Distributed Systems

Security

Rik Sarkar

University of Edinburgh
Fall 2014

Security

- The problem: Whatever your system, there are people trying to attack
  - For money (credit card info)
  - For information (you personal id, company information, etc)
  - To simply harm you or your organization (which benefits competitors)

Some Types of attacks

- Eavesdropping/leakage:
  - Getting information that they are not supposed to get
  - Eg. Listening in the network (easy on wireless), access to storage etc

Some Types of attacks

- Masquerading
  - Pretending to be someone else
  - Eg. Someone intercepts your communication to google and pretends to be gmail web site
    - Gets gmail password
  - In general, sends you a misguiding message pretending to be “node X” which is a friend
    - Either by taking over the communication channel, or by taking over the other node itself

Some Types of attacks

- Disruption
  - Does something to spoil your system operation
  - E.g. denial of service (DOS): send so many requests/messages to a node that it cannot communicate with anyone else (or a server cannot serve any real requests)
    - More powerful: Distributed DOS: same, using many adversary nodes
  - E.g. Jamming: block the communication channel
  - E.g. Somehow cause routers to fail

Model of attacks and security

- Alice sends messages to Bob
- Messages go through a “channel”
- The adversary Eve can read things on the channel (think ethernet or wifi)

- Eve is trying to read/modify/spoof the messages

- Alice and Bob want to avoid Eve

Model of attacks and security

- The model applies even when the channel is not a network medium
- E.g. One app (alice) writes a file to hard drive (channel), which is later read by another app (Bob)
- Point is, eve should not be able to decode the file even with access to HDD

Main Defense: Encryption

- Code the message
- Main strategy in encryption:
  - Alice wants to send a number “25” to Bob
  - Two of them know a secret key “7”
  - Alice sends “32” to Bob
  - Bob compute 32-7 = 25 to recover
  - Someone eavesdropping hears “32” and cannot recover actual message without knowing the secret key
  - The key unlocks the code

Encryption

- Can be applied to any data
  - Since we can treat anything as a “number” based on binary representation
  - Just break into small pieces on which we can apply the “key addition” idea

Example: Caesar cipher

- Take each alphabet “number” and add a key
  - (a + x) mod n

- E.g. for x = 2, n=26
  - The function is (a + 2) mod 26
  - cat -> ecv
  - zoo -> bqq
  - Problem?

More complex encryptions

- Take binary representations, XOR with key in blocks
  - Not very hard for adversary to recover key by analyzing lots of data
  - More complex encryptions are harder to decode
  - E.g multiple layers of encryption

Suggested reading: A. Conan Doyle: Adventure of the dancing men.

Encryption

- Usually, the algorithm is assumed known to everyone. Only the key is secret

- E.g. A web site uses the same algorithm to communicate with everyone. But uses different keys.
  - One user cannot read another's messages.

Encryption

- Use in authentication/signing
- If the decoding using the secret key works, that implies the message was sent by Alice
- Prevents impersonation attacks

Encryption

- Problem:
  - Both parties have to know a shared secret key
  - And have to keep it “secret”
  - Question: How can you share the key without having encryption?

Public key encryptions

- Each node uses 2 different keys:
  - One is public: known to everyone
  - one is private: known only to the node
- Alice encrypts using Bob's public key and sends
- Only bob can decrypt this: secure

Public key encryptions

- Alternatively:
- Alice encrypts using her own key
  - Sends both original and encrypted data
- Bob can verify that decrypting the encrypted part with Alice's public key gives the same data
  - Authenticated, or digitally signed

Public key encryptions

- How do you send a message both secure and authenticated?

Example

- 2 Keys are inverses:
  - Using addition, 7 and -7
  - Or, using multiplication, 7 and 1/7

- Problem: inverse is easy to find given one key

Example: RSA

- M: original plaintext
- C: cipher text (encrypted)
- e = public key; d = private key
- n = p*q; where p and q are primes

Example: RSA

- M: original plaintext
- C: cipher text (encrypted)
- e = public key; d = private key
- n = p*q; where p and q are primes

RSA

- Choose two distinct prime numbers, such as p = 61 and q = 53
- Compute n = 61 * 53 = 3233
- Compute $\phi(n) = (p - 1)(q - 1) = (61 - 1)(53 - 1) = 3120$
- Choose any number $1 < e < 3120$ that is coprime to 3120.
  - Say, e = 17
- Compute $d = e^{-1} \pmod{\phi(n)} = 2753$
- Public: (n,e) = (3233, 17); Private: (n,d) = (3233, 2753)
- M = 65
- Encryption: $C = M^e \bmod n = 65^{17} \bmod 3233 = 2790$
- Decryption: $M = C^d \bmod n = 2790^{2753} \bmod 3233 = M$

- This example is from wikipedia

Public key cryptosystems

- Rely on the following fact:
  - Given a number, finding its prime factors is computationally hard (think NP-complete)
  - There is unlikely to be good algorithms
  - Best strategy is to try out all possibilities
  - Given n, adversary cannot find p & q
  - Except by trying everything or lucky guesses

Public key cryptosystems

- Depend heavily on number theory
- Properties of numbers
  - Primes are the “building blocks” of numbers
- Generating prime numbers is important in cryptography

Public key cryptosystems

- Computing large powers (65^{17} and 2790^{2753} etc) is problematic
  - Even with some mathematical tricks

- Practical systems rely on public key cryptography to exchange a random secret key
- Then use the secret key to actually transfer data

Authentication

- Authentication: checking id
- How do you know you are talking to the right person?
- Send them some text
- They send back encrypted with their provate key
- Decrypt with their public key and cross check with original data
- Problem?

Authentication

- Alice's public key can be used to check that data is from alice

- How do you know that the key is actually alice's public key?

Authentication

- Alice's public key can be used to check that data is from alice
- How do you know that the key is actually alice's public key? That someone has not intercepted communication in the middle and pretending to be alice?
- No good method

Authentication

- Real systems:
  - Depend on trusted third parties
  - Authorities

- But who is trusted?

Authentication

- Real systems:
  - Depend on trusted third parties
  - Authorities who determine who is honest and who is trying fraud
- But who is trusted?
  - Determined by *yet* other parties

Authentication and encryption methods

- SSL
- TLS
- Kerberos etc

Password storage

- Use a encryption with a specific private (throw away the public key)
- Take the passwd, store the encrypted version
  - No need to store the actual password
  - When checking login passwd, encrypt the input, compare with the stored encrypted version
    - Essentially hashing

Data verification

- Use the encryption compute a small hash of the file
- When file is transmitted across a channel, compute the encryption hash again and compare.
- Data corruption over the channel will cause the hash to be different (with high probability)