# Lucrarea nr. 7
# SSL

Mihai IVANOVICI

8 mai 2006

Odată cu răspândirea reţelelor şi a Internet-ului, s-au răspândit şi aplicaţiile ce presupun tranzaţii bancare sau comunicaţii private: utilizatorii fac plăţi "online" folosind cărţi de credit, companiile comunică prin Internet cu furnizorii sau colaboratorii etc.

Din pčate, au apărut şi o serie de atacuri al căror scop este fie subminarea măsurilor de securizare a informaţiilor sau provocarea de stricăciuni. Prin urmare, securitatea este foarte importantă pentru o multitudine de aplicaţii de reţea.

Securizare reţelelor şi a serviciilor oferite de către acestea este o bătălie care se dă pe mai multe fronturi. În primul rând trebuie ca PC-urile şi echipamentul de reţea să fie sigure din punct de vedere fizic—cu alte cuvinte, să fie "sub cheie" şi monitorizate continuu. Apoi atenţia trebuie concentrată asupra software-ului de securizare. Aplicaţiie cu "bug-uri" sau care au fost proiectate fără a se avea în vedere considerente de securitate constituie de asemenea o ameninţare. Un alt important considerent este acela al transmiterii datelor importante printr-o reţea nesigură. Secure Socket Layer (SSL) sau Transport Layer Security (TSL) este proiectat pentru a face comunicaţiile din reţea mult mai sigure, prin folosirea tehnicilor criptografice de autentificare.

În această lucrare vor fi prezentate cele mai cunoscute şi răspândite moduri de atac care vizează sistemele şi reţelele de calculatoare. Apoi vor fi prezentate modalităţi de prevenire a acestor atacuri, folosind SSL.

## 1 Vulnerabilitatea reţelelor

Înainte de a învăţa cum să securizăm un program, este important să înţelegem părţile vulnerabile ale acestuia.

Traficul existent în Internet trece printr-o serie de diverse reţele, fiecare controlată de o altă companie, care nu este neapărat de încredere. Reţelele care sunt fizic aproape de un client sau un server, cum ar fi reţeaua locală a unei companii sau a unei universităţi, este vulnerabilă la atacurile unui

student supărat. Unele servicii cum ar fi modem-urile sau accesul wireless la Internet sunt cunoscute pentru expunerea traficului către oricine dispus să asculte.

Chiar dacă reţelele sunt sigure (de încredere), există întotdeauna un atacator extern care găseşte o modalitate de a sparge securitatea sistemului sau de a intercepta o conexiune. Multe din atacurile descrise în secţiunile următoare sunt cunoscute sub denumirea generică de atacuri "man-in-the-middle" (MIMT), deoarece în general atacatorul se află undeva într-o reţea aflată între nodurile atacate.

## 1.1 Sniffing-ul

Prin "sniffing" se înţelege capacitatea unei persoane din exterior de a monitoriza traficul de reţea. Un "sniffer" nu modifică şi nici nu întrerupe traficul, ceea ce face sniffingul imposibil de detectat.

Sniffing-ul de trafic are ca scop principal furtul de parole sau numere de carduri de credit, citirea de e-mail-uri, sau pur şi simplu monitorizarea activităţilor legate de reţea.

## 1.2 Atacuri de tip "inserare"

Acestea apar atunci când cineva adaugă date nedorite la un stream de pachete din reţea. De exemplu, în cazul unei aplicaţii de tip FTP, un atacator ar putea adăuga o comandă de ştergere de fişiere, la stream-ul de comenzi. În acest fel atacatorul poate avea acces la sistemul de fişiere, fără a cunoaşte parola utilizatorului.

## 1.3 Atacuri de ştergere

Acestea sunt similare atacurilor de tip "inserare", doar că în cazul acestora, pachetele sau datele dintr-un stream sunt şterse. De exemplu, dacă utilizatorul trimite o comandă `rm *.txt` către o maşină Unix, dorind să şteargă fişierele text, atacaul de tip ştergere va şterge şirul de caractere `.txt` din comanda utilizatorului, rezultând o comandă `rm *` care va şterge toate fişierele.

## 1.4 Atacuri de tip "replay"

Aceste atacuri apar atunci când cineva a monitorizat traficul din reţea (sniffing) şi apoi îl retransmite prin reţea (replay). Din acest motiv, o schemă simplă de criptare bazată pe parolă, nu este cea mai bună soluţie.

### 1.5 Deturnarea sesiunii

Aceste atacuri apar atunci când utilizatorul deschide o conexiune pe un server pentru un anumit serviciu, iar un atacator va prelua rolul utilizatorului în comunicaţia cu serverul. Atacul este similar inserării sau ştergerii. Atacatorul va fi capabil să ruleze comenzi în locul utilizatorului şi să aibă acces la datele la care are acces utilizatorul.

### 1.6 Server fals sau redirectarea traficului

Un atacator poate instala/rula un server care să mimeze un server real şi în acest fel să fure parolele utilizatorilor.

### 1.7 Server compromis

Uneori atacatorii pot sparge securitatea unui server, preluând controlul asupra acestuia. În acest caz nimic nu se mai poate face protejarea unui program sau pentru protejarea datelor.

## 2 Reducerea vulnerabilităţii folosind SSL

SSL este proiectat pentru a reduce sau chiar elimina multe din vulnerabilităţile menţionate, prin:

- Autentificarea serverului şi opţional şi a clienţilor, pentru a preveni atacurile de tip server fals.
- Criptarea streamurilor de date în ambele sensuri, pentru a preveni "sniffingul".
- Schimbarea cheilor de criptare pentru ca aceleaşi date să fie reprezentate în mod diferit la momente diferite de timp, pentru a preveni atacurile de tip "replay".
- Verificarea integrităţii datelor pentru a preveni deturnările de sesiune şi atacurile de tip inserarea sau ştergere.

Bibliotecile SSL pun la dispoziţie funţii care se ocupă de toate aceste aspecte.

### 2.1 Autentificarea serverului

SSL foloseşte criptarea bazată pe cheie publică. Pentru criptare, fiecare capăt al conexiunii (clientul şi serverul) are o pereche de chei: o cheie publică şi una privată. Cheia privată este cunoscută doar pe maşina căreia îi aparţine, pe când cea publică poate fi cunoscută de oricine.

Mesajele sunt criptate folosind cheia publică şi decriptate folosind cheia privată. Similar, semnăturile digitale sunt fabricate folosind o cheie privată şi pot fi verificate folosind o cheie publică.

Întrebarea care necesită un răspuns este dacă cheia publică dată de către un server este într-adevăr cheia autentică asociată acelui server, şi nu este cheia publică a unui server care a interceptat şi redirecţionează traficul.

În Internet, problema este în general rezolvată prin folosirea unei liste de "Certificates Authorities (CA)". O autoritate de certificare (sau certificate) este o organizaţie, ca Thawte sau RSA, care verifică faptul că o anumită cheie publică este deţinută de serverul (compania, societatea) care pretinde că o deţine. În acest fel, tot ce are nevoie un browser de Internet, de exemplu, este o lista cu autorităţile de certificare de încredere, şi astfel oricărui server i se poate verifica cheia.

# 3 SSL în Python

Două module separate oferă posibilitatea folosirii SSL-ului în aplicaţiile utilizator: `socket` şi `OpenSSL`.

## 3.1 Socket SSL

Pentru a începe o sesiune SSL, mai întâi se crează un socket, apoi se crează un obiect SSL care comunică prin socket. Iată un exemplu:

```
#Basic SSL example - basicSSL.py

import socket, sys

def sendall( s, buf ):
   byteswritten = 0
   while byteswritten < len( buf ):
   byteswritten += s.write( buf[ byteswritten: ] )

print "Creating socket...",
s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
print "done."

print "Connecting to remote host...",
s.connect(("www.openssl.org", 443))
print "done."

print "Establishing SSL...",
ssl = socket.ssl( s )
print "done."
```

```
print "Requesting document...",
sendall( ssl, "GET / HTTP/1.0\r\n\r\n" )
print "done."

s.shutdown(1)

while 1:
   try:
      buf = ssl.read( 1024 )
   except socket.sslerror, err:
      if (err[0]) in [socket.SSL_ERROR_ZERO_RETURN, socket.SSL_ERROR_EOF]:
         break
      elif (err[0]) in [socket.SSL_ERROR_WANT_READ, socket.SSL_ERROR_WANT_WRITE]:
         continue
      raise
   if len( buf ) == 0:
      break
   sys.stdout.write( buf )

s.close()
```

Programul se conectează la serverul `www.openssl.org`, pe portul 443 (HTTPS), şi după aceea stabileşte o conexiune SSL.

Obiectele SSL oferă doar două metode: `read()` şi `write()`, corespunzătoare, în mare, funcţiilor `recv()` şi `send()` ale obiectelor socket. Funcţia `sendall()` trebuie prin urmare implementată de către utilizator.

### 3.2 OpenSSL

În continuare este prezentat un exemplu care foloseşte modulul OpenSSL:

```
#Basic OpenSSL Example - OpenSSL.py

import socket, sys
from OpenSSL import SSL

#Create SSL context object
ctx = SSL.Context( SSL.SSLv23_METHOD )

print "Creating socket...",
s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
print "done."
```

```
#Create SSL connection object
ssl = SSL.Connection( ctx, s )

print "Establishing SSL...",
ssl.connect(('www.openssl.org', 443))
print "done."

print "Requesting document...",
ssl.sendall( "GET / HTTP/1.0\r\n\r\n" )
print "done."

while 1:
   try:
      buf = ssl.recv( 4096 )
   except SSL.ZeroReturnError:
      break
   sys.stdout.write( buf )

ssl.close()
```

# 4 Verificarea certificatelor folosind OpenSSL

Exemplul precedent s-a conectat la un server SSL, dar fără a verifica autenticitatea acelui server.

## 4.1 Obţinerea certificatelor de autentificare

Primul pas pentru verificarea certificatelor este obţierea lor. Puteţi descărca un set de astfel de certificate de autentificare, ultima arhivă `tar.gz` de la `http://ftp.debian.org/pool/main/c/ca-certificates/`. Despachetaţi arhiva şi generaţi certificatele invocând utilitarul `make`. Apoi grupaţi fişierele rezultate într-unul singur, cu comanda `cat *.crt > filename`.

## 4.2 Verificare certificatelor

În continuare este prezentat un exemplu de program care se conectează la un site şi îi verifică certificatele.

```
#OpenSSL example with verification - osslverify.py

import socket, sys
from OpenSSL import SSL

#Grab the command-line parameters
```

```
cafile, host = sys.argv[1:]

def printx509( x509 ):
   """Display an X.509 certificate"""
   fields = { 'country_name': 'Country',
         'SP': 'State/Province',
         'L': 'Locality',
         'O': 'Organization',
         'OU': 'Organizational Unit',
         'CN': 'Common Name',
         'e-mail': 'E-Mail' }

   for field, desc in fields.items():
      try:
         print "%30s: %s" % (desc, getattr( x509, field ))
      except:
         pass

#Whether or not the certificate name has been verified
cnverified = 0

def verify( connection, certificate, errnum, depth, ok ):
   """Verify a given certificate"""
   global cnverified

   subject = certificate.get_subject()
   issuer = certificate.get_issuer()

   print "Certificate from:"
   printx509( subject )

   print "\nIssued by:"
   printx509( issuer )

   if not ok:
      #OpenSSL could not verify the digital signature.
      print "Could not verify certificate."
      return 0

   #Digital signature verified. Now make sure it's for the server
   #we connected to.
   if subject.CN == None or subject.CN.lower() != host.lower():
      print "Connected to %s, but got cert for %s" % \
            (host, subject.CN)
```

```
   else:
      cnverified = 1

   if depth == 0 and not cnverified:
      print "Could not verify server name; failing."
      return 0

   print "-" * 70
   return 1

ctx = SSL.Context( SSL.SSLv23_METHOD )
ctx.load_verify_locations( cafile )

#Set up the verification. Notive we pass the verify function to
#ctx.set_verify()
ctx.set_verify( SSL.VERIFY_PEER | SSL.VERIFY_FAIL_IF_NO_PEER_CERT, verify )

print "Creating socket...",
s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
print "done."

ssl = SSL.Connection( ctx, s )

print "Establishing SSL...",
ssl.connect((host, 443))
print "done."

print "Requesting document..."
ssl.sendall( "GET / HTTP/1.0\r\n\r\n" )
print "done."

while 1:
   try:
      buf = ssl.recv( 4096 )
   except SSL.ZeroReturnError:
      break
   sys.stdout.write( buf )

ssl.close()
```

Funcţia `printx509()` tipăreşte informaţiile conţinute de un certifcat, folosind diversele atribute ale acestuia (CN, OU etc) ca chei într-un dicţionar. La apelul funcţiei `verify()`, se va apela funcţia `printx509()` de două ori: o dată pentru subiect (certificatul propriu-zis), iar a doua oară pentru numele

organizaţiei care a eliberat certificatul.

Când `depth` devine zero înseamnă că funcţia `verify()` este apelată pentru ultima oară şi că certificatul a fost verificat şi este autentic.

În continuare sunt prezentate două exemple de folosire a programului de mai sus:

```
$ python2.4 osslverify.py certificates www.openssl.org
Creating socket... done.
Establishing SSL... done.
Requesting document...
Certificate from:
                   Common Name: www.openssl.org
                      Locality: None
                  Organization: The OpenSSL Project
            Organizational Unit: None

Issued by:
                   Common Name: OpenSSL CA
                      Locality: None
                  Organization: The OpenSSL Project
            Organizational Unit: Certificate Authority
Could not verify certificate.
Traceback (most recent call last):
  File "osslverify.py", line 80, in ?
    ssl.sendall( "GET / HTTP/1.0\r\n\r\n" )
OpenSSL.SSL.Error: [('SSL routines', 'SSL3_GET_SERVER_CERTIFICATE',
   'certificate verify failed')]
```

În acest caz se generează o eroare deoarece certificatul este semnat chiar de autoritatea OpenSSL, care nu este în lista certificatelor. Un browser de web ar fi avertizat utilizatorul şi l-ar fi interogat cu privire la continuarea încărcării paginii.

Iată şi un exemplu de verificare terminată cu succes:

```
$ python2.4 osslverify.py certificates www.yahoo.com
Creating socket... done.
Establishing SSL... done.
Requesting document...
Certificate from:
                   Common Name: None
                      Locality: None
                  Organization: Equifax
            Organizational Unit: Equifax Secure Certificate Authority
```

```
Issued by:
                   Common Name: None
                      Locality: None
                  Organization: Equifax
           Organizational Unit: Equifax Secure Certificate Authority
Connected to www.yahoo.com, but got cert for None
------------------------------------------------------------------------
Certificate from:
                   Common Name: www.yahoo.com
                      Locality: Santa Clara
                  Organization: Yahoo! Inc.
           Organizational Unit: Yahoo

Issued by:
                   Common Name: None
                      Locality: None
                  Organization: Equifax
           Organizational Unit: Equifax Secure Certificate Authority
------------------------------------------------------------------------
done.
HTTP/1.1 302 Found
Date: Sun, 07 May 2006 18:56:04 GMT
Location: http://www.yahoo.com/
Connection: close
Content-Type: text/html

The document has moved <A HREF="http://www.yahoo.com/">here</A>.<P>
<!-- p7.www.mud.yahoo.com uncompressed Sun May  7 11:56:04 PDT 2006 -->
```