



The SANS/CWE Top 25 Checklist: Friend or Foe?

Matt Bishop

Dept. of Computer Science
University of California, Davis



February 9, 2009
Matt Bishop

CISO Lecture Series

Slide #1



Outline

- ➔ What is security?
- ➔ Security and assurance
- ➔ Using checklists
- ➔ Thoughts on secure programming
- ➔ How the SANS/CWE Top 25 Programming Errors fits into this
- ➔ Conclusion





Bias Alert!

- ➔ I was one of the contributors to the SANS/CWE Top 25 list
- ➔ I have had a long, pleasant, and fruitful relationship with SANS
 - I have disagreed with them about some things, though





Opening Thought

Overconfidence breeds error when we take for granted that the game will continue on its normal course; when we fail to provide for an unusually powerful resource—a check, a sacrifice, a stalemate. Afterwards the victim may wail, “But who could have dreamt of such an idiotic-looking move?”

—Fred Reinfeld, *The Complete Chess Course*





The Problem

Weinberg's Second Law:

If builders built buildings the way
programmers wrote programs, the first
woodpecker to come along would destroy
civilization





The Effect

- ➔ The information infrastructure relies on software
 - Software running the power grid is antiquated
 - Business relies on Windows, well known for security problems
 - Linux based on the UNIX system, which was not designed with security in mind
 - Internet protocols designed for a different environment





What is Security?

“Securing the information infrastructure ...”

What ***exactly*** does that mean?





Defining Security

- ⇒ Defined by “security policy”
- ⇒ Formal definition: partitions system states into “allowed” and “disallowed”
- ⇒ In practice: delineation is not precise
 - Specification of “allowed” not complete
 - Specifications change over time
 - Specifications defined as new situations arise





Key Point #1

- ➔ Security is a function of environment
 - U.S. military wants to keep information *secret*
 - Amazon want to keep (most) information *non-secret*
 - Almost all security mechanisms require strict accountability, to the individual
 - In electronic voting, you cannot associate an individual with a ballot





Security and Assurance

- ➔ Assurance: confidence that an entity meets its requirements
- ➔ Security: policy stating what is, and is not, allowed
- ➔ Security assurance: confidence that an entity correctly implements the given security policy





Establishing Assurance

- ➔ “confidence that an entity meets its security requirements, *based on specific evidence provided by the application of assurance techniques*”
- Informal techniques
 - Semiformal techniques
 - Formal techniques





Evaluating Evidence

- ➔ Best: mathematical and logical
 - Not practical in most environments
- ➔ Next best: rigorous framework of requirements specification, arguments tied to specifically stated assumptions
 - Practical but very time- and budget-consuming





Make It More Practical

- ➔ Requirements become checklists
 - Implies *checklists are meaningful and practical*
- ➔ Adducing of evidence becomes validating that you follow the checklist
 - Need to do this so an independent observer can validate it





Teaching vs. Training

➔ Analogy to teaching

- Academic institutions teach principles and concepts, and focus less on their immediate application
- Training schools focus on how to do things, and less on general principles and concepts

➔ Just different





Checklists

- ➔ First axis: guidance or specific?
 - Guidance checklists prompt memory, and require user to understand when to follow, and ignore, items in list
 - Specific checklists list items to be performed, and require user to perform items in list





Checklists

- ➔ Second axis: use for doing or auditing?
 - Doing checklists list items to be done, and the user must perform them
 - Auditing checklists list items to be checked, but the auditor need not do the items; she must check they are done





Applying This

- ➔ Training: specific checklists usually more appropriate than guidance checklists
- ➔ List of steps to perform
- ➔ Assume a particular environment
- ➔ Often assume users have common basis
 - Understand the steps in the checklist
 - Understand any ancillary, but omitted, information needed to apply the steps





Applying This

- ➔ Academic education: guidance checklists usually more appropriate than specific checklists
- ➔ List of principles, ideas as a memory prompt
 - No substitute for understanding!
- ➔ General enough for most environments
 - Students learn how to apply ideas, in effect specializing the generic checklist





Key Point #2

The nature of a checklist, and the environment in which it is used, determines its usefulness as assurance evidence





Example: CIS Checklists

- ➔ Detailed instructions for securing systems
 - *Intended for practitioners (CIS says this explicitly)*
- ➔ Assumptions about environment
 - CIS doesn't state these overall; some items ask user to determine if the step is appropriate
- ➔ Very specific
 - “Do you need to run telnet,” not “do you want to allow cleartext passwords over the net” or “do you want to allow net access to specific hosts”





Assuring Checklists

- ➔ Does the checklist's (assumed) policy match yours?
 - Devise a system that is secure but violates some of the items in the checklist
 - Devise a system that is not secure but does not violate any item in the checklist
- ➔ **Critical!**
 - Determine the assumptions underlying the recommendations





Key Point #3

Find and question assumptions!!!





Critical Assumption

- ➔ The checklist is meaningful
- ➔ In Common Criteria:
 - Protection Profile describes functionality of system being evaluated
 - Similar to a checklist (but it isn't)
 - Assurance class for PP is “Protection Profile Evaluation” or class APE





Example: Voting System Standards

- ➔ Describe what electronic voting systems must meet
 - General goals of voting
 - Very detailed requirements for the systems
 - Implication: meet these requirements and you will meet the general goals





Nope!

- ➔ Non-security of e-voting systems well known
- ➔ RABA study (2004): red team test, compromised voting system in 5 minutes, county tally system in 25.
- ➔ California Top-to-Bottom Review (2007): rigged voting systems to lie to voter, among other things





Think Sideways

- ➔ Next proposed VVSG:
 - “Open-Ended Vulnerability Testing” (red teaming)
- ➔ Very controversial, apparently
 - Requirements for passing are unclear
 - Even if system meets the requirements, it could still fail





Key Point #4

➔ Just because you meet all elements of a checklist, that does not mean you are secure!





Secure Programming

- ➔ Style of programming intended to make the program more secure
 - Secure defined in terms of security policy
- ➔ Two parts:
 - Security related to the particular problem
 - Security related to generic problems





Particular Problem

- ➔ Implement a web server that restricts access to a particular set of people
 - Adequate identification
 - Adequate authentication
 - What access is appropriate for each individual





Generic Problems

- ➔ Implement a web server that restricts access to a particular set of people
 - Buffer overflows leading to unauthorized access
 - Hijacking connections leading to unauthorized user taking over a legitimate session after it has been established
 - Race condition allowing a one-time password authentication scheme to accept the same password twice





Focus

- ➔ “Secure programming” usually refers to the second
 - Quality of code focuses on buffer overflows, race conditions, type clashes, etc.
- ➔ Secure coding checklists typically emphasize this
 - But you need to consider the first, too!





SANS/CWE Top 25 List

- ➔ Designed for *education*, not verification
 - No specific steps to take
 - Outlines general approaches
- ➔ Focuses on *types* of problems
 - It does *not* list top 25 specific problems
- ➔ Definitely a “guidance checklist”





Categories

- ➔ Insecure Interaction Between Components
- ➔ Risky Resource Management
- ➔ Porous Defenses





Insecure Interaction Between Components

- ⇒ Improper Input Validation
- ⇒ Improper Encoding or Escaping of Output
- ⇒ Failure to Preserve SQL Query Structure (“SQL Injection”)
- ⇒ Failure to Preserve Web Page Structure (“Cross-site Scripting”)
- ⇒ Failure to Preserve OS Command Structure (“OS Command Injection”)
- ⇒ Cleartext Transmission of Sensitive Information
- ⇒ Cross-Site Request Forgery (CSRF)
- ⇒ Race Condition
- ⇒ Error Message Information Leak





Risky Resource Management

- ➔ Failure to Constrain Operations within the Bounds of a Memory Buffer
- ➔ External Control of Critical State Data
- ➔ External Control of File Name or Path
- ➔ Untrusted Search Path
- ➔ Failure to Control Generation of Code (“Code Injection”)
- ➔ Download of Code Without Integrity Check
- ➔ Improper Resource Shutdown or Release
- ➔ Improper Initialization
- ➔ Incorrect Calculation





Porous Defenses

- ➔ Improper Access Control (Authorization)
- ➔ Use of a Broken or Risky Cryptographic Algorithm
- ➔ Hard-Coded Password
- ➔ Insecure Permission Assignment for Critical Resource
- ➔ Use of Insufficiently Random Values
- ➔ Execution with Unnecessary Privileges
- ➔ Client-Side Enforcement of Server-Side Security





Then Specific Ideas

- ➔ Example: Improper Input Validation
 - Summary and Discussion
 - Prevention and Mitigations
 - Architecture and Design
 - Implementation
 - Attack Patterns
 - Pointers to more detailed versions of the problem





Suitable for Teaching

- ⇒ How do you find these problems?
 - Checklist gives good ideas
- ⇒ How do you solve them?
 - Checklist gives general advice
- ⇒ What happens if you don't solve them?
 - Checklist identifies attacks





Threat Model

- ⇒ Authors developed specific threat model
 - Done with input from top security experts, including commercial folks, consultants, government, and academics
- ⇒ Model of attacker is “a skilled, determined attacker to break into the software”





Don't Apply This Blindly

- ➔ Example: Download of Code Without Integrity Check
 - May not be possible
 - Vendor may have created problem, so signature verifies the bad software is unchanged
 - Attacker may have subverted distribution mechanism





Another Example

- ➔ **Cleartext Transmission of Sensitive Information**
 - Good if data goes out on public networks
 - Who cares if only on trusted network?
 - Exercise: what are the assumptions 😊?





Do Apply This as Guidance

- ➔ Ask whether your vendors and your own programmers and developers consider these problems when writing code
 - Critical: provide management support for them!
- ➔ Try to generalize these beyond the computer and network
 - Look at the procedures and policies for your systems





Example

⇒ Untrusted Search Path

- Top 25 context: which library, program, configuration file, etc. are you loading?
- Other context: who sent you that email, really?
 - “Reverse priming the pump”





Another Example

- ➔ Improper Encoding or Escaping of Output
 - Giving output in metric units when English units are expected (or vice versa)
 - Anyone with teenagers knows exactly how this applies!





Key Point #5

➔ Forget a perfect checklist. It is as elusive as perfect security. A key skill is ...

knowing when to ignore it





Conclusion

- ➔ Checklists can be good or bad
- ➔ **Never** use a checklist blindly
 - Always ask what assumptions it makes, and evaluate what it says in that light
- ➔ Beware of those who provide assurance evidence in the form of checklists!
 - They usually don't think sideways, and so miss problems





Key Point #6

- ➔ Satisfying a checklist is not a goal. It is a means to a goal.
- ➔ Never confuse satisfying a checklist with security.
- ➔ **Security is the goal.**
 - Focus on that. If the checklist helps, use it. If not, discard it.





Final Thought: Clear Overall Goals

Gentlemen,

Whilst marching from Portugal to a position which commands the approach to Madrid and the French forces, my officers have been diligently complying with your requests which have been sent by H.M. ship from London to Lisbon and thence by dispatch to our headquarters.

We have enumerated our saddles, bridles, tents and tent poles, and all manner of sundry items for which His Majesty's Government holds me accountable. I have dispatched reports on the character, wit, and spleen of every officer. Each item and every farthing has been accounted for, with two regrettable exceptions for which I beg your indulgence.

Unfortunately the sum of one shilling and ninepence remains unaccounted for in one infantry battalion's petty cash and there has been a hideous confusion as to the number of jars of raspberry jam issued to one cavalry regiment during a sandstorm in western Spain. This reprehensible carelessness may be related to the pressure of circumstance, since we are war with France, a fact which may come as a bit of a surprise to you gentlemen in Whitehall.

This brings me to my present purpose, which is to request elucidation of my instructions from His Majesty's Government so that I may better understand why I am dragging an army over these barren plains. I construe that perforce it must be one of two alternative duties, as given below. I shall pursue either one with the best of my ability, but I cannot do both:

1. To train an army of uniformed British clerks in Spain for the benefit of the accountants and copy-boys in London or perchance:
2. To see to it that the forces of Napoleon are driven out of Spain.

—Duke of Wellington, to the British Foreign Office, London, 1812





Reference

- ➔ SANS/CWE Top 25 Most Dangerous Programming Errors
 - <http://cwe.mitre.org/top25/>





Me!

Matt Bishop
Department of Computer Science
University of California at California
1 Shields Ave.
Davis, CA 95616-8562

phone: +1 (530) 752-8060

email: bishop@cs.ucdavis.edu

www: <http://seclab.cs.ucdavis.edu/~bishop>

