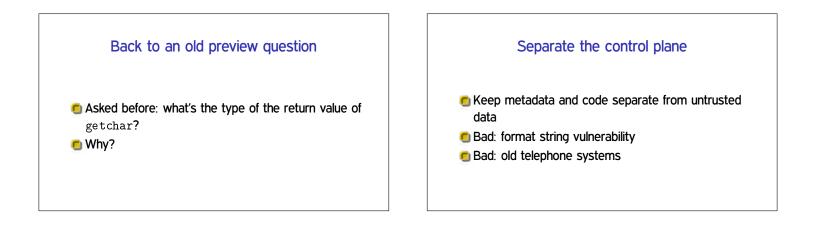
CSci 5271 Introduction to Computer Security Defensive programming and design, cont'd

Stephen McCamant University of Minnesota, Computer Science & Engineering

Outline

More secure design principles Software engineering for security Secure use of the OS Announcements intermission Bernstein's perspective Techniques for privilege separation



Defense in depth

Multiple levels of protection can be better than one Especially if none is perfect

But, many weak security mechanisms don't add up

Canonicalize names

Use unique representations of objects
 E.g. in paths, remove ., ..., extra slashes, symlinks
 E.g., use IP address instead of DNS name

Fail-safe / fail-stop

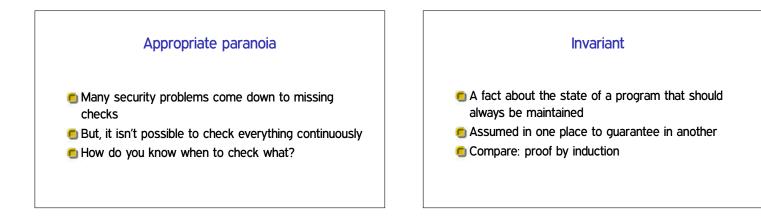
- If something goes wrong, behave in a way that's safe
 Often better to stop execution than continue in
- corrupted state
- E.g., better segfault than code injection

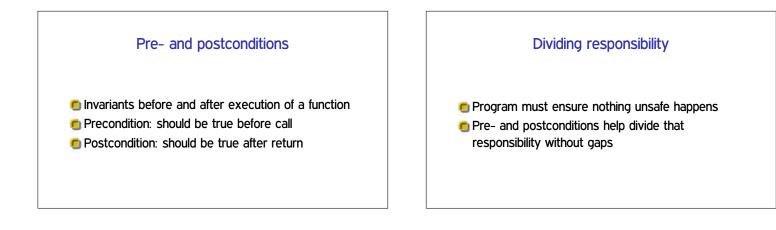
Outline

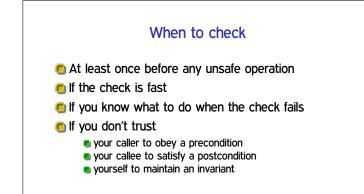
More secure design principles Software engineering for security Secure use of the OS

- Announcements intermission
- Bernstein's perspective
- Techniques for privilege separation









Sometimes you can't check

- ${\color{black} {\scriptsize \hbox{\scriptsize one}}}$ Check that p points to a null-terminated string
- Check that fp is a valid function pointer
- 0 Check that $\mathbf x$ was not chosen by an attacker

Error handling

Every error must be handled

 I.e, program must take an appropriate response action

 Errors can indicate bugs, precondition violations, or situations in the environment

Error codes

- Commonly, return value indicates error if any Bad: may overlap with regular result
- Bad: goes away if ignored

Exceptions

Separate from data, triggers jump to handler
 Good: avoid need for manual copying, not dropped
 May support: automatic cleanup (finally)
 Bad: non-local control flow can be surprising

Testing and security

- "Testing shows the presence, not the absence of bugs" – Dijkstra
- Easy versions of some bugs can be found by targeted tests:
 - Buffer overflows: long strings
 - Integer overflows: large numbers
 - Format string vulnerabilities: %x

Fuzz testing

Random testing can also sometimes reveal bugs
 Original 'fuzz' (Miller): program </dev/urandom
 Even this was surprisingly effective

Modern fuzz testing

- Mutation fuzzing: small random changes to a benign seed input

 Complex benign inputs help cover interesting functionality

 Grammar-based fuzzing: randomly select valid inputs
- Coverage-driven fuzzing: build off of tests that cause new parts of the program to execute
 - Automatically learns what inputs are "interesting"
 - Pioneered in the open-source AFL tool

Outline

More secure design principles Software engineering for security Secure use of the OS Announcements intermission

Announcements intermission

Bernstein's perspective

Techniques for privilege separation

Avoid special privileges

- Require users to have appropriate permissions
 Rather than putting trust in programs
- Anti-pattern 1: setuid/setgid program
- 🖲 Anti-pattern 2: privileged daemon
- But, sometimes unavoidable (e.g., email)

One slide on setuid/setgid

- Unix users and process have a user id number (UID) as well as one or more group IDs
- Normally, process has the IDs of the use who starts it
- A setuid program instead takes the UID of the program binary

Don't use shells or Tcl

- … in security-sensitive applications
- String interpretation and re-parsing are very hard to do safely
- Eternal Unix code bug: path names with spaces

Prefer file descriptors Maintain references to files by keeping them open and using file descriptors, rather than by name References same contents despite file system changes Use openat, etc., variants to use FD instead of directory paths Prefer absolute paths Use full paths (starting with /) for programs and files \$PATH under local user control But FD-like, so can be used in place of openat if missing

Prefer fully trusted paths

- Each directory component in a path must be write protected
- Read-only file in read-only directory can be changed if a parent directory is modified

Don't separate check from use

- Avoid pattern of e.g., access then open
 Instead, just handle failure of open

 You have to do this anyway

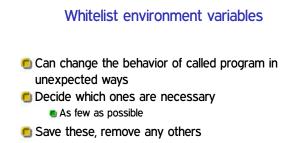
 Multiple references allow races
 - And access also has a history of bugs

Be careful with temporary files

- Create files exclusively with tight permissions and never reopen them
 - See detailed recommendations in Wheeler
- Not quite good enough: reopen and check matching device and inode
 - Fails with sufficiently patient attack

Give up privileges

- Using appropriate combinations of set*id functions
 Alas, details differ between Unix variants
- 🖲 Best: give up permanently
- Second best: give up temporarily
- Detailed recommendations: Setuid Demystified (USENIX'02)



Outline

More secure design principles Software engineering for security Secure use of the OS Announcements intermission Bernstein's perspective Techniques for privilege separation

Last preview question

What is the return type of getchar()?

- A. signed char
- B. int
- C. unsigned char
- D. char
- E. float

BCMTA vulnerability found!

- A user can process incoming messages with a command using .forward
- BCMTA should drop privileges when running this command
- Because of a logic error, did so only the first time

BCECHO part 2, continued

Modifying a system file
 \O-free shellcoding

Shellcode in an environment variable

Outline

More secure design principles Software engineering for security Secure use of the OS Announcements intermission Bernstein's perspective Techniques for privilege separation

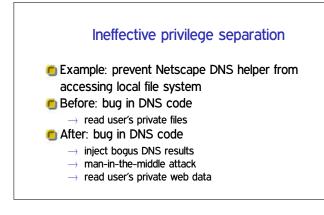
Historical background

Traditional Unix MTA: Sendmail (BSD)

- Monolithic setuid root program
- Designed for a more trusting era
- In mid-90s, bugs seemed endless
- Spurred development of new, security-oriented replacements
 - . Bernstein's qmail
 - Venema et al.'s Postfix

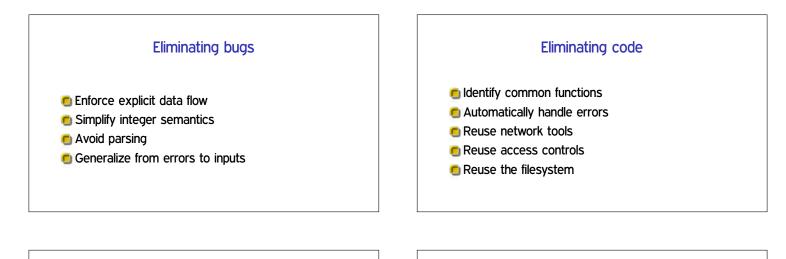
Distinctive qmail features

- Single, security-oriented developer
- Architecture with separate programs and UIDs
- Replacements for standard libraries
- 🖲 Deliveries into directories rather than large files



Effective privilege separation

- Transformations with constrained I/O
- General argument: worst adversary can do is control output
 - Which is just the benign functionality
- MTA header parsing (Sendmail bug)
- 🧐 jpegtopnm inside xloadimage

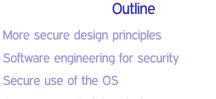


The "qmail security guarantee"

- \$500, later \$1000 offered for security bug
 Never paid out
 Issues proposed:
 - Memory exhaustion DoS
 - Overflow of signed integer indexes
 - Defensiveness does not encourage more submissions

qmail today

- Originally had terms that prohibited modified redistribution
 - Now true public domain
- 🖲 Latest release from Bernstein: 1998; netqmail: 2007
- 🖲 Does not have large market share
- 🖲 All MTAs, even Sendmail, are more secure now



Announcements intermission

Bernstein's perspective

Techniques for privilege separation



- Main application: code provided by untrusted parties
- Packet filters in the kernel
- 🗐 JavaScript in web browsers
 - Also Java, Flash ActionScript, etc.

SFI

- Software-based Fault Isolation
 Instruction-level rewriting like (but predates) CFI
 Limit memory stores and sometimes loads
 Can't jump out except to designated points
- 🖲 E.g., Google Native Client

Separate processes

- OS (and hardware) isolate one process from another
- Pay overhead for creation and communication
- System call interface allows many possibilities for mischief

System-call interposition

- Trusted process examines syscalls made by untrusted
- Implement via ptrace (like strace, gdb) or via kernel change
- 🖲 Easy policy: deny

Interposition challenges

- Argument values can change in memory (TOCTTOU)
- OS objects can change (TOCTTOU)
- How to get canonical object identifiers?
- Interposer must accurately model kernel behavior
- Details: Garfinkel (NDSS'03)

Separate users

- Reuse OS facilities for access control
- Unit of trust: program or application
- 🖲 Older example: qmail
- Newer example: Android
- Limitation: lots of things available to any user

chroot

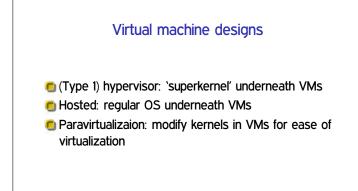
- Unix system call to change root directory
- Restrict/virtualize file system access
- Only available to root
- Does not isolate other namespaces

OS-enabled containers

- 互 One kernel, but virtualizes all namespaces
- 🖲 FreeBSD jails, Linux LXC, Solaris zones, etc.
- Quite robust, but the full, fixed, kernel is in the TCB

(System) virtual machines

- Presents hardware-like interface to an untrusted kernel
- Strong isolation, full administrative complexity
- I/O interface looks like a network, etc.



Virtual machine technologies

- Hardware based: fastest, now common
- Partial translation: e.g., original VMware
- Full emulation: e.g. QEMU proper
 Slowest, but can be a different CPU architecture

Modern example: Chrom(ium) Separates "browser kernel" from less-trusted "rendering engine"

- Pragmatic, keeps high-risk components together
- Experimented with various Windows and Linux sandboxing techniques
- Blocked 70% of historic vulnerabilities, not all new ones
- http://seclab.stanford.edu/websec/chromium/



Protection and isolation
Basic (e.g., classic Unix) access control