói eszközök jelenléte a hálózati infrastruktúrára is veszélyt jelenthet (nem is beszélve a felhasználói elégedetlenségről, amikor a megtámadott végfelhasználói eszközök miatti problémákért a szolgáltatót próbálja a felhasználó felelőssé tenni).

A nem rendeltetésszerűen működő eszközök az infrastruktúra elleni támadásoknak az egyik kiindulópontja is lehetnek. Ilyen veszélyek lehetnek például nagy mennyiségű megtámadott telefon botnetté alakítása, DDOS-támadásra való felhasználása a hálózat ellen.

8. Összefoglalás

Az elmúlt évtizedek során a telekommunikációs infrastruktúrák hatalmas fejlődésen mentek keresztül. Miközben a tényleges applikációs logika (2-3-4-5G) egyre komplexebb lett, egyre több hálózati elem egyre többrétűbb egymáshoz kapcsolódását igényelve, a használt infrastruktúra a kezdetleges gyártóspecifikus HW- és SW-kombinációtól eljutott a teljesen IT-alapú komponensek széles körű felhasználásához. A vitathatatlan előnyök mellett, mint pl. költségoptimalizálás, fejlesztési sebesség, karbantarthatóság, a legújabb technológiák gyors bevezetése és az IT-konvergencia a lehetséges fenyegetések jellegét is megváltoztatta.

A telekom-applikációs logikát fenyegető veszélyek mellett már legalább ugyanolyan, vagy még annál is nagyobb mértékben kell foglalkozni az infrastruktúrát érintő veszélyforrásokkal. Ezeknek célja lehet maga a telekom-infrastruktúra (CIA-elemek bármelyike), valamely az adott telekom-infrastruktúrát használó telekommunikációs alkalmazás logikája, felhasználva az infrastruktúra gyengeségeit, vagy akár egy telekom-szolgáltatásokat használó, de teljesen más iparág részét képző szolgáltatás (például pénzügyi szektor, gyártásautomatizálás, egészségügy, fogyasztói eszközök stb.).

Hivatkozások

[1] SDL: *https://portal.etsi.org/Services/Centre-for-Testing-Interoperability/ETSI-Approach/Specification-Languages/SDL*
[2] Nokia DX 200: *https://en.wikipedia.org/wiki/Nokia_DX_200*
[3] ATCA: *https://en.wikipedia.org/wiki/Advanced_Telecommunications_Computing_Architecture*
[4] ETSI MANO: *https://www.etsi.org/technologies/open-source-mano*
[5] CVE-adatbázis: CVE-CVE (mitre.org)
[6] CIA-hármas: What is the CIA Triad and Why is it important? | Fortinet
[7] ETSI-szabványok: ETSI – Standards, mission, vision, direct member participation
[8] Spectre & Meltdown: *https://meltdownattack.com/*
[9] L1TF: *https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html*
[10] SS7-támadások: A Step by Step Guide to SS7 Attacks – FirstPoint (firstpoint-mg.com)
[11] Syniverse story: Syniverse quietly admits it was hacked for five years | Light Reading
[12] Facebook story: *https://www.theverge.com/2021/10/4/22709260/what-is-bgp-border-gateway-protocol-explainer-internet-facebook-outage*
[13] OSS előnyei: *https://flosshub.org/sites/flosshub.org/files/Benefits%20and%20Drawbacks.pdf*
[14] Transitív függőségek: Transitive Dependency – an overview | ScienceDirect Topics
[15] CI/CD: What is CI/CD? Continuous Integration & Continuous Delivery in 2019 (katalon.com)
[16] CVSS: Common Vulnerability Scoring System SIG (first.org)
[17] Hardening: What is Systems Hardening? Read the Definition in our Security Glossary | BeyondTrust
[18] log4j: Log4j – Apache Log4j 2
[19] CVSS: *https://www.first.org/cvss*
[20] NSO zero click: Researchers call NSO zero-click iPhone exploit 'incredible and terrifying' | Engadget *https://www.news18.com/news/tech/explained-what-are-zero-click-hacks-and-why-are-they-such-a-menace-3988664.html*

Az ábrák elkészítéséhez felhasznált képek forrása:

ATCA HW illusztrálása: Security impact of the ATCA architecture adoption (webcast) – P1 Security

Gyártó specifikus HW illusztrálása: Nokia DX 200 | Core Network | Products | Carritech Telecommunications

IT HW illusztrálása: HPE ProLiant DL380 Gen10 server | HPE Store US

A szerzőről

CSORDÁS GÁBOR a BME Villamosmérnöki Karán diplomázott Irányítástechnikai és Robotinformatikai szakirányon. Másoddiplomáit bank- és pénzinformatika, majd IT-management területen szerezte. Fő szakterülete a beágyazott rendszerek mellett a telekommunikáció, több mint húsz éve ezzel foglalkozik. Részt vett az IP multimédiás alrendszer fejlesztésében és a telekom-applikációk cloud-platformra történő migrálásában. Később telekom-platformok és -infrastruktúrák tervezésével és méretezésével töltött közel tíz évet. Jelenleg a Nokia megoldásait érintő biztonsági kihívásokkal foglalkozik.