Ćwiczenie laboratoryjne
Konstrukcja urzędów certyfikacji w oparciu o pakiet OpenSSL, zarządzanie certyfikatami

Cel ćwiczenia

Celem laboratorium jest utworzenie bezpiecznej ścieżki komunikacji pomiędzy dwoma komputerami. Aby tego dokonać niezbędne jest utworzenie mikro urzędu certyfikacji, wygenerowanie certyfikatu dla serwera OpenSSL i podpisanie go oraz sprawdzenie komunikatów przepływających pomiędzy klientem i serwerem.

1. Wprowadzenie

Technologia SSL(*Secure Sockets Layer*) zaproponowana przez firmę Netscape Communications na potrzeby szyfrowania i uwierzytelniania komunikacji została bardzo dobrze przyjęta przez wszystkich zainteresowanych. Obecnie technologia SSL jest szeroko wykorzystywana jako mechanizm bezpiecznej komunikacji nie tylko w obrębie protokołu HTTP ale służy również do tworzenia sieci VPN(Virtual Private Network). Najnowsza wersja protokołu SSL to wersja 3.0. U podstaw protokołu SSL leży praktyczne wykorzystanie kryptografii asymetrycznej oraz symetrycznej. Celem ćwiczenia jest zrozumienie działania protokołu SSL oraz poznanie pakietu OpenSSL.

Zadaniem protokołu SSL jest zapewnienie szyfrowanego i/lub uwierzytelnionego kanału przesyłania danych. Biorąc pod uwagę warstwowość sieciowego modelu ISO/OSI protokół SSL można umiejscowić powyżej protokołu TCP czyli warstwy transportowej ale poniżej warstw wyższych. Protokół SSL w imieniu warstw wyższych modelu ISO/OSI wykorzystuje protokół TCP do ustanowienia kanału wymiany danych. Analizując działania protokołu SSL można wyróżnić kilka zdarzeń, które mogą zajść w trakcie korzystania z protokołu SSL. Są to:

- Uwierzytelnienie serwera wspierającego SSL – strona będąca klientem połączenia SSL ma możliwość zweryfikowania tożsamości strony serwera poprzez wykorzystanie mechanizmów kryptografii klucza publicznego i sprawdzenie czy certyfikat strony serwera jest poprawny oraz czy został podpisany przez znaną klientowi zaufaną stronę trzecia, centrum certyfikacji(CA, *ang. Certification Authority*).
- Uwierzytelnienie klienta wspierającego SSL – strona będąca serwerem ma możliwość zweryfikować tożsamość klienta żądającego nawiązania bezpiecznego połączenia. Serwer przeprowadza proces weryfikacji klienta identycznie jak zostało to opisane powyżej.
- Nawiązanie szyfrowanego połączenia – obie strony połączenia dążą do nawiązania bezpiecznego połączenia, którym mogą być przesyłane poufne dane.

2. Etap nawiązania sesji SSL

Pierwszym krokiem w celu utworzenia bezpiecznego kanału wymiany danych między serwerem a klientem jest etap przywitania(ang. handshake), w którym strona serwera uwierzytelnia się przed stroną klienta oraz opcjonalnie strona klienta uwierzytelnia się przed stroną serwera. Oba procesy uwierzytelnia wykorzystują mechanizmy kryptografii klucza publicznego, aby w wiarygodny sposób zweryfikować tożsamość drugiej strony. Poniżej zaprezentowano kolejne etapy w procesie przywitania między stroną serwera i klienta.

1. Strona klienta wysyła do strony serwera numer obsługiwanej wersji protokołu SSL, możliwe do użycia algorytmy szyfrujące oraz zestaw danych losowych.
2. Strona serwera wysyła do strony klienta numer obsługiwanej wersji protokołu SSL, możliwe do użycia algorytmy szyfrujące, zestaw danych losowych oraz własny certyfikat zawierający publiczny klucz serwera. Może również zdarzyć się, że serwer zażąda aby klient uwierzytelnił się. W takiej sytuacji serwer wysyła również żądanie o certyfikat strony klienta.
3. Strona klienta przystępuje do weryfikacji tożsamości strony serwera. Jeżeli wystąpi problem z weryfikacją tożsamości strony serwera użytkownik jest o tym informowany.

4. Po pomyślnym uwierzytelnieniu strony serwera, strona klienta generuje tajny ciąg znaków określany mianem *premaster secret*. Dysponując certyfikatem strony serwera, strona klienta używa klucza publicznego strony serwera, szyfruje tajny ciąg *premaster secret* i wysyła go do strony serwera.
5. Opcjonalnie, serwer może zażądać aby strona klienta uwierzytelniła się. Strona klienta wysyła do strony serwera swój certyfikat z kluczem publicznym oraz podpisany swoim kluczem prywatnym losowy ciąg znaków znany stronie klienta i serwera.
6. Strona serwera przystępuje do weryfikacji tożsamości strony klienta. Jeżeli wystąpi problem z weryfikacją sesja SSL jest przerywana. Jeżeli weryfikacja zakończy się pozytywnie strona serwera używając swojego klucza prywatnego odszyfrowuje ciąg *premaster secret* i na jego podstawie generuje tajny ciąg znaków określany mianem *master secret*(strona klienta również na podstawie ciągu *premaster secret* generuje ciąg *master secret*).
7. Obie strony połączenia na podstawie tajnego ciągu znaków *master secret* generują ciąg *session key*. Tajny ciąg *session key* posłuży jako klucz dla algorytmu symetrycznego szyfrowania służącego do szyfrowania komunikacji między stronami.
8. Obie strony połączenia informują siebie wzajemnie, że od tej chwili komunikacja między nimi będzie szyfrowana.
9. Proces ustanowienia sesji SSL zostaje zakończony, bezpieczny kanał komunikacji został ustanowiony.

3. Infrastruktura klucza publicznego i certyfikaty cyfrowe

Protokół SSL wykorzystuje dwie technologie odnośnie mechanizmów szyfrowania. Jest to kryptografia symetryczna i kryptografia asymetryczna. Kryptografia symetryczna służy do szyfrowania danych wymienianych między stronami. Kryptografia asymetryczna służy do uzgodnienia między stronami tajnego klucza sesyjnego. Połączenie obu technologii pozwala na bezpieczne korzystanie z protokołu SSL. Omawiając protokół SSL należy również wspomnieć o Infrastrukturze klucza publicznego(*ang. PKI, Public Key Infrastructure*)oraz certyfikatach standardu X.509.

PKI to koncepcja rozbudowanego systemu kryptograficznego, w którym sieć wzajemnych powiązań między elementami tego systemu gwarantuje bezpieczeństwo komunikacji a konkretnie bezpieczeństwo i możliwość weryfikacji informacji o użytkownikach tego systemu. W skład PKI wchodzą:

- Urzędy certyfikacji(*ang. CA, Certification Authority*) odpowiedzialne za poświadczanie tożsamości klientów, tzw. zaufana strona trzecia (*ang. trusted third party*).
- Użytkownicy dla których wydawane są certyfikaty.
- Urzędy rejestracji(*ang. RA, Registration Authority*) odpowiedzialne za weryfikację danych o użytkownikach oraz ich rejestrację.
- Oprogramowanie oraz sprzęt niezbędny do posługiwania się certyfikatami.
- Repozytoria kluczy, certyfikatów i listy odwołanych certyfikatów(*ang. CRL, Certificate Revocation List*)

Struktura PKI przypomina drzewo, w którym w roli korzenia występuje główny urząd certyfikacji, który określa politykę certyfikacji np. dla danego kraju. Poniżej znajdują się podległe urzędy certyfikacji zajmujące się obsługą np. poszczególnych grup użytkowników lub świadczące usługi certyfikacji dla określonych zastosowań. Na samym dole tego drzewa jako liście występują użytkownicy, którzy zgłaszają się do urzędów certyfikacji w celu wydania im certyfikatów. Instytucja certyfikująca wydaje certyfikat i poświadcza tożsamość użytkownika któremu wydaje certyfikat. Użytkownik może sprawdzić, który urząd wydał dany certyfikat, może również sprawdzić całą ścieżkę zaufania od danego użytkownika do głównego urzędu certyfikacji. Cała struktura drzewiasta tworzy hierarchię zaufania. Należy zwrócić uwagę, iż w certyfikacie znajduje się tylko klucz publiczny użytkownika. Klucz prywatny(tajny), który jest zawsze matematycznie powiązany z kluczem publicznym jest znany tylko użytkownikowi. Urząd certyfikacji wydając certyfikat poświadcza, że podpis wykonany tajnym kluczem użytkownika został złożony przez tego użytkownika.

Struktura PKI podlega standaryzacji. Standard opiera się na dwóch elementach. Jednym jest standard X.509 opisujący strukturę certyfikatów oraz drugi standard określany mianem PKCS(*ang. Public Key Cryptography Standards*) którego autorem jest firma RSA Data Security.

Certyfikat cyfrowy jest informacją elektroniczną poświadczającą związek między danym użytkownikiem a kluczem, którego używa dany użytkownik do procedury podpisywania dokumentów. Standard X.509 definiuje budowę takiego certyfikatu i wskazuje jakie elementy powinny być zawarte w certyfikacie. Są to m.in.:

- numer wersji standardu X.509
- numer wystawionego certyfikatu
- identyfikator algorytmu podpisu certyfikatu
- nazwa funkcji skrótu kryptograficznego użyta przez wystawcę
- nazwa wystawcy certyfikatu
- daty ważności certyfikatu
- nazwa podmiotu dla którego wystawiany jest certyfikat
- parametry klucza publicznego podmiotu
- klucz publiczny podmiotu
- opcjonalne rozszerzenia
- podpis wystawcy certyfikatu

4. Oprogramowanie OpenSSL

Oprogramowanie Openssl jest ogólnie dostępną biblioteką funkcji kryptograficznych. Openssl zawiera popularną implementację protokołu SSL. Najczęściej można ją spotkać w systemach Unix/Linux, gdzie dużo oprogramowania korzysta z tej biblioteki. Openssl to nie tylko biblioteka funkcji użytecznych dla wymagającego obsługi protokołu SSL oprogramowania. Użytkownicy za pomocą tej biblioteki mogą również samodzielnie tworzyć część infrastruktury klucza publicznego poprzez utworzenie własnego centrum certyfikacji, generowanie par kluczy oraz wystawianie certyfikatów. W dalszej części opracowania zostanie pokazane w jaki sposób można utworzyć własne centrum certyfikacji, wygenerować prośbę o wystawienie certyfikatu oraz jak wystawić certyfikat.

Wraz z biblioteką Openssl dostarczany jest zestaw użytecznych skryptów, które ułatwiają realizację wyżej wymienionych zadań.

Biblioteka OpenSSL w wersji Linux/Windows jest dostępna pod adresem http://www.openssl.org.

4.1 Tworzenie własnego centrum certyfikacji

Do tworzenia własnego centrum certyfikacji można wykorzystać skrypt CA.sh (skrypt napisany w języku interpretera bash powinien znajdować się w katalogu /etc/pki/tls/misc). Aby wygenerować i utworzyć własne mikro centrum certyfikacji należy wydać komendę:

/usr/lib/ssl/misc/CA.sh –newca

Poniżej zaprezentowano przykład utworzenia centrum certyfikacji:

```
root@slinux:/usr/share/ssl/misc # ./CA.sh -newca

CA certificate filename (or enter to create)
Making CA certificate ...
Generating a 1024 bit RSA private key
.......++++++
.................++++++
writing new private key to './demoCA/private/./cakey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

phrase is too short, needs to be at least 4 chars
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:**PL**
State or Province Name (full name) [Some-State]: **Mazowieckie**
Locality Name (eg, city) []:**Warszawa**
Organization Name (eg, company) [Internet Widgits Pty Ltd]:**SGGW**
Organizational Unit Name (eg, section) []:**Katedra Informatyki**
Common Name (eg, YOUR name) []:**mikro_CA**
Email Address []: **admin@SGGW_x.pl**
…..
root@slinux:/usr/share/ssl/misc #

Pogrubionym drukiem zaznaczono przykładowy tekst, który został wpisany przez użytkownika. Utworzenie centrum certyfikacji spowoduje utworzenie podkatalogu demoCA. W tym katalogu będzie znajdował się plik cacert.pem, który jest certyfikatem dla utworzonego urzędu certyfikacji. W podkatalogu demoCA/private/ będzie znajdował się klucz prywatny urzędu certyfikacji, plik cakey.pem. Podczas procesu tworzenia centrum certyfikacji skrypt zapyta użytkownika o frazę hasłową, która będzie użyta do zaszyfrowania klucza prywatnego centrum certyfikacji przechowywanego w pliku cakey.pem.

UWAGA! Nazwy katalogów i nazwy plików pomą się różnić w zależności od dystrybucji systemu Linux.

UWAGA! Zgubienie tego klucza spowoduje niemożność wystawiania certyfikatów przez to centrum certyfikacji.

4.2 Generowanie prośby o wystawienie certyfikatu

Aby uzyskać certyfikat od urzędu certyfikatu należy najpierw zgłosić żądanie wystawienia certyfikatu. Polecenie:

/usr/lib/ssl/misc/CA.sh -newreq

wygeneruje prośbę o wystawienie certyfikatu w formie pliku tekstowego.

Poniżej zaprezentowano przykład generacji żądania o wystawienie certyfikatu:

root@slinux:/usr/share/ssl/misc # **./CA.sh -newreq**

Generating a 1024 bit RSA private key
.....++++++
.++++++
writing new private key to 'newreq.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:**PL**
State or Province Name (full name) [Some-State]:**Zachodniopomorskie**

Locality Name (eg, city) []:**Szczecin**
Organization Name (eg, company) [Internet Widgits Pty Ltd]:**ZUT**
Organizational Unit Name (eg, section) []:**Wydzial Informatyki**
Common Name (eg, YOUR name) []:**firmy_X**
Email Address []:**admin@firma_x.pl**
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []: **nic**
An optional company name []: **nic**
Request (and private key) is in newreq.pem

root@slinux:/usr/share/ssl/misc #

W pliku *newreq.pem* będzie znajdowało się żądanie wydania certyfikatu oraz klucz prywatny podmiotu żądającego wystawienia certyfikatu.

4.3 Wystawienie certyfikatu

Aby wystawić certyfikat na podstawie żądania wystawienie certyfikatu należy wydać polecenie:

/usr/lib/ssl/misc/CA.sh -sign

Poniżej zaprezentowano przebieg procedury wystawienia certyfikatu:

root@slinux:/usr/share/ssl/misc # ./CA.sh -sign

Using configuration from /etc/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
Serial Number: 1 (0x1)
Validity
Not Before: Jul 15 09:10:00 2006 GMT
Not After : Jul 15 09:10:00 2007 GMT
Subject:
countryName = PL
stateOrProvinceName = Zachodniopomorskie
localityName = Szczecin
organizationName = FIRMA X
organizationalUnitName = Dzial Y
commonName = nazwa_serwera_firmy_X
emailAddress = admin@firma_x.pl
X509v3 extensions:
X509v3 Basic Constraints:
CA:FALSE
Netscape Comment:
OpenSSL Generated Certificate
X509v3 Subject Key Identifier:
12:B8:BF:4D:BF:B9:40:C9:2E:C1:7A:DE:A8:4B:75:58:D8:C6:CD:3D
X509v3 Authority Key Identifier:
keyid:BF:62:BF:05:7C:05:30:E5:E3:7A:90:82:0C:61:45:60:7E:D8:F3:49
DirName:/C=PL/ST=Zachodniopomorskie/L=Szczecin/O=ZPSB/OU=Katedra
Informatyki/CN=mikro_CA
serial:8D:A7:61:FC:60:94:39:C1
Certificate is to be certified until Dec 32 09:10:00 2008 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
Certificate:

```
  Data:
    Version: 3 (0x2)
    Serial Number: 1 (0x1)
    Signature Algorithm: md5WithRSAEncryption
    Issuer: C=PL, ST=Zachodniopomorskie, L=Szczecin, O=ZPSB, OU=Katedra Informatyki,
CN=mikro_CA
    Validity
      Not Before: Dec 31 09:10:00 2006 GMT
      Not After : Dec 31 09:10:00 2007 GMT
    Subject: C=PL, ST=Zachodniopomorskie, L=Szzcecin, O=FIRMA X, OU=Dzial Y,
CN=nazwa_serwera_firmy_X/emailAddress=admin@firma_x.pl
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (1024 bit)
        Modulus (1024 bit):
          00:b8:b1:2b:00:8b:de:2a:bf:93:ca:e4:a9:ac:2b:
          7a:be:b5:3e:0c:ca:24:55:f7:65:6b:66:fa:34:d2:
          0c:96:76:bb:01:f0:2b:24:27:99:65:11:78:51:f9:
          6b:a7:96:ee:b7:98:45:99:46:ed:1e:13:c9:bb:ab:
          99:c1:59:0e:ac:b8:89:9f:6a:11:b4:04:97:66:7b:
          92:c1:19:c1:82:90:5e:df:c1:85:96:e7:9b:44:d3:
          ee:67:8a:c1:e9:e3:14:a2:ef:0c:13:39:c0:55:19:
          8a:3b:25:aa:49:b9:1e:ab:c4:1d:81:56:90:cc:76:
          89:b1:ea:d8:40:e7:c7:0b:4f
        Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Basic Constraints:
        CA:FALSE
      Netscape Comment:
        OpenSSL Generated Certificate
      X509v3 Subject Key Identifier:
        12:B8:BF:4D:BF:B9:40:C9:2E:C1:7A:DE:A8:4B:75:58:D8:C6:CD:3D
      X509v3 Authority Key Identifier:

keyid:BF:62:BF:05:7C:05:30:E5:E3:7A:90:82:0C:61:45:60:7E:D8:F3:49
          DirName:/C=PL/ST=Zachodniopomorskie/L=Szczecin/O=ZPSB/OU=Katedra
Informatyki/CN=mikro_CA
          serial:8D:A7:61:FC:60:94:39:C1
  Signature Algorithm: md5WithRSAEncryption
    b2:bb:a6:c5:cf:5b:c7:25:42:dd:49:6d:34:f7:a0:8f:33:1d:
    2a:3f:6a:12:62:b6:48:d3:c2:1a:4a:d6:5b:14:4e:8e:98:86:
    e9:86:94:57:14:92:1a:38:46:76:0d:7f:8c:82:a0:c8:99:9c:
    60:03:f7:5a:f8:46:d7:f5:07:7a:97:19:46:5c:b8:f0:e8:ce:
    e6:0b:cd:fd:4d:22:91:28:83:af:a1:4a:b4:51:eb:bb:7d:1a:
    47:b3:8d:b5:f3:ba:dc:75:a9:5a:7d:02:43:ae:e4:37:8c:8d:
    b4:47:70:b4:86:2a:a3:d2:94:86:23:d8:8d:43:db:18:a8:1b:
    c1:39
-----BEGIN CERTIFICATE-----
MIID+jCCA2OgAwIBAgIBATANBgkqhkiG9w0BAQQFADCBrDELMAkGA1UEBhMCUEwx
DTALBgNVBAgTBFdMS1AxEDAOBgNVBAcUB1Bvem5hxYQxIDAeBgNVBAoUF1BvbGl0
ZWNobmlrYSBQb3puYcWEc2thMR0wGwYDVQQLExRJbnN0eXR1dCBJbmZvcm1hdHlr
aTEVMBMGA1UEAxQMY2FfbG9jYWxob3N0MSQwIgYJKoZIhvcNAQkBFhVjYV9hZG1p
bkBsb2NhbGhvc3QucGwwHhcNMDYwNzE1MDkxMDAwWhcNMDcwNzE1MDkxMDAwWjCB
lDELMAkGA1UEBhMCUEwxDTALBgNVBAgTBFdMS1AxEDAOBgNVBAcUB1Bvem5hxYQx
EDAOBgNVBAoTB0ZJUk1BIFgxETAPBgNVBAsUCER6aWHFgiBZMR4wHAYDVQQDFBVu
YXp3YV9zZXJ3ZXJhX2Zpcm15X1gxHzAdBgkqhkiG9w0BCQEWEGFkbWluQGZpcm1h
```

X3gucGwwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBALixKwCL3iq/k8rkqawr
er61PgzKJFX3ZWtm+jTSDJZ2uwHwKyQnmWUReFH5a6eW7reYRZlG7R4TyburmcFZ
Dqy4iZ9qEbQEl2Z7ksEZwYKQXt/BhZbnm0TT7meKwenjFKLvDBM5wFUZijslqkm5
HqvEHYFWkMx2ibHq2EDnxwtPAgMBAAGjggFAMIIBPDAJBgNVHRMEAjAAMCwGCWCG
SAGG+EIBDQQfFh1PcGVuU1NMIEdlbmVyYXRlZCBDZXJ0aWZpY2F0ZTAdBgNVHQ4E
FgQUEri/Tb+5QMkuwXreqEt1WNjGzT0wgeEGA1UdIwSB2TCB1oAUv2K/BXwFMOXj
epCCDGFFYH7Y80mhgbKkga8wgawxCzAJBgNVBAYTAlBMMQ0wCwYDVQQIEwRXTEtQ
MRAwDgYDVQQHFAdQb3puYcWEMSAwHgYDVQQKFBdQb2xpdGVjaG5pa2EgUG96bmHF
hHNrYTEdMBsGA1UECxMUSW5zdHl0dXQgSW5mb3JtYXR5a2kxFTATBgNVBAMUDGNh
X2xvY2FsaG9zdDEkMCIGCSqGSIb3DQEJARYVY2FfYWRtaW5AbG9jYWxob3N0LnBs
ggkAjadh/GCUOcEwDQYJKoZIhvcNAQEEBQADgYEAsrumxc9bxyVC3UltNPegjzMd
Kj9qEmK2SNPCGkrWWxROjpiG6YaUVxSSGjhGdg1/jIKgyJmcYAP3WvhG1/UHepcZ
Rly48OjO5gvN/U0ikSiDr6FKtFHru30aR7ONtfO63HWpWn0CQ67kN4yNtEdwtIYq
o9KUhiPYjUPbGKgbwTk=
-----END CERTIFICATE-----
Signed certificate is in newcert.pem

root@slinux:/usr/share/ssl/misc #

Po zakończeniu procedury wystawienia certyfikatu w pliku newcert.pem znajduje się wystawiony certyfikat.

4.4 Podsumowanie

Biblioteka Openssl jest szeroko stosowanym oprogramowaniem, które często jest elementem obowiązkowym innych programów. Niewątpliwą zaletą tej biblioteki jest również możliwość tworzenia certyfikatów w przyjazny dla użytkownika sposób, co pozwala ja wykorzystanie wsparcia dla SSL np. w serwerach www np. serwer Apache.

5. Ćwiczenia

Czynności początkowe

Uruchomić system *ubuntu10tm* i wejść na konto "user", hasło "123456".
Po uruchomieniu systemu Linux Ubuntu przejść do konsoli i przełączyć się w tryb superużytkownika
 # sudo -i
Jeśli w systemie nie jest zainstalowany kompilator języka C++, to zainstaluj go:
 # apt-get install g++
Sprawdzić czy w systemie zainstalowany jest OpenSSL, jeśli nie to zainstaluj go:
 # openssl version
 # apt-get install openssl
Uwaga! W przypadku pojawienia się błędów podczas instalowania pakietów (nie tylko tych powyżej, ale także innych) należy wykonać aktualizację oprogramowania:
 # apt-get update
Zainstalować biblioteki pakietu OpenSSL:
$ sudo apt-get install libssl-dev
Pobierz i zainstaluj pakiet **Wireshark**.
$ sudo apt-get install wireshark
Do ćwiczenia w pliku Aplikacje dołączone są cztery pliki źródłowe (napisane w języku C). Programy te należy pobrać z poczty i umieścić je w katalogu **Desktop/openssl/Aplikacje**.
Dwa pierwsze pliki **bzssl** (client.c i server.c) są odpowiednio oprogramowaniem klienta umożliwiającego połączenie się z podanym adresem IP i portem, na którym nasłuchuje serwer. Po nawiązaniu połączenia można podać wiadomość, która jest przesyłana tekstem otwartym do serwera. Oba otrzymane pliki należy skompilować (np. za pomocą kompilatora **gcc**) i utworzyć pliki wykonywalne.
Desktop/openssl/Aplikacje/bzssl #gcc server.c –o server
Desktop/openssl/Aplikacje/bzssl #gcc client.c –o client
Dwa kolejne pliki **zssl** (ssl_client cpp i ssl_server.cpp**)** odpowiadają funkcjom aplikacji przedstawionych powyżej, z tym tylko, że tym razem pomiędzy klientem a serwerem zestawiane jest połączenie szyfrowane.

Zadanie 1: Sprawdzenie pakietów przesyłanych podczas połączenia nieszyfrowanego.

1. Uruchom program **Wireshark z prawami administratora** i rozpocznij podsłuchiwanie swojej karty sieciowej. Z menu Capture wybierz opcję "start", a następnie wybierz interfejs sieciowy i kliknij „ok.". W tym momencie program **Wireshark** odczytuje wszystkie informacje przechodzące przez kartę sieciową Twojego komputera.
Odczytaj i zapisz adres ip swojego komputera wpisując polecenie
Desktop/openssl/Aplikacje/bzssl #ifconfig
Aby móc uruchomić program, należy zmienić tryb dostępu dla pliku – dodać tryb wykonywania
Desktop/openssl/Aplikacje/bzssl # chmod +x server
Czynność tą należy wykonać tylko w przypadku, gdy plik wykonywalny nie posiada odpowiednio ustawionych praw wykonania.
UWAGA! Aby uruchomić program w systemie Linux należy przed jego nazwą dodać znaki ./, np.
Desktop/openssl/Aplikacje/bzssl # ./server 1101 (numer portu)
Utwórz nową konsolę. Aby móc uruchomić program, należy zmienić tryb dostępu dla pliku – dodać tryb wykonywania
Desktop/openssl/Aplikacje/bzssl # chmod +x client
Czynność tą należy wykonać tylko w przypadku, gdy plik wykonywalny nie posiada odpowiednio ustawionych praw wykonania.
UWAGA! Aby uruchomić program w systemie Linux należy przed jego nazwą dodać znaki ./, np.
Desktop/openssl/Aplikacje/bzssl # ./client 127.0.0.1 1101 lub Desktop/openssl/Aplikacje/bzssl # ./client 10.0.2.15 1101
W programie **Wireshark** zakończ przechwytywanie i spróbuj odczytać jakie informacje zostały przesłane do Ciebie. Czy jest to możliwe?

Zadanie 2: Tworzenie certyfikatu CA oraz certyfikatów klienta i serwera

Aby programy z zadania 3 mogły działać, niezbędny jest certyfikat serwera oraz certyfikat CA. Certyfikat CA jest certyfikatem urzędu certyfikacji. Ponieważ na potrzeby laboratorium nie jest wymagany certyfikat zaufanego CA – wygenerowany zostanie certyfikat zastępczy (lokalnie na komputerze).
Certyfikat jest kluczem publicznym serwera wraz z jego opisem oraz podpisem cyfrowym instytucji wyższego poziomu.
Wykonaj teraz poniższe czynności:
1. Utwórz własny mikrourząd certyfikacji za pomocą biblioteki openssl (patrz rozdz.4.1)

Desktop/openssl/Aplikacje/zssl#/usr/lib/ssl/misc/CA.sh –newca

2. Wygeneruj żądania wystawienia certyfikatu dla serwera (program **ssl_server**), patrz rozdz.4.2.

Desktop/openssl/Aplikacje/zssl#/usr/lib/ssl/misc/CA.sh –newreq

3. Wystaw certyfikat dla serwera (patrz rozdz.4.3).

Desktop/openssl/Aplikacje/zssl#/usr/lib/ssl/misc/CA.sh –sign

Zamien nazwy newcert.pem na servercert.pem oraz newkey.pem na serverkey.pem.

Punkt 2 i 3 powtórzyć. Dwa pliki newkey.pem i newcert.pem zamienić na clientkey.pem i clientcert.pem.

Zadanie 3: Tworzymy połączenia szyfrowane

Serwer jest napisany następująco:

- zdefiniowano położenie klucza i certyfikatu

  ```
  #define CERTF "plik certyfikatu serwera"
  #define KEYF "plik klucza serwera"
  #define CAFILE "certyfikat Root CA"
  ```

- zainicjowano OPENSSL i wybrano protokół komunikacji (SSLv2)

  ```
  // Inicjowanie open ssl
  SSL_load_error_strings();
  SSLeay_add_ssl_algorithms();
  meth = SSLv23_server_method();
  ```

- utworzono strukturę SSL_CTX (zawiera ona domyślne wartości połączenia SSL) za pomocą funkcji:

SSL_CTX *SSL_CTX_new(SSL_METHOD *)

- wskazano serwerowi certyfikat za pomocą funkcji:

  SSL_CTX_use_certificate_file(SSL_CTX *struktura_ctx, char *sciezka_certyfikatu, SSL_FILETYPE_PEM);

- wskazano serwerowi jakiego ma użyć klucza za pomocą funkcji:

  SSL_CTX_use_PrivateKey_file(SSL_CTX *struktura_ctx , char *sciezka_klucza, SSL_FILETYPE_PEM);

- uworzono obiekt SSL

  SSL *SSL_new(SSL_CTX*);

- pobrano certyfikat klienta

  X509 * SSL_get_peer_certificate(SSL*);

- wymiana danych:

  - odczytanie

    int SSL_read(SSL *struktura_ssl, char *bufor, sizeof(bufor) -1);

  - - wysłanie

    int SSL_write(SSL *struktura_ssl, char *tekst, strlen(*tekst));

Kompilowanie serwera

w pliku ssl_server.cpp należy zmienić deklarację o postaci:
const SSL_METHOD *meth; na deklarację: SSL_METHOD *meth;
oraz #define CAFILE ”cacert.pem” na #define CAFILE ”demoCA/cacert.pem”
nastepnie:

Desktop/openssl/Aplikacje/zssl#g++ ssl_server.cpp - lssl -lcrypto -o ssl_server

gdzie: -lssl – dołącza biblioteki ssl, zaś -lcrypto – dołącza biblioteki kryptograficzne

Z kolei:

Klient jest napisany następująco:

- zdefiniowano położenie klucza i certyfikatu

  #define CERTF "plik certyfikatu klienta"
  #define KEYF "plik klucza klienta"
  #define CAFILE "certyfikat Root CA"

- zainicjowano OPENSSL i wybrano protokół komunikacji (SSLv2)

  // Inicjowanie open ssl
  SSL_load_error_strings();
  SSLeay_add_ssl_algorithms();
  meth = SSLv2_client_method();

- utworzono strukturę SSL_CTX (zawiera ona domyślne wartości połączenia SSL) za pomocą funkcji:

  SSL_CTX *SSL_CTX_new(SSL_METHOD *)

- wskazano klientowi certyfikat za pomocą funkcji:

  SSL_CTX_use_certificate_file(SSL_CTX *struktura_ctx, char *sciezka_certyfikatu, SSL_FILETYPE_PEM);

- wskazano klientowi jakiego ma użyć klucza za pomocą funkcji:

  SSL_CTX_use_PrivateKey_file(SSL_CTX *struktura_ctx , char *sciezka_klucza, SSL_FILETYPE_PEM);

- uworzono obiekt SSL

SSL *SSL_new(SSL_CTX*);

- pobrano certyfikat serwera

  X509 * SSL_get_peer_certificate(SSL*);

- wymiana danych:

  - odczytanie

    int SSL_read(SSL *struktura_ssl, char *bufor, sizeof(bufor) -1);

  - - wysłanie

    int SSL_write(SSL *struktura_ssl, char *tekst, strlen(*tekst));

Kompilowanie klienta

w pliku ssl_client.cpp należy zmienić deklarację o postaci:
const SSL_METHOD *meth; na deklarację: SSL_METHOD *meth;
oraz #define CAFILE ”cacert.pem” na #define CAFILE ”demoCA/cacert.pem”
nastepnie:
Desktop/openssl/Aplikacje/zssl#g++ ssl_client.cpp - lssl -lcrypto -o ssl_client

Zadanie 3: Tworzymy połączenia szyfrowane

Czynność wstępna:

- Uruchom program **ssl_server**.

  Po uruchomieniu servera podajemy hasło do klucza prywatnego podanego podczas tworzenia certyfikatu serwera. Po podaniu hasła program nasłuchuje na określonym porcie na tekst od klienta.

Czynności kolejne:
Uruchom program **Wireshark** i rozpocznij podsłuchiwanie swojej karty sieciowej

Uruchomienie klienta **ssl_client**.

Po uruchomieniu programu należy wpisać dowolny tekst, który zostanie przesłany do serwera.

Czynności końcowe:
W programie **Wireshark** spróbuj odczytać informacje, które zostały przesłane przez Ciebie. Czy jest to możliwe? Wnioski należy umieścić w sprawozdaniu
Spróbuj odczytać swoje hasło pocztowe programem **Wireshark**, logując się na swój serwer pocztowy.

Sprawozdanie z ćwiczenia

W trakcie ćwiczenia należy notować wszystkie czynności oraz uzyskiwane wyniki. Po zakończeniu ćwiczenia należy przygotować sprawozdanie z przebiegu ćwiczenia, zawierające m.in. krótki opis ćwiczenia, uzyskane wyniki oraz podsumowanie i wnioski z ćwiczenia.

Sprawozdanie powinno zawierać:

1. Wprowadzenie, cel ćwiczenia.
2. Dokładny opis i analiza wykonanych zadań – kierować się wskazówkami i pytaniami z treści ćwiczenia.
3. Wnioski.

Na ocenę z tego ćwiczenia będzie wpływać: przygotowanie teoretyczne do ćwiczenia, praca w czasie realizacji zadań w laboratorium oraz sprawozdanie oddane na następnych (po wykonaniu zadania) zajęciach.

Literatura

- Strona projektu Openssl http://www.openssl.org
- Strona firmy SUN poświęcona protokołowi SSL http://docs.sun.com/source/816-6156-10/contents.htm#1041643

- Strona Unizeto Certum http://ssl.certum.pl/certyfikaty/certy,support_faq.xml
- P. Chandra, M. Messier, J. Viega *Network Security with OpenSSL*, O'Reilly, June 2002