Outline for today

Administrative:

Objective:

– Basics of crypto
– Byzantine Generals
– Access Control

Security Goals and Threats

<table>
<thead>
<tr>
<th>Goal</th>
<th>Threat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data confidentiality</td>
<td>Exposure of data</td>
</tr>
<tr>
<td>Data integrity</td>
<td>Tampering with data</td>
</tr>
<tr>
<td>System availability</td>
<td>Denial of service</td>
</tr>
</tbody>
</table>
Reliability Mechanisms (Redundancy)

- Replication of data, geographically distributed
  - Backups (Phoenix)
  - First-class replication (LOCKSS)
- Error detection-correction
  - Parity bits, checksums
  - Voting schemes

Basics of Crypto
Basics of Cryptography

Relationship between the plaintext and the ciphertext

Secret-Key Cryptography

- Monoalphabetic substitution
  - each letter replaced by different letter

- Given the encryption key,
  - easy to find decryption key

- Secret-key crypto called symmetric-key crypto
  - If keys are long enough there are OK algorithms
  - Secret key must be shared by both parties
  - DES (Data Encryption Standard)
Public-Key Cryptography

- All users pick a public key/private key pair
  - publish the public key
  - private key not published

- Public key is the encryption key
- Private key is the decryption key

- RSA (Rivest, Shamir, Adelman)

One-Way Functions

- Function such that given formula for $f(x)$
  - easy to evaluate $y = f(x)$
- But given $y$
  - computationally infeasible to find $x$

- Example: Hash functions – produce fixed size result
  - MD5
  - SHA
Digital Signatures

- Computing a signature block
  - Hash is fixed length – apply private key
- What the receiver gets
  - Use public key on signature block to get hash back
  - Compute the hash of document part
  - Do these match?
- Assumes $E(D(x)) = x$ when we usually want $D(E(x)) = x$
- Public key must be known by receiver somehow – certificate

Distributing Public Keys

- Certificate authority
  - Trusted 3rd party
  - Their public key known
- Send name and public key, digitally signed by ca
Byzantine Generals Problem
(Agreement)

Consensus problem

- Models distributed agreement
- Comes in various forms (with subtle differences in the associated results)!
- With a leader: leader gives an order, like “attack”, and non-faulty participants either attack or do nothing, despite some limited number of failures: Byzantine Agreement
- Without a leader: participants have an initial vote; protocol runs and eventually all non-faulty participants choose the same outcome, and it is one of the initial votes (typically, 0 or 1): Fault-tolerant Consensus

Ack: Ken Birman
Byzantine Agreement

• General commands soldiers
• If *all* loyal soldiers attack victory is certain
• If *none* attack, the Empire survives
• If *some* attack, Empire is lost
• Gong keeps time
• … but they don’t need to all attack at once

Byzantine soldiers

• The enemy works by corrupting the soldiers
• Orders are distributed by exchange of messages, corrupt soldiers violate protocol at will
• But corrupt soldiers can’t intercept and modify messages between loyal troops
• The gong sounds slowly: there is ample time for loyal soldiers to exchange messages (all to all)
More formal

• Agreement:
  – Every correct node chooses the same value
  – If all the correct nodes have the same input, that input must be the value chosen

Impossibility Results

• Let $t$ be the maximum number of faulty processes that our protocol is supposed to tolerate
• Byzantine agreement is not possible with fewer than $3t+1$ processes
Possibility results

• Using digital signatures, we can solve the problem fairly easily
• A digital signature is some form of proof that a message is (1) uncorrupted, and (2) that it originated with the “claimed” sender
• Suppose F asserts to B that it received \( m \) from A
• Lacking signatures, F could corrupt the message
• With signatures, F says nothing, or tells truth

Using signatures to solve Byzantine Agreement

• Initially, leader (who could be faulty) broadcasts a proposed value, signing it.
• Now run \( t \) additional rounds (for a total of \( t+1 \)):
  – Input to round \( j \) for process \( p \) is a set of messages broadcast in round \( j-1 \)
  – Discard any that lack \( j \) signatures
  – Sign remaining messages
  – Broadcast these
• In round \( t+1 \) if the leader proposed two values pick 0, else pick the leader’s proposed value, within the set of messages that have \( t+1 \) signatures.
Example

Ack: Ken Birman
Making the decision

- We need to convince ourselves that the correct processes make the same decision.
- That is, they base their decision on the same set of messages.
- So, if correct process $p$ includes message $m$ in the set of messages it considers valid, we need to know that all other correct processes $q$ also have $m$ and consider it valid.
- Notice that $m$ need not have reached $q$ from the same source. That is, the signatures are used to decide which messages to consider, but are not “part” of the message.
- In this sense the set only contains 1 or 2 messages.

Two cases

- Message $m$ was broadcast by a correct process in the final round.
  - In this case, since the process was correct, all other correct processes will have received $m$.
  - Keep in mind that in this model, message loss is counted as a “failure of the sender process.”

Ack: Ken Birman
Two cases

• What if message \( m \) was sent to \( p \) by an incorrect process?
  – There is a chain of \( t+1 \) rounds of relaying \( m \)
  – At most \( t \) failures occurred
  – So one round was a broadcast of \( m \) by a correct process
  – In the next round \( m \) reached every correct process
  – And so every correct process still has \( m \) in its “decision set”

Ack: Ken Birman

File Access Control
File Access Control

- Access control lists - detailed list attached to file of users allowed (denied) access, including kind of access allowed/denied.
- UNIX RWX - owner, group, everyone
- Capabilities – permitted accesses associated with subject (user), similar to an address space.
  - Un-forgeable object reference, like a pointer.

Access Control Lists

- **Approach**: represent the access matrix by storing its columns with the objects.
  - Tag each object with an access control list (ACL) of authorized subjects/principals.
- To authorize an access requested by $S$ for $O$
  - search $O$'s ACL for an entry matching $S$
  - compare requested access with permitted access
  - access checks are often made only at bind time
Access Control Lists

Use of access control lists of manage file access

Access Control Lists

<table>
<thead>
<tr>
<th>File</th>
<th>Access control list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>tana, sysadm: RW</td>
</tr>
<tr>
<td>Pigeon_data</td>
<td>bill, pigfan: RW; tana, pigfan: RW; ...</td>
</tr>
</tbody>
</table>

Two access control lists
**UNIX access control**

- Each file carries its access control with it.
  
  rwx rwx rwx setuid
  
  Owner UID  Group GID  Everybody else
  
  - Owner has chmod, chgrp rights (granting, revoking)
  
  When bit set, it allows process executing object to assume UID of owner temporarily - enter owner domain (rights amplification)

---

**Capabilities**

- *Approach*: represent the access matrix by storing its rows with the subjects.
  
  - Tag each subject with a list of *capabilities* for the objects it is permitted to access.
  
  - A *capability* is an unforgeable object reference, like a pointer.
  
  - It endows the holder with permission to operate on the object
    
    - e.g., permission to invoke specific methods
    
  - Typically, capabilities may be passed from one subject to another.
    
    - Rights propagation and confinement problems
Each process has a capability list

<table>
<thead>
<tr>
<th>Server</th>
<th>Object</th>
<th>Rights</th>
<th>f(Objects, Rights, Check)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Capabilities

- Cryptographically-protected capability
- Generic Rights
  1. Copy capability
  2. Copy object
  3. Remove capability
  4. Destroy object
Dynamics of Protection Schemes

Problems:

• Mutual Suspicion
  When a caller requests a service, the callee should be limited in what stuff of the caller’s it has access to (only what it is passed). (entering domain - setuid vs. passing in single object)

• Modification
  When caller does pass an object as parameter, we want to control whether the callee can modify it (I send you a photo, you remove my copyright mark)

• Conservation
  When caller passes an object to gain service, we want to know that the callee doesn’t retain or even pass on the right to the object after we think they are done with it.
  (our credit card number – and selling such databases)

• Confinement
  We want to prevent information leaks (besides the objects themselves)

Dynamics of Protection Schemes

• How to revoke privileges?
• What about adding new subjects or new objects?
• How to dynamically change the set of objects accessible (or vulnerable) to different processes run by the same user?
  – Need-to-know principle / Principle of minimal privilege
  – How do subjects change identity to execute a more privileged module?
    • protection domain, protection domain switch (enter)
Protection Domains

- Processes execute in a protection domain, initially inherited from subject
- Goal: to be able to change protection domains
- Introduce a level of indirection
- Domains become protected objects with operations defined on them: owner, copy, control

<table>
<thead>
<tr>
<th></th>
<th>gradefile</th>
<th>solutions</th>
<th>proj1</th>
<th>luvlr</th>
<th>hotgossip</th>
<th>Domain0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
<td>rw</td>
<td>rwo</td>
<td>rxc</td>
<td>r</td>
<td></td>
<td>ctl</td>
</tr>
<tr>
<td>grp</td>
<td>r</td>
<td>rwx</td>
<td></td>
<td></td>
<td>enter</td>
<td></td>
</tr>
<tr>
<td>Terry</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lynn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain0</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td>enter</td>
<td></td>
</tr>
</tbody>
</table>

- If domain contains copy on right to some object, then it can transfer that right to the object to another domain.
- If domain is owner of some object, it can grant that right to the object, with or without copy to another domain.
- If domain is owner or has ctl right to a domain, it can remove right to object from that domain.
- Rights propagation.
Dynamics of Protection Schemes

Problems:

• Mutual Suspicion
  enter my domain vs. me, as owner, granting right to another domain (temp)

• Modification
  rights for particular operations

• Conservation
  ctl – being able to revoke rights; not granting copy rights

• Confinement
  remove rights to write or create anything else?