Secure Coding

CSH6 Chapter 38

"Writing Secure Code"

Lester E. Nichols, M. E. Kabay, & Timothy Braithwaite

1

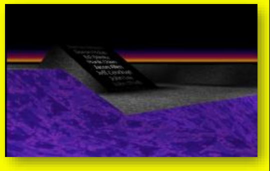
Topics

Introduction

- Chapter / lecture serves as overview & introduction to large subject
- Secure coding complex issue
- Involves human factors & technical issues
- Requires coordination & cooperation of many sectors in organization
- Starts with a few funny (or scary) examples of SQA failures

3

Easter Egg in Excel 97



- Undocumented keystrokes would activate flight simulator using full-screen graphics
  - ❑1. On a new Worksheet, Press F5
  - ❑2. Type X97:L97 and *Enter*
  - ❑3. Press the *Tab* key
  - ❑4. Hold *Ctrl-Shift*
  - ❑5. Click on the *Chart Wizard* toolbar button
  - ❑6. Use mouse to fly around - Right button forward/ Left button reverse
- Included rolling credits of developer names
- Pressing ESC would crash some systems
- http://www.youtube.com/watch?v=c6nY0QkG9nQ

4

Secret Writer's Society (1998)

- Game for children
- Read kids' writing back to them out loud
- Included filter of prohibited nasty words
  - ❑Curses, obscenities…
- Bug: proceeded to read ALL the bad words out loud to the children!
- "Children and parents were startled by the streams of foul language erupting from their computers."
- "The company's response was to deny that it was a significant problem."

5

Belligerent Crapper (2001)

- A 51-year-old woman was subjected to a harrowing two-hour ordeal [on 16 Apr 2001] when she was imprisoned in a hi-tech public convenience.
- Maureen Shotton, from Whitley Bay, was captured by the maverick cyberloo during a shopping trip to Newcastle-upon-Tyne.
- The toilet, which boasts state-of-the-art electronic auto-flush and door sensors, steadfastly refused to release Maureen, and further resisted attempts by passers-by to force the door.
- Maureen was finally liberated when the fire brigade ripped the roof off the cantankerous crapper.
- Maureen's terrifying experience confirms that it is a short step from belligerent bogs to Terminator-style cyborgs hunting down and exterminating mankind. [RISKS 21:35]

6

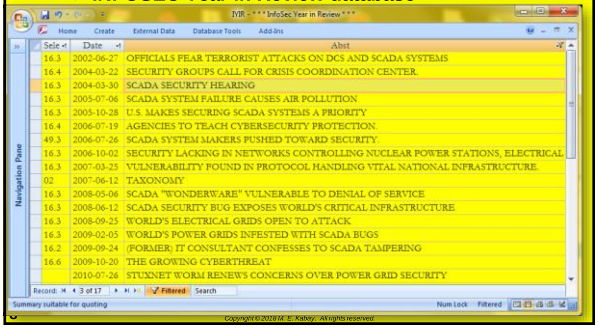
Waldo Goes Wild (2005)

- UCSF Medical Center
  - "Waldo" (named after famous Heinlein story) dispensed pills & potions
  - Size of a small washing machine
- Waldo suddenly refused to return to dispensary for new pills
- Went roaring past destination at high speed
- Crashed into radiation oncology department
  - Patient examination in progress
- "The psychotic pill pusher reportedly refused to leave, sending both doctor and patient fleeing for their lives."

7

SCADA System Insecurity

- Supervisory Control and Data Acquisition Systems
- INFOSEC Year in Review database



Sele	Date	Abst
16.3	2002-06-27	OFFICIALS FEAR TERRORIST ATTACKS ON DCS AND SCADA SYSTEMS
16.4	2004-03-22	SECURITY GROUPS CALL FOR CRISIS COORDINATION CENTER.
16.3	2004-03-30	SCADA SECURITY HEARING
16.3	2005-07-06	SCADA SYSTEM FAILURE CAUSES AIR POLLUTION
16.3	2005-10-28	U.S. MAKES SECURING SCADA SYSTEMS A PRIORITY
16.4	2006-07-19	AGENCIES TO TEACH CYBERSECURITY PROTECTION.
49.3	2006-07-26	SCADA SYSTEM MAKERS PUSHED TOWARD SECURITY.
16.3	2006-10-02	SECURITY LACKING IN NETWORKS CONTROLLING NUCLEAR POWER STATIONS, ELECTRICAL
16.3	2007-03-25	VULNERABILITY FOUND IN PROTOCOL HANDLING VITAL NATIONAL INFRASTRUCTURE.
02	2007-06-12	TAXONOMY
16.3	2008-05-06	SCADA "WONDERWARE" VULNERABLE TO DENIAL OF SERVICE
16.3	2008-06-12	SCADA SECURITY BUG EXPOSES WORLD'S CRITICAL INFRASTRUCTURE
16.3	2008-09-25	WORLD'S ELECTRICAL GRIDS OPEN TO ATTACK
16.3	2009-02-05	WORLD'S POWER GRIDS INFESTED WITH SCADA BUGS
16.2	2009-09-24	(FORMER) IT CONSULTANT CONFESSES TO SCADA TAMPERING
16.6	2009-10-20	THE GROWING CYBERTHREAT
	2010-07-26	STUXNET WORM RENEWS CONCERNS OVER POWER GRID SECURITY

8

Policy & Management Issues

- Security of code has become essential
  - Strategic importance
  - Yet many SW projects produce
    - ✓ Inadequate functionality (wrong goals)
    - ✓ Buggy code (not achieving goals)
- Fundamental problems
  - Short-term accounting fails to recognize long-term benefits of investing in low-bug code
  - Difficulty in proving negative: absence of bugs
- Topics on following slides:
  - Software TQM
  - Due Diligence
  - Regulatory & Compliance Considerations

9

Software TQM

- Software must adapt to constantly changing needs
  - ISO 9000 family of standards
  - Plan-do-check-act / plan-fix-monitor-assess
- Integrate security planning into every phase of SW cycle
  - Analysis, requirements, design, coding, implementation
  - Cannot effectively or efficiently retrofit security
- Expect iterative approach to compliance
  - Must cope with changing threat environment
  - Include security in modifications

10

Due Diligence

- Management must integrate security into performance metrics
- Evolving security information forces changes in best practices
- Boards / C-level executives becoming personally liable for failures
- Must establish & document security risk management in SW development
  - Thus demonstrate compliance with current standards
  - Meet standard of *due care and diligence in exercising fiduciary responsibilities*

11

Regulatory & Compliance Considerations

- Specific regulations usually dictate need for records; e.g.,
  - Sarbanes-Oxley
  - Gramm-Leach-Bliley
  - Health Insurance Portability & Accountability Act
- Keep records of problems
  - Identification date & agent
  - Severity (implications, systems affected)
  - Report to management
  - Remediation target & completion date

12

Technical & Procedural Issues

- Development team often under time pressure
  - Sales / management personnel may value time to market over lack of bugs
  - Must fight to adhere to systematic SW development methodology with adequate prevention, monitoring & correction of errors
- Topics on following slides:
  - Requirements Analysis
  - Design
  - Operating System
  - Best Practices & Guidelines
  - Languages

13

Requirements Analysis

- Staircase principle: delaying correction multiplies cost of error 10x
  - Requirements analysis
  - Requirements definition
  - Design
  - Coding
  - Implementation
- Analysis must include discussions of security needs (confidentiality, control, integrity, authenticity, availability, utility)
- Definition must explicitly define function goals that include these security aspects

14

Design

- Data structures design *instantiates* information model
- Logic design instantiates relationships among elements of model
- Procedural model instantiates data flow and object relations
  - Include access privileges, restrictions
- Project planning must allow for adequate software quality assurance [See *CSH6* Chapter 39, "Software Development & Quality Assurance"]

15

Operating System

- OS is at core of security implementation
- Secure OS implements
  - Completeness: all access to information managed by kernel
  - Isolation: kernel protected against unauthorized access
  - Verifiability: kernel proven to meet design specifications

16

Best Practices & Guidelines (1)

- Excellent guides to best practices:
  - NIST Special Publications Series 800
    - ✓ http://csrc.nist.gov/publications/PubsSPs.html
  - List of recommendations in §38.3.4 (below)
- Impose strong I&A
- Document code thoroughly
- Use local variables, not global variables, when storing sensitive data
- Reinitialize temporary storage immediately after the last legitimate use
- Limit functionality in a specific module to what is required for a specific job

17

Best Practices (2)

- Define views of data in databases that conform to functional requirements and limit access to sensitive data
- Use strong encryption (*not* homegrown encryption)
- Disallow access by programmers to production databases
- Randomize or otherwise mask sensitive data when generating test subsets from production data
- Use test-coverage monitors
- Integrate logging capability into all applications

18

Best Practices (3)

- Create log-file records with cryptographically sound message authentication code (MAC) that itself includes the MAC of the preceding record
- Log all process initiations for a program and log process termination
- Log all modifications to records
- Use record-level locking
- Unlock a sequence of locks in the inverse order of the lock sequence to prevent deadlocks
- Sign your source code using digital signatures
- Use checksums in production executables

19

Best Practices (4)

- Design code holistically, including tests of what should *not* be accepted
- Establish criteria for defining and determining sensitivity of data being processed
- Implement formal SQA control processes
- Identify mandatory OS & NW security settings for code to run securely
- Verify digital signatures of routines being loaded for execution
- Verify digital signatures or checksums of all executables being loaded at system restart

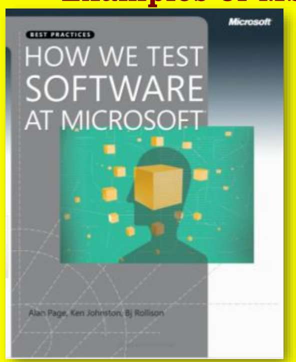
20

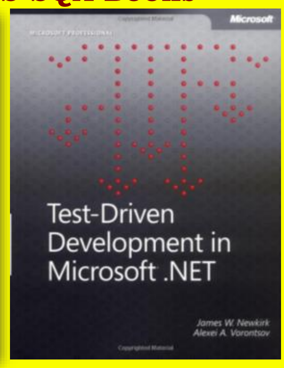
Microsoft SQA Project

- Nov 2001: Bill Gates -- two top priorities
  - Improving reliability of MS sw
  - Conquering market for "tablet" computers
- Jan 2002: Trustworthy Computing initiative launched
  - Choose security over features
  - Emphasize security right out of the box
  - Privacy key concern
- Feb 2002: MS hires top security expert
  - Scott Charney – famous expert
  - Oversee MS strategies for enhanced security

21

Examples of MS SQA Books





22

Languages

- To degree possible, take advantage of security features of programming tools
- Different languages offer different advantages
  - Java includes sandbox for isolation of processes
  - PASCAL offers strong typing
  - But C & C++ have almost no security restrictions during execution
- Security utilities and routines available for integration
  - RSA toolkits
  - Textbooks (e.g, Schneier's *Applied Cryptography*)

23

Types of Software Errors

Internal Design or Implementation

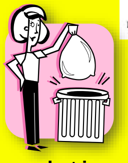
- Initialization
- Logic Flow
- Calculation
- Boundary Condition Violations
- Parameter Passing
- Race Condition
- Load Condition
- Resource Exhaustion
- Resource, Address, or Program Conflict with the Operating System or Application(s)
- Regulatory Compliance Considerations
- Other Errors

OOPS!

DARN!

24

Initialization Errors



- Insidious & difficult to find
- Failing to initialize data may leave garbage in registers
  - ❑So program *may* fail depending on what is in registers from some previous use
  - ❑And program may fail on 1st use
  - ❑Intermittent problem – *race condition*
- Others may always fail on 1st use because of zero and blank values from OS or language rules
- Some programs write initialized values to disk
  - ❑So fail only on 1st use

25

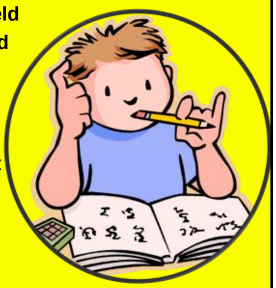
Logic Flow

- Modules pass control to each other or other programs
- So calling wrong function causes error
- Problems occur when code branches to subroutine lacking a RETURN code
- Often data dependent
  - ❑Intermittent failures
- Use DEBUG utility to check current execution, step through code

26

Calculation

- Frequent problem: wrong size of storage element
  - ❑Store long value in short field
  - ❑Truncates data, leads to bad calculations
- Errors in formula may be hard to catch
  - ❑Especially if specifics of formula are data dependent



27

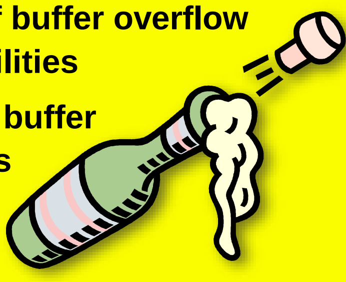
Boundary Condition Violations

- Critically important to define limits on acceptable data values = *bounds*
- Most common error: buffer overflow
  - ❑Data exceeds expected length of storage
  - ❑Usually exploited using input buffers
  - ❑Send long data string into input which is not checked for length
  - ❑Program writes data beyond end of array
  - ❑May execute portions of data stream
- Always check data length before storing
  - ❑Reject; or
  - ❑Truncate

28

Buffer Overflows

- What is a buffer overflow?
- Origin of buffer overflow vulnerabilities
- Fighting buffer overflows

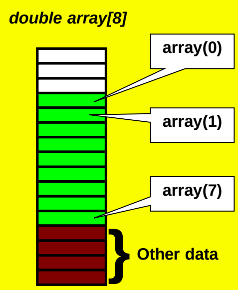


29

What Is a Buffer Overflow?

Programming concept:

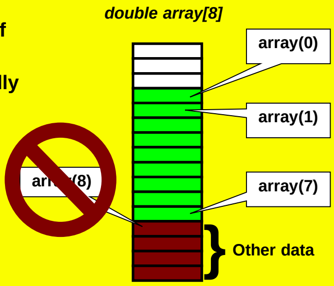
- Define (declare, dimension)
  - ❑list (array, indexed variable, string)
  - ❑of certain size
- To reserve area of memory for specific use during execution



30

Origin of Buffer Overflow Vulnerabilities

- In using a member of an array (an indexed variable), it is critically important to avoid addressing *out of bounds*
- Doing so is called a *bounds violation*
- Can corrupt data of other variables



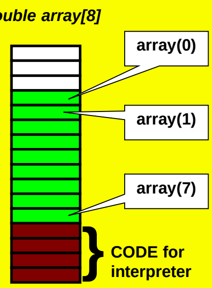
Consequences of Bounds Violations

Possible to see
- Compiler error
- Run-time error
- Program errors – bad results
- Program crash
- System crash

But most dangerous problem occurs in *interpreters*
- Programs that dynamically interpret instructions
- E.g., browsers, Web server programs

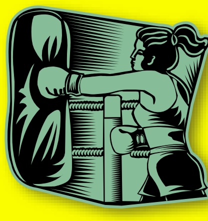
Bounds Violations in Interpreters

- Some interpreters read areas of data as instructions (code)
- Bounds violation can put *data* into *code* areas of working memory
- Thus *bad data* can become equivalent to *bad code*
- Can sometimes execute *arbitrary code*
- Obtain unauthorized privileges



Fighting Buffer Overflows

- Programmers need to use good quality assurance techniques
  - Test long input strings
  - Test below, at and above boundary conditions
- System / network / security staff: *check* for new buffer overflows & install *patches*
  - Use ICAT Metabase frequently
  - Subscribe to CERT-CC alerts from http://www.cert.org

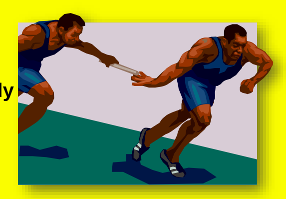


Fighting Buffer Overflows (cont'd)

- Managers need to understand that ***every buffer overflow is a failure of quality assurance***
- Stop allowing manufacturers to publish inadequately tested software as production versions
- Stop letting manufacturers push quality assurance onto the client base
- Complain loudly to manufacturers when there are buffer overflows in their software – and, if possible, buy competing products with better quality assurance

Parameter Passing

- Parameters passed among routines
- Wrong name or wrong subscript in array variable may pass wrong data
- Can cause errors in
  - Calculations
  - Logic flow
- Cascade of errors likely
  - Data corruption
  - Aborts

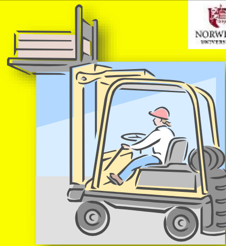


Race Condition

- Problems occur when
  - ❑Specific sequence of events required for correct operation
  - ❑But no enforcement or guarantee of sequence
- Known as race condition because correct operation is a race between events
- Classic example occurs in incorrect locking strategies
  - ❑A locks 1 and then locks 2
  - ❑B locks 2 and then locks 1
  - ❑OK if B tries to lock 2 AFTER A locks 2
  - ❑But deadlock if B locks 2 before A tries to locks 2

37

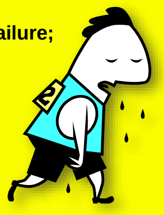
Load Condition



- Exceeding expected
  - ❑Storage
  - ❑Transactions
  - ❑Users
  - ❑Network bandwidth utilization
- May cause major declines in throughput
  - ❑Problems of availability
- Should use automated testing for simulation
  - ❑Identify bottlenecks
  - ❑Take preventative actions

38

Resource Exhaustion

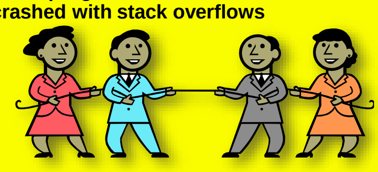


- Exhausting resources can cause failure; e.g.,
  - ❑Cannot write to full disk
  - ❑Cannot obtain memory location
  - ❑Cannot obtain CPU in time for real-time processing
  - ❑Running out of system table entries
- Running with inadequate main memory may lead to excessive swapping between virtual memory and main memory
  - ❑*Thrashing*

39

Interapplication Conflicts

- OS makers routinely provide application SW makers with coding guidance & kits
- But as OS versions & patches move on, older applications may fail
  - ❑E.g., HP3000 MPE increased stack requirements
  - ❑Older programs close to stack limitations crashed with stack overflows



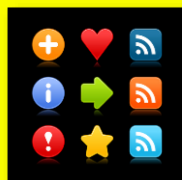
40

Other Errors

- Sending bad data to devices
- Ignoring error codes from devices
- Trying to use busy or missing devices
- Improper builds of code (using wrong routines)

41

User Interface



- User Virtual Machine
  - ❑Screens
  - ❑Mouse & keyboard
  - ❑Printed outputs
- Especially important to realize that user is not telepathic:
  - ❑Cannot automatically know what designer / programmer knows and assumes ("HIT ANY KEY" → "Where's the ANY key??")
  - ❑So programmers responsible for envisaging possible pitfalls and preventing problems
- Documentation & training essential

42

Functionality (1)

- When performance reasonably expected is missing, confusing, awkward, difficult or impossible, we have a functionality problem
- See list of suggestions in §38.4.1.1.2 & on following slides

43

Functionality (2)

- Features are not documented
- Required information is missing
- A program fails to acknowledge legitimate input
- There are factual errors or conflicting names for features
- There is information overload
- The material is written to an inappropriate reading level
- The cursor disappears, or is in the wrong place
- Screen displays are wrong
- Instructions are obscured

44

Functionality (3)

- Identical functions require different operations in different screens
- Improperly formatted input screens exist
- Passwords or other confidential information not obscured or protected adequately
- Tracing user data entry or changes unavailable or incomplete
- Segregation of duties not enforced (Can be particularly critical for organizations subject to legal and regulatory requirements)

45

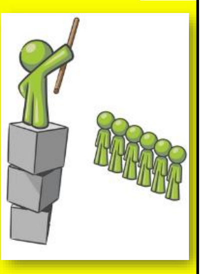
Control (Command) Structure (1)

Control structure errors can cause serious problems because they can result in:

- Users getting lost in a program
- Users wasting time because they must deal with confusing commands
- Loss of data or unwanted exposure of data
- Work delay
- Financial cost
- Unanticipated exposure to data leakage or compromise; can result in significant liability if consumers' personal identifying information (PII) compromised
- Data not being encrypted as intended or being visible to unauthorized users

46

Control (Command) Structure (2)



Some common errors listed in §38.4.1.13 include:

- Inability to move between menus
- Confusing and repetitive menus
- Failure to allow adequate command-line entries
- Requiring command-line entries that are neither intuitive nor clearly defined on screen
- Failure of the application program to follow the operating system's conventions
- Failure to distinguish between source and parameter files, resulting in wrong values being made available to user through interface, or failure to identify source of error

47

Control (Command) Structure (3)

- Inappropriate use of keyboard, when new programs do not meet standard of a keyboard that has labeled function keys tied to standard meanings
- Missing commands from code and screens resulting in user being unable to access information, to utilize programs, or to provide for system to be backed up and recoverable
- Inadequate privacy or security that can result in confidential information being divulged, in complete change or loss of data without recoverability, in poor reporting, and even in undesired access by outside parties

48

Performance (1)

- Speed important in interactive software
- Problem can include
  - Slow response
  - Unannounced case sensitivity,
  - Uncontrollable and excessively frequent automatic saves
  - Inability to save
  - Limited scrolling speed

49

Performance (2)

- Slow operation can depend on (but is not limited to)
  - OS
  - Other applications running concurrently
  - Memory saturation and thrashing
  - Memory leakage (the failure to deallocate memory that is no longer needed)
  - Disk I/O inefficiencies (e.g., reading single records from very large blocks),
  - Program conflicts (e.g., locking errors)
- See *CSH6* Chapter 52 "Application Controls"

50

Performance (3)

- Program designs can make it difficult to change their functionality
  - Response to changing requirements
- E.g., database design – defining primary index field
  - Determines how records stored on disk
  - Can greatly speed access to records during sequential reads on key values for that index
  - But can be counterproductive if main method for accessing records = sequential reads on completely different index

51

Output Format

- User cannot change appearance of output
  - Font
  - Underlining
  - Boldface
  - Spacing
- Delays in printing or saving document
- Problems in scaling tables, figures, graphs
- Errors in displayed precision of numbers

52

Assurance Tools & Techniques

- Education Resources
  - See list in *CSH6* §38.5.1 p 38.13
- Code Examination & Application Penetration Testing
  - White Box
  - Black Box
  - Gray Box
- Standards & Best Practices

See next slides

53

White Box

- AKA *glass, structural, open, clear* box
- Tests using knowledge of internals
- Advantages of white box testing:
  - Easy to find out which type of input/data can help in testing
  - Helps in optimizing code
  - Helps in removing extra (useless) lines of code which can bring in hidden defects
- Disadvantages of white box testing:
  - Skilled tester needed → increases cost
  - Nearly impossible to look into every bit of code to find hidden errors

54

Black Box (1)

- Testing without knowledge of internal workings
- AKA *behavioral, functional, opaque box, closed box*
- Tester & programmer can be independent of one another
- Avoid programmer bias toward own work
- Test groups often used
- Test planning can begin as soon as specifications written

55

Black Box (2)

Advantages of black box testing:

- More effective on larger units of code than glass box testing
- Tester needs no knowledge of implementation, including specific programming languages
- Tester and programmer independent of each other
- Tests done from user's point of view
- Help to expose ambiguities or inconsistencies in specifications.
- Test cases can be designed as soon as specifications are complete.

56

Black Box (3)

Disadvantages of black box testing:

- Few possible inputs can actually be tested
- Without clear and concise specifications, test cases hard to design
- Unnecessary repetition of test inputs if the tester not informed of test cases programmer has already tried.
- May leave many program paths untested
- Cannot be directed toward specific segments of code that may be very complex

57

Gray Box

- Combination of black box testing and white box testing
- Tester does know some of internal workings of software under test
- Applies limited number of test cases to internal workings of software under test
- Then takes black box approach in applying inputs to software under test and observing outputs

58

Standards & Best Practices

- Consensus
  - Perform automated testing
  - Make a test plan
  - Follow a specific methodology
  - Test at every stage
  - Test all system components
- Standards
  - ISO 17799, Information Technology: Code of Practice for Information Security Management
  - ISO/IEC 15408, Evaluation Criteria for IT Security (the Common Criteria)
  - SSE-CMM, System Security Engineering Capability Maturity Model
  - ISO/IEC WD 15443, Information Technology: Security Techniques

59

Now go and study

60