Secure Software Development Software Security Touchpoints

Objectives

- Explain the concept of a Touchpoint
- List the Software Security Touchpoints (McGraw)
- Identify the most effective security practices to have within software development (McGraw)
- Define the acronym OWASP
- Explain the basic premise for CLASP
- List the 7 best practices for application security according to CLASP
- Explain the difference between the reactive and proactive approaches to software security
- Recognize an example of a bug and a flaw within software
- Explain the economic impact of various security activities

- Define the following terms from last week's reading
  - Single Loss Expectance

Value of Asset × Exposure factor

  - Annual Rate of Occurrence

Number of threats expected to manifest themselves in a year

  - Annual Loss Expectancy

Magnitude of risk in a given year.

- Define the following terms from last week's reading
  - Single Loss Expectance
    - An estimate of potential loss from a security breach.

SLE=Asset Value ($) X Exposure Factor (%)

  - Annual Rate of Occurrence
    - An expression of the number of incidents that can be expected in a year.
    - Very much guesswork goes into this entry
  - Annual Loss Expectancy
    - An indication of the magnitude of risk in a given year.
    - ALE = SLE X ARO

Discussion

- What are the best practices for software development?

Plan

Review

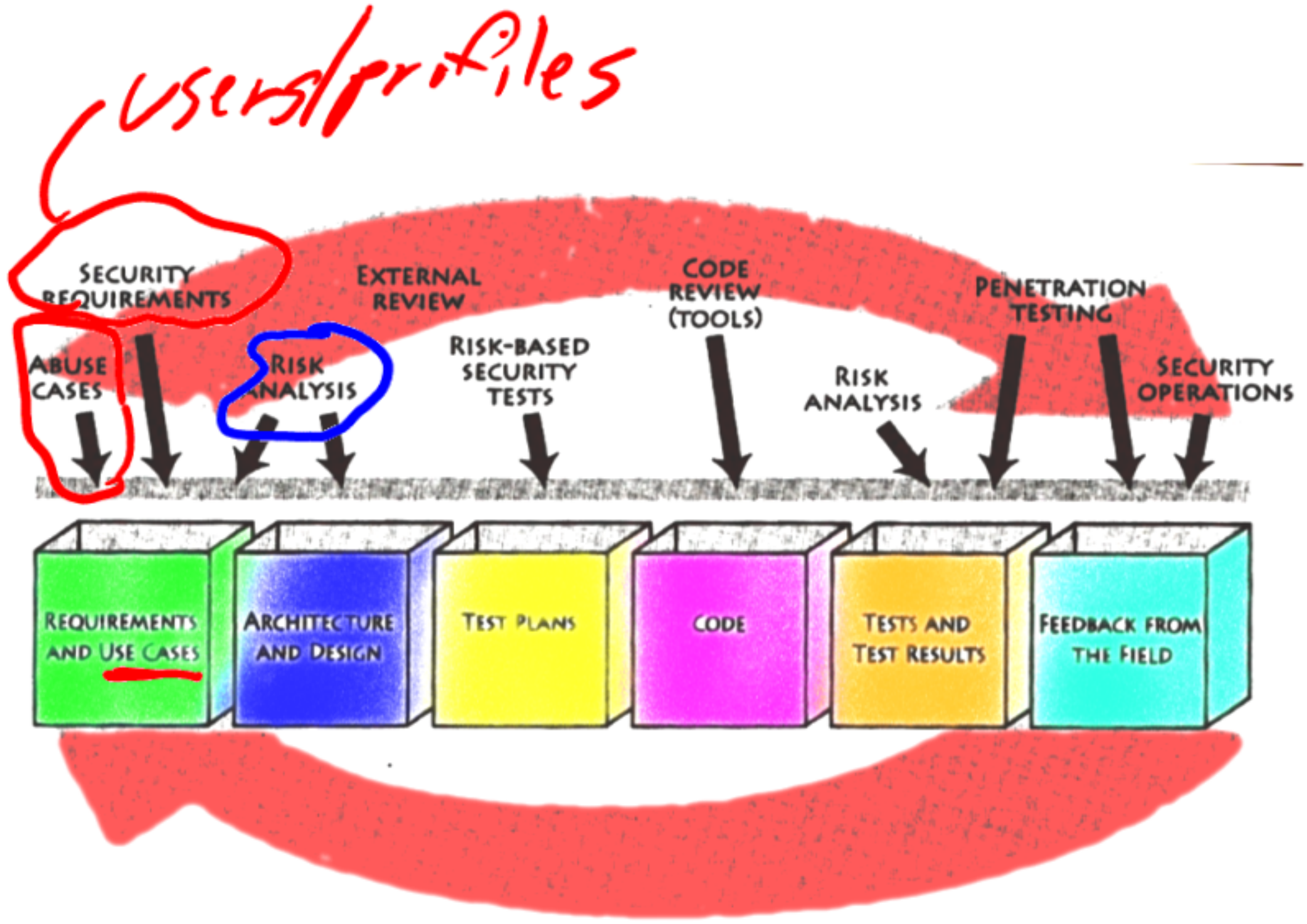
Design/Architecture

Testing => Test Planning & CI

What is a security touchpoint

- A best practice for ensuring software security
  - Based on the concepts of normal software development

Software Security Touchpoints



Software Security Touchpoints by effectiveness (McGraw)

1. Code Review
2. Architectural Risk Analysis
3. Penetration Testing
4. Risk-Based Security testing
5. Abuse Cases
6. Security Requirements
7. Security Operations

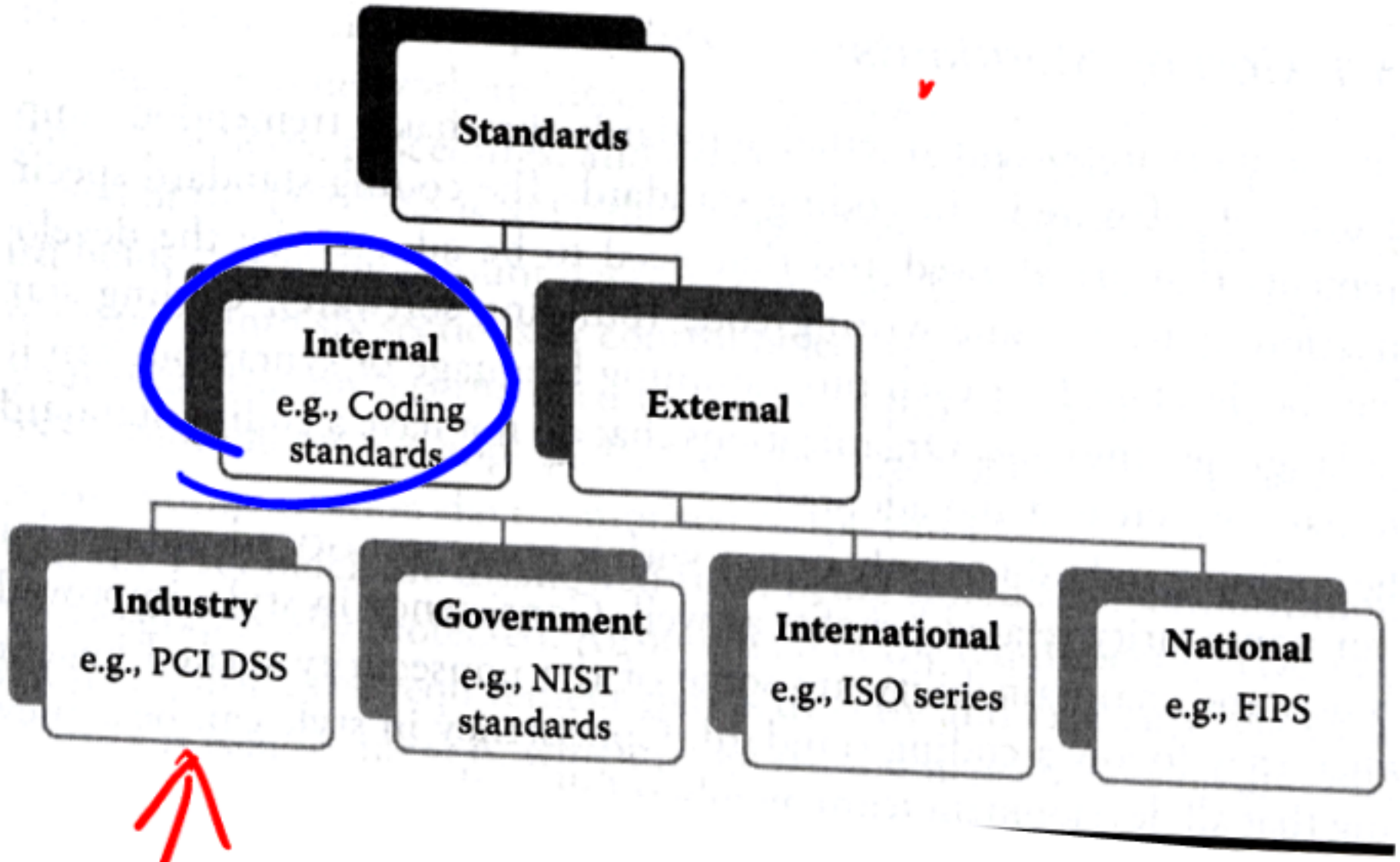
Where do we get our security approaches?

- Attack
- Learn the vector
- Fix Problem
- Repeat

Standards

organizational issues

Sources of standards



Domain Specific

Trivia Time

- According to OWASP in 2013, the top web application vulnerability was
  a. Injection flaws
  b. Insecure Communications
  c. Malicious file execution
  d. Cross Site Scripting (XSS)
  e. Buffer overflow
  f. Information leakage and improper error management
  g. What'chu **talkin**' 'bout, **Willis?**

A1-Injection	Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2-Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.
A3-Cross-Site Scripting (XSS)	XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A4-Insecure Direct Object References	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.
A5-Security Misconfiguration	Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.
A6-Sensitive Data Exposure	Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
A7-Missing Function Level Access Control	Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.
A8-Cross-Site Request Forgery (CSRF)	A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.
A9-Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.
A10-Unvalidated Redirects and Forwards	Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

OWASP 2010 versus 2007

OWASP Top 10 – 2007 (Previous)	OWASP Top 10 – 2010 (New)
A2 – Injection Flaws	A1 – Injection
A1 – Cross Site Scripting (XSS)	A2 – Cross-Site Scripting (XSS)
A7 – Broken Authentication and Session Management	A3 – Broken Authentication and Session Management
A4 – Insecure Direct Object Reference	A4 – Insecure Direct Object References
A5 – Cross Site Request Forgery (CSRF)	A5 – Cross-Site Request Forgery (CSRF)
<was T10 2004 A10 – Insecure Configuration Management>	A6 – Security Misconfiguration (NEW)
A8 – Insecure Cryptographic Storage	A7 – Insecure Cryptographic Storage
A10 – Failure to Restrict URL Access	A8 – Failure to Restrict URL Access
A9 – Insecure Communications	A9 – Insufficient Transport Layer Protection
<not in T10 2007>	A10 – Unvalidated Redirects and Forwards (NEW)
A3 – Malicious File Execution	<dropped from T10 2010>
A6 – Information Leakage and Improper Error Handling	<dropped from T10 2010>

Buffer overflow

METHODOLOGY

Our methodology for the Top 10 2007 was simple: take the MITRE Vulnerability Trends for 2006, and distill the Top 10 *web application security* issues. The ranked results are as follows:

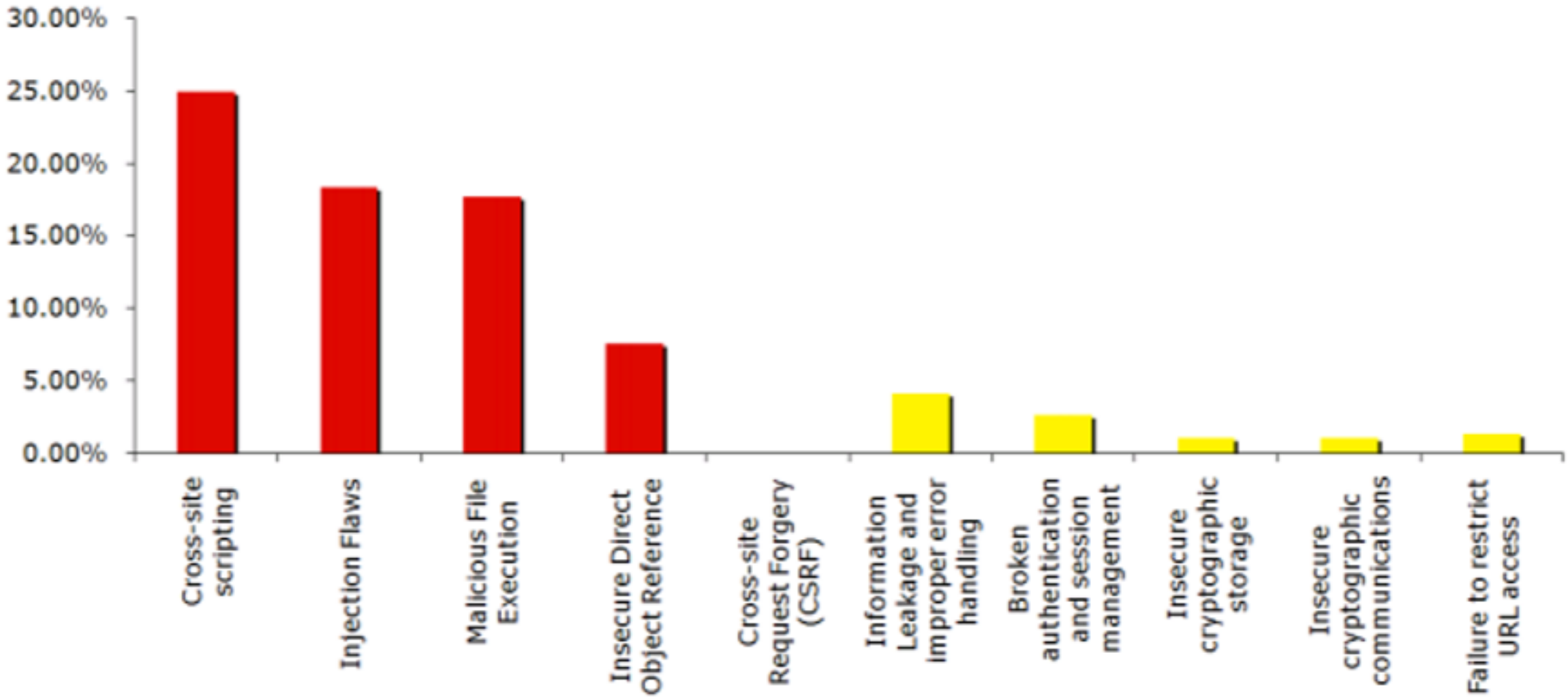


Figure 2: MITRE data on Top 10 web application vulnerabilities for 2006

- Source: OWASP

What is CLASP

- The Comprehensive Lightweight Application Security Process
  - Supports the incorporation of security processes into each phase of the software development lifecycle

CLASP Practices

- Institute awareness programs
- Perform application assessments
- Capture security requirements
- Implement secure development practices
- Build vulnerability remediation procedures
- Define and monitor metrics
- Publish operational security guidelines

CLASP has 30 activities

Misuse cases
VS Functionality

All listed in
text

Reactive versus proactive security

Reactive
something goes wrong
How do we make sure
Secure

Proactive
How do we build this
Secure?

Bugs versus flaws again

Bugs	Flaws
Buffer overflow: stack smashing	Method over-riding problems (subclass issues)
Buffer overflow: one-stage attacks	Compartmentalization problems in design
Buffer overflow: string format attacks	Privileged block protection failure (DoPrivilege())
Race conditions: TOCTOU	Error-handling problems (fails open)
Unsafe environment variables	Type safety confusion error
Unsafe system calls (fork(), exec(), system())	Insecure audit log design
Incorrect input validation (black list vs. white list)	Broken or illogical access control (role- based access control [RBAC] over tiers)
	Signing too much code

Tr ivia

- What is the relative percentage of bugs versus flaws

a. 20% bugs, 80% flaws

b. 30% bugs, 70% flaws

c. 40% bugs, 60% flaws

d. 50% bugs, 50% flaws

e. 60% bugs, 40% flaws

f. 70% bugs, 30% flaws

g. 80% bugs, 20% flaws

Flaw Example

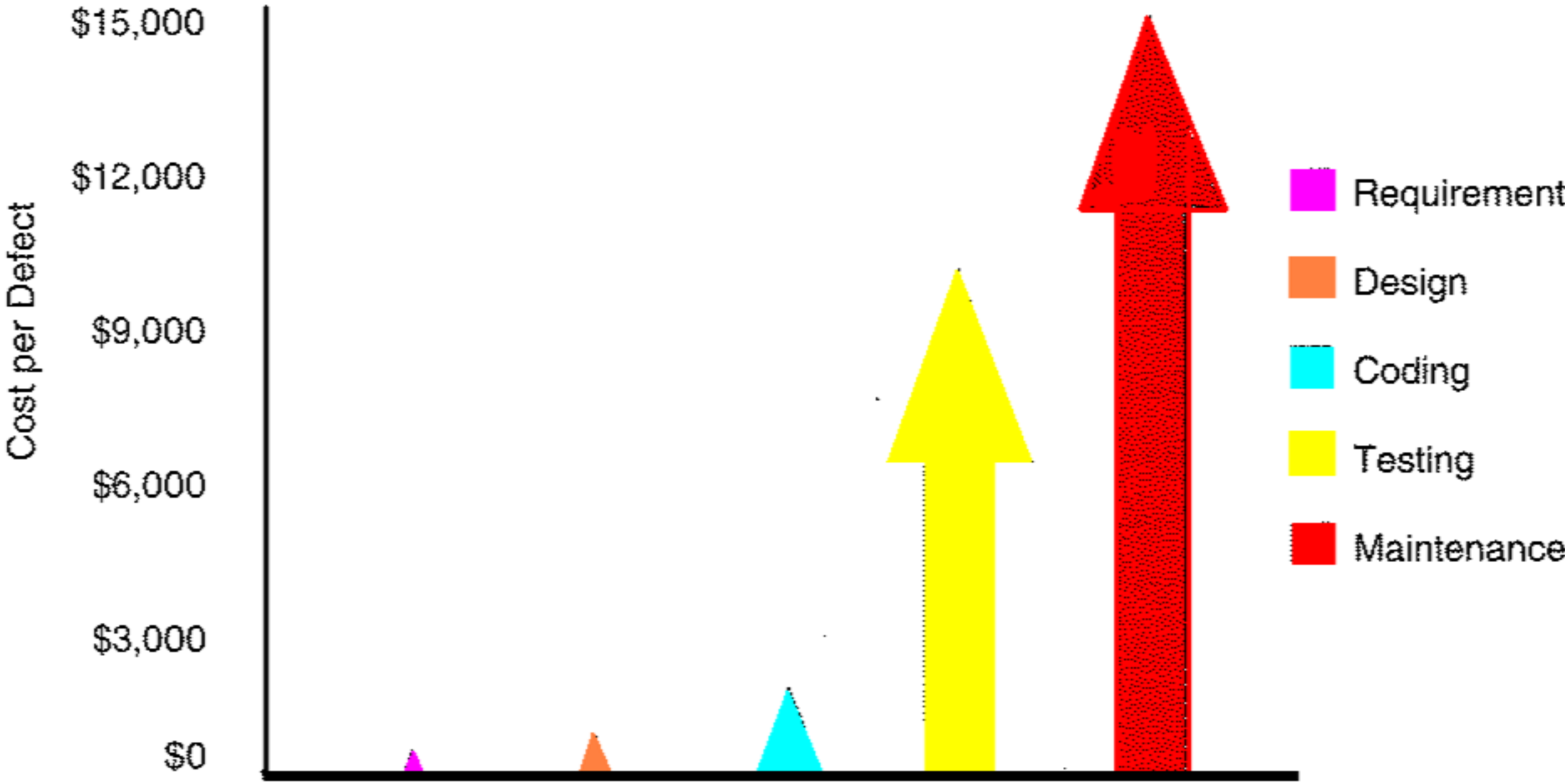
```
public class Pass_controlflow_good
{
        public static void main (String[]args){
        try {
        BufferedReader read  new BufferedReader(new FileReader("Passw
ords.txt"));
        String adminPass = read.readLine();
        if (args[0].equals(adminPass))
                System.out.println("Access Granted");
        else
                System.out.println("Your password is not valid, please
reenter it.");
                }
                catch(IOException e){
                System.out.println("io error");}
                return;
                }
}
```

Bug Example

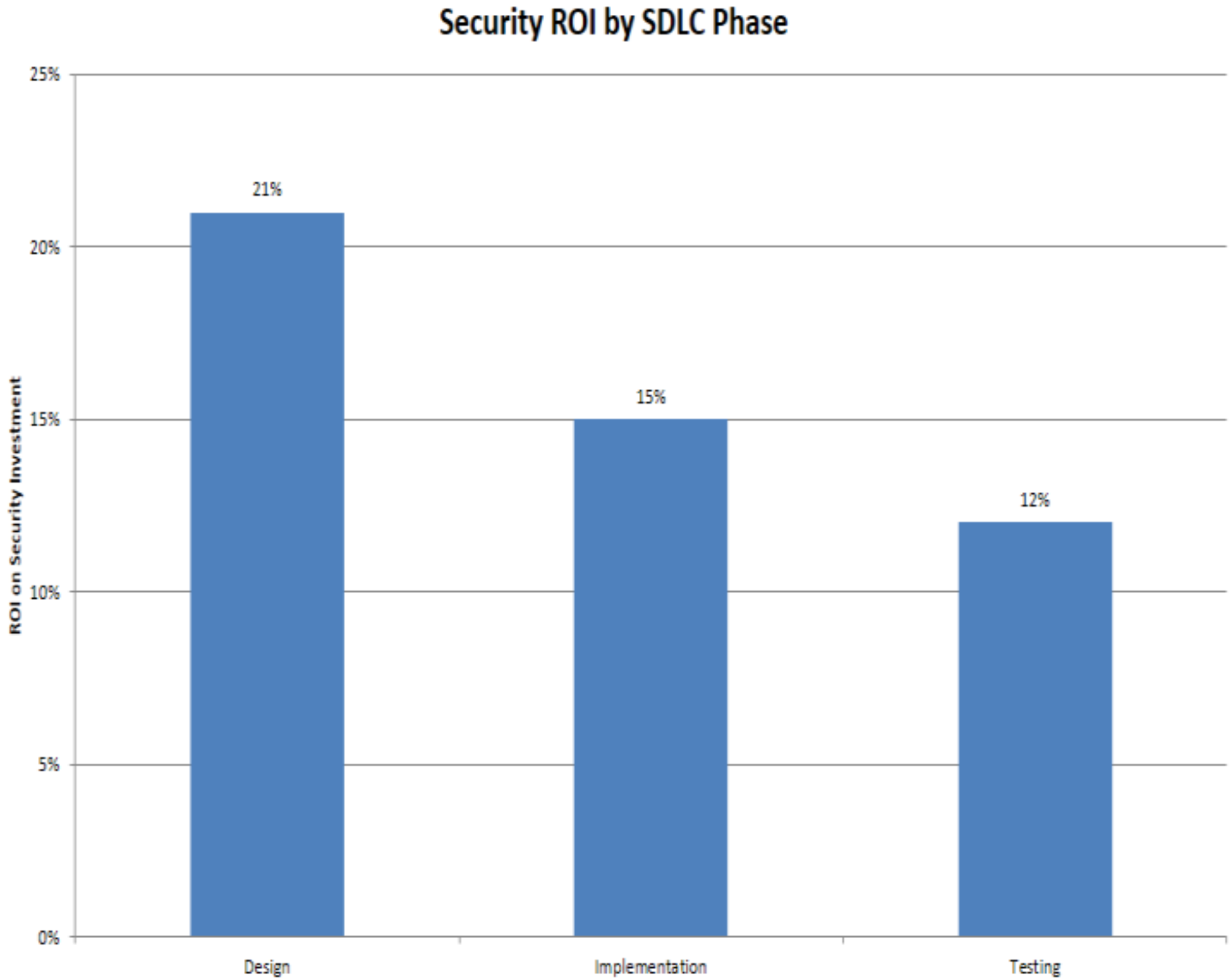
```
int get_buff(char *user_input) {
  char buff[4];
  int length = strlen(user_input)+1;
  memcpy(buff, user_input, length);
  return length;
}

int main(int argc, char*argv[]){
  get_buff(argv[1]);
  return 0;
}
```

Cost of Fixing Defects at Each Stage
of Software Development



Return on Investment for Security Operations
Secure Business Quarterly, Q4 2001



Code Review With a tool

- Artifact(s)
  - Source code
- Very effective at finding implementation bugs
- Extremely important to do this in C / C++ Development

Code Review With a tool

- Highly efficient with modern tools

Risk Analysis

- Artifact(s)
  - Design
  - Specification

- Identifies problems with core design of the software
  - Poor protection of critical data

Architectural

  - Authentication failures
  - Poor design practices

Penetration Testing

- Artifact(s)
  - Deployed System
- Can be useful at verifying architectural risk mitigations
- Gives a good understanding of the software in its deployed environment
  - May be more of a badness-o-meter than an actual security metric

Risk Based Security Testing

- Artifact(s)
  - Units
  - Completed System

- Tests that the software can handle abuse cases properly

- Artifact(s)
  - Requirements
  - Specification

Abuse Cases

- Puts the developers into the mindset of an attacker
- Describe the systems behavior when under attack

Security requirements

- Artifact(s)
  - Requirements

- Good security requirements convey required functional security as well as behaviors of the system
  - Generally considered to be constrains on functional requirements in order to achieve security goals