Problems

- Flaw Hypothesis Methodology depends on caliber of testers to hypothesize and generalize flaws
- Flaw Hypothesis Methodology does not provide a way to examine system systematically
  – Vulnerability classification schemes help here

Vulnerability Classification

- Describe flaws from differing perspectives
  - Exploit-oriented
  - Hardware, software, interface-oriented
- Goals vary; common ones are:
  - Specify, design, implement computer system without vulnerabilities
  - Analyze computer system to detect vulnerabilities
  - Address any vulnerabilities introduced during system operation
  - Detect attempted exploitations of vulnerabilities

Example Flaws

- Use these to compare classification schemes
- First one: race condition (*xterm*)
- Second one: buffer overflow on stack leading to execution of injected code (*fingerd*)
- Both are very well known, and fixes available!
  - And should be installed everywhere …

Flaw #1: xterm

- *xterm* emulates terminal under X11 window system
  - Must run as *root* user on UNIX systems
    - No longer universally true; reason irrelevant here
- Log feature: user can log all input, output to file
  - User names file
  - If file does not exist, *xterm* creates it, makes owner the user
  - If file exists, *xterm* checks user can write to it, and if so opens file to append log to it

File Exists

- Check that user can write to file requires special system call
  - Because *root* can append to any file, check in *open* will always succeed

Check that user can write to file "/usr/tom/X"

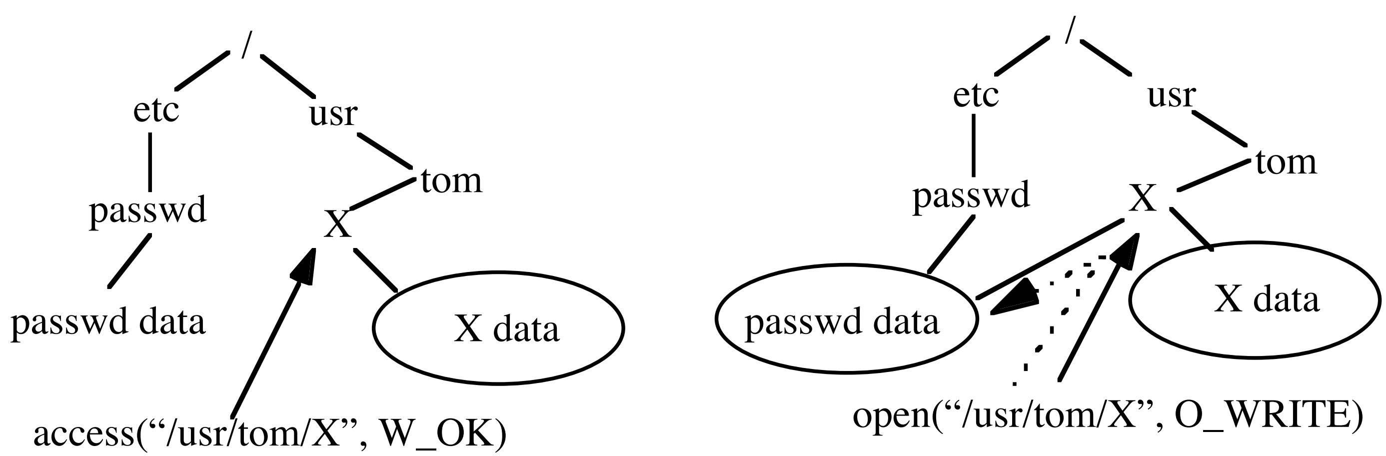
```
if (access("/usr/tom/X", W_OK) == 0){
```

Open "/usr/tom/X" to append log entries

```
   if ((fd = open("/usr/tom/X", O_WRONLY|O_APPEND))< 0){
             /* handle error: cannot open file */
   }
}
```

Problem

- Binding of file name "/usr/tom/X" to file object can change between first and second lines
  - (a) is at *access*; (b) is at *open*
  - Note file opened is *not* file checked



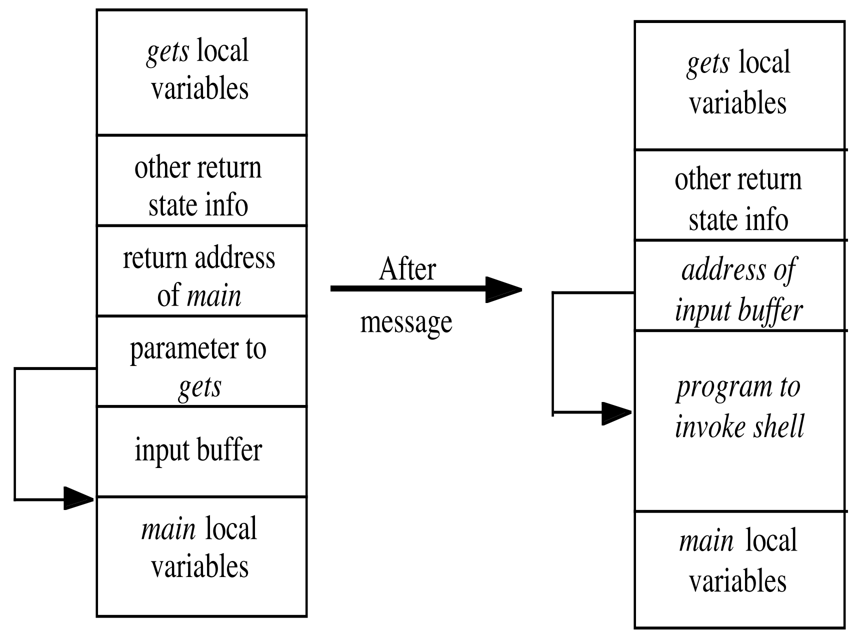
(a) (b)

Flaw #2: *fingerd*

- Exploited by Internet Worm of 1988
  - Recurs in many places, even now
- *finger* client send request for information to server *fingerd* (*finger* daemon)
  - Request is name of at most 512 chars
  - What happens if you send more?

Buffer Overflow

- Extra chars overwrite rest of stack, as shown
- Can make those chars change return address to point to beginning of buffer
- If buffer contains small program to spawn shell, attacker gets shell on target system



Frameworks

- Goals dictate structure of classification scheme
  - Guide development of attack tool $\Rightarrow$ focus is on steps needed to exploit vulnerability
  - Aid software development process $\Rightarrow$ focus is on design and programming errors causing vulnerabilities
- Following schemes classify vulnerability as n-tuple, each element of n-tuple being classes into which vulnerability falls
  - Some have 1 axis; others have multiple axes

Research Into Secure Operating Systems (RISOS)

- Goal: aid computer, system managers in understanding security issues in OSes, and help determine how much effort required to enhance system security
- Attempted to develop methodologies and software for detecting some problems, and techniques for avoiding and ameliorating other problems
- Examined Multics, TENEX, TOPS-10, GECOS, OS/MVT, SDS-940, EXEC-8

Classification Scheme

- Incomplete parameter validation
- Inconsistent parameter validation
- Implicit sharing of privileged/confidential data
- Asynchronous validation/inadequate serialization
- Inadequate identification/authentication/authorization
- Violable prohibition/limit
- Exploitable logic error

Incomplete Parameter Validation

- Parameter not checked before use
- Example: emulating integer division in kernel (RISC chip involved)
  - Caller provided addresses for quotient, remainder
  - Quotient address checked to be sure it was in user's protection domain
  - Remainder address *not* checked
    - Set remainder address to address of process' level of privilege
    - Compute 25/5 and you have level 0 (kernel) privileges
- Check for type, format, range of values, access rights, presence (or absence)

Inconsistent Parameter Validation

- Each routine checks parameter is in proper format for that routine but the routines require different formats
- Example: each database record 1 line, colons separating fields
  - One program accepts colons, newlines as pat of data within fields
  - Another program reads them as field and record separators
  - This allows bogus records to be entered

Implicit Sharing of Privileged / Confidential Data

- OS does not isolate users, processes properly
- Example: file password protection
  - OS allows user to determine when paging occurs
  - Files protected by passwords
    - Passwords checked char by char; stops at first incorrect char
  - Position guess for password so page fault occurred between 1st, 2nd char
    - If no page fault, 1st char was wrong; if page fault, it was right
  - Continue until password discovered

Asynchronous Validation / Inadequate Serialization

- Time of check to time of use flaws, intermixing reads and writes to create inconsistencies
- Example: *xterm* flaw discussed earlier

Inadequate Identification / Authorization / Authentication

- Erroneously identifying user, assuming another's privilege, or tricking someone into executing program without authorization
- Example: OS on which access to file named "SYS$*DLOC$" meant process privileged
  - Check: can process access any file with qualifier name beginning with "SYS" and file name beginning with "DLO"?
  - If your process can access file "SYSA*DLOC$", which is ordinary file, your process is privileged

Violable Prohibition / Limit

- Boundary conditions not handled properly
- Example: OS kept in low memory, user process in high memory
  - Boundary was highest address of OS
  - All memory accesses checked against this
  - Memory accesses not checked beyond end of high memory
    - Such addresses reduced modulo memory size
  - So, process could access (memory size)+1, or word 1, which is part of OS …

Exploitable Logic Error

- Problems not falling into other classes
  - Incorrect error handling, unexpected side effects, incorrect resource allocation, etc.
- Example: unchecked return from monitor
  - Monitor adds 1 to address in user's PC, returns
    - Index bit (indicating indirection) is a bit in word
    - Attack: set address to be –1; adding 1 overflows, changes index bit, so return is to location stored in register 1
  - Arrange for this to point to bootstrap program stored in other registers
    - On return, program executes with system privileges

Legacy of RISOS

- First funded project examining vulnerabilities
- Valuable insight into nature of flaws
  - Security is a function of site requirements and threats
  - Small number of fundamental flaws recurring in many contexts
  - OS security not critical factor in design of OSes
- Spurred additional research efforts into detection, repair of vulnerabilities

Program Analysis (PA)

- Goal: develop techniques to find vulnerabilities
- Tried to break problem into smaller, more manageable pieces
- Developed general strategy, applied it to several OSes
  - Found previously unknown vulnerabilities

Classification Scheme

- Improper protection domain initialization and enforcement
  - Improper choice of initial protection domain
  - Improper isolation of implementation detail
  - Improper change
  - Improper naming
  - Improper deallocation or deletion
- Improper validation
- Improper synchronization
  - Improper indivisibility
  - Improper sequencing
- Improper choice of operand or operation

Improper Choice of Initial Protection Domain

- Initial incorrect assignment of privileges, security and integrity classes
- Example: on boot, protection mode of file containing identifiers of all users can be altered by any user
  - Under most policies, should not be allowed

Improper Isolation of Implementation Detail

- Mapping an abstraction into an implementation in such a way that the abstraction can be bypassed
- Example: virtual machines modulate length of time CPU is used by each to send bits to each other
- Example: Having raw disk accessible to system as ordinary file, enabling users to bypass file system abstraction and write directly to raw disk blocks

Improper Change

- Data is inconsistent over a period of time
- Example: *xterm* flaw
  - Meaning of "/usr/tom/X" changes between *access* and *open*
- Example: parameter is validated, then accessed; but parameter is changed between validation and access
  - Burroughs B6700 allowed allowed this

Improper Naming

- Multiple objects with same name
- Example: Trojan horse
  - *loadmodule* attack discussed earlier; “bin” could be a directory or a program
- Example: multiple hosts with same IP address
  - Messages may be erroneously routed

Improper Deallocation or Deletion

- Failing to clear memory or disk blocks (or other storage) after it is freed for use by others
- Example: program that contains passwords that a user typed dumps core
  – Passwords plainly visible in core dump

Improper Validation

- Inadequate checking of bounds, type, or other attributes or values
- Example: *fingerd*'s failure to check input length

Improper Indivisibility

- Interrupting operations that should be uninterruptable
  - Often: "interrupting atomic operations"
- Example: *mkdir* flaw (UNIX Version 7)
  - Created directories by executing privileged operation to create file node of type directory, then changed ownership to user
  - On loaded system, could change binding of name of directory to be that of password file after directory created but before change of ownership
  - Attacker can change administrator's password

Improper Sequencing

- Required order of operations not enforced
- Example: one-time password scheme
  - System runs multiple copies of its server
  - Two users try to access same account
    - Server 1 reads password from file
    - Server 2 reads password from file
    - Both validate typed password, allow user to log in
    - Server 1 writes new password to file
    - Server 2 writes new password to file
  - Should have every read to file followed by a write, and vice versa; not two reads or two writes to file in a row

Improper Choice of Operand or Operation

- Calling inappropriate or erroneous instructions
- Example: cryptographic key generation software calling pseudorandom number generators that produce predictable sequences of numbers

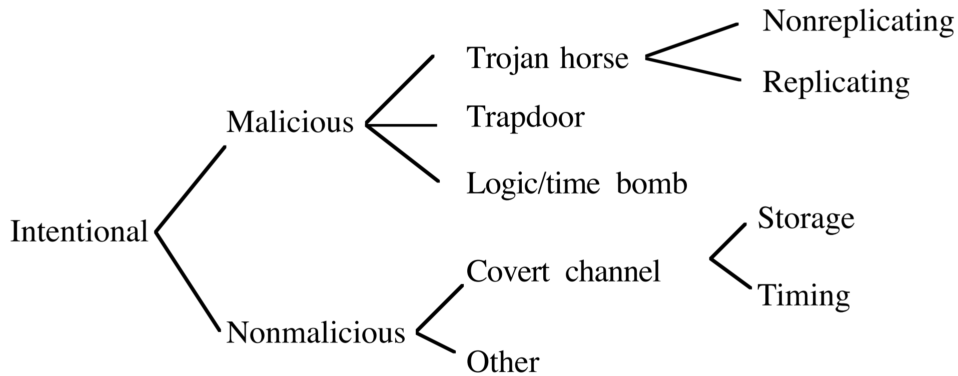
Legacy

- First to explore automatic detection of security flaws in programs and systems
- Methods developed but not widely used
  - Parts of procedure could not be automated
  - Complexity
  - Procedures for obtaining system-independent patterns describing flaws not complete

NRL Taxonomy

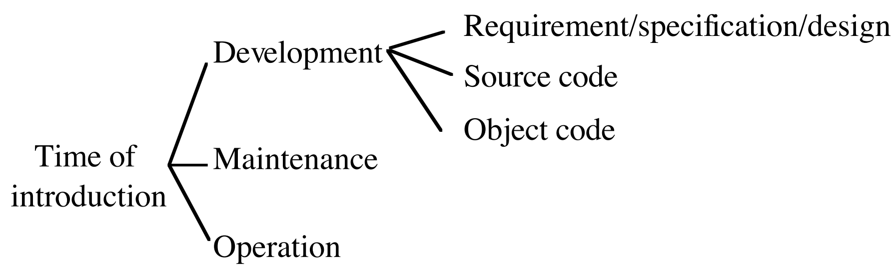
- Goals:
  - Determine how flaws entered system
  - Determine when flaws entered system
  - Determine where flaws are manifested in system
- 3 different schemes used:
  - Genesis of flaws
  - Time of flaws
  - Location of flaws

Genesis of Flaws



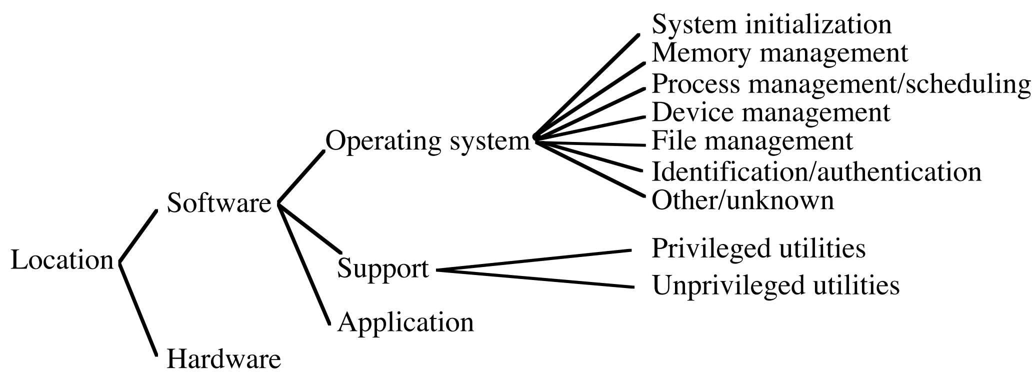
- Inadvertent (unintentional) flaws classified using RISOS categories; not shown above
  - If most inadvertent, better design/coding reviews needed
  - If most intentional, need to hire more trustworthy developers and do more security-related testing

Time of Flaws



- Development phase: all activities up to release of initial version of software
- Maintenance phase: all activities leading to changes in software performed under configuration control
- Operation phase: all activities involving patching and not under configuration control

Location of Flaw



- Focus effort on locations where most flaws occur, or where most serious flaws occur

Legacy

- Analyzed 50 flaws
- Concluded that, with a large enough sample size, an analyst could study relationships between pairs of classes
  - This would help developers focus on most likely places, times, and causes of flaws
- Focused on social processes as well as technical details
  - But much information required for classification not available for the 50 flaws

Aslam's Model

- Goal: treat vulnerabilities as faults and develop scheme based on fault trees
- Focuses specifically on UNIX flaws
- Classifications unique and unambiguous
  - Organized as a binary tree, with a question at each node. Answer determines branch you take
  - Leaf node gives you classification
- Suited for organizing flaws in a database

Top Level

- Coding faults: introduced during software development
  - Example: *fingerd*'s failure to check length of input string before storing it in buffer
- Emergent faults: result from incorrect initialization, use, or application
  - Example: allowing message transfer agent to forward mail to arbitrary file on system (it performs according to specification, but results create a vulnerability)

Coding Faults

- Synchronization errors: improper serialization of operations, timing window between two operations creates flaw
  - Example: *xterm* flaw
- Condition validation errors: bounds not checked, access rights ignored, input not validated, authentication and identification fails
  - Example: *fingerd* flaw

Emergent Faults

- Configuration errors: program installed incorrectly
  - Example: *tftp* daemon installed so it can access any file; then anyone can copy any file
- Environmental faults: faults introduced by environment
  - Example: on some UNIX systems, any shell with "-" as first char of name is interactive, so find a setuid shell script, create a link to name "-gotcha", run it, and you has a privileged interactive shell

Legacy

- Tied security flaws to software faults
- Introduced a precise classification scheme
  - Each vulnerability belongs to exactly 1 class of security flaws
  - Decision procedure well-defined, unambiguous

Comparison and Analysis

- Point of view
  - If multiple processes involved in exploiting the flaw, how does that affect classification?
    - *xterm*, *fingerd* flaws depend on interaction of two processes (*xterm* and process to switch file objects; *fingerd* and its client)
- Levels of abstraction
  - How does flaw appear at different levels?
    - Levels are abstract, design, implementation, etc.

xterm and PA Classification

- Implementation level
  - *xterm*: improper change
  - attacker's program: improper deallocation or deletion
  - operating system: improper indivisibility

xterm and PA Classification

- Consider higher level of abstraction, where directory is simply an object
  - create, delete files maps to writing; read file status, open file maps to reading
  - operating system: improper sequencing
    - During read, a write occurs, violating Bernstein conditions
- Consider even higher level of abstraction
  - attacker's process: improper choice of initial protection domain
    - Should not be able to write to directory containing log file
    - Semantics of UNIX users require this at lower levels

xterm and RISOS Classification

- Implementation level
  - *xterm*: asynchronous validation/inadequate serialization
  - attacker's process: exploitable logic error and violable prohibition/limit
  - operating system: inconsistent parameter validation

xterm and RISOS Classification

- Consider higher level of abstraction, where directory is simply an object (as before)
  - all: asynchronous validation/inadequate serialization
- Consider even higher level of abstraction
  - attacker’s process: inadequate identification/authentication/authorization
    - Directory with log file not protected adequately
    - Semantics of UNIX require this at lower levels

xterm and NRL Classification

- Time, location unambiguous
  - Time: during development
  - Location: Support:privileged utilities
- Genesis: ambiguous
  - If intentional:
    - Lowest level: inadvertent flaw of serialization/aliasing
  - If unintentional:
    - Lowest level: nonmalicious: other
  - At higher levels, parallels that of RISOS

xterm and Aslam's Classification

- Implementation level
  - attacker's process: object installed with incorrect permissions
    - attacker's process can delete file
  - *xterm*: access rights validation error
    - *xterm* doesn't properly validate file at time of access
  - operating system: improper or inadequate serialization error
    - deletion, creation should not have been interspersed with access, open
  - Note: in absence of explicit decision procedure, all could go into class race condition

The Point

- The schemes lead to ambiguity
  - Different researchers may classify the same vulnerability differently for the same classification scheme
- Not true for Aslam's, but that misses connections between different classifications
  - *xterm* is race condition as well as others; Aslam does not show this

fingerd and PA Classification

- Implementation level
  - *fingerd*: improper validation
  - attacker's process: improper choice of operand or operation
  - operating system: improper isolation of implementation detail

fingerd and PA Classification

- Consider higher level of abstraction, where storage space of return address is object
  - operating system: improper change
  - *fingerd*: improper validation
    - Because it doesn't validate the type of instructions to be executed, mistaking data for valid ones
- Consider even higher level of abstraction, where security-related value in memory is changing and data executed that should not be executable
  - operating system: improper choice of initial protection domain

fingerd and RISOS Classification

- Implementation level
  - *fingerd*: incomplete parameter validation
  - attacker's process: violable prohibition/limit
  - operating system: inadequate identification/authentication/authorization

fingerd and RISOS Classification

- Consider higher level of abstraction, where storage space of return address is object
  - operating system: asynchronous validation/inadequate serialization
  - *fingerd*: inadequate identification/authentication/authorization
- Consider even higher level of abstraction, where security-related value in memory is changing and data executed that should not be executable
  - operating system: inadequate identification/authentication/authorization

fingerd and NRL Classification

- Time, location unambiguous
  - Time: during development
  - Location: support: privileged utilities
- Genesis: ambiguous
  - Known to be inadvertent flaw
  - Parallels that of RISOS

fingerd and Aslam Classification

- Implementation level
  - *fingerd*: boundary condition error
  - attacker's process: boundary condition error
    - operating system: environmental fault
      - If decision procedure not present, could also have been access rights validation errors

Common Vulnerabilities

- Unknown interaction with other system components
- Overflow
- Race conditions
- Environment variables
- Not resetting privileges

Unknown Interactions

- DNS with bad information
- Assumption about servers running on ports

Poisoned DNS

- DNS: returns a host name given an IP address
- Attacker "poisons the DNS"
  - Say that address 169.237.4.199 corresponds to host "a.com null; cp /bin/sh /etc/telnetd;"
- Attacker connect to a system running server that notifies *root* of any connections or connection attempts
- It runs a command like this:

```
echo Login | mail –s nob null; cp /bin/sh /etc/telnetd
```

Wrong Server Listening

- Connect to port 79 to obtain information
  - *fingerd* takes name, name and host, and sends back information about that user on named host (or local, if no host named)
- Server is *chargen* (supposed to be on port 19), not *fingerd*
  - *finger* client prints whatever it is sent

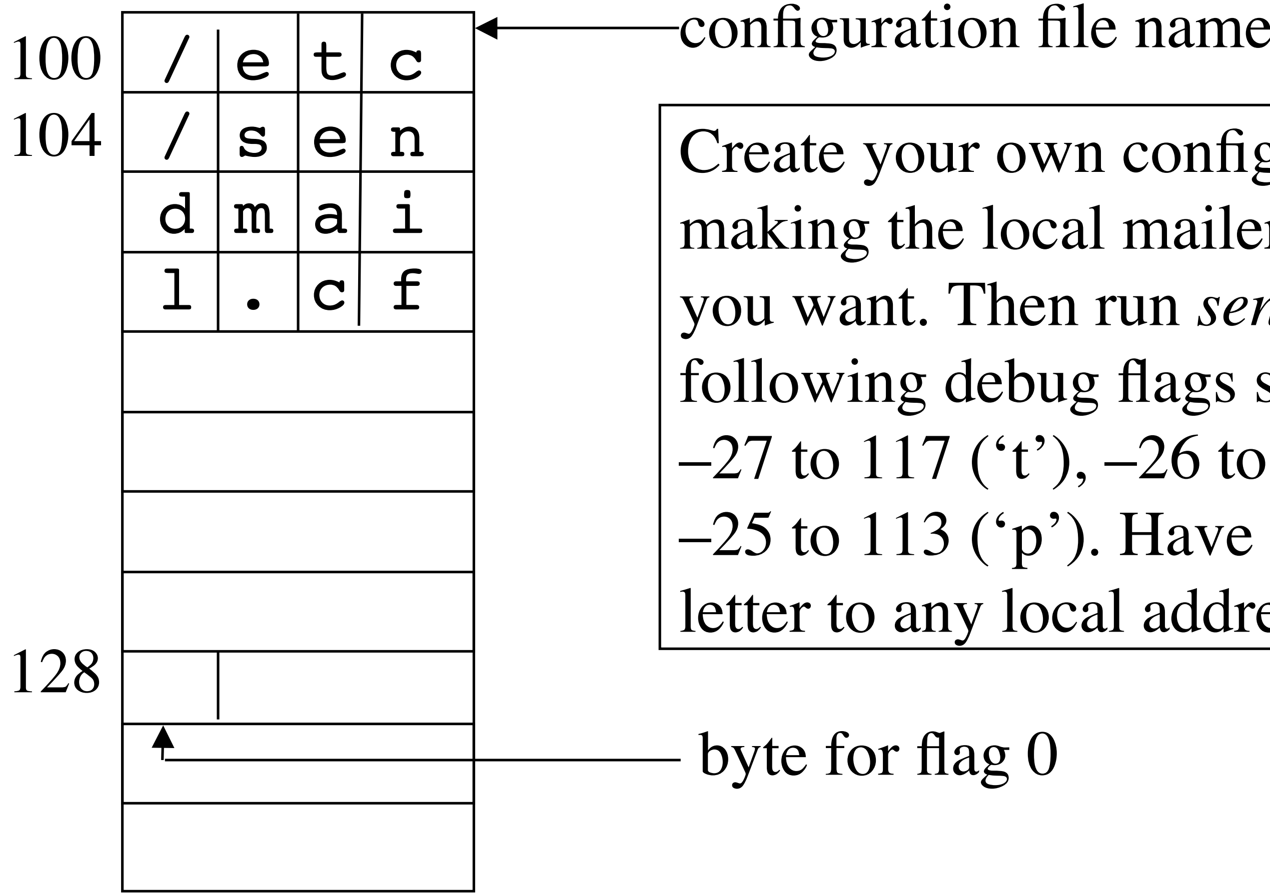
Overflows

- Buffer overflows
- Integer overflows

Example: sendmail config file

- *sendmail* takes debugging flags of form *flag*.*value*
  - `sendmail -d7,102` sets debugging flag 7 to value 102
- Flags stored in array in data segment
- So is name of default configuration file
  - It's called *sendmail.cf*
- Contains name of local delivery agent
  - `Mlocal` line; usually /bin/mail …

In Pictures



Create your own config file, making the local mailer be whatever you want. Then run *sendmail* with the following debug flags settings: flag –27 to 117 (‘t’), –26 to 110 (‘m’), and –25 to 113 (‘p’). Have it deliver a letter to any local address …

Problems and Solutions

- Sendmail won't recognize negative flag numbers
- So make them unsigned (positive)!
  - –27 becomes 2^{32} – 27 = 4294967269
  - –26 becomes 2^{32} – 26 = 4294967270
  - –25 becomes 2^{32} – 26 = 4294967271
- Command is:

```
sendmail -d4294967269,117 -d4294967270,110 -
  d4294967271,113 …
```

Race Conditions

- TOCTTOU flaws like *xterm*
- Races in signaling, interprocess communication

Race Signals

- FTP clients aborting:
  - ABOR on control connection with urgent flag set
  - Closing data connection
- FTP server getting two signals and catching both:
  - SIGURG for the ABOR
  - SIGPIPE for the close
- FTP server has real UID as *root* so it can honor USER
  - Once authenticated, effective UID drops to user

FTP Race Condition

- SIGPIPE causes server to get effective UID *root*, write entry to the *wtmp* file, exit
  – No signal handling changed here
- SIGURG sends FTP server back to command loop
  – Window is if SIGURG arrives after SIGPIPE but before *exit*()
  – If SIGURG occurs at that point, FTP server re-enters FTP command loop and is running with effective UID *root*

Environment Variables

- *vi* file
  - Edit file, then hang up without saving it …
  - *vi* invokes *expreserve*
    - *expreserve* saves buffer in protected area not accessible to ordinary users, including editor of the file
- *expreserve* invokes *mail* to send letter to user

Where Is the Privilege?

- *vi* is not setuid to *root*
  - You don't need that to edit your files
- *expreserve* is setuid to *root*
  - the buffer is saved in a protected area so *expreserve* needs enough privileges to copy the file there
- *mail* is run by *expreserve*
  - so unless reset, it runs with *root* privileges

The First Attack

- Do this:

```
$ cat > ./mail
#! /bin/sh
cp /bin/sh /usr/attack/.sh
chmod 4755 /usr/attack/.sh
^D
$ PATH=.:$PATH
$ export PATH
```

… and then run *vi* and hang up.

So vi Fixed it …

- Instead of resetting PATH, change

```
popen("mail user", "w")
```

to

```
popen("/bin/mail user", "w")
```

But … still uses Bourne shell … so

The Second Attack

- Bourne shell determines white space with **IFS**
- Use same program as before, but call it *b* and set it up this way:

```
% IFS="/inmal\t\n "; export IFS
% PATH=.:$PATH; export PATH
```

- Then run *vi* and hang up.
  - Then "/bin/mail" user acts like *b* and runs my program

Fixing This

- Fix given in most books is:

```
system("IFS='\n\t ';PATH=/bin:/usr/bin;\
                              export IFS
  PATH;command");
```

- This sets **IFS**, **PATH**, you may want to fix more

WRONG

How to Break This

- Before invoking your program *plugh*, I do:

```
% IFS="I$IFS"
% PATH=".:$PATH"
% plugh
```

- Now your **IFS** is unchanged since the Bourne shell interprets the **I** in `IFS="I$IFS"` as a blank, and reads the first part as `FS="\n\t "`

Privilege Problems

- At a university, games very popular, owned as *root*
  - Needed to update high score files
- Graduate students discovered that effective UID was not reset when a subshell spawned
  - So they could start a game which kept a high score file, and run a subshell – as *root*!