Memory Management Part 4
Note that the mmobj’s that are not attached to vnodes are shadow objects.
Usage Examples

```c
for (j=0; j<iMax; j++) {
    for (i=0; i<iMax; i++) {
        sum += A[i][j];
    }
}

for (i=0; i<iMax; i++) {
    for (j=0; j<iMax; j++) {
        sum += A[i][j];
    }
}
```
Note that the matrix occupies roughly 9.5GB of memory. The program was run on a machine with 8GB of memory.
Providing Advice to the Kernel

- `madvise(start, length, advice)`
  - normal
  - sequential
  - random
  - will need
  - don’t need
  - and others ...
Results

• 48k x 48k matrix

• \textit{ij} loop
  – 37:00 (normal)
  – 38:01 (random)
  – 29:49 (sequential)

• \textit{ji} loop
  – 4:12 (normal)
  – 3:03 (sequential)
  – 4:15 (random)
Security Part 1
Code Defensively

• Make sure your program does only what it’s supposed to do
  – does the “right thing” for all possible sets of arguments
  – doesn’t have weird (and unanticipated) interactions with other programs
• Particularly important if your program has “special privileges”
Change Roles

• It’s more fun to play the attacker
• You can learn a lot by thinking through the attacker’s role
Code Offensively

- Your opponent's an idiot
  - take advantage of it
- Your opponent's a slob
  - profit from it
- Your opponent lives in an alternate reality
  - exploit it
This challenge was actually issued at a trade show in the early 1990s.
The attacker tricked the attackee into running `ps` while superuser and while in a directory containing code provided by the attacker.
A file was created containing a script that added to the password file the account “bogus” that has super-user privileges, yet no password. After doing this, the script then ran the real `ps` command. The path variable of the root account specified that the current directory should be searched for commands before the other directories were searched.

In the real-life story, the company who offered the challenge declared that the attacker had “cheated” and thus did not deserve $10,000 ...
Concerns

• Authentication
  – who are you?
• Access control
  – what are you allowed to do?
• Availability
  – can others keep you out?
Logging In …

- Username/password
  - who knows the passwords?
One-Way Functions

- $f(x)$ is easy to compute
- $f^{-1}(x)$ is extremely difficult, if not impossible, to compute
  - Unix password file contains image of each password
    » /etc/passwd contains twd:y
    » twd logs in, supplies x
    » if $f(x) == y$, then ok
    » /etc/passwd is readable by all
Systems that employ just one-way functions to protect their passwords are vulnerable to dictionary attacks.
Unix uses “salt” as a means to foil dictionary attacks, though it’s probably not of tremendous use anymore.
Counter Counter Attacks

• Don’t allow common access to password images
  – /etc/passwd contains everything but password images and is readable by all
  – /etc/shadow contains password images

• Use better passwords
  – “w7%3nGibwy6” rather than “fido”

• Use strong cryptography and smart cards
  – combined with PINs

• Use two-factor authentication
Defeating Authentication

- What are the prime factors of

\[5325138870287932192846843055513588820529482732761 \]
\[648423706300613697153947391340902293733259038472039713333596954925632 \]
\[2620979036686633213903952966175107096769180017646161851573147596390153\]

Hint: one of them is:
\[6438080068035544392301298549614926991513861075340134329180734395241382 \]
\[648423706300613697153947391340902293733259038472039713333596954925632 \]
\[2620979036686633213903952966175107096769180017646161851573147596390153\]

The point is that defeating a decent authentication technique is probably too tough to bother, particularly when there are probably other, much easier ways of breaking in.
Since defeating a decent crypto scheme is far too difficult, we might try stealing someone's password. If you walked up to a PC with the contents of the this slide on the screen, you might be tempted to type in your password. However, there’s the risk that this screen was not put up by the system, but by some evil user who’s trying to trick you into yielding your password. Recent versions of Windows provide a means for protection against this sort of attack: if you type ctrl-alt-delete, the response is guaranteed to be directly from the operating system and not from any application program. If there is no user logged on, the response will be a guaranteed legitimate login screen. If a user is logged on, the response will be a window to the system’s task-manager application. This notion of an input that’s guaranteed to be handled by a trusted component (the operating system in this case) is called trusted path.
We now turn our attention to hacking. Though we don’t provide a whole lot of detail in our description. All the details (and excellent working code) are provided at numerous sites on the Web.
Attacks

- Trap doors
- Trojan horses
- Viruses and worms
- Exploit bugs
- Exploit features
Trap Doors

- You supply a CD driver
- ioctl(cd_file_descriptor, 0x5309)
  - standard command to eject the CD
- ioctl(cd_file_descriptor, 0xe311)
  - second argument is passed to your driver
  - on receipt, your driver sets UID of current process to zero

On Unix systems, “superuser” has a UID of zero.
Trap-Door Prevention

• Make sure everything that goes into kernel is ok
  – the Linux kernel has over 19,000 source-code files
  – also must worry about all setuid programs
  – Windows probably has more files

• How?
  – Windows
    - really careful management
  – Linux
    - thousands of eyes checking things out
The Karger/Schell paper was “Multics Security Evaluation: Vulnerability Analysis.” They reported that even though Honeywell, who owned Multics, did an excellent job of managing and securing all the code, it still might be possible for the compiler to insert a trap door. Ken Thompson reported on how he modified the Unix C compiler to do this sort of thing in his Turing Award Lecture in 1984 (the Turing Award is the CS equivalent of the Nobel Prize): see http://cm.bell-labs.com/who/ken/trust.html/.
Trojan Horses

- Free software!!!
  - upgrades your four-core processor to eight-core!!
Viruses and Worms

- Virus: an “infection” of a program that replicates itself
- Worm: a standalone program that actively replicates itself
How to Write a Virus (1)

Program
(date)

Virus
(/bin/rm -rf /)
How to Write a Virus (2)

Program

(/bin/rm -rf /)
How to Write a Virus (3)

Program
(date;
/bin/rm -rf /)
How to Write a Virus (4)

Program
(date;
if (day ==
Tuesday)
/bin/rm -rf /)
How to Write a Virus (5)

Program
(date;
if (day ==
Tuesday)
/bin/rm -rf /
; infect
others)
Further Issues

• Make program appear unchanged
  – don’t change creation date
  – don’t change size
• How to infect others
  – email
  – web
  – direct attack
  – etc.
Programs susceptible to buffer-overflow attacks are amazingly common and thus such attacks are probably the most common of the bug-exploitation techniques. Even drivers for network interface devices have such problems, making machines vulnerable to attacks by maliciously created packets.
void proc() {
    char buf[80];
    ...
    fgets(buf, 80, stdin);
    ...
}
Better Defense

- Why should the stack contain executable code?
  - no reason whatsoever
- So, don’t allow it
  - mark stack *non-executable*
    - (how come no one thought of this earlier?)
    - (Intel didn’t support it till recently)
- Data execution prevention (DEP)
  - adopted by Windows and Linux in 2004
  - by Apple in 2006
This particular form of return-oriented programming is known as “return to libc”, since the code we’re using is in the C library.
Defense

- Example assumes parameters passed on stack
  - 32-bit x86 convention
- Switch to x86-64
  - parameters passed in registers
  - example breaks
- Offense foiled?
Defense

- Address space layout randomization (ASLR)
  - start sections at unpredictable locations
Offense

• One possibility
  – guess the start address
    - perhaps $1/2^{16}$ chance of getting it right
    - repeat attack a 100,000 times
      • won’t be noticed on busy web server
      • very likely it will (eventually) work

See “Launching Return-Oriented Programming Attacks against Randomized Relocatable Executables,” by Liu, Han, Gao, Jing, and Zha, flyer.sis.smu.edu.sg/trustcom11.pdf.