CS 162 Lecture April 4-23 by Jing-Tao
Notes by Kevin Stich

Phase 4 Suggestions

Read operation: Non-blocking
No threads required at C programming level to implement chat server or chat client since the operations are non-blocking.

May be beneficial to create extra classes
Implementation may become very long, putting everything in NetKernel or NetProcess may be unmanageable.

Review From Monday:
3 Major topics
Authentication – Method to let OS know that you are who you claim to me. Several techniques, Username/Password is pretty popular. May be easy to attack without enforcement of password policies. Other techniques, smartcard, biometrics (fingerprint, retinal scan)
Authorization/Access Enforcement – Determine whether an operation can be performed on an object or not.
Users given "keys" for each object which is required to access it (User Level)
ACM – Big Table (authorization enforced at Access Object Level)

	Printer	File…
User0	No	Yes
User1	Yes	Yes

Different approaches, use depends on how you plan to use access right

Access Enforcement –
Paradox, bullet proof enforcer. Only known way to make enforcer as small as possible. Easierto make correct, but simple minded protection model. (Smaller Kernel, but protection must be handled in Kernel mode)
Fancy Protection, tries to adhere to principles of least privilege. Really hard to get right. (Larger Kernel, but don't need to be in Kernel mode to handle enforcement [hopefully])

Authentication: Encryption – almost noone encrypts or has public key identity
Authorization: Access Control
Many systems only provide very coarse grained access, In UNIX need to turn off protection to enable sharing
Enforcement: Kernel Mode
*Hard to write a million line program without bugs, any bug is a potential security loophole :X

Challenges:

Abuse of valid privileges – super user can do anything, if super user does something dumb, bad news bears mister Matthau
Imposter or Trojan Horse – Hijack someone else's privileges

Listener – Eavesdrop on terminal wire, or listen in on local network traffic
Spoiler – User up all resources and make system crash
Create doctored version of some standard program - Make a screen that looks just like a login screen, and then steal a users password)
Phishing Attack – imitate a legitimate source in order to trick a user into revealing their password and therefore stealing their credentials
Hijacked browser or rogue toolbar could steal info
Permission on lists of /dev files will lead to access to raw I/O devices
User leaves fake shell on terminal
Walk up to terminal that wasn't logged out
Find account with null password
Create a fake file system and have the system mount it
Buffer overflow – Copying a large input buffer to a small output buffer with a strcpy function can result in the output buffer overflowing, resulting in corruption of stack variables. Modified code can be injected into the system and executed, or the return address of a function call may be changed to return to malicious code. If program has kernel level access, the code then has kernel access. Oh nos! Basically human beings are too lazy.
Techniques to help prevent buffer overflows –
Strncpy instead of strcpy, check stack before and after a function call to ensure no overflow occurred (may help you detect an overflow, but doesn't guarantee that one didn't happen if the attacker is clever).
Tools exist to detect buffer overflow dangers. Unfortunately they produce many false positives and it may not be feasible to manually fix them.
Mostly C/C++ languages are vulnerable to this problem. Using a typesafe language is the best defense (Java, C#, Python, etc.) but doesn't help if the program is already in C/C++.
Hot topic, $10's of millions of dollars in research. 1000's of different techniques with various advantages/disadvantages.

The Morris internet worm (Self-reproducing)
Author Robert Morris, first-year Cornell grad student
Launched close to workday on November 2, 1988
Within a few hours of release, it consumed resources to the point of bringing down infected machines.
Exploited UNIX networking features (remote access)
Bugs in finger (buffer overflow) and sendmail programs (debug mode allowed remote login)
Dictionary lookup-based password cracking
BTW: Robert Morris is now a Professor at MIT

Tenex – Early 70's BBN
Most popular system at universities before UNIX

Thought to be very secure, gave “red team” all the source code and documentation (want code to be publicly available, as in Unix)
In 48 hours they figured out how to get every password in the System

Code for password check:

```
For(I = 0; I < 8; i++)
  If ( userPasswd[i] != realPasswd[i])
    Go to error
```

Tenex used VM, and interacts badly with the above code
Key idea: force page faults at inoppourtune times to break passwords quickly
Arrange first char in string to be last char in pg, rest on next pg
Then arrange for pg w/ 1st char to be in memory, and the rest to be on disk, (eg ref lots of other pgs then ref 1st page)
A| AAAAAA
Mem| Disk

Page out time indicates that first character was correct
Time password check to determine if first char was correct
- If fast, 1st char wrong
- If slow, 1st char right, page fault, one of the others is wrong
- So try all first characters, until one is slow
- So try all first characters, until one is slow
- Repeat with first two characters in memory, rest on disk

Only 256*8 attempts to crack passwords
- Fix is easy, don’t stop until you look at all the characters

Not valid anymore, but indication of how simple things can become huge security holes
If attacker is smarter than original designer (or has more free time, ex: no job) cracks can occur

Consequences of System Breakin:
Once system has be en penetrated, it may be impossible to secure it again

It’s not always possible to tell when the system has been penetrated, since the villain can clean up all traces behind himself

If we can never be sure that there are no bugs, then we can never be sure that the system is secure, since bugs could provide loopholes in the protection mechanism
(Nightmare)
If system is found compromised, might be best to reinstall from scratch

Countermeasures:
Logging
Get humans involved at key steps – ID Checks, Manual review of activity

Principle of minimum privilege – You should only provide the minimal privilege necessary for a program
Correctness proofs
Callback used to avoid abuse of accounts – Don't accept incoming connections, if a connection comes in, figure out who it is and then contact the address on record for that account (prevent people spoofing their location, used back in the day for secure banking systems)
Consistency or plausibility check (Is this user spending $10,000 when his largest previous purchase was $100)

Inference Control
The goal – allowing users to be able to get statistical information (e.g. average) out of a database, but not get individual data
The problem – can design sets of queries that will generate individual information
Average Salary of all X
Average salary of X – delta
Size of X
Can be exploited, for example if we find the average of everyone, and then find the average of every except for X, if we know the size of the database we can determine the value of X.

No good solution to this problem, can do some things
Radomize Data

The Confinement Problem
Mutually suspicious customer and service – want to insure that the service can only reach information provided by the customer and that the service is protected from the customer
Idea is concept of information utility. Idea currently resurfacing as server/web based software

Two problems remain
- Service may not perform as advertised
- Service may leak – ie transmit confidential data

Possible leaks
Service has memory, it can collect data
The service can send a message to a process controlled by its owner
The information can be encoded

Viruses –
Computer transfer around executable files and code, E.G. in email
User executes this code, and bad things happen
- Virus usually replicates itself elsewhere
- And does something unpleasant to your machine

Windows gets a lot of viruses because it is popular and (early versions) basic users have root access rights (if you can run a program, you can fuck up the machine) Kernel is fairly uniform

General Anti-Virus Techniques:

Search for known viruses by looking for their object code

Problem is that viruses can encrypt themselves, solution is to search for decryption code

Virus may change the decryption code

Solution is to interpretively execute the suspected virus code for some portion of time, to see if the code decrypts itself into something that is recognized as a common virus.

No good defense against an unknown virus, since the code patterns can't be recognized.

Best Bet: Don't run suspicious code, run a checksum of files to make sure nothing has happened.