Fermilab

Opreated by Fermi Research Alliance, LLC for the U.S. Department of Enrgy Office of Science.

Kerberos, Certificates, and Proxies

Marc W. Mengel

2015-09-18

[<] [>]

Computer Security Systems

- Basic Issues
- Principles
- Symmetric Encryption
  - Kerberos
- Asymmetric Encryption
  - GPG
  - Certificates
  - Proxies

Basic Issues:

- Authentication
  - who are you?
- Authorization
  - what can you do?

Principles:

- Security Token
  - Result of Authentication
  - Used for Authorization
  - Not modifiable by you

Examples:

Physical

- Fermi ID
- Credit Card
- Drivers License

Examples:

- Electronic
    - Unix User-id
    - Kerberos TGT
    - SSL Certificate
    - GRID Proxy
    - PGP/GPG key
    - SSH User Key

[<] [>]

Symmetric/Shared Key Encryption

- One key-based function crypt()
  - x = crypt(crypt(x))
- People who share a key can communicate secretly.
- lots of algorithms:
  - blowfish
  - des
  - 3des
  - rot13

Kerberos

- Authentication system
- symmetric keys
- Key server
  - knows everyone's keys
  - kept very secure

Kerberos: example

- Amy wants to talk to Bert:
    - sends keyserver:
      crypt_A(request: Bert)
    - keyserver sends back
      crypt_A(use: crypt_S
      intro: crypt_B(this_is: Amy, use: crypt_S)
)
    - Amy sends Bert
      crypt_B(this is Amy, use crypt_S)
    - Bert and Amy talk with crypt_S

Kerberos: Important Details

- Block of Really Random Bits
  makes key guessing hard
- Timestamps
  prevent "replay" attacks
- ticket_granting_ticket -- message from keyserver with short-term key to use instead of your regular one

Exercise:

- setup:
  - Groups of 4
    2 people who want to send a message
    1 keyserver
    1 snooper
  - Envelopes for Encryption
    name on envelope means encrypted
    in that key
- play kerberos -- send a key request contact other person, etc.
- If you need to see 1000 envelopes in a given key to figure out the key, how many messages can you send?

[<] [>]

Asymmetric/Public Key systems

- Intro
  - Two key-based-functions priv, pub
    x = priv(pub(x))
    x = pub(priv(x))
  - used in pgp, gpg, etc.
  - also for certificates, proxies

[<] [>]

Sending Messages:

- combined with shared keys
- message sent to a,b,c is:
  - a: pub_a(shared_key)
  - b: pub_b(shared_key)
  - c: pub_c(shared_key)
  - shared_key(message)

[<] [>]

Signing Messages:

- combined with hash/checksum
- signature = priv(hash(message))
- checking person sees if:
- hash(message) == pub(signature)
  == pub(priv(hash(message))
  == hash(message)

Signing Keys:

- Get someone to make a signature of your public key to "prove" it is yours, and not someone else's.
- Idea behind certificate systems -- trusted Certificate Authorities sign public keys.
- Allows "web of trust" setups (i.e. CAcert)

[<] [>]

RSA: Fun with Prime Numbers

- key pair is based on p,q,e,d
  - prime(p)
  - prime(q),
  - gcd(e,d)==1
  - (e * d) % ((p-1)*(q-1)) == 1
- n = p*q
- pub(x): (x ** e) % (n)
- priv(x): (x ** d) % (n)
- can't encode numbers bigger than n -- slice into blocks.

RSA: Cont.

- Public key is pair of integers (n,e).
- Private key is pair of integers (n,d).
- "breaking" RSA consists of factoring n, so you pick Really Big Primes for p & q to make it hard.

Exercise:

- is (p = 7, q = 13, e = 5, d = 1037) a valid RSA key tuple?
- Encode/decode 66, 77, via RSA with key pair
- (91,5) (91,1037)

- bc:
    - (66 ^ 5) % 91
    - (40 ^ 1037) % 91
- pub(66) -> 40
- priv(40) -> 66

Certificates

More Fun with Signatures

- Certificate Authority(CA): Place with a public key
- Secondary CA: CA whose key is signed by another CA

- Certificate:
  - public_key,
  - CA1
  - signature_CA1(public_key)
  - *CA1_key*
  - *CA2*
  - *signature_CA2(CA1_key)*

[<] [>]

ISO Certificates: x509

- File formats
- Naming conventions (x500)
- Mechanisms
  - Certificate Requests
  - Certificates
  - Expiration
  - Revocation Lists

Homework/Exercise

Look at the openssl cookbook

- Setup your own CA
- Make a CSR, sign it
- install Cert in apache
- install CA key in browser
- revoke certificate

Proxies

Proxies are special certificate/key bundles, with:

- a short lifetime (i.e. max 24 hours)
- a private key/public key pair
- one or more Attribute Certificates from a VOMS granting VO Roles/membership
- whole thing is digitally signed by *you*
- a copy of your personal cert, signed by CA

Proxies (cont)

Important bits about Proxies

- It contains a key, so it's a 'bearer bond'
- checking one requires the signature chain to the CA so using with "curl" etc. is tricky, permutations needed like

```
curl \
--cert proxy \
--cacert proxy \
--key proxy \
--capath /etc/grid-security/certificates \
...
```

- works differently depending on openssl vs. nsl libraries...
- See Funny Curly THings

Grid vs Proxy

How to grid tools use Proxies?

- grid tools are Web Services that accept proxy certificates.
- Generall map proxies to local accounts (via GUMS/Gridmap files)
- Here at Fermi we used to map:

```
fermilab:/fermilab/EXP/Role=Production -> EXPpro account
fermilab:/fermilab/EXP/Role=Analysis -> EXPana account
```

- We now map Role=Analysis to local user account
- Future: CILogon cert Distinguished Names -- put the extra CN=UID:whoever bits in the DN, too?

Thus proxies are Only Useful if they have VO attributes.

Proxy Issues

- multi stage process:
  - get identifying certificate
    (which can involve kerberos auth first!)
  - get Attribute Certs from VOMS(es)
  - build proxy
- voms-proxy-init defaults/specify many arguments
- garbled proxy cert (0.1% failure rate?)
- finding proxy / env vars
  - X509_USER_PROXY
  - X509_USER_CERT
  - X509_USER_KEY