Security:
Principles and Practice

Question

- Can you write a self-replicating C program?
  - program that when run, outputs itself
    - without reading any input files!
  - ex: main() { printf("main () { printf("main () …

Last Time

- Approaches to storage reliability
  - Careful sequencing of file system operations
  - Copy-on-write (WAFL, ZFS)
  - Journalling (NTFS, linux ext4)
  - Log structure (flash storage)

Main Points

- Wrapup storage reliability
  - RAID
- Security theory
  - Access control matrix
  - Passwords
  - Encryption
- Security practice
  - Example successful attacks

Storage Availability

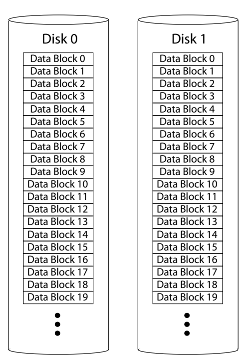
- Storage reliability: data fetched is what you stored
  - Transactions, redo logging, etc.
- Storage availability: data is there when you want it
  - More disks => higher probability of some disk failing
  - Data available ~ Prob(disk working)^k
    - If failures are independent and data is spread across k disks
  - For large k, probability system works -> 0

RAID

- Replicate data for availability
  - RAID 0: no replication
  - RAID 1: mirror data across two or more disks
    - Google File System replicated its data on three disks, spread across multiple racks
  - RAID 5: split data across disks, with redundancy to recover from a single disk failure
  - RAID 6: RAID 5, with extra redundancy to recover from two disk failures

RAID 1: Mirroring

- Replicate writes to both disks
- Reads can go to either disk



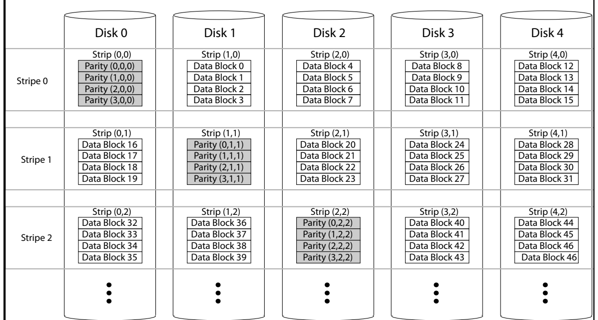
Parity

- Parity block: Block1 xor block2 xor block3 …

```
10001101    block1
01101100    block2
11000110    block3
-------------
00100111    parity block
```

- Can reconstruct any missing block from the others

RAID 5: Rotating Parity



RAID Update

- Mirroring
  - Write every mirror
- RAID-5: to write one block
  - Read old data block
  - Read old parity block
  - Write new data block
  - Write new parity block
    - Old data xor old parity xor new data
- RAID-5: to write entire stripe
  - Write data blocks and parity

Non-Recoverable Read Errors

- Disk devices can lose data
  - One sector per 10^15 bits read
  - Causes:
    - Physical wear
    - Repeated writes to nearby tracks
- What impact does this have on RAID recovery?

Read Errors and RAID recovery

- Example
  - 10 1 TB disks, and 1 fails
  - Read remaining disks to reconstruct missing data
- Probability of recovery =
  (1 – 10^15)^(9 disks * 8 bits * 10^12 bytes/disk)
  = 93%
- Solutions:
  - RAID-6: two redundant disk blocks
    - parity, linear feedback shift
  - Scrubbing: read disk sectors in background to find and fix latent errors

Security: Theory

- Principals
  - Users, programs, sysadmins, …
- Authorization
  - Who is permitted to do what?
- Authentication
  - How do we know who the user is?
- Encryption
  - Privacy across an insecure network
  - Authentication across an insecure network
- Auditing
  - Record of who changed what, for post-hoc diagnostics

Authorization

- Access control matrix
  - For every protected resource, list of who is permitted to do what
  - Example: for each file/directory, a list of permissions
    - Owner, group, world: read, write, execute
    - Setuid: program run with permission of principal who installed it
  - Smartphone: list of permissions granted each app

Principle of Least Privilege

- Grant each principal the least permission possible for them to do their assigned work
  - Minimize code running inside kernel
  - Minimize code running as sysadmin
- Practical challenge: hard to know
  - what permissions are needed in advance
  - what permissions should be granted
    - Ex: to smartphone apps
    - Ex: to servers

Authorization with Intermediaries

- Trusted computing base: set of software trusted to enforce security policy
- Servers often need to be trusted
  - E.g.: storage server can store/retrieve data, regardless of which user asks
  - Implication: security flaw in server allows attacker to take control of system

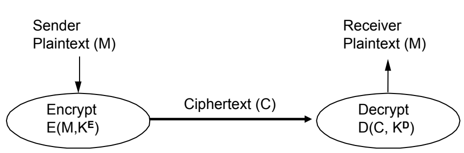
Authentication

- How do we know user is who they say they are?
- Try #1: user types password
  - User needs to remember password!
  - Short passwords: easy to remember, easy to guess
  - Long passwords: hard to remember

Question

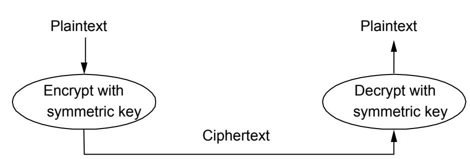
- Where are passwords stored?
  - Password is a per-user secret
  - In a file?
    - Anyone with sysadmin permission can read file
  - Encrypted in a file?
    - If gain access to file, can check passwords offline
    - If user reuses password, easy to check against other systems
  - Encrypted in a file with a random salt?
    - Hash password and salt before encryption, foils precomputed password table lookup

Encryption



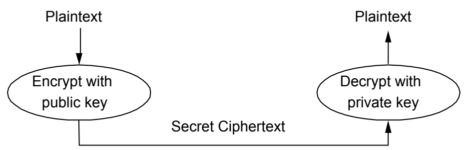
- Cryptographer chooses functions E, D and keys K^E, K^D
  - Suppose everything is known (E, D, M and C), should not be able to determine keys K^E, K^D and/or modify msg
  - provides basis for authentication, privacy and integrity

Symmetric Key (DES, IDEA)



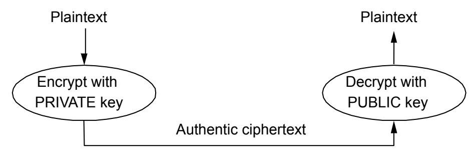
- Single key (symmetric) is shared between parties, kept secret from everyone else
  - Ciphertext = (M)^K; Plaintext = M = ((M)^K)^K
  - if K kept secret, then both parties know M is authentic and secret

Public Key (RSA, PGP)



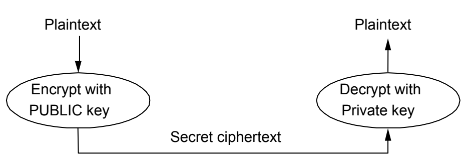
Keys come in pairs: public and private
- Each principal gets its own pair
- Public key can be published; private is secret to entity
  - can't derive K-private from K-public, even given M, (M)^K-priv

Public Key: Authentication



Keys come in pairs: public and private
- M = ((M)^K-private)^K-public
- Ensures authentication: can only be sent by sender

Public Key: Secrecy



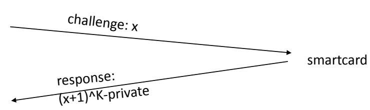
Keys come in pairs: public and private
- M = ((M)^K-public)^K-private
- Ensures secrecy: can only be read by receiver

Encryption Summary

- Symmetric key encryption
  - Single key (symmetric) is shared between parties, kept secret from everyone else
  - Ciphertext = (M)^K
- Public Key encryption
  - Keys come in pairs, public and private
  - Secret: (M)^K-public
  - Authentic: (M)^K-private

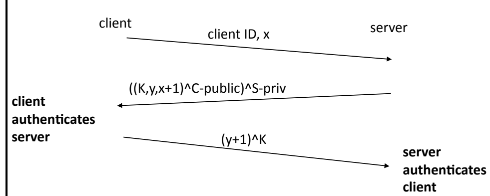
Two Factor Authentication

- Can be difficult for people to remember encryption keys and passwords
- Instead, store K-private inside a chip
  - use challenge-response to authenticate smartcard
  - Use PIN to prove user has smartcard



Public Key -> Session Key

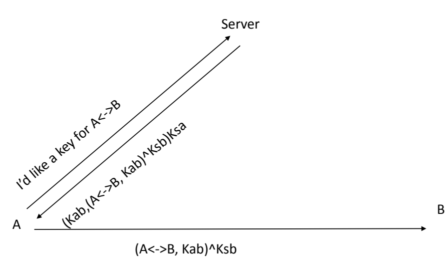
- Public key encryption/decryption is slow; so can use public key to establish (shared) session key
  - assume both sides know each other's public key



Symmetric Key -> Session Key

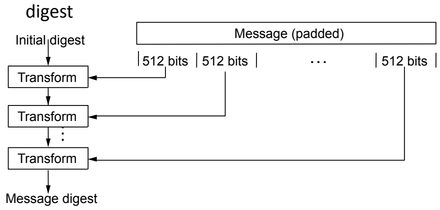
- In symmetric key systems, how do we gain a session key with other side?
  - infeasible for everyone to share a secret with everyone else
  - solution: "authentication server" (Kerberos)
    - everyone shares (a separate) secret with server
    - server provides shared session key for A <-> B
  - everyone trusts authentication server
    - if compromise server, can do anything!

Kerberos Example



Message Digests (MD5, SHA)

- Cryptographic checksum: message integrity
  - Typically small compared to message (MD5 128 bits)
  - "One-way": infeasible to find two messages with same digest



Security Practice

- In practice, systems are not that secure
  - hackers can go after weakest link
    - any system with bugs is vulnerable
  - vulnerability often not anticipated
    - usually not a brute force attack against encryption system
  - often can't tell if system is compromised
    - hackers can hide their tracks
  - can be hard to resecure systems after a breakin
    - hackers can leave unknown backdoors

Tenex Password Attack

- Early system supporting virtual memory
- Kernel login check:

```
for (i = 0; i < password length; i++) {
  if (password[i] != userpwd[i]) return error;
}
return ok
```

Internet Worm

- Used the Internet to infect a large number of machines in 1988
  - password dictionary
  - sendmail bug
    - default configuration allowed debug access
    - well known for several years, but not fixed
  - fingerd: finger tom@cs
    - fingerd allocated fixed size buffer on stack
    - copied string into buffer without checking length
    - encode virus into string!
- Used infected machines to find/infect others

Ping of Death

- IP packets can be fragmented, reordered in flight
- Reassembly at host
  - can get fragments out of order, so host allocates buffer to hold fragments
- Malformed IP fragment possible
  - offset + length > max packet size
  - Kernel implementation didn't check
- Was used for denial of service, but could have been used for virus propagation

Netscape

- Used time of day to pick session key
  - easy to predict, break
- Offered replacement browser code for download over Web
  - four byte change to executable made it use attacker's key
- Buggy helper applications (ex: pdf)
  - if web site hosts infected content, can infect clients that browse to it

Code Red/Nimda/Slammer

- Dictionary attack of known vulnerabilities
  - known Microsoft web server bugs, email attachments, browser helper applications, …
  - used infected machines to infect new machines
- Code Red:
  - designed to cause machines surf to whitehouse.gov simultaneously
- Nimda:
  - Left open backdoor on infected machines for any use
  - Infected ~ 400K machines; approx ~30K still infected
- Slammer:
  - Single UDP packet on MySQL port
  - Infected 100K+ vulnerable machines in under 10 minutes
- 350K node botnets now common

More Examples

- Housekeys
- ATM keypad
- Automobile backplane
- Pacemakers

Thompson Virus

- Ken Thompson self-replicating program
  - installed itself silently on every UNIX machine, including new machines with new instruction sets

Add backdoor to login.c

- Step 1: modify login.c

  A:
```
if (name == "ken") {
   don't check password;
   login ken as root;
}
```
- Modification is too obvious; how do we hide it?

Hiding the change to login.c

- Step 2: Modify the C compiler

  B:
```
if see trigger {
   insert A into the input stream
}
```
- Add trigger to login.c

  /* gobblygook */
- Now we don't need to include the code for the backdoor in login.c, just the trigger
  - But still too obvious; how do we hide the modification to the C compiler?

Hiding the change to the compiler

- Step 3: Modify the compiler

  C:
```
if see trigger2 {
   insert B and C into the input stream
}
```
- Compile the compiler with C present
  - now in object code for compiler
- Replace C in the compiler source with trigger2

Compiler compiles the compiler

- Every new version of compiler has code for B,C included
  - as long as trigger2 is not removed
  - and compiled with an infected compiler
  - if compiler is for a completely new machine: cross-compiled first on old machine using old compiler
- Every new version of login.c has code for A included
  - as long as trigger is not removed
  - and compiled with an infected compiler

Question

- Can you write a self-replicating C program?
  - program that when run, outputs itself
    - without reading any input files!
  - ex: main() { printf("main () { printf("main () …

Security Lessons

- Hard to resecure a machine after penetration
  - how do you know you've removed all the backdoors?
- Hard to detect if machine has been penetrated
  - Western Digital example
- Any system with bugs is vulnerable
  - and all systems have bugs: fingerd, ping of death, Code Red, nimda, …