Security

G54ACC – IP and Up

Lecture 9

Recap

- Authentication *vs*. Identification *vs*. Encryption
- Use of something you know/have/are
- Have previously encountered middleboxes, e.g., firewalls
- Network security involves techniques such as traffic analysis, anomaly detection
- General principles of one time pads, symmetric/asymmetric encryption, hashing

Contents

- Common Requirements
- OpenID *vs*. OAuth
- OpenID
- OAuth
- SSL/TLS/HTTPS

Contents

- Common Requirements
  - Network security
  - Security of the network
  - Techniques
- OpenID *vs*. OAuth
- OpenID
- Oauth
- SSL/TLS/HTTPS

Common Requirements

- Identification: naming a principal
- Authentication: proving your identity
  - Three-factor: know, possess, are (biometrics)
    - E.g., chip and PIN and retina/fingerprint scan
  - Two-factor: know, possess (far more common)
    - E.g., chip and PIN
- Authorization: proving you're allowed
- Confidentiality: hiding what you're doing
- Defence in depth: mitigation at all levels
  - Perfect security generally too expensive

Network Security

- Identification
  - Remote party states who they are
- Authentication
  - Prove remote party's identity
- Authorization
  - Control (authenticated) access to service
- Integrity
  - Prove a message has not been tampered with
- Confidentiality
  - Prevent third-party observation of sensitive data

Security of the Network

- An inherent conflict!
  - Controlling use of your network (but by what?)
  - How to authorise the anonymous, unknown user?
- Intrusion detection
  - Detect if host (or router) is hacked
- (Distributed) Denial of Service, BotNets
  - Handle many hacked hosts used to overload service
  - May need to distinguish from transient popularity...

Techniques

- Encryption
  - One time pads
  - Private keys (symmetric – same key to encrypt/decrypt)
  - Public keys (asymmetric – different encrypt/decrypt keys)
- Hashing: result of a one-way function
  - $\text{MAC}_k(M) = h(k, M)$
  - $\text{HMAC}_k(M) = h(k \oplus A, h(k \oplus B, M))$
  - Setting `A,B=0x36,0x5C` makes collision finding harder
- Traffic analysis
  - Communication patterns may give you away
  - E.g., Hacked hosts often start using network oddly

Contents

- Common Requirements
- OpenID *vs*. OAuth
- OpenID
- Oauth
- SSL/TLS/HTTPS

OpenID *vs*. OAuth

- OpenID, http://openid.net/
  - Allows you to use one identity with multiple sites
- OAuth, http://oauth.net/
  - Authenticates third-party access to your data

Contents

- Common Requirements
- OpenID *vs*. OAuth
- OpenID
  - Dumb mode
  - Smart mode
- Oauth
- SSL/TLS/HTTPS

OpenID

- Identity
  - Not trust. Not authentication. Not authorization.
  - Enables a site to use a third-party to verify identity of a user
- Dumb (stateless) *vs*. Smart (stateful) modes
  - Simpler code on the consumer side
  - More computational and network resources used
  - (Optimization on dumb mode)
  - http://wiki.openid.net/Introduction

Dumb Mode

- Actors
  - Alice (user/you), Bob (relying party/Slashdot), Carol (provider/myOpenID.com)
- Phase 1:
  - Alice types in her identity URL with Carol to Bob's site
  - Bob GETs Carol's server URL from that page
  - Bob redirects Alice to Carol's URL adding params
    - identity, return_to, nonce
- Phase 2:
  - Carol verifies Alice *is* Alice. Somehow.
  - Carol sends Alice back adding params
    - assoc_handle (opaque handle), sig (base64 HMAC signature)
- Phase 3:
  - Bob contacts Carol POSTing all the parameters so far
  - Carol computes sig' using the secret pointed to be assoc_handle
  - If sig' == sig then tell Bob Alice *is* the Alice Carol thinks she is. Else fail.

Smart Mode

- Same as dumb mode *except*
  - Upon the first invocation, Bob and Carol setup a *shared secret*
  - In phase 2, Carol uses the shared secret as before
  - But now, in phase 3, Bob can do the check himself
- Reduces network latencies
- Reduces work done by Carol
- Shared secrets usually have a limited lifetime

Contents

- Common Requirements
- OpenID *vs*. OAuth
- OpenID
- OAuth
  - Basic flow
  - Evolution
- SSL/TLS/HTTPS

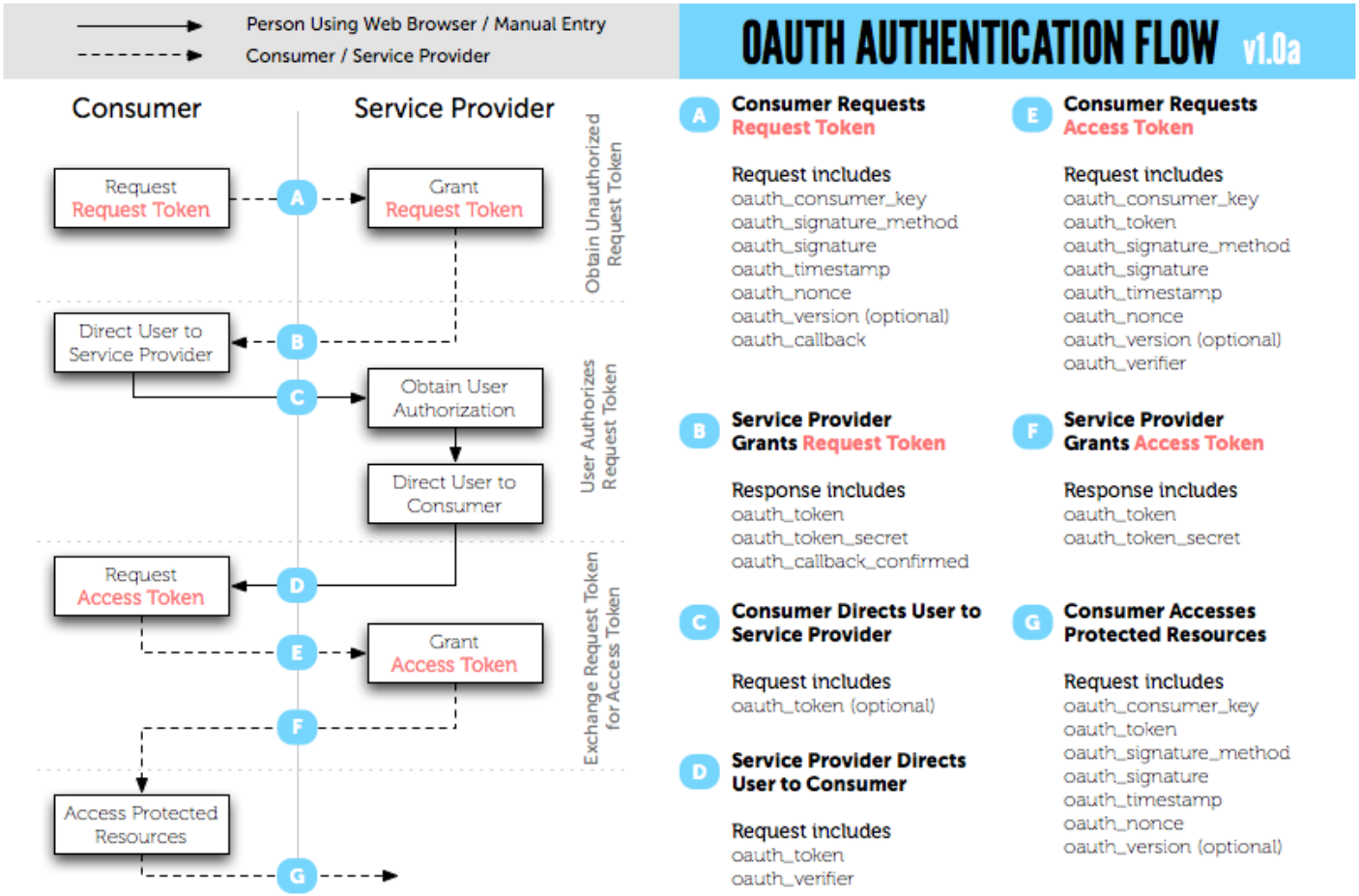
OAuth

- Appears similar to OpenID on the surface
  - In fact, it (largely) grew out of that community
  - But it serves a quite different purpose!
- Concerned with *delegating access to resources*
  - E.g., Allowing third-party apps to use your Flickr, Twitter, ... accounts
  - ...*without* giving them access to your account credentials
- http://hueniverse.com/oauth/

OAuth 1.0a

- Actors
  - Client (consumer/tinychat.com), server (service provider/Twitter), resource owner (user/you)
- Credentials
  - Temporary credentials (request token)
  - Token credentials (access token)
- Basic technique: HMAC-SHA1
  - Hashing incorporating a shared secret (password)
  - Prevents need to throw password around
- Additional protection via
  - Nonce (number-used-once) – but get expensive to track
  - Timestamp – enable old requests to be dropped

http://oauth.net/core/1.0/



OAuth 2.0

- Undergoing ratification still
- Attempts to fix problems
  - Performance at scale
  - Absence of cryptography-free options
  - Lifetime of tokens *vs*. authorization
  - Limited number of *flows*
- Adds influence from Facebook Connect
  - ...to original Flickr API Auth and Google AuthSub

Contents

- Common Requirements
- OpenID *vs*. OAuth
- OpenID
- OAuth
- SSL/TLS/HTTPS
  - Basic flow

SSL/TLS/HTTPS

- Transport Layer Security
  - Grew out of Secure Sockets Layer by Netscape
  - Incredibly complex
  - Incredibly hard to get right
  - Incredibly widely used...
- HTTPS uses TLS/SSL to
  - Provide secure channel over an insecure network
  - Verify the identity of the server
  - (Occasionally) Verify identity of the client

Using HTTPS

- For the user:
  - URLs begin https:// (TCP/443) instead of http:// (TCP/80)
- On the server
  - Generate a private/public key pair
  - Have the public key signed (signature appended using *certification authority* private key)
  - Return public certificate to client on request
- On the client
  - Client generates session key
  - Encrypts using public key to send to server
- Communication continues using symmetric encryption

Summary

- Security is complex, involving *engineering*, not just cryptography
- Generally reuse other implementations
  - Make sure you use them correctly though!
- It is a cost-benefit trade-off
  - No such thing as “perfect”
  - Sometimes using the easily available beats using the best known solution
- The system evolves: it’s an arms race