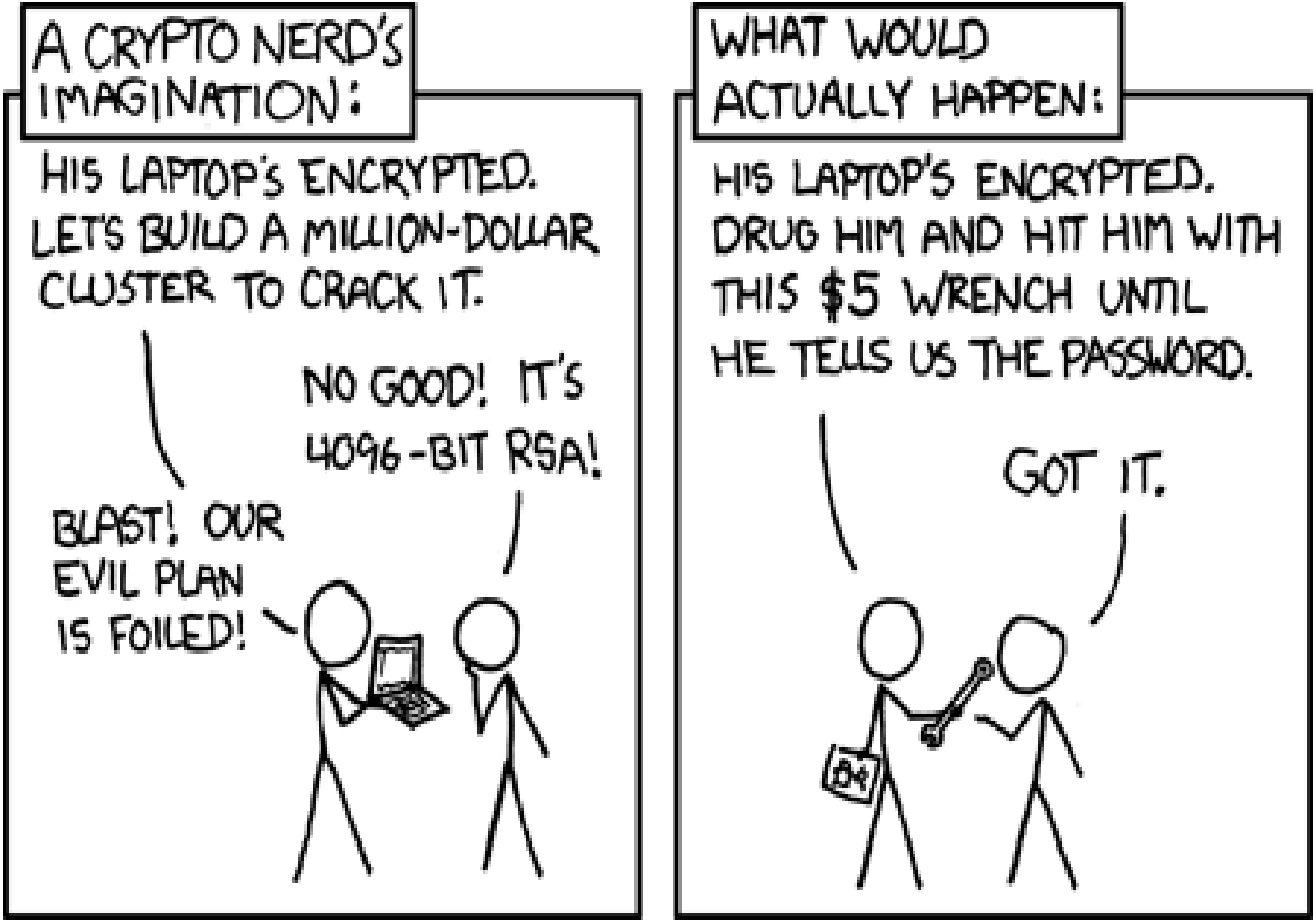
Protection and Security

Jingtao Wang

cs162-tb@imail.eecs

4/20/2009

Note: Slides based on those by Thomas Kho, Karl Chen and Adrian Mettler



* From http://xkcd.com/538/

Administrivia

- Nachos Phase 4 Initial Design Doc due this Thursday
- Please Sign up the design review meeting
  - Available on Thursday and Friday

Our Goals Today

- Conceptual understanding of how to make systems secure
- Some examples, to illustrate why providing security is really hard in practice

Protection vs Security

- Protection: To prevent accidental or intentional misuse of a system while permitting controlled sharing (It is relatively easy to achieve protection with complete isolation)
- Security: Use of protection mechanisms to prevent misuse of resources
  - Misuse defined with respect to policy
    - E.g.: prevent exposure of certain sensitive information
    - E.g.: prevent unauthorized modification/deletion of data
  - Requires consideration of the external environment within which the system operates
    - Most well-constructed system cannot protect information if user accidentally reveals password

Accidental and Intentional Misuse

- Accidental
  - Program mistakenly overwrites the file used by the system shell. Nobody else can log in.
  - You accidently destroy a file you'd like to keep.
- Malicious Abuse
  - Some high school brat who can't get a date, so instead he transfers $3 billion from B to A.
  - Someone break into a web site and steal all the credit card information stored in the database

Other Problems

- Fake timesheets for paychecks
- Repeat button printer to print extra paychecks
- Round off amounts and put into special account.
- Make up deposit slips with your account # on them.
- Make up checks with your name, but some other account # on them. (Paid out of other account).

Functional Levels of Information Protection

- Unprotected system
- All or nothing system
- Controlled sharing
- User programmed sharing controllers
  - Users want to put complex restrictions on use, such as time of day, or concurrence of another user.

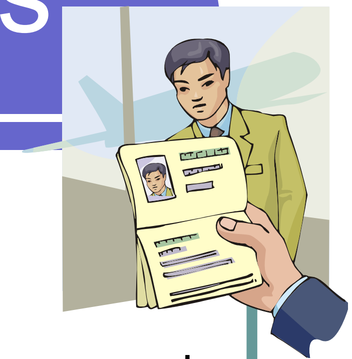
Three Pieces to Security

- Authentication
  - Who the user actually is
- Authorization
  - Who is allowed to do what
- Enforcement
  - Make sure people do only what they are supposed to do

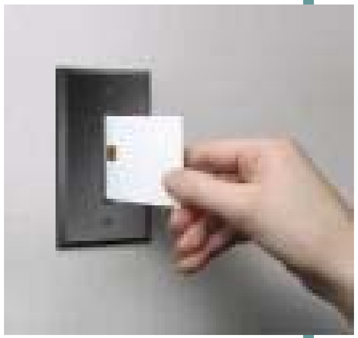
Loopholes in any carefully constructed system

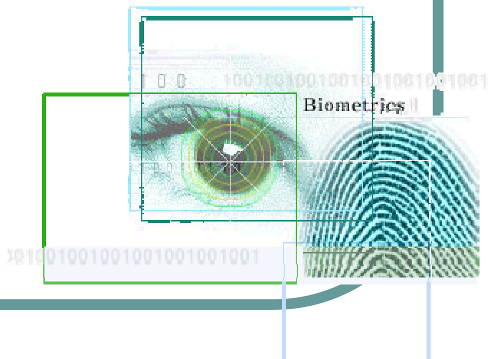
- Log in as superuser and you’ve circumvented authentication
- Log in as self and can do anything with your resources; for instance: run program that erases all of your files
- Can you trust software to correctly enforce Authentication and Authorization?????

Authentication: Identifying Users



- How to identify users to the system?
  - Passwords
    - Shared secret between two parties
    - Since only user knows password, someone types correct password ⇒ must be user typing it
    - Very common technique
  - Smart Cards
    - Electronics embedded in card capable of providing long passwords or satisfying challenge → response queries
    - May have display to allow reading of password
    - Or can be plugged in directly; several credit cards now in this category
  - Biometrics
    - Use of one or more intrinsic physical or behavioral traits to identify someone
    - Examples: fingerprint reader, palm reader, retinal scan
    - Becoming quite a bit more common





Passwords: Secrecy

- System must keep copy of secret to check against passwords
  - What if malicious user gains access to list of passwords?
    - Need to obscure information somehow
  - Mechanism: utilize a transformation that is difficult to reverse without the right key (e.g. encryption)
- Example: UNIX /etc/passwd file
  - passwd→one way transform(hash)→encrypted passwd
  - System stores only encrypted version, so OK even if someone reads the file!
  - When you type in your password, system compares encrypted version
- Problem: Can you trust encryption algorithm?
  - Example: one algorithm thought safe had back door
    - Governments want back door so they can snoop

Passwords: How easy to guess?

- Ways of Compromising Passwords
  - Password Guessing:
    - Often people use obvious information like birthday, favorite color, girlfriend's name, etc…
  - Dictionary Attack:
    - Work way through dictionary and compare encrypted version of dictionary words with entries in /etc/passwd
  - Dumpster Diving:
    - Find pieces of paper with passwords written on them
    - (Also used to get social-security numbers, etc)
- Paradox:
  - Short passwords are easy to crack
  - Long ones, people write down!
- Technology means we have to use longer passwords
  - UNIX initially required lowercase, 5-letter passwords: total of 26^5=10million passwords
    - In 1975, 10ms to check a password→1 day to crack
    - In 2005, .01μs to check a password→0.1 seconds to crack
  - Takes less time to check for all words in the dictionary!

Passwords: Making harder to crack

- How can we make passwords harder to crack?
  - Can't make it impossible, but can help
- Technique 1: Extend everyone's password with a unique number (stored in password file)
  - Called "salt". UNIX uses 12-bit "salt", making dictionary attacks 4096 times harder
  - Without salt, would be possible to pre-compute all the words in the dictionary hashed with the UNIX algorithm: would make comparing with /etc/passwd easy!
  - Also, way that salt is combined with password designed to frustrate use of off-the-shelf DES hardware
- Technique 2: Require more complex passwords
  - Make people use at least 8-character passwords with upper-case, lower-case, and numbers
    - $70^8=6\text{x}10^{14}$=6million seconds=69 days@0.01μs/check
  - Unfortunately, people still pick common patterns
    - e.g. Capitalize first letter of common word, add one digit

Passwords: Making harder to crack (con't)

- Technique 3: Delay checking of passwords
  - If attacker doesn't have access to /etc/passwd, delay every remote login attempt by 1 second
  - Makes it infeasible for rapid-fire dictionary attack
- Technique 4: Assign very long passwords
  - Long passwords or pass-phrases can have more entropy (randomness→harder to crack)
  - Give everyone a smart card (or ATM card) to carry around to remember password
    - Requires physical theft to steal password
    - Can require PIN from user before authenticates self
  - Better: have smartcard generate pseudorandom number
    - Client and server share initial seed
    - Each second/login attempt advances to next random number
- Technique 5: "Zero-Knowledge Proof"
  - Require a series of challenge-response questions
    - Distribute secret algorithm to user
    - Server presents a number, say "5"; user computes something from the number and returns answer to server
    - Server never asks same "question" twice
  - Often performed by smartcard plugged into system

More Password Attacks

- How To handle background keyboard logging, over-the-shoulder sniffing and network traffic sniffing?
- What about recording and analyzing the sound generated when typing passwords?

Using Badge or Key for Authentication

- Does not have to be kept secret.
- Should not be forgeable or copyable.
- Can be stolen, but the owner should know if it is.
- Pain to carry
- The key paradox:
  - key must be cheap to make, hard to duplicate. This means there must be some trick (i.e. secret) that has to be protected.

Authorization: Who Can Do What?

- Access Control Matrix: contains all permissions in the system
  - Resources across top
    - Files, Devices, etc…
  - Domains in columns
    - A domain might be a user or a group of permissions
    - E.g. above: User D_3 can read F_2 or execute F_3

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

How Do We Implement ACM?

- In practice, table would be huge and sparse!

Access Control Lists

- With each file(or object), indicate which users are allowed to perform which operations
  - In the most general form, each file has a list of <user, privilege> pairs (vertical split)
  - Users are usually grouped into classes
  - Relatively simple, widely used in almost all file systems. However, the overhead is relatively high.

Access Control Lists - Continue

- Easy to determine who has access, easy to revoke access
- Hard to determine what a given user can access
- Still might be lots of users!
- UNIX limits each file to: r,w,x for owner, group, world

Capabilities

- With each user, indicates which files may be accessed, and in what ways
  - Store a list of <object, privilege> pairs with each user. This is called capability list or C-List
  - Semantically, a capability is like a key
  - Popular in the past, idea out of favor today
  - Consider page table: Each process has list of pages it has access to, not each page has list of processes …

Capabilities - Implementation

- Ensure capabilities can’t be forged
- Tagged architecture
  - Each capability has a tag, which can only be set by the system.
  - Users can manipulate capabilities, but not set tag
- Segregated Architecture
  - Capabilities are segregated, and are only touched by the system

Capability Based Real Systems

- Intel 423
- Cambridge CAP System
- IBM System/38

Design Principles

- Economy of mechanism
  - keep the design as simple and small as possible. (KISS - keep it simple, stupid.)
- Fail safe defaults
  - base access decisions on permission, not exclusion. If the system fails, the default is lack of access.
- Complete mediation
  - every access to every object must be checked for authority.
- Open design
  - the design should not be secret. Its effectiveness should not be impaired by knowledge of the mechanism.

Design Principles

- Separation of privilege
  - where feasible, a protection mechanism that requires two keys to unlock it is more robust than one that allows access to the presenter of only one.
- Least privilege
  - give no more than the required access.
- Least common mechanism
  - minimize the amount of mechanism common to more than one user and depended upon by all users. (These represent the information paths.)
- Psychological acceptability
  - human interface must be convenient!

Access Enforcement

- Some part of the system must make sure the only authorized actions take place
  - Enforcer checks passwords, ACLs, etc
  - Bugs in enforcer⇒things for malicious users to exploit
- In UNIX, superuser can do anything
  - Because of coarse-grained access control, lots of stuff has to run as superuser in order to work
  - If there is a bug in any one of these programs, you lose!

Access Enforcement - Continue

- Paradox
  - Bullet-proof enforcer
    - Only known way is to make enforcer as small as possible
    - Easier to make correct, but simple-minded protection model
  - Fancy protection
    - Tries to adhere to principle of least privilege
    - Really hard to get right