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
- Perfect security is too expensive

Contents

- Common Requirements
- Application Security
- Network Security

Contents

- Common Requirements
  - Terminology
  - Techniques
- Application Security
- Network Security

Terminology

- 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

Terminology

- Integrity
  - Prove a message not tampered with
- Confidentiality
  - Hide what you’re doing
- Non-repudiation
  - Inability to later deny sending/receiving data
- Defence in depth
  - Mitigation at many (all) levels

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
  - $\mathrm{MAC}_k(M) = h(k, M)$
  - $\mathrm{HMAC}_k(M) = h(k \oplus A, h(k \oplus B, M))$
  - 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
- Application Security
  - SSL/TLS/HTTPS
  - OpenID
  - OAuth
- Network Security

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)
  - May find themselves clicking to carry on regardless
- 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

Contents

- Common Requirements
- Application Security
  - SSL/TLS/HTTPS
  - OpenID
  - OAuth
- Network Security

OpenID *vs.* OAuth

- OpenID, http://openid.net/
  - Prove that you own (are identified with) a URL
  - Allows you to use one identity with many sites
  - Not the same as *Single Sign-On*
- OAuth, http://oauth.net/
  - Authenticates third-party access to your data
  - Using cryptographically generated tokens

OpenID

- Verified identity
  - Not trust. 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
  - (Smart mode is an 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 by 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
- Application Security
  - SSL/TLS/HTTPS
  - OpenID
  - OAuth
- Network Security

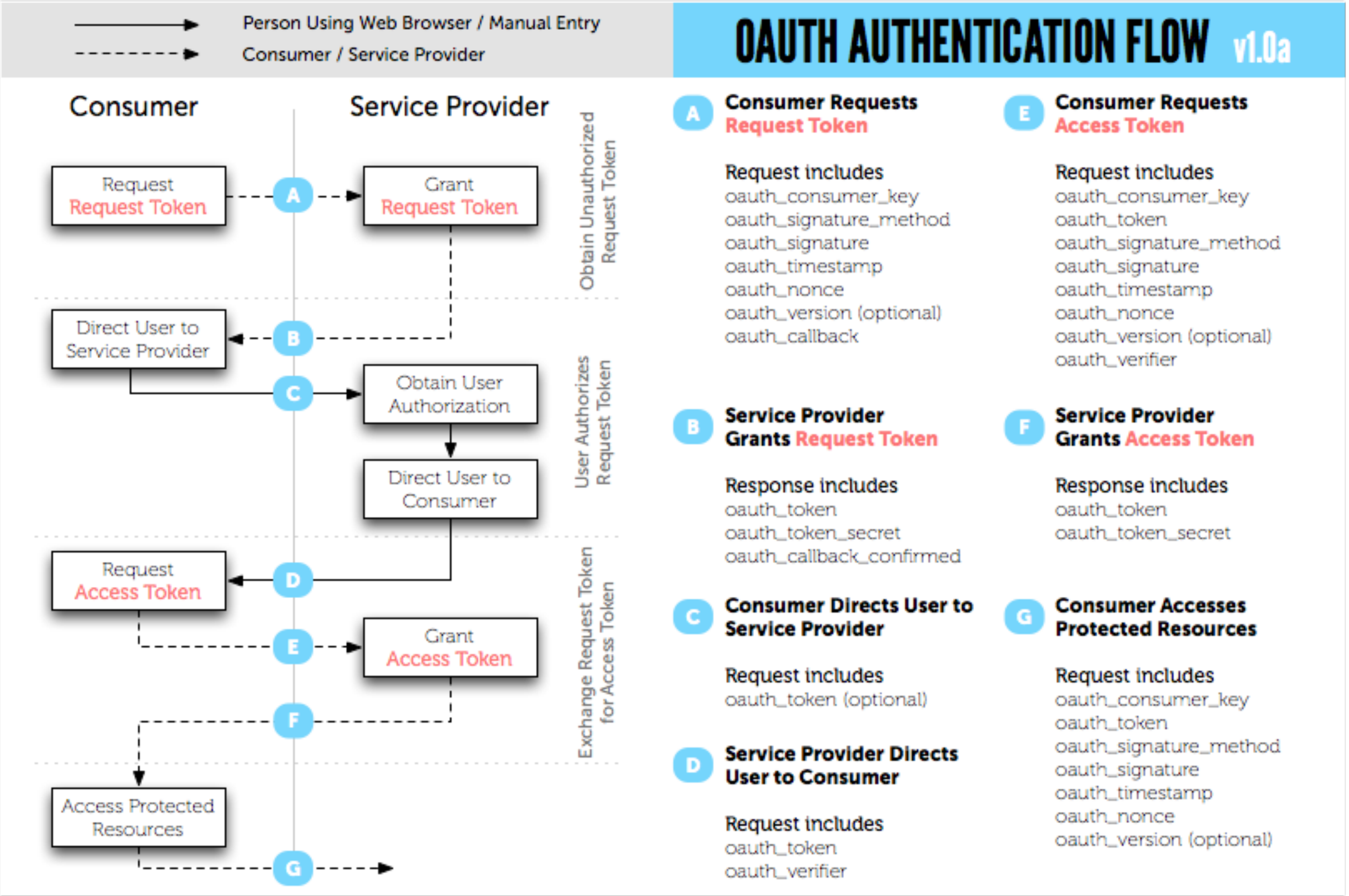
OAuth

- 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/


Person Using Web Browser / Manual Entry
Consumer / Service Provider
Consumer
Service Provider
Request Request Token
A
Grant Request Token
Obtain Unauthorized Request Token
Direct User to Service Provider
B
C
Obtain User Authorization
Direct User to Consumer
User Authorizes Request Token
Request Access Token
D
E
Grant Access Token
F
Exchange Request Token for Access Token
Access Protected Resources
G

OAUTH AUTHENTICATION FLOW v1.0a

A Consumer Requests Request Token

Request includes
oauth_consumer_key
oauth_signature_method
oauth_signature
oauth_timestamp
oauth_nonce
oauth_version (optional)
oauth_callback

B Service Provider Grants Request Token

Response includes
oauth_token
oauth_token_secret
oauth_callback_confirmed

C Consumer Directs User to Service Provider

Request includes
oauth_token (optional)

D Service Provider Directs User to Consumer

Request includes
oauth_token
oauth_verifier

E Consumer Requests Access Token

Request includes
oauth_consumer_key
oauth_token
oauth_signature_method
oauth_signature
oauth_timestamp
oauth_nonce
oauth_version (optional)
oauth_verifier

F Service Provider Grants Access Token

Response includes
oauth_token
oauth_token_secret

G Consumer Accesses Protected Resources

Request includes
oauth_consumer_key
oauth_token
oauth_signature_method
oauth_signature
oauth_timestamp
oauth_nonce
oauth_version (optional)

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* through protocol
- Adds influence from Facebook Connect
  - ...to original Flickr API Auth and Google AuthSub

Contents

- Common Requirements
- Application Security
- Network Security
  - Concepts
  - Intrusion Detection
  - Challenges

Network Security

- 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
  - – Many hacked hosts used to overload service
  - – How paranoid are you?
    - Attack *vs*. mistake *vs*. success!

Network Intrusion Detection

- Examine network traffic to detect intrusion
  - Analyse traffic patterns: *traffic analysis*
  - Analyse traffic contents: *deep packet inspection*
- For example,
  - V. Paxson, *Bro: A System for Detecting Network Intruders in Real-Time*, Computer Networks, 31 (23-24), pp. 2435-2463, 14 Dec. 1999.
  - http://www.icir.org/vern/papers/bro-CN99.html

Challenges

- Configuration and policy
  - Complex, application-specific, stateful logic
  - Need to keep separate from mechanisms
- Operation at line-rate
  - String matching, timer management
  - Cf. NPUs in the other half
- Timeliness
  - Keep signature databases up-to-date
  - Trigger *events* from observations in a timely fashion
- Monitor itself will be attacked
  - Overload, DDoS, crash, subterfuge
  - Trust nothing! E.g., Attacker may have custom TCP

Summary

- Security is complex
  - *Engineering*, not mathematics (cryptography)
- Try to reuse known-good 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