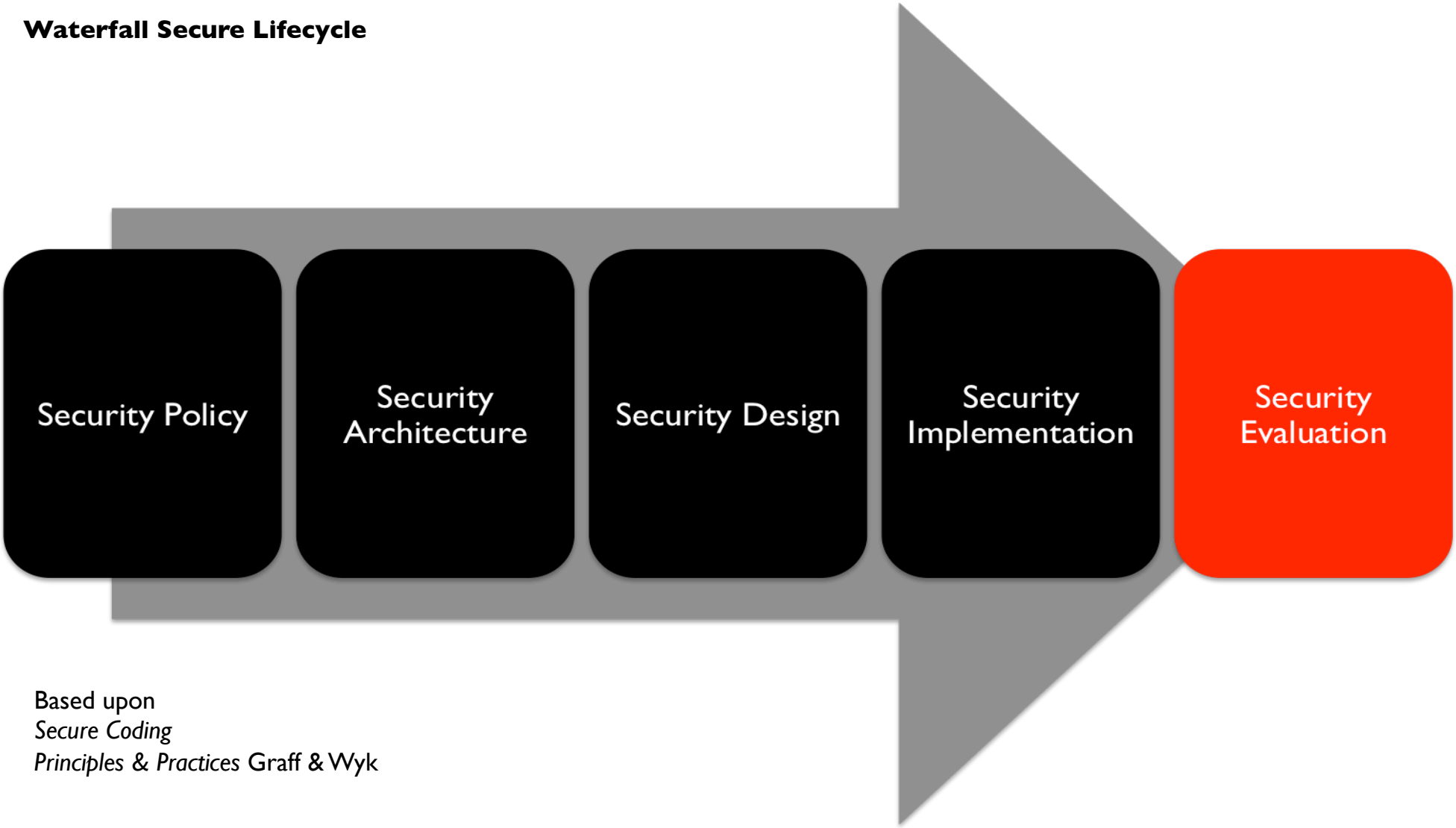
NWEN405: Security Engineering

Lecture 15

Secure Software Engineering: Security Evaluation

Engineering & Computer Science
Victoria University of Wellington
Dr Ian Welch (ian.welch@vuw.ac.nz)

Waterfall Secure Lifecycle



Based upon
Secure Coding
Principles & Practices Graff & Wyk

Security Evaluation

How do we know that we have met our security requirements?

Testing is the answer but how does security testing differ from ordinary testing?

How does testing your own code differ from testing someone else's code?

What do we do post-testing?

Identify, implement, and perform security tests

Purpose:	• Find security problems not found by implementation review. • Find security risks introduced by the operational environment. • Act as a defense-in-depth mechanism, catching failures in design, specification, or implementation.
Role:	Test Analyst
Frequency:	As necessary; generally multiple times per iteration.

- Aim of testing is to find bugs that perhaps are triggered due to interactions between components or environmental conditions.

True or False?

Users normally search for security bugs.

Attackers will act like ordinary users.

A hard-to-find bug is of low risk to a system.

Security tests focus on what should happen.

It is acceptable to mark some assumptions as too unlikely to hold.

True or False?

Users normally search for security bugs.

Few users do this, attackers take pleasure in finding corner cases.

True or False?

Attackers will act like ordinary users.

Normal test cases might neglect some boundary conditions because they are highly unlikely, attackers will test for these.

True or False?

A hard-to-find bug is of low risk to a system.

A small number of users might trigger a hard-to-find bug therefore restricting potential damage. Attackers will write a script to weaponise the bug and potentially deploy it against many systems (widespread damage).

True or False?

Security tests focus on what should happen.

Ordinary tests are usually positive, what the application should do. Security tests emphasize the negative, what shouldn't happen.

True or False?

It is acceptable to mark some assumptions as too unlikely to hold.

"An attacker should never be able to take control of the application" might be rejected as untestable from ordinary point-of-view but should be investigated from a security point-of-view.

Software testing v. Software Security Testing

A piece of software with 90% fewer bugs than another piece of software is usually more reliable under ordinary circumstances.

A piece of software with 90% fewer security-related bugs than another piece of software is not necessarily more secure if the 10% is easily exploitable.

Adapting Testing Strategies

Some strategies:

1. Functional security testing.

2. Risk-based security testing.

3. Penetration testing.

1. Functional-based testing

In simplest case, derive from requirements.

"When X happens, software should do Y".

Functional testing is positive in orientation.

Despite that, it can be used for security testing.

"Test that encrypting with key K and plaintext P generates ciphertext PK"

Here we are testing security functionality = positive test.

Can be black-box or white-box, apply to control flow, data flow etc.
All the standard testing considerations.

2. Risk-based testing

What about the negative tests?

Mitigations can used to derive extra security requirements for positive testing.

"Three failed logins cause user account to be disabled"

Note that result of testing is evidence of the presence of problems, not their absence.

Can also look at dependencies to test what happens in cases of failure

"Authentication server fails, everyone by superuser locked out of system (instead of allowing everyone into the system)"

Doesn't help with risks that you do not know about.

3. Penetration Testing

Often done on operational system.

Assemble outside experts, they use hacking toolkits to attack the system and find holes.

Most expensive way to fix your system (catch faults after complete system is deployed).

However, can be integrated into the lifecycle at earlier stage and carried out by software development team themselves.

3.1. Fuzz testing

What about the corner cases and negative testing.

No set of requirements that give hints as to the possible inputs.

One approach is fuzz testing (derived from software fault injection).

Randomly generate inputs that are invalid or unexpected and monitoring for crashes or assertions.

3.1. Fuzz testing

SPI Dynamics (http://www.spidynamics.com)

Fortify Software (http://www.immunitysec.com/products-canvas.shtml)

Shim (inject faults into web apps directly)
http://www.ieinspector.com/httpanalyzer/

3.2 Bug Finding Tools

FindBugs and other similar static analysis tools.

Looking for common programmer errors.

Purify is another tool for C and C++ -- control flow and data flow analysis.

3.3 Imitating the Hacker

Disassemblers and decompilers.

What information is disclosed that you don't know about?

Rootkits.

What can you do once installed? Can you change configuration files through a backdoor?

Attack creation.

Launch your own buffer overflow, XSS exploits, etc.

Time consuming!

3.3 Imitating the Hacker (Tools)

SATAN and SANTA

System penetration tools (the original), operating system is the main target.

METASPLOIT

Includes web applications, network services in general. Opensource with a commercial version available.

ICEPACK and its relations

“Commercial” tools for criminals.

What about external libraries, third-party code?

Penetration testing is a possibility.

Also trust-based approaches (described in Anderson).

Orange Book (DoD 1980s) – *generally discredited.*

Common Criteria (International project, 1990s onwards) – *more accepted, key idea is idea of profiles.*

Address reported security issues

Purpose:	• Ensure that identified security risks in an implementation are properly considered.
Role:	Designer
Frequency:	As required.

- Closes the loop on testing.
- Do something about it.
- Decide (using risk modeling) whether and when to fix a problem uncovered due to testing (or post-implementation) after release of the software.