Exploiting Software: How to Break Code

Gary McGraw, Ph.D.
CTO, Cigital

http://www.cigital.com

Pop quiz

- What do wireless devices, cell phones, PDAs, browsers, operating systems, servers, personal computers, public key infrastructure systems, and firewalls have in common?

Software

So what’s the problem?

Commercial security is reactive

- Defend the perimeter with a firewall
  - To keep stuff out
- Over-rely on crypto
  - “We use SSL”
- “Review” products when they’re done
  - Why your code is bad
- Promulgate “penetrate and patch”
- Disallow advanced technologies
  - Extensible systems (Java and .NET) are dangerous

The “ops guy with keys” does not really understand software development.

Builders versus operators

- Most security people are operations people
  - Network administrators
  - Firewall rules manipulators
  - COTS products glommers
  - These people need training

Security means different things to different people

- Most builders are not security people
  - Software development remains a black art
  - How well are we doing teaching students to engineer code?
  - Emergent properties like security are hard for builders to grok
  - These people need academic education

Making software behave is hard

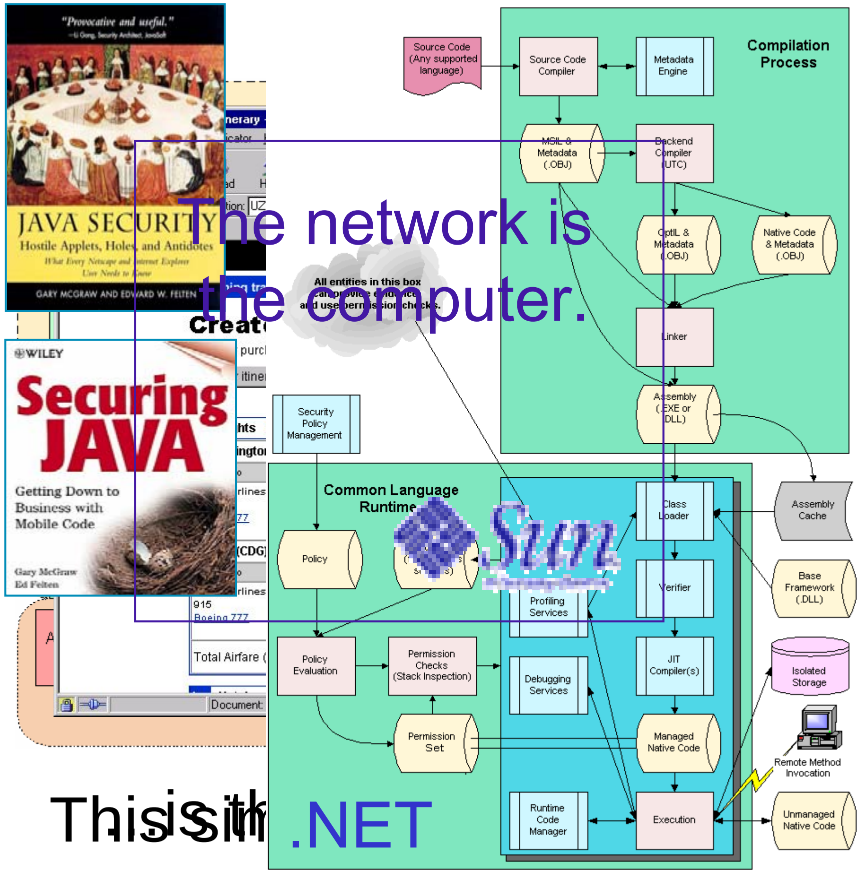
- Can you test in quality?
- How do you find (adaptive) bugs in code?
- What about bad guys doing evil on purpose?

- What’s the difference between security testing and functional testing?
- How can you teach security design?
- How can you codify non-functional, emergent requirements like security?
- Can you measure security?

Attaining software security is even harder

The Trinity of Trouble

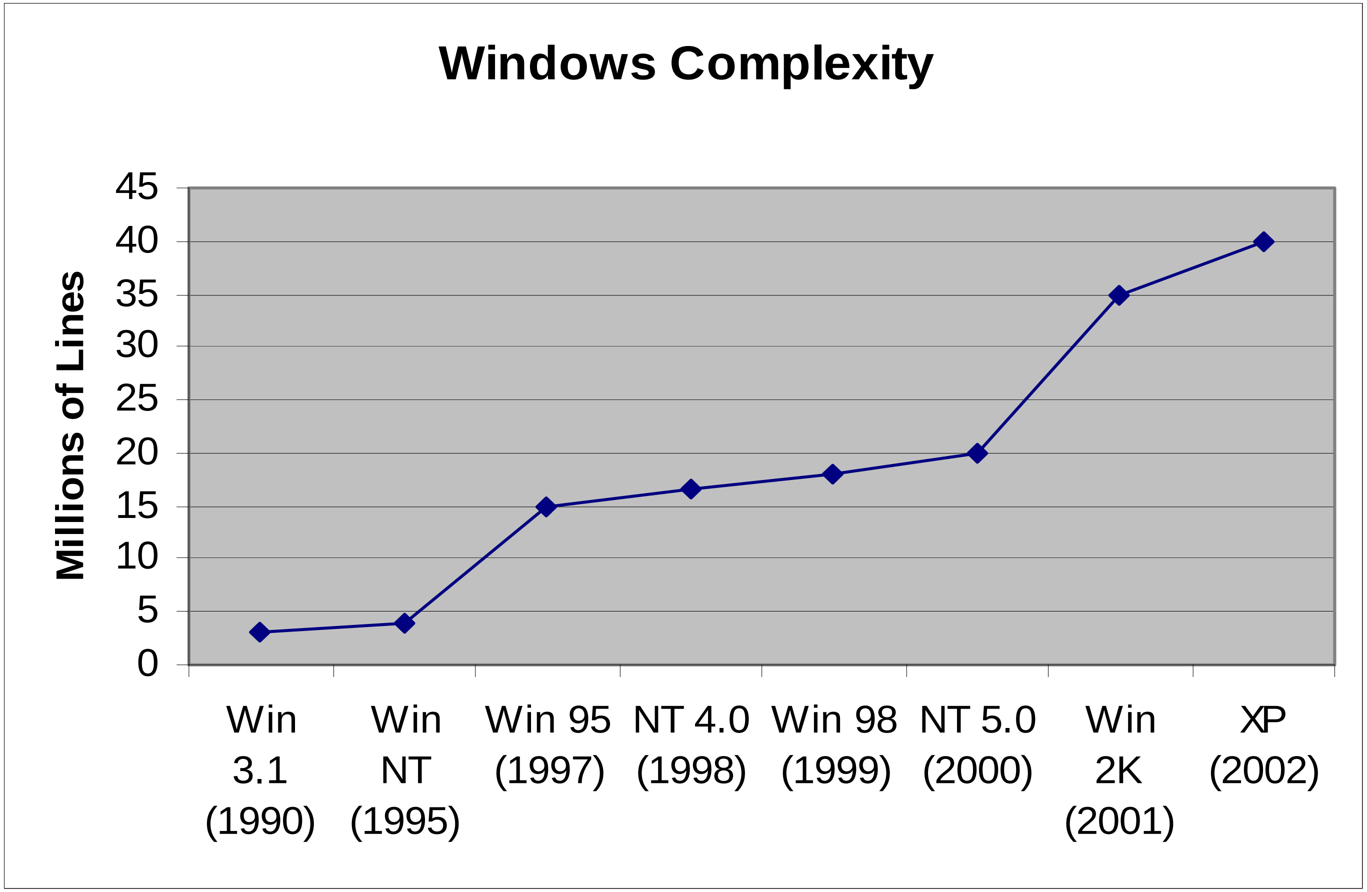
- **Connectivity**
  - The Internet is everywhere and most software is on it
- **Complexity**
  - Networked, distributed, mobile code is hard
- **Extensibility**
  - Systems evolve in unexpected ways and are changed on the fly



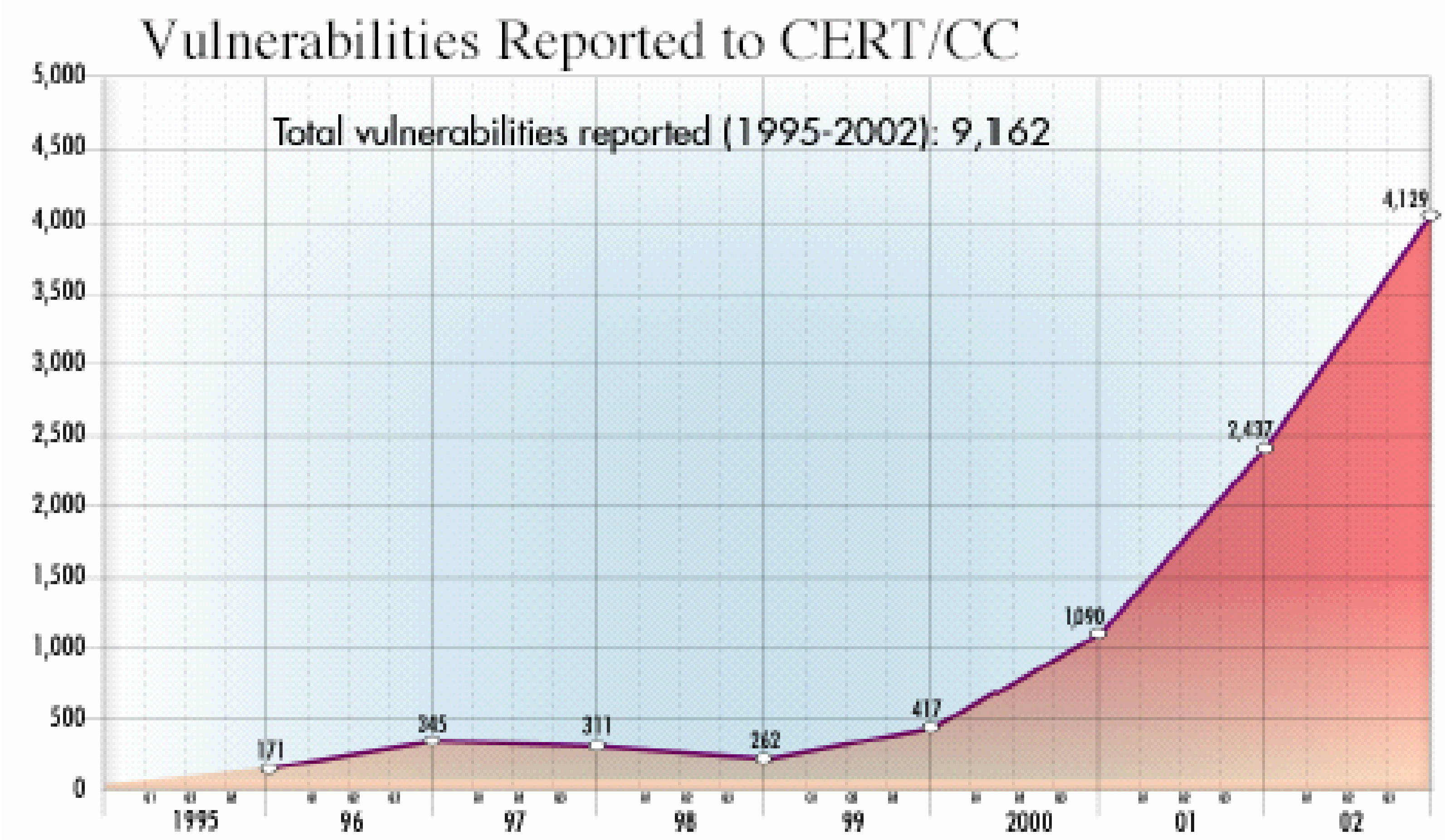
The network is the computer.

This is th .NET

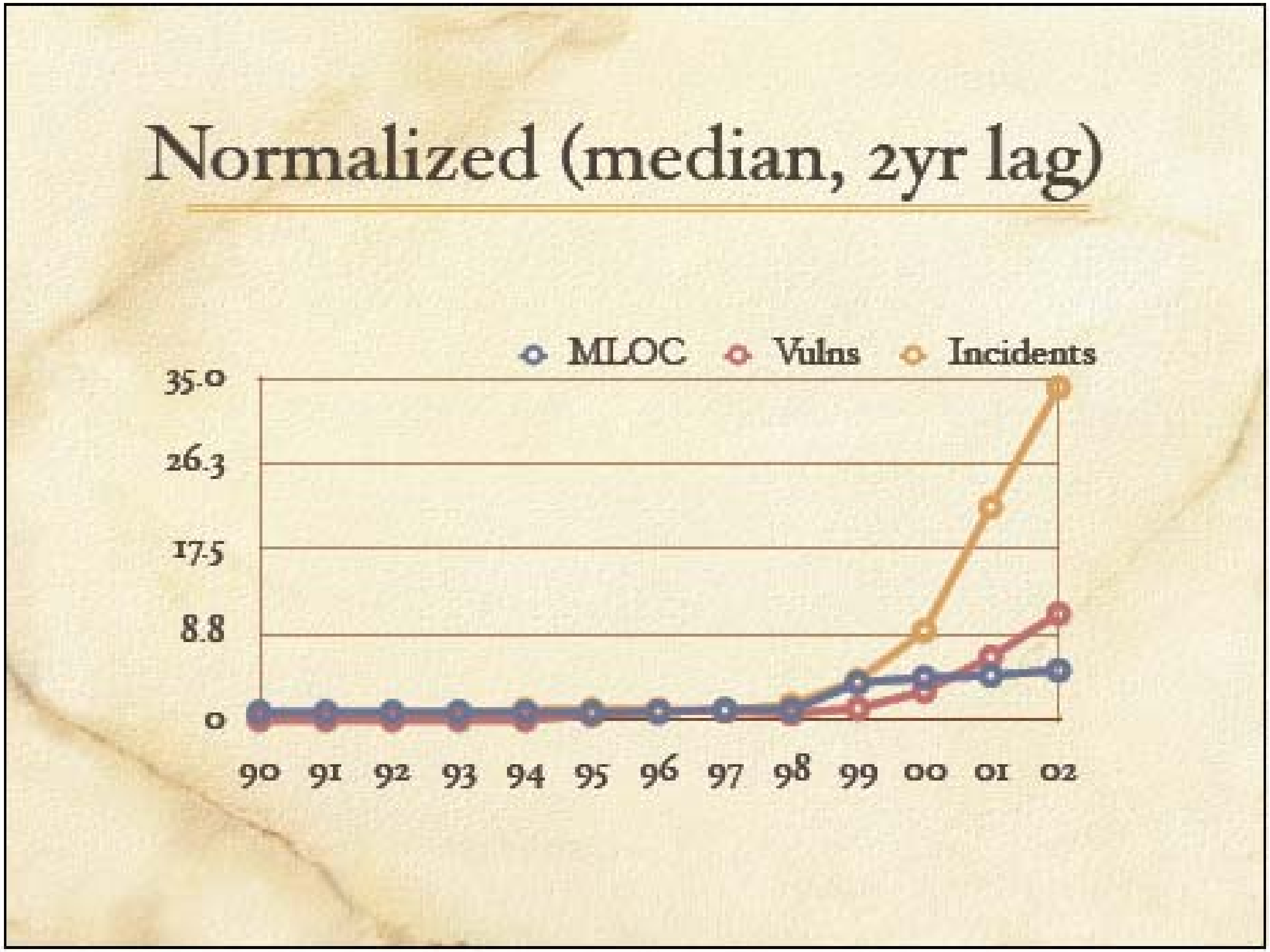
Software complexity growth



Software vulnerability growth



Normalized (and slightly shifted) data from Geer



Science please

- Basic understanding of complexity and its impact on security problems is sorely needed
- Do the LOC and vulnerability graphs really correlate?
- What are software security problems really like?
  - How common are basic categories?
  - How can we teach students something that now takes years of fieldwork to merely intuitively grasp?

Who is the bad guy?

- Hackers
  - “Full disclosure” zealots
- “Script kiddies”
- Criminals
  - Lone guns or organized
- Malicious insiders
  - Compiler wielders
- Business competition
- Police, press, terrorists, intelligence agencies

Above: Deputy Attorney General Christopher Bubb, center, N.J. Gov. Christie Whitman (left) and Attorney General Peter Verniero conferencing on alleged virus writer David L. Smith (right).

AP, ZDTV

History is quirky

1995

- Dan Geer fired from Silicon Graphics for releasing SATAN with Wietse Venema
- FUD: possible attack tool!

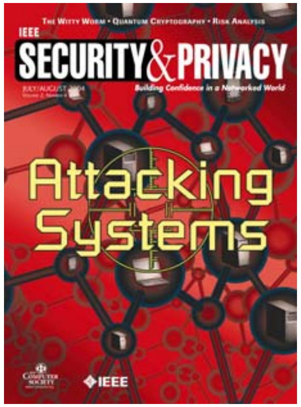
2004

- Any system administrator not using a port scanner to check security posture runs the risk of being fired

Fall 2004

- John Aycock at University of Calgary publicly criticized for malware course
- FUD: possible bad guy factory

Should we talk about attacking systems?



The good news and the bad news

Good news

- The world loves to talk about how stuff breaks
- This kind of work sparks lots of interest in computer security

Bad news

- The world would rather not focus on how to build stuff that does not break
- It’s harder to build good stuff than to break junky stuff

Know your enemy: How stuff breaks

Security problems are complicated

IMPLEMENTATION BUGS

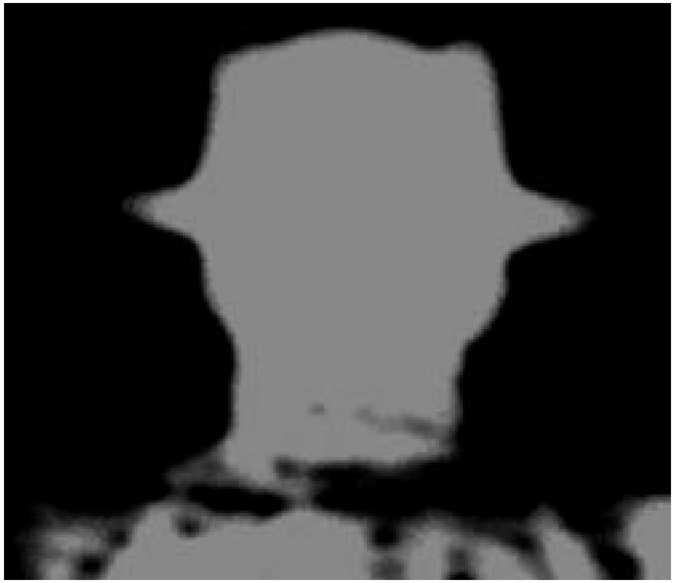
- Buffer overflow
  - String format
  - One-stage attacks
- Race conditions
  - TOCTOU (time of check to time of use)
- Unsafe environment variables
- Unsafe system calls
  - System()
- Untrusted input problems

ARCHITECTURAL FLAWS

- Misuse of cryptography
- Compartmentalization problems in design
- Privileged block protection failure (DoPrivilege())
- Catastrophic security failure (fragility)
- Type safety confusion error
- Insecure auditing
- Broken or illogical access control (RBAC over tiers)
- Method over-riding problems (subclass issues)
- Signing too much code

Attackers do not distinguish bugs and flaws

- Both bugs and flaws lead to vulnerabilities that can be exploited
- Attackers write code to break code
- Defenders are network operations people
  - Code?! What code?



The attacker’s toolkit

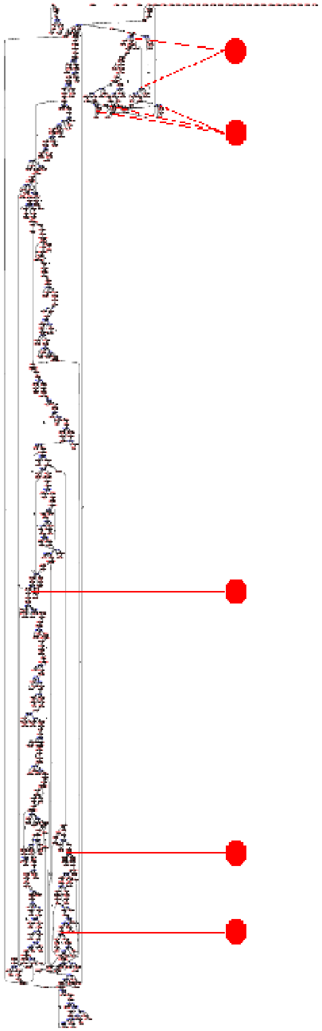
- The standard attacker’s toolkit has lots of (software analysis) stuff
  - Disassemblers and decompilers
  - Control flow and coverage tools
  - APISPY32
  - Breakpoint setters and monitors
  - Buffer overflow
  - Shell code
  - Rootkits

Attacker’s toolkit: dissasemblers and decompilers

- Source code is not a necessity for software exploit
- Binary is just as easy to understand as source code
- Disassemblers and decompilers are essential tools
- Reverse engineering is common and must be understood (not outlawed)
- IDA allows plugins to be created
- Use bulk auditing

Attacker’s toolkit: control flow and coverage

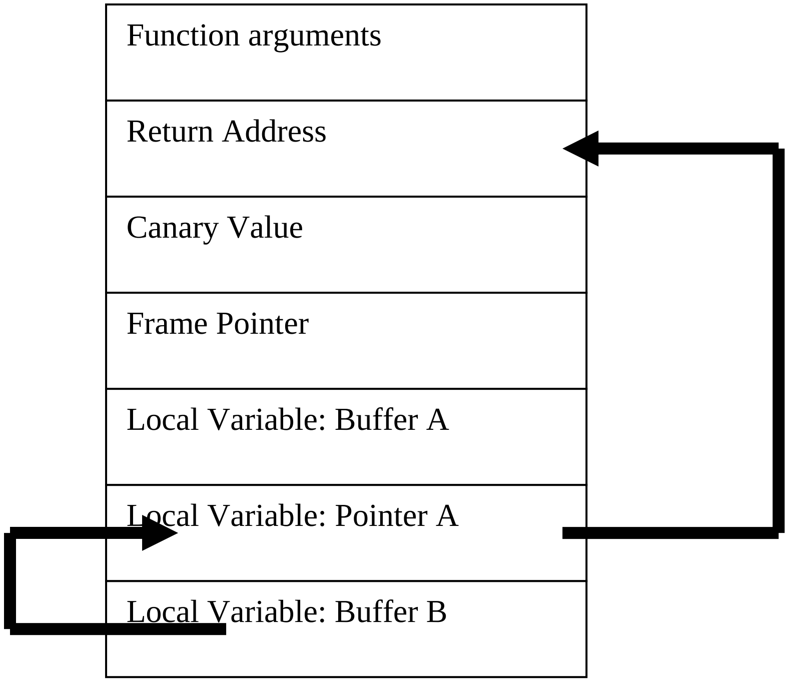
- Tracing input as it flows through software is an excellent method
- Exploiting differences between versions is also common
- Code coverage tools help you know where you have gotten in a program
  - dyninstAPI (Maryland)
  - Figure out how to get to particular system calls
  - Look for data in shared buffers



Attacker’s toolkit: buffer overflow foo

- Find targets with static analysis
- Change program control flow
  - Heap attacks
  - Stack smashing
  - Trampolining
  - Arc injection
- Particular examples
  - Overflow binary resource files (used against Netscape)
  - Overflow variables and tags (Yamaha MidiPlug)
  - MIME conversion fun (Sendmail)
  - HTTP cookies (apache)

- Trampolining past a canary



Function arguments

Return Address

Canary Value

Frame Pointer

Local Variable: Buffer A

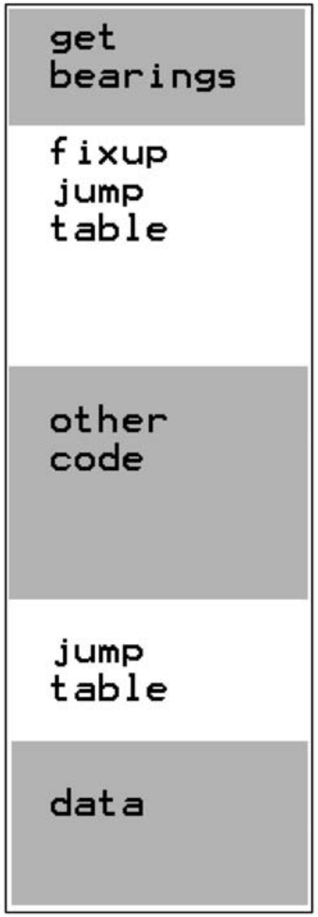
Local Variable: Pointer A

Local Variable: Buffer B

Attacker’s toolkit: shell code and other payloads

- Common payloads in buffer overflow attacks
- Size matters (small is critical)
- Avoid zeros
- XOR protection (also simple crypto)

- Payloads exist for
  - X86 (win32)
  - RISC (MIPS and sparc)
  - Multiplatform payloads



get bearings

fixup jump table

other code

jump table

data

Attacker's toolkit: rootkits

- The apex of software exploit…complete control of the machine
- Live in the kernel
  - XP kernel rootkit in the book
  - See http://www.rootkit.com

- Hide files and directories by controlling access to process tables
- Provide control and access over the network
- Get into the EEPROM (hardware viruses)

Attacker’s toolkit: other miscellaneous tools

- Debuggers (user-mode)
- Kernel debuggers
  - SoftIce
- Fault injection tools
  - FUZZ
  - Failure simulation tool
  - Hailstorm
  - Holodeck
- Boron tagging
- The “depends” tool
- Grammar rewriters

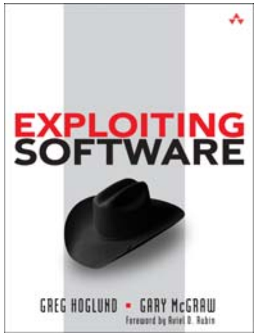
How attacks unfold

- The standard process
  - Scan network
  - Build a network map
  - Pick target system
  - Identify OS stack
  - Port scan
  - Determine target components
  - Choose attack patterns
  - Break software
  - Plant backdoor

- Attacking a software system is a process of discovery and exploration
  - Qualify target (focus on input points)
  - Determine what transactions the input points allow
  - Apply relevant attack patterns
  - Cycle through observation loop
  - Find vulnerability
  - Build an exploit

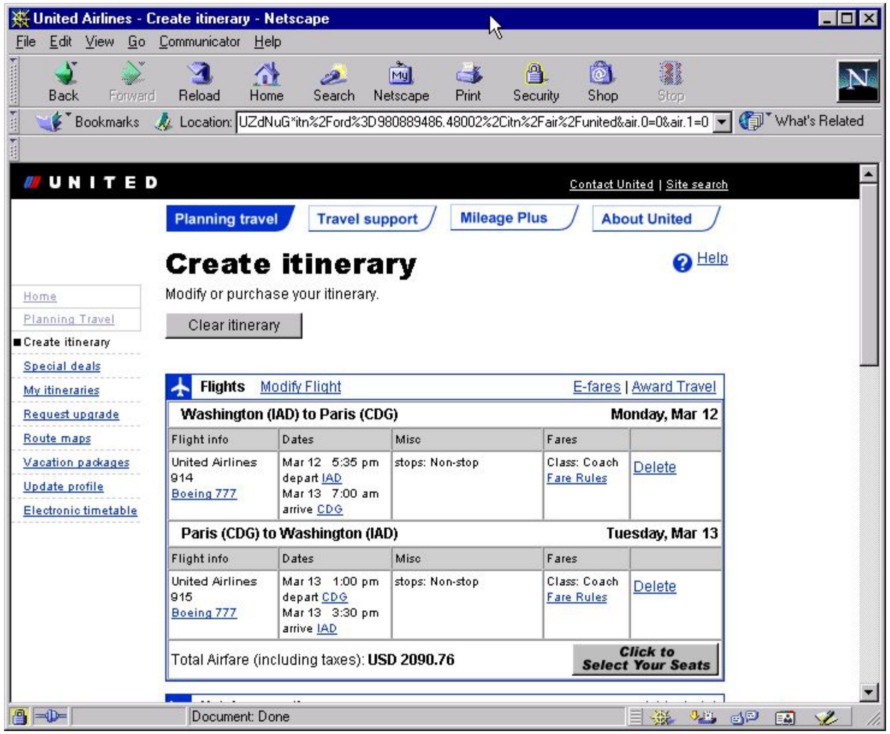
Knowledge: 48 Attack Patterns

- Make the Client Invisible
- Target Programs That Write to Privileged OS Resources
- Use a User-Supplied Configuration File to Run Commands That Elevate Privilege
- Make Use of Configuration File Search Paths
- Direct Access to Executable Files
- Embedding Scripts within Scripts
- Leverage Executable Code in Nonexecutable Files
- Argument Injection
- Command Delimiters
- Multiple Parsers and Double Escapes
- User-Supplied Variable Passed to File System Calls
- Postfix NULL Terminator
- Postfix, Null Terminate, and Backslash
- Relative Path Traversal
- Client-Controlled Environment Variables
- User-Supplied Global Variables (DEBUG=1, PHP Globals, and So Forth)
- Session ID, Resource ID, and Blind Trust
- Analog In-Band Switching Signals (aka “Blue Boxing”)
- Attack Pattern Fragment: Manipulating Terminal Devices
- Simple Script Injection
- Embedding Script in Nonscript Elements
- XSS in HTTP Headers
- HTTP Query Strings
- User-Controlled Filename
- Passing Local Filenames to Functions That Expect a URL
- Meta-characters in E-mail Header
- File System Function Injection, Content Based
- Client-side Injection, Buffer Overflow
- Cause Web Server Misclassification
- Alternate Encoding the Leading Ghost Characters
- Using Slashes in Alternate Encoding
- Using Escaped Slashes in Alternate Encoding
- Unicode Encoding
- UTF-8 Encoding
- URL Encoding
- Alternative IP Addresses
- Slashes and URL Encoding Combined
- Web Logs
- Overflow Binary Resource File
- Overflow Variables and Tags
- Overflow Symbolic Links
- MIME Conversion
- HTTP Cookies
- Filter Failure through Buffer Overflow
- Buffer Overflow with Environment Variables
- Buffer Overflow in an API Call
- Buffer Overflow in Local Command-Line Utilities
- Parameter Expansion
- String Format Overflow in syslog()



Attack pattern 1: Make the client invisible

- Remove the client from the communications loop and talk directly to the server
- Leverage incorrect trust model (never trust the client)
- Example: hacking browsers that lie (opera cookie foo)



Location: UZdNuG*itn%2Ford%3D980889486.48002%2Citn%2Fair%2Funited&air.0=0&air.1=0

Create itinerary

Modify or purchase your itinerary.

Flights Modify Flight — E-fares | Award Travel

Washington (IAD) to Paris (CDG) — **Monday, Mar 12**

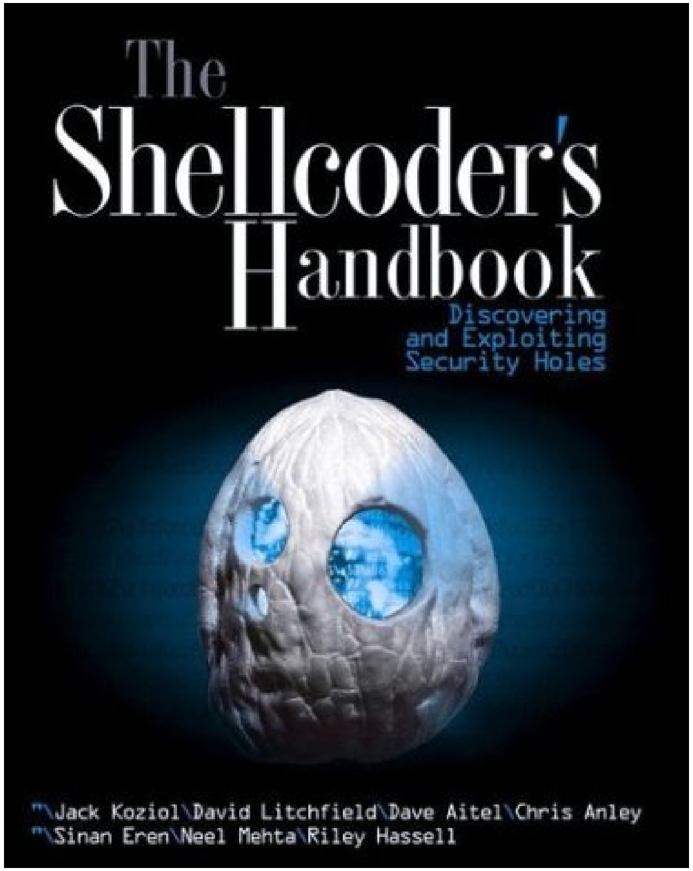
Flight info	Dates	Misc	Fares	
United Airlines 914 Boeing 777	Mar 12 5:35 pm depart IAD Mar 13 7:00 am arrive CDG	stops: Non-stop	Class: Coach Fare Rules	Delete

Paris (CDG) to Washington (IAD) — **Tuesday, Mar 13**

Flight info	Dates	Misc	Fares	
United Airlines 915 Boeing 777	Mar 13 1:00 pm depart CDG Mar 13 3:30 pm arrive IAD	stops: Non-stop	Class: Coach Fare Rules	Delete

Total Airfare (including taxes): **USD 2090.76**

Breaking stuff is important


The Shellcoder's Handbook
Discovering and Exploiting Security Holes
Jack Koziol\David Litchfield\Dave Aitel\Chris Anley
Sinan Eren\Neel Mehta\Riley Hassell

- Learning how to think like an attacker is essential
- Do not shy away from teaching attacks
  - Engineers learn from stories of failure
- Attacking group projects can be the most fun part of a course
- Fun is good! Software engineering is too boring!

Great, now what do we do about this?

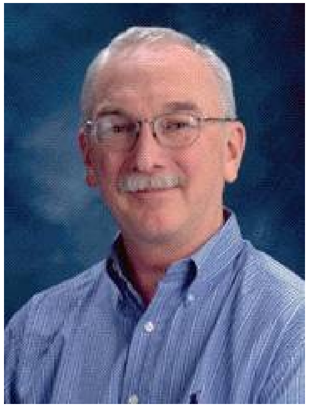
Software security critical lessons

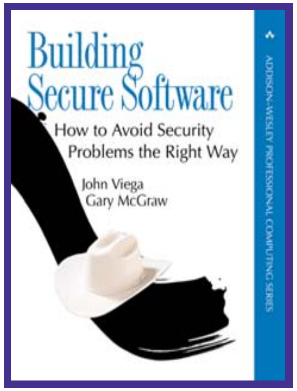
- Software security is more than a set of security functions
  - Not magic crypto fairy dust
  - Not silver-bullet security mechanisms
  - Not application of very simple tools
- Non-functional aspects of design are essential
- Security is an emergent property of the entire system (just like quality)
- To end up with secure software, deep integration with the SDLC is necessary

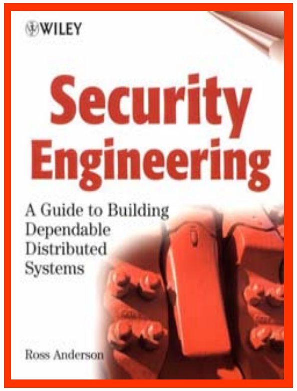
Ten guiding principles for secure design

1. Secure the weakest link
2. Practice defense in depth
3. Fail securely
4. Follow the principle of least privilege
5. Compartmentalize

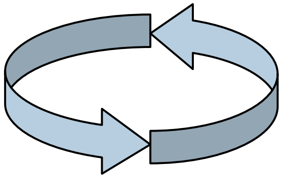
– Keep it simple
– Promote privacy
– Remember that hiding secrets is hard
– Be reluctant to trust
– Use your community resources

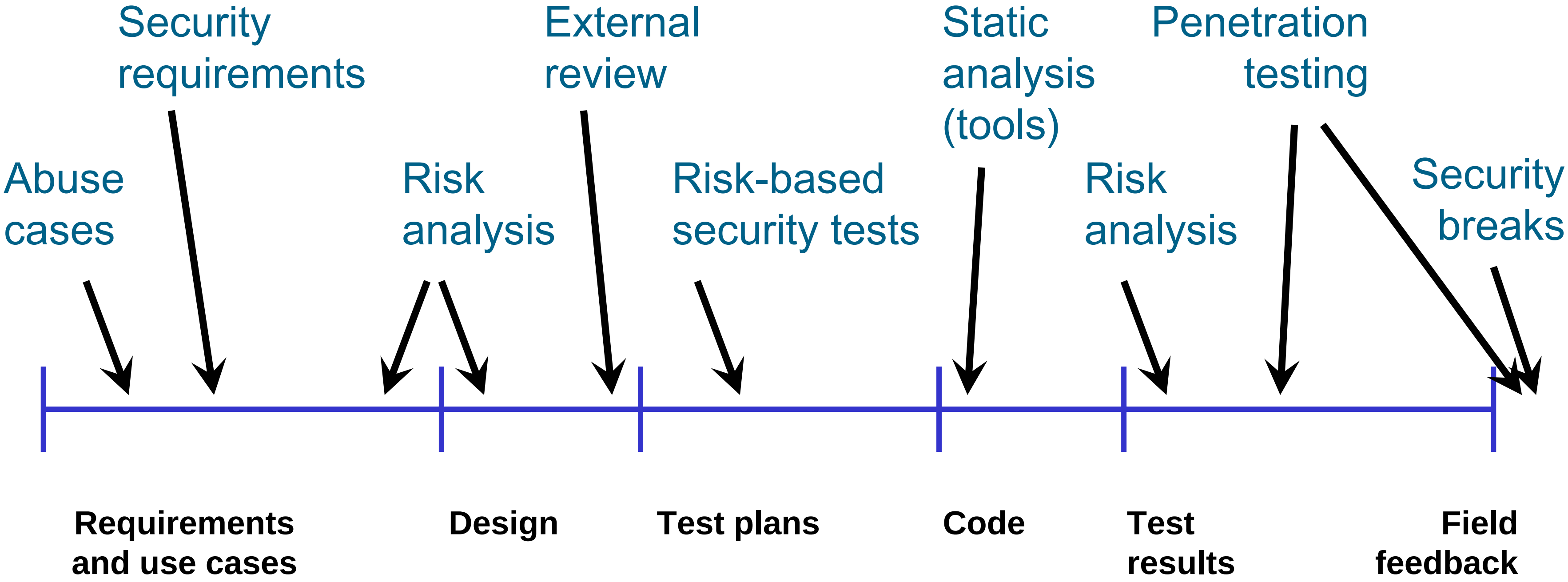



Building Secure Software
How to Avoid Security Problems the Right Way
John Viega
Gary McGraw
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



The antidote: Software security in the SDLC





Security requirements

External review

Static analysis (tools)

Penetration testing

Abuse cases

Risk analysis

Risk-based security tests

Risk analysis

Security breaks

Requirements and use cases

Design

Test plans

Code

Test results

Field feedback

Software security best practices

- Security best practices should be applied throughout the dev lifecycle
- Tendency is to “start right” (penetration testing) and declare victory
  - Not cost effective
  - Hard to fix problems
- Start as early as possible

- Abuse cases
- Security requirements analysis
- Architectural risk analysis
- Risk analysis at design
- External review
- Test planning based on risks
- Security testing (malicious tests)
- Code review with static analysis tools

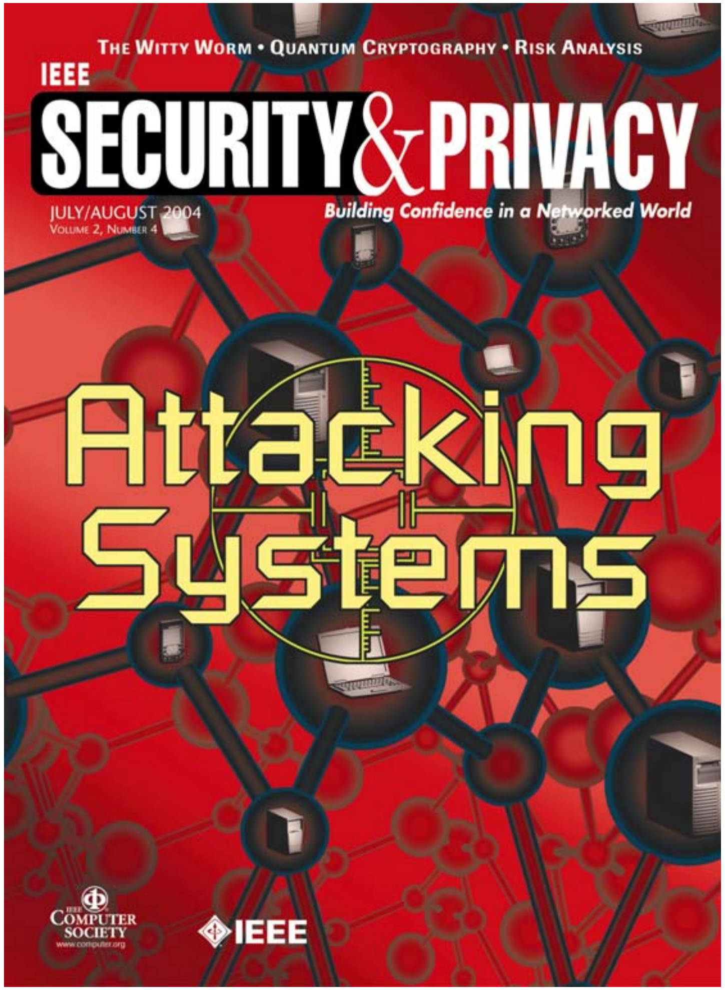
Where to learn more

IEEE Security & Privacy Magazine

- See the department on Software Security best practices called “Building Security In”

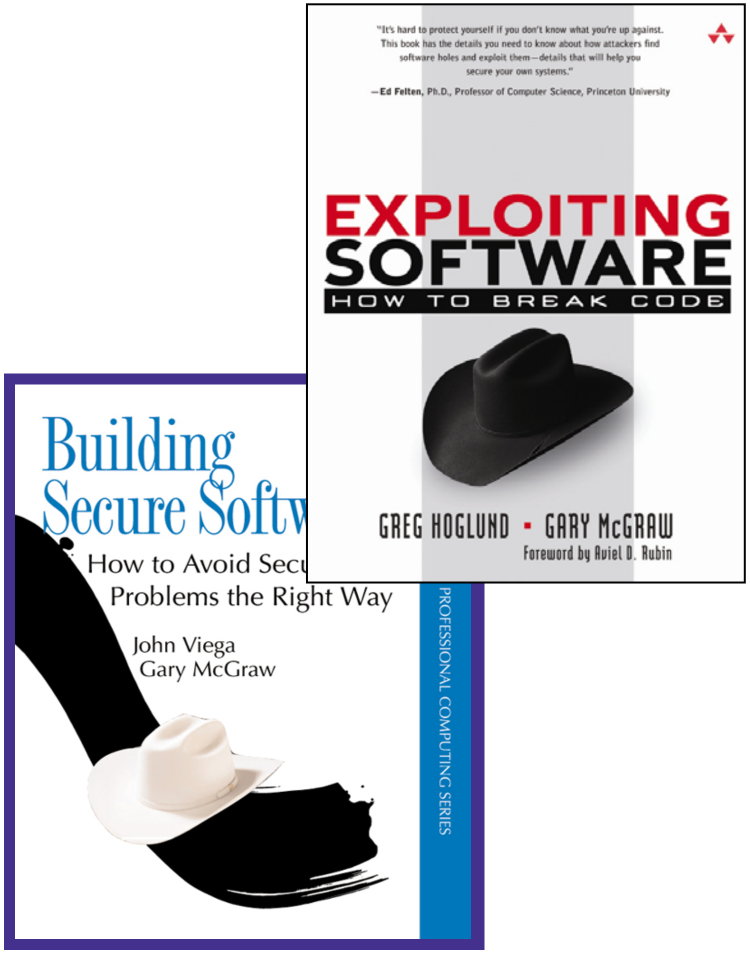
- Also see this month’s special issue on breaking stuff

http://www.computer.org/security



Pointers

- Cigital’s Software Security Group invents and practices Software Quality Management
  - WE NEED PEOPLE
- http://www.cigital.com/presentations/exploit04
- Use Exploiting Software and Building Secure Software
- Send e-mail: gem@cigital.com


"It's hard to protect yourself if you don't know what you're up against. This book has the details you need to know about how attackers find software holes and exploit them—details that will help you secure your own systems."
—Ed Felten, Ph.D., Professor of Computer Science, Princeton University
EXPLOITING SOFTWARE
HOW TO BREAK CODE
GREG HOGLUND ▪ GARY McGRAW
Foreword by Aviel D. Rubin
Building Secure Softw
How to Avoid Secu
Problems the Right Way
John Viega
Gary McGraw
PROFESSIONAL COMPUTING SERIES