Network Security

EE 122: Intro to Communication Networks
Fall 2010 (MW 4-5:30 in 101 Barker)
Scott Shenker

TAs: Sameer Agarwal, Sara Alspaugh, Igor Ganichev, Prayag Narula
http://inst.eecs.berkeley.edu/~ee122

Materials with thanks to Jennifer Rexford, Ion Stoica, Vern Paxson
and other colleagues at Princeton and UC Berkeley

Basic Requirements for Secure Communication

• **Availability**: Will the network deliver data?
  – Infrastructure compromise, DDoS

• **Authentication**: Who is this actor?
  – Spoofing, phishing

• **Integrity**: Do messages arrive in original form?

• **Confidentiality**: Can adversary read the data?
  – Sniffing, man-in-the-middle

• **Provenance**: Who is responsible for this data?
  – Forging responses, denying responsibility
    – Not who sent the data, but who created it

Other Desirable Security Properties

• **Authorization**: is actor allowed to do this action?
  – Access controls

• **Accountability/Attribution**: who did this activity?

• **Audit/forensics**: what occurred in the past?
  – A broader notion of accountability/attribute

• **Appropriate use**: is action consistent with policy?
  – E.g., no spam; no games during business hours; etc.

• **Freedom from traffic analysis**: can someone tell
  when I am sending and to whom?

• **Anonymity**: can someone tell I sent this packet?
  – …

Today’s Lecture

• Focus on basic requirements

• Simple cryptographic methods

• Cryptographic toolkit (Hash, Digital Signature, …)

• PKIs and HTTPS

• Compromises, worms, and underground market

• Dealing with DDoS

Basic Forms of Cryptography

Confidentiality through Cryptography

• **Cryptography**: communication over insecure
  channel in the presence of adversaries

• Studied for thousands of years
  – See the Singh’s *The Code Book* for an excellent history

• Central goal: how to encode information so that an adversary can’t extract it …but a friend can

• General premise: a **key** is required for decoding
  – Give it to friends, keep it away from attackers

• Two different categories of encryption
  – Symmetric: efficient, requires key distribution
  – Asymmetric (Public Key): computationally expensive, but no key distribution problem
Symmetric Key Encryption

- Same key for encryption and decryption
  - Both sender and receiver know key
  - But adversary does not know key
- For communication, problem is key distribution
  - How do the parties (secretly) agree on the key?
- What can you do with a huge key? One-time pad
  - Huge key of random bits
  - To encrypt/decrypt: just XOR with the key!
    - Provably secure! ... provided:
      - You never reuse the key... and it really is random/unpredictable
      - Spies actually use these

Using Symmetric Keys

- Both the sender and the receiver use the same secret keys

Asymmetric Encryption (Public Key)

- Idea: use two different keys, one to encrypt (e) and one to decrypt (d)
  - A key pair
- Crucial property: knowing e does not give away d
- Therefore e can be public: everyone knows it!
- If Alice wants to send to Bob, she fetches Bob’s public key (say from Bob’s home page) and encrypts with it
  - Alice can’t decrypt what she’s sending to Bob ...
  - … but then, neither can anyone else (except Bob)

Public Key / Asymmetric Encryption

- Sender uses receiver’s public key
  - Advertised to everyone
- Receiver uses complementary private key
  - Must be kept secret

Works in Reverse Direction Too!

- Sender uses his own private key
- Receiver uses complementary public key
- Allows sender to prove he knows private key

Realizing Public Key Cryptography

- Invented in the 1970s
  - Revolutionized cryptography
  - (Was actually invented earlier by British intelligence)
- How can we construct an encryption/decryption algorithm with public/private properties?
  - Answer: Number Theory
- Most fully developed approach: RSA
  - Rivest / Shamir / Adleman, 1977; RFC 3447
  - Based on modular multiplication of very large integers
  - Very widely used (e.g., SSL/TLS for https)
Cryptographic Toolkit

- Confidentiality: Encryption
- Integrity: ?
- Authentication: ?
- Provenance: ?

Integrity: Cryptographic Hashes

- Sender computes a digest of message $m$, i.e., $H(m)$
  - $H()$ is a publicly known hash function
- Send $m$ in any manner
- Send digest $d = H(m)$ to receiver in a secure way:
  - Using another physical channel
  - Using encryption (why does this help?)
- Upon receiving $m$ and $d$, receiver re-computes $H(m)$ to see whether result agrees with $d$

Cryptographically Strong Hashes

- Hard to find collisions
  - Adversary can’t find two inputs that produce same hash
  - Someone cannot alter message without modifying digest
  - Can succinctly refer to large objects

- Hard to invert
  - Given hash, adversary can’t find input that produces it
  - Can refer obliquely to private objects (e.g., passwords)
    - Send hash of object rather than object itself

Operation of Hashing for Integrity

Internet

Plaintext

Digester (MD5)

Plaintext

Digester (MD5)

NO

corrupted msg

Digest

Digest

Effects of Cryptographic Hashing
Cryptographic Toolkit

- **Confidentiality**: Encryption
- **Integrity**: Cryptographic Hash
- **Authentication**: ?
- **Provenance**: ?

Public Key Authentication

- Each side need only to know the other side’s public key
  - No secret key need be shared
- A encrypts a nonce (random number) x using B’s public key
- B proves it can recover x
- A can authenticate itself to B in the same way

Digital Signatures

- Suppose Alice has published public key $K_E$
- If she wishes to prove who she is, she can send a message x encrypted with her private key $K_D$  
  - Therefore: anyone w/ public key $K_E$ can recover x, verify that Alice must have sent the message  
  - It provides a digital signature  
  - Alice can’t deny later deny it ⇒ non-repudiation

RSA Crypto & Signatures, con’t

Summary of Our Crypto Toolkit

- If we can securely distribute a key, then  
  - Symmetric ciphers (e.g., AES) offer fast, presumably strong confidentiality
- Public key cryptography does away with problem of secure key distribution  
  - But not as computationally efficient  
  - Often addressed by using public key crypto to exchange a session key  
  - And not guaranteed secure  
    - but major result if not
Summary of Our Crypto Toolkit, con’t

• Cryptographically strong hash functions provide major building block for integrity (e.g., SHA-1)
  – As well as providing concise digests
  – And providing a way to prove you know something (e.g., passwords) without revealing it (non-invertibility)
  – But: worrisome recent results regarding their strength
• Public key also gives us signatures
  – Including sender non-repudiation
• Turns out there’s a crypto trick based on similar algorithms that allows two parties who don’t know each other’s public key to securely negotiate a secret key even in the presence of eavesdroppers

What is Missing?

• How can you relate a key to a person?
  – Trust
• How do all these pieces fit together?
  – SSL
• What about availability?

Announcements

• HW#4 available after holiday
  – No work over break
• Section after holiday is review
  – Bring questions!
• What particular review topics would you like to have covered in the final lecture?
  – Send email!

Public Key Infrastructure (PKI)

• Public key crypto is very powerful …
• … but the realities of tying public keys to real world identities turn out to be quite hard

• PKI: Trust distribution mechanism
  – Authentication via Digital Certificates
• Trust doesn’t mean someone is honest, just that they are who they say they are…
Managing Trust

• The most solid level of trust is rooted in our direct personal experience
  – E.g., Alice’s trust that Bob is who they say they are
  – Clearly doesn’t scale to a global network!

• In its absence, we rely on delegation
  – Alice trusts Bob’s identity because Charlie attests to it …
  – … and Alice trusts Charlie

Managing Trust, con’t

• Trust is not particularly transitive
  – Should Alice trust Bob because she trusts Charlie …
  – … and Charlie vouches for Donna …
  – … and Donna says Eve is trustworthy …
  – … and Eve vouches for Bob’s identity?

• Two models of delegating trust
  – Rely on your set of friends and their friends
    o “Web of trust” -- e.g., PGP
  – Rely on trusted, well-known authorities (and their minions)
    o “Trusted root” -- e.g., HTTPS

PKI Conceptual Framework

• Trusted-Root PKI:
  – Basis: well-known public key serves as root of a hierarchy
  – Managed by a Certificate Authority (CA)

  • To publish a public key, ask the CA to digitally sign a statement indicating that they agree (“certify”) that it is indeed your key
    – This is a certificate for your key (certificate = bunch of bits)
    – Includes both your public key and the signed statement
    – Anyone can verify the signature

  • Delegation of trust to the CA
    – They’d better not screw up (duped into signing bogus key)
    – They’d better have procedures for dealing with stolen keys
    – Note: can build up a hierarchy of signing

Components of a PKI

Digital Certificate

• Signed data structure that binds an entity with its corresponding public key
  – Signed by a recognized and trusted authority, i.e., Certification Authority (CA)
  – Provide assurance that a particular public key belongs to a specific entity

  • Example: certificate of entity Y
    Cert = E({name_Y, KY_public}, KCA_private)
    – KCA_private: private key of Certificate Authority
    – name_Y: name of entity Y
    – KY_public: public key of entity Y
    – In fact, they may sign whatever glob of bits you give them

  • Your browser has a bunch of CAs wired into it

Certification Authority

• People, processes responsible for creation, delivery and management of digital certificates

  • Organized in an hierarchy
    – To verify signature chain, follow hierarchy up to root
Registration Authority

• People & processes responsible for:
  – Authenticating the identity of new entities (users or computing devices), e.g.,
    o By phone, or physical presence + ID
  – Issuing requests to CA for certificates
• The CA must trust the Registration Authority

Certificate Repository

• A database accessible to all users of a PKI
• Contains:
  – Digital certificates
  – Policy information associated with certs
  – Certificate revocation information
    o Vital to be able to identify certs that have been compromised
    o Usually done via a revocation list

Putting It All Together: HTTPS

• Steps after clicking on https://www.amazon.com
• https = “Use HTTP over SSL/TLS”
  – SSL = Secure Socket Layer
  – TLS = Transport Layer Security
    o Successor to SSL, and compatible with it
  – RFC 4346
• Provides security layer (authentication, encryption) on top of TCP
  – Fairly transparent to the app

HTTPS Connection (SSL/TLS), con’t

• Browser (client) connects via TCP to Amazon’s HTTPS server
• Client sends over list of crypto protocols it supports
• Server picks protocols to use for this session
• Server sends over its certificate
  (all of this is in the clear)

Inside the Server’s Certificate

• Name associated with cert (e.g., Amazon)
• Amazon’s public key
• A bunch of auxiliary info (physical address, type of cert, expiration time)
• URL to revocation center to check for revoked keys
• Name of certificate’s signatory (who signed it)
• A public-key signature of a hash (MD5) of all this
  – Constructed using the signatory’s private RSA key

Validating Amazon’s Identity

• Browser retrieves cert belonging to the signatory
  – These are hardwired into the browser
• If it can’t find the cert, then warns the user that site has not been verified
  – And may ask whether to continue
  – Note, can still proceed, just without authentication
• Browser uses public key in signatory’s cert to decrypt signature
  – Compares with its own MD5 hash of Amazon’s cert
• Assuming signature matches, now have high confidence it’s indeed Amazon …
  – … assuming signatory is trustworthy
HTTPS Connection (SSL/TLS), con’t

- Browser constructs a random session key $K$
- Browser encrypts $K$ using Amazon’s public key
- Browser sends $E(K, KA_{public})$ to server
- Browser displays $K$
- All subsequent communication encrypted w/ symmetric cipher using key $K$
  - E.g., client can authenticate using a password

It is a Big Bad World Out There...

Host Compromise

- Tricking a host into executing on your behalf
- Can consider what is attacked (server or client) and the semantic level at which it is attacked
- Attacks on servers: client sends subversive requests
  - Happens at attacker’s choosing
  - Some hosts are servers unknowingly!
- Attacks on clients: server (attacker) waits for client to connect, sends it a subversive reply
  - E.g., “drive-by” spyware
  - E.g., 2006 study found 15% of popular P2P files infected by one of 52 different viruses

Automated Compromise: Worms

- When attacker compromises a host, they can instruct it to do whatever they want
- Instructing it to find more vulnerable hosts to repeat the process creates a worm: a program that self-replicates across a network
  - Often spread by picking 32-bit Internet addresses at random to probe …
  - … but this isn’t fundamental
- As the worm repeatedly replicates, it grows exponentially fast because each copy of the worm works in parallel to find more victims

Worms: Exponentially Fast …. and Big

- Code Red 1 (2001)
  - 369K hosts in 10 hours
- Blaster (2003)
  - 9M hosts in 9 days
  - 25M hosts total
- Slammer (2003)
  - 75K hosts …
  - … in < 10 minutes
  - Peak scanning rate:
    - 55M addresses/sec
    - Limited by Internet’s capacity
- Theoretical worms
  - 1M hosts in 1.3 sec (2004)

Automated Compromise: Bots

- Big worms are flashy but rare …
- … With the commercialization of malware, the tool of choice has shifted to the less noisy, more directly controlled botnets
- When host is (automatically) compromised, don’t continue propagation
  - Instead install a command and control platform (a bot)
- Now can monetize malware: sell access to bots
  - Spamming, phishing web sites, flooding attacks
  - “Crock’s Google Desktop”: sell capability of searching the contents of 100,000s of hosts
- (Note: we still worry about worms for cyberwarfare)
Underground Marketplace Ads for Goods

Marketplace Ads for Services

Protecting Availability

Threats to Availability

Denial of Service (DoS)

DoS: Network Flooding

- Attacker prevents legitimate users from using something (network, server)
- Motives?
  - Retaliation
  - Extortion (e.g., betting sites just before big matches)
  - Commercial advantage (disable your competitor)
  - Cripple defenses (e.g., firewall) to enable broader attack
- Often done via some form of flooding
- Can be done at different semantic levels
  - Network: clog a link or router with a huge rate of packets
  - Transport: overwhelm victim’s ability to handle connections
  - Application: overwhelm victim’s ability to handle requests

- Goal is to clog network link(s) leading to victim
  - Either fill the link, or overwhelm their routers
  - Users can’t access victim server due to congestion
- Attacker sends traffic to victim as fast as possible
  - It will often use (many) spoofed source addresses …
- Using multiple hosts (slaves, or zombies) yields a Distributed Denial-of-Service attack, aka DDoS
- Traffic is varied (sources, destinations, ports, length) so no simple filter matches it
- If attacker has enough slaves, often doesn’t need to spoof - victim can’t shut them down anyway! :-(

- Infrastructure compromise:
  - Design protocols to have limited Byzantine vulnerability
  - Prevent outsiders from posing as infrastructure (crypto)
- Defend against Denial-of-Service Attacks
  - What are they?
  - How to defend against them?
**Distributed Denial-of-Service (DDoS)**

- Control traffic directs slaves at victim.
- Slaves send streams of traffic (perhaps spoofed) to victim.

**Very Nasty DoS Attack: Reflectors**

- **Reflection**
  - Cause one non-compromised host to help flood another
  - E.g., host A sends DNS request or TCP SYN with source V to server R.
  - R sends reply to V

**Defending Against Network Flooding**

- How do we defend against such floods?
- Answer: basically, we don’t! Big problem today!
- Techniques exist to trace spoofed traffic back to origins, but this isn’t useful in face of a large attack
- Techniques exist to filter traffic, but a well-designed flooding stream defies stateless filtering
- Best solutions to date:
  - **Overprovision** - have enough raw capacity that it’s hard to flood your links
    - Largest confirmed botnet to date: 1.5 million hosts
    - Floods seen to date: 40+ Gbps
  - **Distribute** your services - force attacker to flood many points
    - E.g., the root name servers

**Proposed Solutions**

- **Network-level attacks:**
  - Capabilities: don’t let flows send without permission
  - Shut-up message
- **Application-level attacks:**
  - Proof-of-work
  - Ask clients to send more
The End....

• Next lecture: Monday, Nov 29th

• Then review

• Enjoy your holiday!