The Two “Auth-” Operations

- **Authentication**
  - “Process of accepting credentials from a user and validating those credentials against some authority”
  - The result is an authenticated identity

- **Authorization**
  - “Process of determining whether the authenticated identity has access to a given resource”

- Both steps follow this order and both are essential!

What Can Go Wrong?

- **Authentication breaks if:**
  - Credentials are forged
  - Authority is subverted
  - Validating function is replaced

- **Authorization breaks if:**
  - Authentication identity is forged
  - Access matrix is tampered with
  - Matrix lookup function is replaced

- **Lesson:** Security needs to be provisioned on each step!

Types of Authentication

- **Server authentication**
  - Necessary in e-commerce
  - Achieved via:
    - X.509 certificates, signed by known certificate authorities (CA)
    - Digital signatures using public/private key encryption

- **Client authentication**
  - Necessary in e-commerce
  - Majority of clients typically do not use X.509 certificates, or public/private key pairs
    - How many of you use one of these methods for authentication?

Client Authentication Methods

- **Client certificates**
  - No incentive for clients to have one ⇒ not widely deployed

- **Digital signatures**
  - No PKI yet ⇒ hard to safely distribute public keys

- **Passwords**
  - Most primitive, pervasive method
  - Easy to use, easy to crack: passwords are guessable (or users forget)
    - Copy-and-store-in-wallet - works well in practice with random passwords
    - Visual passwords - random art; a drawing in lieu of a word
    - S/Key protocol - changing passwords on every communication
    - Smart cards - store random password safely; PIN for theft protection; activated only by a special card reader; European invention

How to Evaluate Proposed Approaches?

Ask:
1. **What problem is the approach trying to solve?**
2. **What are the ways in which the approach can fail (including, be deliberately made to fail)?**
3. **Given the ways the approach can fail, does it really solve the problem at hand?**
4. **What are the costs (financial and otherwise) of deploying a real implementation of the approach?**
5. **Given the failure conditions and costs, is it worthwhile?**
Client Authentication Methods

- **Biometrics**
  - Unique, inherently tied to the individual
  - But:
    - **Fingerprinting** - non-permanent, could be tampered with
    - **Retina scans** - non-permanent, invasive, even dangerous
    - **Voice recognition** - high false positive and false negative rate, recordable
    - **DNA analysis** - slow, extremely invasive, may be non-permanent
    - **Normal Signature** - varies widely (high false negative rate), more appropriate for non-repudiation that authentication
    - **Typing Timing** - Local startup. Test timing & rhythm when typing password

Client Authentication on the Web

- What assumptions / constraints does the Web environment imply?
- Which of the above methods are unsuitable for authentication on the Web?
- What remains?

Motivation

- Growing need for personalized, access-controlled Web-based services
  - E.g.: nytimes.com, myuw.washington.edu, hotmail.com
- Some popular authentication mechanisms not suitable for the Web environment
  - Designed for long-running connections
  - Involve expensive computations - public/private key crypto
  - Authentication identities can be replayed - biometrics
- Developers lack proper background in security
- Result: Proliferation of home-grown weak authentication schemes

Limitations on Web Authentication Schemes

- Must use only widely deployed, portable and lightweight technologies
  - No smart cards or client certificates; JavaScript may be ok
- Must require minimum user involvement
  - No password re-typing or perpetual dialog boxes
- Must not unduly overload servers with expensive computations
  - No public-key crypto; cryptographic hashes are fine
- Must store client state in a very limited space
  - E.g.: cookies on the client, (maybe) a database on server

Not All Web Authentication Schemes Are Created Equal

- Designs differ depending on:
  - **Type of service**
    - General subscription
    - Online newspapers and libraries
    - User customization
    - Online identities, per-user content filtering
  - **Security needs**
    - Sensitivity of the client data
      - Store data on server and put an index to it in a client cookie
      - Load tolerance on the server
    - Delicate tradeoff with clients’ need for strong protection

Threat Model: What Attacks Do We Fear?

- Forging* an authentication token for
  - A random user (a.k.a. existential forgery)
    - Useful for free access to subscription services
  - A chosen user (a.k.a. selective forgery)
    - Allows access to data for any selected user
  - All users (a.k.a. total break)
    - Allows forging tokens for all users at any time
- * forging ≠ replay attack
Threat Model: What Adversaries Do We Fear?

Active Adversary

Eavesdropping Adversary

Interrogative Adversary
- Queries the server (adaptively, based on previously seen data)
- Creates new accounts (assuming no out-of-bound throttling)
- Uses publicly known information
- Records traffic between users and the server
- Replays selected captured messages

- Modifies / injects traffic between users and the server
- Mounts man-in-the-middle attack

Resolution: Viable schemes must at least protect against interrogative adversaries!

Hints for Designing Client Authentication Schemes

Disclaimer:
Hints are useful, but following them is neither necessary, nor sufficient for security

Hints: Use Cryptography Appropriately

- Using crypto is inescapable if you want to protect from adversaries!
- Hint #1: Assess your needs for protection
  - Tradeoffs between usability and complexity
- Hint #2: Choose a “tried and true” existing scheme
  - Home-grown schemes are almost always trivial to break

If you absolutely must design your own scheme:
- Hint #3: Think twice! Ask those who know better!
- Hint #4: Have it reviewed by security experts
  - Announcing it loudly is good but not sufficient
- Hint #5: Keep the scheme simple
  - Makes it easier to analyze for security
- Hint #6a: Do not rely on the secrecy of the protocol
  - Gives you false sense of security until someone figures it out
- Hint #6b: Instead, rely on the secrecy of keys

Hints: Use Cryptography Appropriately

- Hint #7: Understand the properties and details of crypto primitives you use
  - Many provide some assurances, but not other (e.g., SSL)
  - Many make fine-print assumptions
    - UNIX crypt() hash function truncates input beyond 8 characters
- Hint #8: Avoid composing security schemes
  - May weaken the composite, even if secure in isolation
    - E.g., using the same secret key for multiple purposes

Hints: Use Cryptography Appropriately

- Users don’t want passwords
  - Tradeoff between usability and security
  - Users tend to pick poor (easy) passwords
    - Do not suggest ideas - they will blindly follow it
- Users tend to reuse passwords across many sites
  - How many different passwords do you use?
  - How many of them do you commit to memory?
  - How many of them do you have written somewhere (as a backup)?
- Compromising a password leads to impersonation

Status on Using Passwords
Hints: Protect Passwords

• Hint #9: Prohibit easy-to-guess passwords
  • Otherwise: an easy prey for dictionary attacks
  • Change periodically, enforce non-similarity, minimum password length, special characters
  • Giving out (random) passwords may turn off users

• Hint #10: Never reveal a user’s password
  • User knows it, everyone else has no reason to ask for it
  • Keep passwords always encrypted in transfer
    • Login over SSL for confidentiality of password exchange
    • Avoid unnecessary password transfers
    • Give out and use (temporary) client authentication tokens instead

Hints: Protect Passwords

• Hint #11: Redo authentication before security-sensitive operations
  • E.g.: changing passwords
  • Avoids attacks through replayed authentication tokens

Hints: Handle Authentication Tokens Wisely

• Hint #12: Avoid predictable authentication tokens
  • E.g.: publicly available info, sequential ID numbers, etc.

• Hint #13: Protect tokens from tampering
  • Tokens may contain sensitive user info
  • Use only strong cryptographic hash functions (e.g., no CRC)
  • Use a keyed message digest (e.g., MAC, no MD5)

• Hint #14: If combining multiple data into a token, separate components unambiguously
  • Avoids a splicing attack:
    • “Alice” • “213” • “Bob” == “Alice2” • “13” • “Bob”

Hints: Handle Authentication Tokens Wisely

• Hint #15: Encrypt tokens
  • For tokens stored in cookies and sent over SSL, set Secure flag
  • Prevents eavesdroppers from capturing and replaying tokens

• Hint #16: Do not include a token as part of a URL
  • Otherwise, token may leak through plaintext channels
    • E.g.: cross-site scripting attack using the HTTP Referer field

• Hint #17: Avoid using persistent cookies
  • If cookie (file) is leaked, attacker can impersonate user
  • Can users defend against this threat (the authentication scheme designer may have been negligent)?

Hints: Handle Authentication Tokens Wisely

• Hint #18: Make authentication tokens expire:
  • Store a tamper-resistant timestamp in cookie, or keep token expiration time on the server
    • Limits the potential damage in case a token leaks out

• Hint #19: Do not trust the client…
  • … to enforce token expiration (manipulating a cookie is easy)
  • … (in general) for anything that the client can possibly forge

• Hint #20: To prevent replays of leaked tokens
  • Keep tokens confidential and mint new ones after each use
  • Bind tokens to network addresses
    • But DHCP users’ tokens may expire prematurely

Sample Authentication Scheme

• Goals
  • Statelessly verify authenticity of request and its contents
  • Explicitly control lifetime of token
  • Portability

• Design choice
  • Authentication cookies
    • Anyone with a valid cookie has access to protected server content

• Claim
  • Secure against an interrogative adversary
    • If layered over SSL with server authentication, secure against an active adversary
**Cookie Basics**

- HTTP is a stateless protocol
- Client IDs generated by server, stored on client
- Sent back to server with subsequent requests
- Cookie attributes:
  - Data - used to uniquely identify client
  - Domain - cookie only applies to this server domain
  - Path - server path
  - Secure flag - should cookie data be encrypted?
  - Expiration - current session or physical time

**Suggested Cookie Structure**

```
exp=t&data=s&digest=MAC_k(exp=t&data=s)
```

- `exp` → expiration time (seconds past 1970 GMT)
- `s` → data, associated with the client
- `k` → server secret key
- `MAC` → strong cryptographic hash function

```
HMAC_k(M) ::= H(k ⊕ 0x5c • H(k ⊕ 0x36 • M))
```

where \( H \in \{SHA1, MD5\} \), \( M \) is the message

**Dissecting the Scheme**

- **Expiration time:**
  - Avoids keeping server state
  - Tradeoff between potential damage and frequent reauthentication (security vs. usability)
  - Should users be allowed to control it?
- **Data:**
  - Sensitive data should not be stored here
  - If needed, store cryptographically random session ID, while keeping important data on server
  - Balance between respecting users’ privacy and saving server resources
  - Likely to be biased in favor of the latter

**Dissecting the Scheme**

- **Key:**
  - Recommended length is twice that of block encryption ciphers (~160 bits or more)
  - Fends off birthday attacks

**Dissecting the Scheme**

- **Strengths:**
  - Simplicity
  - Authenticating clients:
    - Requires \( O(1) \) server state (for the key)
    - Takes \( O(1) \) time
    - Would depend on number of clients if server state were kept
  - Easier to deploy multiserver systems
    - No need for dynamically shared data between servers

**Dissecting the Scheme**

- **Weaknesses:**
  - Server is vulnerable against colluding clients
    - Clients more likely to share temporary tokens than passwords
    - How many other people’s passwords do you know?
  - No mechanism for selective secure token revocation
    - Unnecessary for short sessions
      - Separation of policy and mechanism?
    - If needed, keep session status on server
      - Yahoo does it
      - But, allows simultaneous revocation of all tokens
      - By changing the secret server key
Security Analysis

Strength of authentication scheme depends on:
- Strength of MAC function
- Secrecy of server key
- Strength of server key and frequency of changing it
  - Longer keys adversely affect performance of hash functions
- Strength of client passwords against guessing and dictionary attacks

Performance Factors

- HMAC-SHA1
  - 1.2 ms / request
  - Runs on small chunks of data
- SSL
  - 90 ms / request
  - Runs on the entire HTTP stream
  - New connections are costly to setup, session resumption helps

Other Authentication Schemes

- HTTP Basic Authentication
  - Sends username and password repeatedly in cleartext
  - Falls prey to eavesdropping adversaries
  - dsniff - automated tool for sniffing authentication exchanges
- HTTP Digest Authentication
  - Encrypts username and password before transmitting
  - Little client support yet
- SSL
  - Requires public-key crypto in X.509 certificates
  - No global PKI → no wide support for client certificates
  - Involves heavyweight operations

Conclusions

- No single authentication scheme can effectively and efficiently meet the requirements of all Web sites and Web clients
- There are clear guidelines (but no standards yet) for designing secure authentication schemes

Open Issues

- What can end users do to protect themselves?
  - Those who can provide a solution (i.e., vendors) have no incentive to do so.
  - Those who really care about finding a solution (i.e., clients) cannot create one.
- Should there be a standard for authentication protocols? What factors play against establishing such a standard?
- Would you trust a centralized authentication service (such as Microsoft Passport) with your data? A step in which direction is this - forward or backward?

SPAM

- Problem
  - Zero marginal cost of sending an email
- Solutions
  - Machine learning client to detect spam
  - Brightmail
    - Dummy accounts
    - Correlate SPAM messages
    - Supply fingerprint to enterprise customers
  - Client refuses messages from unknown senders, until
    - They respond to a Turing test query
    - They execute a computationally expensive applet
    - Micropayment
Link Spam

- **Keyword / Meta tag stuffing**
  - Linguistic spoofing
- **Multiple titles**
- **Tiny fonts**
- **Invisible text**
  - `<body bgcolor="FFFFFF">`
  - `<font color="#FFFFFF" size ="1">Your text here</font>`
  - Problem: takes up space. Size=1? Bottom?
- **Doorway / jump pages**
  - Fast meta refresh
- **Cloaking ~ Code swapping**
- **Pagerank spoofing (Link newtworks)**

Robots

- **Threat: automatic creation of accounts**
  - Paypal
  - Storage associated: Hotmail, Yahoo communities…
  - Adbots in chat rooms
  - Online polls
- **Solutions**
  - Turing tests
  - Distorted speech recognition
  - Overlayed distorted text recognition
  - CAPTCHA
  - Automated public Turing test to tell computers and humans apart
  - http://www.captcha.net/

Gimpy: Type 3 words

Mori & Malik (UCB) program solving ez-gimpy with accuracy 83%

Semantic Tests

ESP Game

Viruses

- **Defn**
  - Requires human action to spread
  - Infects most files on local computer
  - Doesn’t automatically spread across network
  - Carries payload (destructive or annoying messages)
- **Common Modus Operandi**
  - Macro attached to office document
- **Solutions**
  - Fingerprint based (to detect viruses)
  - Application checksums (to detect tampering)
Worms

- **Defn**
  - Automatically spreads to other systems
- **Modus Operandi**
  - Protocol worms
  - Hybrid virus / worms
- **Solutions**