CSE 451: Operating Systems
Autumn 2004

Some Principles of Security

Security

(Based on Steve Gribble's & Mike Swift's past security slides)

- Computer Security
  - techniques for computing in the presence of adversaries
    - three categories of security goals:
      - confidentiality: preventing the unauthorized release of information
      - integrity: preventing the unauthorized modification of information
      - availability: preventing denial of service attacks
    - protection is all about providing all three on a single machine
      - usually considered to be the responsibility of operating systems
      - Could be in runtime (e.g. verification in JVM)
- Cryptography
  - techniques for communicating in the presence of adversaries

Trusted Computing Base (TCB)

- Think carefully about what you are trusting with your information
  - if you type your password on a keyboard, you're trusting:
    - the keyboard manufacturer
    - your computer manufacturer
    - your operating system
    - the password library
    - the application that's checking the password
  - TCB = set of components (hardware, software, wetware) that you trust your secrets with
- Public web kiosks should not be in your TCB
  - should your OS? (e.g., IE and active-X extensions)
  - how about your compiler?
    - A great read: "Reflections on Trusting Trust" by Ken Thompson

Design Principles for Security-Conscious Systems

- Security is much more than just crypto
  - if there is a fundamental flaw in the design of a system, then all the crypto in the world won't help you
  - unfortunately, systems design is still as much an art as it is a science…
    - but, decades of building systems the wrong way have helped us cull some learned wisdom
    - we'll cover just a few of these
- Why is security hard?
  - Security slows things down
  - Security gets in the way
  - Security adds no value if there are no attacks
  - Only the government used to pay for security
    - The Internet made us all potential victims

Principle of Least Privilege

- Figure out exactly which capabilities a program needs to run, and grant it only those
  - not always easy, but: start out granting none, run program, and see where it breaks. add new privileges as needed.
- Unix: concept of root is not a good example of this
  - some programs need to run as root just to get a small privilege, such as running with a port < 1024
    - e.g., ftpd

Tractorbeaming wu-ftpd

- wu-ftp tries to run with least privilege
  - but occasionally tries to elevate its privilege with:
```
seteuid(0);
// some privileged "critical section" runs here
seteuid(getuid());
```
  - however, wu-ftp does not disable UNIX signals
    - thus, while it is in critical section, it can be tractor-beamed away to a signal handler
    - remote user can cause signal handler to run by terminating a download in midstream!
      - but need to catch wu-ftp in the critical section
    - wu-ftp doesn't relinquish privileges after signal handler
      - result: abort a download, use wu-ftp as root
      - get full access to entire remote file system

Principle of Complete Mediation

- Check **every** access to **every** object
  - in rare cases, can get away with less (caching)
    - but only if sure nothing relevant in environment has changed…and there is a lot that's relevant!
- e.g., NFS and file handles
  - NFS is not a good example of complete mediation
  - NFS protocol:
    - client contacts remote "mountd" to get a filehandle to a remotely exported NFS filesystem
      - this is done when remote system is mounted
      - remote mountd checks access control at mount time
    - filehandle is a capability: client presents it to read/write file
      - access control is not checked after mount time!
    - use network sniffer to get filehandle
      - access exported filesystem without access control check

Fail-Safe defaults

- Start by denying all access, then allow only that which has been explicitly permitted
  - oversights will then show up as "false negatives"
    - somebody is denied access that should be given it
  - opposite leads to "false positives"
    - somebody is given access that shouldn't get it
    - bad guys usually don't report this kind of failure…
- Examples:
  - Irix shipped with "xhost +" by default
    - Allows the world to open windows on your screen and grab the keystrokes you type

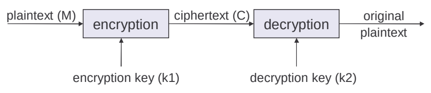
"Security through Obscurity" = bad

- Security through obscurity
  - attempting to gain security by hiding the implementation details of a system
  - claim: a secure system should be secure even if all implementation details are published
    - in fact, a system grows more secure as people scour over implementation details and find flaws
    - rely on mathematics and sound design to keep secure
- Counterexample: GSM cell phones
  - GSM committee designed their own crypto algorithm, but hid it from the world - "impossible to clone"
    - social engineering + reverse engineering revealed the algorithm
    - it turned out to be very weak
      - could essentially play questions with identity chip on cell phone, and eventually learn its secret key in a few hours

Authentication

- Identifying users and programs
- How does a computer know who I am?
  - User name / password
    - How does it store the password?
    - How does it check the password?
    - How secure is a password?
  - Public/Private Keys
  - Biometrics
- What does the computer do with this information?
  - Assign you an identifier
    - Unix: 32 bit number stored in process structure
    - Windows NT: 27 byte number, stored in an *access token* in kernel

Aside on Encryption



- Encryption: takes a key and data and creates ciphertext: $E_{k1}(M) = C$
- Decryption: takes ciphertext and a key and recovers data: $D_{k2}(C) = M$
- Symmetric algorithms (aka secret-key algorithms):
  - k1 = k2 (or can get k2 from k1)
- Public-Key Algorithms
  - decryption key (k2) cannot be calculated from encryption key (k1)
  - encryption key can be made public!
    - encryption key = "public key", decryption key = "private key"
- Hashing: takes data and creates a fixed-size fingerprint, or hash
  - H(Attack at Dawn) = 183870
  - H(attack at dawn) = 465348
  - Can't determine data from hash or find two pieces of data with same hash

Storing passwords

- CTSS (1962): password file

  Bob: 14: "12.14.52"
  David: 15: "god"
  Mary: 16: "!ofotc2n"

- Unix (1974): encrypted passwords
  - Weakness: dictionary attack

  Bob: 14: S6Uu0cYDVdTAk
  David: 15: J2ZI4ndBL6X.M
  Mary: 16: VW2bqvTalBJKg

- Unix (1979): salted passwords

  Bob: 14: S6Uu0cYDVdTAk: 45
  David: 15: J2ZI4ndBL6X.M: 392
  Mary: 16: VW2bqvTalBJKg: 152

Most common passwords

0, 000000, 00000000, 007, 1, 110, 111, 111111, 11111111, 12, 121212, 123, 123123, 1234, 12345, 123456, 1234567, 12345678, 123456789, 1234qwer, 123abc, 123asd, 123qwe, 2002, 2003, 2600, 54321, 654321, 88888888, a, aaa, abc, abc123, abcd, Admin, admin, admin123, administrator, alpha, asdf, computer, database, enable, foobar, god, godblessyou, home, ihavenopass, Internet, Login, login, love, mypass, mypass123, mypc, mypc123, null, oracle, owner, pass, passwd, Password, password, pat, patrick, pc, pw, pw123, pwd, qwer, root, secret, server, sex, super, sybase, temp, temp123, test, test123, win, xp, xxx, yxcv, zxcv

13

Password Security

- 26 letters used, 7 letters long
  - 8 billion passwords (33 bits)
  - Checking 100,000/second breaks in 22 hours
    - System should make checking passwords slow
- Adding symbols and numbers and longer passwords
  - 95 characters, 14 characters long
  - 10^{27} passwords = 91 bits
  - Checking 100,000/second breaks in 10^{14} years
- SDSC computed 207 billion hashes for 50 million passwords in 80 minutes.

14

Do longer passwords work?

- People can't remember 14-character strings of random characters
- Random number generators aren't always that good.
- People write down difficult passwords
- People give out passwords to strangers
- Passwords can show up on disk

15

Cool password attack

- VMS password checking flaw
  - password checking algorithm:

```
for (I=0; I<password.length( ); I++) {
    if password[I] == supplied_password[I]
        return false;
}
return true;
```

  - can you see the problem?
    - hint: think about virtual memory…
    - another hint: think about page faults…
    - final hint: who controls where in memory supplied_password lives?

16

Attacks: Trojan Horses

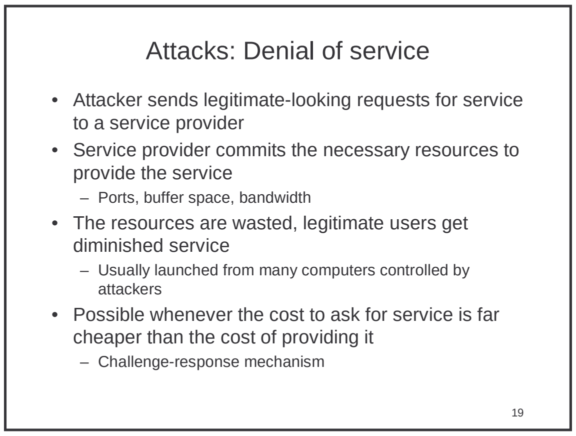
- A malicious program disguised as an innocent one
- Login spoofers are a specialized class of Trojan horses
  - Can be circumvented by requiring an operation that unprivileged programs cannot perform
  - E.g. Start login sequence with a key combination user programs cannot catch, CTRL+ALT+DEL on Windows

17

Attacks: Viruses and Worms

- Viruses: passive code attached to other programs
  - E.g. a program that modifies MS Word
- Worms: code that actively replicates itself and does not depend on the execution of another program to spread
  - E.g. the Internet worm
- Buffer overflow
  - C string libraries hard to use correctly
    - e.g. easy to write outside string bounds
  - Most OS code is written in C, many systems have vulnerabilities
  - If a string is stored on the stack, someone can modify the behavior of a program by going off the end of the string and changing a return address stored on stack

18



Attacks: Denial of service

- Attacker sends legitimate-looking requests for service to a service provider
- Service provider commits the necessary resources to provide the service
  - Ports, buffer space, bandwidth
- The resources are wasted, legitimate users get diminished service
  - Usually launched from many computers controlled by attackers
- Possible whenever the cost to ask for service is far cheaper than the cost of providing it
  - Challenge-response mechanism

19