

Secure Software Development Lifecycle



This lecture provides reference material for the book entitled "The Art of Software Security Testing" by Wysopal et al. © 2007

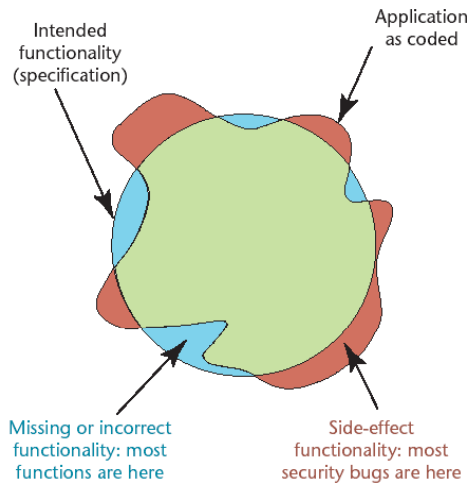
This lecture material is copyrighted by Laurie Williams (2007). However, you are encouraged to download, forward, copy, print, or distribute it, provided you do so in its entirety (including this notice) and do not sell or otherwise exploit it for commercial purposes.

For PowerPoint version of the slides, contact Laurie Williams at williams@csc.ncsu.edu.

Security . . . Not getting better

Year	Total vulnerabilities cataloged
Q1-Q2, 2007	3,907
2006	8,064
2005	5,990
2004	3,780
2003	3,784
2002	4,129
2001	2,437
2000	1,090
1999	417
1998	262
1997	311
1996	345
1995	171
Totals	34,687

Security Testing: Testing for What It's NOT supposed to do



S>> Re.search Group
Software Engineering @ NCSU

Thompson, Herbert, " , IEEE Security and Privacy, July/Aug 2003, pp. 83-86.

NC STATE UNIVERSITY

Security vs. Reliability

- **Reliability** - The ability of a system or component to perform its required functions under stated conditions for a specified period of time.
- **Security** – The ability of a system or component to perform its required functions without confidentiality, integrity, and availability losses for a specified period of time.

S>> Re.search Group
Software Engineering @ NCSU

© 2005 Laurie Williams

NC STATE UNIVERSITY

Latent Code Problem Definitions

- **Fault** - *reliability concept*: An incorrect step, process, or data definition in a computer program.
- **Fault-prone component** – *reliability concept*: A component that will likely contain faults.
- **Vulnerability** - *security concept*: an instance of a [fault] in the specification, development, or configuration of software such that its execution can violate an [implicit or explicit] security policy.
- **Vulnerability-prone component** - *security concept*: a component that is likely to contain vulnerabilities.

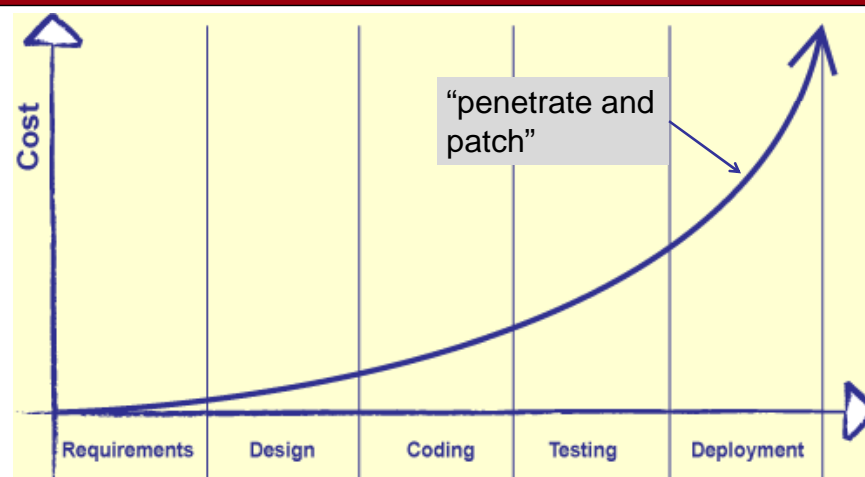
Realized Code Problem Definitions

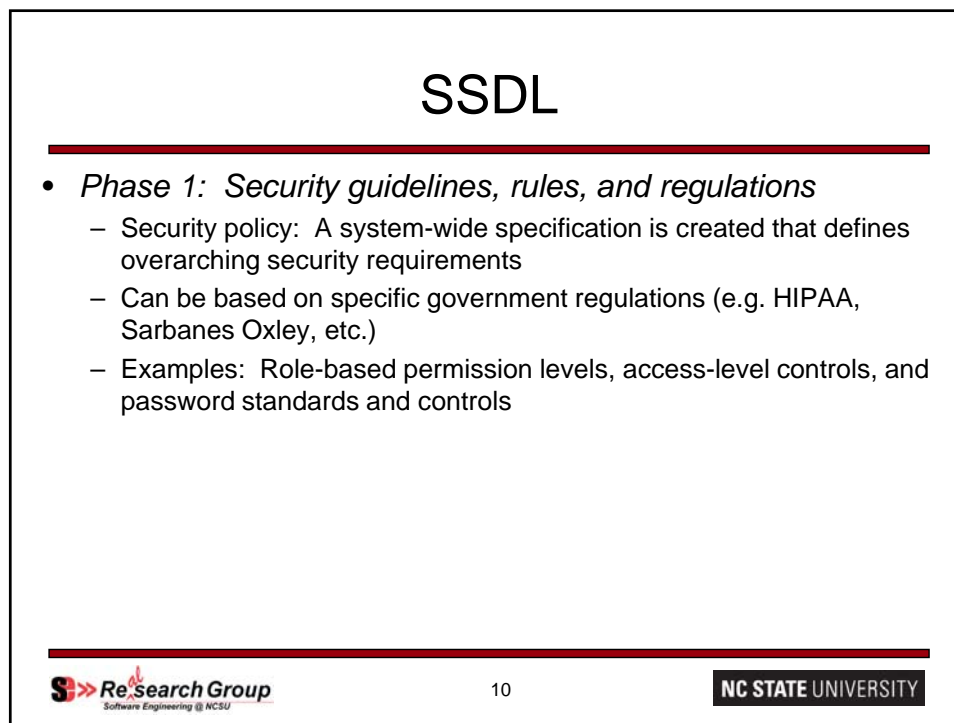
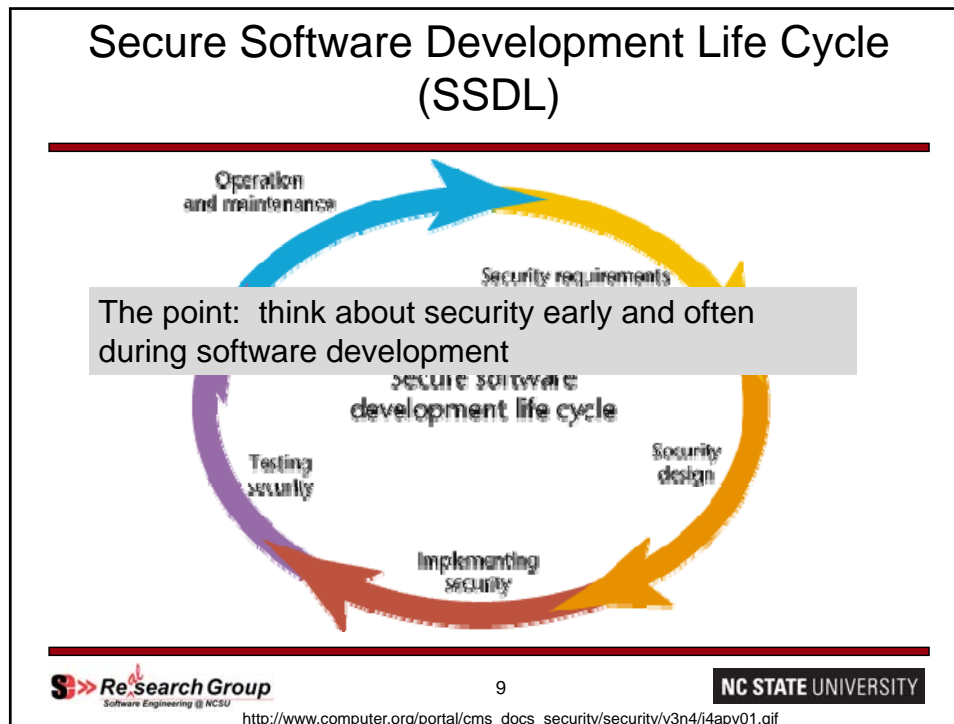
- **Failure** - *reliability concept*: the inability of a software system or component to perform its required functions within specified performance requirements.
- **Failure-prone component** – *reliability concept*: A component that will likely contain failures.
- **Attack** - *security concept*: the actions that cause a violation of security.
- **Attack-prone component** - *security concept*: a component with a high probability that the component will be exploited.

Vulnerability Types

- **design vulnerability** – a mistake in the design that precludes the program from operating securely, no matter how perfectly it is implemented by the coders
- **implementation vulnerabilities** – a mistake in the actual coding of the software that causes a security problem [design might be fine, code is the problem]

Cost of Change Curve





SSDL

- *Phase 2: Security requirements*
 - Misuse case/abuse case/attack use cases can be developed that show behavioral flows that are not allowed or are unauthorized.
 - Example security requirements [p. 62-63]
 - The application stores sensitive user information that must be protected for HIPAA compliance. To that end, strong encryption must be used to protect all sensitive user information wherever it is stored.
 - The application must remain available to legitimate users. Resource utilization by remote users must be monitored and limited to prevent or mitigate denial-of-service attacks.
 - The application supports multiple users with different levels of privilege. The application assigns users to multiple privilege levels and defines the actions each privilege level is authorized to perform. Mitigations for authorized bypass attacks need to be defined.
 - The application takes user input and uses SQL. SQL injection mitigations are a requirement.

SSDL

- *Phase 3: Architectural and design reviews/threat modeling.*
 - Software practitioners need a solid understanding of the product's architecture and design so they can devise better and more complete security strategies.
- *Phase 4: Secure coding guidelines.*
 - See Chapter 2 in book. (buffer overflow, input validation, etc.)
 - Static analysis tools can detect many implementation errors by scanning the source code or the binary executable.
 - Using the secure coding standards as baselines, testers can then develop test cases to verify that the standard is being followed.
 - There are also services where you can send your code and have a third party analyze it for defects/vulnerabilities.

SSDL

- *Phase 5: Black/gray/white box testing*
 - Gray box is hybrid black/white
 - Penetration testing – black box testing in which the tester specifically thinks like an attacker
- *Phase 6: Determining exploitability.*
 - A vulnerability's exploitability [attack proneness] is an important factor in gauging the risk it presents. Determining a vulnerability's exploitability involves weighing five factors:
 - The access or positioning required by the attacker to attempt exploitation
 - The level of access or privilege yielded by successful exploitation
 - The time or work factor required to exploit the vulnerability
 - The exploit's potential reliability
 - The repeatability of exploit attempts

SSDL

- Also:
 - Software should be deployed using security defaults
 - A software patch management process should be in place

Guidelines for Testers [p. 70-71]

- Define security/software development roles and responsibilities
- Understand the security regulations your system has to abide by, as applicable.
- Request a security policy if none exists.
- Request documented security requirements and/or attack use cases.
- Develop and execute test cases for adherence to umbrella security regulations if applicable. Develop and execute test cases for the security requirements/attack use cases.

Guidelines for Testers

- Request secure coding guidelines and train software developers and testers on them.
- Test for adherence to secure coding practices.
- Participate in threat modeling walkthroughs, and prioritize security tests.
- Understand and practice secure deployment practices.
- Maintain a secure system by having a patch management process in place, including evaluating exploitability.

Phase 2: Misuse/Abuse Case

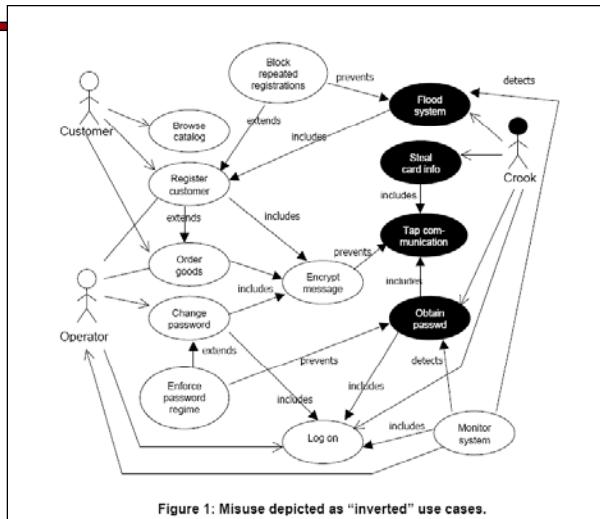


Figure 1: Misuse depicted as "inverted" use cases.