Bezpieczeństwo Usług Sieciowych
OpenSSL

dr inż. Tomasz Surmacz

26 listopada 2014

SSL – Secure Socket Layer

- szyfruje dane
- pozwala zabezpieczyć się przed atakami typu Man-in-the-middle
- korzysta z certyfikatów w celu sprawdzenia tożsamości obu stron

Certyfikaty

- weryfikacja tożsamości obu stron
- zabezpieczenie przed spoofingiem (podszywaniem się)
- automatyczne pobieranie certyfikatów podczas nawiązywania połączenia
- system podpisów – "Certificate Authorities"

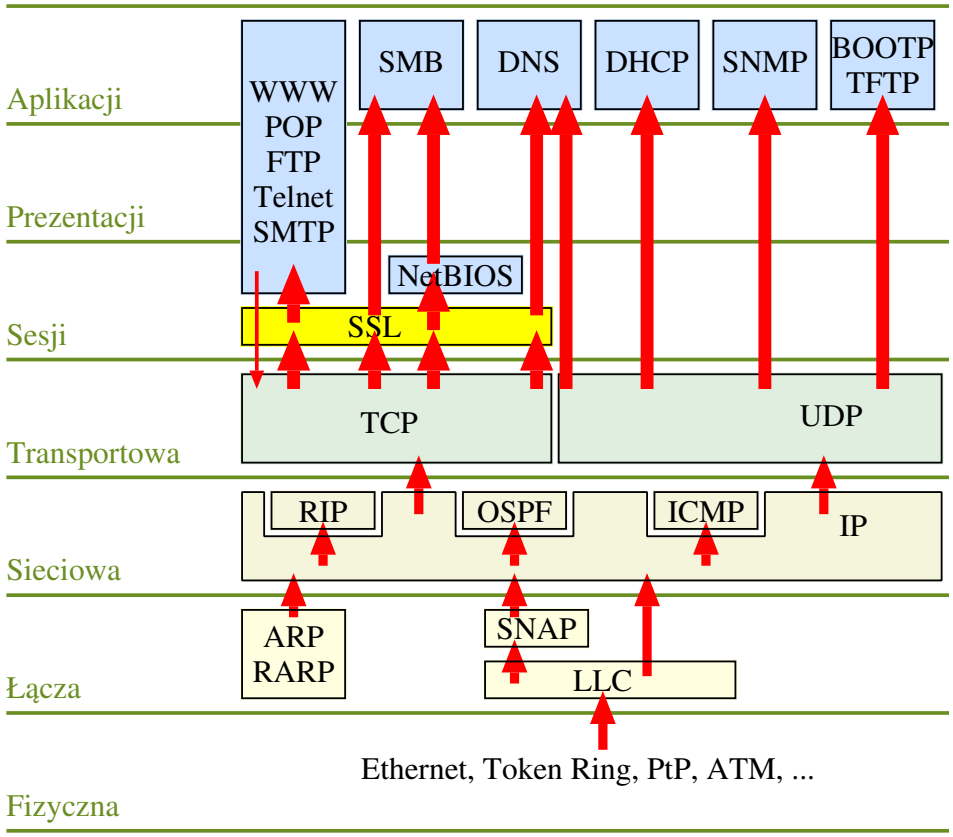
Korzystanie z SSL

- W sposób jawny:
  - przez użycie protokołu takiego jak `HTTPS`
  - przez korzystanie z bibliotek SSL poprzez odpowiednie funkcje
- Niejawnie:
  - z użyciem TLSWRAP

SSL – Secure Socket Layer

Dwa podstawowe cele korzystania z SSL:

- dwustronne uwierzytelnienie i zapewnienie bezpieczeństwa
- utworzenie bezpiecznego kanału komunikacyjnego, który mogą wykorzystać wyższe warstwy protokołu



SSL/TLS – krótka historia rozwoju

- We wczesnych latach '90 IETF ustanowiła grupę roboczą Web Transaction Security (WTS)
- Spowodowało to powstanie protokołu warstwy aplikacji nazwanego Secure Hypertext Transfer Protocol (S-HTTP), opisanego w eksperymentalnych RFCs 2659 i 2660
- Nieezależnie od tego, Netscape Communications zaproponowało i opatentowało protokół warstwy transportowej nazwany Secure Sockets Layer (SSL)
- SSL jest umieszczony nad połączeniową warstwą transportową zapewniającą wolny od błędów przesył danych (TCP lub podobna)
- W 1994 SSL 2.0 został zaimplementowany przez Netscape Communications w przeglądarce Netscape Navigator
- Microsoft zaproponował podobną koncepcję nazwaną Private Communication Technology (PCT) i poprawioną/rozbudowaną wersję Secure Transport Layer Protocol (STLP) w roku 1995
- W 1996 powstała specyfikacja SSL 3.0
- W 1999 IETF założył grupę roboczą Transport Layer Security (TLS) (RFC 2246), aby rozwiązać spór między Netscape Communications (SSL) i Microsoftem (PCT/STLP)
- Dalszy rozwój skupił się na TLS, owocując standardem TLS 1.1 (RFC 4346) w kwietniu 2006, jak również protokołem Datagram TLS 1.0 (DTLS) (RFC 4347)
- W kwietniu 2008 została ogłoszona specyfikacja TLS 1.2 – RFC 5246

Ograniczenia/wady SSL

- SSL/TLS mają zapewniać bezpieczną komunikację pomiędzy dwoma punktami
- SSL/TLS nie są kompletną platformą bezpieczeństwa
- nie rozwiązują problemów takich jak:
  - SQL injection
  - Cross-site scripting (XSS)
  - Cross-site request forgery (CSRF)
- Korzystanie z SSL/TLS może powodować problemy z implementacją polityki bezpieczeństwa (content screening) i może wymagać instalacji proxy SSL/TLS

Ataki MITM

- W teorii – zabezpieczenie przed atakami „Man in the Middle”
- W praktyce – problemy z zarządzaniem certyfikatami i strukturą ich uwierzytelniania

Błędy w kodzie SSL

- Błędy w interpretowaniu kodów błędu weryfikacji certyfikatów
- Ostatnie odkrycia (8.04.2014) – Heartbleed attack (OpenSSL 1.0.1 do 1.0.1e włącznie)

Definicje i standardy

- ANSI X.509 – certyfikaty
- RFC 2246 – TLS
- RFC 2487 – SMTP over TLS
- RFC 2818 – HTTPS (`https://...`, port 443)
- RFC 4346 – TLS 1.1
- RFC 5246 – TLS 1.2

Wykorzystanie SSL

- OpenSSL (`http://www.openssl.org/`)
- Apache SSL (`http://www.apache-ssl.org/`)
- mod_ssl (`http://www.modssl.org/`)

SSL – Konfiguracja

- Sprawdzenie wersji SSL i dostępnych metod szyfrowania

```
linux% openssl version -a
OpenSSL 0.9.8o 01 Jun 2010
platform: debian-amd64
options:  bn(64,64) md2(int) rc4(ptr,char) des(idx,cisc,16,int) blowfish(ptr2)
...
linux% openssl ciphers -v
DHE-RSA-AES256-SHA      SSLv3 Kx=DH       Au=RSA  Enc=AES(256)  Mac=SHA1
DHE-DSS-AES256-SHA      SSLv3 Kx=DH       Au=DSS  Enc=AES(256)  Mac=SHA1
AES256-SHA              SSLv3 Kx=RSA      Au=RSA  Enc=AES(256)  Mac=SHA1
DES-CBC3-SHA            SSLv3 Kx=RSA      Au=RSA  Enc=3DES(168) Mac=SHA1
DES-CBC3-MD5            SSLv2 Kx=RSA      Au=RSA  Enc=3DES(168) Mac=MD5
AES128-SHA              SSLv3 Kx=RSA      Au=RSA  Enc=AES(128)  Mac=SHA1
...
```

- Sprawdzenie gdzie SSL szuka swoich plików: `openssl version -d` (zwykle */usr/lib/ssl*)
  - *openssl.conf* – konfiguracja CA, certyfikatów, wartości domyślne
  - *cert.pem* – zbiór uznawanych certyfikatów (Verisign, Thawte)
  - podkatalog *certs* – pojedyncze pliki *.pem* z certyfikatami CA
  - symlinki w katalogu *certs* do przechowywanych certyfikatów. Nazwy – hash certyfikatu

SSL – Certyfikaty

- Certyfikat jest odpowiednikiem pary kluczy w kryptografii z kluczem publicznym
- Certyfikat publiczny oprócz publicznej części klucza zawiera informacje o jego właścicielu:
  - Certificate Name – identyfikator certyfikatu
  - Expiry Date – Data wygaśnięcia ważności (z reguły – rok od utworzenia certyfikatu)
  - Common Name – nazwa intytucji/osoby (widoczna jako "issued to/wystawiony dla")
  - Email Address – adres email osoby kontaktowej (administrator związany z serwerem/domeną)
  - Organisation – nazwa instytucji, dla której wystawiany jest certyfikat
  - Department – dział/oddział/jednostka organizacyjna
  - City / Town – miasto związane z instytucją/oddziałem
  - State / Province – województwo itp.
  - Country – kraj
  - Private Key Length – długość klucza prywatnego: obecnie zalecane: 2048 bitów, minimum 1024
  - A także: numer wersji, sumer seryjny, algorytm użyty do podpisania, instytucję podpisującą (issuer), czas ważności, klucz publiczny
- Część publiczna certyfikatu musi być podpisana przez CA (Certificate Authority) lub samodzielnie
- Część prywatna – z definicji nie powinna być nigdzie publikowana i pozostać tajna

SSL – Certyfikaty

- Utworzenie certyfikatu

```
openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout mycert.pem -out mycert.pem
```

  - Większość pytań – oczywista (Country Name, State, City, itd.)
  - „Common Name" – nazwa DNS serwera lub jego CNAME
  - Można tworzenie certyfikatu zautomatyzować dzięki opcji `-subj`:

```
openssl req  -x509 -nodes -days 365  -newkey rsa:1024 -keyout mycert.pem -out mycert.pem \
    -subj '/C=PL/ST=Dolnoslaskie/L=Wroclaw/CN=www.jakasfirma.pl'
```

  - Minimalistyczny certyfikat musi zawierać pole CN, pozostałe są opcjonalne.

- Automatyzacja za pomocą `-subj`
  - `/C=`*PL* – kraj
  - `/ST=`*Dolnoslaskie* – stan, województwo, region, itp,
  - `/L=`*Wroclaw* – „locality": miasto
  - `/O=`*Politechnika Wroclawska* – organizacja
  - `/OU=`*Wydzial Elektroniki* – „organizational unit" – jednostka organizacyjna, oddział, itp.
  - `/CN=`***www.pwr.wroc.pl*** – „common name" – DNS name
  - `/emailAddress=`*Jakis.Adres@pwr.wroc.pl* – adres email osoby kontaktowej

- Sprawdzenie certyfikatu:

```
linux% openssl verify mycert.pem
mycert.pem: /C=PL/ST=Doln/L=Wroclaw/O=PWr/OU=Elektronika/CN=www.pwr.wroc.pl/emailAddress=Jakis.Adres@pwr.wroc.pl
error 18 at 0 depth lookup:self signed certificate
OK
```

SSL – podpisywanie certyfikatu

1. Tworzenie certyfikatu to wygenerowanie „certificate requests” (dokładniej: „certificate signing request”)

```
openssl req -new -key privkey.pem -out cert.csr
```

✪ Sprawdzenie treści tego żądania:

```
openssl req -in cert.csr -noout -text
```

2. Tak utworzone „żądanie podpisu” (plik *cert.csr*) może być wysłane do CA lub można je samodzielnie podpisać, jeśli dysponujemy własnym CA

```
# Własne CA w katalogu demoCA (demoCA/cacert.pem, demoCA/private/cakey.pem)
openssl ca -policy policy_anything -out newcert2.pem -infiles newreq.pem
```

1. Wygenerowanie samodzielnie podpisanego certyfikatu:

```
openssl genrsa -des3 -out privkey.pem 2048
openssl req -new -x509 -key privkey.pem -out cacert.pem -days 1095
```

   - Pierwsza linia generuje klucz. Druga – certyfikat.
   - Opcja `-des3` powoduje zabezpieczenie klucza hasłem (bez niej – klucz bez hasła)
   - Jeśli certyfikat/klucz zabezpieczony hasłem ma być używany przez serwer, będzie konieczne każorazowe odblokowanie go po restarcie aplikacji.

✪ Sprawdzenie informacji:

```
openssl x509 -text -in cacert.pem
```

- Wybiórcze informacje – dodając opcje `-issuer`, `-subject`, `-dates`, `-hash` czy `-fingerprint`

SSL – Własne CA

- Do utworzenia/zarządzania własnym CA najlepiej posłużyć się skryptem `CA.pl` lub `CA.sh` (standardowo w katalogu */usr/lib/ssl/misc/*)
- Domyślna konfiguracja z */usr/lib/ssl/openssl.cnf* wskazuje na katalog *./demoCA* – certyfikaty CA, baza podpisanych certyfikatów i inne pliki
- Inną konfigurację można wskazać ustawiając zmienną środowiskową `OPENSSL_CONF`

Podstawowe użycie:

- `CA -newca` – utworzenie nowego CA
- `CA -newreq [-nodes]` – nowe żądanie (nie podpisany certyfikat)
- `CA -sign` – podpisanie certyfikatu
  Podpisuje żądanie z pliku *newreq.pem* zapisując wynik w *newcert.pem*

Nowo podpisany certyfikat trafia także do bazy danych CA (katalog *demoCA/newcerts/*)

- Komunikat „failed to update database” oznacza, że podpisywany certyfikat już istnieje w bazie (Usunięcie starego: `openssl ca -revoke xyz.crt`)
- Domyślne wartości (np. `countryName_default=AU`) używane przez CA, a także przy generowaniu certyfikatów, można zmienić modyfikując plik */usr/lib/ssl/openssl.cnf*

Certyfikaty – przykłady

- Utworzenie cert-request, podpisanie go przez CA:

```
% openssl req -new -key privkey.pem -out new4.csr
% openssl ca -policy policy_anything -out newcert4.pem -infiles new4.csr

Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:    a7:aa:d5:ee:0e:ce:77:d0
        Validity          Not Before: Nov  1 20:36:03 2011 GMT         Not After : Oct 31 20:36:03 2012 GMT
        Subject:
            countryName               = PL               organizationName          = ccc
            stateOrProvinceName       = aaa              organizationalUnitName    = ddd
            localityName              = bbb              commonName                = eee
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                CA:B1:FD:89:DD:04:5F:A0:E3:9E:C7:50:3A:01:88:1F:8F:9E:8F:E9
            X509v3 Authority Key Identifier:
                keyid:BC:7D:A2:9B:2D:FA:F9:51:6E:73:82:2B:30:02:EC:E3:33:2A:D7:40
Certificate is to be certified until Oct 31 20:36:03 2012 GMT (365 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

- Sprawdzenie podpisanego certyfikatu:

```
demoCA% openssl x509 -text -in newcerts/A7AAD5EE0ECE77D0.pem
Certificate:
  Data: Version: 3 (0x2)
        Serial Number:    a7:aa:d5:ee:0e:ce:77:d0
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=PL, ST=Dolnoslaskie, O=Demo CA org, CN=laptok CA/emailAddress=laptok@laptok
        Validity:     Not Before: Nov  1 20:36:03 2011 GMT, Not After : Oct 31 20:36:03 2012 GMT
        Subject: C=PL, ST=aaa, L=bbb, O=ccc, OU=ddd, CN=eee
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:d4:be:ee:2a:ef:df:e7:4a:aa:9b:0d:65:7b:f7:
        ...
                    2d:28:ae:ad:7e:08:b0:70:a6:28:7d:8c:e5:3f:ac:15:5d
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:          CA:FALSE
            X509v3 Subject Key Identifier:     CA:B1:FD:89:DD:04:5F:A0:E3:9E:C7:50:3A:01:88:1F:8F:9E:8F:E9
            X509v3 Authority Key Identifier:   keyid:BC:7D:A2:9B:2D:FA:F9:51:6E:73:82:2B:30:02:EC:E3:33:2A:D7:40
    Signature Algorithm: sha1WithRSAEncryption
        a8:ea:f2:52:ea:83:45:44:0b:06:6b:9d:30:6e:ae:2c:7b:be:
        ...
        03:98:74:0f:90:d5:43:08:93:c6:b6:09:b4:ad:9c:08:d5:aa:cb:6c
-----BEGIN CERTIFICATE-----
MIIDPDCCAqWgAwIBAgIJAKeq1e4OznfQMA0GCSqGSIb3DQEBBQUAMGwxCzAJBgNV
...
nEzxzqMbRtwVhbl5IhEo2tK8fwSpm6iiI1zWbO+euHBJyFpOJaL9jjLMJ1gDmHQP
kNVDCJPGtgm0rZwI1arLbA==
-----END CERTIFICATE-----
```

SSL – Wykorzystanie certyfikatów

Wykorzystanie w serwerze Apache:

- Konfiguracja */etc/apache/mod_ssl.conf*

```
SSLCertificateKeyFile /etc/apache/ssl.key/server.key
SSLCertificateFile /etc/apache/ssl.crt/server.crt
```

  W tych 2 plikach należy umieścić wygenerowany wcześniej klucz oraz podpisany certyfikat.

- Jeśli klucz jest zabezpieczony hasłem, każdorazowe uruchomienie serwera będzie wymagało jego odblokowania
- Odblokowanie klucza i zapisanie wersji bez hasła:

```
openssl rsa < server.key > unencrypted.key
```

Po stronie klienta (zdalny dostęp do serwera po HTTP)

- Zdalne pobranie certyfikatu serwera:

```
openssl s_client -connect ad.res.serwe.ra:443 | \
  sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > cert.pem
```

- Sprawdzenie zawartości certyfikatu:

```
openssl x509 -noout -subject -dates -in cert.pem
openssl x509 -noout -text -in cert.pem
```

SSL – Using client certificates

Certyfikaty klienckie:

- Certyfikaty klienckie są tworzone w taki sam sposób
- Muszą być podpisane przez CA rozpoznawane przez serwer
- Autoryzacja dostępu działa bazując na weryfikacji podpisu certyfikatu a następnie – porównaniu treści certyfikatu (niektórych pól) z wymaganiami
- Reguły dostępu – w konfiguracji modułu `mod_ssl`

Korzystanie z cetryfikatów klienckich

- Apache może wymusić stosowanie certyfikatów przez wszystkich klientów
- Niektóre -e mogą wymagać certyfikatów do dostępu, podczas gdy reszta katalogów serwera jest dostępna dla wszystkich, także nie autoryzujących dostępu certyfikatami.
- Serwer może udostępniać pliki/katalogi na podstawie nazwy użytkownika w certyfikacie, albo dowolnego innego pola certyfikatu (np. nazwy organizacji), pozwalając w ten sposób na dostęp grupowy

Więcej informacji: (http://httpd.apache.org/docs/2.2/ssl/ssl_howto.html.en#upgradeenc)

SSL – certyfikat klienta

Certyfikat klienta to nic innego jak zwykły certyfikat podpisany przez odpowiednie CA i zapisany w formacie PKCS12:

❶ Utworzenie klucza (*user.key*) oraz certyfikatu (*user.csr*) to wygenerowanie „certificate signing requests". Tak utworzony klucz będzie zabezpieczony hasłem. Bez hasła: `-nodes`

```
openssl req -new  -newkey rsa:1024 -keyout user.key -out user.csr
```

✪ Sprawdzenie treści tego żądania:

```
openssl req -in user.csr -noout -text
```

❷ Podpisanie tego żądania za pomocą klucza CA:

```
# Własne CA w katalogu demoCA (demoCA/cacert.pem, demoCA/private/cakey.pem)
openssl ca -policy policy_anything -out user.pem -infiles user.csr
```

❸ Połączenie podpisanego certyfikatu (*user.pem*) i klucza (*user.key*) do jednego pliku (*user2.pem*):

```
cat user.key user.pem  > user2.pem
```

❹ Eksport certyfikatu użytkownika do pliku PKCS12:

```
openssl pkcs12 -export -out user.pfx -in user2.pem -name "User Cert"

Enter pass phrase for user2.pem: ******
Enter Export Password: ******
Verifying - Enter Export Password: ******
```

- Plik *user.pfx* zawiera certyfikat i klucz uzytkownika w formacie PKCS12. Sprawdzenie zawartości:

```
openssl pkcs12 -in  user.pfx -nodes

Enter Import Password:
MAC verified OK
Bag Attributes
friendlyName: User Cert
localKeyID: 84 EB F1 51 33 40 55 62 08 EC 99 86 FC E5 68 70 77 87 59 E4
subject=/C=PL/ST=aaaa/L=BBBB/O=cccc/OU=DDDD/CN=User Name/emailAddress=user@here
issuer=/C=PL/ST=Dolnoslaskie/O=Demo CA org/CN=laptok CA/emailAddress=laptok@laptok
-----BEGIN CERTIFICATE-----
MIIC3DCCAkWgAwIBAgIJAKeq1e4OznfRMA0GCSqGSIb3DQEBBQUAMGwxCzAJBgNV
...
Wo4MSI19bUnc3Fojovg8CQ==
-----END CERTIFICATE-----
Bag Attributes
friendlyName: User Cert
localKeyID: 84 EB F1 51 33 40 55 62 08 EC 99 86 FC E5 68 70 77 87 59 E4
Key Attributes: <No Attributes>
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,971347A4DFA8E807

Bo5YL9BPbiI7iIyppQNnXe3QJCpscKNVQFs3ZyHPlExvZZ+L55V6aIeD2IRF8hkq
...
-----END RSA PRIVATE KEY-----
```

- Plik *user.pem* (krok ❷) zawiera certyfikat użytkownika podpisany przez CA:

```
Issuer: C=PL, ST=Dolnoslaskie, O=Demo CA org, CN=laptok CA/emailAddress=laptok@laptok
Subject: C=PL, ST=aaaa, L=BBBB, O=cccc, OU=DDDD, CN=User Name/emailAddress=user@here
```

Literatura

[1] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176.

[2] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[3] E. Rescorla and N. Modadugu. Datagram Transport Layer Security. RFC 4347 (Proposed Standard), April 2006. Updated by RFC 5746.

[4] E. Rescorla and A. Schiffman. Security Extensions For HTML. RFC 2659 (Experimental), August 1999.

[5] E. Rescorla and A. Schiffman. The Secure HyperText Transfer Protocol. RFC 2660 (Experimental), August 1999.