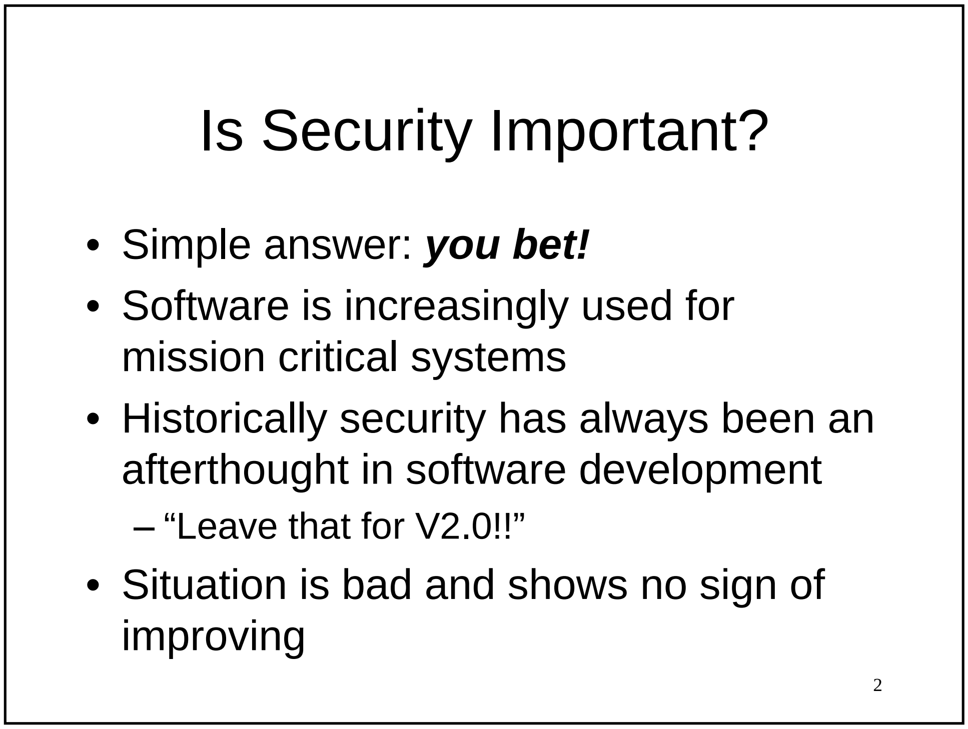
Security for Software Developers



Is Security Important?

- Simple answer: ***you bet!***
- Software is increasingly used for mission critical systems
- Historically security has always been an afterthought in software development
  - “Leave that for V2.0!!”
- Situation is bad and shows no sign of improving

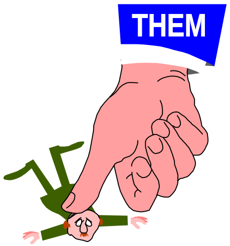
Is Security Important? *(cont)*

- Thoughtful answer: ***it depends on what you're doing***
- Amount of security-related effort expended should match significance of data in terms of dollars or damage
- But amount of security should not put you out of business
  - Time to market dominates industry

Disclaimer

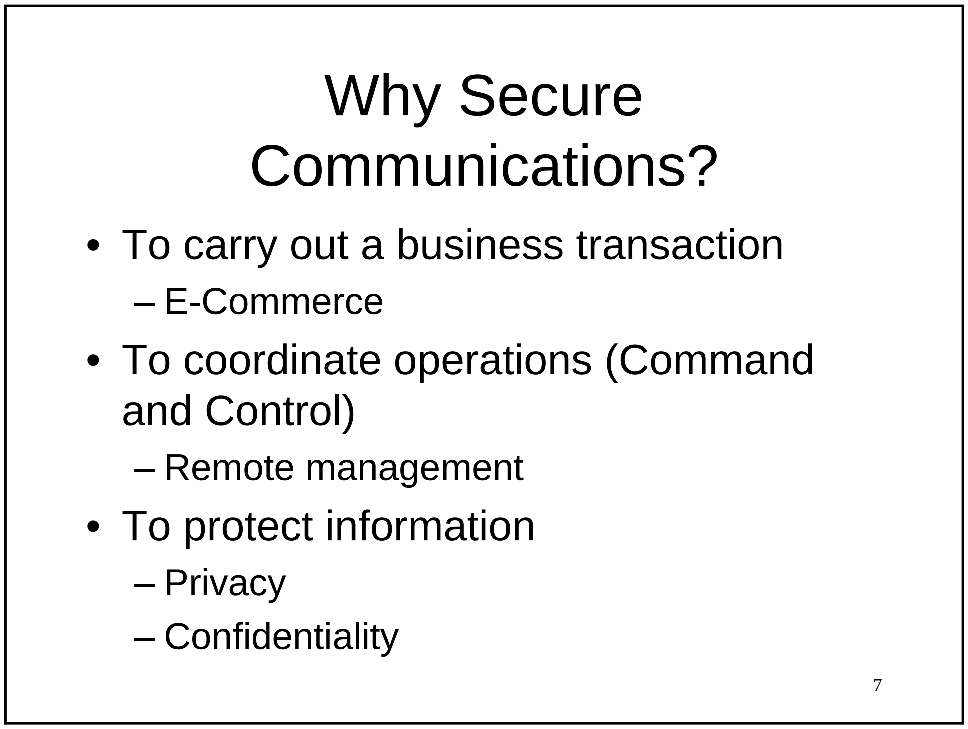
- Security requires a paranoid mindset
  - If you're going to play then you need to look at the big picture
  - This tutorial is intended to give a background on communications security
  - You could spend your life doing this stuff and still make mistakes
- ***Nothing*** is secure

Just because you’re paranoid doesn’t mean “they” aren’t out to get you.



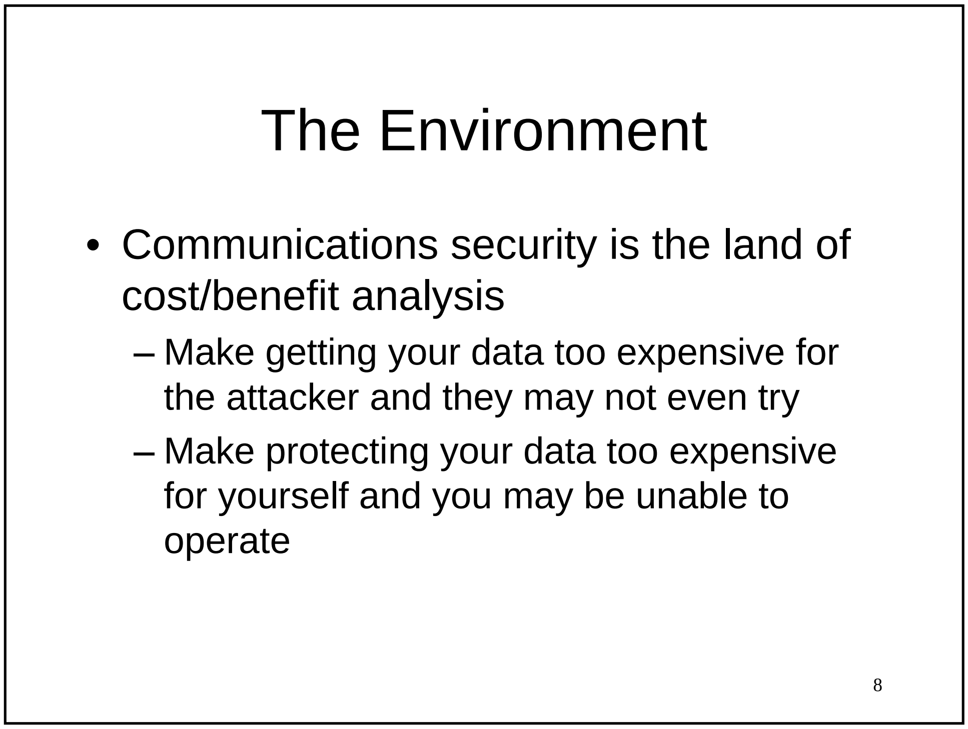
Before You Start

- Risk Assessment:
  - What are you trying to hide?
  - How much will it hurt if “they” find it out?
  - How hard will “they” try?
  - How much are you willing to spend? “spend” means a combination of:
    - Time
    - Pain
    - Money



Why Secure Communications?

- To carry out a business transaction
  - E-Commerce
- To coordinate operations (Command and Control)
  - Remote management
- To protect information
  - Privacy
  - Confidentiality



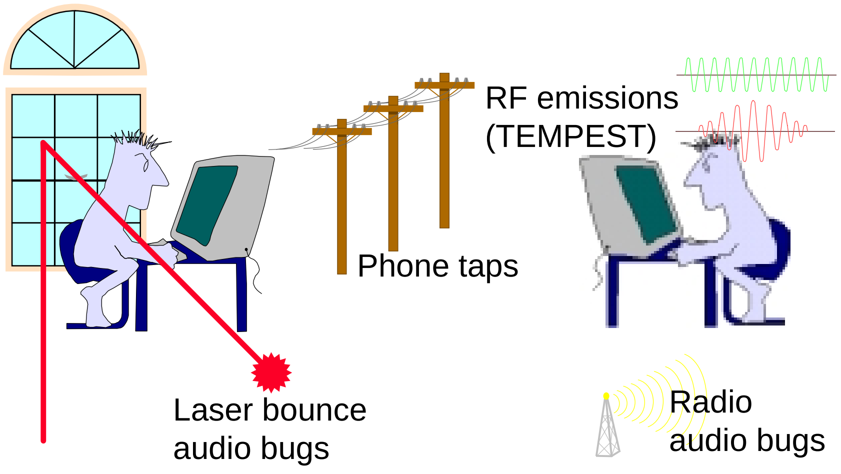
The Environment

- Communications security is the land of cost/benefit analysis
  - Make getting your data too expensive for the attacker and they may not even try
  - Make protecting your data too expensive for yourself and you may be unable to operate

Target Analysis

- Target analysis is the (hypothetical) art of analyzing a target's communications security to identify the weakest link
- You'd better do it, because "they" will do it, too

Target Analysis

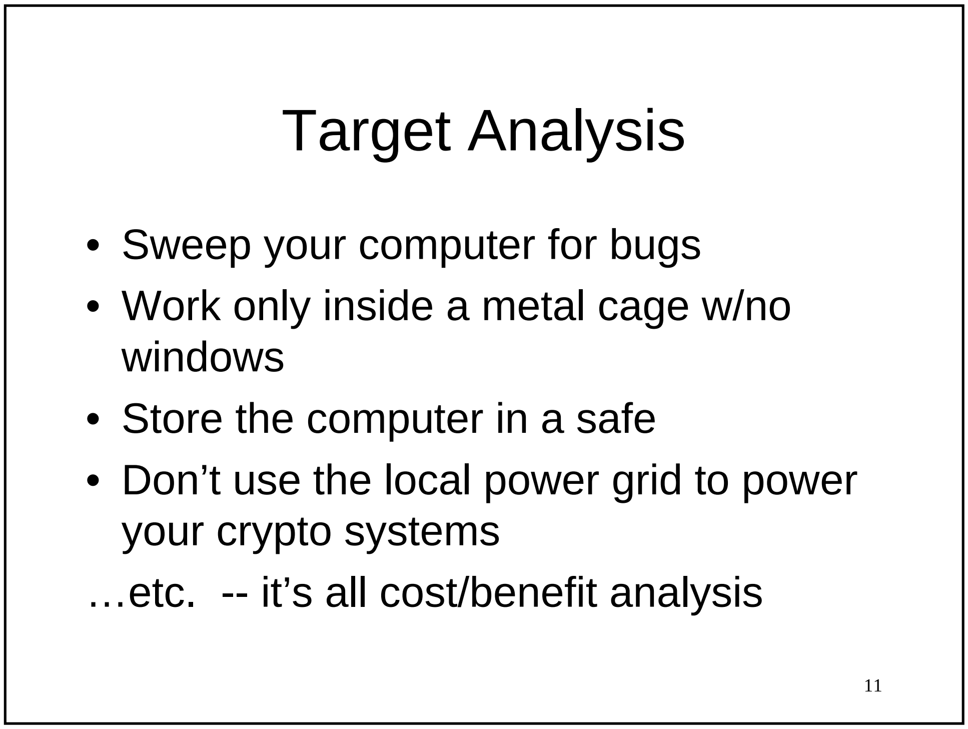


RF emissions (TEMPEST)

Phone taps

Laser bounce audio bugs

Radio audio bugs

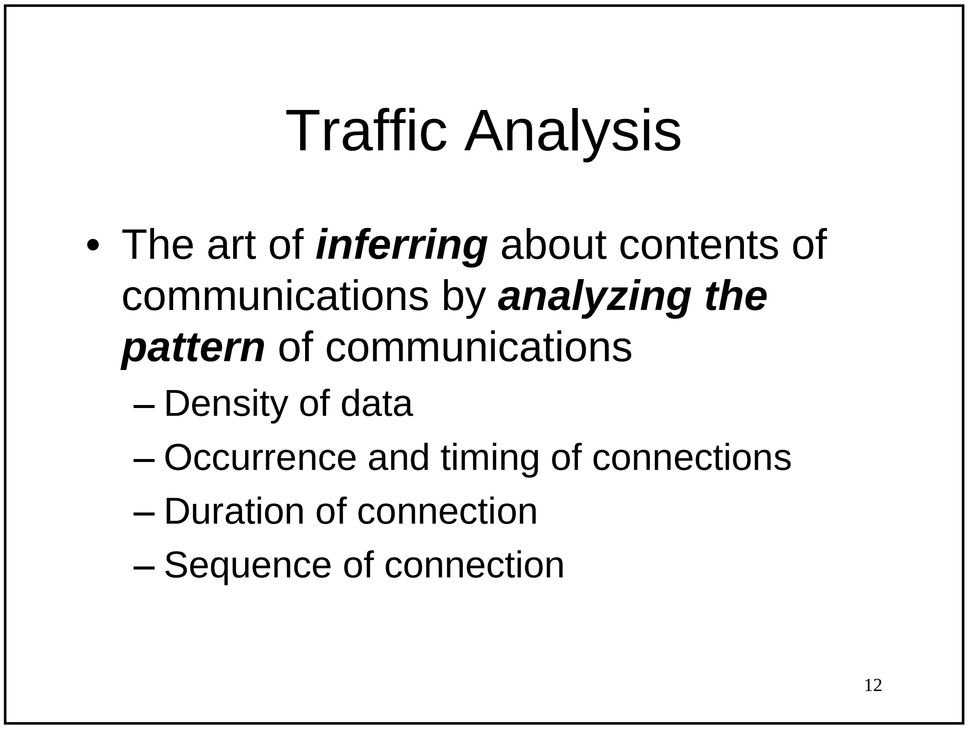


Target Analysis

- Sweep your computer for bugs
- Work only inside a metal cage w/no windows
- Store the computer in a safe
- Don't use the local power grid to power your crypto systems

…etc. -- it's all cost/benefit analysis

11

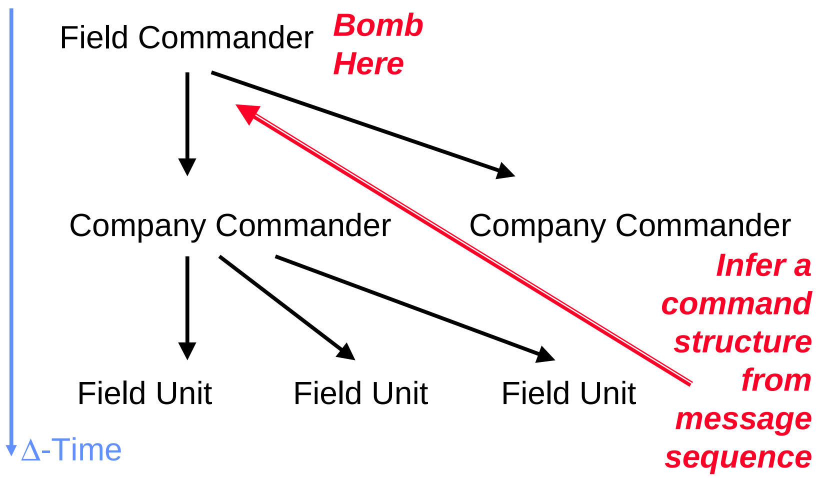


Traffic Analysis

- The art of ***inferring*** about contents of communications by ***analyzing the pattern*** of communications
  - Density of data
  - Occurrence and timing of connections
  - Duration of connection
  - Sequence of connection

12

Traffic Analysis



Bomb Here

Infer a command structure from message sequence

Traffic Analysis in Open Networks

- In open networks the majority of traffic is in the clear (unsecured)

… Therefore securing it becomes a dead giveaway to the traffic analyst!

- Ideally your secure communications will somehow look like unsecure communications or get lost in the noise

Traffic Analysis in Open Networks

- Incidentally, US law enforcement appears to be building an argument around a mindset that “if it’s encrypted that indicates that someone is probably doing something they shouldn’t”
  - I.e.: ***Honest*** people don’t ***need*** secure communications

Traffic Analysis: Example

- Terrorist hit in Paris*
- French intelligence agency correlates
  - All payphone calls near kill zone
  - Calls within “time window” of kill
  - Calls to another payphone that makes a call outside of France within a 20 minute period
  - Iranian agent in south of France is caught

*Amazingly, this was reported in Time magazine

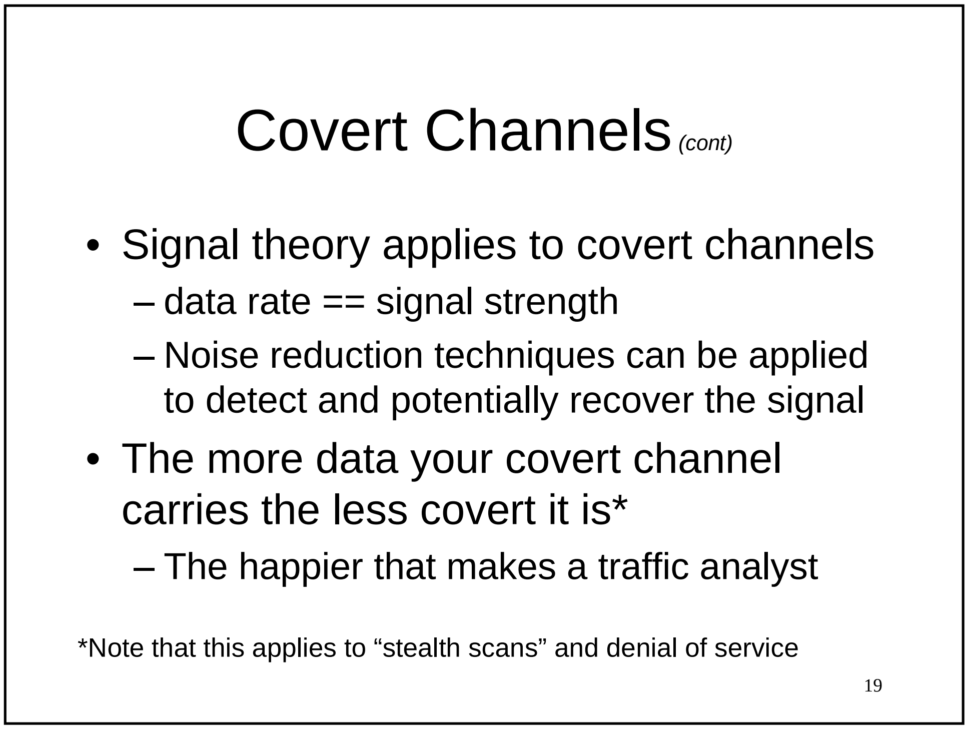
Traffic Analysis: Internet

- Identify software pirates by correlating file download activity
  - Large size files
  - Download rate
  - Frequency of particular files
  - Correlate file sizes/volumes across networks and you can backtrack users*

*Almost nobody keeps good enough logs to do this

Covert Channels

- Low data-rate communications encoded and hidden within another communication
  - Computers are great for this because they are patient!
- Example: Let’s say we agree that if I hit your web site within an hour, it’s a 1. If not it’s a 0. I can send 24 bits/day.

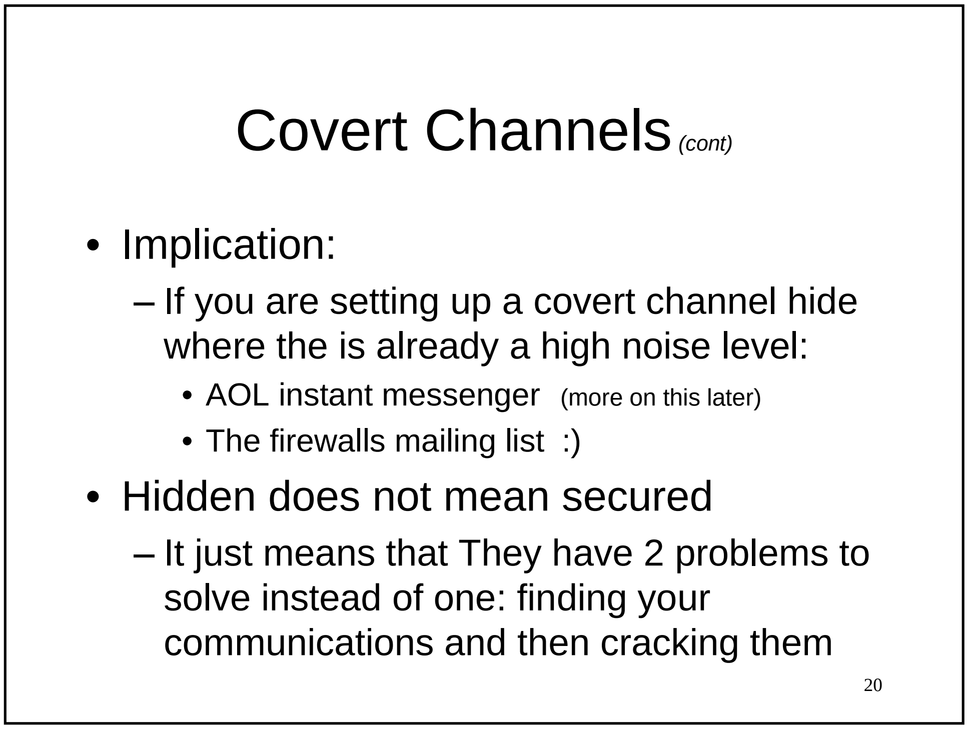


Covert Channels *(cont)*

- Signal theory applies to covert channels
  - – data rate == signal strength
  - – Noise reduction techniques can be applied to detect and potentially recover the signal
- The more data your covert channel carries the less covert it is*
  - – The happier that makes a traffic analyst

*Note that this applies to “stealth scans” and denial of service

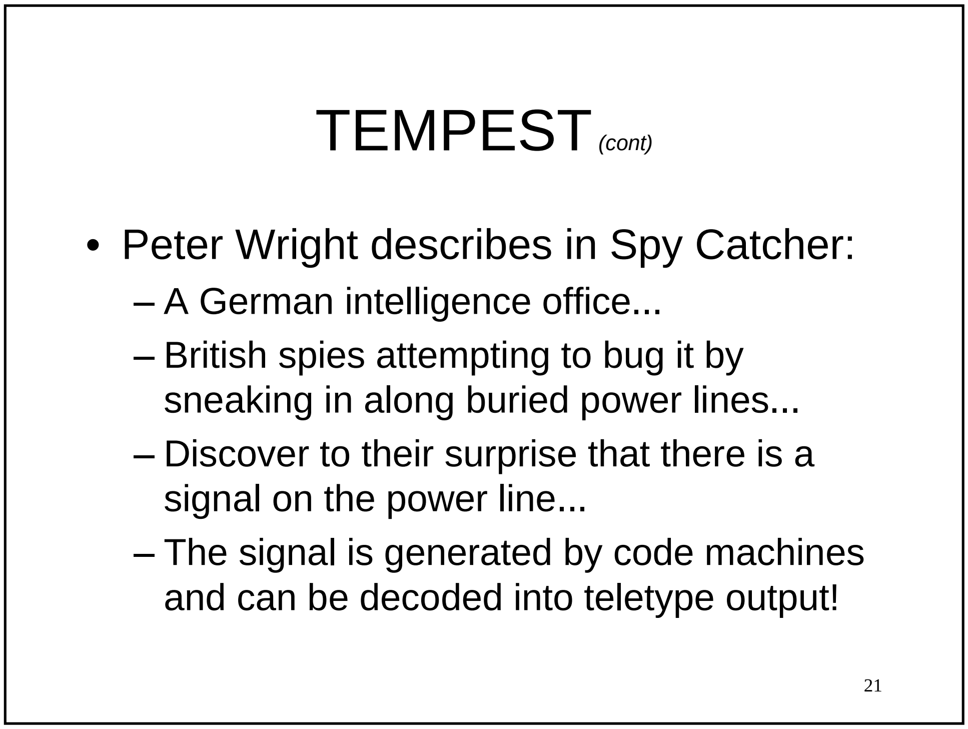
19



Covert Channels *(cont)*

- Implication:
  - – If you are setting up a covert channel hide where the is already a high noise level:
    - AOL instant messenger (more on this later)
    - The firewalls mailing list :)
- Hidden does not mean secured
  - – It just means that They have 2 problems to solve instead of one: finding your communications and then cracking them

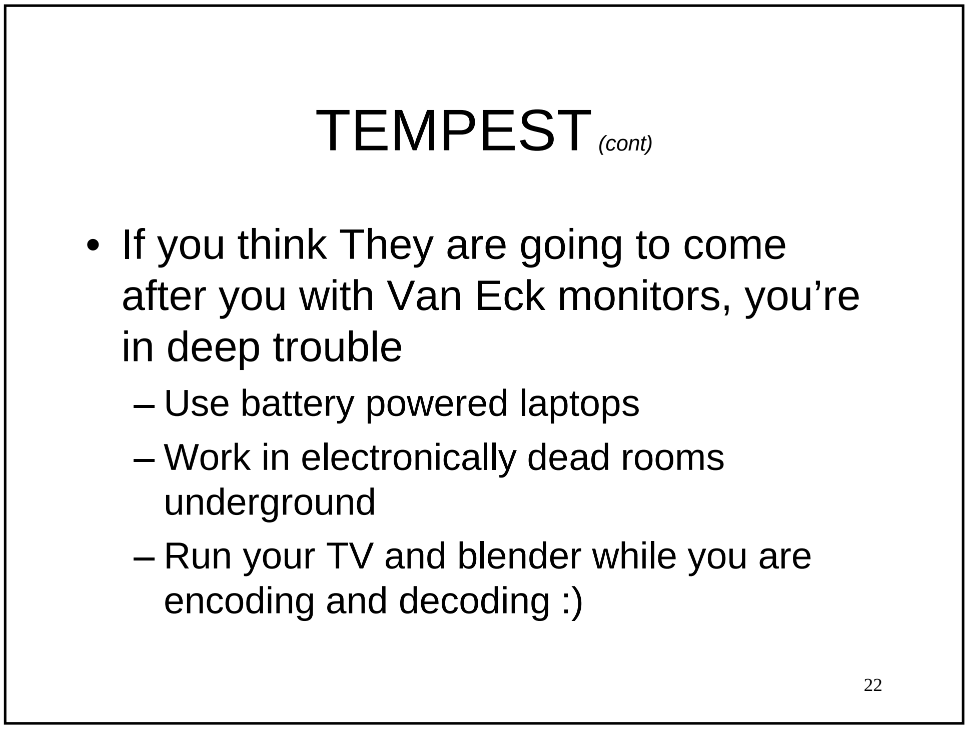
20



TEMPEST *(cont)*

- Peter Wright describes in Spy Catcher:
  - – A German intelligence office...
  - – British spies attempting to bug it by sneaking in along buried power lines...
  - – Discover to their surprise that there is a signal on the power line...
  - – The signal is generated by code machines and can be decoded into teletype output!

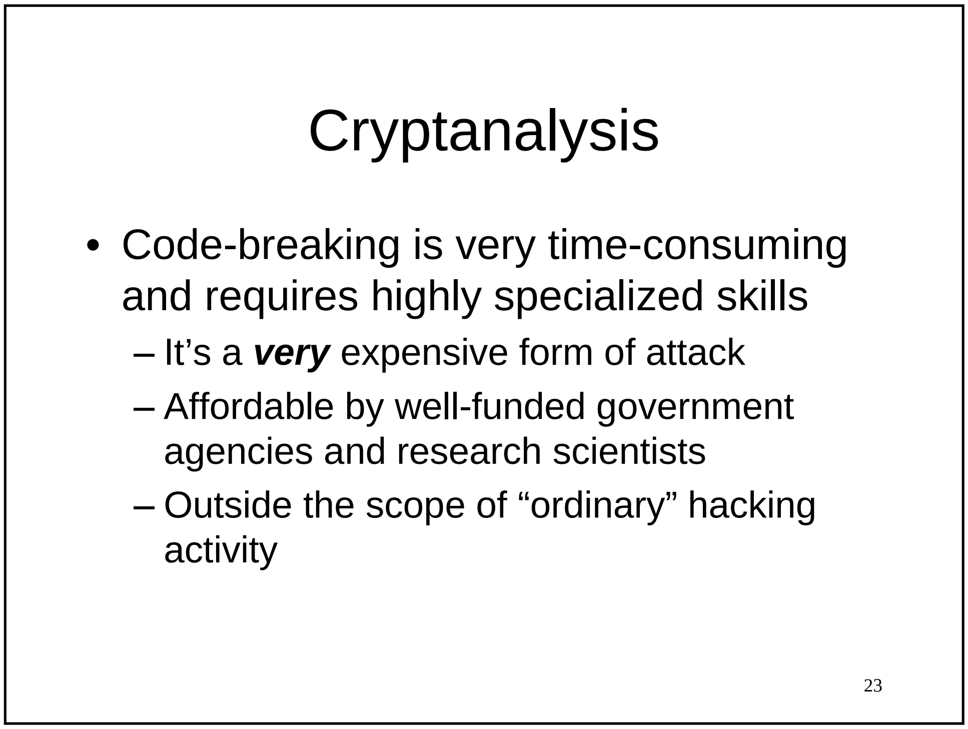
21



TEMPEST *(cont)*

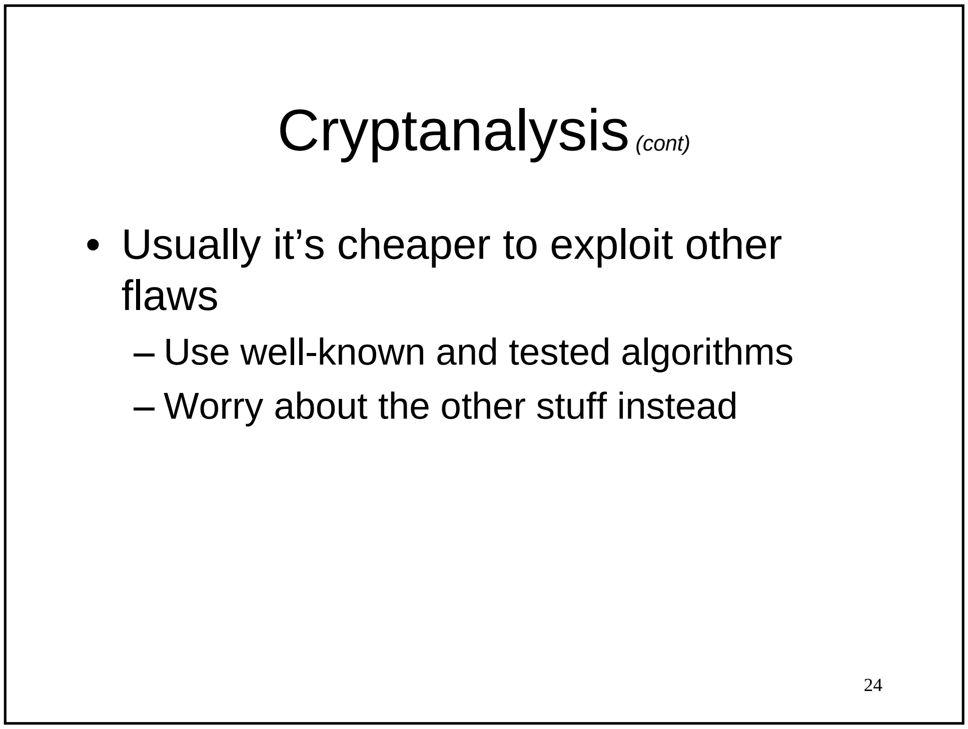
- If you think They are going to come after you with Van Eck monitors, you're in deep trouble
  - – Use battery powered laptops
  - – Work in electronically dead rooms underground
  - – Run your TV and blender while you are encoding and decoding :)

22



Cryptanalysis

- Code-breaking is very time-consuming and requires highly specialized skills
  - It’s a ***very*** expensive form of attack
  - Affordable by well-funded government agencies and research scientists
  - Outside the scope of “ordinary” hacking activity



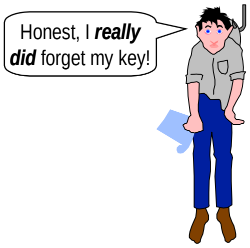
Cryptanalysis *(cont)*

- Usually it’s cheaper to exploit other flaws
  - Use well-known and tested algorithms
  - Worry about the other stuff instead

Rubber Hose Cryptanalysis

- If your communications security is so good They can't break it…

….the only thing left for Them to break is ***you***



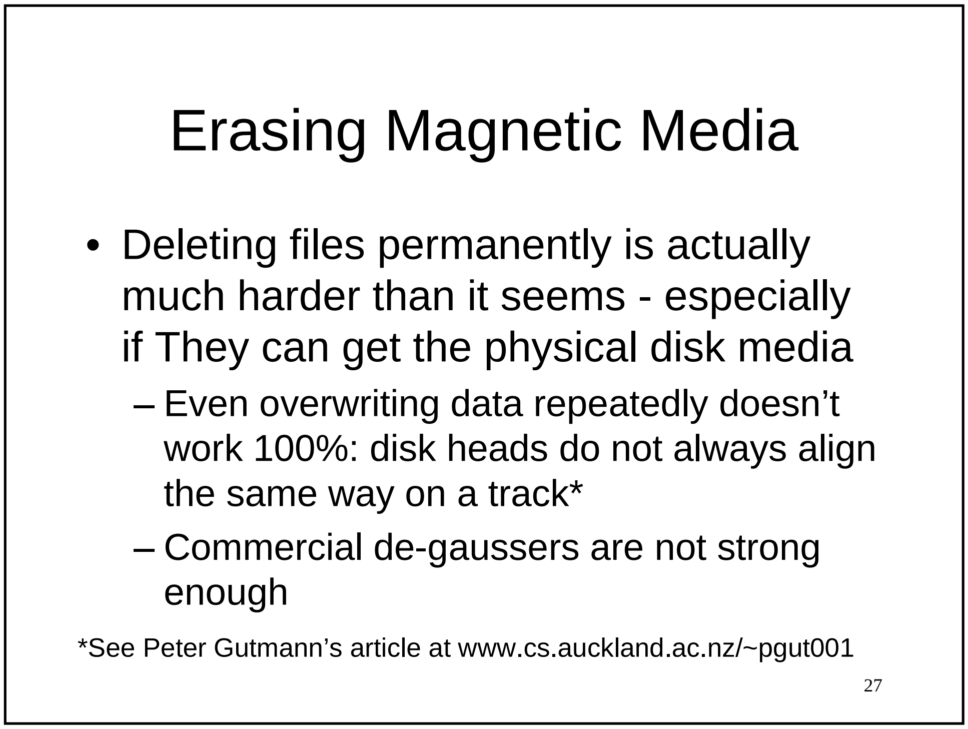
25

Key Purchase Attack

- There is no castle so strong that it cannot be overthrown by money
  - *Cicero*

… How valuable is your data?

26

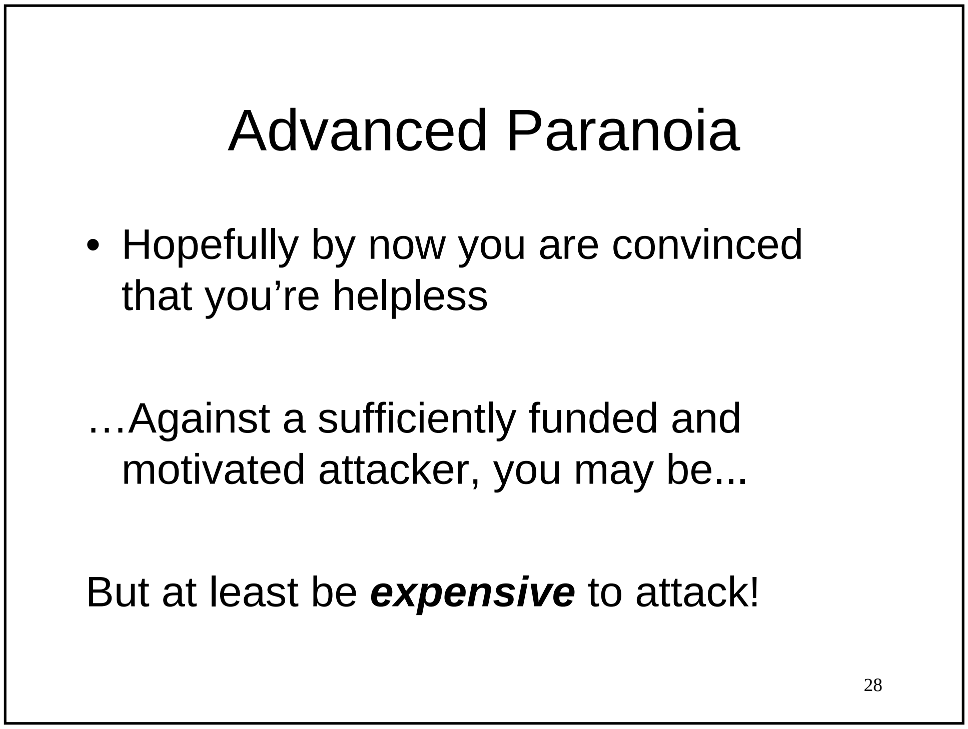


Erasing Magnetic Media

- Deleting files permanently is actually much harder than it seems - especially if They can get the physical disk media
  – Even overwriting data repeatedly doesn’t work 100%: disk heads do not always align the same way on a track*
  – Commercial de-gaussers are not strong enough

*See Peter Gutmann’s article at www.cs.auckland.ac.nz/~pgut001

27



Advanced Paranoia

- Hopefully by now you are convinced that you’re helpless

…Against a sufficiently funded and motivated attacker, you may be...

But at least be ***expensive*** to attack!

28

Goofy Comsec Stories: 1

- Peter Wright tells of Egyptians using a Hagelin rotor-based cipher machine
  - British agents place a bug in the code room, posing as telco workers
  - Whenever the Egyptians change their keys the British listeners count the *>click<* sounds of the rotors being set
  - Reduces the strength of the cipher to a few minutes' guesswork

Goofy Comsec Stories: 2

- Peter Wright tells of British embassy staff using one time pads in a secured code room
  - Russians plant an audio bug in the room
  - British cipher clerk reads the message aloud as another enciphers it one letter at a time
  - An unbreakable cryptosystem is completely sidestepped

Goofy Comsec Stories: 3

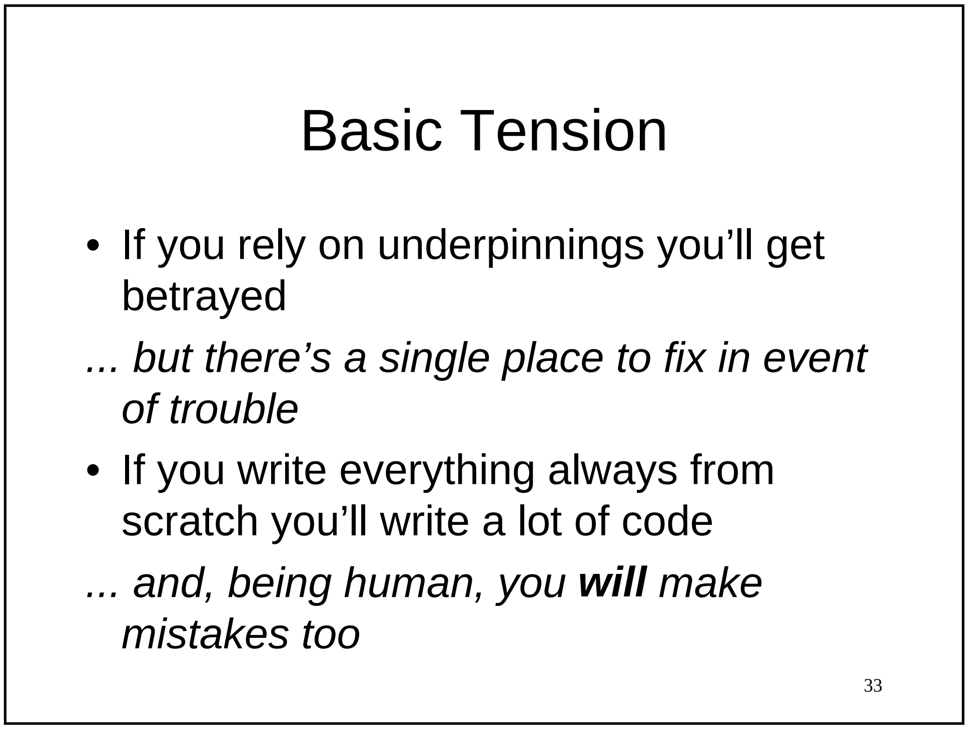
- Soviet agents subvert an NSA employee whose job it is to destroy classified documents
  - Since the documents are to be destroyed there is no audit trail for futher access
  - Instead of destroying them he sells them

31

Security Leverage

- Build on top of secure underpinnings
- Worry less about what’s going on below your horizon
- Sometimes your underpinnings may betray you!
  - ex: *syslog()* buffer overrun affects firewall toolkit, sendmail, etc.

32



Basic Tension

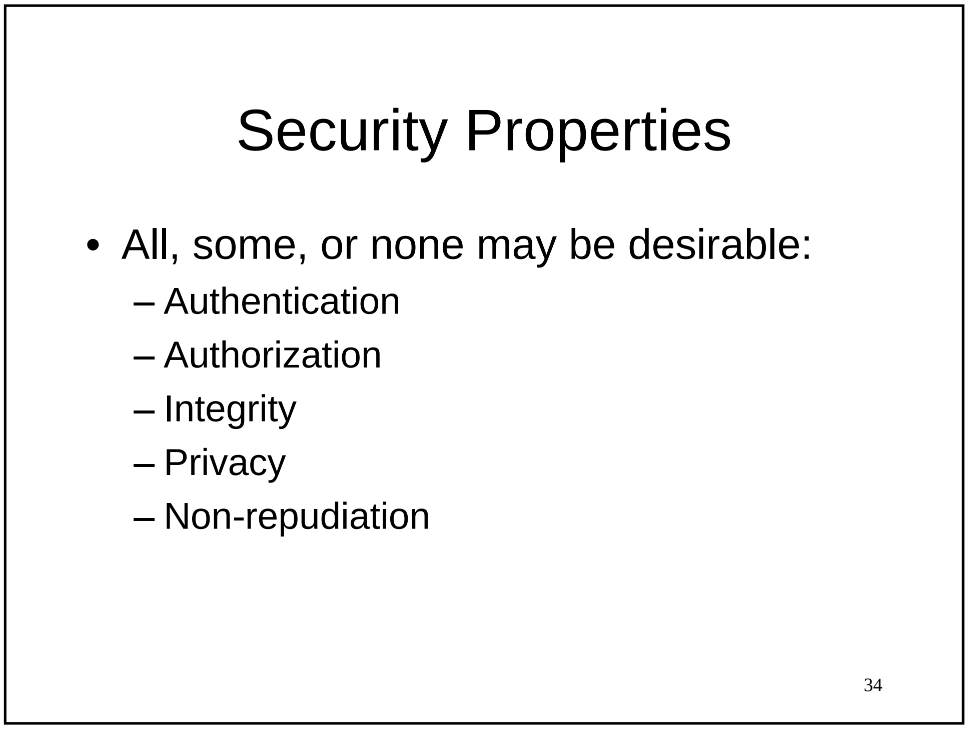
- If you rely on underpinnings you’ll get betrayed

... but there’s a single place to fix in event of trouble

- If you write everything always from scratch you’ll write a lot of code

... and, being human, you ***will*** *make mistakes too*

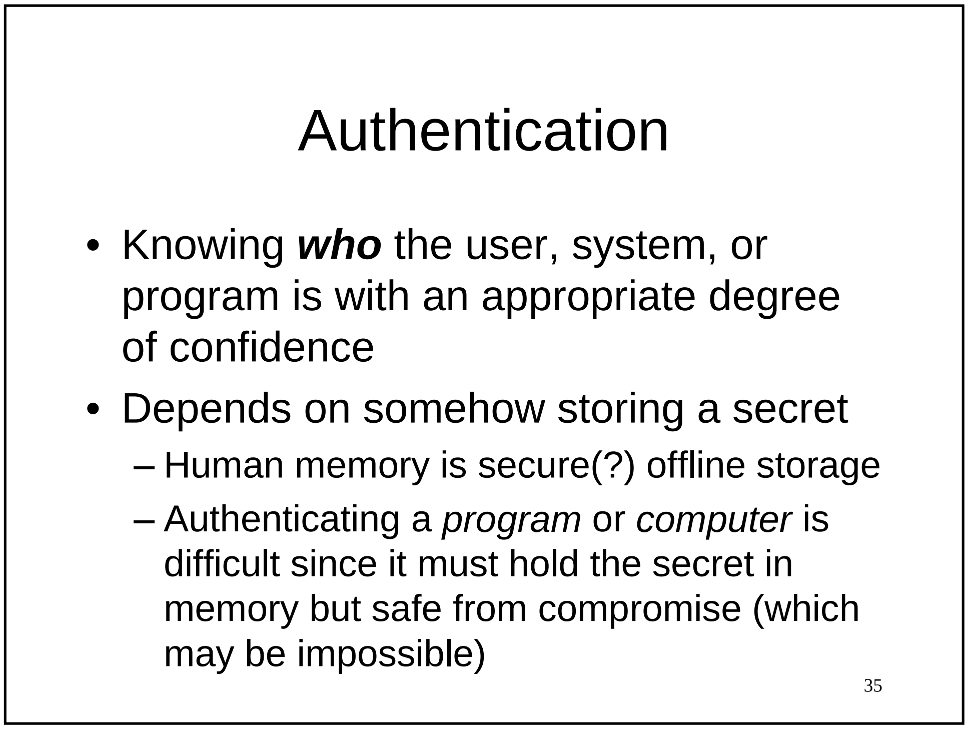
33



Security Properties

- All, some, or none may be desirable:
  - Authentication
  - Authorization
  - Integrity
  - Privacy
  - Non-repudiation

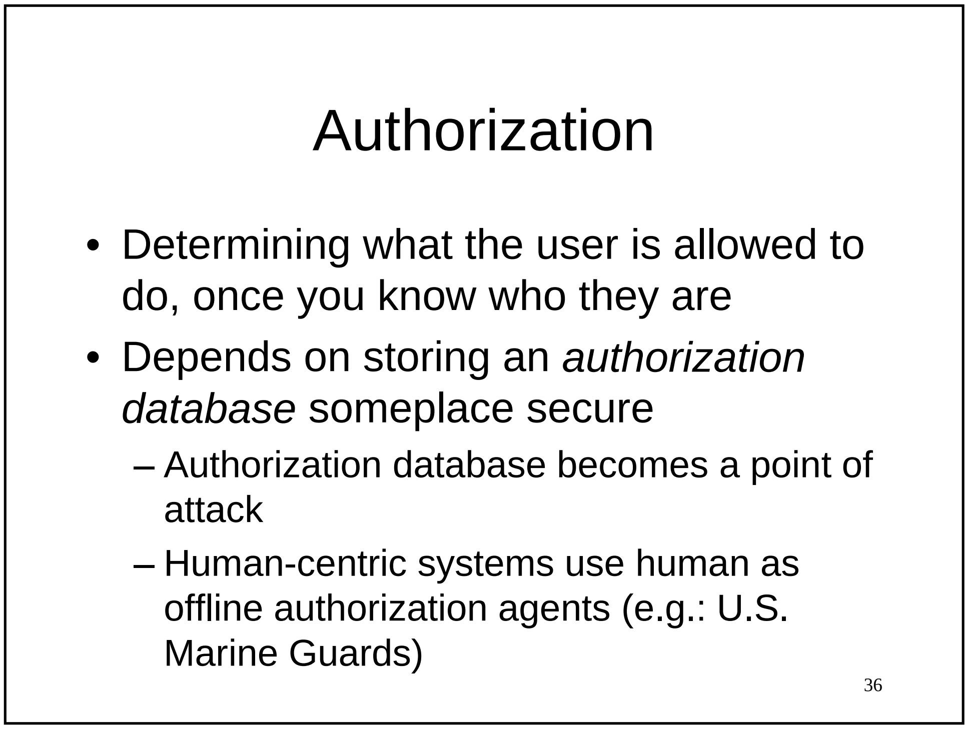
34



Authentication

- Knowing ***who*** the user, system, or program is with an appropriate degree of confidence
- Depends on somehow storing a secret
  - Human memory is secure(?) offline storage
  - Authenticating a *program* or *computer* is difficult since it must hold the secret in memory but safe from compromise (which may be impossible)

35



Authorization

- Determining what the user is allowed to do, once you know who they are
- Depends on storing an *authorization database* someplace secure
  - Authorization database becomes a point of attack
  - Human-centric systems use human as offline authorization agents (e.g.: U.S. Marine Guards)

36

Integrity

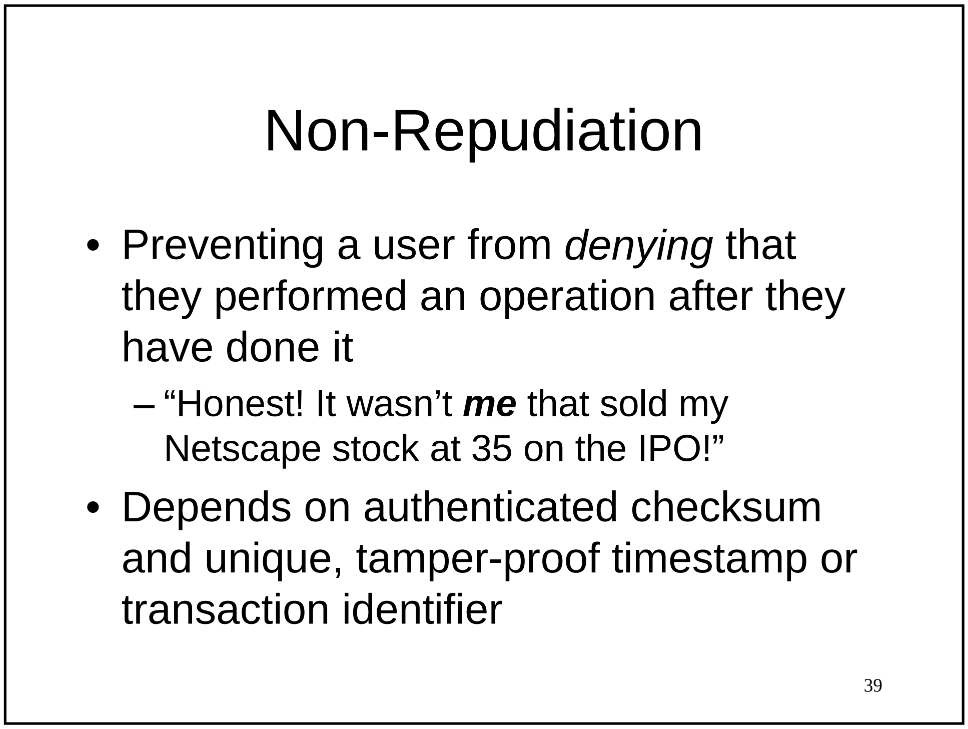
- Knowing that important data hasn’t been altered by accident or on purpose
- Usually relies on some kind of checksum or other one-way function
  - For high value transactions an authenticated checksum is needed
  - Important because computers lack common sense unless it is programmed in

37

Privacy

- Ensuring that data is not disclosed to unauthorized users
- Depends on careful data flow or encryption
  - If encryption is used the encryption *key* becomes the target of attack
  - In order to *use* the data it must be decrypted someplace safe

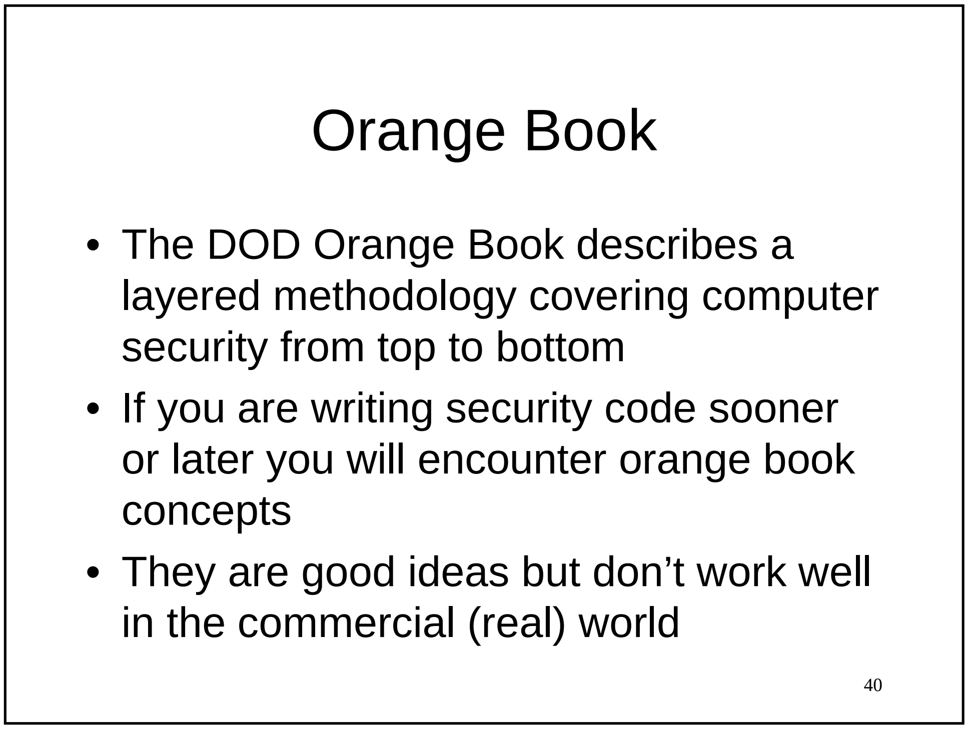
38



Non-Repudiation

- Preventing a user from *denying* that they performed an operation after they have done it
  - “Honest! It wasn’t ***me*** that sold my Netscape stock at 35 on the IPO!”
- Depends on authenticated checksum and unique, tamper-proof timestamp or transaction identifier

39



Orange Book

- The DOD Orange Book describes a layered methodology covering computer security from top to bottom
- If you are writing security code sooner or later you will encounter orange book concepts
- They are good ideas but don’t work well in the commercial (real) world

40

Important O.B. Concepts

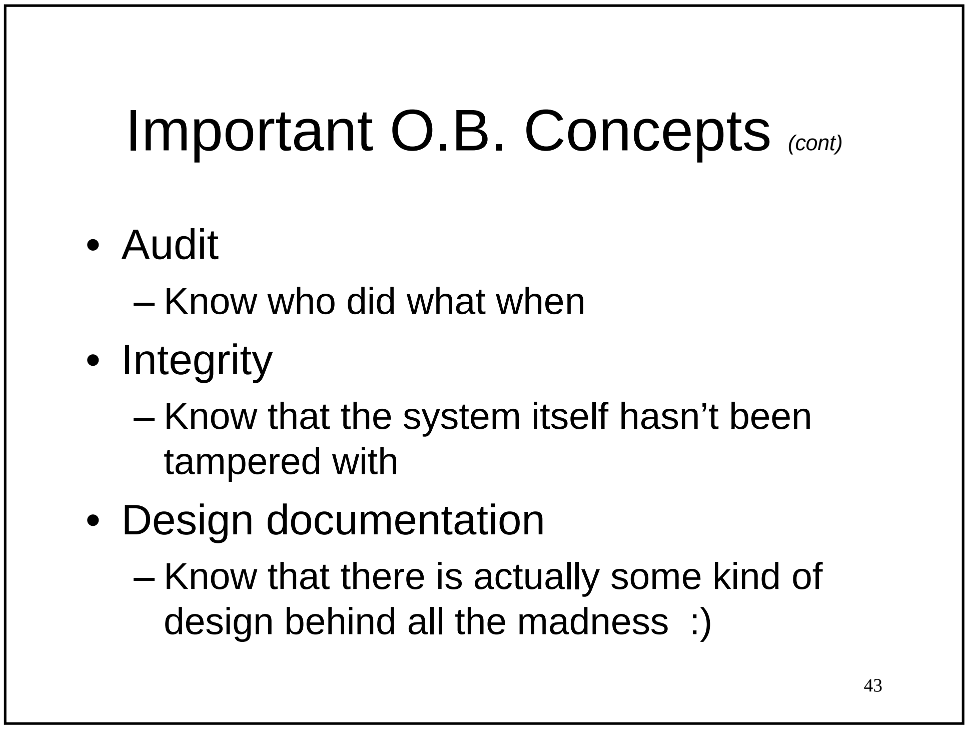
- Assurance - Knowing that the software is secure
  - Based on its design
  - Based on review
  - Based on the fact that it is built atop of other high-assurance components
- These are laudable goals for any software design

41

Important O.B. Concepts *(cont)*

- Discretionary and Mandatory access control
  - Each object has permissions associated with it that can be applied by the operating system to control access
  - The means for doing so must be high assurance

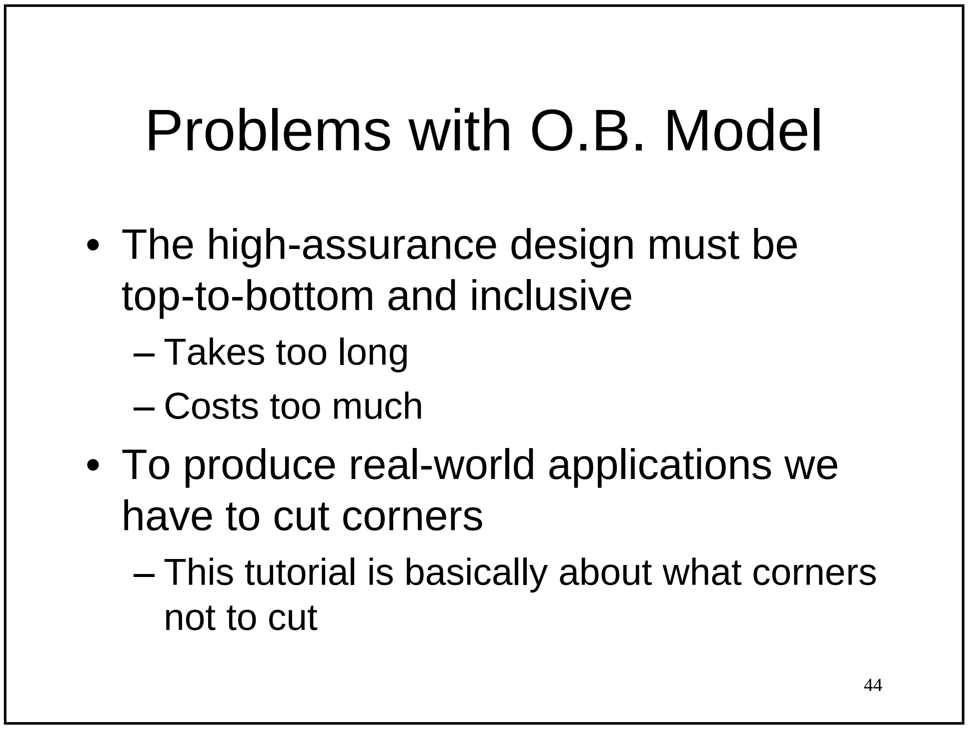
42



Important O.B. Concepts *(cont)*

- Audit
  - – Know who did what when
- Integrity
  - – Know that the system itself hasn’t been tampered with
- Design documentation
  - – Know that there is actually some kind of design behind all the madness :)

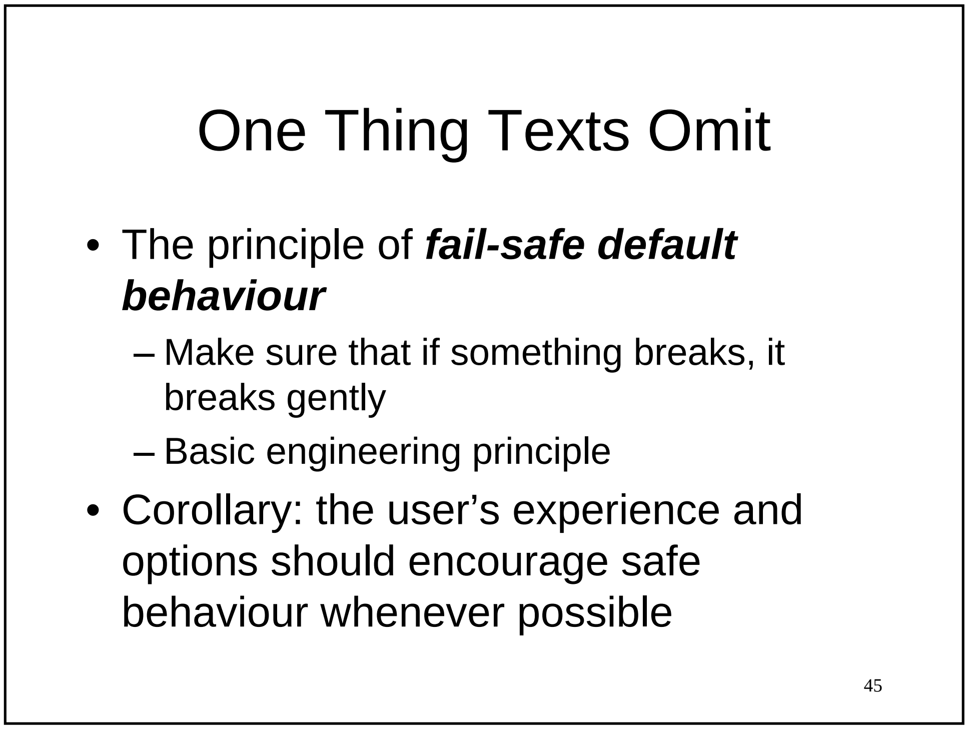
43



Problems with O.B. Model

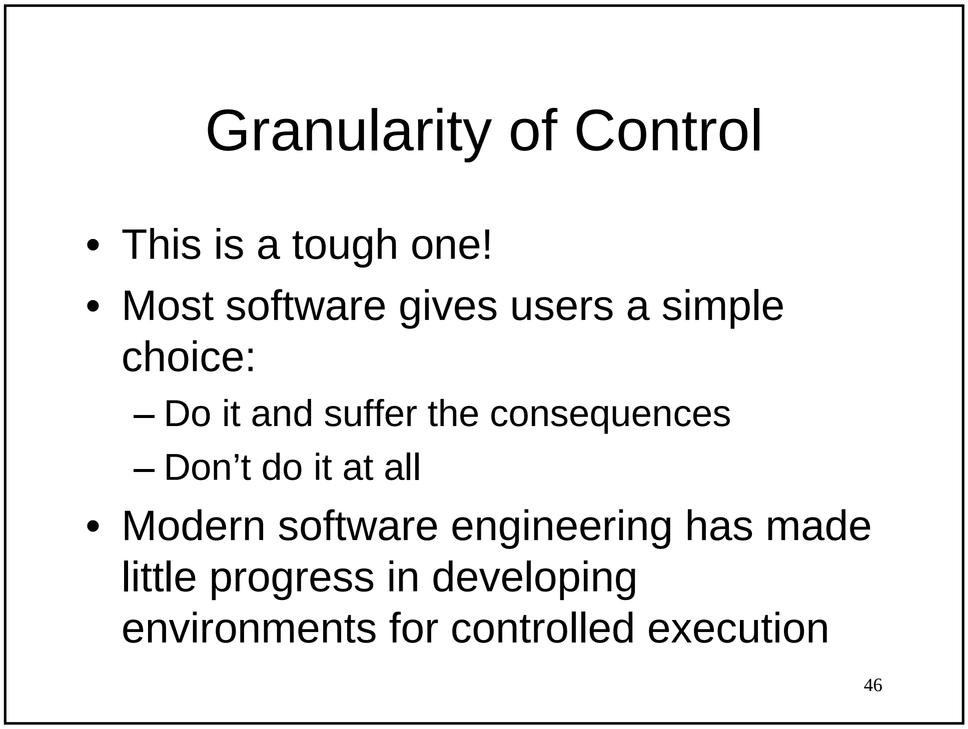
- The high-assurance design must be top-to-bottom and inclusive
  - – Takes too long
  - – Costs too much
- To produce real-world applications we have to cut corners
  - – This tutorial is basically about what corners not to cut

44



One Thing Texts Omit

- The principle of ***fail-safe default behaviour***
  - Make sure that if something breaks, it breaks gently
  - Basic engineering principle
- Corollary: the user’s experience and options should encourage safe behaviour whenever possible



Granularity of Control

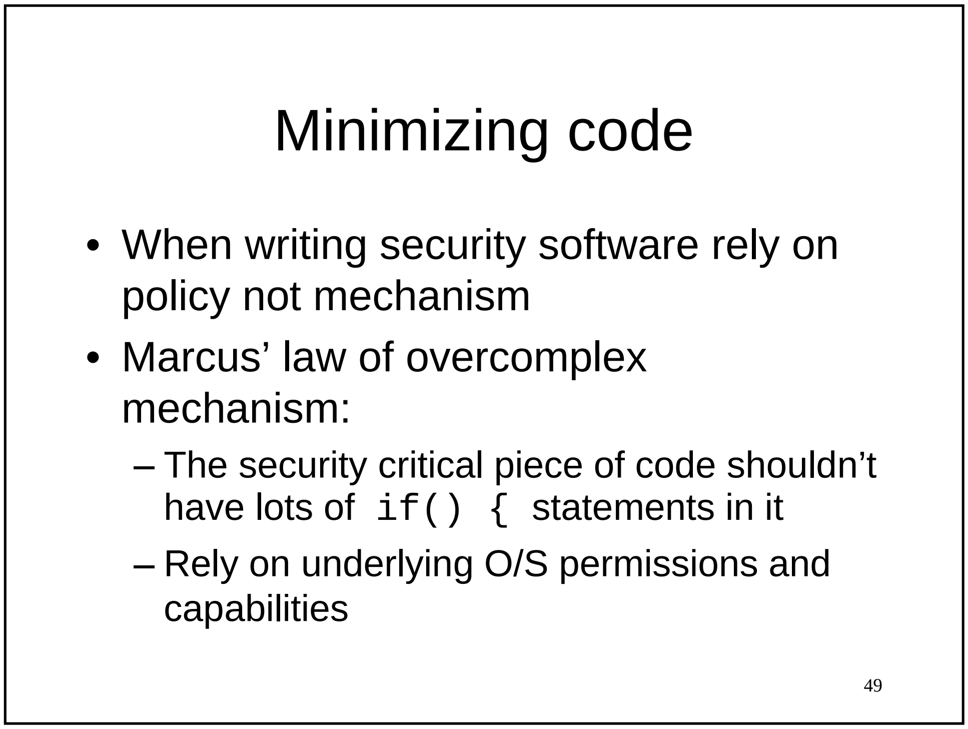
- This is a tough one!
- Most software gives users a simple choice:
  - Do it and suffer the consequences
  - Don’t do it at all
- Modern software engineering has made little progress in developing environments for controlled execution

Granularity of Control *(cont)*

Click Here
and ***something***
will happen

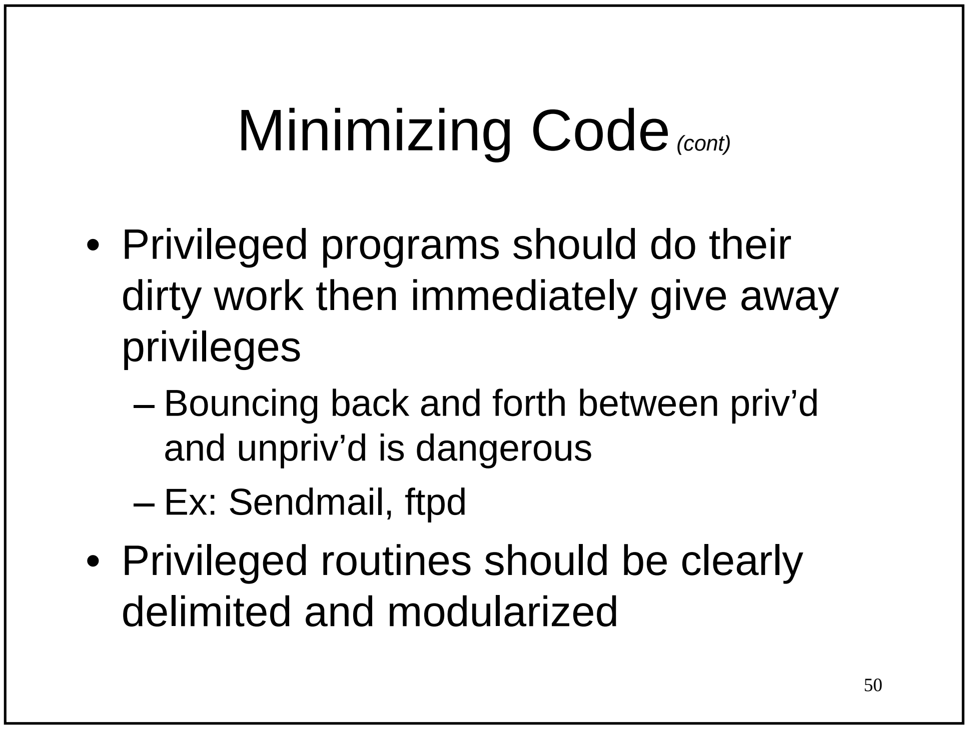
Security is a User Interface Problem

- It is possible to write secure software that hides the details from users
- Security problems (outside the scope of software flaws) often crop up when users try to avoid onerous security
- When designing security code ***stick with paradigms your users will understand*** (e.g., automatic teller)



Minimizing code

- When writing security software rely on policy not mechanism
- Marcus’ law of overcomplex mechanism:
  - The security critical piece of code shouldn’t have lots of `if() {` statements in it
  - Rely on underlying O/S permissions and capabilities



Minimizing Code *(cont)*

- Privileged programs should do their dirty work then immediately give away privileges
  - Bouncing back and forth between priv’d and unpriv’d is dangerous
  - Ex: Sendmail, ftpd
- Privileged routines should be clearly delimited and modularized

Minimizing Code *(cont)*

```
main(ac,av) /* From aftpd.c */
{
   ...
   /* sanity check command line args */
   /* THIS IS THE SECURITY CRITICAL CODE */
   if(chdir(homedir)) {
        syslog(LOG_NOTICE,"Aborting - chdir %s: %m",homedir);
        exit(1);
   }
   if(chroot(homedir)) {
        syslog(LOG_NOTICE,"Aborting - chroot %s: %m",homedir);
        exit(1);
   }
   /* END SECURITY CRITICAL CODE */
```

Taxonomies of flaws

- Software flaws tend to follow time-honored patterns
- Like cockroaches you can often find them in common places:
  - When crossing permission boundaries
  - When ignoring errors
  - Where important files are altered
  - Where other programs are run

Taxonomies of Flaws *(cont)*

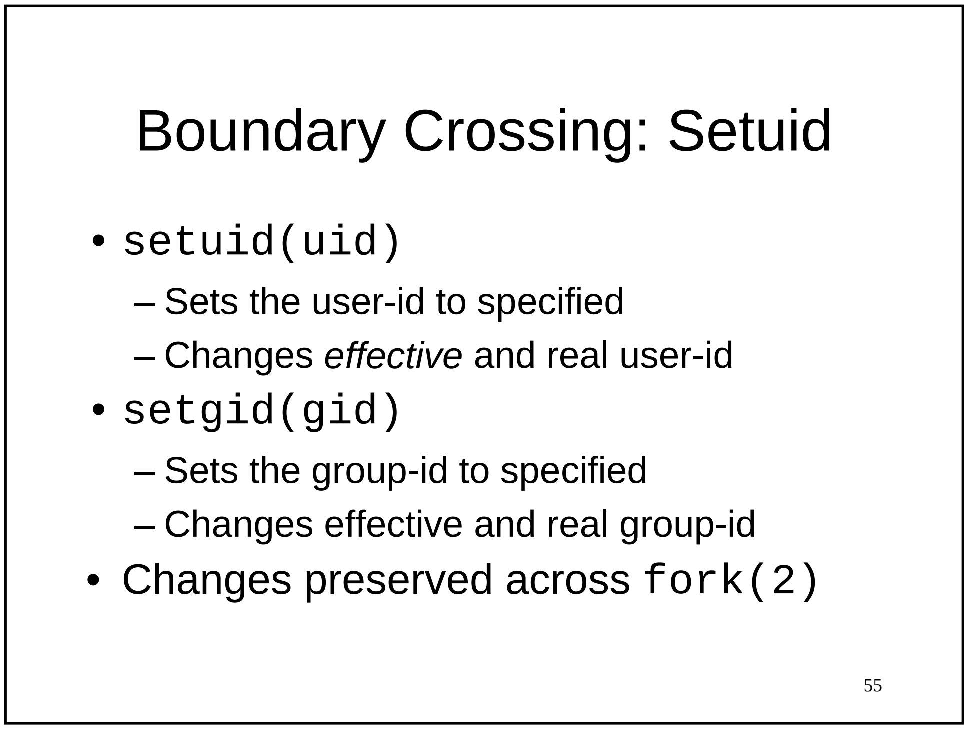
- Hackers also know taxonomies of flaws
  - Like structural mechanics you learn where weaknesses usually are and begin probing there
  - DOD software evaluation uses flaw taxonomy analysis to look for “the usual holes” and is usually shockingly successful
  - Hackers are also usually shockingly successful

53

Flaws: Crossing Boundaries

- Changing permissions state from higher to lower
  - Must be irreversible
  - Must be done carefully
  - Must be done for right reasons
- Changing from lower to higher
  - Must be done ***extremely*** cautiously
  - Cannot be “fooled” by user

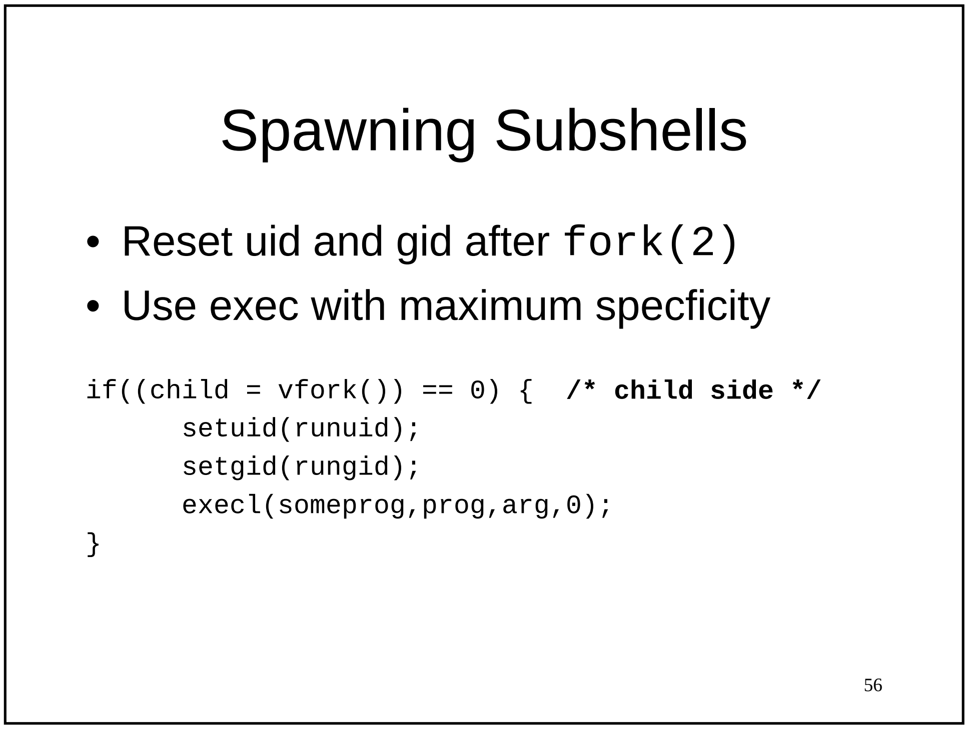
54



Boundary Crossing: Setuid

- `setuid(uid)`
  - Sets the user-id to specified
  - Changes *effective* and real user-id
- `setgid(gid)`
  - Sets the group-id to specified
  - Changes effective and real group-id
- Changes preserved across `fork(2)`

55

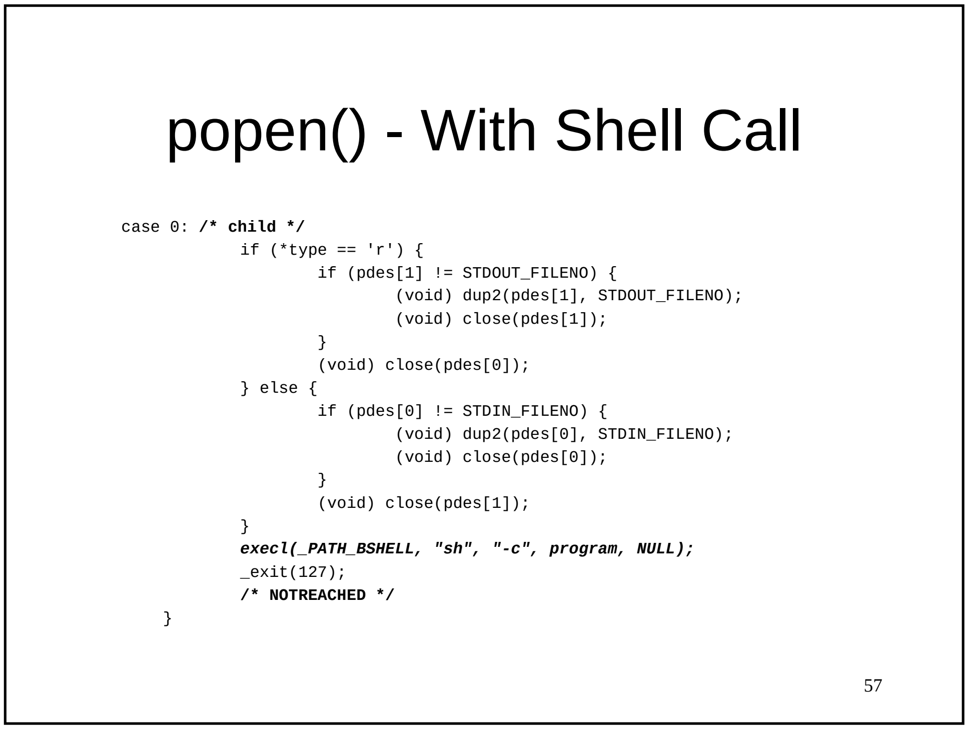


Spawning Subshells

- Reset uid and gid after `fork(2)`
- Use exec with maximum specficity

```
if((child = vfork()) == 0) {  /* child side */
      setuid(runuid);
      setgid(rungid);
      execl(someprog,prog,arg,0);
}
```

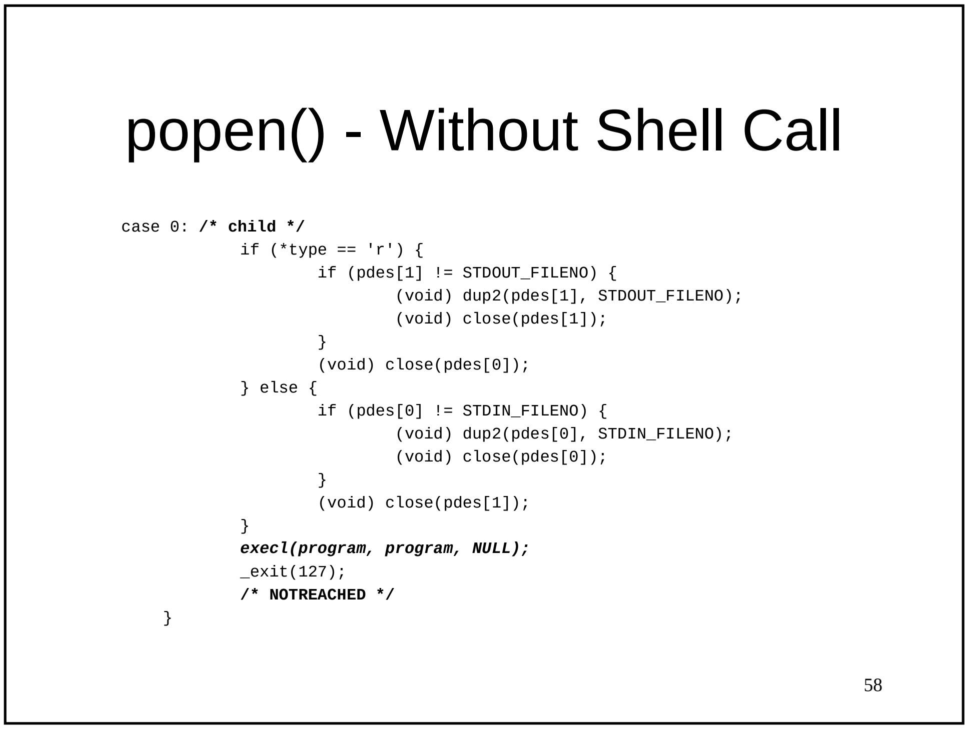
56



popen() - With Shell Call

```
case 0: /* child */
            if (*type == 'r') {
                    if (pdes[1] != STDOUT_FILENO) {
                            (void) dup2(pdes[1], STDOUT_FILENO);
                            (void) close(pdes[1]);
                    }
                    (void) close(pdes[0]);
            } else {
                    if (pdes[0] != STDIN_FILENO) {
                            (void) dup2(pdes[0], STDIN_FILENO);
                            (void) close(pdes[0]);
                    }
                    (void) close(pdes[1]);
            }
            execl(_PATH_BSHELL, "sh", "-c", program, NULL);
            _exit(127);
            /* NOTREACHED */
    }
```

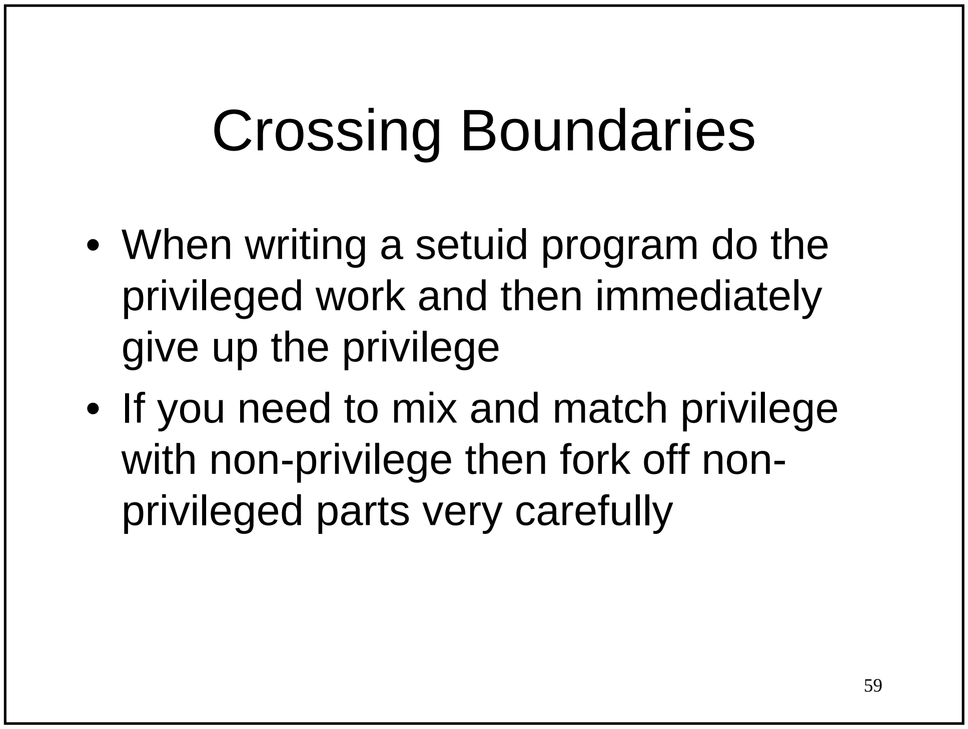
57



popen() - Without Shell Call

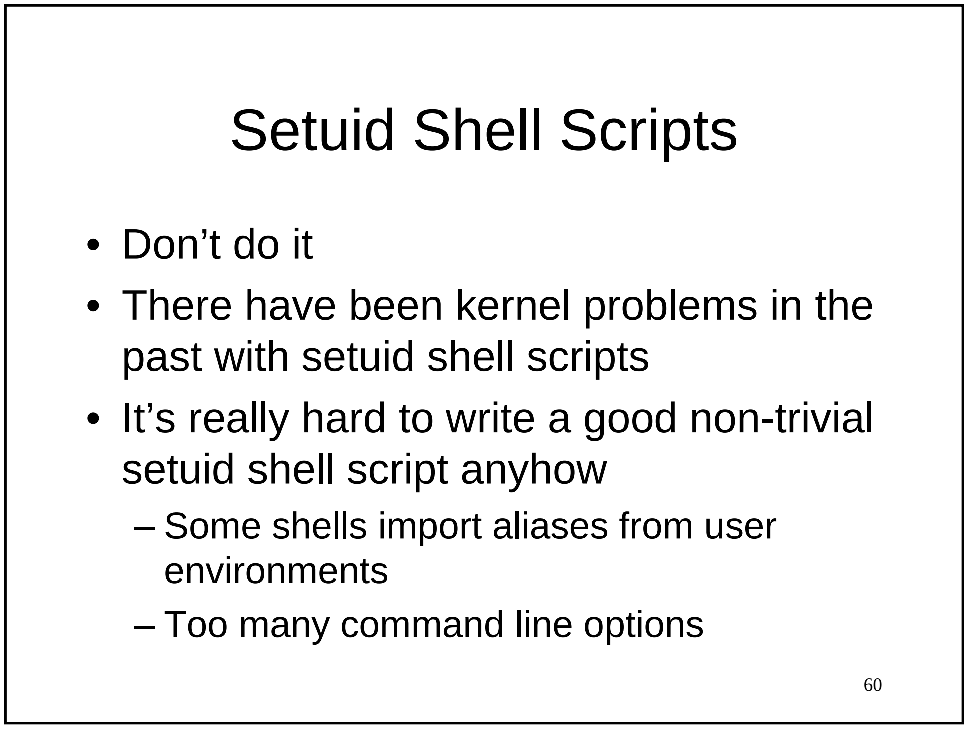
```
case 0: /* child */
            if (*type == 'r') {
                    if (pdes[1] != STDOUT_FILENO) {
                            (void) dup2(pdes[1], STDOUT_FILENO);
                            (void) close(pdes[1]);
                    }
                    (void) close(pdes[0]);
            } else {
                    if (pdes[0] != STDIN_FILENO) {
                            (void) dup2(pdes[0], STDIN_FILENO);
                            (void) close(pdes[0]);
                    }
                    (void) close(pdes[1]);
            }
            execl(program, program, NULL);
            _exit(127);
            /* NOTREACHED */
    }
```

58



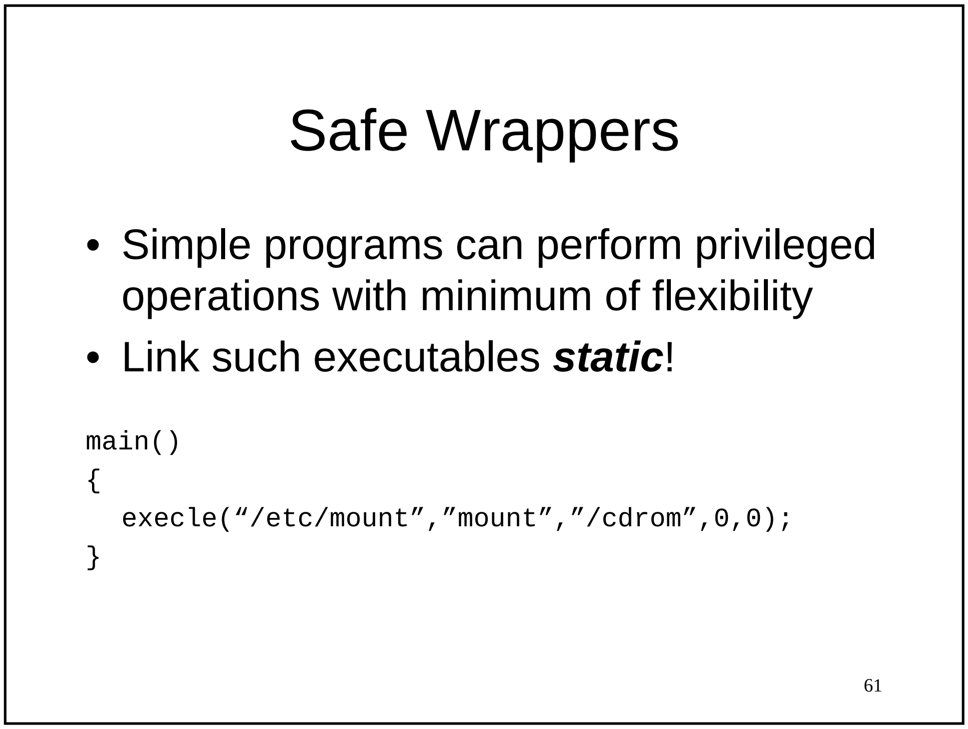
Crossing Boundaries

- When writing a setuid program do the privileged work and then immediately give up the privilege
- If you need to mix and match privilege with non-privilege then fork off non-privileged parts very carefully



Setuid Shell Scripts

- Don’t do it
- There have been kernel problems in the past with setuid shell scripts
- It’s really hard to write a good non-trivial setuid shell script anyhow
  - Some shells import aliases from user environments
  - Too many command line options

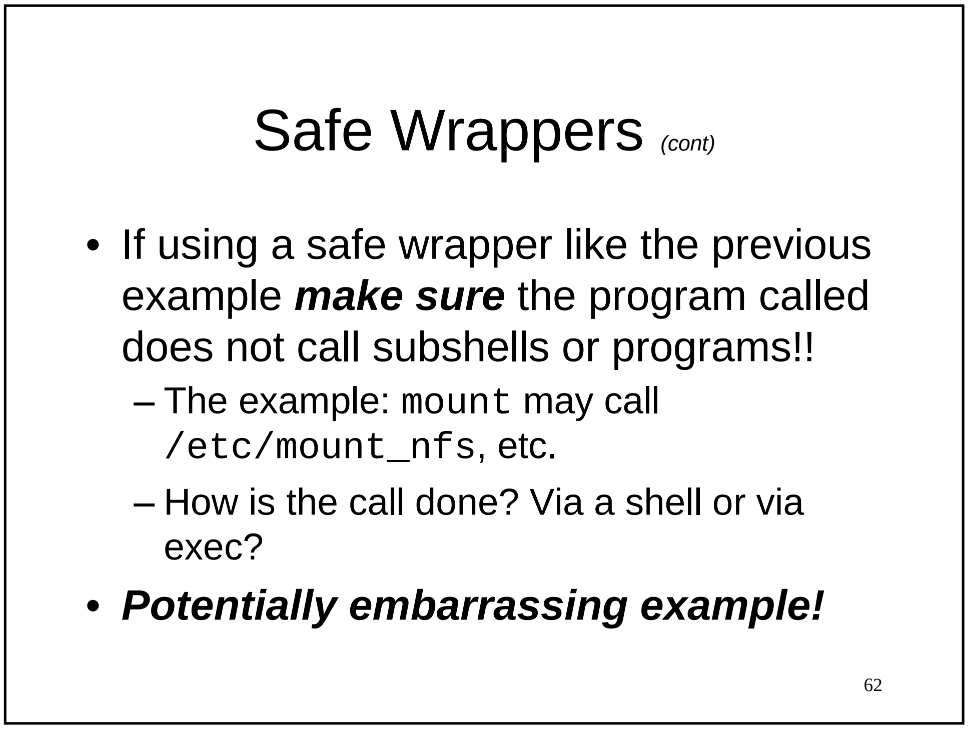


Safe Wrappers

- Simple programs can perform privileged operations with minimum of flexibility
- Link such executables ***static***!

```
main()
{
  execle(“/etc/mount”,”mount”,”/cdrom”,0,0);
}
```

61



Safe Wrappers *(cont)*

- If using a safe wrapper like the previous example ***make sure*** the program called does not call subshells or programs!!
  - The example: `mount` may call `/etc/mount_nfs`, etc.
  - How is the call done? Via a shell or via exec?
- ***Potentially embarrassing example!***

62

Safe Wrappers *(cont)*

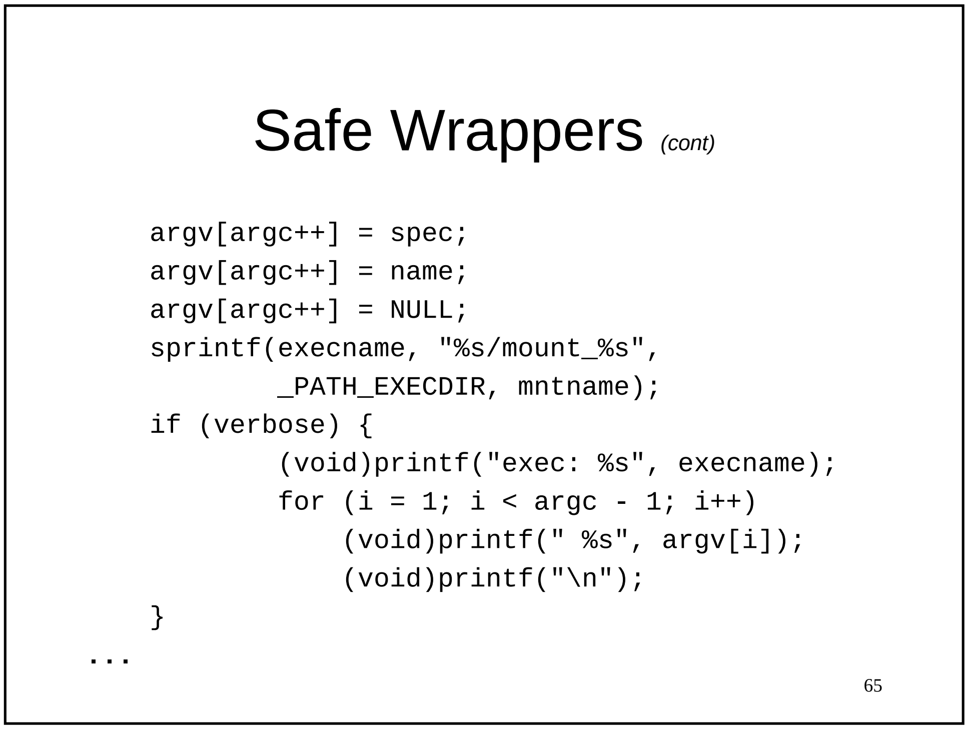
- On Solaris, we see something scary:

```
$ strings /etc/mount | more
...
%s/%s/%s
%s/%s/%s
%s -F %s
%s: cannot execute %s - permission denied
/sbin/sh
%s: cannot execute %s - permission denied
/sbin/sh
...
```

Safe Wrappers *(cont)*

- BSD (as usual) does it right:

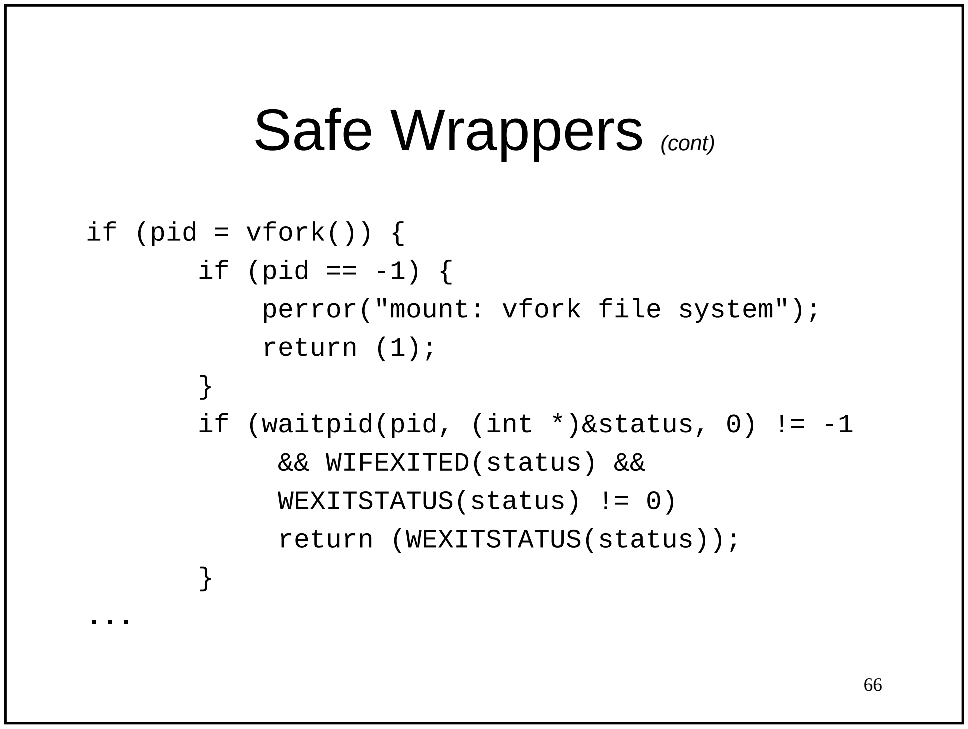
```
case MOUNT_MFS:
   default:
          argv[0] = mntname;
          argc = 1;
          if (flags) {
                  argv[argc++] = "-F";
                  sprintf(flagval, "%d", flags);
                  argv[argc++] = flagval;
          }
...
```



Safe Wrappers *(cont)*

```
    argv[argc++] = spec;
    argv[argc++] = name;
    argv[argc++] = NULL;
    sprintf(execname, "%s/mount_%s",
            _PATH_EXECDIR, mntname);
    if (verbose) {
            (void)printf("exec: %s", execname);
            for (i = 1; i < argc - 1; i++)
                (void)printf(" %s", argv[i]);
                (void)printf("\n");
    }
...
```

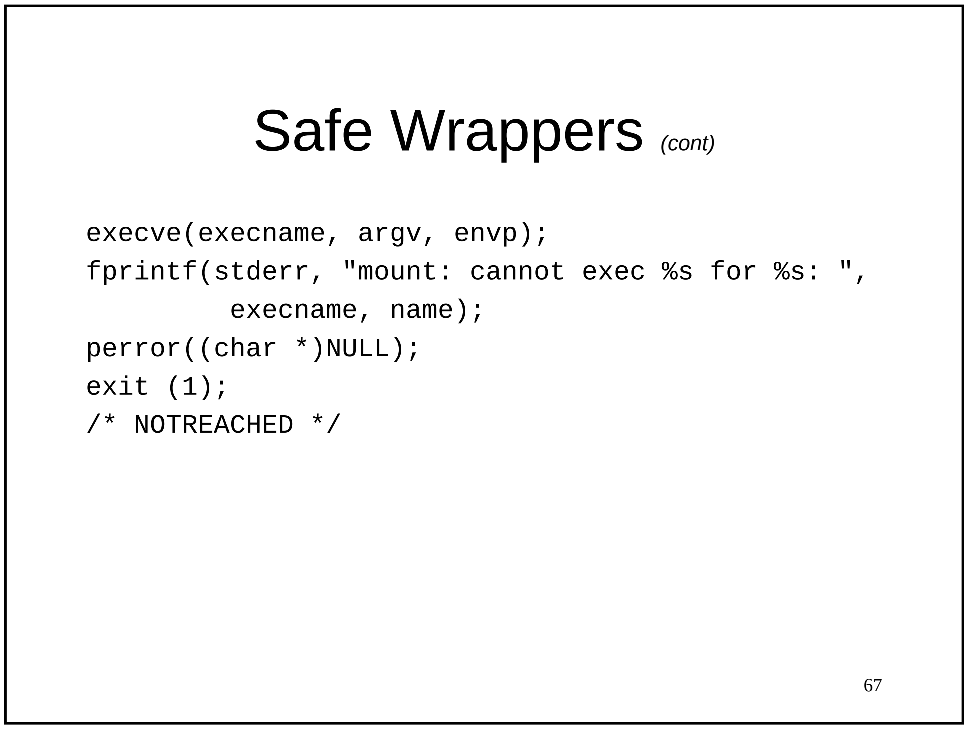
65



Safe Wrappers *(cont)*

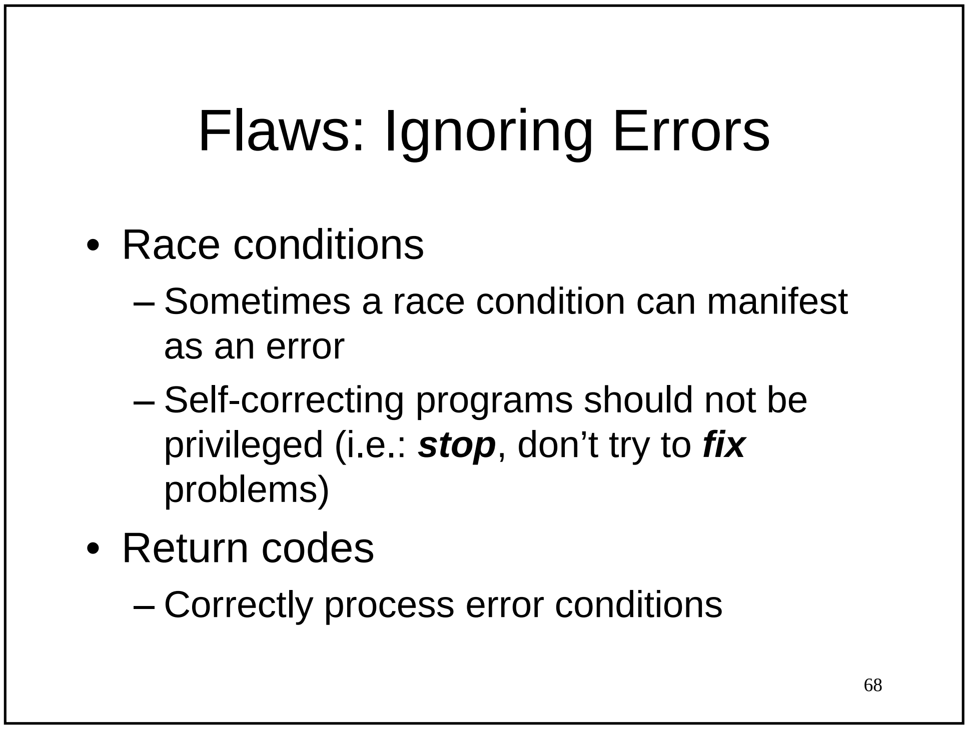
```
if (pid = vfork()) {
       if (pid == -1) {
           perror("mount: vfork file system");
           return (1);
       }
       if (waitpid(pid, (int *)&status, 0) != -1
            && WIFEXITED(status) &&
            WEXITSTATUS(status) != 0)
            return (WEXITSTATUS(status));
       }
...
```

66



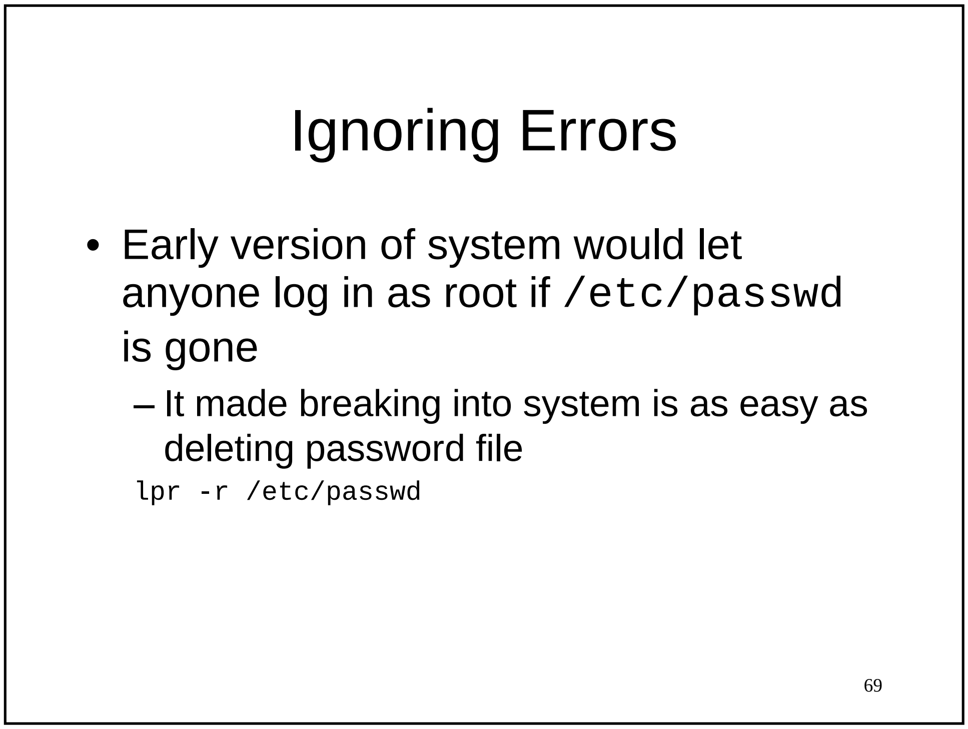
Safe Wrappers *(cont)*

```
execve(execname, argv, envp);
fprintf(stderr, "mount: cannot exec %s for %s: ",
         execname, name);
perror((char *)NULL);
exit (1);
/* NOTREACHED */
```



Flaws: Ignoring Errors

- Race conditions
  - Sometimes a race condition can manifest as an error
  - Self-correcting programs should not be privileged (i.e.: ***stop***, don’t try to ***fix*** problems)
- Return codes
  - Correctly process error conditions

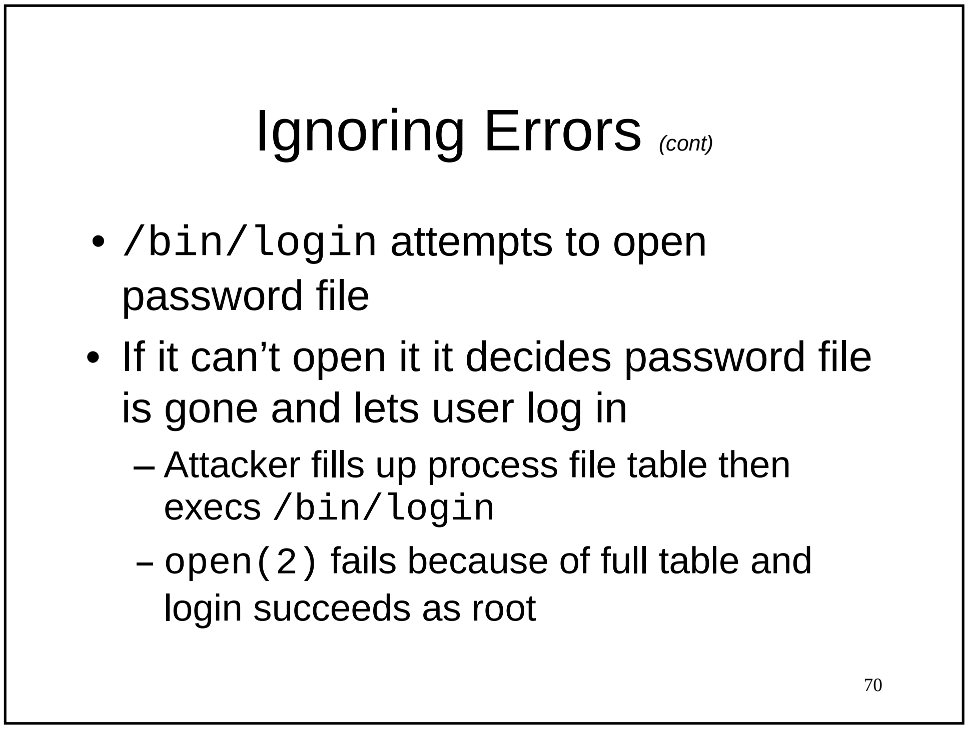


Ignoring Errors

- Early version of system would let anyone log in as root if `/etc/passwd` is gone
  - It made breaking into system is as easy as deleting password file

```
lpr -r /etc/passwd
```

69



Ignoring Errors *(cont)*

- `/bin/login` attempts to open password file
- If it can’t open it it decides password file is gone and lets user log in
  - Attacker fills up process file table then execs `/bin/login`
  - `open(2)` fails because of full table and login succeeds as root

70

Ignoring Errors *(cont)*

- ***Always*** check return codes
- ***Always*** scream if you detect an error
- ***Always*** stop processing when you detect an error
- Errno is your friend
  - Don't forget that `perror()` may reset errno when completed

Flaws: File Updates

- Writing files
  - If file is replaced so another is overwritten
  - If file will be input to another program
  - If file contains authorization data
- Reading files
  - If permission file can be replaced or altered
  - If file contents are displayed can be used to reveal hidden information

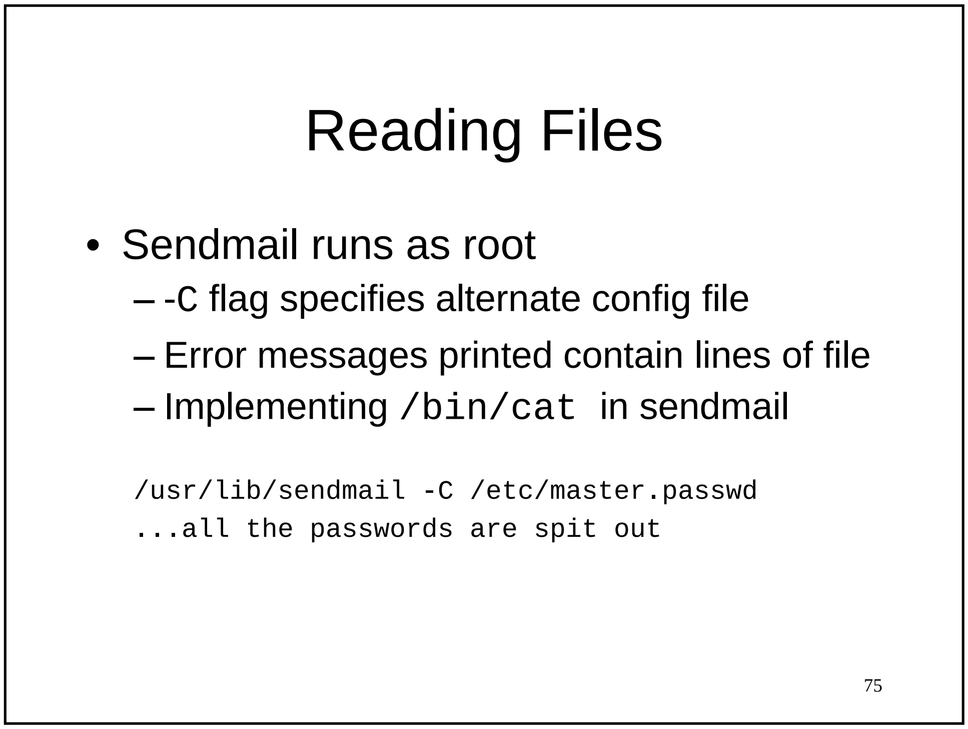
Writing Files

- ULTRIX 4.1 X-server runs as root and has a save options capability
  - Options file saved as ~/.Xoptions and chowned to user
  - No check made for symbolic link or ownership of overwrite

```
ln -s /etc/passwd ~/.Xoptions
# save options
vi /etc/passwd
```

Writing Files *(cont)*

- Common mistake is writing a file and changing its ownership/modes as root
  - write /tmp/foo
  - `chmod(“/tmp/foo”,0600);`
  - `chown(“/tmp/foo”,1234);`
  - What happens if /tmp/foo is renamed and replaced with a symlink to /etc/passwd between the chmod and the chown?

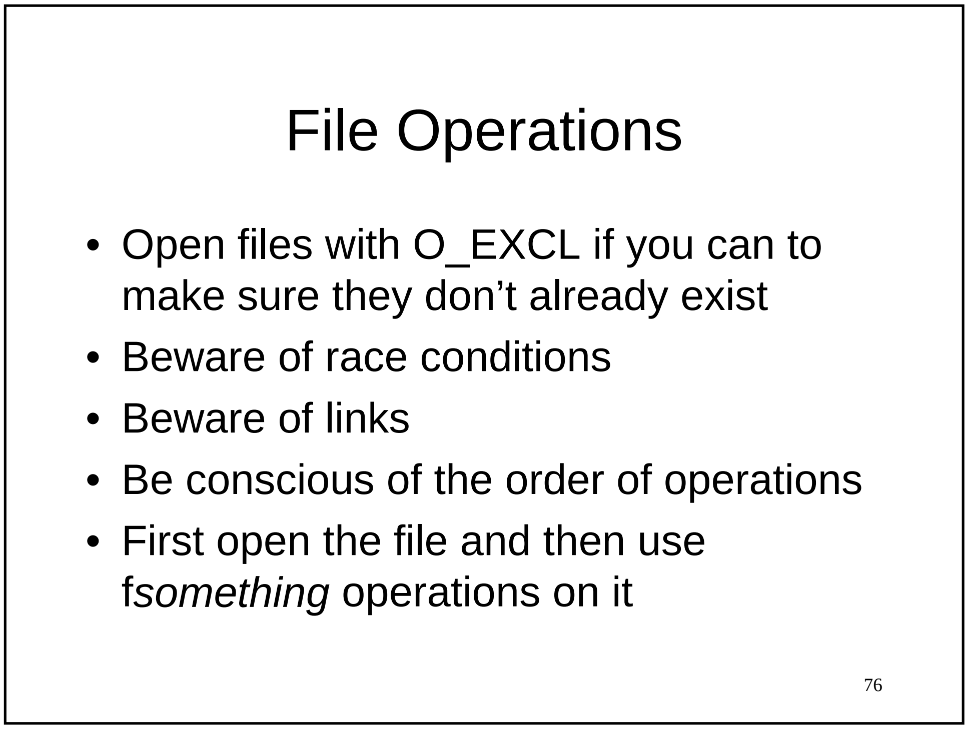


Reading Files

- Sendmail runs as root
  - `-C` flag specifies alternate config file
  - Error messages printed contain lines of file
  - Implementing `/bin/cat` in sendmail

```
/usr/lib/sendmail -C /etc/master.passwd
...all the passwords are spit out
```

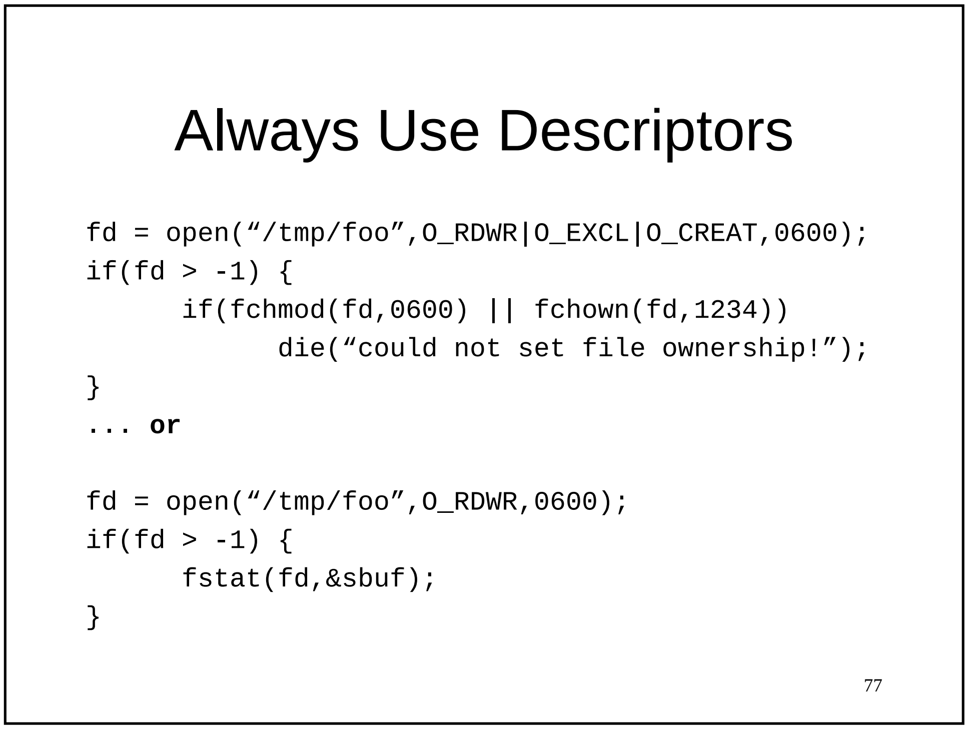
75



File Operations

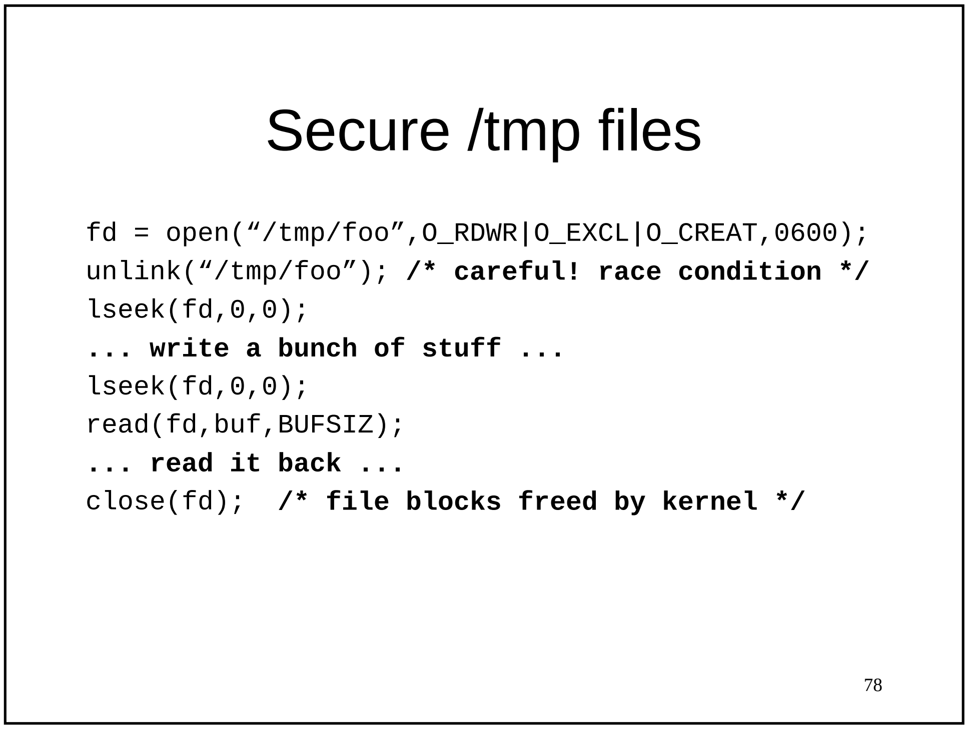
- Open files with O_EXCL if you can to make sure they don’t already exist
- Beware of race conditions
- Beware of links
- Be conscious of the order of operations
- First open the file and then use f*something* operations on it

76



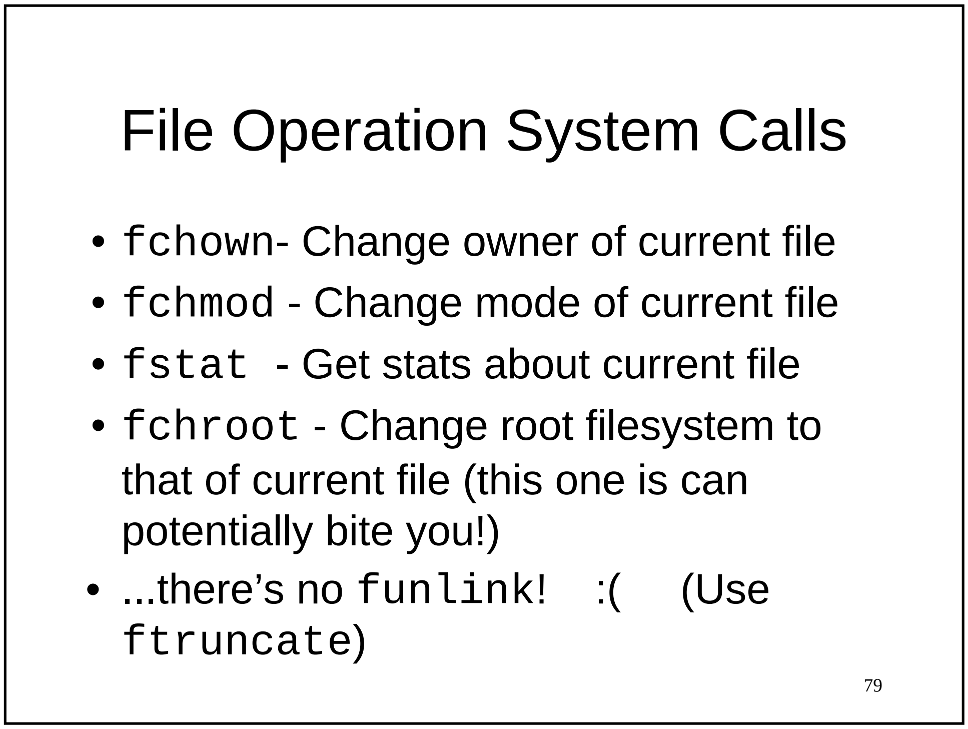
Always Use Descriptors

```
fd = open(“/tmp/foo”,O_RDWR|O_EXCL|O_CREAT,0600);
if(fd > -1) {
      if(fchmod(fd,0600) || fchown(fd,1234))
            die(“could not set file ownership!”);
}
... or

fd = open(“/tmp/foo”,O_RDWR,0600);
if(fd > -1) {
      fstat(fd,&sbuf);
}
```



Secure /tmp files

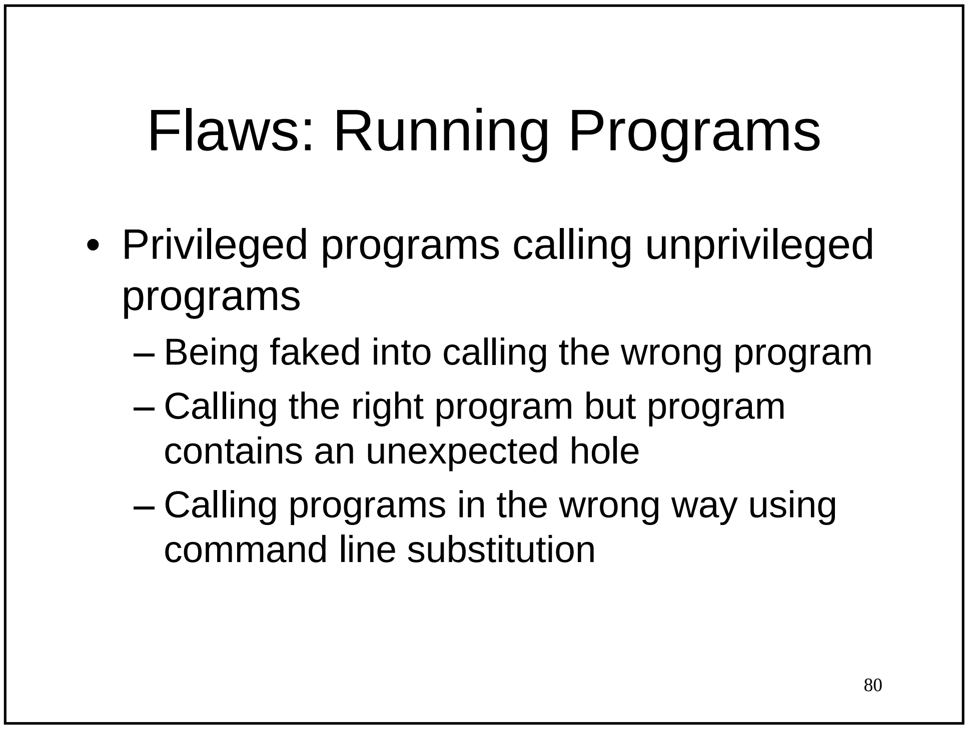
```
fd = open(“/tmp/foo”,O_RDWR|O_EXCL|O_CREAT,0600);
unlink(“/tmp/foo”); /* careful! race condition */
lseek(fd,0,0);
... write a bunch of stuff ...
lseek(fd,0,0);
read(fd,buf,BUFSIZ);
... read it back ...
close(fd);  /* file blocks freed by kernel */
```



File Operation System Calls

- `fchown`- Change owner of current file
- `fchmod` - Change mode of current file
- `fstat` - Get stats about current file
- `fchroot` - Change root filesystem to that of current file (this one is can potentially bite you!)
- ...there's no `funlink`! :((Use `ftruncate`)

79



Flaws: Running Programs

- Privileged programs calling unprivileged programs
  - Being faked into calling the wrong program
  - Calling the right program but program contains an unexpected hole
  - Calling programs in the wrong way using command line substitution

80

system(3) bugs

- Guts of library routine do:

```
execlp(“/bin/sh”,”sh”,”-c”,commandline,0);
```

- Shell relies on:
  - Search PATH - `system(“mail mjr”);` will call whatever “mail” is found first
  - Values in environment - `system(“/bin/mail mjr”);` may use additional shell syntax

system(3) and IFS

- shell uses IFS to determine whitespace characters

```
$ IFS=“/ \t\n”; export IFS
$ PATH=.:$PATH; export PATH
$ vi somefile
..hangup
```

- vi calls `system(“/bin/mail mjr”);`
- Shell interprets “/bin/mail mjr” as “bin mail mjr” calls a program `./bin`

Calling Programs Safely

- ***Avoid*** `execlp` and `execvp` which use $PATH
- ***Avoid*** `popen(3)` unless you spend a lot of effort resetting the environment first or develop `mypopen()`
- Use `exec{l,le,v}` directly with `fork(2)` - it is easy!

Calling Programs Safely *(cont)*

- `Ftpd_popen()` from `aftpd` is a good example of secure `popen()` implementation
  - See appendix
- Changing `system()` to use `exec()` directly instead of shell is not too hard

Shared Libraries

- Shared libraries are a problem because they may permit user to control properties of an executable
  - User builds their own dynamic library with a copy of, e.g.: `fgets(3)` that calls a shell
  - Put library in `~/libc.so`
  - Sets

```
LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
```

Shared Libraries *(cont)*

- Simple attempts to fix by system designers don’t take into account inheritance of environment (e.g.: programs that call programs that call programs...)
  - /bin/login calls /bin/sync
- ***Always*** link your privileged executables staticly

Programs w/Holes

- Early version of a root-admin shell used `more` as a viewer for help files
  - System was easy to break into by calling the admin shell and asking for help, then doing a shell escape from `more`
  - Did not elegantly handle a large variety of shell escapes (e.g., `more`, `vi`, `ftp`)

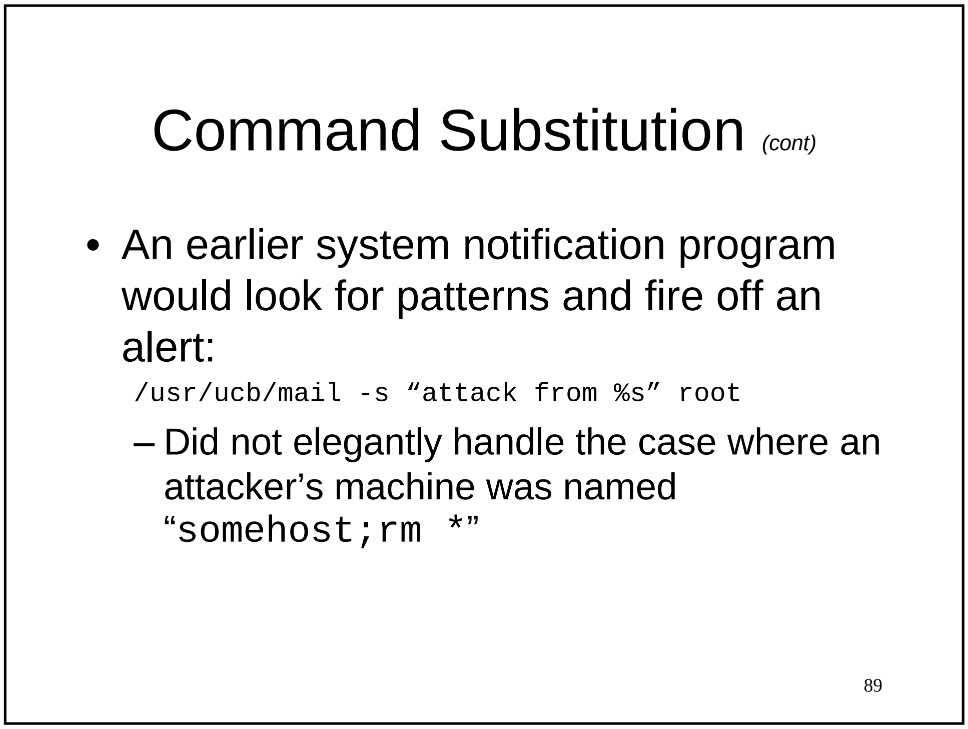
Command Substitution

- XMosaic authors took a cheap route for handling TELNET URLs*

```
sprintf(buf,”telnet %s”,resid_url);
system(buf);
```

  - Did not elegantly handle URLs in the form of `telnet://somesite;rm -rf *`
  - Blocking ‘;’ is not enough, what about: `telnet://somesite&&rm -rf *`

* No, I am not making this up

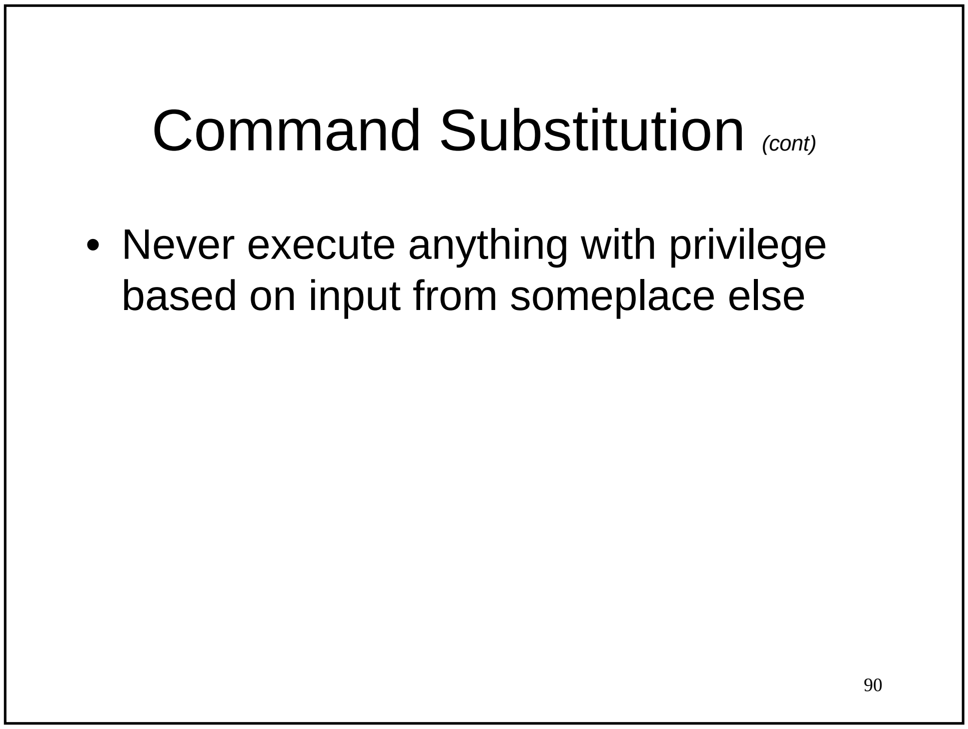


Command Substitution *(cont)*

- An earlier system notification program would look for patterns and fire off an alert:

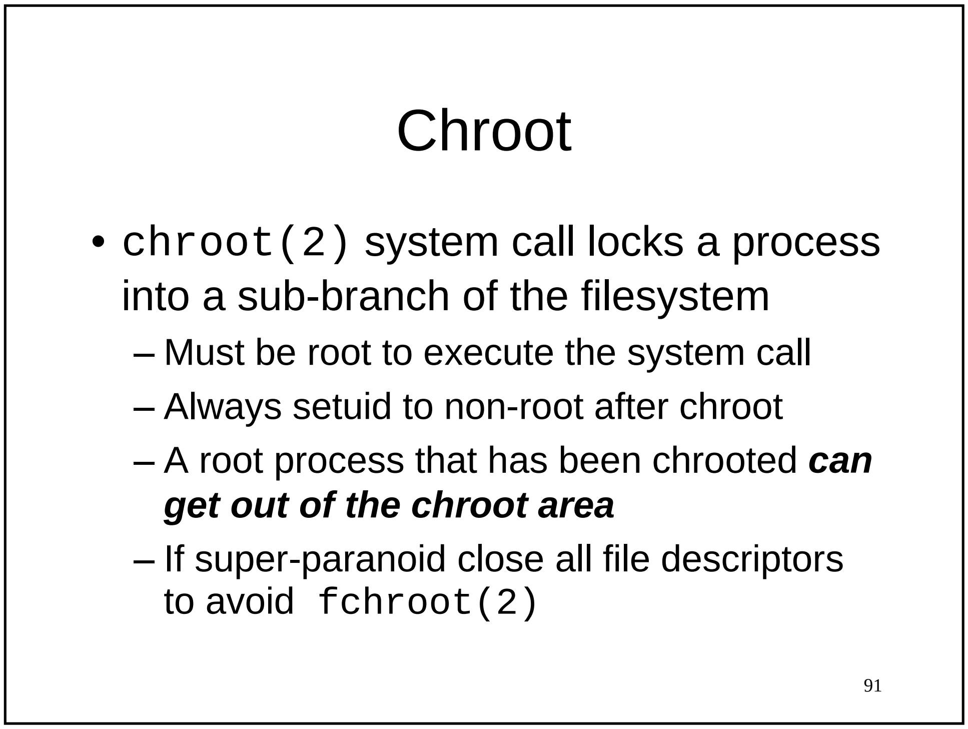
```
/usr/ucb/mail -s “attack from %s” root
```

  - Did not elegantly handle the case where an attacker’s machine was named “`somehost;rm *`”



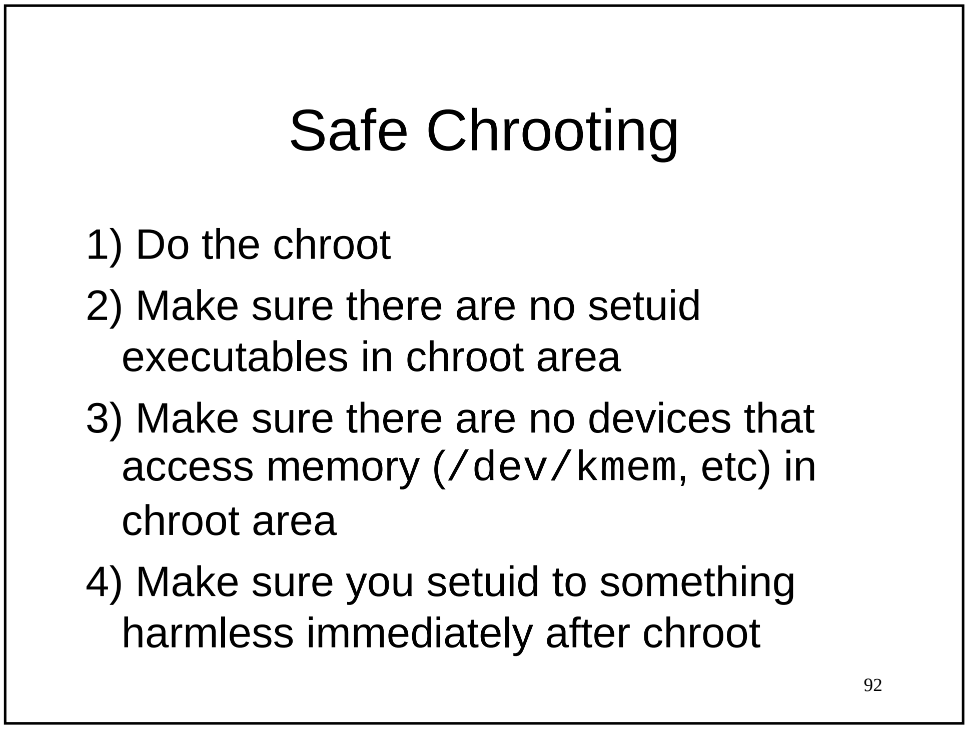
Command Substitution *(cont)*

- Never execute anything with privilege based on input from someplace else



Chroot

- `chroot(2)` system call locks a process into a sub-branch of the filesystem
  - Must be root to execute the system call
  - Always setuid to non-root after chroot
  - A root process that has been chrooted ***can get out of the chroot area***
  - If super-paranoid close all file descriptors to avoid `fchroot(2)`



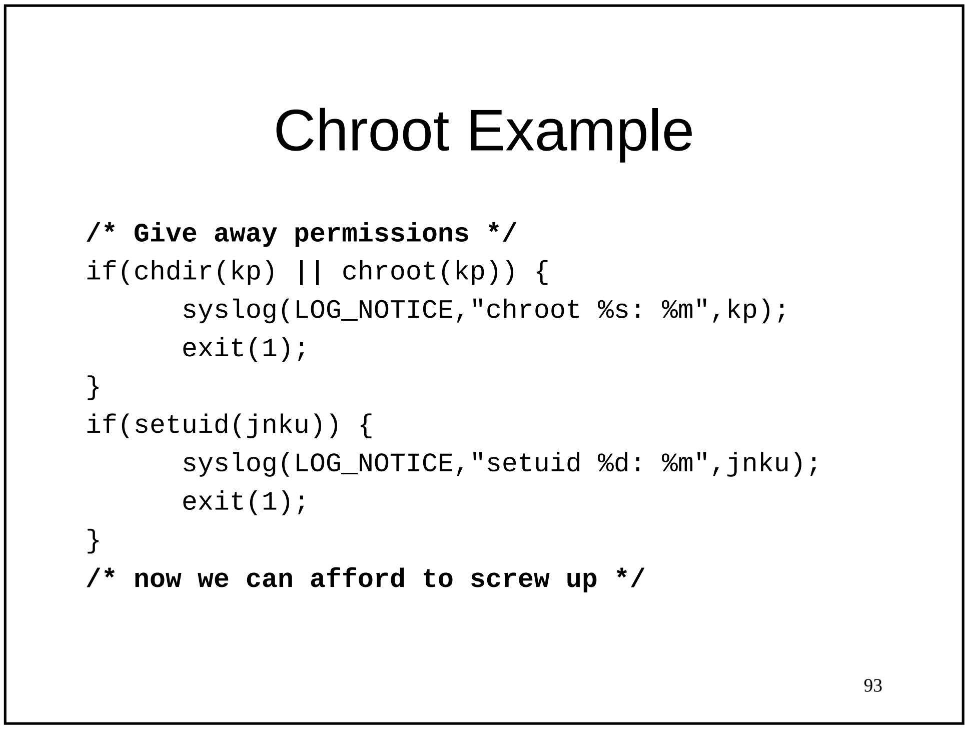
Safe Chrooting

1) Do the chroot

2) Make sure there are no setuid executables in chroot area

3) Make sure there are no devices that access memory (`/dev/kmem`, etc) in chroot area

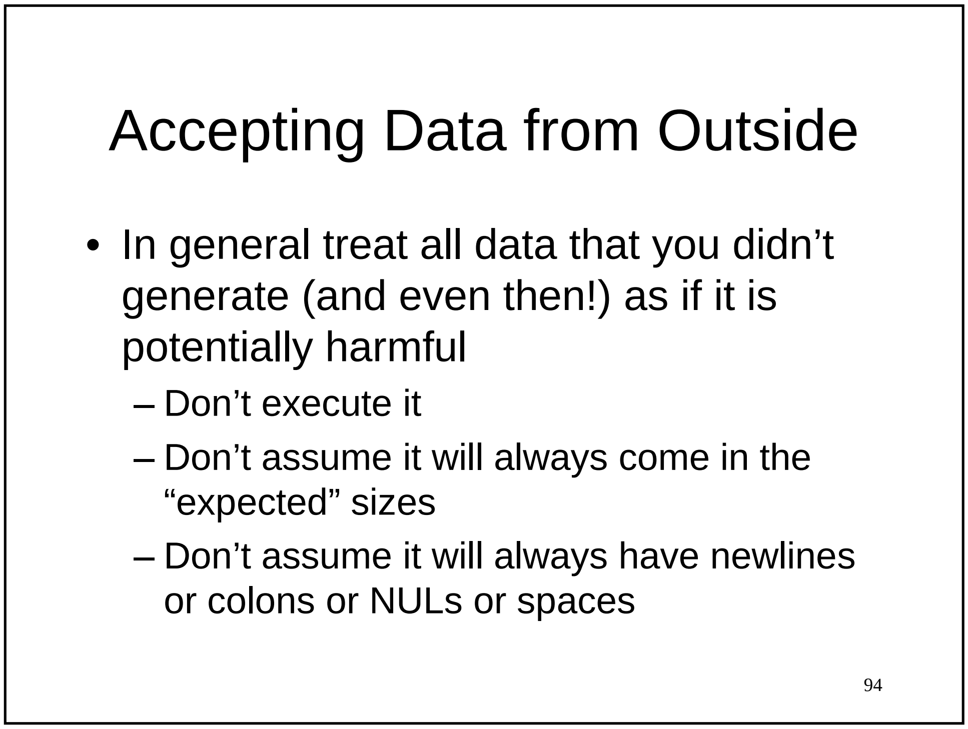
4) Make sure you setuid to something harmless immediately after chroot



Chroot Example

```
/* Give away permissions */
if(chdir(kp) || chroot(kp)) {
      syslog(LOG_NOTICE,"chroot %s: %m",kp);
      exit(1);
}
if(setuid(jnku)) {
      syslog(LOG_NOTICE,"setuid %d: %m",jnku);
      exit(1);
}
/* now we can afford to screw up */
```

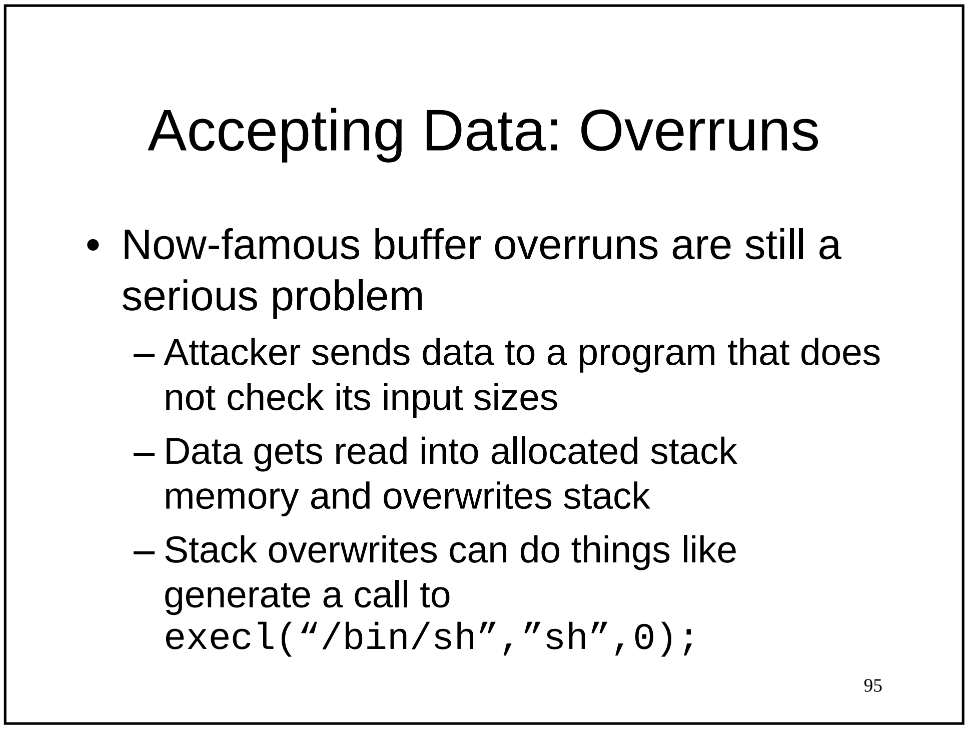
93



Accepting Data from Outside

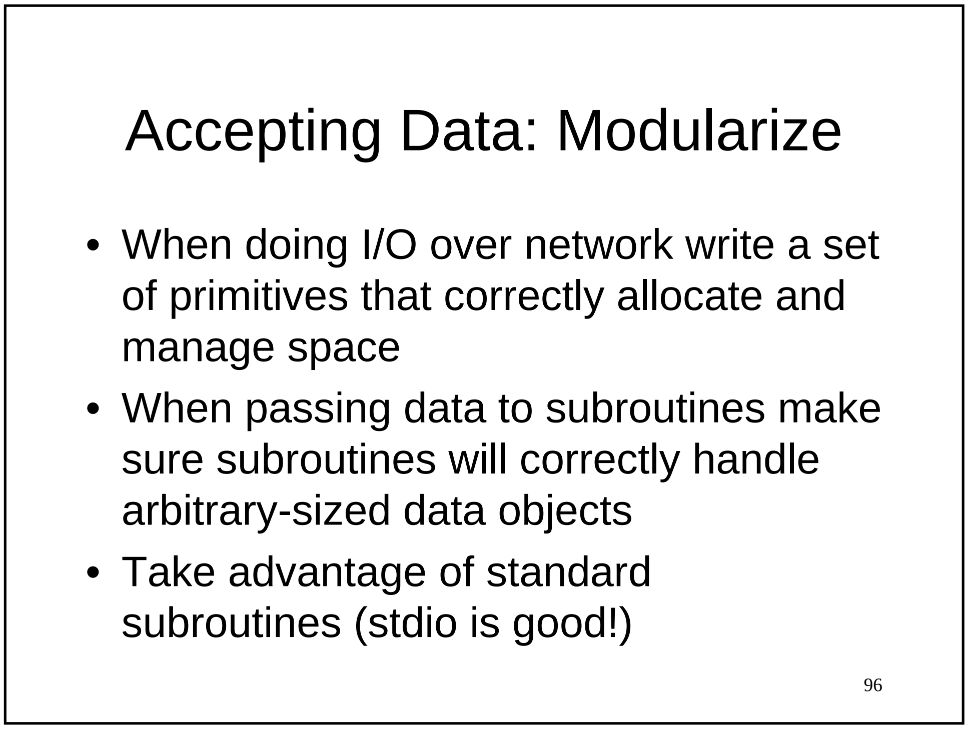
- In general treat all data that you didn’t generate (and even then!) as if it is potentially harmful
  - Don’t execute it
  - Don’t assume it will always come in the “expected” sizes
  - Don’t assume it will always have newlines or colons or NULs or spaces

94



Accepting Data: Overruns

- Now-famous buffer overruns are still a serious problem
  - Attacker sends data to a program that does not check its input sizes
  - Data gets read into allocated stack memory and overwrites stack
  - Stack overwrites can do things like generate a call to `execl("/bin/sh","sh",0);`



Accepting Data: Modularize

- When doing I/O over network write a set of primitives that correctly allocate and manage space
- When passing data to subroutines make sure subroutines will correctly handle arbitrary-sized data objects
- Take advantage of standard subroutines (stdio is good!)

Accepting Data: Suffer

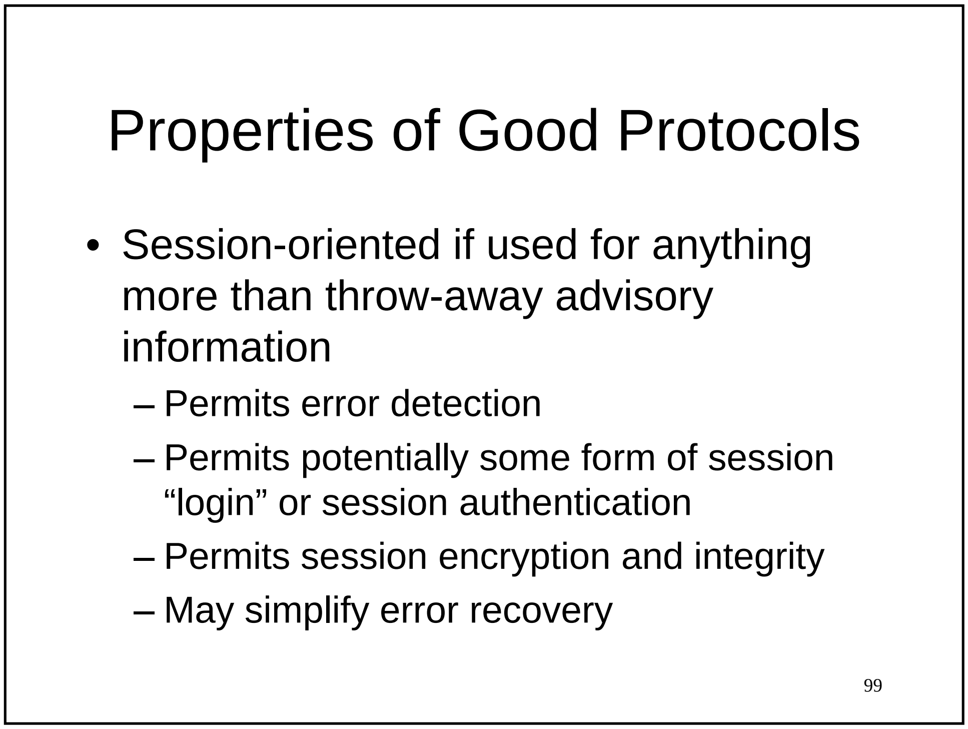
- No matter what, you still cannot trust that you will not have overruns
  - Example: `syslog()` routine has overrun that goes unnoticed for years and years
- If possible use tools like CodeCenter or Purify that check boundaries
- If possible test your own routines carefully using test harnesses

97

Protocols

- Application protocols take practice
- Several well-known examples to plagiarize from:
  - NNTP
  - SMTP
- Do not steal ideas from:
  - HTTP
  - NFS

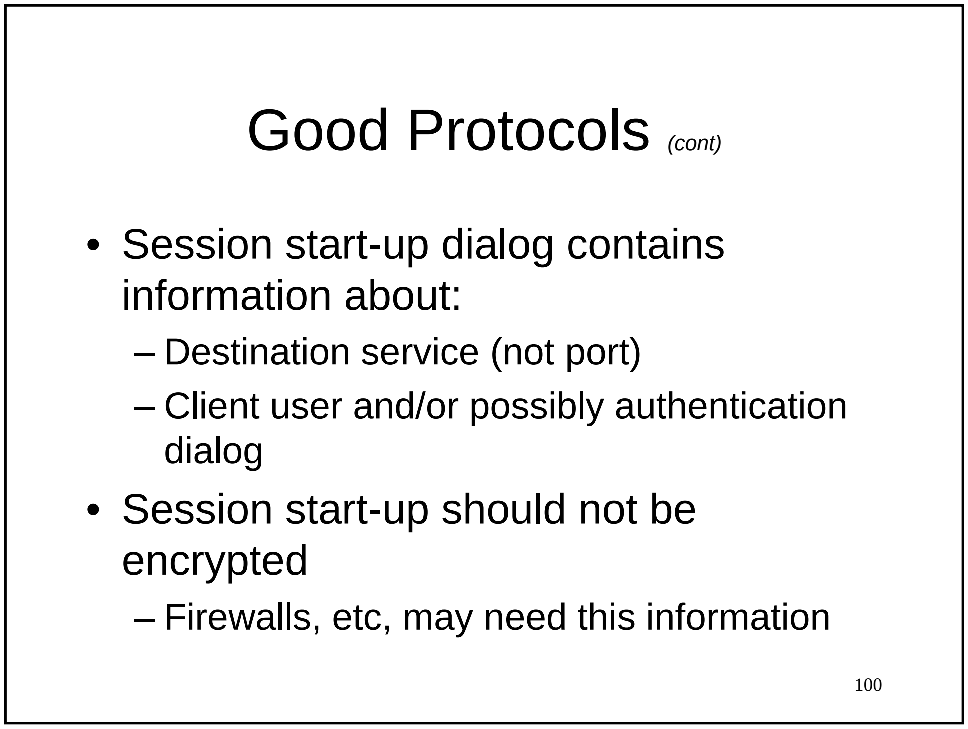
98



Properties of Good Protocols

- Session-oriented if used for anything more than throw-away advisory information
  - Permits error detection
  - Permits potentially some form of session “login” or session authentication
  - Permits session encryption and integrity
  - May simplify error recovery

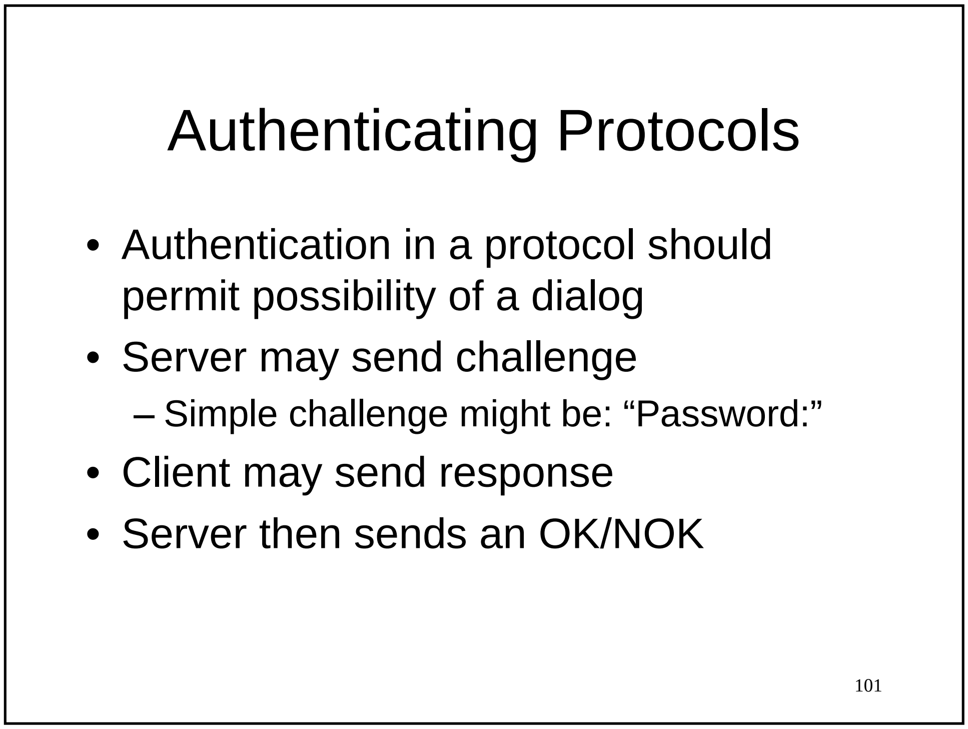
99



Good Protocols *(cont)*

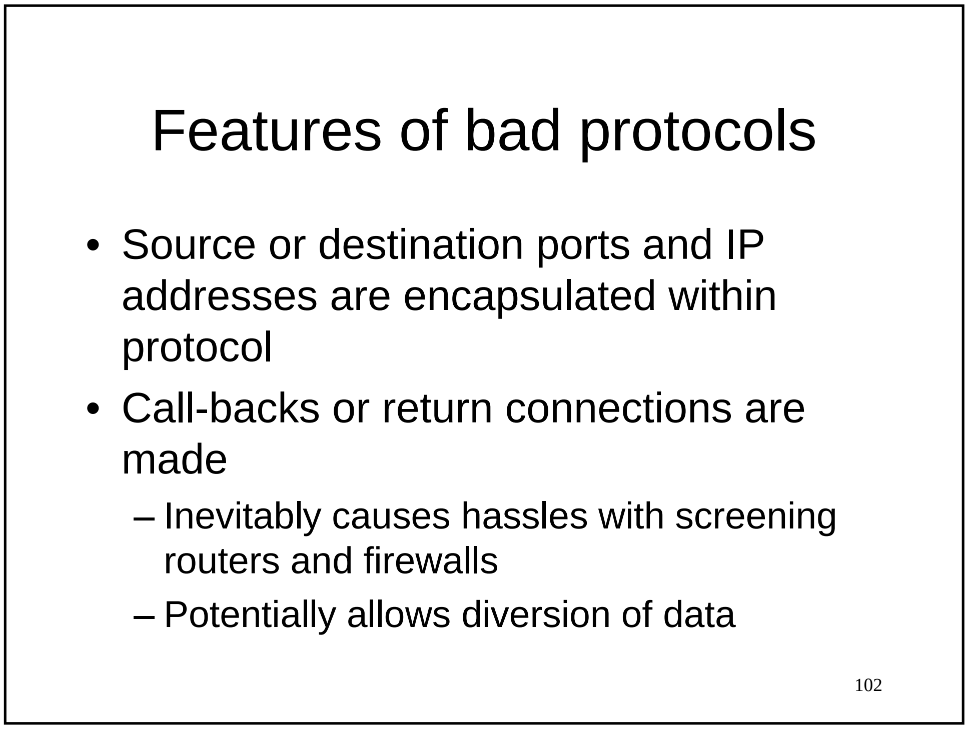
- Session start-up dialog contains information about:
  - Destination service (not port)
  - Client user and/or possibly authentication dialog
- Session start-up should not be encrypted
  - Firewalls, etc, may need this information

100



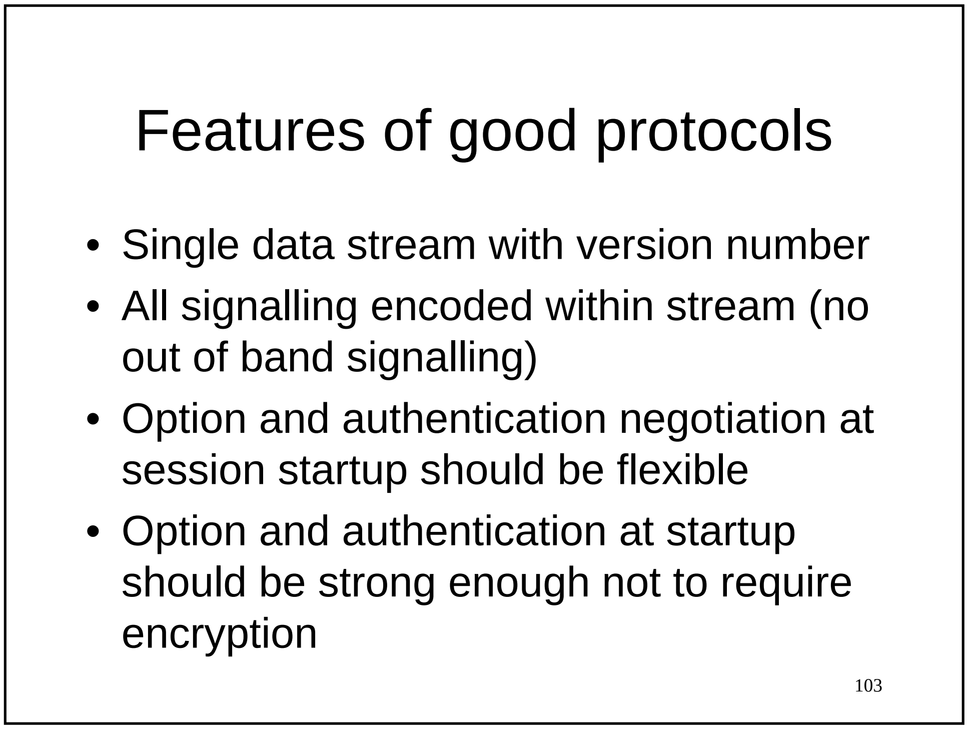
Authenticating Protocols

- Authentication in a protocol should permit possibility of a dialog
- Server may send challenge
  - Simple challenge might be: “Password:”
- Client may send response
- Server then sends an OK/NOK



Features of bad protocols

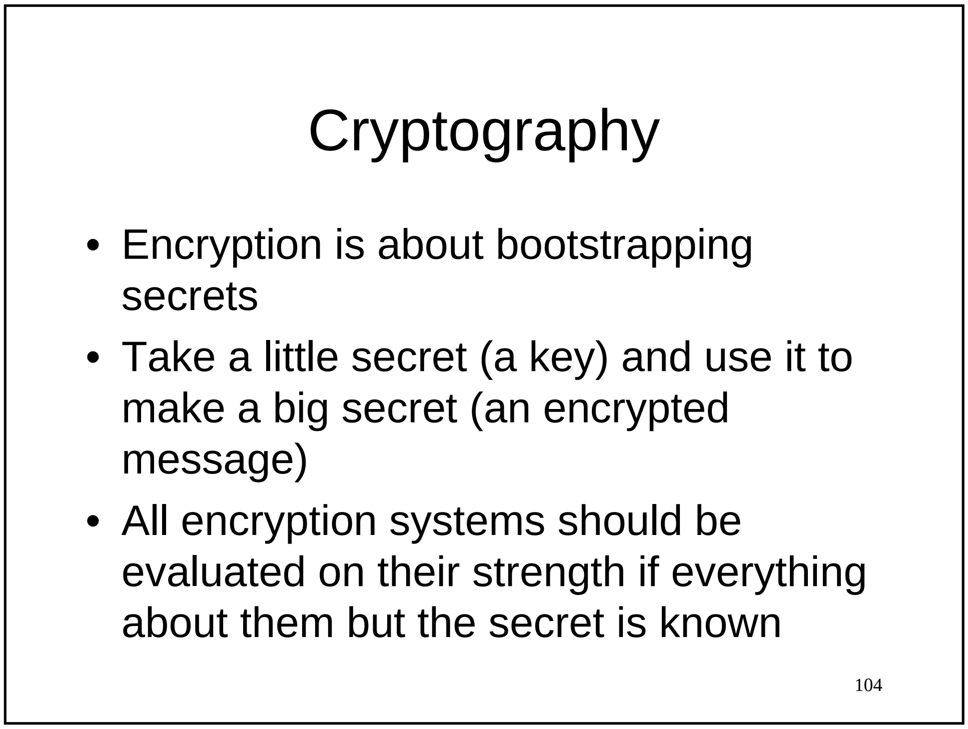
- Source or destination ports and IP addresses are encapsulated within protocol
- Call-backs or return connections are made
  - Inevitably causes hassles with screening routers and firewalls
  - Potentially allows diversion of data



Features of good protocols

- Single data stream with version number
- All signalling encoded within stream (no out of band signalling)
- Option and authentication negotiation at session startup should be flexible
- Option and authentication at startup should be strong enough not to require encryption

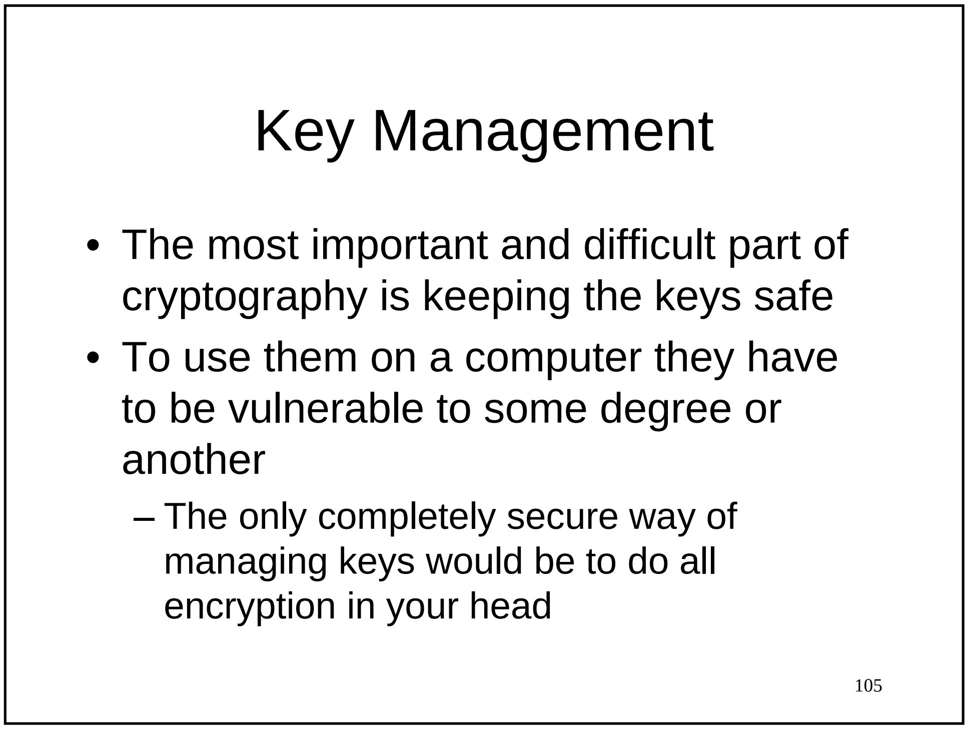
103



Cryptography

- Encryption is about bootstrapping secrets
- Take a little secret (a key) and use it to make a big secret (an encrypted message)
- All encryption systems should be evaluated on their strength if everything about them but the secret is known

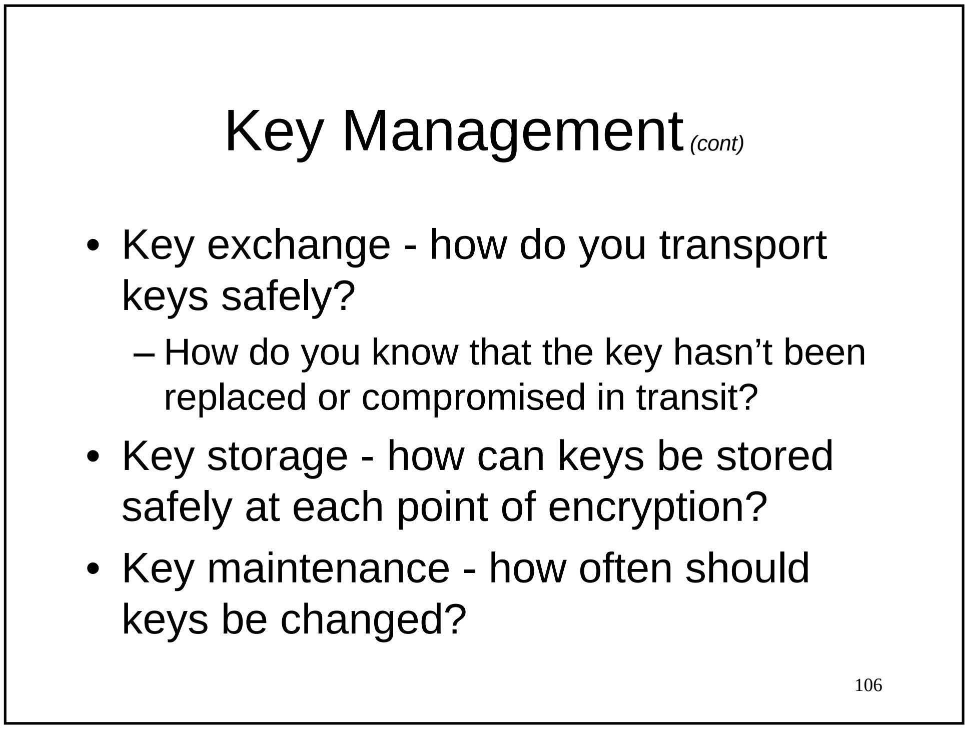
104



Key Management

- The most important and difficult part of cryptography is keeping the keys safe
- To use them on a computer they have to be vulnerable to some degree or another
  - The only completely secure way of managing keys would be to do all encryption in your head

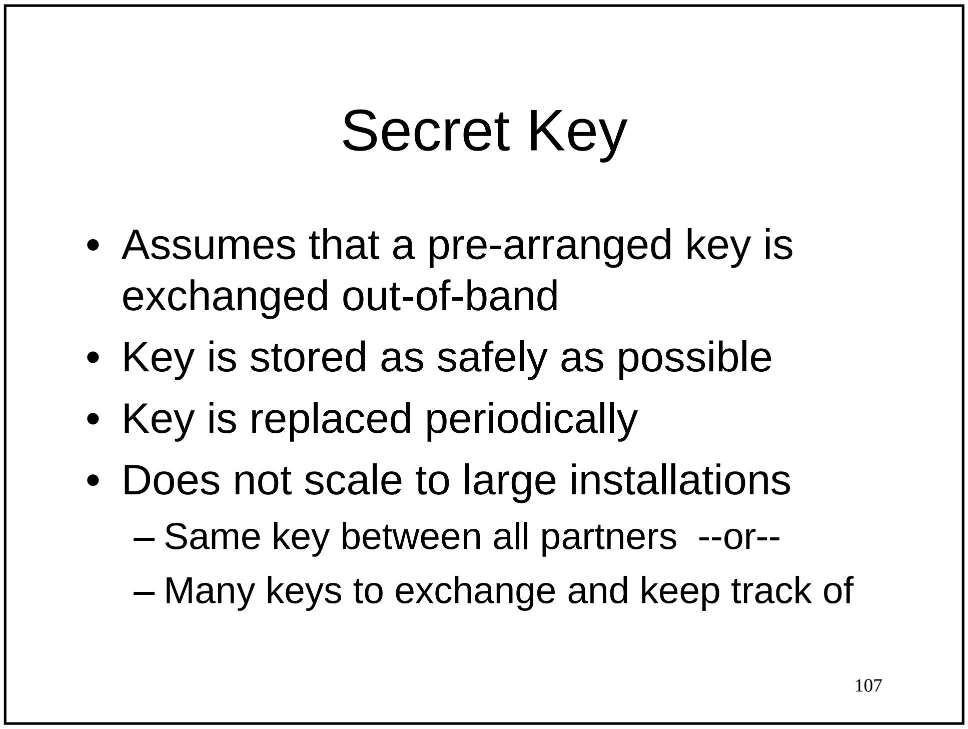
105



Key Management *(cont)*

- Key exchange - how do you transport keys safely?
  - How do you know that the key hasn’t been replaced or compromised in transit?
- Key storage - how can keys be stored safely at each point of encryption?
- Key maintenance - how often should keys be changed?

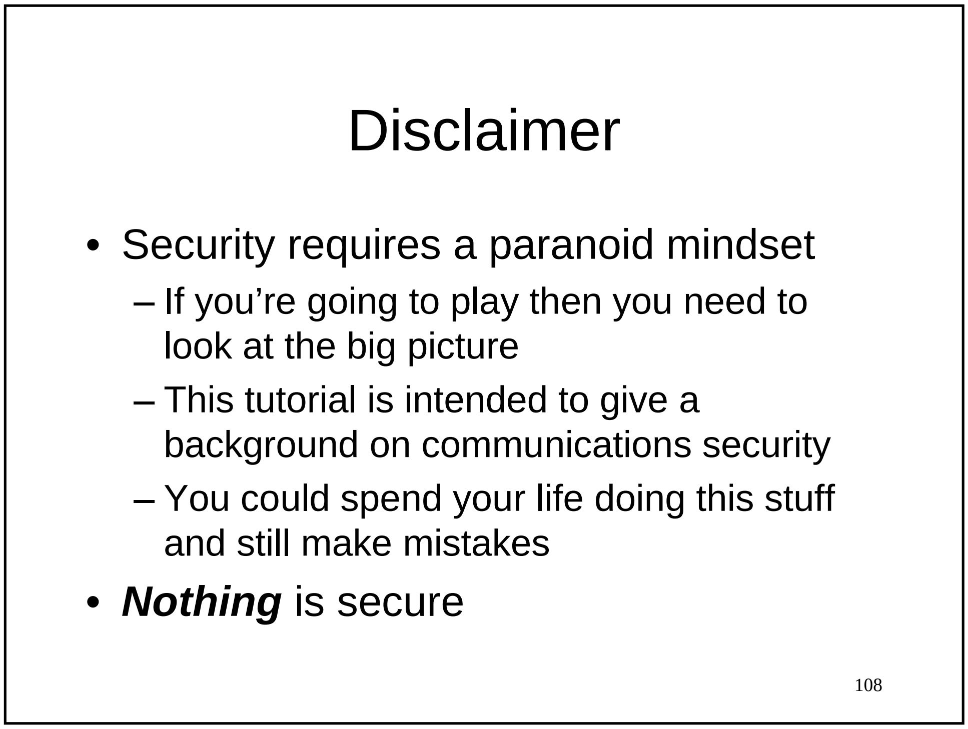
106



Secret Key

- Assumes that a pre-arranged key is exchanged out-of-band
- Key is stored as safely as possible
- Key is replaced periodically
- Does not scale to large installations
  - Same key between all partners --or--
  - Many keys to exchange and keep track of

107

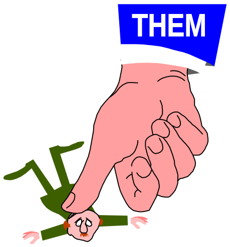


Disclaimer

- Security requires a paranoid mindset
  - If you’re going to play then you need to look at the big picture
  - This tutorial is intended to give a background on communications security
  - You could spend your life doing this stuff and still make mistakes
- ***Nothing*** is secure

108

Just because you're paranoid doesn't mean "they" aren't out to get you.

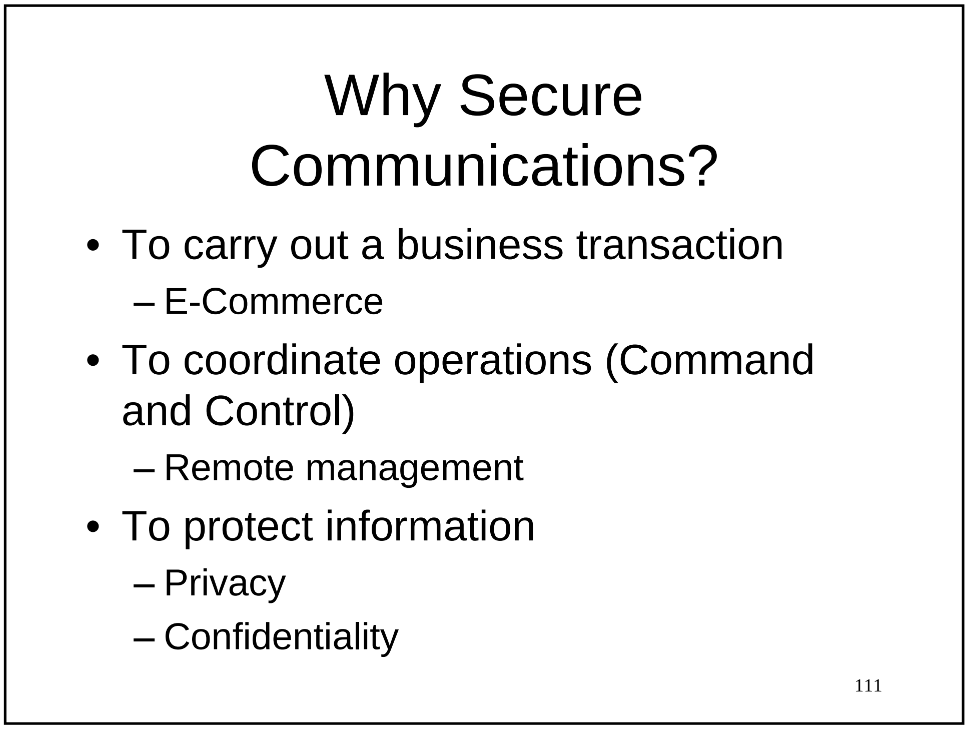


109

Before You Start

- Risk Assessment:
  - What are you trying to hide?
  - How much will it hurt if "they" find it out?
  - How hard will "they" try?
  - How much are you willing to spend? "spend" means a combination of:
    - Time
    - Pain
    - Money

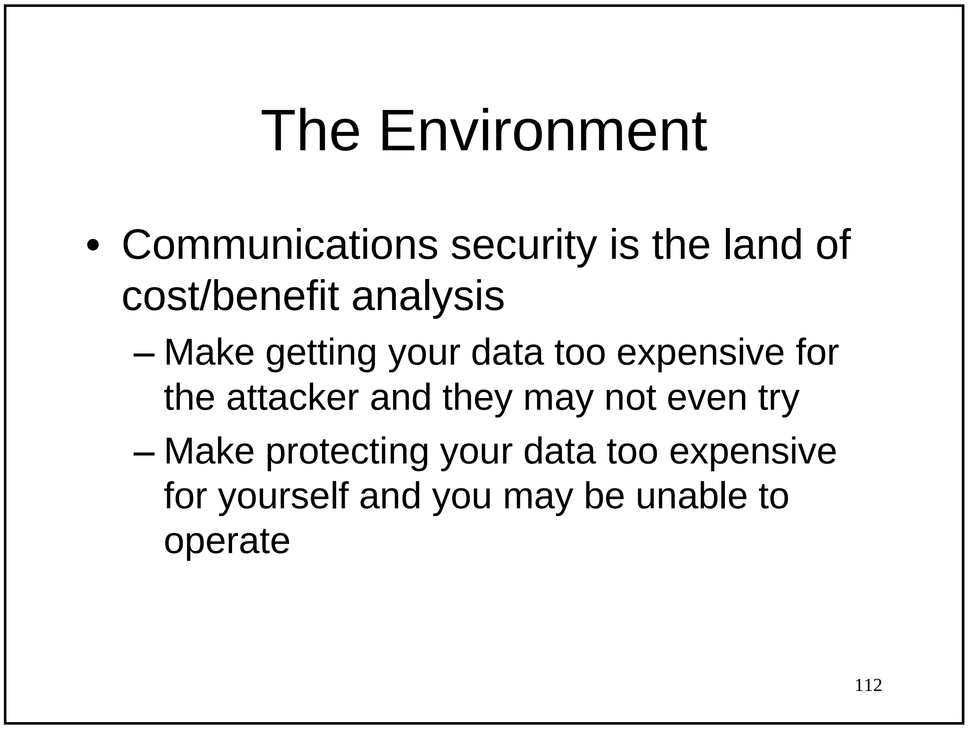
110



Why Secure Communications?

- To carry out a business transaction
  - E-Commerce
- To coordinate operations (Command and Control)
  - Remote management
- To protect information
  - Privacy
  - Confidentiality

111



The Environment

- Communications security is the land of cost/benefit analysis
  - Make getting your data too expensive for the attacker and they may not even try
  - Make protecting your data too expensive for yourself and you may be unable to operate

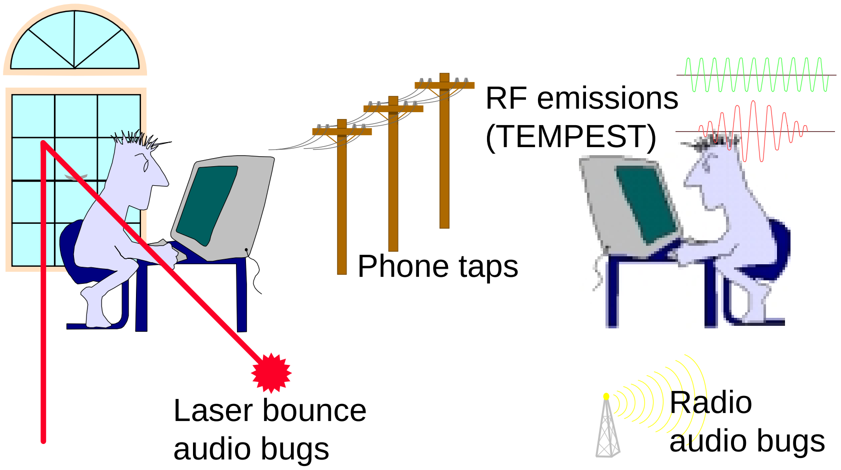
112

Target Analysis

- Target analysis is the (hypothetical) art of analyzing a target's communications security to identify the weakest link
- You'd better do it, because "they" will do it, too

113

Target Analysis



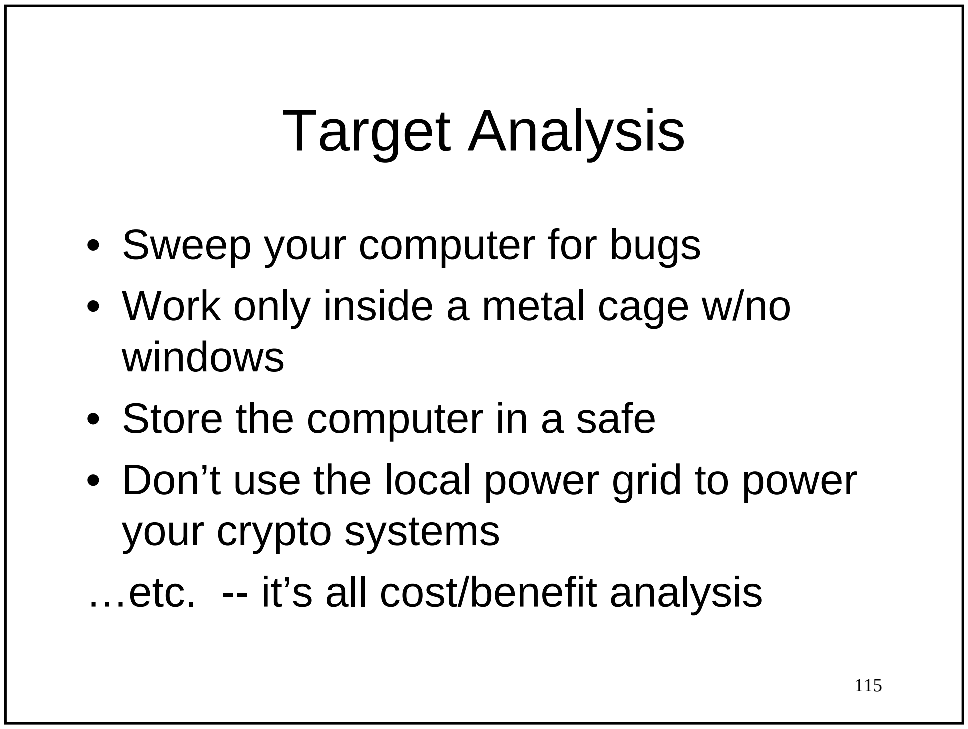
RF emissions (TEMPEST)

Phone taps

Laser bounce audio bugs

Radio audio bugs

114

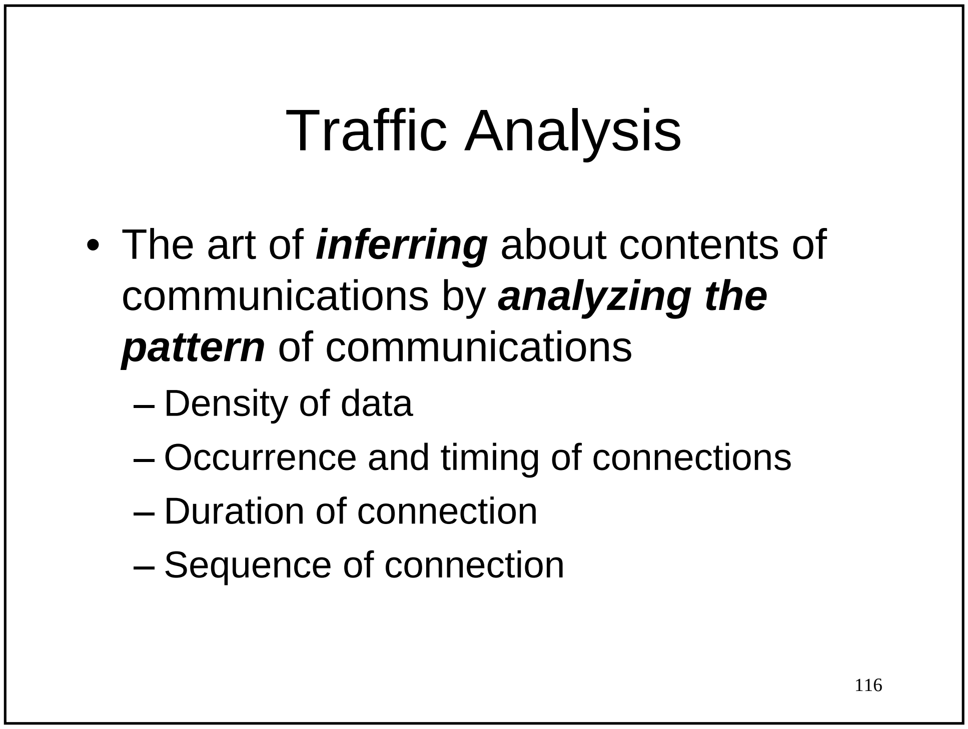


Target Analysis

- Sweep your computer for bugs
- Work only inside a metal cage w/no windows
- Store the computer in a safe
- Don’t use the local power grid to power your crypto systems

…etc. -- it’s all cost/benefit analysis

115

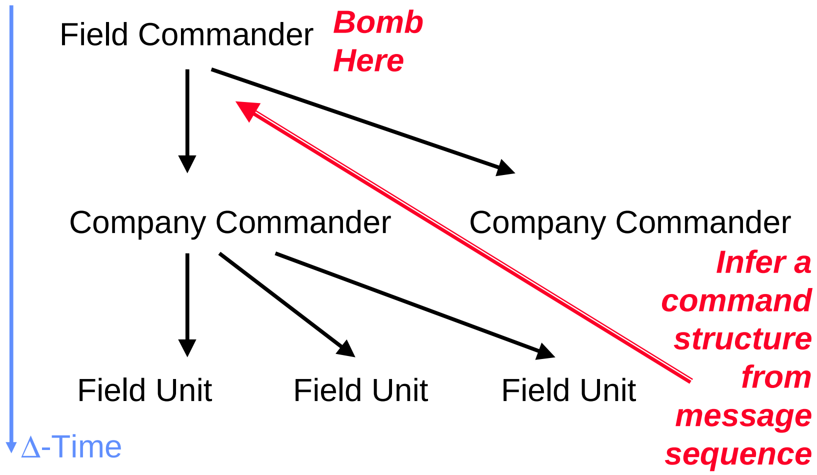


Traffic Analysis

- The art of ***inferring*** about contents of communications by ***analyzing the pattern*** of communications
  - Density of data
  - Occurrence and timing of connections
  - Duration of connection
  - Sequence of connection

116

Traffic Analysis



117

Traffic Analysis in Open Networks

- In open networks the majority of traffic is in the clear (unsecured)

… Therefore securing it becomes a dead giveaway to the traffic analyst!

- Ideally your secure communications will somehow look like unsecure communications or get lost in the noise

118

Traffic Analysis in Open Networks

- Incidentally, US law enforcement appears to be building an argument around a mindset that “if it’s encrypted that indicates that someone is probably doing something they shouldn’t”
  - I.e.: ***Honest*** people don’t ***need*** secure communications

Traffic Analysis: Example

- Terrorist hit in Paris*
- French intelligence agency correlates
  - All payphone calls near kill zone
  - Calls within “time window” of kill
  - Calls to another payphone that makes a call outside of France within a 20 minute period
  - Iranian agent in south of France is caught

*Amazingly, this was reported in Time magazine

Traffic Analysis: Internet

- Identify software pirates by correlating file download activity
  - Large size files
  - Download rate
  - Frequency of particular files
  - Correlate file sizes/volumes across networks and you can backtrack users*

*Almost nobody keeps good enough logs to do this

Covert Channels

- Low data-rate communications encoded and hidden within another communication
  - Computers are great for this because they are patient!
- Example: Let’s say we agree that if I hit your web site within an hour, it’s a 1. If not it’s a 0. I can send 24 bits/day.

Covert Channels *(cont)*

- Signal theory applies to covert channels
  - data rate == signal strength
  - Noise reduction techniques can be applied to detect and potentially recover the signal
- The more data your covert channel carries the less covert it is*
  - The happier that makes a traffic analyst

*Note that this applies to “stealth scans” and denial of service

123

Covert Channels *(cont)*

- Implication:
  - If you are setting up a covert channel hide where the is already a high noise level:
    - AOL instant messenger (more on this later)
    - The firewalls mailing list :)
- Hidden does not mean secured
  - It just means that They have 2 problems to solve instead of one: finding your communications and then cracking them

124

TEMPEST *(cont)*

- Peter Wright describes in Spy Catcher:
  - A German intelligence office...
  - British spies attempting to bug it by sneaking in along buried power lines...
  - Discover to their surprise that there is a signal on the power line...
  - The signal is generated by code machines and can be decoded into teletype output!

125

TEMPEST *(cont)*

- If you think They are going to come after you with Van Eck monitors, you’re in deep trouble
  - Use battery powered laptops
  - Work in electronically dead rooms underground
  - Run your TV and blender while you are encoding and decoding :)

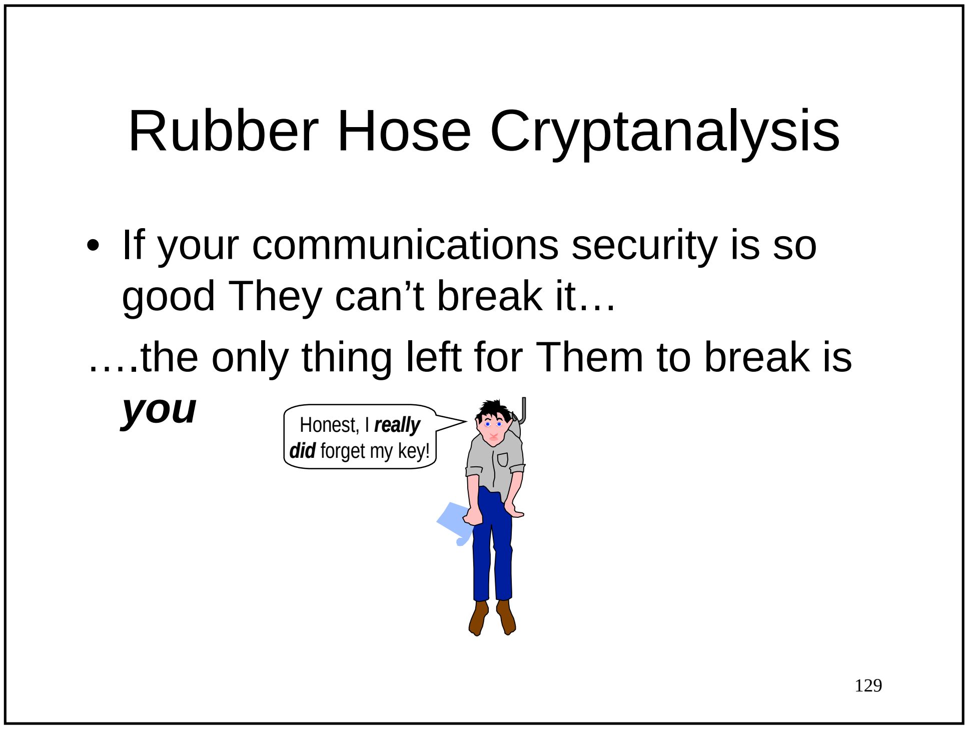
126

Cryptanalysis

- Code-breaking is very time-consuming and requires highly specialized skills
  - It’s a ***very*** expensive form of attack
  - Affordable by well-funded government agencies and research scientists
  - Outside the scope of “ordinary” hacking activity

Cryptanalysis *(cont)*

- Usually it’s cheaper to exploit other flaws
  - Use well-known and tested algorithms
  - Worry about the other stuff instead



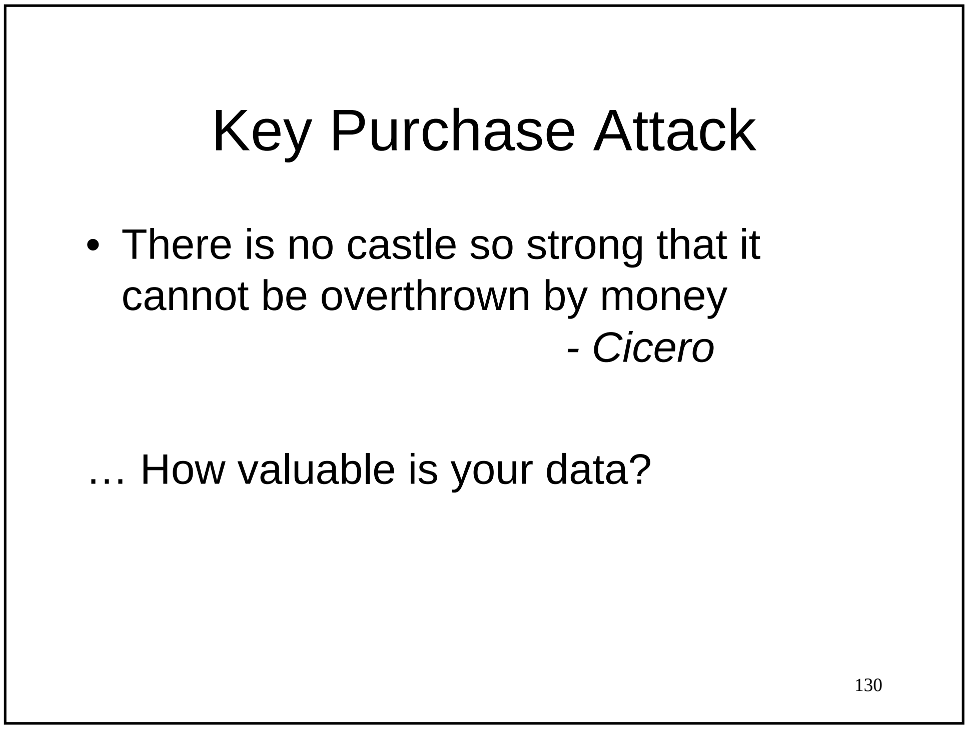
Rubber Hose Cryptanalysis

- If your communications security is so good They can't break it…

….the only thing left for Them to break is ***you***

Honest, I ***really did*** forget my key!

129

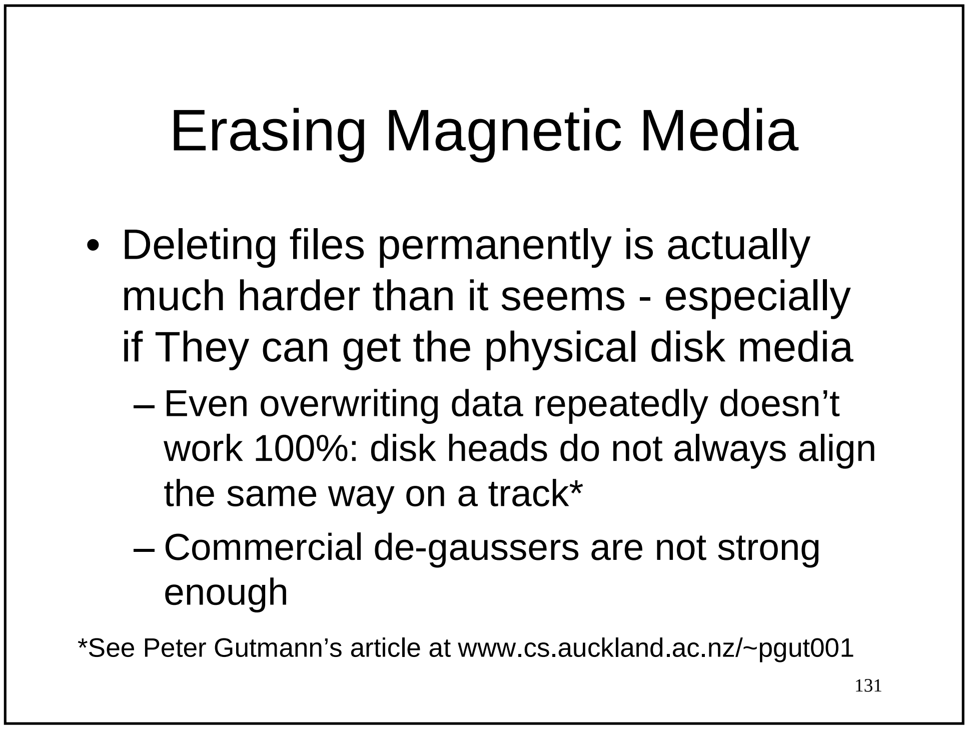


Key Purchase Attack

- There is no castle so strong that it cannot be overthrown by money
  - *Cicero*

… How valuable is your data?

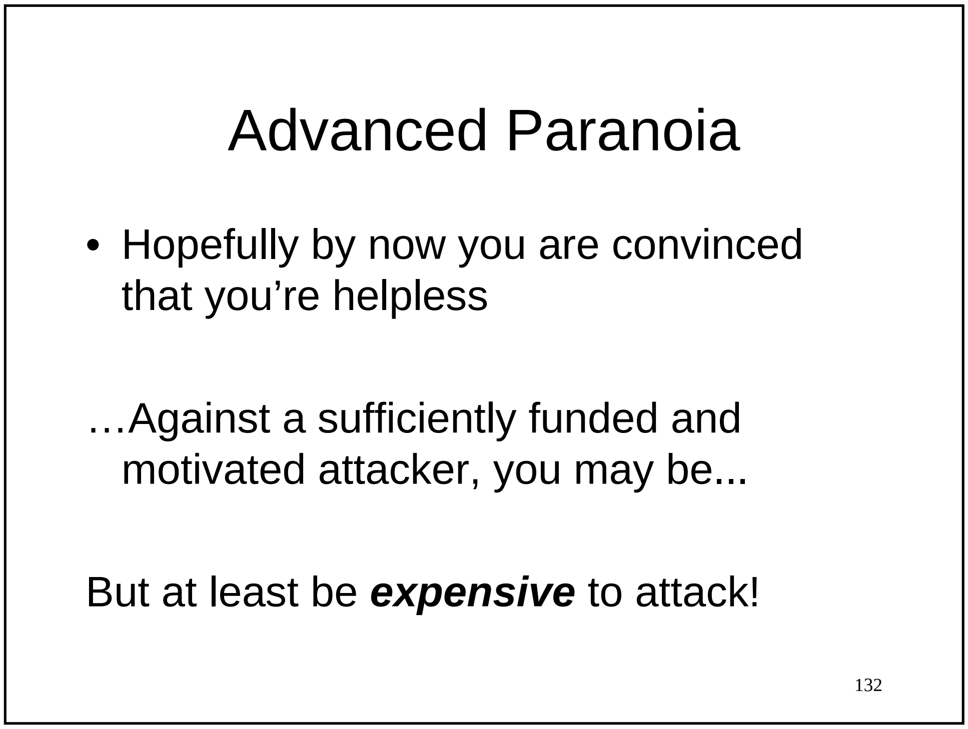
130



Erasing Magnetic Media

- Deleting files permanently is actually much harder than it seems - especially if They can get the physical disk media
  - Even overwriting data repeatedly doesn’t work 100%: disk heads do not always align the same way on a track*
  - Commercial de-gaussers are not strong enough

*See Peter Gutmann’s article at www.cs.auckland.ac.nz/~pgut001



Advanced Paranoia

- Hopefully by now you are convinced that you’re helpless

…Against a sufficiently funded and motivated attacker, you may be...

But at least be ***expensive*** to attack!

Goofy Comsec Stories: 1

- Peter Wright tells of Egyptians using a Hagelin rotor-based cipher machine
  - British agents place a bug in the code room, posing as telco workers
  - Whenever the Egyptians change their keys the British listeners count the >*click*< sounds of the rotors being set
  - Reduces the strength of the cipher to a few minutes' guesswork

Goofy Comsec Stories: 2

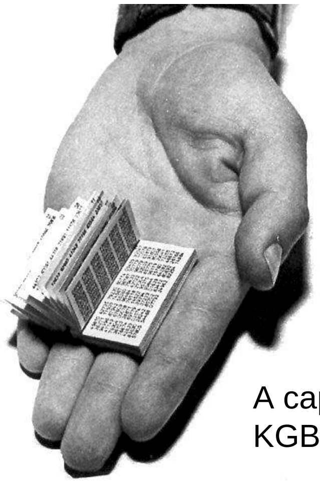
- Peter Wright tells of British embassy staff using one time pads in a secured code room
  - Russians plant an audio bug in the room
  - British cipher clerk reads the message aloud as another enciphers it one letter at a time
  - An unbreakable cryptosystem is completely sidestepped

Goofy Comsec Stories: 3

- Soviet agents subvert an NSA employee whose job it is to destroy classified documents
  - Since the documents are to be destroyed there is no audit trail for futher access
  - Instead of destroying them he sells them

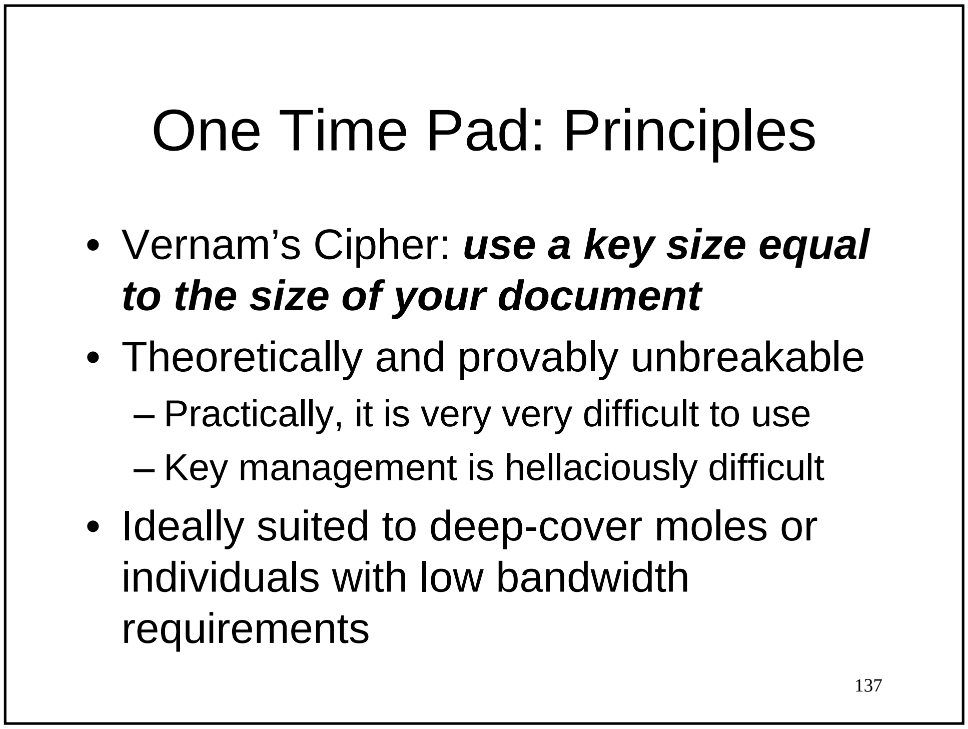
135

One Time Pads



A captured
KGB one time pad

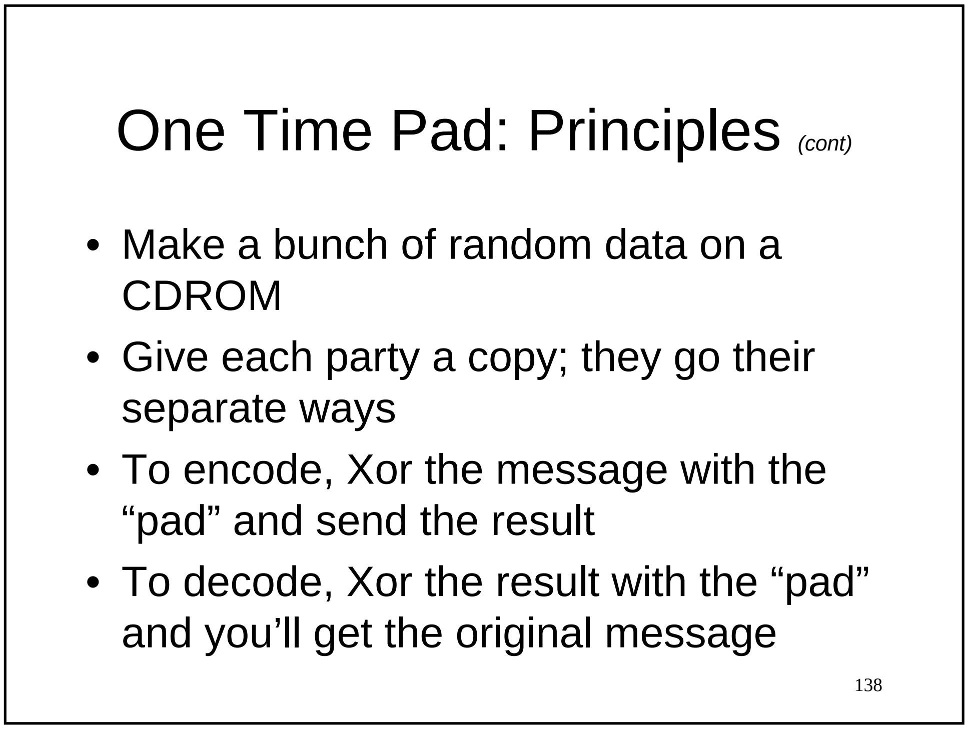
136



One Time Pad: Principles

- Vernam’s Cipher: ***use a key size equal to the size of your document***
- Theoretically and provably unbreakable
  - Practically, it is very very difficult to use
  - Key management is hellaciously difficult
- Ideally suited to deep-cover moles or individuals with low bandwidth requirements

137

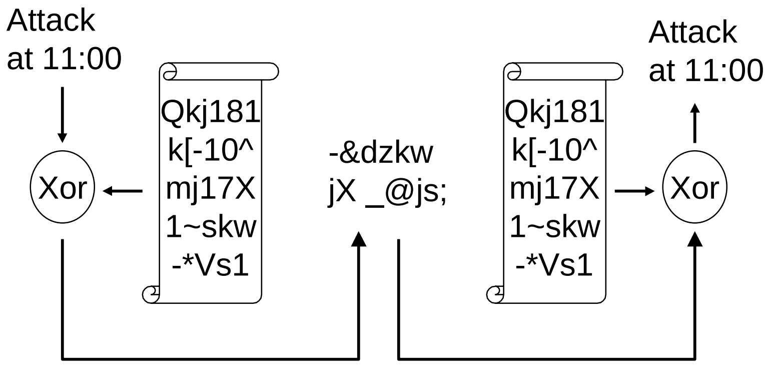


One Time Pad: Principles *(cont)*

- Make a bunch of random data on a CDROM
- Give each party a copy; they go their separate ways
- To encode, Xor the message with the “pad” and send the result
- To decode, Xor the result with the “pad” and you’ll get the original message

138

One Time Pad: Principles *(cont)*



139

One Time Pad: Randomness

- One Time Pads data must be completely random to be secure
  - Do NOT use output of DES, a music CD, etc.
  - Do use:
    - radioactive decay
    - MD5 output of a series of video-capture frames of a lava lamp in action
    - amplified background noise, sampled

140

One Time Pad: Exchanging Pads

- The tricky part is exchanging the pad
  - – If you are caught with a one time pad it is prima facie evidence of espionage
  - – If the pad is copied then you're completely compromised
    - Peter Wright tells of breaking into soviet spies' houses and copying their pads then reading their messages
  - – Make sure you give it to the right person!

141

VENONA

- Soviets use one time pads to operate deep cover moles
  - – Duplicates of pads were printed
  - – Duplicates accidentally are used to secure communications for shipboard monitoring
  - – A British code clerk recognizes patterns
  - – For several years the British are able to piece together tantalizing bits of KGB communications

142

VENONA *(Cont)*

VENONA

~~TOP SECRET~~

[redacted]

USSR

[redacted]

Ref. No: [redacted] [illegible]

Issued: [redacted] [illegible]

Copy No.: [illegible]

[illegible]

[illegible]

From: NEW YORK

To: MOSCOW

No.: [illegible]

[illegible] July 1943

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[Continued overleaf]

143

Public key

- Use clever mathematical tricks to exchange a key with another party over an insecure link
  - Diffie-Hellman key exchange
  - RSA key exchange
- Eavesdropper cannot access key
- Knowing you exchanged the key with the ***right partner*** is still tricky

144

Public Key *(cont)*

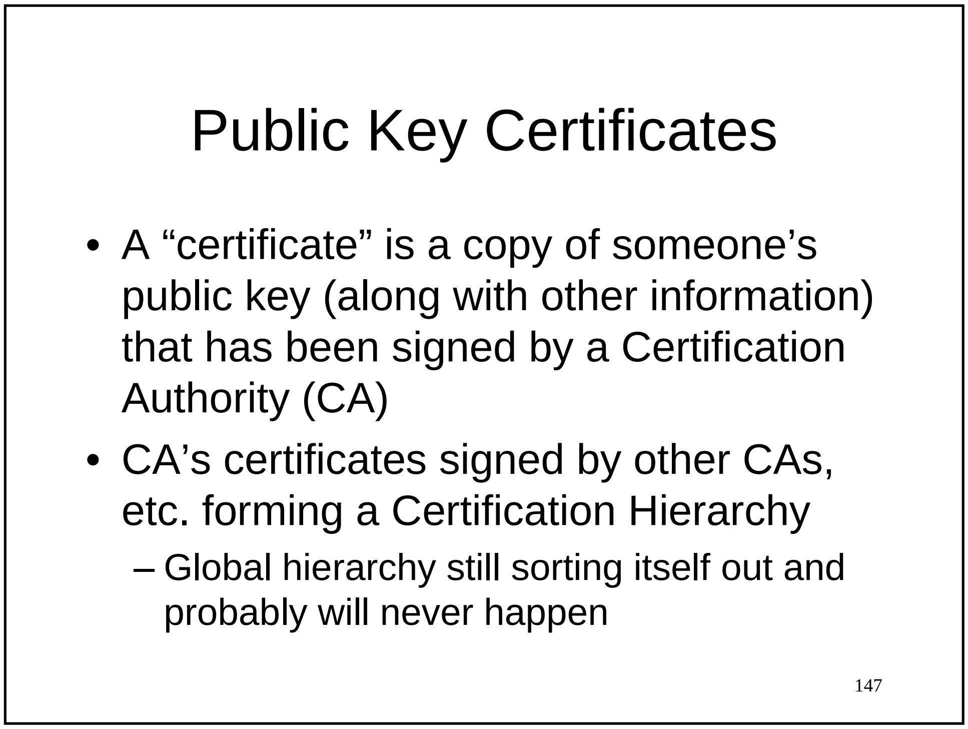
- Public key (RSA) can also be used in non-interactive exchanges
  - One party publishes a public half of a key pair keeping the other half of the key pair secret
  - The other party generates a message to the first party based on their published half which can only be decoded by the holder of the secret half

145

Public Key *(cont)*

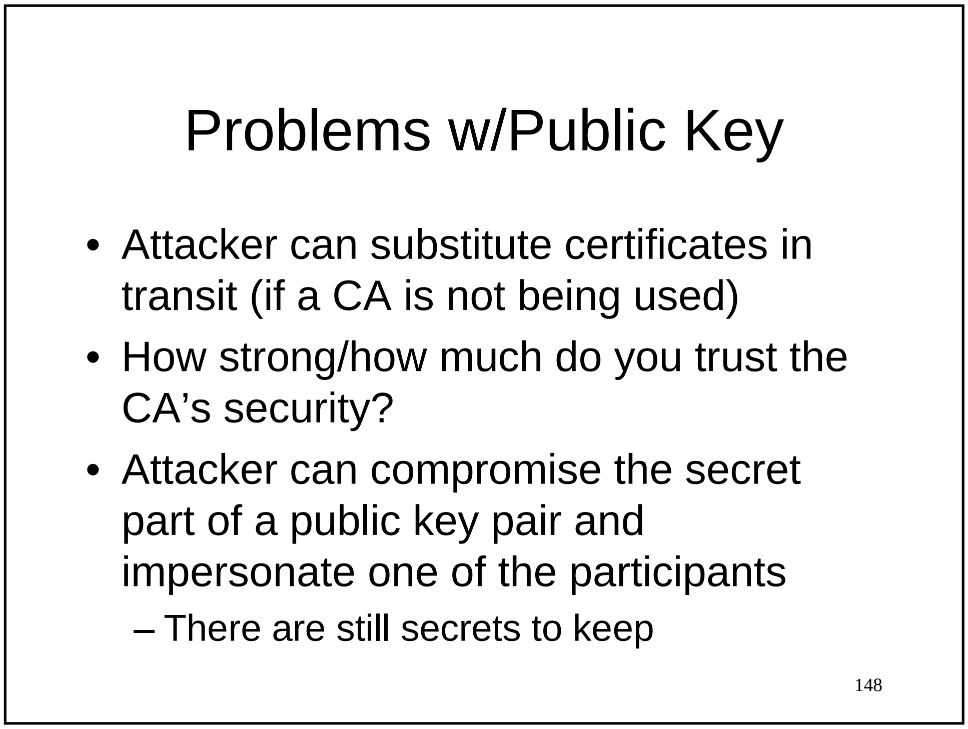
- Public key pairs may be used to “sign” a message by encrypting it with the secret key
- Recipient can check signature by decrypting with the public key
  - Usually instead of encrypting the entire document a cryptographic hash function is applied and the result is encrypted

146



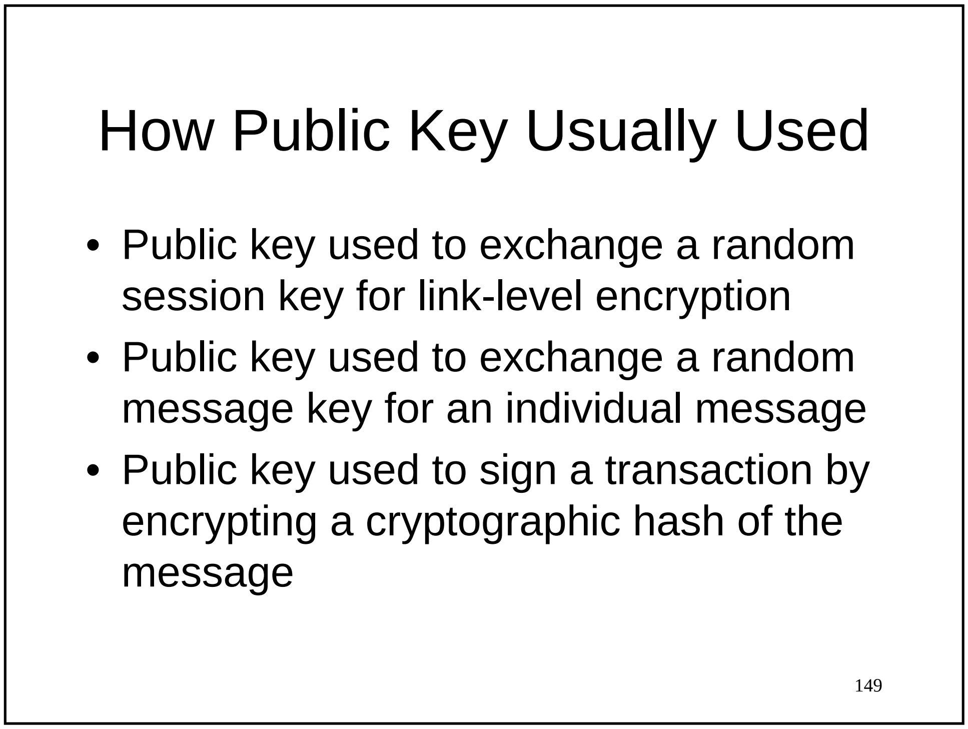
Public Key Certificates

- A “certificate” is a copy of someone’s public key (along with other information) that has been signed by a Certification Authority (CA)
- CA’s certificates signed by other CAs, etc. forming a Certification Hierarchy
  - Global hierarchy still sorting itself out and probably will never happen



Problems w/Public Key

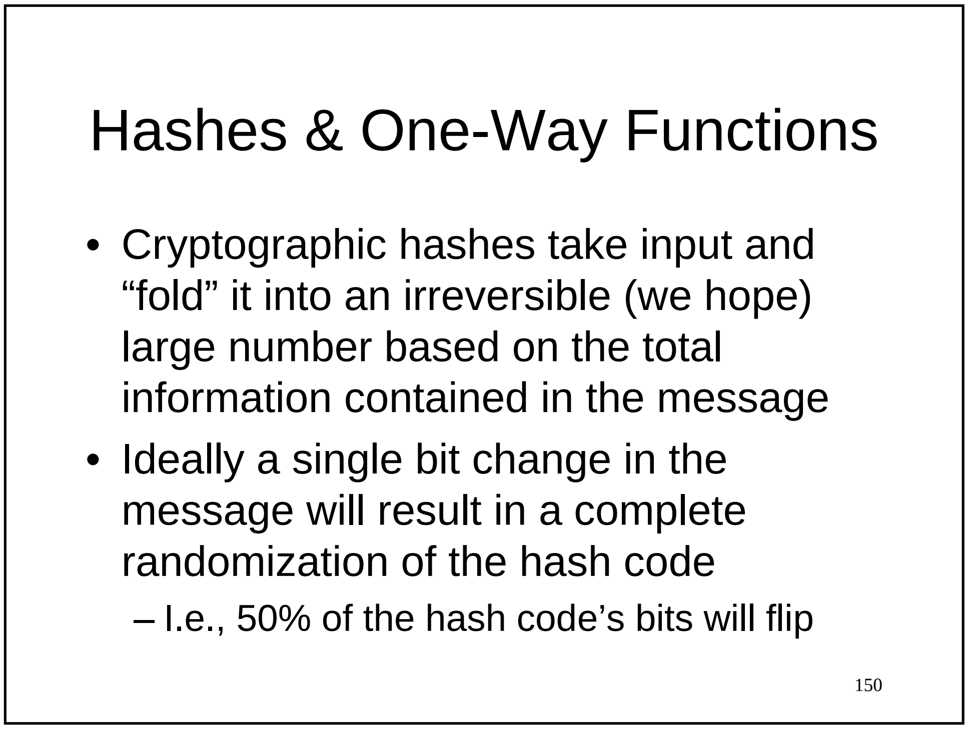
- Attacker can substitute certificates in transit (if a CA is not being used)
- How strong/how much do you trust the CA’s security?
- Attacker can compromise the secret part of a public key pair and impersonate one of the participants
  - There are still secrets to keep



How Public Key Usually Used

- Public key used to exchange a random session key for link-level encryption
- Public key used to exchange a random message key for an individual message
- Public key used to sign a transaction by encrypting a cryptographic hash of the message

149



Hashes & One-Way Functions

- Cryptographic hashes take input and “fold” it into an irreversible (we hope) large number based on the total information contained in the message
- Ideally a single bit change in the message will result in a complete randomization of the hash code
  - – I.e., 50% of the hash code’s bits will flip

150

MD5

- MD5 is very popular cryptographic hash function
  - High performance
  - Freely available
  - Believed to be quite strong
  - Very easy to use

MD5 In Action

```
MD5_CTX     ct;
char        output[16];

extern char *hex(char *);

foo(char *s) {
      MD5_INIT(&ct);
      MD5_Update(&ct,s,strlen(s));
      MD5_Final(output,&ct);
      printf(“MD5(\”%s\”) is %s\n”,s,hex(out));
}
```

Randomness

- PseudoRandom numbers are used frequently in public key systems
- If a “random” number used is not unpredictable then it may be brute-force searched - don’t do this:

```
time_t      now;
time(&now);
srandom(now);
key = random();
```

Good PRNGs

- ***Ok:*** Take a variety of system states and crunch them through MD5
- ***Good:*** Take the contents of the secret message and some system states and crunch them through MD5
- ***Good:*** use /dev/random
- ***Best:*** Use something like `truerand()` which times system interrupt latencies

Key exchange

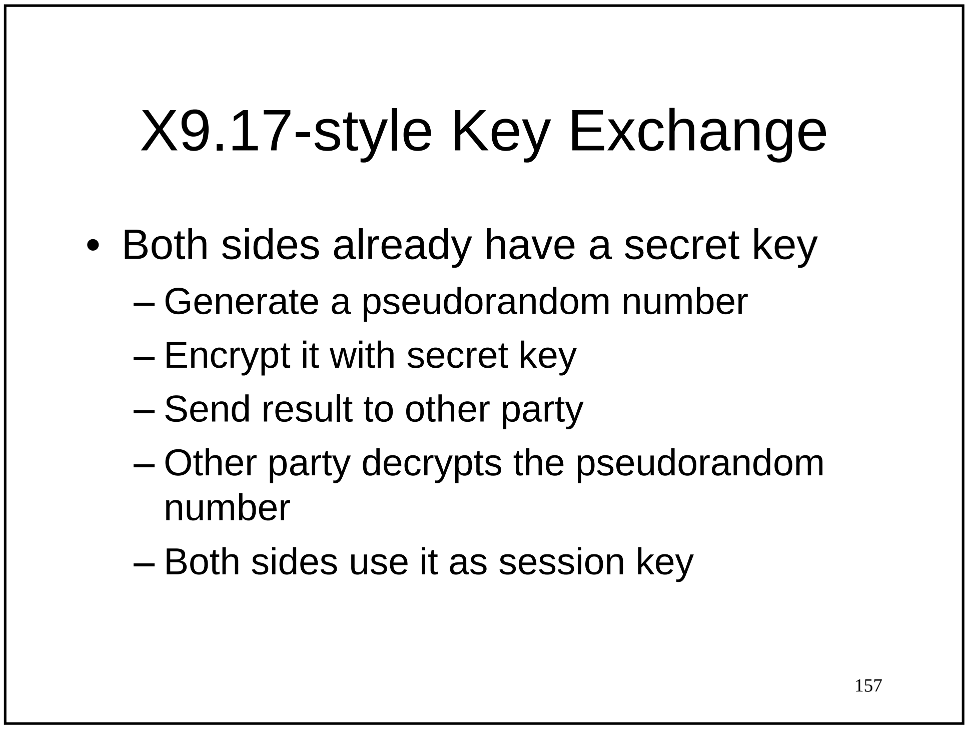
- Diffie-Hellman or RSA public key exchanges
  - Advantage: good
  - Disadvantage: slow, and patented
- Secret key based key exchanges (ANSI X9.17)
  - Advantage: fast, free, and easy
  - Disadvantage: needs a shared secret

155

Diffie-Hellman / RSA Code

- Use RSAREF
- Unfortunately the interfaces to public key routines are perforce complex
  - (more so than are coverable here)
- Examples to look at:
  - ssh
  - pgp
  - Stel

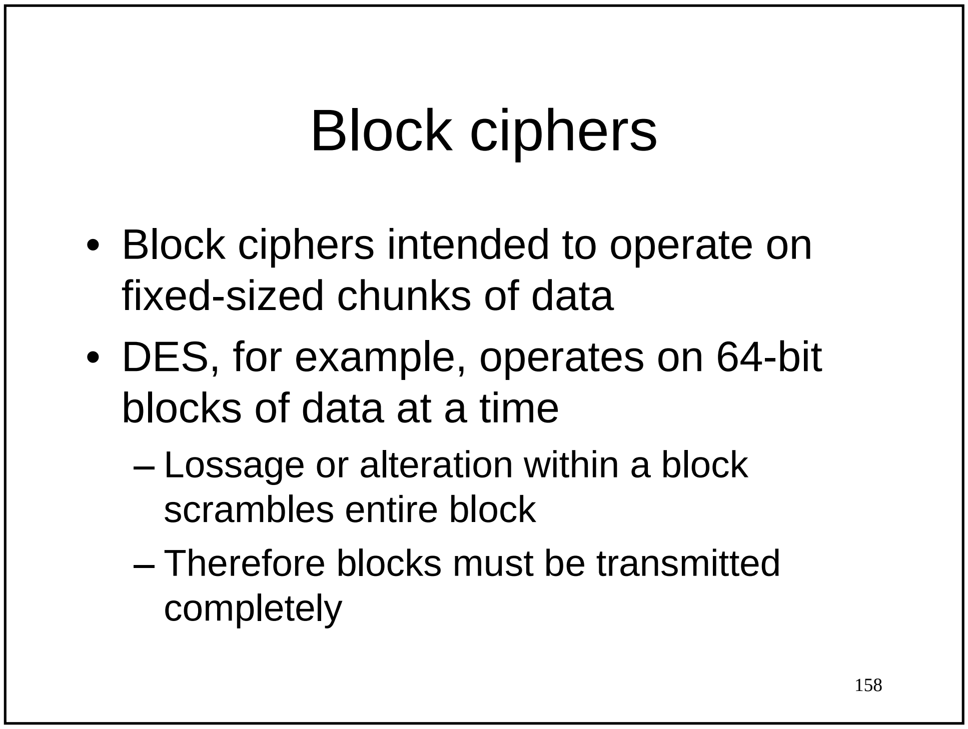
156



X9.17-style Key Exchange

- Both sides already have a secret key
  - Generate a pseudorandom number
  - Encrypt it with secret key
  - Send result to other party
  - Other party decrypts the pseudorandom number
  - Both sides use it as session key

157



Block ciphers

- Block ciphers intended to operate on fixed-sized chunks of data
- DES, for example, operates on 64-bit blocks of data at a time
  - Lossage or alteration within a block scrambles entire block
  - Therefore blocks must be transmitted completely

158

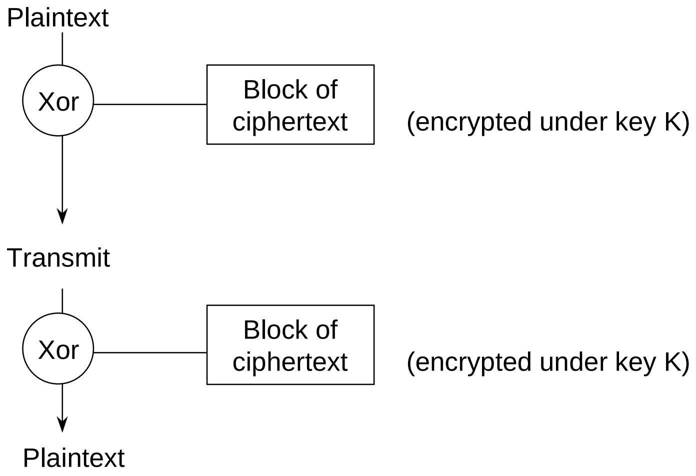
Basic DES Blocks

```
des_cblock         k;
des_keyschedule    ksched;
char               kp[128];
char               jnk[8];
char               ojnk[8];
/* no error checking - this is an example */
fprintf(stderr,”? “,);
fgets(kp,sizeof(kp),stdin);
strcpy(jnk,”spamit!”);
des_string_to_key(kp,k);
des_set_key(k,ksched);
des_ecb_encrypt(jnk,ojnk,ksched,DES_ENCRYPT);
```

Stream ciphers

- Streaming capability can be added by using cipher engine to produce blocks of ciphertext against which data is XORed
  - – No need to send data a block at a time
  - – Data can be sent a bit (or more often byte) at a time with high performance
  - – Security is equal to encryption algorithm’s

Stream Cipher



Plaintext

Xor

Block of ciphertext

(encrypted under key K)

Transmit

Xor

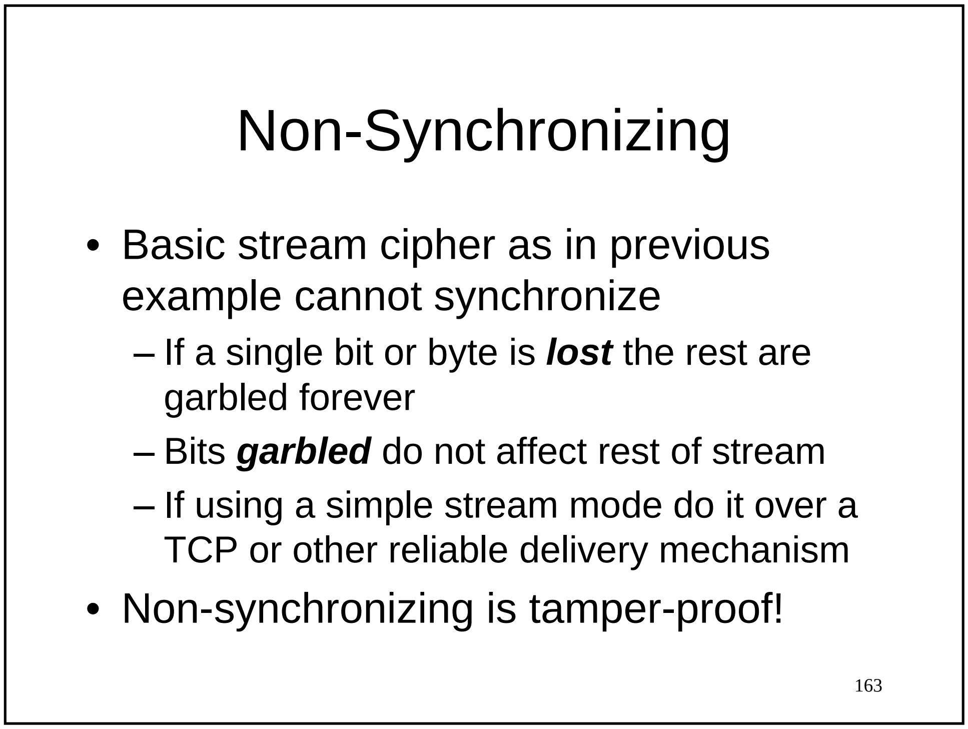
Block of ciphertext

(encrypted under key K)

Plaintext

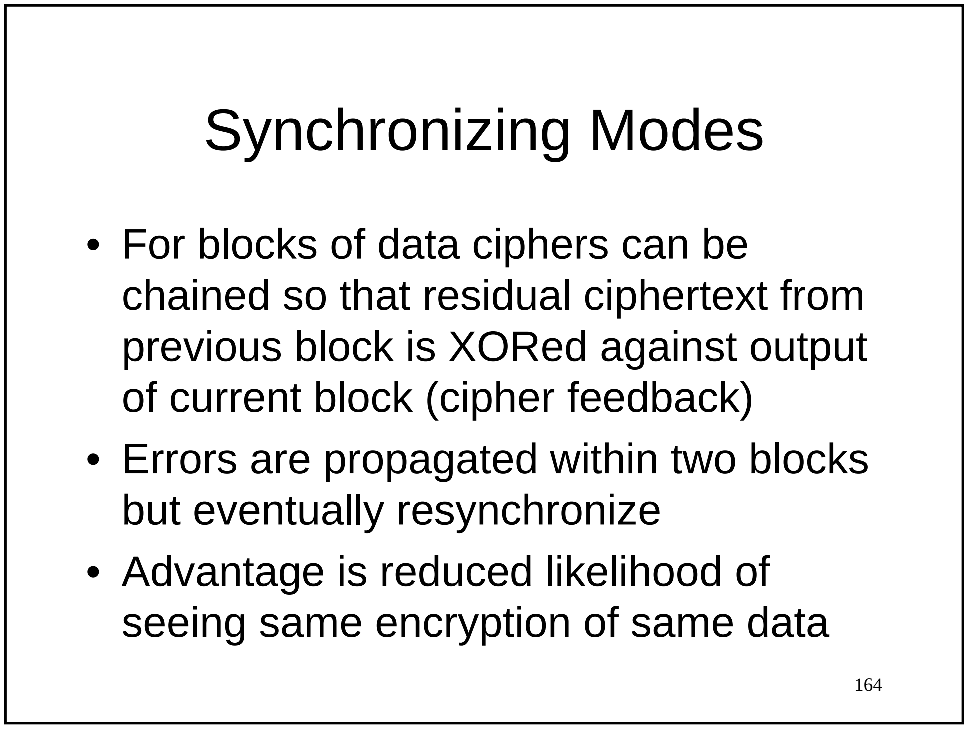
Cipher synchronization

- What happens if a bit or a byte of ciphertext is lost in transmission?
  - Sometimes the rest of the stream of data is unrecoverable
  - Other times the cipher may resynchronize
- Implications of synchronizing versus non-synchronizing are important



Non-Synchronizing

- Basic stream cipher as in previous example cannot synchronize
  - If a single bit or byte is ***lost*** the rest are garbled forever
  - Bits ***garbled*** do not affect rest of stream
  - If using a simple stream mode do it over a TCP or other reliable delivery mechanism
- Non-synchronizing is tamper-proof!



Synchronizing Modes

- For blocks of data ciphers can be chained so that residual ciphertext from previous block is XORed against output of current block (cipher feedback)
- Errors are propagated within two blocks but eventually resynchronize
- Advantage is reduced likelihood of seeing same encryption of same data

Synchonizing Errors

Original text:

```
Gregory led him down a low, vaulted passage, at the end of
which was the red light.  It was an enormous crimson lantern,
nearly as big as a fireplace, fixed over a small but heavy iron door.
In the door there was a sort of hatchway or grating, and on this
```

Edit Process:

```
. des -e thursday.txt > thursday.des
Enter key:
Verifying password Enter key:
. uuencode thursday.des thursday.des > thursday.des.uu
. vi thursday.des.uu
thursday.des.uu: 208 lines, 12695 characters.
. uudecode thursday.des.uu
. des -d thursday.des > thursday.new
Enter key:
. more thursday.new
```

165

Synchonizing Errors *(cont)*

Final text:

```
Gregory led him down a low, vaulted passage, at the end of
which was the red light.  It was an enormous crimson lantern,
nearly as big as a fireplace, fixed over a small but xbd^@
lg0x990xed^Xoon do0xa10xf2. In the door there was a sort of
hatchway or grating, and on this
```

- This could be a serious problem if the data being transmitted were a financial transaction

166

Problems with Synchronizing

- If a ***sophisticated*** attacker can obtain the key for a session it is possible to insert traffic into the session
  - Desynchronize the cipherstream with 2 blocks
  - Insert your data
  - Let the stream resynchronize
  - Victim sees 2 periods of disruption

167

Cipher Block Chaining

- Initial value used to fill a block (initialization vector)
- Each block is encrypted then XORed with IV and transmitted
- Block is then saved for next XORing of next block
- Makes each block depend on previous block

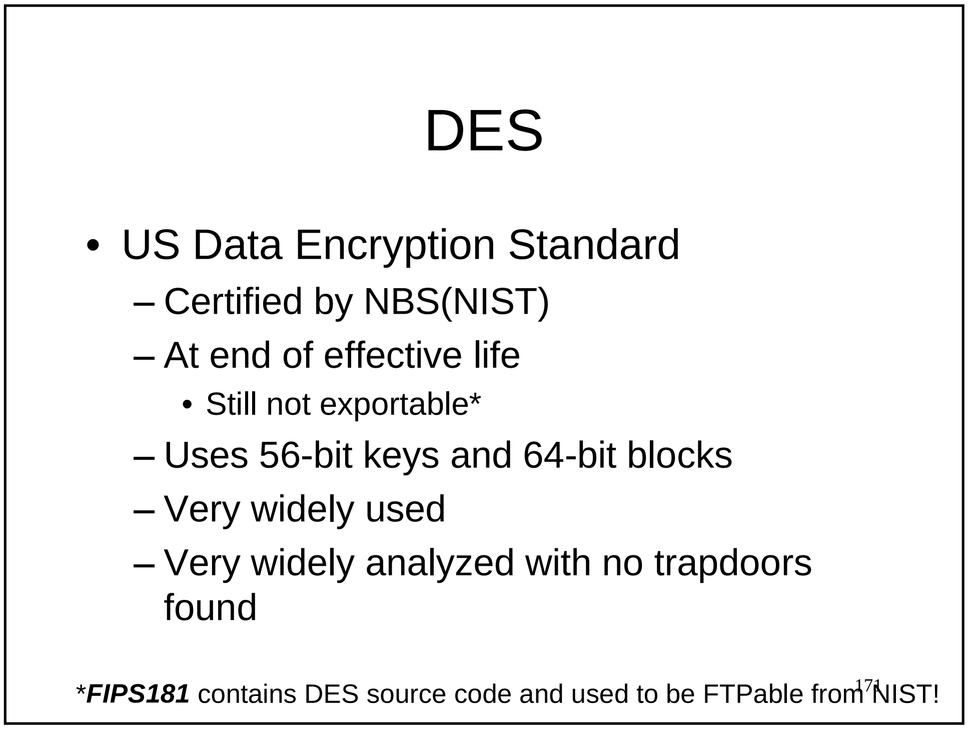
168

Encrypting File Access

- For encrypting files for random access
  - Use large size blocks
  - Encrypt each block individually using a feedback mode with an initialization based on block number
  - Prevents large areas that encrypt same
  - Predictable for access

Algorithms

- Do ***not*** try to write your own encryption routines
  - It’s hard
- Choose widely used publicly available algorithms that have been extensively examined
  - Use publicly available implementations

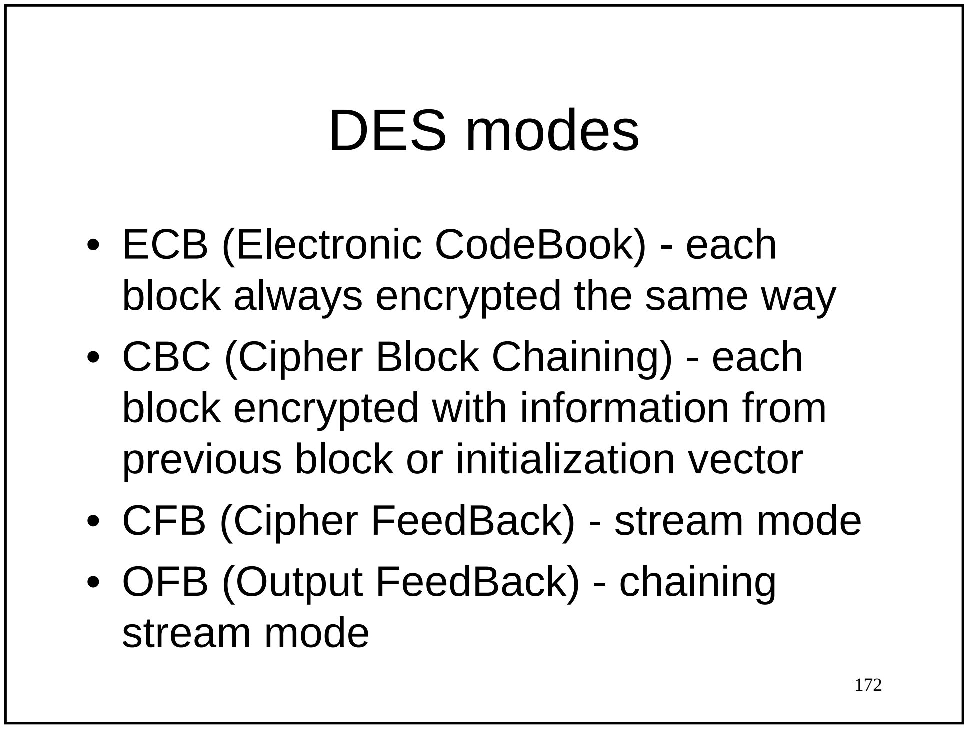


DES

- US Data Encryption Standard
  - Certified by NBS(NIST)
  - At end of effective life
    - Still not exportable*
  - Uses 56-bit keys and 64-bit blocks
  - Very widely used
  - Very widely analyzed with no trapdoors found

FIPS181 contains DES source code and used to be FTPable from NIST!* —

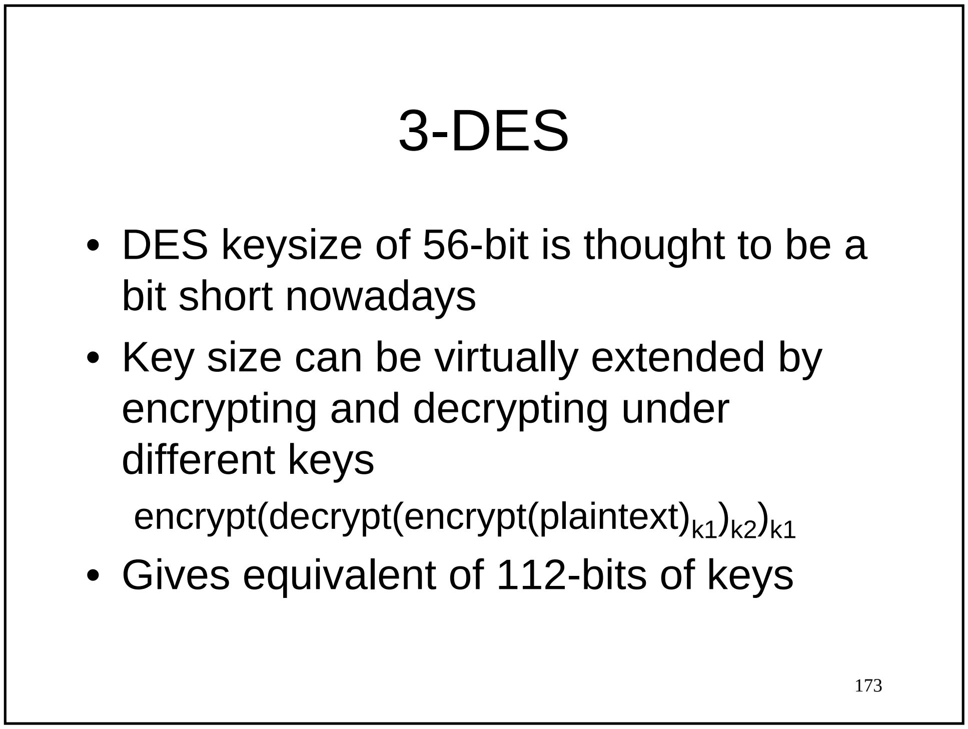
171



DES modes

- ECB (Electronic CodeBook) - each block always encrypted the same way
- CBC (Cipher Block Chaining) - each block encrypted with information from previous block or initialization vector
- CFB (Cipher FeedBack) - stream mode
- OFB (Output FeedBack) - chaining stream mode

172

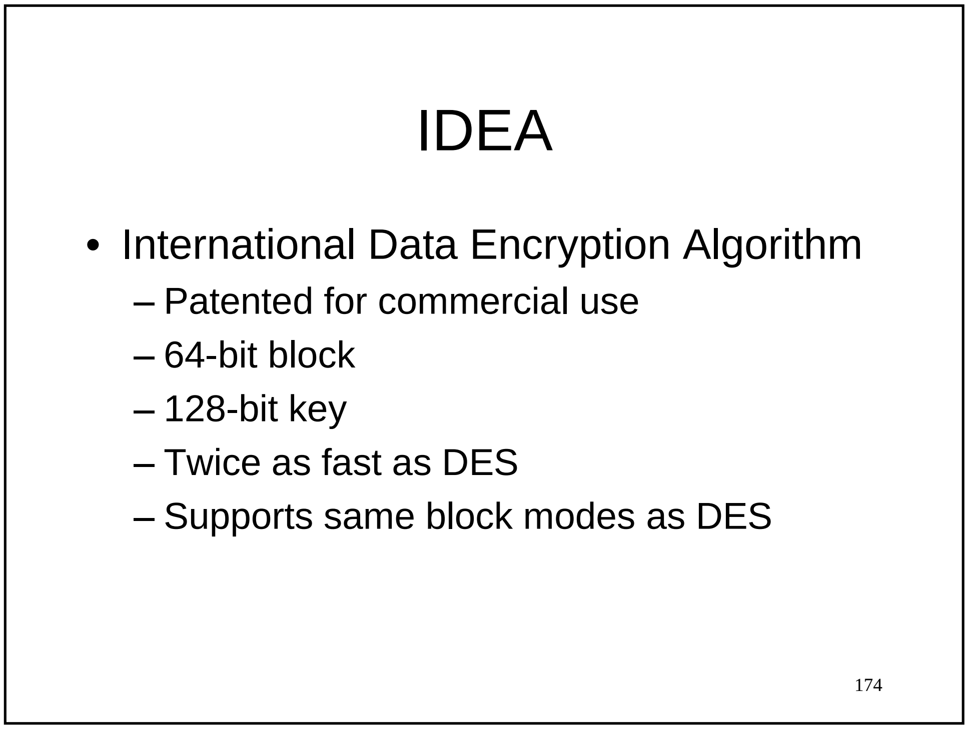


3-DES

- DES keysize of 56-bit is thought to be a bit short nowadays
- Key size can be virtually extended by encrypting and decrypting under different keys

  $\text{encrypt(decrypt(encrypt(plaintext)}_{k1}\text{)}_{k2}\text{)}_{k1}$
- Gives equivalent of 112-bits of keys

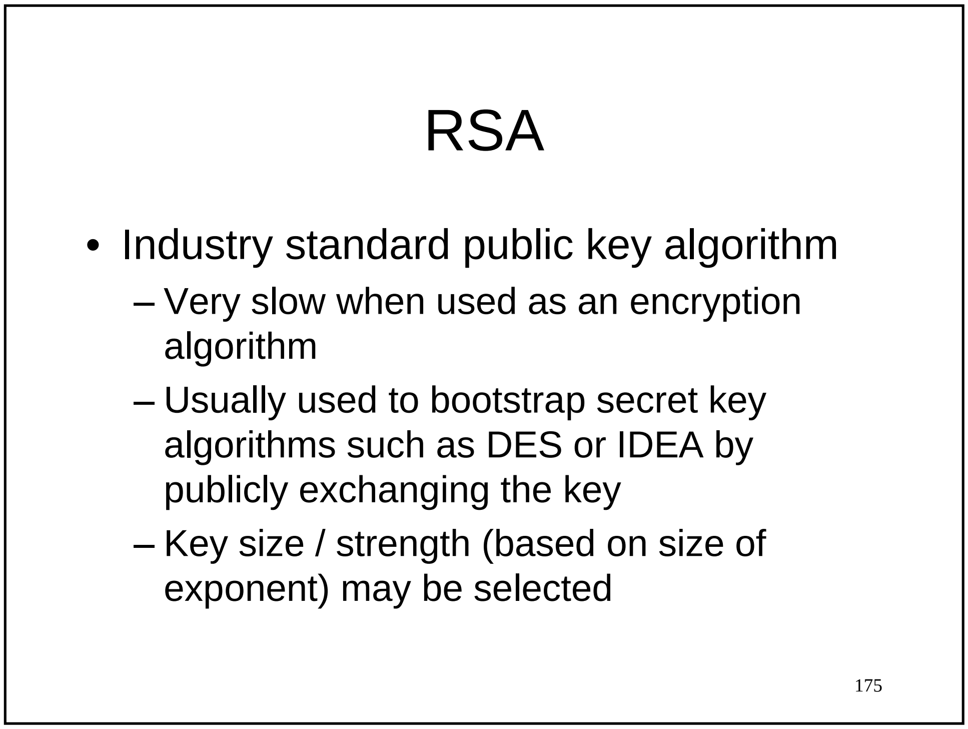
173



IDEA

- International Data Encryption Algorithm
  - Patented for commercial use
  - 64-bit block
  - 128-bit key
  - Twice as fast as DES
  - Supports same block modes as DES

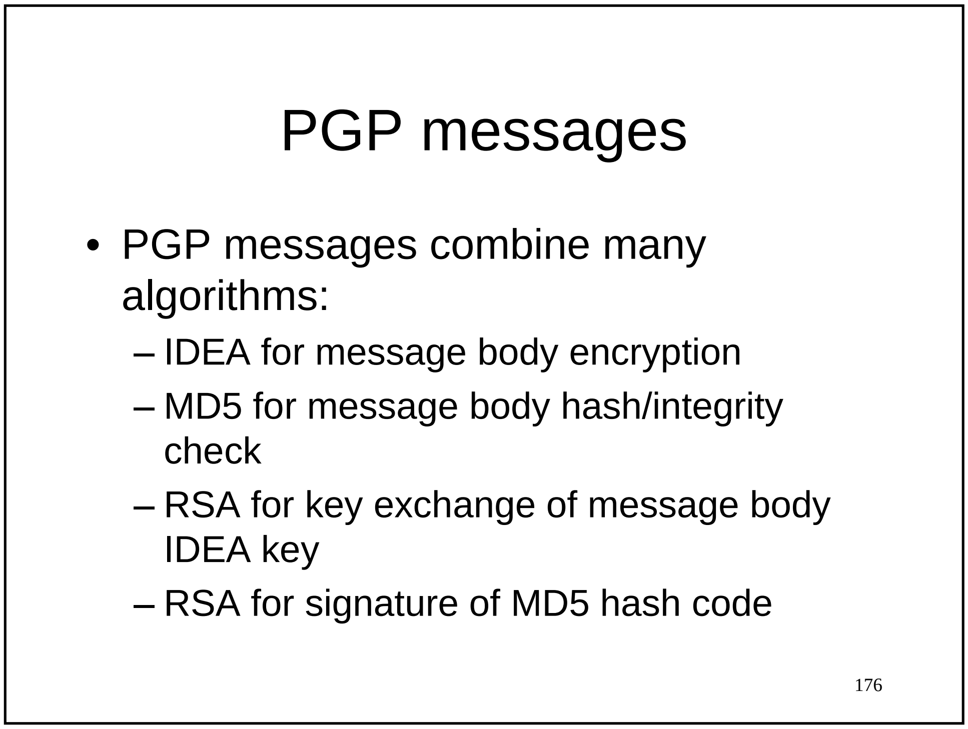
174



RSA

- Industry standard public key algorithm
  - Very slow when used as an encryption algorithm
  - Usually used to bootstrap secret key algorithms such as DES or IDEA by publicly exchanging the key
  - Key size / strength (based on size of exponent) may be selected

175



PGP messages

- PGP messages combine many algorithms:
  - IDEA for message body encryption
  - MD5 for message body hash/integrity check
  - RSA for key exchange of message body IDEA key
  - RSA for signature of MD5 hash code

176

Authentication

- For application software several popular techniques are used
  - Challenge/response
  - Public key signature
  - One-time keys
- No matter what, some kind of secure value needs to be stored at both ends or in the user’s head

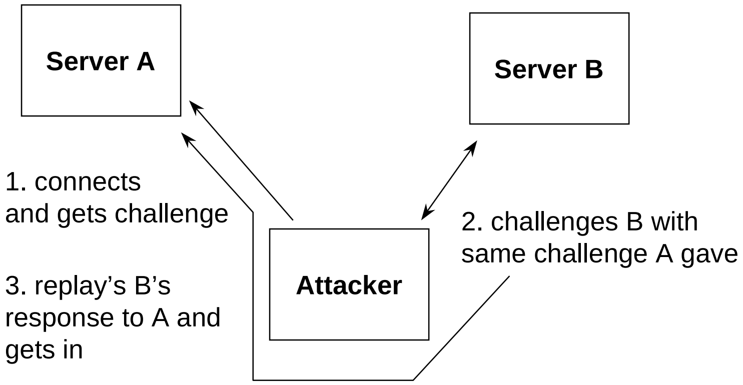
177

Authentication: Challenge/response

- Client connects to server
- Server sends client a random number
- Client encrypts random number with secret key and sends result to server
- Server encrypts the random number with its copy of the secret key and compares results
- If they match the user is authenticated

178

Challenge/response laundering



Server A

Server B

Attacker

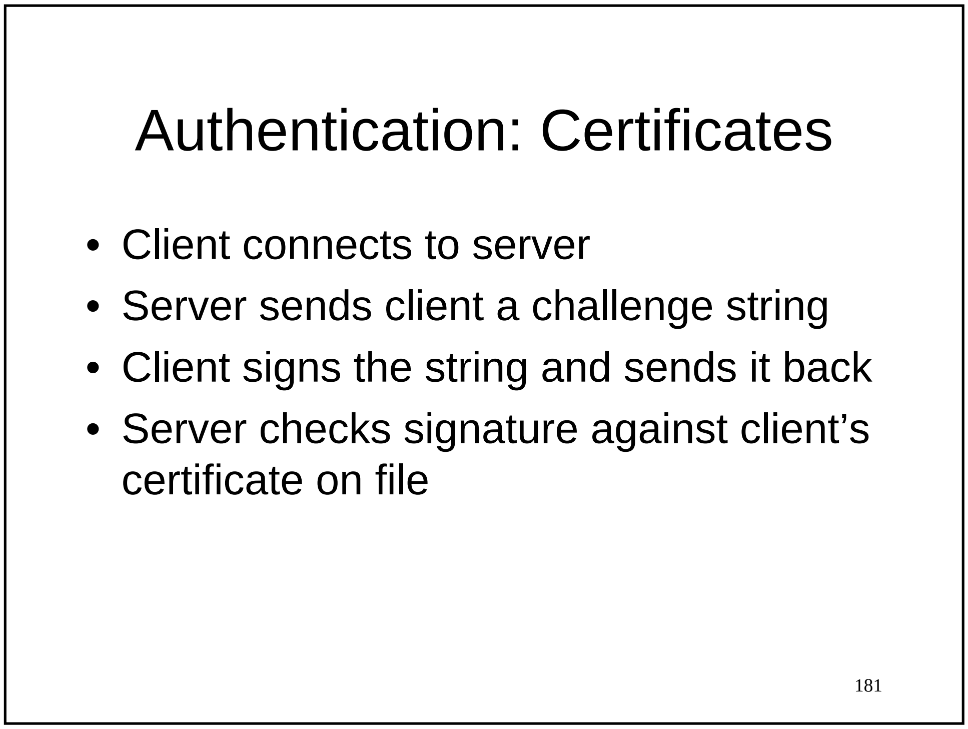
1. connects and gets challenge

2. challenges B with same challenge A gave

3. replay's B's response to A and gets in

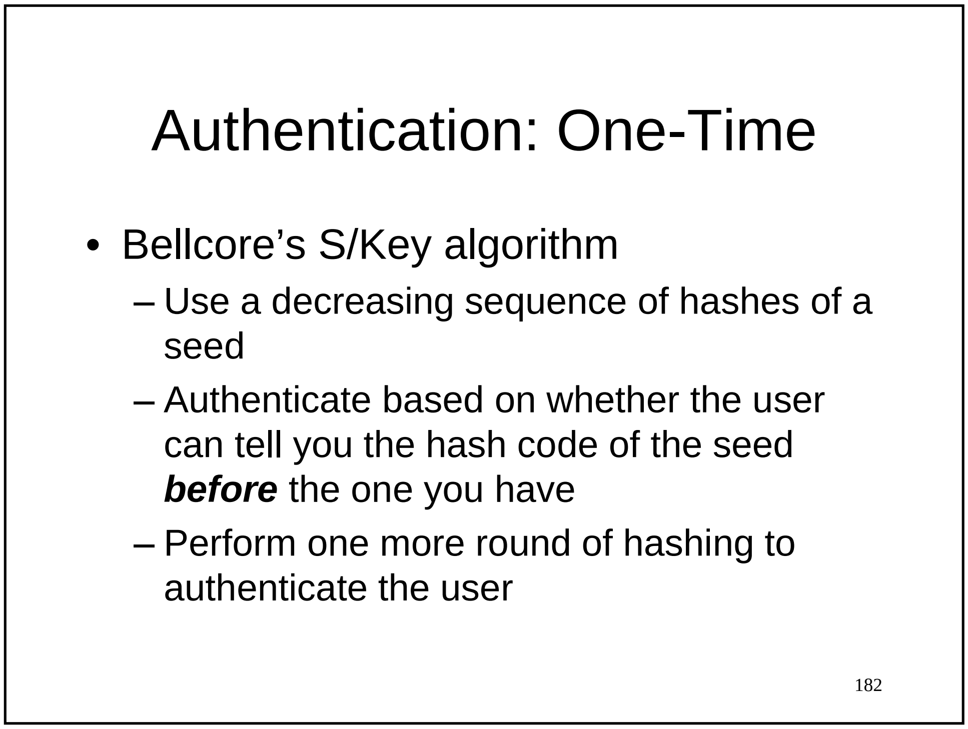
Prevent Laundering

- Never be both a challenge/response client ***and*** server
- Use different keys between entities



Authentication: Certificates

- Client connects to server
- Server sends client a challenge string
- Client signs the string and sends it back
- Server checks signature against client’s certificate on file



Authentication: One-Time

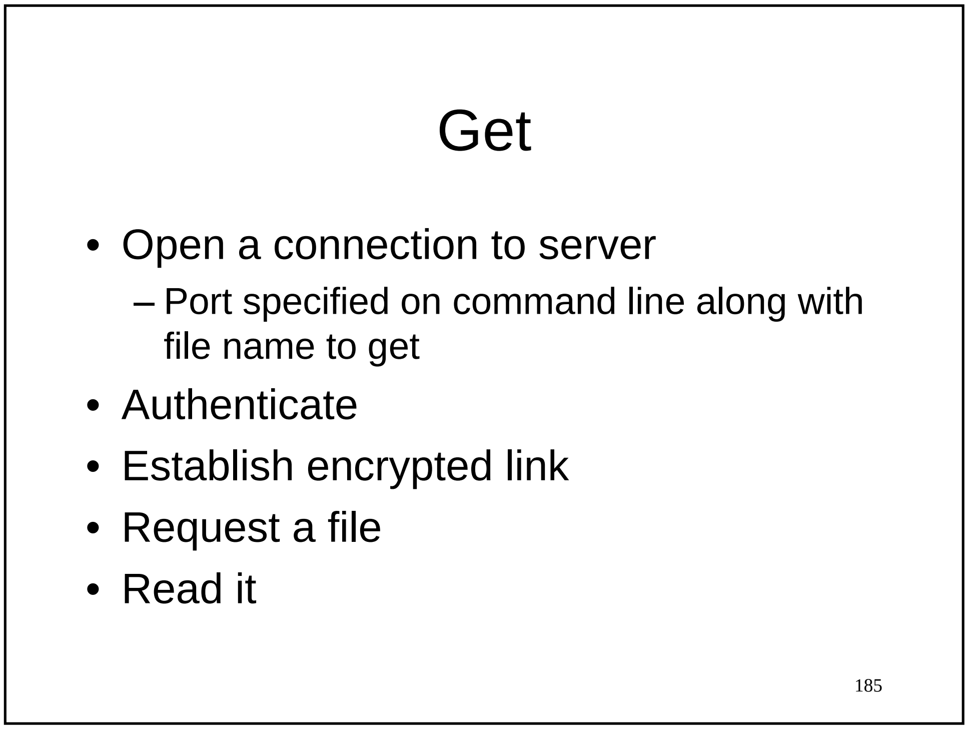
- Bellcore’s S/Key algorithm
  - Use a decreasing sequence of hashes of a seed
  - Authenticate based on whether the user can tell you the hash code of the seed ***before*** the one you have
  - Perform one more round of hashing to authenticate the user

Forward One-Time

- Set up a secret once
- Client preceeds each message with a sequence number encrypted with the secret
- Sequence number is incremented at each end
- Prevents rollback or replay

A Simple Example

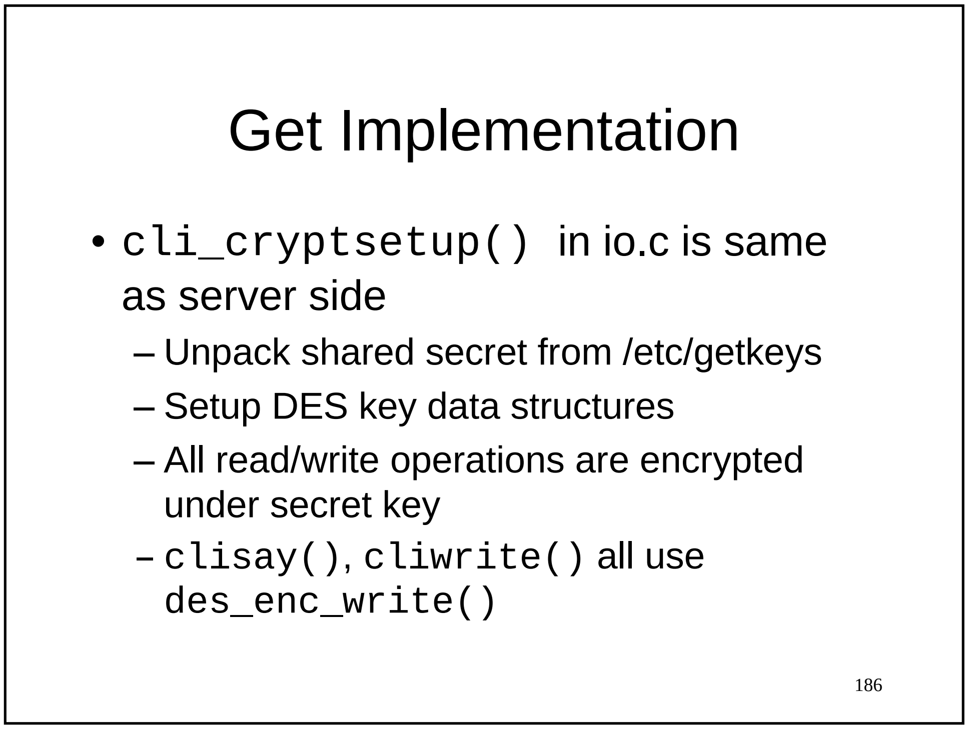
- Get/Put a simple daemon for quick and dirty encrypted file transfer
  - Use secret key for authentication and encryption
  - Use very simple dialog with single stream
  - Use standard DES for encryption (Eric Young version)
  - Minimal fanciness



Get

- Open a connection to server
  - Port specified on command line along with file name to get
- Authenticate
- Establish encrypted link
- Request a file
- Read it

185



Get Implementation

- `cli_cryptsetup()` in io.c is same as server side
  - Unpack shared secret from /etc/getkeys
  - Setup DES key data structures
  - All read/write operations are encrypted under secret key
  - `clisay()`, `cliwrite()` all use `des_enc_write()`

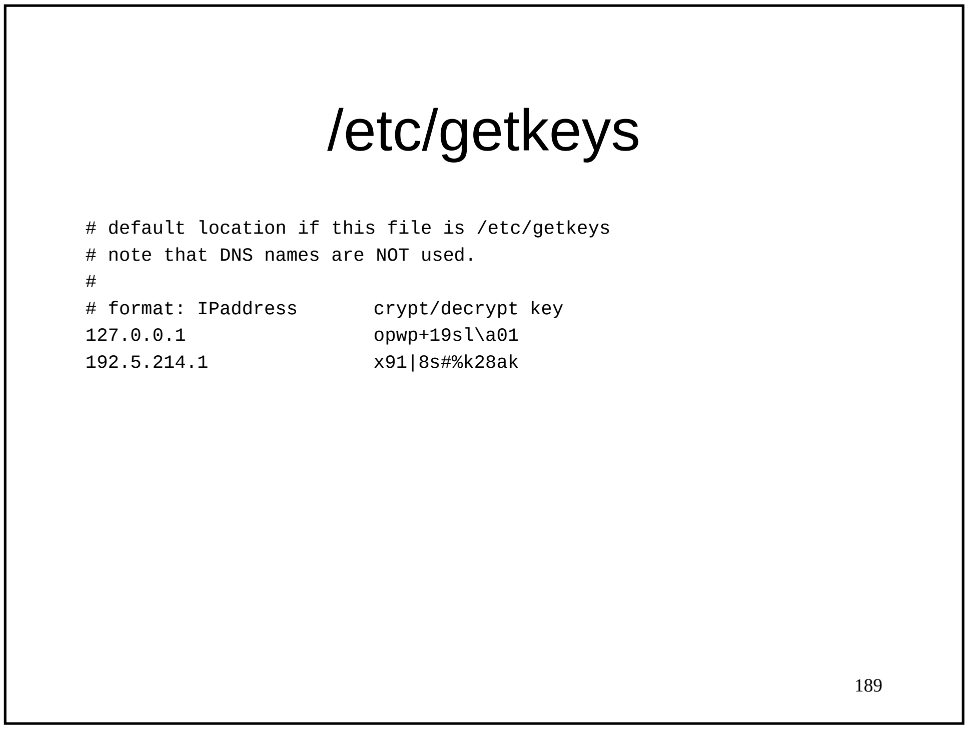
186

Get *(cont)*

- Uses a single socket with encrypted end-to-end communication
- Challenge/response ensures no replay
- File size is transmitted as part of setup
  - Exact byte counts read
  - End of file should have a synchronous “bye” message
- To compromise, attacker needs key

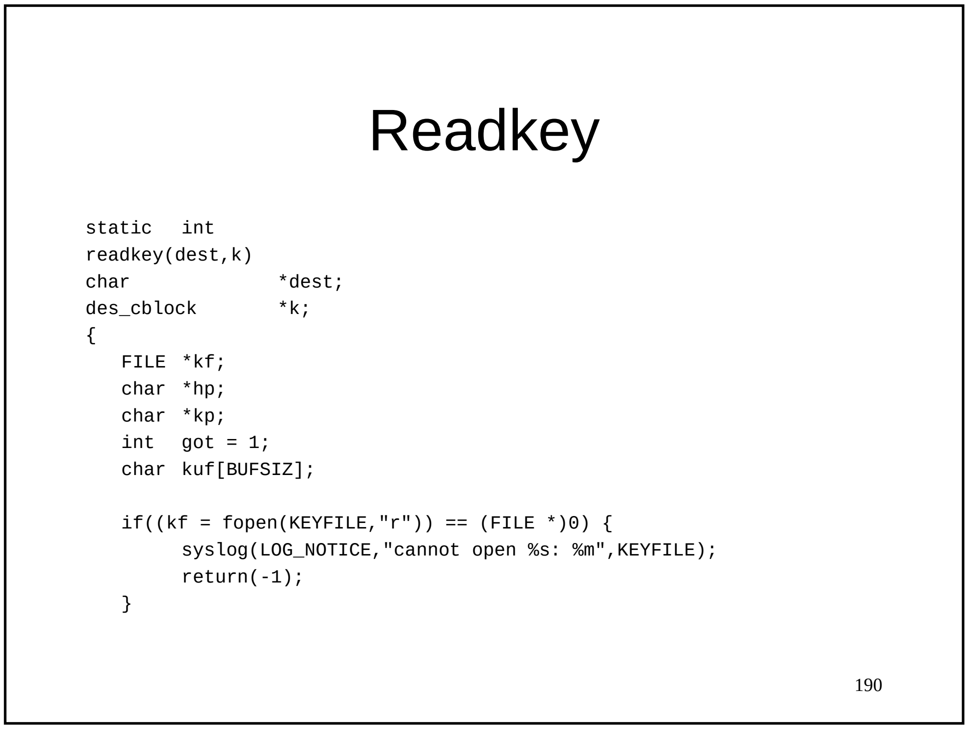
Cli_Cryptsetup

```
cli_cryptsetup(h)
char    *h;
{
   return(srv_cryptsetup(h));
}
/* this may do a key exchange or whatever */
srv_cryptsetup(h)
char    *h;
{
   if(readkey(h,&kblock)) /* we will study readkey next */
        return(1);
   des_set_key(kblock,ksched);
   bzero(kblock,sizeof(kblock));
   bzero(ivec_r,sizeof(ivec_r));
   bzero(ivec_w,sizeof(ivec_w));
   return(0);
}
```



/etc/getkeys

```
# default location if this file is /etc/getkeys
# note that DNS names are NOT used.
#
# format: IPaddress       crypt/decrypt key
127.0.0.1                 opwp+19sl\a01
192.5.214.1               x91|8s#%k28ak
```



Readkey

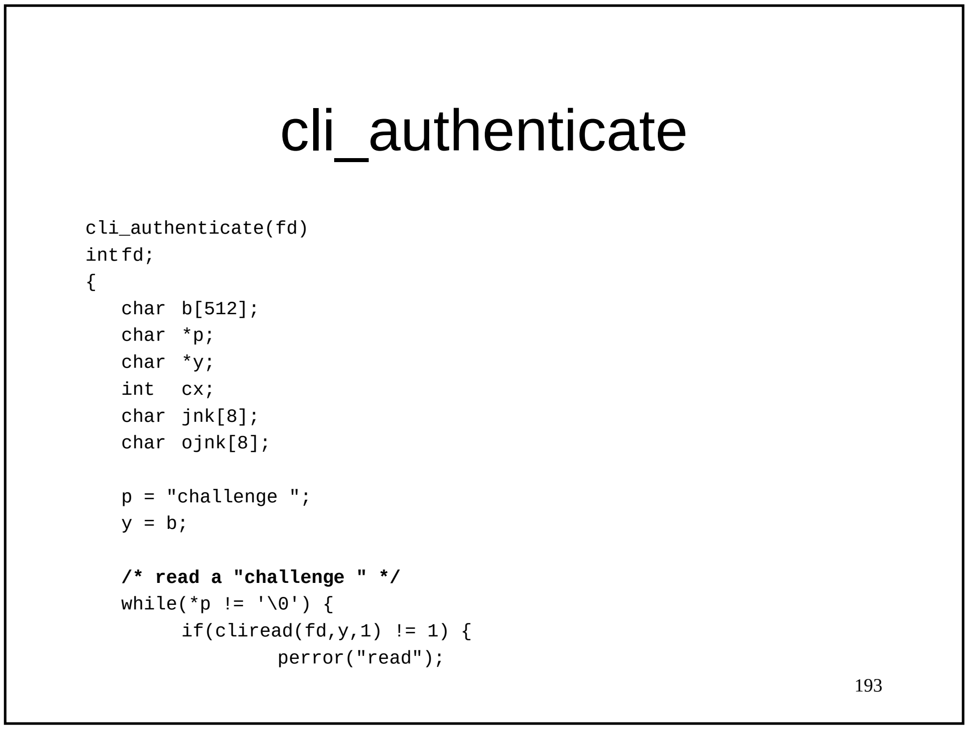
```
static  int
readkey(dest,k)
char            *dest;
des_cblock      *k;
{
   FILE *kf;
   char *hp;
   char *kp;
   int  got = 1;
   char kuf[BUFSIZ];

   if((kf = fopen(KEYFILE,"r")) == (FILE *)0) {
        syslog(LOG_NOTICE,"cannot open %s: %m",KEYFILE);
        return(-1);
   }
```

Readkey

```
    while(fgets(kuf,sizeof(kuf),kf)) {
         if(kuf[0] == '#')
                 continue;
         if((hp = strtok(kuf," \t\n")) == (char *)0)
                 continue;
         if((kp = strtok((char *)0," \t\n")) == (char *)0)
                 continue;
         if(!strcasecmp(dest,hp)) {
                 des_string_to_key(kp,k);
                 got = 0;
                 break;
         }
    }
    if(got == 1)
         syslog(LOG_NOTICE,"no key for %s",dest);
    fclose(kf); return(got);
}
```

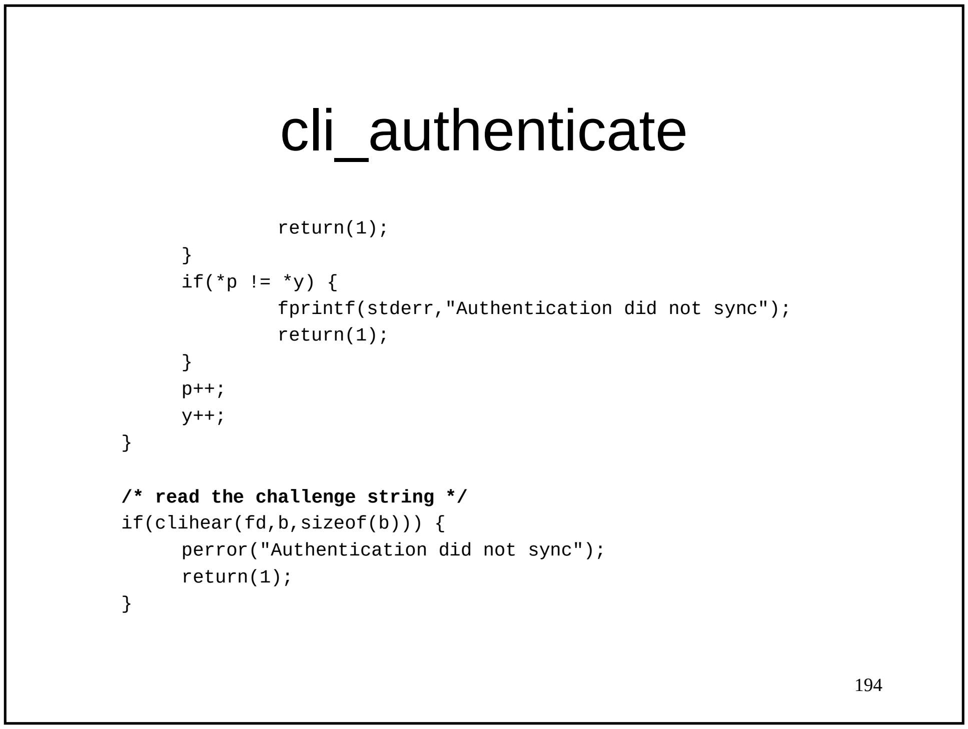
Get *(cont)*

- `cli_authenticate()` in io.c does challenge/response with server
  - Read a challenge
    - Note that if encryption keys are not matched the challenge read attempt will fail and client will disconnect
    - This is slightly sub-optimal since it is known plaintext
  - Encrypt it in ECB mode and return it



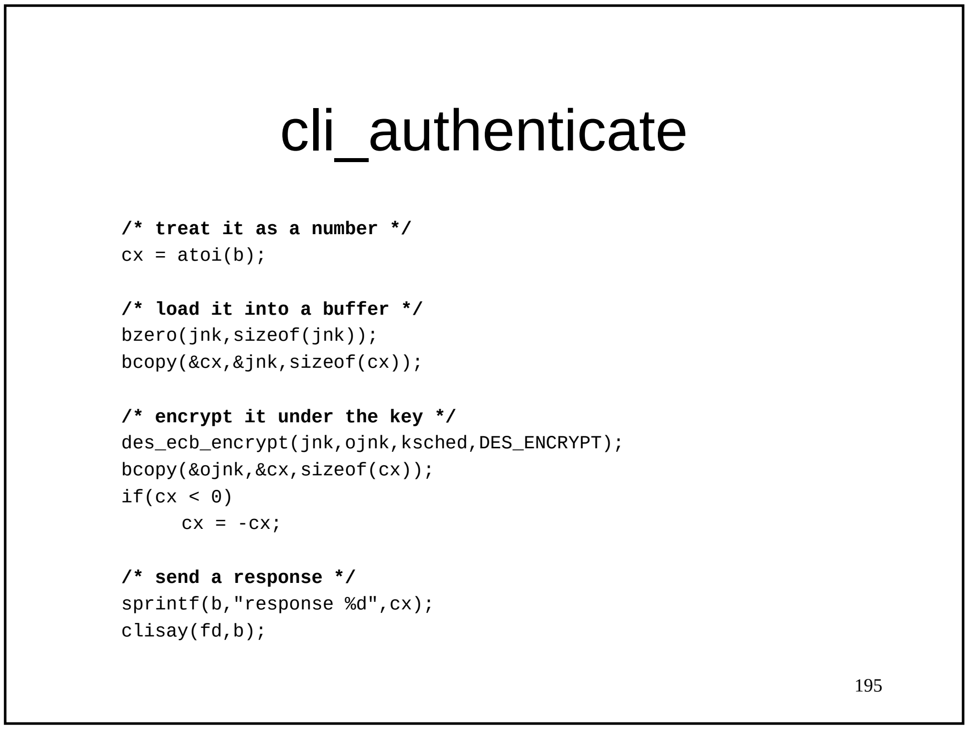
cli_authenticate

```
cli_authenticate(fd)
int	fd;
{
   char  b[512];
   char  *p;
   char  *y;
   int   cx;
   char  jnk[8];
   char  ojnk[8];

   p = "challenge ";
   y = b;

   /* read a "challenge " */
   while(*p != '\0') {
        if(cliread(fd,y,1) != 1) {
                perror("read");
```



cli_authenticate

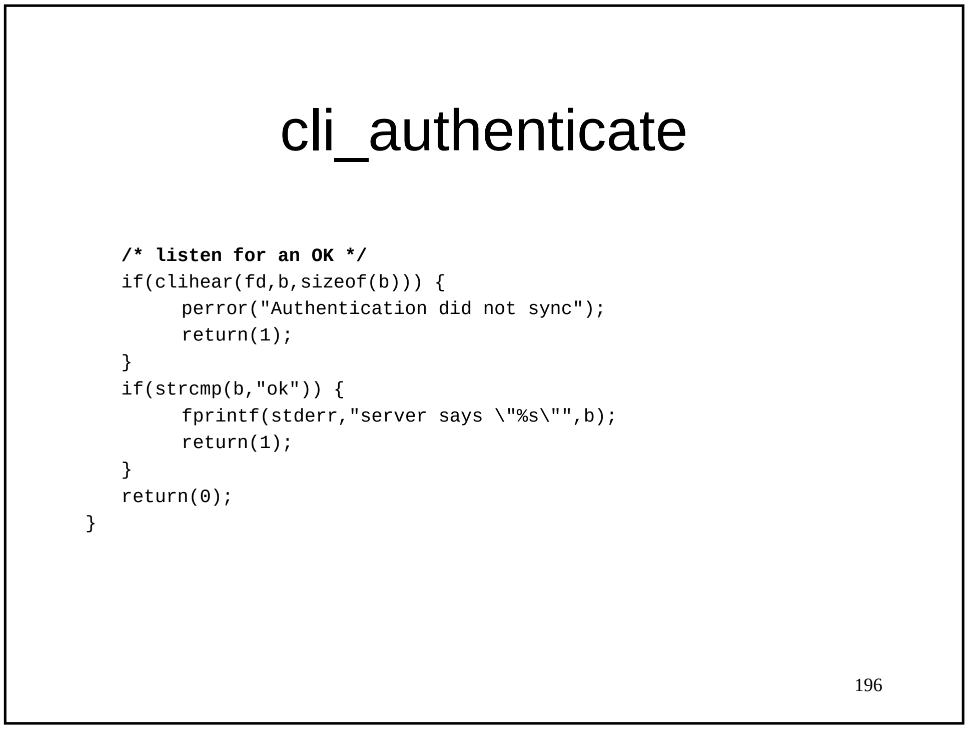
```
                return(1);
        }
        if(*p != *y) {
                fprintf(stderr,"Authentication did not sync");
                return(1);
        }
        p++;
        y++;
   }

   /* read the challenge string */
   if(clihear(fd,b,sizeof(b))) {
        perror("Authentication did not sync");
        return(1);
   }
```



cli_authenticate

```
/* treat it as a number */
cx = atoi(b);

/* load it into a buffer */
bzero(jnk,sizeof(jnk));
bcopy(&cx,&jnk,sizeof(cx));

/* encrypt it under the key */
des_ecb_encrypt(jnk,ojnk,ksched,DES_ENCRYPT);
bcopy(&ojnk,&cx,sizeof(cx));
if(cx < 0)
     cx = -cx;

/* send a response */
sprintf(b,"response %d",cx);
clisay(fd,b);
```

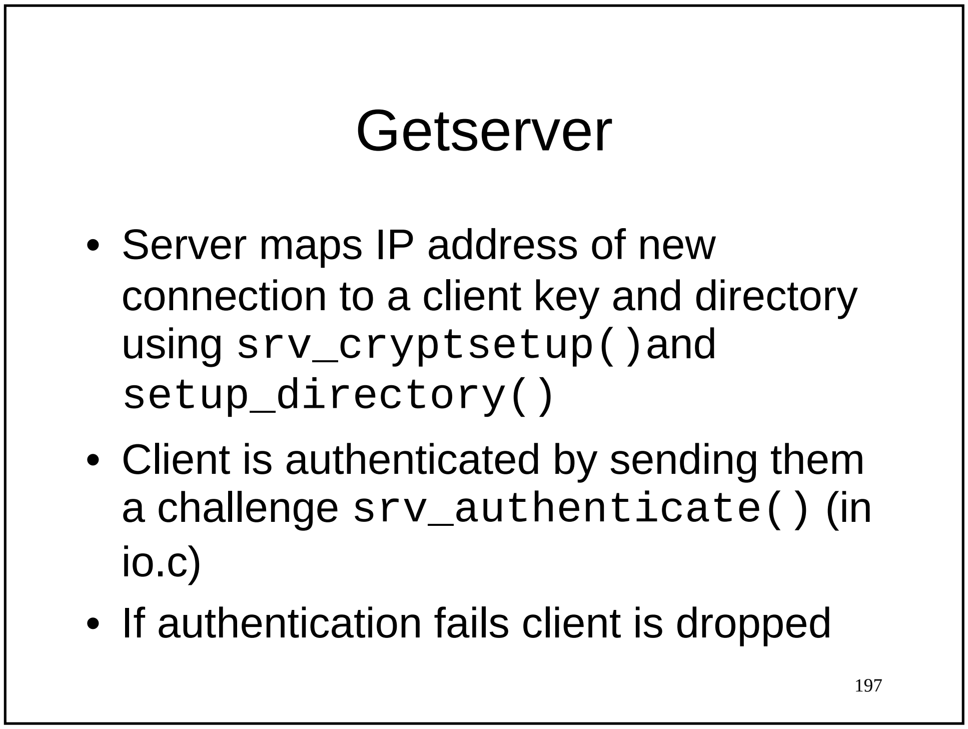
195



cli_authenticate

```
    /* listen for an OK */
    if(clihear(fd,b,sizeof(b))) {
         perror("Authentication did not sync");
         return(1);
    }
    if(strcmp(b,"ok")) {
         fprintf(stderr,"server says \"%s\"",b);
         return(1);
    }
    return(0);
}
```

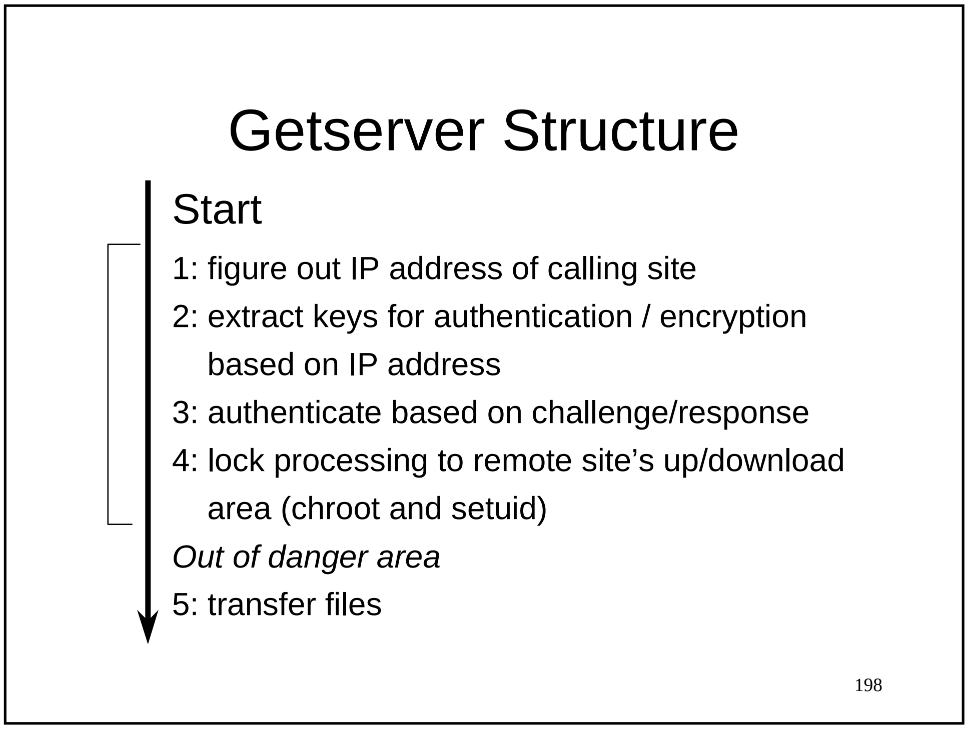
196



Getserver

- Server maps IP address of new connection to a client key and directory using `srv_cryptsetup()`and `setup_directory()`
- Client is authenticated by sending them a challenge `srv_authenticate()` (in io.c)
- If authentication fails client is dropped

197



Getserver Structure

Start

1: figure out IP address of calling site

2: extract keys for authentication / encryption based on IP address

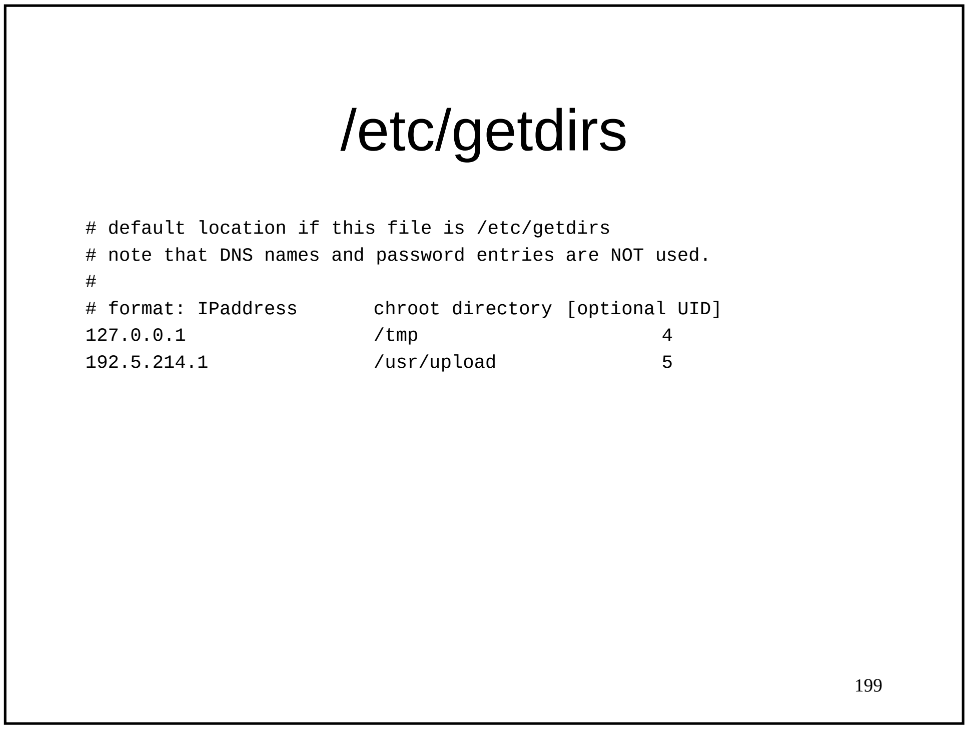
3: authenticate based on challenge/response

4: lock processing to remote site’s up/download area (chroot and setuid)

Out of danger area

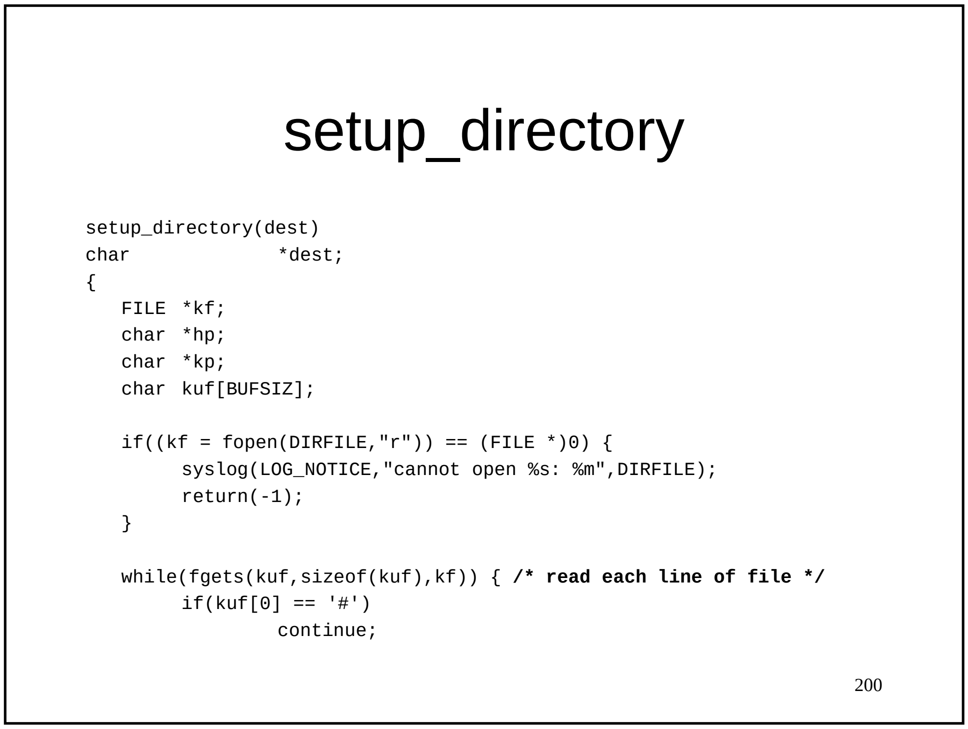
5: transfer files

198



/etc/getdirs

```
# default location if this file is /etc/getdirs
# note that DNS names and password entries are NOT used.
#
# format: IPaddress       chroot directory [optional UID]
127.0.0.1                 /tmp                      4
192.5.214.1               /usr/upload               5
```



setup_directory

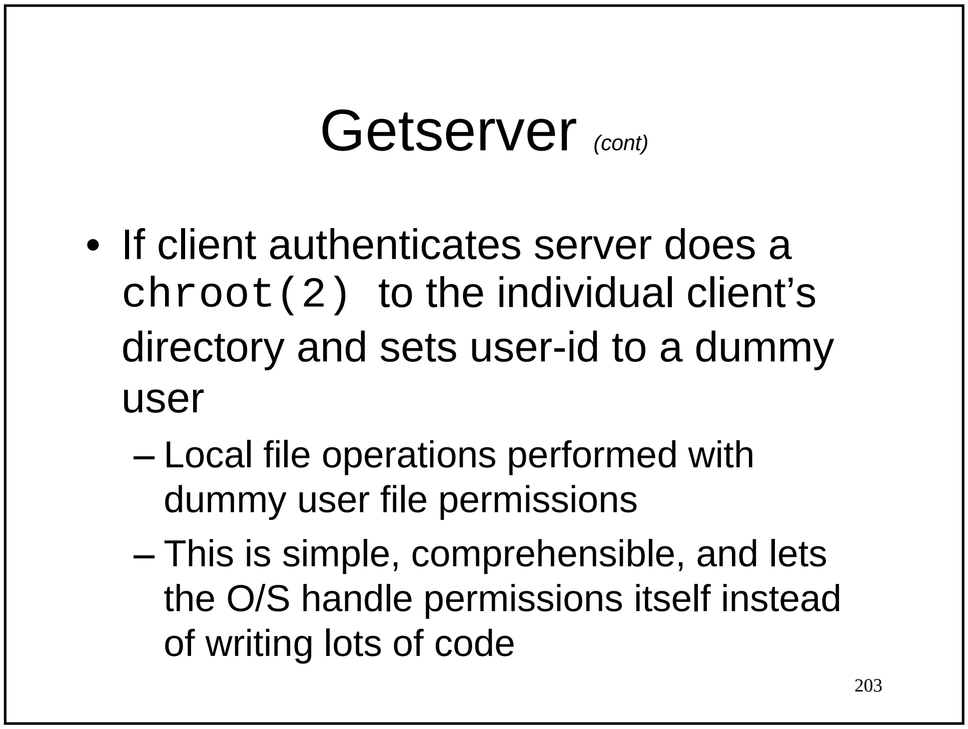
```
setup_directory(dest)
char             *dest;
{
   FILE  *kf;
   char  *hp;
   char  *kp;
   char  kuf[BUFSIZ];

   if((kf = fopen(DIRFILE,"r")) == (FILE *)0) {
         syslog(LOG_NOTICE,"cannot open %s: %m",DIRFILE);
         return(-1);
   }

   while(fgets(kuf,sizeof(kuf),kf)) { /* read each line of file */
         if(kuf[0] == '#')
                 continue;
```

setup_directory

```
         if((hp = strtok(kuf," \t\n")) == (char *)0)
                 continue;
         /* compare name and system name for directory */
         if(!strcasecmp(dest,hp) || !strcasecmp(hp,"default")) {
                 if((kp = strtok((char *)0," \t\n")) == (char *)0) {
                          syslog(LOG_NOTICE,"missing directory spec for
   %s",hp);
                          goto puke;
                 }
                 if(chdir(kp) || chroot(kp)) { /* lock the dir */
                          syslog(LOG_NOTICE,"chroot %s: %m",kp);
                          goto puke;
                 }
                 /* if a uid# is specified setuid, too */
                 if((kp = strtok((char *)0," \t\n")) != (char *)0) {
                          int     jnku;
```

setup_directory

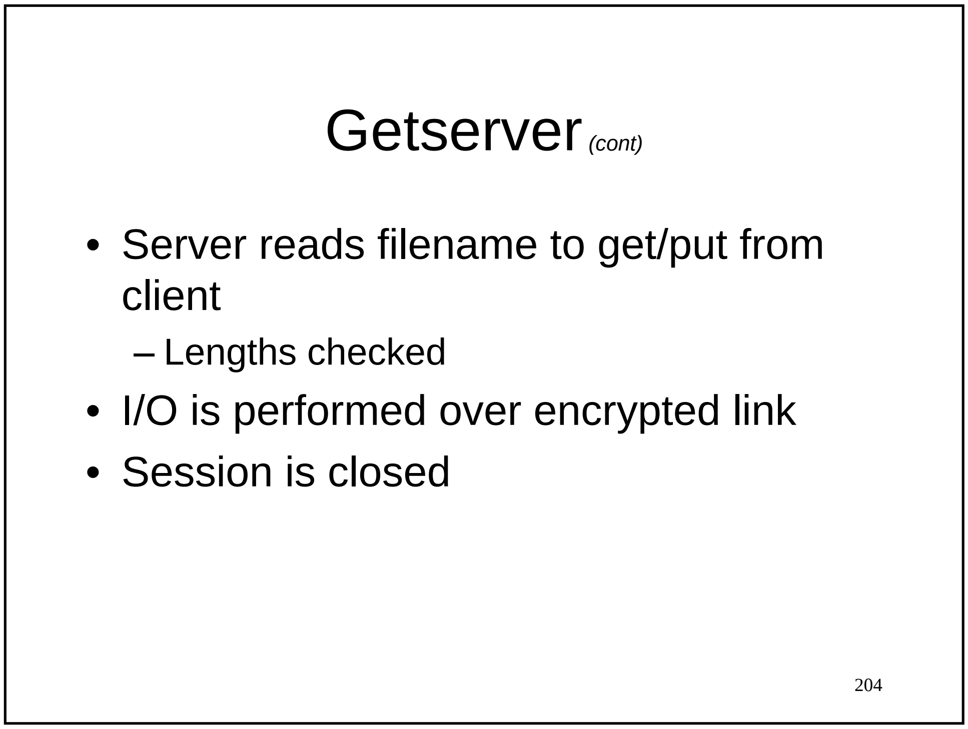
```
                          jnku = atoi(kp);
                          if(setuid(jnku)) {
                                  syslog(LOG_NOTICE,"setuid %d:
   %m",jnku);
                                  goto puke;
                          }
                 }
                 fclose(kf);
                 return(0);
         }
   }
   syslog(LOG_NOTICE,"no directory for %s",dest);
puke:
   fclose(kf);
   return(1);
}
```



Getserver *(cont)*

- If client authenticates server does a `chroot(2)` to the individual client's directory and sets user-id to a dummy user
  - Local file operations performed with dummy user file permissions
  - This is simple, comprehensible, and lets the O/S handle permissions itself instead of writing lots of code

203



Getserver *(cont)*

- Server reads filename to get/put from client
  - Lengths checked
- I/O is performed over encrypted link
- Session is closed

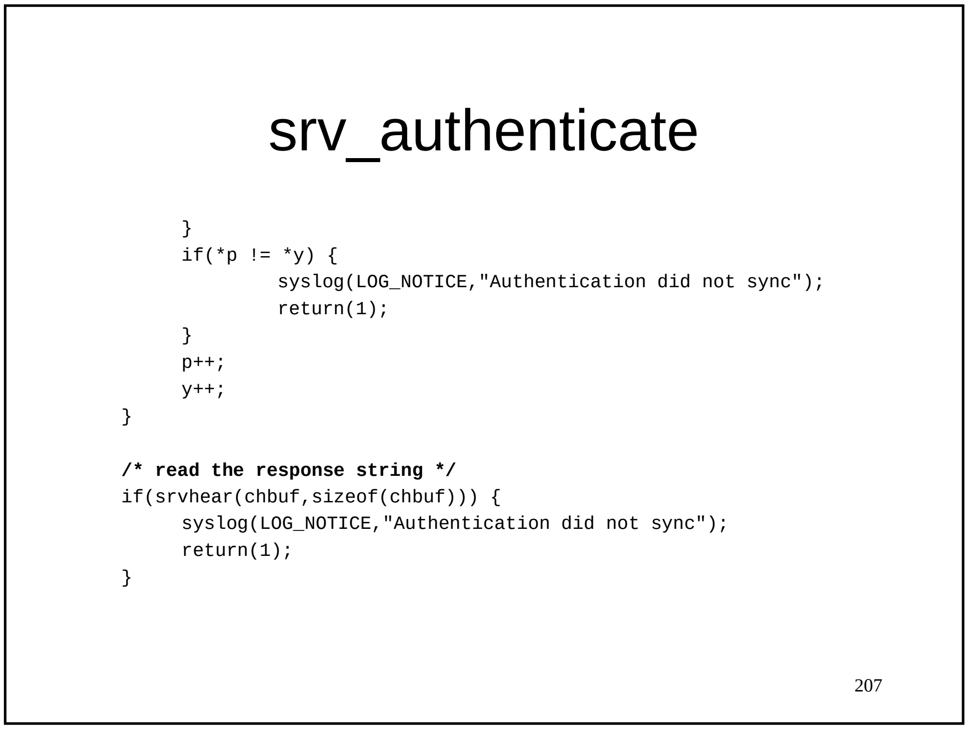
204

srv_authenticate

```
srv_authenticate()
{
   time_t       now;
   int   j4;
   int   j5;
   char  jnk[8];
   char  ojnk[8];
   char  chbuf[512];
   char  *y;
   char  *p;

   /* stuff some junk inna buffer -- should use /dev/random */
   time(&now);
   bcopy(&now,&jnk,sizeof(now));
   now = (time_t)getpid();
   if(sizeof(now) * 2 <= sizeof(jnk))
        bcopy(&now,&jnk[sizeof(now)],sizeof(now));
```

srv_authenticate

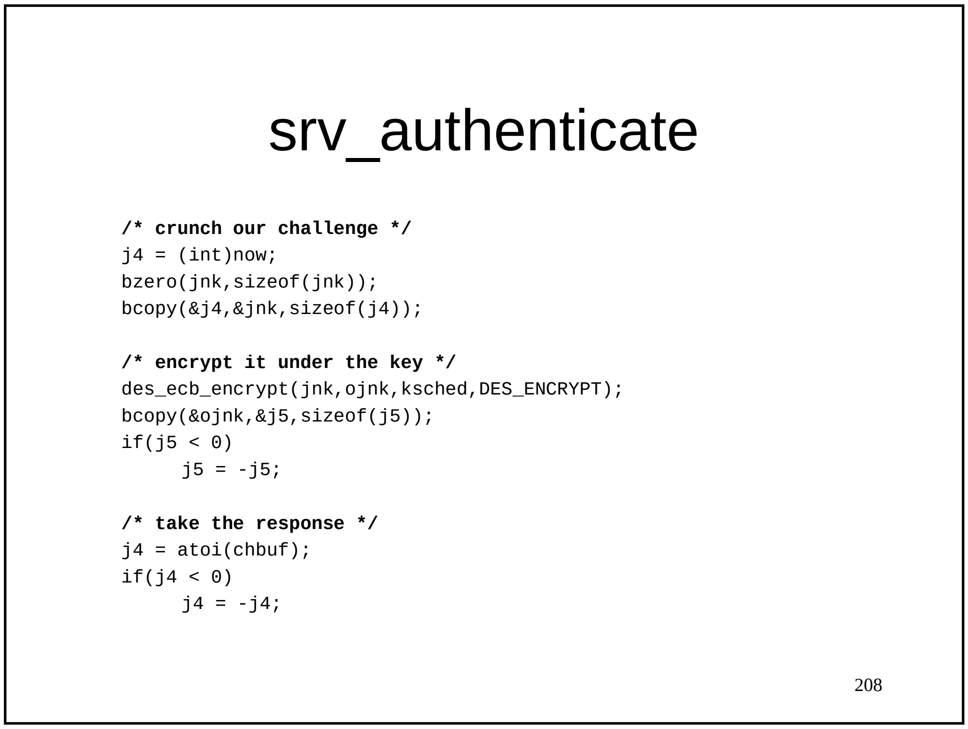
```
   /* crypt the buffer to make it look more random */
   des_ecb_encrypt(jnk,ojnk,ksched,DES_ENCRYPT);
   bcopy(&ojnk,&now,sizeof(now));
   if(now < 0)
        now = -now;

   /* send a challenge */
   sprintf(chbuf,"challenge %d",now);
   srvsay(chbuf);

   /* now read a response */
   p = "response ";
   y = chbuf;
   while(*p != '\0') {
        if(srvread(0,y,1) != 1) {
                syslog(LOG_NOTICE,"read");
                return(1);
```



srv_authenticate

```
     }
     if(*p != *y) {
              syslog(LOG_NOTICE,"Authentication did not sync");
              return(1);
     }
     p++;
     y++;
}

/* read the response string */
if(srvhear(chbuf,sizeof(chbuf))) {
     syslog(LOG_NOTICE,"Authentication did not sync");
     return(1);
}
```

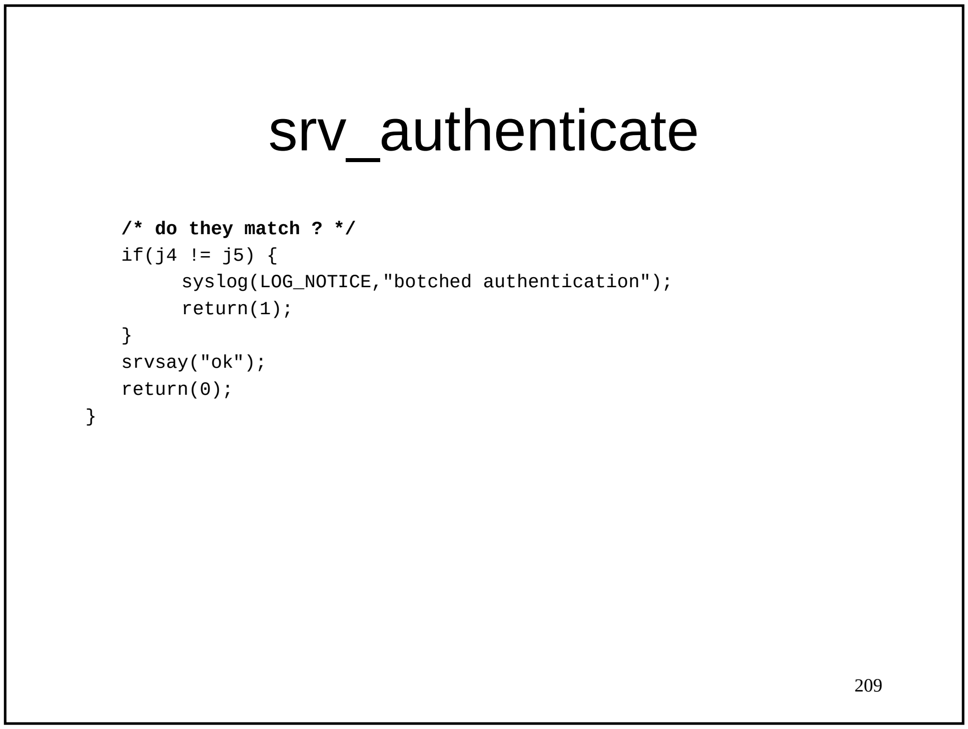
207



srv_authenticate

```
/* crunch our challenge */
j4 = (int)now;
bzero(jnk,sizeof(jnk));
bcopy(&j4,&jnk,sizeof(j4));

/* encrypt it under the key */
des_ecb_encrypt(jnk,ojnk,ksched,DES_ENCRYPT);
bcopy(&ojnk,&j5,sizeof(j5));
if(j5 < 0)
     j5 = -j5;

/* take the response */
j4 = atoi(chbuf);
if(j4 < 0)
     j4 = -j4;
```

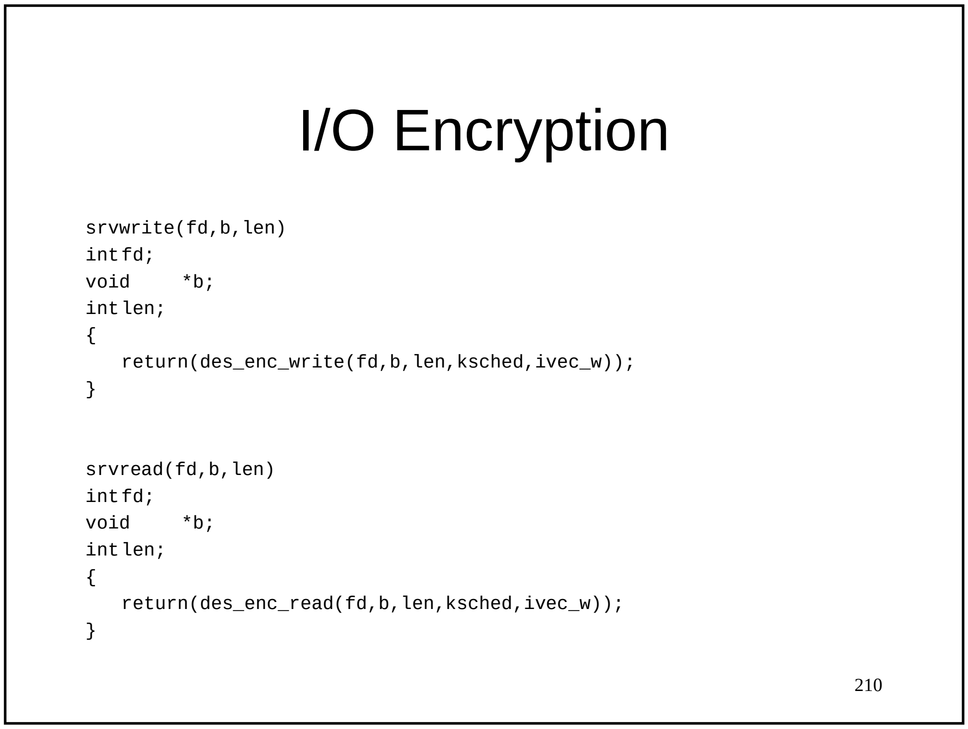
208



srv_authenticate

```
    /* do they match ? */
    if(j4 != j5) {
          syslog(LOG_NOTICE,"botched authentication");
          return(1);
    }
    srvsay("ok");
    return(0);
}
```

209



I/O Encryption

```
srvwrite(fd,b,len)
int fd;
void     *b;
int len;
{
    return(des_enc_write(fd,b,len,ksched,ivec_w));
}


srvread(fd,b,len)
int fd;
void     *b;
int len;
{
    return(des_enc_read(fd,b,len,ksched,ivec_w));
}
```

210

Keep it Simple

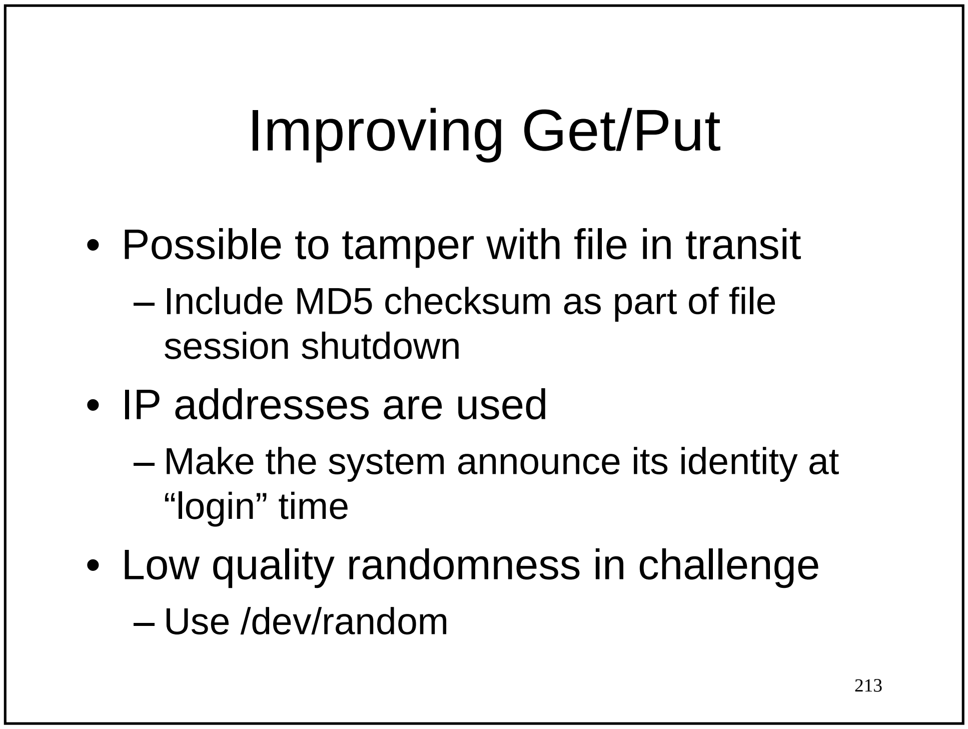
- get/getserv is not a paragon of beauty
  - It is a simple application designed to provide a secure service with minimal mechanism and a simple interface
  - The advantage of how it works is that a sysadmin could then build on top of its simple capability and not sweat the underlying details
  - Getserv in turn relies on O/S for basics

211

Keep it Balanced

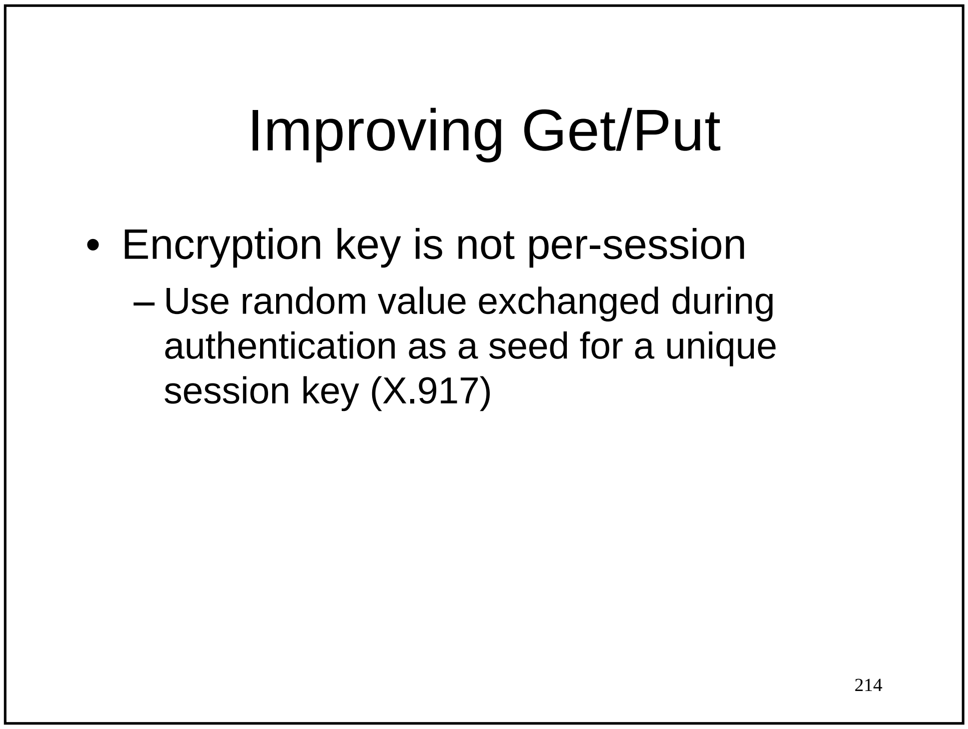
- get/getserv uses simple secret key encryption - Is it good enough?
  - Yes - after all, it is a file transfer program between two machines
  - If either is already compromised so an attacker could steal the keys then there's no need to attack get/getserv
  - Military grade crypto not necessary here

212



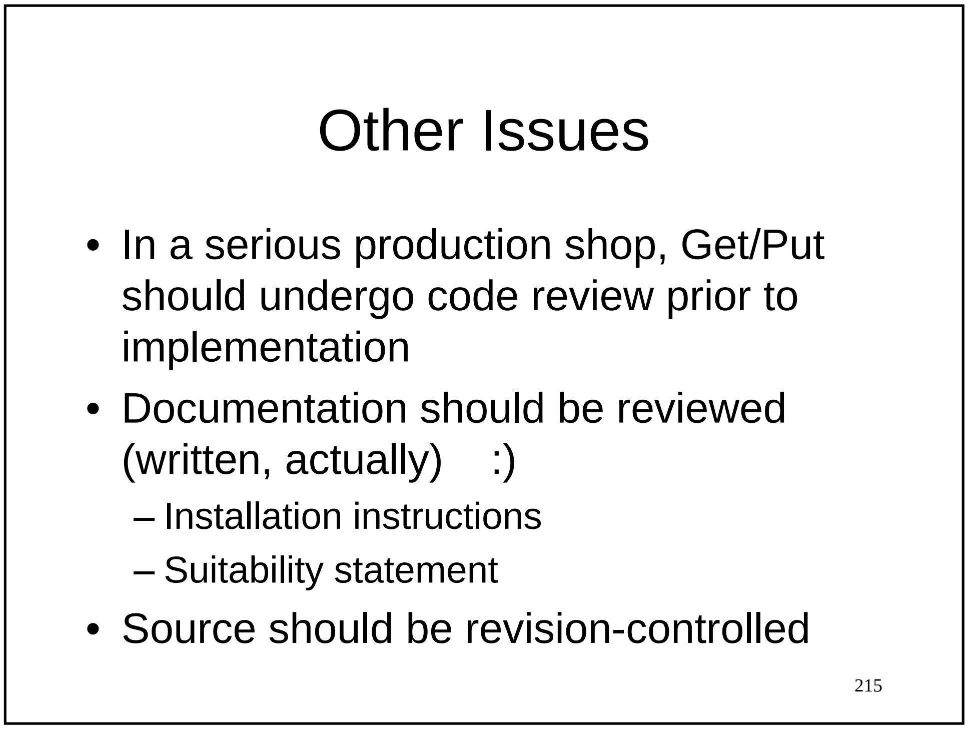
Improving Get/Put

- Possible to tamper with file in transit
  - Include MD5 checksum as part of file session shutdown
- IP addresses are used
  - Make the system announce its identity at “login” time
- Low quality randomness in challenge
  - Use /dev/random



Improving Get/Put

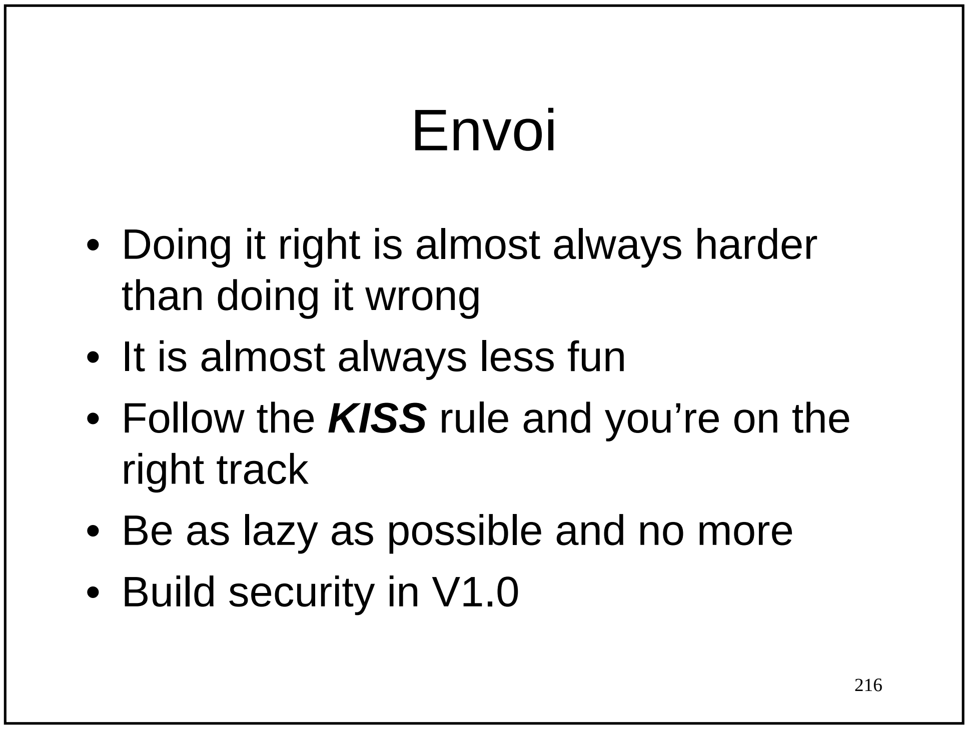
- Encryption key is not per-session
  - Use random value exchanged during authentication as a seed for a unique session key (X.917)



Other Issues

- In a serious production shop, Get/Put should undergo code review prior to implementation
- Documentation should be reviewed (written, actually) :)
  - Installation instructions
  - Suitability statement
- Source should be revision-controlled

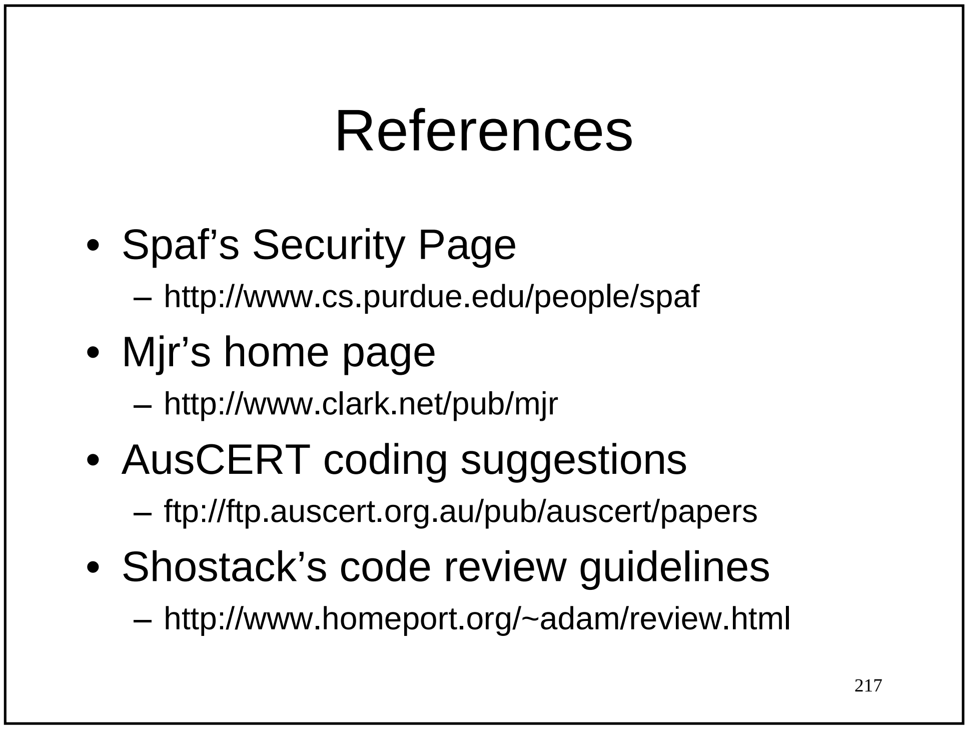
215



Envoi

- Doing it right is almost always harder than doing it wrong
- It is almost always less fun
- Follow the ***KISS*** rule and you’re on the right track
- Be as lazy as possible and no more
- Build security in V1.0

216



References

- Spaf’s Security Page
  - http://www.cs.purdue.edu/people/spaf
- Mjr’s home page
  - http://www.clark.net/pub/mjr
- AusCERT coding suggestions
  - ftp://ftp.auscert.org.au/pub/auscert/papers
- Shostack’s code review guidelines
  - http://www.homeport.org/~adam/review.html