Összetett protokollelemzés Zorp (GPL) segítségével

A web fejlődésének, az üzleti szolgáltatások (e-business) és az e-bank terjedésének előfeltétele volt egy olyan, biztonságos protokoll fejlesztése, mely garantálja a kliens és a szerver közötti kommunikáció bizalmasságát és sértetlenségét. Ennek segítségével a kliens és a szerver maradéktalanul megbizonyosodhat arról, hogy a másik fél valóban az, amit állít magáról, illetve a kommunikáló felek közötti adatáramlást ártó szándékkal senki nem módosította.

Ez a protokoll a *Netscape* által fejlesztett ***SSLv1 (Secure Socket Layer)***, melynek 3-as verzióját (***SSLv3***) 1996-ban fejezték be. A kapcsolatok kriptografikus védelmét az alkalmazás rétegben (***TCP Layer4*** vagy ***ISO/OSI Layer 7***) valósítja meg, úgy, hogy az eredeti protokollt magába foglalja, köré egy védőburkot (biztonságos csatornát) képez.
Az ilyen protokollok átengedése a hálózati határpontokon egy vállalat esetében komoly kockázatokat rejt magában, melyet többféle módon lehet megoldani.

- Az első megoldás az ilyen protokollok teljes tiltása, mely nem célravezető, mivel az ügyintézés lassulása komoly anyagi veszteségeket is jelenthet.
- A második lehetőség az ilyen protokollok teljes vagy részleges engedése, úgy hogy a forgalmat csak alacsonyabb szinten (csomagszűrők) korlátozzuk. Ennek mellékhatása, hogy a kémprogramok, trójaik illetve a vállalat vezetése által üldözött alkalmazások ezeket a lyukakat használják kommunikációra, mivel feltételezik, hogy a vállalati tűzfalak ezeket a protokollokat nem szűrik. Ilyen lehet akár egy *ICQ* vagy *Peer-to-Peer* fájlcsere kliens.
- A harmadik (és egyben a legjobb) megoldást pedig az SSL protokoll valamit a benne alkalmazott úgynevezett beágyazott protokoll ellenőrzése jelenti. Ebben az esetben a tűzfal a kliens és a szerver közé ékelődve a forgalmat ellenőrzi és valóban csak a megengedett forgalmakat engedi át.

Ehhez nyújt segítséget a ***BalaBit IT Security Kft.*** által fejlesztett ***Zorp GPL*** és ***Professional*** termékek, melyről korábbi számainkban már több cikk jelent meg ***Scheidler Balázs*** a program szerzőjének (***Linuxvilág #14***), valamint ***Michael D. Bauernek*** (***Linuxvilág #40*** és ***#41***) tollából. Jelen cikk az ***SSL*** proxyzásról szól ***Zorp GPL*** és ***Professional*** segítségével.

Az SSL protokollról: rejtjelezés dióhéjban

A rejtjelezés célja, hogy meggyőződjünk adataink bizalmasságáról (vagyis hogy az információhoz csak azok a szereplők férhetnek hozzá, akiknek ehhez jogosultsága van) és sértetlenségéről (tehát, hogy az információtartalom nem sérült vagy módosult a útközben).
A bizalmasság védelmére a digitális világban a rejtjelezést használják, melynek két alapvető módszere létezik.

A szimmetrikus titkosítás esetén a rejtjelezéshez és a visszafejtéshez ugyanazt a kulcsot használják. Ez tulajdonképpen a történelmi módszerek elektronikus változataiban ugyanúgy működik. A módszer előnye gyorsaságában rejlik, hátránya viszont, hogy meg kell oldani a közös kulcs kitalálásának és cseréjének problémáját.
Az asszimetrikus (nyilvános) kulcsú rejtjelezés pontosan ezt a problémát oldja meg. Minden résztvevő fél két kulcsot generál, úgy, hogy az egyik ismeretében ne lehessen kitalálni a másikat, mégis amit az egyikkel rejtjelezünk, azt kizárólag a másik kulcs birtokában lehet visszafejteni. Az egyik kulcsot kijelöljük publikus kulcsnak és a világ számára elérhetővé tesszük. Amennyiben valaki rejtjelezett üzenetet kíván küldeni nekünk, az megteheti a publikus kulcs segítségével. Az így előállított üzenetet csak a privát kulcs tulajdonosa képes megfejteni.
Kivonatkészítés (***hashing*** algoritmus):
A technológia másik fontos algoritmusa a kivonat képzés, ami annyit jelent, hogy egy bármekkora dokumentumból fix hosszúságú kivonatot képezünk, úgy, hogy:

1. az csak az adott dokumentumra legyen jellemző
2. egyetlen bit változás az eredeti dokumentumba változtassa meg a teljes kivonatot

3. a kivonatból ne lehessen előállítani az eredeti dokumentumot

Autentikáció: az a folyamat melynek során meggyőződünk a másik fél személyazonosságáról.

Digitális aláírás: A digitális aláírásnál a dokumentum kivonatát képezzük, majd azt rejtjelezzük a privát kulcsunkkal. Az aláírást ellenőrző fél pedig megoldja a rejtjelezett üzenetet, leképzi a megkapott dokumentum kivonatát, majd összehasonlítja a két kivonatot. Ha az megegyezik, a dokumentum pontosan megegyezik a küldő által aláírt dokumentummal.

Néhány szó a tanúsítványokról

A tanúsítvány nem más, mint egy megbízhatónak ítélt harmadik fél által aláírt bizonyítvány (***ASN.1*** struktúra) és a tanúsítvány tulajdonosának publikus kulcsa. Ez a harmadik fél a hitelesítés szolgáltató – vagy ***CA (Certificate Authority)***.
Amikor tanúsítványt készítünk akkor a publikus kulcsunk felhasználásával kérvényt készítünk (***CSR – certificate signing request***), mely tartalmazza a ránk vonatkozó adatokat.
Ezt a kérvényt gondosan csomagolva a ***CA*** megfelelő intézményébe (***RA – registration authority***) visszük, ahol megvizsgálják a személyi azonosságunkat. Amennyiben mindent rendben találnak, a ***CA*** saját privát kulcsával aláírja a kérvényünket.
Az így előállt tanúsítványt bárki, bármikor ellenőrizheti a ***CA*** tanúsítványával, vagyis a ***CA*** publikus kulcsának segítségével.
A ***CA*** vállalja, hogy rendszeresen közzéteszi a visszavonási listát (azon tanúsítványok listáját, amit valamilyen okból visszavontak, tehát többé már nem érvényesek).
Mikor érvényes egy tanúsítvány?
Egy tanúsítvány minimálisan akkor érvényes, ha:

1. Általunk megbízhatónak ítélt ***CA*** bocsájtotta ki. Egy tanúsítványt több ***CA*** (úgynevezett ***subCA***) is aláírhat. Ilyenkor a tanúsítvány láncon addig kell végigmenni, míg nem találunk olyan aláírót, akiben megbízunk.
2. A tanúsítvány érvényessége nem járt le (vagyis a dátum az ellenőrzés pillanatában a tanúsítványban feltüntetett *„Not Before”* és *„Not After”* dátumok között van)
3. A tanúsítványt nem vonták vissza (nincs visszavonási listán – ***CRL***)
4. A cél tartomány és a tanúsítványban feltüntetett cél megegyező.
5. A tanúsítványt arra használják, amire a ***CA*** kiadta.

Persze saját feltételeket is támaszthatunk, amennyiben szükség van rá.
Az ***SSL*** protokoll a működése a következőkből áll:

1. A kliens fordul a szerverhez egy *„ClientHello”* üzenettel, melyben felsorolja a protokollokat a tömörítési módszereket valamint a legmagasabb protokoll verziókat továbbá egy véletlenszámot, amit később foknak felhasználni.
2. A kliens fogadja a szervertől a *„ServerHello”* üzenetet, amiben a szerver kiválasztja a kapcsolódási paramétereket azok közül, amit a kliens felajánlott.

3. Amikor a kapcsolódási paraméterek ismertek a kliens és a szerver tanúsítványt cserél. Ezek a tanúsítványok jelenleg csak *X.509*-es formátumban lehetségesek, de létezik piszkozat (draft) állapotú szabvány az *OpenPGP*-re is. A tanúsítvány segítségével a kliens meggyőződik a szerver személyazonosságáról (autentikál), vagyis eldönti tényleg ahhoz a szerverhez csatlakozott, aminek állítja magát.
4. A szerver kérhet tanúsítványt a klienstől is, így a szerver is ellenőrizheti a kliens személyét, ezt kétirányú vagy kölcsönös autentikációnak (mutual) hívják.
5. A kliens és a szerver megegyezik egy közös úgynevezett mester titokban (master secret). Minden további kulcs adat ebből a titokból a szerver és a kliens által előállított véletlen számokból származik, amit egy gondosan megtervezett *„Pseudo Random Function”* függvénnyel készítenek.

Amennyiben az öt lépés lezajlott, rendelkezésünkre áll egy rejtjelezett csatorna, melyben megkezdődhet a tényleges kommunikáció.

Az SSL proxyzás elmélete

Amennyiben *SSL* protokollba ágyazott kommunikációt szeretnénk tűzfalon elemezni, az első megoldandó probléma az *SSL* kapcsolat végződtetése. Ebben az esetben a tűzfal mutat tanusítványt a kliensnek, a szerver tanúsítványát pedig maga a tűzfal ellenőrzi. Az így lecsupaszított kapcsolatot pedig továbbadja egy protokoll értelmező proxynak. Ehhez a következő lépésekre van szükségünk:

1. Valamilyen saját *CA* felállítása
2. Tanúsítvány készítése a tűzfal részére, melyet saját *CA*-nk ír alá
3. A saját *CA*-nk tanúsítványát telepíteni kell a kliensekre
4. Lehetőség szerint minél több minősített *CA* tanúsítványának beszerzése és telepítése a tűzfalra. Ez könnyen beszerezhető a *Debian GNU/Linux 3.1 (sarge) ca-certificates* csomagból pontosan ezt a problémát oldja meg.
5. Proxy beállítása

A proxy stackelés

A *proxy stackelés* lényege, hogy egy egy *Zorp proxy* képes a protokoll adattartalmát átadni egy másik proxynak. Ez az adattartalom lehet akár egy másik alkalmazás szintű protokoll is. Az *SSL proxy* esetében is ez történik. Amikor a beágyazott proxy végzett a munkájával, az eredményt egyszerűen visszaadja az *SSL proxynak*, mely folytatja tovább a kommunikációt.

Az SSL proxyzásról pro és kontra

Előnyei:

- összetett protokollok értelmezése
- a szerver tanúsítványát a proxy ellenőrzi nem a felhasználó, akik gyakran nem rendelkeznek megfelelő biztonsági tudatossággal
- szabályozható az elfogadott *CA*-k listája

Hátrányai:

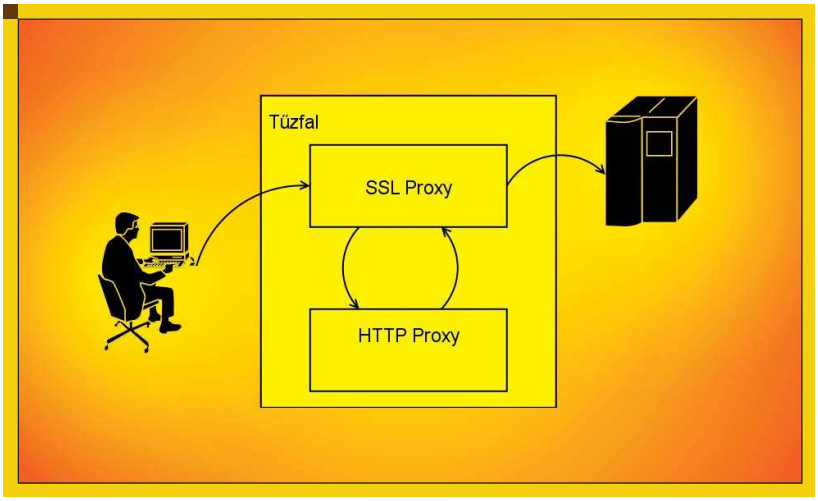
- Személyiségi jogi (privacy) kérdések és azok kiküszöbölése. A rejtjelezett protokollokba való betekintés komoly kérdéseket vet fel, melyeket a munkaszerződésekben valamint az informatikai biztonsági szabályzatban (*IBSz*) rögzíteni kell, illetve tudatosítani kell a felhasználókkal ennek a lehetőségét
- a kliens nem látja az eredeti tanúsítványt, mely csak akkor jelenthet problémát, ha (hibásan) úgy konfiguráljuk proxynkat, hogy mindenféle tanúsítványt elfogadjon.
- helyesen beállított konfiguráció esetén csak megbízható *CA*-k által kibocsátott tanúsítványokat fogadunk el. Ennek hátránya, hogy az saját tanúsítvánnyal rendelkező szerverek elérhetetlenek lesznek.

HTTPS Proxy Zorp GPL-el

Első példánkban egy *SSL proxyt* fogunk beállítani, mely *HTTP proxyt* indít a beágyazott protokoll ellenőrzésére, így kikényszerítve, hogy tényleg csak *HTTPS* protokoll menjen át az ott megengedett (*TCP/443*-as) porton. Ennek módja a következő.

CA készítése OpenSSL-el

Hozzunk létre egy könyvtárat a *CA* számára, majd a */etc/ssl/openssl.cnf*-ban a `stateOrProvinceName_default` paraméter értékéből töröljük a `SomeState` értéket! Ezek után futtassuk a `/usr/lib/ssl/misc/CA.pl -newca` parancsot. Az *Enter* megnyomására a szkript legenerálja a *CA* tanúsítványához szükséges privát-publikus kulcspárt, amihez jelszó is rendelhető. A további kérdések a CA tanúsítványának egyedi nevében (*Distinguished Name*) megjelenő információkra vonatkoznak – a számunkra nem releváns mezőket (például tartomány neve) nyugodtan hagyjuk üresen. (Az előforduló mező: ország kétbetűs kódja (`Country: HU`), állam vagy tartomány neve (`State or Province Name: -`), város (`Locality: Budapest`), szervezet neve (`Organization Name: Proba Kft.`), szervezeti egység (`Organizational Unit Name: IT Biztonsagi Csoport`), a tanúsítvány neve/tulajdonosa (`Common Name: Proba CA`), a tulajdonos e-mail címe (`Email Address: info@proba.hu`).



1. kód

```
Import Zorp.Pssl *

class HttpsProxy(PsslProxy)
       def config(self):
              PsslProxy.config(self)

              self.server_need_ssl = TRUE

              self.self.server_verify_type =
              ➥PSSL_VERIFY_REQUIRED_UNTRUSTED
              self.server_verify_type =
              ➥PSS_VERIFY_REQUIRED_TRUSTED
              self.verify_depth = 5
              self.server_ca_directory = '/etc/zorp/ca.d'

              self.client_need_ssl = TRUE
              self.client_verify_tpye = PSS_VERIFY_NONE
              self.client_key_file = '/etc/zorp/https/server.key'
              self.client_cert_file = '/etc/zorp/https/server.crt'

              self.stack_proxy = HttpProxy
```

Kulcspár és tanúsítvány igénylés (CSR) készítése

A tűzfal számára szükségünk lesz egy tanúsítványra, amihez előbb egy kulcspárt és egy tanúsítvány igénylést kell elkészítenünk. Ezt az előző lépéshez hasonlóan szintén *OpenSSL*-el tehetjük meg. Adjuk ki az `openssl req -newkey rsa:1024 -new -out server.csr -keyout server.key` parancsot, majd adjuk meg a tűzfal tanúsítványába szánt információkat. Érdemes a tulajdonos nevét (*Common Name*) úgy megadni, hogy abból kiderüljön, hogy ez a tűzfal tanúsítványa. (A *CSR* és a kulcspár a `server.csr` és a `server.key` fájlokban kerülnek tárolásra.)

Tanúsítvány hitelesítése és elhelyezése a tűzfalon

Az előző lépésben elkészített tanúsítvány igénylést már csak hitelesíteni kell a folyamat elején létrehozott *CA*-val. Ezt a `openssl ca -in server.csr -out server.crt -policy policy_anything` paranccsal tehetjük meg, a feltett kérdésre (`Sign the certficate?`) adott igen (y) válasszal. Ha ezzel megvagyunk a `server.key` és a `server.crt` fájlokat másoljuk a tűzfalra a */etc/zorp/https/* könyvtárba, majd telepítsük a *ca-certificates* csomagot. Készítsünk linket az összes tanúsítványra a */etc/zorp/ca-d/* könyvtárba (`find /usr/share/ca-certificates/ -type f -exec ln -s {} \;`), és minden *ca.cert*-re (`for i in $(find /usr/share/ca-certificates/ -type f); do echo ln -s $i $(openssl x509 -noout -hash -in $i).0; done`).

Zorp Proxy konfigurálása

Hozzuk létre a saját *SSL* proxykat. A *Zorp* konfigurálására a *Python* nyelvet használjuk, így bármilyen proxy osztályt létrehozhatunk úgy, hogy származtatjuk egy már meglévő osztályból és csak a különbségeket állítjuk be, a többi paraméter öröklődik. Ennek megfelelően egy készítsük el első *HTTPS* proxynkat (*1. kód*).
Az *SSL* proxy megfelelő beállításához meg kell adnunk az *SSL* kapcsolat paramétereit. A paraméter elején szereplő „client” és „server” előtag határozza meg, hogy a kapcsolat melyik oldalára vonatkozzon a beállítás. Az alábbiakban összefoglaljuk a legfontosabb paramétereket és funkcióikat.
`client_need_ssl/server_need_ssl`: jelenti, hogy a kliens vagy a szerver oldalon egyáltalán kell-e *SSL* kapcsolat, így akár féloldalas proxyt is készíthetünk. Például a kliens nem tud *SSL* kapcsolatot felépíteni, de a szerver felé mindenképpen rejtjelezni akarunk.

`client_key_file/server_key_file`: ha valamelyik oldalon autentikálni szeretnénk, itt kell megadni a privát kulcs elérhetőségét.

`client_ca_directory/server_ca_directory`: a kapcsolat tanúsítványait az itt található *CA*-k publikus kulcsaival ellenőrizzük, vagyis ha van olyan *CA* tanúsítvány a könyvtárban, aminek a privát párjával a tanúsítványt kiállították, akkor az a tanúsítvány megbízható.

`client_crl_directory/server_crl_directory`: a *CA*-k visszavonási listáit tartalmazó könyvtár.

`client_verify_type/server_verify_type`: a tanúsítvány ellenőrzésének mélysége, mely a következő lehet:

- `SSL_VERIFY_NONE` – a proxy minden tanúsítványt elfogad, ellenőrzés nem végez.
- `SSL_VERIFY_OPTIONAL` – a másik fél küldhet tanúsítványt.
- `SSL_VERIFY_REQUIRED_UNTRUSTED` – a másik félnek kell tanúsítványt küldeni, de bármilyen CA adhatja.
- `SSL_VERIFY_REQUIRED_TRUSTED` – a másik félnek megbízható tanúsítványt kell küldenie.

`client_verify_depth/server_verify_depth`: ha több *CA* hitelesítette a tanúsítványt, akkor az így kapott láncon hány *CA* mélységben ellenőrizze a proxy tanúsítványokat. A tanúsítványt az első megbízhatónak ítélt *CA*-nál fogadja el megbízhatónak.

`stack_proxy`: ez az egyik legfontosabb beállítás, hiszen ezért használjuk. Az *SSL* proxy az itt megadott proxyt fogja stackelni, indítani. Alapértelmezésben *PlugProxyt* indít.

Összefoglalás

Az *SSL* napjaink egyik legfontosabb protokollja, melynek használata és a határpontokon való átengedése komoly biztonsági kockázatokat jelent. Ennek kiküszöbölésére alkalmas az *SSL* kapcsolatok végződtetésére és a beágyazott protokollok elemzésére is alkalmas *Zorp GPL* proxy.

Höltzl Péter
(holtzl.peter@balabit.hu)


Linuxvilág
www.linuxvilag.hu