Security and Software Engineering

Steven M. Bellovin

AT&T Labs – Research

http://www.research.att.com/~smb

Click to add title

□ *“If our software is buggy, what does*

□ *that say about its security?”*

□

□ --Robert H. Morris

Some Principles of Software Engineering

- Simplicity is a virtue.
- If code is complex, you don’t know if it’s correct (but it probably isn’t).
- Break up complex systems into simple, well-defined modules.

Security is Hard

- “Reasonable” assumptions don’t apply.
  - File name length bounds don’t apply.
  - Any input field can be arbitrarily weird.
- Your adversary is creating improbabilities.
  - Race conditions *will* happen.
- “Nature is subtle but not malicious” – but the hackers are both.

Case Study: *rcp* and *rdist*

- *rcp* and *rdist* use the *rsh* protocol.
- The *rsh* protocol requires that the client program be on a privileged port.
- Thus, *rcp* and *rdist* run as *root*.
- Both have a long history of security holes…

Solutions

- □Don’t implement the protocol directly in *rcp* and *rdist*; invoke the *rsh* command.
- □Or invoke a small, trusted program that sets up the connection and passes back an open file descriptor.
- □Best of all, use a *real* authentication mechanism.

Using an Outboard Program

□Separates functions

□Improves modularity

□Improves security.

□*Maybe* a small loss in efficiency -- but note the difference between "efficiency" and *efficiency*: why do the wrong thing quickly?

Case Study: Kerberized *telnet*

- The DES library wanted 56-bit keys plus proper parity, in a 64-bit number.
- The “generate a 64-bit random key” code used by *telnet* didn’t set the parity bits properly.
- When handed a bad key, the DES library treated the key as all zeroes.
- With probability 255/256, the session was encrypted with a known, constant key!

Analysis

- Interfaces matter.
- Interfaces should be consistent – why did the encryption routine and the key generation routine behave differently?
  - If there was no key generation routine, there should have been.
- Error-checking matters.

Case Study: Many C Programs

- □About half of all newly-reported security holes are due to buffer overflows in C.
- □This shouldn’t be possible!
- □Tony Hoare warned us of this in his Turing Award lecture:

Hoare’s Turing Award Lecture:

“The first principle was *security*… A consequence of this principle is that every occurrence of every subscript of every subscripted variable was on every occasion checked at run time… I note with fear and horror that even in 1980, language designers and users have not learned this lesson.”

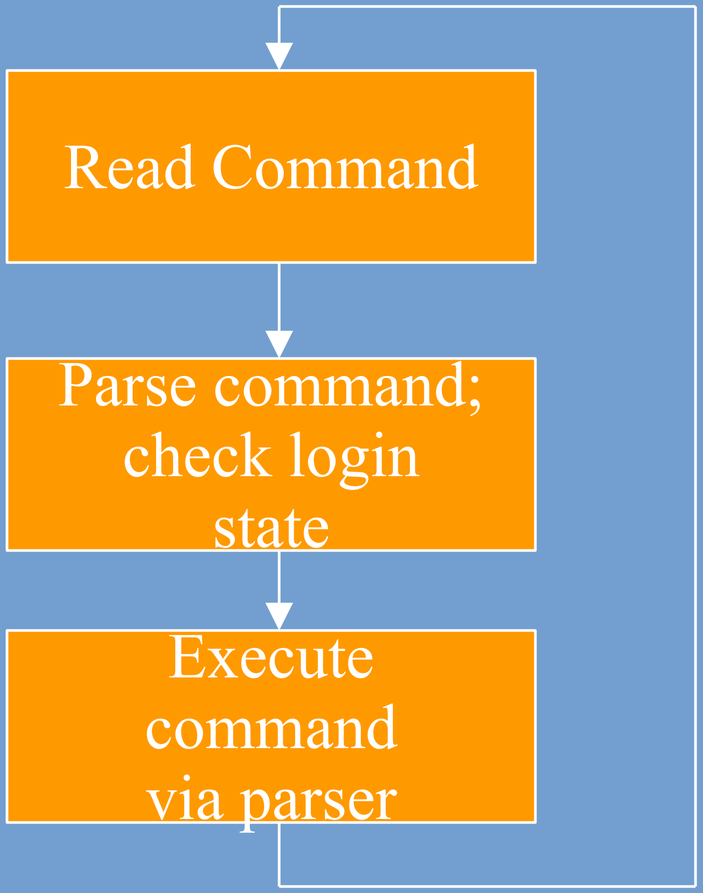
How to Fix Buffer Overflows

- Write better C.
  - Admittedly, that's hard, even with the best intent and the best programmers.
- Use C++ with the **string** class.
- Use Java.
- Use Stackguard.
- Use the bounds-checking C compiler.

Case Study: *ftpd*

- Original Berkeley implementation (and many of its descendants) used *yacc* to parse network input.
- *USER* and *PASS* were separate commands.
- Result: flag-setting, ubiquitous flag-testing, global state – and at least three different security holes.
  - Newer *ftpd*'s have more complex access control mechanisms – and more security holes.

Main Loop of *ftpd*



Read Command

Parse command; check login state

Execute command via parser

Login Sequence

- USER command
    - clear login state
    - Get */etc/passwd* entry
    - Check for *anonymous*; set flag if so.
- PASS command
    - If not anonymous, check password;
        - If failure, clear state and exit PASS
    - Set directory and uid from *passwd* entry
    - If anonymous, use *chroot()*
    - Set logged-in flag

Solution

- Separate the login code from the rest.
  - Put it in a separate, small program: ~100 lines.
- Activate your strong security measures (*chroot*, *setuid*) in the login module.
- The remaining thousands of lines of code can run unprivileged.
  - (Let the OS do access control – it's good at it.)

Cryptography is Even Harder

- □The oldest (public) cryptographic protocol was published in 1978.
- □A flaw was found in 1983.
- □The original authors found a flaw in the revised protocol in 1994.
- □A new error in the original was found in 1996.
- □Note: the protocol was only 5 lines long!

Sample Protocol Failure

A->S: A,B

S->A: CA, CB

A->B: CA, CB, $\{\{K_{ab}, T_a\}_{K_a^{-1}}\}_{K_b}$

We can replay a modified message 3:

B -> C: CA, CC, $\{\{K_{ab}, T_a\}_{K_a^{-1}}\}_{K_c}$

Other Rules for Cryptography

- Don’t invent your own cryptographic protocols.
- Don’t invent your own ciphers.
- And look askance at any product that has done either…

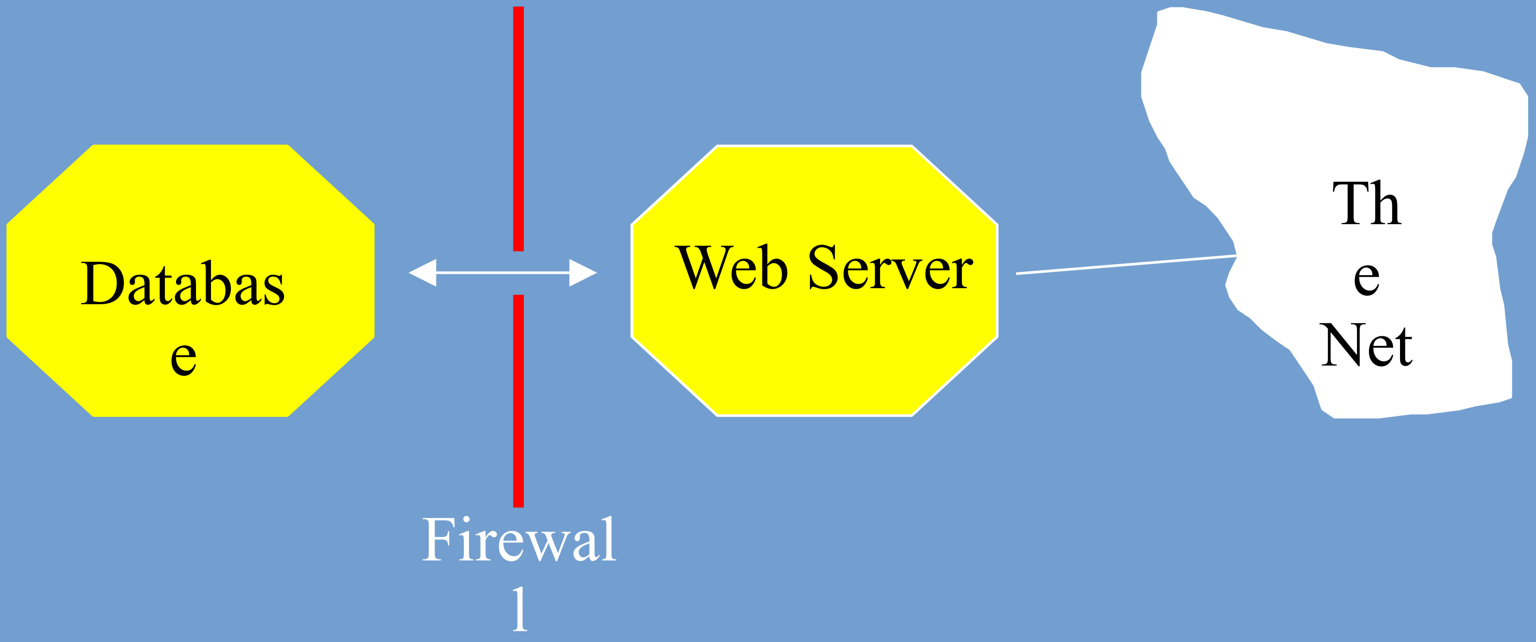
Bug Fixes

- Most system penetrations caused by known vulnerabilities, for which patches already exist.
- But blindly patching production systems is dangerous.
- There's a new scheme afoot to have vendors automatically install patches...

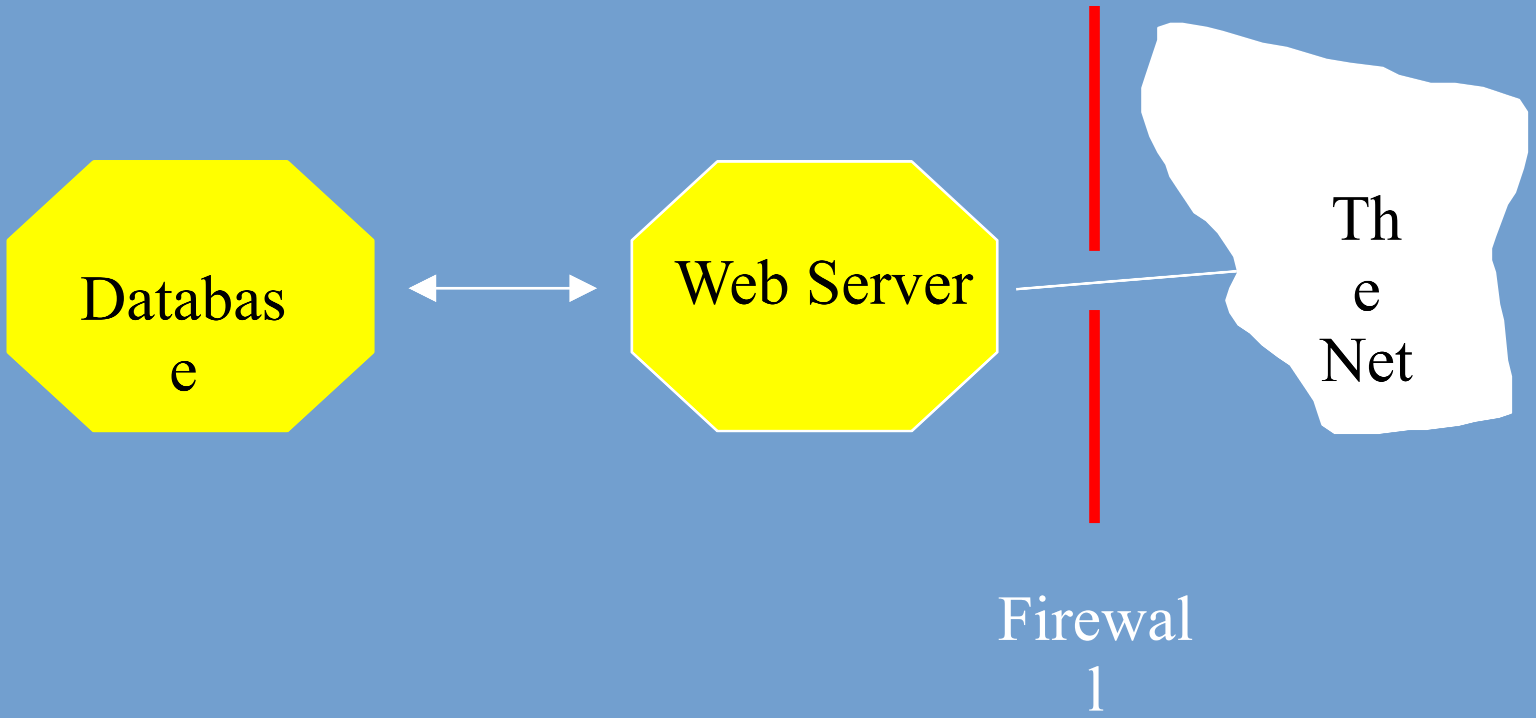
Today’s Challenges

- Large-scale, heterogeneous distributed systems.
  - Must design for component “failure”.
- Limited security tools (firewalls, hardened hosts, cryptography).
- Ubiquitous networking.
- Mobile code or near-code.

Firewalls and Databases



The Wrong Choice



Firewalls

- Firewalls are touted as a solution to the network security problem.
- Nonsense – they're the network's response to the *host* security problem.
- The real function of a firewall is to keep bad guys away from complex, buggy code.
- Today's firewalls are getting very complex…

Where to From Here?

- Sound software engineering matters more than ever.
- Shipping code on “Internet time” has exacerbated the problem.
  - But the economy seems to have solved it…
- We need to add a new dimension to our modular decomposition: security.