Hálózati protokollok biztonsági tesztelése

LÉCZ BALÁZS*, ZÖMBIK LÁSZLÓ**

** Budapesti Műszaki és Gazdaságtudományi Egyetem, Távközlési és Médiainformatikai Tanszék, lecz@alpha.tmit.bme.hu*

*** Ericsson Magyarország, BME-TMIT, laszlo.zombik@ericsson.com*

Reviewed

Kulcsszavak: biztonság, imprementációk, forgalomelterelés, konformancia

A protokollok és implementációik viselkedését több szempontból lehet vizsgálni, ezért igen sok tesztelési módszer létezik. A legszélesebb körben kutatott és alkalmazott módszerek az implementációk konformanciáját és teljesítményét vizsgálják. Cikkünkben a protokollok megvalósításainak biztonsági vizsgálatára koncentrálunk. Ismertetünk egy új, biztonsági tesztelésre alkalmazható módszert, majd bemutatjuk alkalmazásának lehetőségeit. Végül ismertetjük az általunk megvalósított szoftverkeretrendszert és bemutatunk néhány példát annak gyakorlati alkalmazására.

1. Protokoll tesztelési módszerek

Informatikai és távközlési rendszereink mindennapos, megbízható működése egyre jobban függ a kommunikációs protokollok és implementációik stabilitásától és biztonságától. Az infokommunikációs rendszereink általános biztonságának eléréséhez a protokolloknak meg kell felelniük bizonyos szintű biztonsági elvárásoknak, melyek nagyban függenek az adott protokoll alkalmazási területétől. Ezek teljesítéséhez a protokollokat igen körültekintően kell megtervezni és megvalósítani. Mind a tervezési, mind a megvalósítási fázis ki van téve az emberi hibáknak. Az implementáció során további veszélyforrásokat hozhatnak be a felhasznált szoftvereszközök (például egy hibás függvény-könyvtár felhasználásakor a könyvtárban levő hibák öröklődhetnek). A végterméket (protokoll specifikáció és kész termék) ezért minden esetben meg kell vizsgálni.

A jelenlegi, tesztelésre alkalmas szoftver és hardver eszközök egy-egy fő területet céloznak meg az alábbiak közül: konformancia, teljesítmény és biztonság. Léteznek eszközök, melyek a protokoll formális leírását veszik alapul, míg más eszközök a megvalósítást tesztelik valós vagy emulált környezetben. A vizsgálat módszere lehet formális verifkáció, szimuláció és a megvalósítás ellenőrzése. A biztonsági vizsgálatokhoz mind a három módszert alkalmazzák. Az alábbiakban bemutatunk néhány példát ezekre.

1.1. Formális protokoll verifikáció

A formális protokoll verifikáció elsődleges célja, hogy már a tervezési fázisban visszacsatolást nyújtson. Így a protokoll implementációjának elkészítése előtt fény derülhet a problémákra. Amennyiben a verifikáció hibát mutat ki, vissza kell lépni a tervezőasztalhoz (WEP [5]). További alkalmazásként említhető a már meglévő protokollok megfelelőségének vizsgálata: formálisan ellenőrizhető, hogy egy adott protokoll megfelelő-e egy bizonyos célra (például biztosítja-e az újrajátszás vagy a lehallgatás elleni védelmet).

A protokollok teljesítményének, hibatűrésének és bizonyos biztonsági paraméterek vizsgálatára már léteznek formális módszerek. Mivel a formális protokoll verifikációs módszerek a protokollok leírását használják csak fel, az implementációkban jelentkező hibák nem detektálhatók segítségükkel. Sok esetben fordult már el ő, hogy egy protokollt a formális ellenőrzés során biztonságosnak minősítettek, azonban az implementációba került programozási hiba – ami amúgy a funkcionalitást nem befolyásolta – biztonsági szempontból végzetesnek bizonyult. Rontja a helyzetet továbbá, hogy egy formális tesztelés során biztonságosnak nyilvánított protokoll hamis biztonságérzetet kelt a végfelhasználókban.

A formális ellenőrzéshez szükséges az adott protokoll helyes és teljes formális specifikációja, melyet legtöbbször az adott vizsgáló szoftver leírónyelvén kell megfogalmazni. Ilyen leírás elkészítése sok esetben igen összetett és nagy szakértelmet igénylő feladat. Ez a folyamat is ki van téve az emberi hibáknak, így előfordulhatnak hibás pozitív és hibás negatív eredmények is.

A formális módszerek legnagyobb hátránya, hogy a vizsgálható protokollok halmazát erősen leszűkítik a verifikációs eljárások által a protokollokkal szemben támasztott előfeltételek. Ezen megszorítások sok létező és tervezett protokollt kizárnak a vizsgálható protokollok köréből. További nehézséget jelent, hogy ezek az algoritmusok nem minden esetben garantálják a véges futásidőt. A gyakorlatban alkalmazott, többnyire komplex protokollok vizsgálatához igen nagy számítási- és memóriakapacitásra van szükség.

Megemlítünk néhány, biztonsági szempontból fontosabb, formális ellenőrző szoftvert: FDR [11], Casper [9], NRL Protocol Analyzer [10].

1.2. Szimuláció

A szimulációs módszerek igen elterjedtek, távközlési protokollok teljesítmény-vizsgálatára alkalmazzák a leggyakrabban (például, hogy egy adott protokoll eléggé hatékony-e), azonban biztonsági vizsgálatok elvégzésére is haszánlhatók ezek a szimulációs eszközök.

A legismertebb hálózati protokoll-szimulációs szoftver az NS2 [3]. Ezzel többek között szolgáltatásbénító, túlterheléses támadásokat is lehet szimulálni.

1.3. Protokoll implementációk vizsgálata

Különböző módszerek és eszközök léteznek a protokollok implementációinak tesztelésére. Minden egyes módszer a tulajdonságok egy jól körülhatárolható osztályára összpontosít. A protokollok megvalósítását ellenőrző szakemberek a *konformanciát és az együttműködési képességet* vizsgáló módszereket alkalmazzák elsősorban. A hálózatüzemeltetők által leggyakrabban alkalmazott eszközök a *protokoll-analizátorok* és a *behatolás-detektáló rendszerek* (Intrusion Detection System – IDS).

A biztonsági ellenőrző szoftvereket a hálózatbiztonsági szakemberek és a rosszindulatú támadók egyaránt használják. Ide sorolhatók a különböző *biztonsági letapogatók (security scanner), forgalom-generátorok* és a *biztonsági réseket kihasználó programok (security exploit).*

Az alábbiakban néhány mondatban bemutatjuk az eddig említett módszereket, valamint alkalmazhatóságukat a protokollok biztonsági réseinek kimutatásában.

Konformancia tesztelés: ezzel az eljárással a protokoll megvalósításának funkcionális helyességét vizsgáljuk. A teszt eredménye megmutatja, hogy az adott implementáció a specifikációnak megfelelően működike. A távközlési ipar által szorgalmazott trend a konformancia vizsgálat és szoftver-keretrendszerének szabványosítása felé mutat. Ennek egyik eredménye a TTCN3 [4]. A biztonsági szempontból fontos hibák egy része kimutatható ezen módszerek segítségével: a specifikáció félreértelmezéséből, programmozási hibákból adódó biztonsági rések nagy része felfedezhető ezzel az eljárással.

Együttműködési képesség vizsgálat: ezek a módszerek a különböző implementációk együttműködési képességét vizsgálják. A vizsgálat eredménye egyedül azt mutatja meg, hogy a két megvalósítás képes-e az együttműködésre. Biztonsági rések felfedezésére nem alkalmas ez a módszer.

Teljesítmény vizsgálat: a teljesítményvizsgálat során az implementáció viselkedését figyelik különböző terhelési feltételek mellett. Fő alkalmazása a különböző implementációk teljesítőképességének és hatékonyságának összehasonlítása, illetve a szűk keresztmetszetek felkutatása. Ennek ellenére egy biztonsági szempontból fontos tulajdonság vizsgálatára is használható: segítségével kimutatható, hogy a protokoll adott esetben érzékeny-e a túlterheléses (Denial of Service – DoS) támadásokra, illetve hogy az ez elleni védekezési módszer megfelelően működik-e.

Protokoll-analizátorok: céljuk, hogy valós időben megfigyeljék a hálózati forgalmat és el őre definiált szabályok szerint analizálják a csomagok tartalmát. Ezek a szabályok tartalmazhatnak protokoll adategységek értelmezésére vonatkozó információkat, így a protokollanalizátor ember által is olvasható formában képes megjeleníteni a csomagokat. A legtöbb protokoll-analizátort a hálózati vagy szoftveres hibák keresésére, illetve forgalmi statisztikák gyűjtésére fejlesztették ki. Közvetlenül nem alkalmazhatók biztonsági tulajdonságok vizsgálatára, azonban alapvető megfigyelő eszközként minden hálózattal foglalkozó szakember használja őket.

Elterjedten használt szoftverek:

`tcpdump` [8], `ethereal`.

Forgalom-elemzők (NIDS): a hálózati forgalomelemzők passzív hálózati eszközök, melyek gyanús tevékenység után kutatva folyamatosan figyelik a hálózati forgalmat. Amennyiben abnormális forgalmi szituációt vagy illetéktelen behatolást detektálnak, riasztják a hálózat üzemeltetőjét, vagy automatikus ellenlépéseket tehetnek. Ezek a rendszerek nem alkalmazhatóak közvetlenül biztonsági vizsgálatra, de az általuk felfedezett incidensek nyomán fény derülhet eddig ismeretlen biztonsági résekre is.

Példa: `snort`.

Biztonsági letapogatók (security scanners): ezek olyan aktív hálózati szoftverek, melyekkel egy adott hálózat vagy végpont sebezhetőségét lehet felmérni. Léteznek rendszer-specifikus és általános letapogatók is. Fő céljuk akár a célhálózat, akár a cél hoszt biztonsági réseinek felfedezése.

Az elterjedt szoftverek: `Nmap` [2], `Nessus` [1].

Forgalom-generátorok: olyan alapvet ő eszközök, melyekkel tetszés szerinti hálózati forgalom generálható. Intelligensebb fogralom-generátorok kiválthatnak egy vagy több kommunikációs felet vagy támadót. Önmagukban nem alkalmazhatóak biztonsági vizsgálatra, azonban a legtöbb biztonsági tesztelő szoftvernek részét képezik.

Biztonsági réseket kihasználó programok: céljuk, hogy ismert biztonsági réseket használjanak ki, általában rossz szándékkal. Elsődlegesen támadók használják őket, de a hálózatbiztonsággal foglalkozók is felhasználhatják azokat egy adott megvalósításban levő biztonsági hiányosság demonstrálására, illetve a megfelelő védekezési módszer kidolgozására. Használatuk csak az adott hiba felderítésére terjed ki.

2. A biztonsági tesztelés egy új megközelítése

Megvizsgálva az eddig kidolgozott biztonsági ellenőrző eljárásokat, arra a megállapításra jutottunk, hogy egy igen fontos terület nincs kellőképpen lefedve. A hálózati támadások egy része a beékelődésre épül, azaz a támadó kettő vagy több jóhiszemű kommunikáló fél közötti adatúton helyezkedik el. A meglévő módszerek általában nem képesek megmutatni az ilyen beékelődéses támadások hatásait, mivel az általuk alkalmazott hagyományos elrendezésben két fél kommunikál: az egyik a tesztelés alatt álló implementáció (IUT – Implementation Under Test), míg a másik maga a teszt eszköz. Ilyen elrendezés esetén a vizsgáló szoftvernek tel-

jes tudással kell rendelkeznie a protokollról, hogy a tesztelés alatt álló implementációval kommunikálhasson.

A valós életben gyakran előfordul, hogy több végpont bonyolít le forgalmat egy olyan hálózaton, mely teljesen, vagy részlegesen a támadó kezében van. Ilyen esetben a támadó megfigyelheti a felek kommunikációját, tetszőlegesen késleltetheti, eldobhatja, módosíthatja csomagjaikat, valamint generálhat tetszése szerinti csomagokat, akár más felhasználó nevében is.

A meglévő módszerekkel nem, vagy csak nehézkesen vizsgálhatók az e fajta támadások. Ezekre a szituációkra dolgoztunk ki egy általános, beékelődésre alapozott módszert. Az általunk alkalmazott vizsgáló elrendezés esetén a teszt szoftver képes végrehajtani ezeket a módosításokat, így vizsgálhatóvá válik az implementációk viselkedése beékel ődéses támadások esetén. A teszt szoftver itt az átviteli hálózat és a támadó szerepét tölti be. A módszer alkalmazásával emulálhatóak a hálózati problémák is (késleltetés, csomagvesztés, bithibák, csomagtöbbszörözés). Az elrendezés előnye, hogy a teszt szoftvernek nem szükséges implementálnia a vizsgálandó protokollt.

A következő pontban bemutatjuk az általunk megvalósított, beékelődéses elrendezésre épűlő biztonsági ellenőrzésre alkalmazható rendszert.

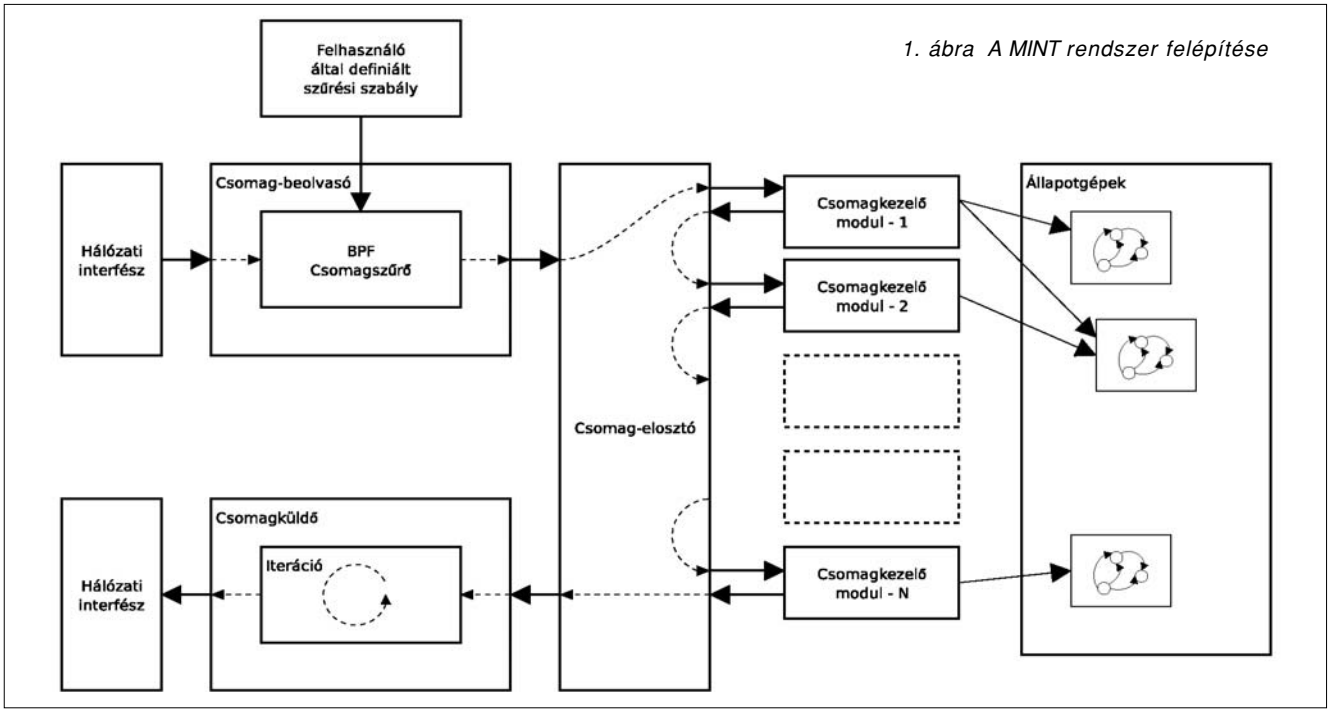
3. Megvalósítás

Fő célunk egy olyan keretrendszer megalkotása volt, mely általánosan alkalmazható hálózati protokollok implementációinak biztonsági teszteléséhez, beékelődéses elrendezésben. A tervezés során az alábbi elvárásokat fogalmaztuk meg a rendszerrel szemben:

- a hálózati csomagok kezelése adatkapcsolati szinten, ezzel a protokollfüggetlenség biztosítása;
- alapvető csomagtovábbító funkciók megvalósítása (útvonalválasztó és kapcsoló funkciók);
- a hálózati csomagok megkülönböztetése a felhasználó által megfogalmazott szabályok szerint;
- moduláris felépítés;
- általános programozási interfész (API) biztosítása és a dinamikus modulok kezelése az egyszerű bővíthetőség érdekében;
- a teszt eszköz felhasználója által betöltött modulok tetszőlegesen módosíthassák a rendszeren keresztülhaladó csomagokat;
- a felhasználó moduljai tetszőlegesen állíthassák egy csomag elküldéseinek számát (hogy emulálható legyen a csomagvesztés és csomagtöbbszörözés);
- a felhasználó küldhessen tetszőlegesen összeállított csomagokat;
- a felhasználó definiálhasson tetszőleges számú állapotgépet, melyeket a rendszer, illetve a felhasználó által generált események vezérelhetnek;
- a felhasználó rendelhessen össze eseménykezelő függvényeket az állapotgépek állapotátmeneteivel, illetve az állapotok belépési/kilépési eseményeivel.

Fejlesztési és futtatási környezetnek a C programozási nyelvet és a Linux operációs rendszert választottuk, a rendszernek a MINT nevet adtuk (MINT – Man-In-themiddle Networking Toolkit).

A rendszer kerneltől független, afelett futó program. A felépítést és működést szemlélteti az *1. ábra*. A MINT csomag-olvasó modulja a hálózati interfészről olvassa be az interfészre érkező csomagokat, majd a felhasználó által definiált szűrési feltételeknek megfelelőket továbbítja a csomageloszto modulnak. A csomag-elosztó sorban meghívja a felhasználó által definiált csomagkezelő modulokat, melyek megvizsgálhatják és tetszés szerint módosíthatják a csomagot. A csomag végül a



1. ábra A MINT rendszer felépítése

csomagküldő modulhoz kerül, mely a kimeneti hálózati interfészen elküldi a csomagot.

A fejlesztést gyorsította, hogy sok funkcióra már létezik jól működő, nyílt forráskódú függvény-könyvtár. Az általunk felhasznált könyvtárak a `libpcap` (hálózati csomagok alacsony szintű olvasása) [8], a `libnet` (hálózati csomagok összeállítása és alacsony szintű elküldése) [12] és a `libconfig` (hierarchikus konfigurációs fájl feldolgozása) [13].

A `libpcap` függvénykönyvtár a kernel hálózati szolgáltatásaitól függetlenül, alacsony szinten képes a hálózaton megjelenő csomagok beolvasására. A csomagok hatékony kezelésében segít a csomagszűrési szolgáltatása. Egy magas szintű, kényelmes leírónyelven megfogalmazott szűrési feltételt (például IP cím illetve TCP port alapú szűrés) képes lefordítani a kernelben található BPF (Berkley Packet Filter) szűrő byte-kódjára. Csak az így beállított szűrési feltételeknek megfelelő csomagokat továbbítja a kernel a programnak, így nem kell a tesztelés szempontjából irreleváns csomagokat kezelni.

Az állapotgépek hatékony megvalósításához nem tal áltunk megfelelő, szabadon felhasználható függvénykönyvtárat, ezért magunk készítettünk egyet. Az állapotgép szoftvermodul a rendszertől független, saját API-val és konfigurációval rendelkezik, így akár más szoftverekben is alkalmazható.

Alapmodulok

Megvalósítottunk néhány alapvető funkciót ellátó csomagmódosító modult:

Minta-modul: egy olyan modul, amely nem tölt be valós csomagkezelési funkciót, azonban prototípusként használható újabb modulok kifejlesztésénél.

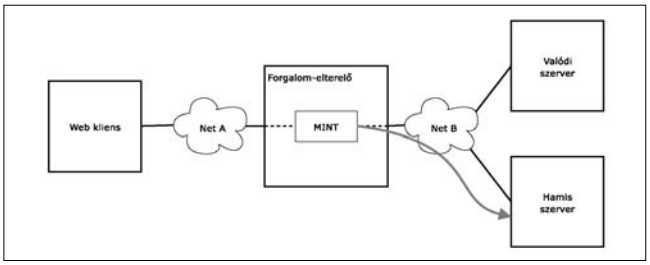
Stochasztikus hiba modul: a felhasználó által megadott hibaaránynak (BER – Bit Error Rate) megfelelően bithibákat illeszt a forgalomba. Használatával vizsgálható a protokollok hibatűrő képessége és így akár bizonyos DoS támadásokra való érzékenysége is.

Ethernet, TCP/IP fejrész-módosító modul: a kapcsoló- és forgalomirányító funkciók megvalósításához szükséges az adatkapcsolati réteg címzésének módosítása. Képes módosítani az Ethernet keretek forrás- és célcímét, az IP csomagok, valamint TCP csomagok fejrészét. Módosítás után újraszámolja a TCP ellenőrzőösszeget.

4. Alkalmazási példa – HTTPS forgalom elterelése

A MINT rendszer alkalmazására bemutatunk egy egyszerű, de tanulságos példát. Tekintsük a webszerverek és böngészők közötti biztonságos kommunikáció protokollját, mely nem más, mint a HTTP az SSL/TLS [7,6] protokoll fölött. Az SSL/TLS protokoll feladata a kommunikáló felek authentikációja és a kommunikáció titkosítása.

A HTTPS kommunikáció az SSL/TLS kézfogással (handshake) kezdődik. A kliens – esetünkben a böngésző – elküldi a ClientHello üzenetet a szervernek. A szerver válaszul ServerHello üzenet mellett elküldi a saját tanusítványát, majd a ServerHelloDone-al zárja a kommunikációt. A kliens, miután ellenőrizte a tanusítványt, előállítja a titkosításhoz szükséges adatokat, majd ennek publikus részét átküldi a szervernek a ClientKeyExchange üzenetben. Ezen kívül ChangeCipherSpec üzenettel jelzi, hogy ő már készen áll a titkosításra. A kommunikációt a kliens zárja a Finished üzenettel. A szerver miután kinyerte a közös, osztott titkot a ClientKeyExchange segítségével, Finished üzenettel válaszol. A handshake után a kliens és a szerver titkosítottan kommunikál. Ez történik például egy internetes banki belépésnél is, ahol a felhasználói név és jelszó már titkosítva kerül átvitelre.

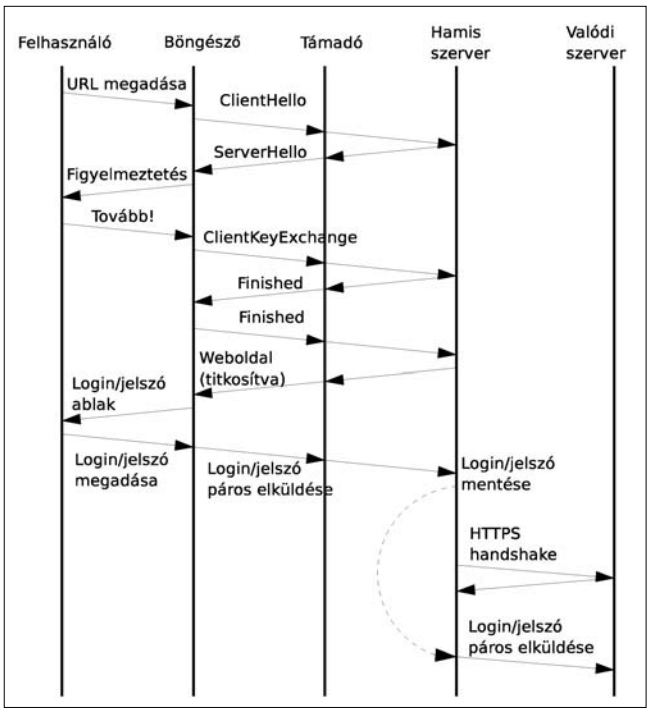


2. ábra HTTPS forgalom elterelése – Teszt topológia

Tesztünk során egy felhasználó a böngészője segítségével egy webszerverrel veszi fel a kapcsolatot. Az alkalmazott HTTPS protokoll authentikációs eljárása ellenére sikerült megtévesztenünk a felhasználót. HTTPS-en való csatlakozáskor a böngésző ellenőrzi az SSL handshake során kapott szerver-tanusítványt, hogy megbizonyosodjon a szerver valódi kiléte felől. Amennyiben az ellenőrzés sikertelen, figyelmezteti a felhasználót, majd megkérdezi, hogy ennek ellenére akarja-e folytatni a kommunikációt. A felhasználók sajnálatos módon figyelmen kívül hagyják ezeket a figyelmeztetéseket (sokszor annak elolvasása nélkül), s így hamis tanusítványokat is könnyen elfogadnak.

A tesztünk során felállítottunk egy hamis webszervert, majd a MINT szoftver segítségével eltereltük felé a HTTPS forgalmat *(2. ábra)*. Ezek alapján láthatjuk, hogy egy támadó, akinek sikerült beékelődnie a felhasználó és a szerver közé, képes a szervert megszemélyesíteni. Ehhez egyszerűen el kell terelnie a felhasználótól a valódi szerver irányába folyó forgalmat egy általa üzemeltetett hamis szerverre. Amennyiben a hamis szerveren a valódinak megfelelő vagy hasonló tartalom van, a támadó nagy valószínűséggel meg tudja téveszteni a felhasználót. Ezután a megtévesztett felhasználó jóhiszeműen megadhat bizalmas információkat, például bankkártyaszámát, jelszavait, melyekkel később a támadó visszaélhet. Egy ilyen támadás üzenetváltásait szemlélteti a *3. ábra (lásd a következő oldalon)*.

Kimutattuk tehát, hogy a TLS protokoll biztonsági szolgáltatásai ellenére a felhasználó gondatlansága miatt beékelődéses támadással célt érhetnek a támadók.



3. ábra Megszemélyesítéses támadás – Üzenetváltások

5. Összegzés

Cikkünkben bemutattunk egy olyan biztonsági vizsgálati módszert, valamint az ezen módszert alkalmazó eszközt, melynek segítségével a tesztelendő rendszerről eldönthetjük, hogy közbeékelődéses támadások esetén is megfelel-e a biztonsági elvárásoknak. Ez az eljárás ezen kívül protokollok biztonsági hibáinak felfedésére is alkalmas.

Protokollok tesztelésénél általában is nagy segítséget nyújthat az általunk megvalósított keretrendszer, mivel ezzel olyan helyzeteket tudunk teremteni, amelyek felszínre hozhatják a protokoll vagy annak megvalósításának általános hibáit. A keretrendszer használatával a fejlesztő a tesztelés szempontjából fontos részletekre koncentrálhat, anélkül, hogy az alacsony szintű csomagkezeléssel vagy állapotgép-reprezentáció megvalósításával kellene foglalkoznia.

A beékelődéses módszer és a keretrendszer gyakorlati alkalmazhatóságát szemléltette a fent bemutatott forgalomelterelési példa is.

Irodalom

[1] Nessus – a remote network security scanner, http://www.nessus.org/
[2] Nmap – Network Security Scanner, http://www.nmap.org/
[3] The Network Simulator – ns-2, http://www.isi.edu/nsnam/ns/
[4] TTCN3 – Methods for Testing and Specification (MTS) The Testing and Test Control Notation version 3, ETSI Document Nr.: ES 201 873-1.
[5] IEEE Standard 802.11, part 11., 1997. Wireless LAN Medium Access Control and Physical Layer Specification.
[6] T. Dierks and C. Allen: The TLS Protocol, 1999. FC 2246, Proposed Standard.
[7] Kocher Frier, Karlton: The SSL 3.0 Protocol, 1996. Internet Draft, Work in Progress.
[8] The Tcpdump Group: `libpcap`: Packet capture library http://www.tcpdump.org/
[9] Gevin Lowe: Casper: A compiler for the analysis of security protocols, Journal of Computer Security, 6:53–84, 1998.
[10] Catherine Meadows: The NRL Protocol Analyzer: An overview, Journal of Logic Programming, 26:113–131, 2. 1996.
[11] A. W. Roscoe: The Theory and Practice of Concurrency, Prentice Hall, 1998.
[12] Mike D. Schiffman: `libnet`: A C library for portable packet creation and injection, http://www.packetfactory.net/libnet
[13] Abraham vd Merwe: `libconfig`: A C library for parsing hierarchical configuration files, http://oasis.frogfoot.net/