Lecture 31

Security

April 13, 2005

SSL

- Secure Sockets Layer (Netscape 1994)
- A Platform independent, application independent protocol to secure TCP based applications
- Currently the most popular internet crypto-protocol

**History**

- 1994 version 1 of protocol designed
- 1994/1995 SSL v2.0 - implemented in Netscape
- 1995 SSL v3.0
- 1999 TLS <RFC 2246>

**SSL: Secure Socket Layer**

- SSL is a communication protocol:
  - provides higher level of abstraction than normal sockets:
    - server authentication
    - client authentication
    - encryption
  - SSL uses normal sockets as base and adds security to it
  - Many implementations of SSL, e.g., OpenSSL (quite difficult to use!)

**SSL**

- Establishing an SSL Connection
  - The client (browser) opens a connection to server port
  - Browser sends “client hello” message. Client hello message contains:
    - version of SSL browser uses
    - ciphers and data compression methods it supports
    - The Server responds with a “server hello” message. Server hello message contains:
      - session id
      - the chosen versions for ciphers and data compression methods.

- Establishing an SSL Connection (con’t.)
  - The server sends its certificate
  - used to authenticate server to client
  - Optionally the server may request client’s certificate
  - If requested, client will send its certificate of authentication
  - if client has no certificate then connection failure
  - Client sends a “ClientKeyExchange” message
  - symmetric session key chosen
  - digital envelope is created using server’s public key and contains the symmetric session key
SSL

- Establishing an SSL Connection (con't.)
  - Optionally, if client authentication is used the client will send a “CertificateVerify” message.
  - Server and client send “ChangeCipherSpec” message indicating they are ready to begin encrypted transmission.
  - Client and server send “Finished” messages to each other
    - These are a message digest of their entire conversation up to this point.
    - If the digests match then messages were received without interference.

SSL

- SSL security services:
  - server authentication
  - data encryption
  - client authentication (optional)
- Server authentication:
  - SSL-enabled browser includes public keys for trusted CAs.
  - Browser requests server certificate, issued by trusted CA.
  - Browser uses CA’s public key to extract server’s public key from certificate.
  - Visit your browser’s security menu to see its trusted CAs.

SSL

- Available in two versions
  - one version uses 128-bit symmetric keys (originally for use in USA only)
  - one version uses 40-bit keys (was used for international commerce)
  - latter complied with US export regulations (restrictions on the export of cryptographic technology)
  - most restrictions have been removed (after years of public controversy, series of lawsuits, ...)
  - Fast, efficient, accepted, and widely used
  - Integrated with browser (unlike SET)

OpenSSL

- A free fully featured Open-Source toolkit implementing SSL and TLS
  - Consists of two libs
    - libssl.a — SSL/TLS functionality
    - libcrypto.a — cryptography library
    - used to secure http traffic (https)
  - Libcrypto used in implementations of ssh, kerberos, and IPsec.

OpenSSL

- libssl.a:
  - SSL/TLS protocols
  - symmetric cryptographic operations
    - ciphers and message digests
  - asymmetric cryptographic operations
    - digital signatures, enveloping
  - PKI
    - extensive X.509 certificate support
- libcrypto.a:
  - symmetric ciphers
  - asymmetric ciphers
  - authentication and hashing
  - digital signatures
OpenSSL

- Start by initializing SSL:
  - get yourself a key pair, get it certified by a CA
  - initialize the library
  - create “SSL context”
  - load your key in the context, define the list of CAs whom you trust
- Create a normal TCP connection.
- Create an SSL connection (uses the TCP connection):
  - number of checks are realized anyway (e.g., the server key must be certified)
  - use the SSL connection to transfer data
- Close the SSL connection and the TCP socket.

stunnel: an SSL Tunnel

- stunnel allows you to use SSL without messing with the code:
  - http://www.stunnel.org/
  - here: version 3.23
- stunnel creates a daemon which converts insecure connections into SSL

stunnel Configuration

- stunnel can be used at one or both sides:
  - a non-SSL client with stunnel can interoperate with an SSL server
  - a non-SSL server with stunnel can interoperate with an SSL client
- You must configure each stunnel daemon:
  - which port to listen to
  - where to forward incoming connections
  - keys, certificates, etc.

Using stunnel at Server Side

- Generate a key for your server and self-sign it:
  openssl req -new -x509 -nodes -out server.pem -keyout server.pem
  chmod 0600 server.pem
- Start stunnel at server side:
  stunnel -f -d 9999 -r localhost:1234
  -f: run in foreground mode (otherwise it will fork and run in background)
  -d 9999: wait for incoming SSL connections on port 9999
  -r localhost:1234: forward incoming connections to localhost, on port 1234 (using TCP)
  -p server.pem: the server key/certificate can be found in server.pem

Using stunnel at Client Side

- Generate a key for your client and self-sign it:
  openssl req -new -x509 -nodes -out client.pem -keyout client.pem
  chmod 0600 client.pem
- Start stunnel at client side:
  stunnel -f -c 9998 -r myserver.cse.nd.edu:9999
  -f: run in foreground mode
  -c: run in client mode
  -r myserver.cse.nd.edu:9999: forward incoming connections to myserver, on port 9999 (using SSL)
  -p client.pem: the client key/certificate can be found in client.pem

Authentication with stunnel

- Did you notice that we used self-signed client and server keys?
  - keys were not certified by CA
  - each other’s identity cannot be checked
  - by default, stunnel does not check certificates
- You can set up authentication:
  1. create a CA
  2. create a key for each party which needs to prove its ID (usually the server): creates a key/certificate pair and a request for the CA to sign it
  3. sign each key/certificate pair with your CA
  4. run stunnel and ask it to check certificates
  5. you will be using the CA.sh script that comes with openssl:
     - Solaris: /usr/local/openssl/misc/CA.sh
     - Linux: /usr/lib/ssl/misc/CA.sh (often)
Creating a CA

CA.sh -newca
- you will be asked a certificate filename, type enter to create one
- you will be asked a passphrase, this is to protect the key of your CA (don’t forget it)
- you will be asked other information: company name, country, email, ...
- create a CA in directory demoCA:
  - CA’s public key is in demoCA/cacert.pem
  - CA’s private key is in demoCA/cakey.pem

Create a Server Key

Repeat the following instructions for each party which needs to authenticate itself (usually only the server).
- CA.sh -newreq
  - you will be asked a passphrase (do not use the same as for CA)
  - you will be asked the same questions when creating the CA
  - when asked for a Common Name, use fully qualified name of the machine (mymachine.cse.nd.edu)
- Request (with private key) is now in newreq.pem

Sign the Key with the CA

- Give your request to the CA and make it sign it: CA.sh -sign
  - you will be asked the CA’s passphrase
  - will create the signature file in newcert.pem
- Create your own certified key file (private key and certificate in same file):
  - cat newreq.pem newcert.pem > server-signed-key.pem
- Then you can use server-signed-key.pem with stunnel

Using stunnel with Server Authentication

- Start stunnel at server side with your new authenticated key:
  - stunnel -f -d 9999 -r localhost:1234 -p server-signed-key.pem
- Start stunnel at client side:
  - stunnel -c -v2 -f -d 9998 -r mymachine:9999 -p client.pem -A demoCA/cacert.pem
  - -v2: require authentication by the other party
  - -A demoCA/cacert.pem: indicates which CA I trust (public key)

SSL and SET

- SET (Secure Electronic Transactions) similar to SSL
- Developed by Visa, Mastercard, Netscape, and Microsoft
- Relies on cryptography and digital signatures
- Merchant never sees credit card information!
- Thus far, has not caught on much, due to costs involved in integrating SET into existing systems and lack of interest among consumers

Online Credit Card Use
Problems

- Security – Neither merchant nor consumer are authenticated. Merchant gets consumers credit card number for possible later misuse.
- Cost – for merchants, around 3.5% of purchase price plus transaction fee of 20-30 cents per transaction.
- Social equity – many people do not have access to credit cards (young adults, plus almost 100 million other adult Americans who cannot afford cards or are considered high risk)

Secure Electronic Transactions (SET)

- designed for payment-card transactions over Internet.
- provides security services among 3 players:
  - customer
  - merchant
  - merchant’s bank
- All must have certificates.
- Customer’s card number passed to merchant’s bank without merchant ever seeing number in plain text.
- prevents merchants from stealing, leaking payment card numbers.

SET

- SET uses SSL and PKI. Customer must have a SET enabled browser and merchant needs SET enabled server.
- Consumer’s credit card issuing bank issues a digital certificate (electronic wallet) with consumer’s public key and bank’s public key (signed with bank’s private key).
- Merchants get a similar digital certificate from bank.

Payment Request

- 1. Cardholder initiates request to merchant
- 2. Merchant
  - generates response
  - digitally signs response with merchant’s private key
  - send response to cardholder along with merchant’s signature certificate and key exchange certificate
- 3. Cardholder receives and verifies both certificates and merchant’s digital signature
- 4. Cardholder creates ordering and payment instructions
- 5. Cardholder
  - generates digital signatures on order and payment instructions
  - encrypts payment instructions with random symmetric key
  - encrypts order with random symmetric key and cardholder account information with payment server’s public key
Payment Request

6. Cardholder sends to merchant
- order information
- encrypted payment instructions
- encrypted symmetric key
- signature certificate
7. Merchant verifies cardholder’s signature certificate and digital signature on order
8. Merchant forwards payment info and encrypted symmetric key to payment gateway

Payment Authorization

1. Merchant creates digitally signed authorization request and encrypts it with a randomly generated symmetric key
2. Merchant encrypts symmetric key with payment gateway’s key exchange (public) key
3. Merchant sends to payment gateway
- encrypted authorization request
- encrypted symmetric key
- encrypted payment info from cardholder
- signature certificate of cardholder
- signature certificate of merchant
- key exchange certificate of merchant

Payment Authorization

4. Payment gateway
- verifies the two merchant certificates
- decrypts and verifies the digital signature of the authorization request
5. Payment gateway
- verifies cardholder certificate
- decrypts and verifies the digital signature of the payment information
- ensures consistency between merchant’s authorization request and cardholder’s payment information (e.g., amounts match)

Payment Authorization

6. Payment gateways sends authorization request through its financial network to cardholder’s financial institution
7. If approved, payment gateway
- creates a digitally signed authorization response
- encrypts it with a randomly generated symmetric key
- encrypts symmetric key with with merchant’s key exchange (public) key

Payment Authorization

8. Payment gateway sends to merchant
- encrypted authorization response
- encrypted symmetric key
- its signature certificate
9. Merchant
- verifies payment gateway’s signature certificate
- verifies digital signature of response
- stores response as proof of authorization
SET Comments

- To its credit, SET
  - is well designed
  - is based upon sound PKC principles
  - is well documented
  - encourages anyone/everyone to implement, could be added to browsers/servers

SET Comments

- SET has three weaknesses
  - implementations not as clean as the specification, so interoperability is an issue
  - given the complexity, not surprising that in initial trials the transactions were slow (30 seconds)
  - Secure Sockets Layer has taken the lead by integrating security directly into the browser