Lecture 10 - Authentication

CMPSC 443 - Spring 2012
Introduction to Computer and Network Security
Professor Jaeger
www.cse.psu.edu/~tjaeger/cse443-s12/
Kerberos: What to know

1) Alice → Trent : \{Alice + Bob + rand_1\}
2) Trent → Alice : \{Alice+Bob+rand_1+K_{AB}+\{Alice+K_{AB}\}K_{BT}\}K_{AT}
3) Alice → Bob : \{Alice + K_{AB}\}K_{BT}
4) Bob → Alice : \{rand_2\}K_{AB}
5) Alice → Bob : \{rand_2 - 1\}K_{AB}

- Replaced by single “authenticator” message \{time\}K_{AB}

• Kerberos Properties
  - Initial Goals: secure communication, mutual authentication
  - Extra Goal: single signon
  - Compare result to SSH (and PKI today)

• Deployment of Needham-Schroeder
  - Two-phase protocol
  - Limited to single administrative domain
Public Key Authentication

• Public Key Cryptography is the answer
  – easy to distribute the public key
  – never give the private key to anyone else
  – key agreement is easy (sans Needham-Schoeder)
  – keys can be global

• While PK is used, not as broadly as expected

• Requires a significant infrastructure
  – Global systems are difficult (impossible) to build
Public Key Infrastructure

• System to “securely distribute public keys”
  – Q: Why is that hard?

• Terminology:
  – Alice signs a certificate for Bob’s name and key
    • Alice is **issuer**, and Bob is **subject**
  – Alice wants to find a path to Bob’s key
    • Alice is **verifier**, and Bob is **target**
  – Anything that has a public key is a **principal**
  – Anything trusted to sign certificates is a **trust anchor**
    • Its certificate is a **root certificate**
What is a certificate?

• A certificate …
  – … makes an association between a user identity/job/attribute and a private key
  – … contains public key information \{e,n\}
  – … has a validity period
  – … is signed by some certificate authority (CA)

• Issued by CA for some purpose
  – Verisign is in the business of issuing certificates
  – People trust Verisign to vet identity
Why do I trust the certificate?

• A collections of “root” CA certificates
  – … baked into your browser
  – … vetted by the browser manufacturer
  – … supposedly closely guarded (yeah, right)
• Root certificates used to validate certificate
  – Vouches for certificate’s authenticity
• Who is “Bob Jones?” …
What is a PKI?

- Rooted tree of CAs
- Cascading issuance
  - Any CA can issue cert
  - CAs issue certs for children

```
Root
  ├── CA1
  │    ├── CA11
  │    │    ├── Cert11a
  │    │    └── Cert11b
  │    └── CA12
  │        └── Cert11c
  └── CA1n
      └── ...
  ├── CA2
  │    └── CA21
  │         └── ...
  └── CA3
      └── CA22
          └── ...
```
Certificate Validation
PKI and Revocation

• Certificate may be revoked before expiration
  – Lost private key
  – Compromised
  – Owner no longer authorized

• Revocation is hard …
  – The “anti-matter” problem
  – Verifiers need to check revocation state
    • Loses the advantage of off-line verification
  – Revocation state must be authenticated
PKI Challenges

• Must trust a CA
  – Which one?
  – What is it trusted to do?

• Key storage
  – Who can access my key?
  – Similar problem for Kerberos, SSH, etc.

• Certificate bindings must be correct
  – Which John Smith is this?
  – Who authorizes attributes in a certificate?
  – How long are these value valid?
  – What process is used to verify the key holder?
Pretty Good Privacy

• Alternative infrastructure for public key
  – Peer-to-Peer approach
  – E.g., for email

• Key management is manual
  – Public key exchange between peers
  – Add public key to personal ‘keyring’
  – Can authenticate messages from these parties

• Used mainly by computer security types
  – Johnny can’t encrypt
  – GNU Privacy Guard
Authentication Architecture

Cients of These Programs

- Remote Service (sshd, telnet)
- Local Service (su, login)
- Application Service (ftp, httpd)
Authentication Architecture

Clients of These Programs

- Remote Service (sshd, telnet)
- Local Service (su, login)
- Application Service (ftp, httpd)

Common Authentication Architecture
Pluggable Authentication Modules

- Centralized authentication service for Linux/Solaris

- Advantages
  - Provides a common authentication scheme that can be used with a wide variety of applications.
  - Allows a large amount of flexibility and control over authentication for both the system administrator and application developer.
  - Allows application developers to develop programs without creating their own authentication scheme.

- PAM-ified application
  - Uses PAM authentication technique and config
  - Receives identity
  - May be entrusted to forward identity to system
Clients of These Programs

- PAM Remote Service
- PAM Local Service
- PAM Application Service

Authentication Mechanism (may be different for each service)
PAM Concepts

• Module Interface
  – Auth: authentication
  – Account: management + authorization
    • Use service; password expire
  – Password: set and verify passwords
  – Session: configure session
    • E.g., mount home directory

• One module may provide all
  – pam_stack.so for each interface

• Modules may be ‘stacked’
  – Multiple support same interface
  – Required and optional session interfaces modules
PAM Usage

• PAMify an application
  – Must be able to modify the application code
  – Build with PAM libraries (libpam, libpam-misc, ...)

• Authenticate first
  – Build pam_handle_t data structure
  – Call pam_authenticate (calls PAM module for authenticate)
    • Use pam_get_item to get authenticated identity

• Example
  – Call pam_authenticate (uses module specified in config)
  – PAM gets username, password (or whatever)
  – Returns PAM_SUCCESS
  – Use pam_get_item to get the actual identity
PAM Usage (con’t)

• Session management
  – pam_setcred() before open session
    • application-specific credentials to PAM
  – pam_open_session()
  – pam_close_session()
  – based on module specified in config

• Account management
  – pam_acct_mgmt()
  – based on module specified in config

• Password management
  – pam_chauthtok()
  – based on module specified in config

• Where is responsibility for correct authentication?
pam_unix.so

- Auth:
  - Authentication
  - `pam_authenticate()` and `pam_setcred()` (RPC credentials)
- Session
  - Session logging
- Account
  - Check that password has not expired
- Password
  - Password update, includes cracklib to check strength
PAM Policies

- Config files: /etc/pam.d/
  - For each PAMified application

- su -- /etc/pam.d/su or /etc/pam.conf

```
<module interface>  <control flag>  <module path>  <module arguments>

#%PAM-1.0
auth       required     /lib/security/$ISA/pam_stack.so service=system-auth
account    required     /lib/security/$ISA/pam_stack.so service=system-auth
password   required     /lib/security/$ISA/pam_stack.so service=system-auth
session    required     /lib/security/$ISA/pam_stack.so service=system-auth
session    optional     /lib/security/$ISA/pam_xauth.so
```
Control Flags

• Required
  – Must be successful
  – Notify after all modules on interface run

• Requisite
  – Must be successful
  – Notify immediately

• Sufficient
  – Result is ignored if failed
  – Pass if succeeds and no previous modules failed

• Optional
  – Result is ignored
  – Must pass if no other modules
Modules and Arguments

- Modules are in
  - /lib/security/

- Arguments are module-specific
  - For pam_stack
    - auth sufficient … service=x509-auth
    - auth required … service=system-auth
      - Tries using x.509; password is backup plan
  - Could apply other authentication techniques
    - Kerberos, biometrics, etc.
Take Away

• Authentication Systems
  – A variety of ways to authenticate a principal
  – And generate a session key for secure communication

• Use limited by trust
  – Trust in KDC administration: Kerberos
  – Trust in machine-public mapping: SSH
  – Trust in public key-identity mapping: PKIs
  – Trust in public key storage

• PAM enables integration of authentication with applications