Web Security Essentials for Universities

James Walden
Northern Kentucky University
waldenj@nku.edu

Is your web site secure?

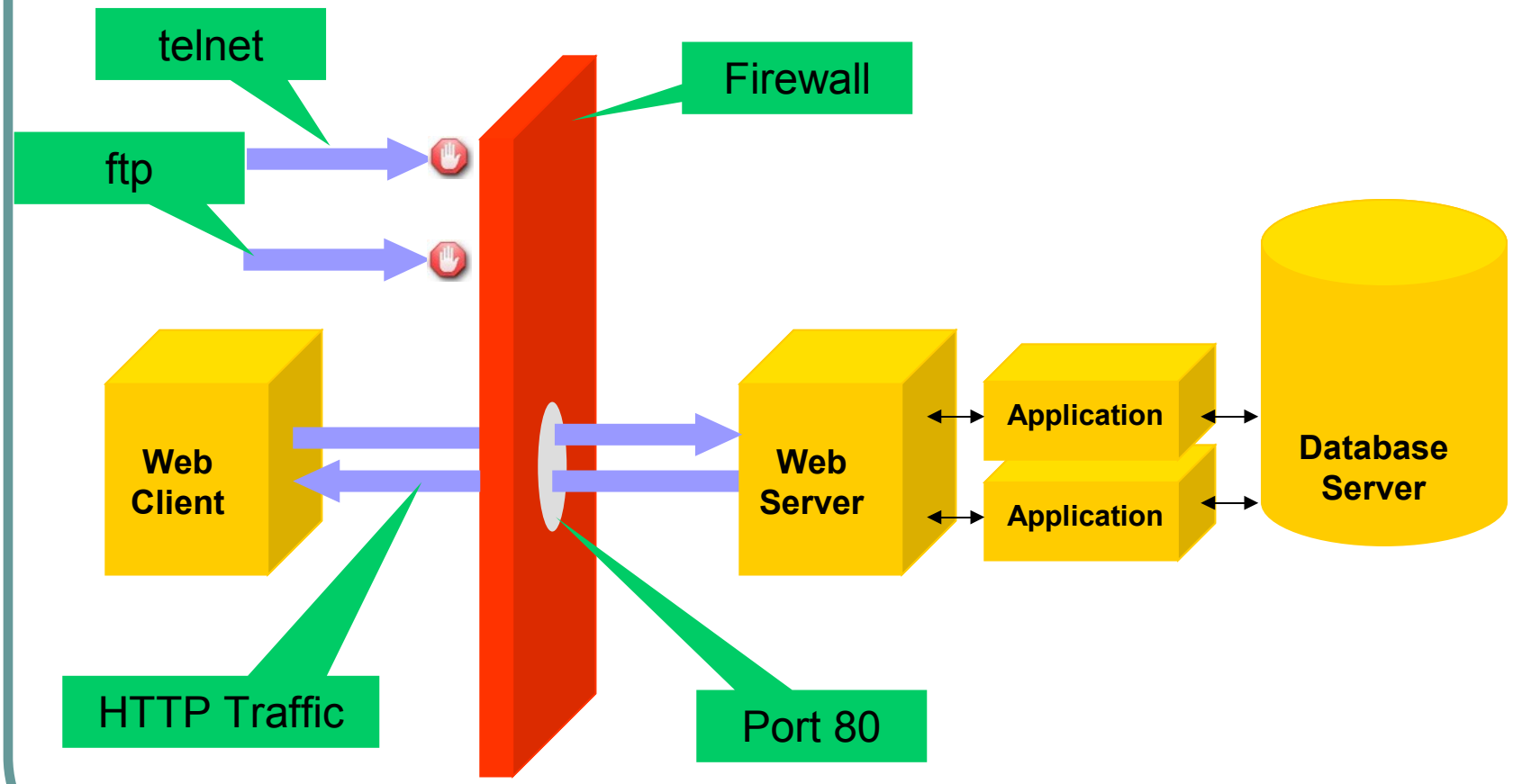
Is your web site secure?

Is your web site secure?

Yes, we deployed SSL, firewall, etc.

- Does SSL protect all communications?
- What about stored data?
- What about injection attacks and XSS?

Firewalls don’t protect web apps



Is your web site secure?

Yes, we have logs of blocked attacks.

- Better, you have some real evidence.
- Did you log non-blocked requests too?

Is your web site secure?

Yes, we have identified and categorized our assets, have a SDLC, and monitor network, host, and application-based logs.

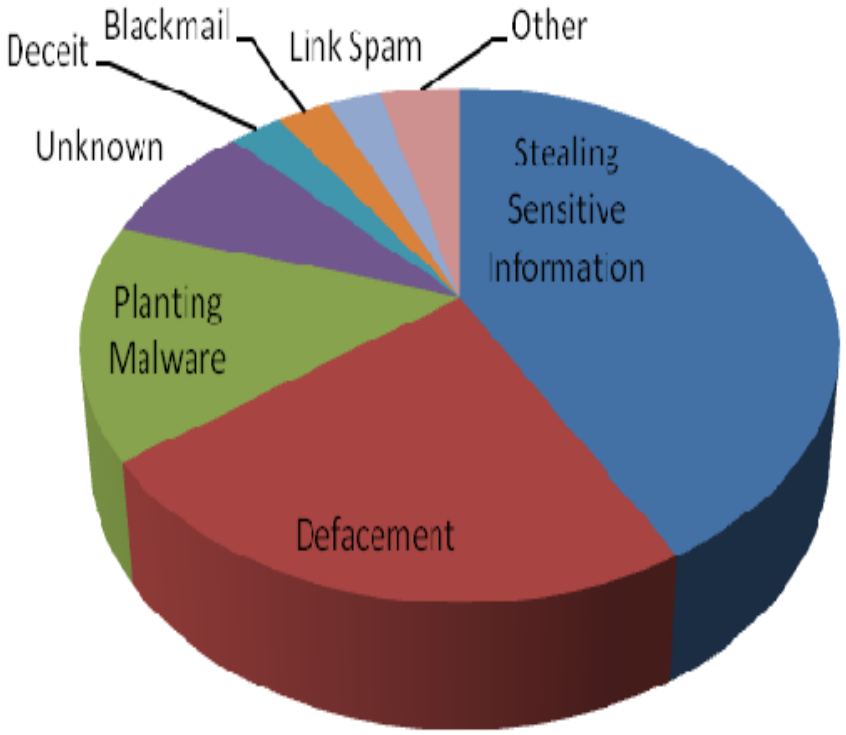
- Threat Modeling
- Secure Development LifeCycle
  - Risk analysis
  - Secure design
  - Code reviews
  - Security testing
- Correlate logs for multi-perspective picture.

Web Security Statistics

- SQL Injection is primary means of spreading malware.
  - Inject JavaScript into DB instead of reading your data out.
  - Malicious JavaScript IFRAMEs distribute malware.
  - SQL injection attacks against 10,000s of hosts are common
- Sophos found one infected web page every 4.5 seconds in 2008.
- Websense: 77% of sites hosting malware are legitimate sites that have been hacked.
- Websense: 61% of 100 most popular web sites served malware at some point in 2009.

Reasons for Attacking Web Apps

Attack Goal	%
Stealing Sensitive Information	42%
Defacement	23%
Planting Malware	15%
Unknown	8%
Deceit	3%
Blackmail	3%
Link Spam	3%
Worm	1%
Phishing	1%
Information Warfare	1%

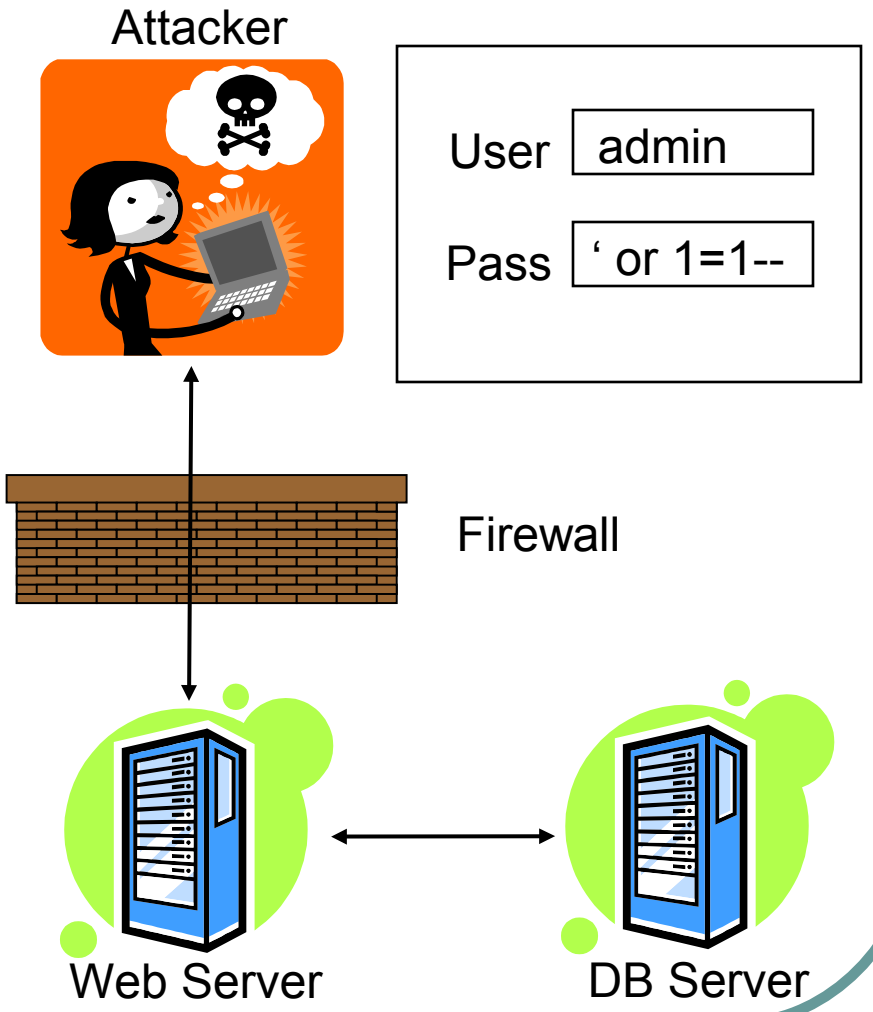


OWASP Top 10 Vulnerabilities

1. Cross-Site Scripting (XSS)
2. Injection Flaws (SQL and others)
3. Remote File Inclusion
4. Insecure Direct Object Reference
5. Cross-Site Request Forgery (XSRF)
6. Information Leakage
7. Broken Authentication or Session Management
8. Insecure Storage
9. Insecure Communications
10. Failure to Restrict URL Access

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.



SQL Injection in PHP

```
$link = mysql_connect($DB_HOST, $DB_USERNAME,
    $DB_PASSWORD) or die ("Couldn't connect: " .
    mysql_error());
mysql_select_db($DB_DATABASE);
$query = "select count(*) from users where username =
    '$username' and password = '$password'";
$result = mysql_query($query);
```

SQL Injection Attack #1

Unauthorized Access Attempt:

`password =` **' or 1=1 --**

SQL statement becomes:

select count(*) from *users* **where** *username* = *'user'* **and** *password* = *''* **or 1=1 --**

Checks if password is empty OR 1=1, which is always true, permitting access.

SQL Injection Attack #2

Database Modification Attack:

`password =` foo'; **delete from table** *users* **where** *username* **like** '%

DB executes ***two*** SQL statements:

select count(*) from *users* **where** *username* = *'user'* **and** *password* = *'foo'*

delete from table *users* **where** *username* **like** '%'

SQL Injection Demo

Mitigating SQL Injection

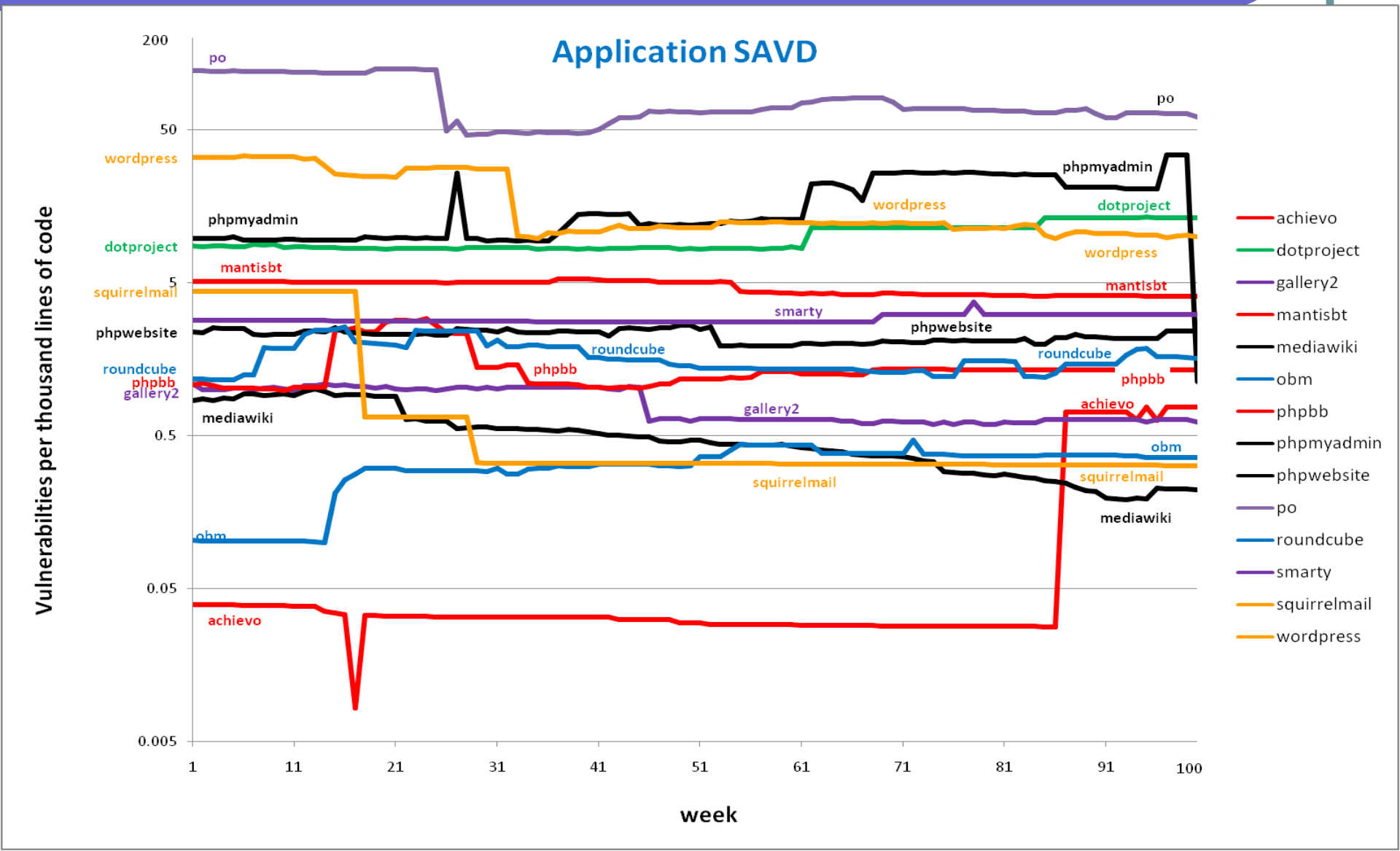
Partially Effective Mitigations

- Blacklists
- Stored Procedures
- Whitelists

Effective Mitigations

- Prepared Queries

Variation between Web Apps



How do I protect my systems?

1. Threat modeling: identify what to protect
2. Segregate data based on threat model
3. Install a Web Application Firewall
4. Fix critical applications
5. Plan for incident response
6. Monitor security
7. Improve your development process

Threat Modeling

1. Identify System Assets.
   - System resources that an adversary might attempt to access, modify, or steal
   - Ex: personally-identifiable info, grades, availability, reputation
2. Identify Entry Points.
   - Data or control transfers between systems
   - Ex: open ports, web forms, logins, file transfers
3. Determine Trust Levels.
   - Privileges external entities have to legitimately use system resources

Segregating data

Critical data on its own web and db servers

- May need multiple sets of servers
- If cannot afford servers, use VMs to reduce costs

Separate users on single server

- suEXEC, suPHP, fastCGI run scripts as users
- Limits access to data owned by user
- Leaves audit trail, letting you know which user's scripts were source of compromise

Install a Web Application Firewall

What's a WAF?

- Intrusion detection and prevention for HTTP
- Just-In-Time Patching: it's faster to add a new rule for WAF than to fix + redeploy app

Deployment

- Expensive: buy a hardware WAF box
  - Need to find point in network to deploy
- Cheap: install mod_security in Apache
  - Install on existing web server

Install mod_security

1. Download binary package from http://www.modsecurity.org/download/
2. Install dependencies
   - yum install httpd-devel libxml2
3. Install with yum, apt-get, Win installer
4. Load mod_security in Apache config
   - LoadFile /usr/lib/libxml2.so
   - LoadModule mod_security2.so

Configuring mod_security

- Use default Core Rules v2.0
- Configure in IDS mode first
  - Watch logs for a month or so and
  - Check if rules are triggered by normal traffic
  - Remove offending rules
  - Add rules specific to your applications
- Then turn on blocking of requests

ID Flaws in Critical Applications

Types of flaws to search for

- OWASP Top 10
- Logic flaws: student can change grades

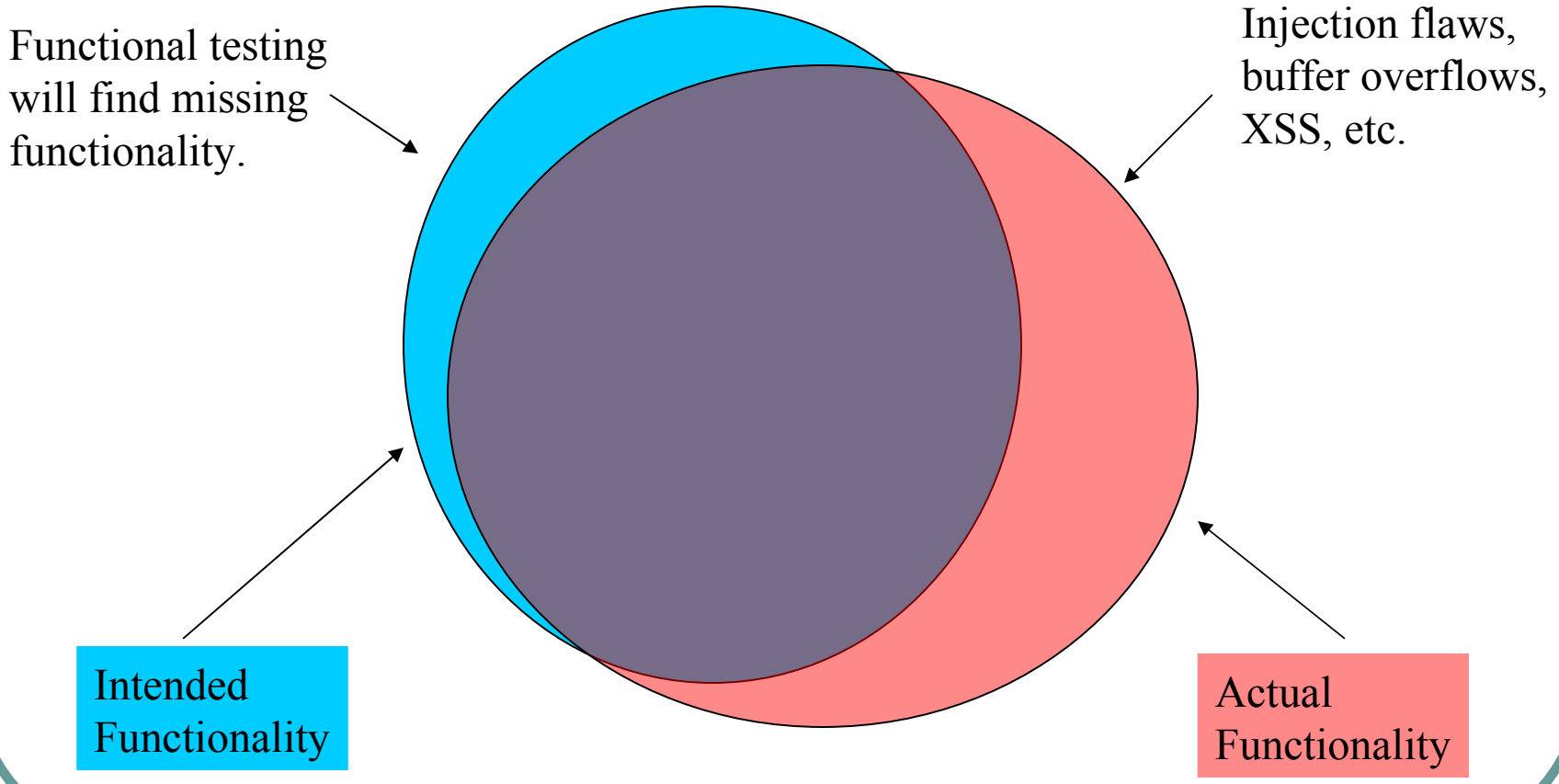
Manual identification techniques

- Code reviews
- Penetration testing (with proxy)

Automatic identification techniques

- Dynamic analysis
- Static analysis

Security vs. Functional Testing



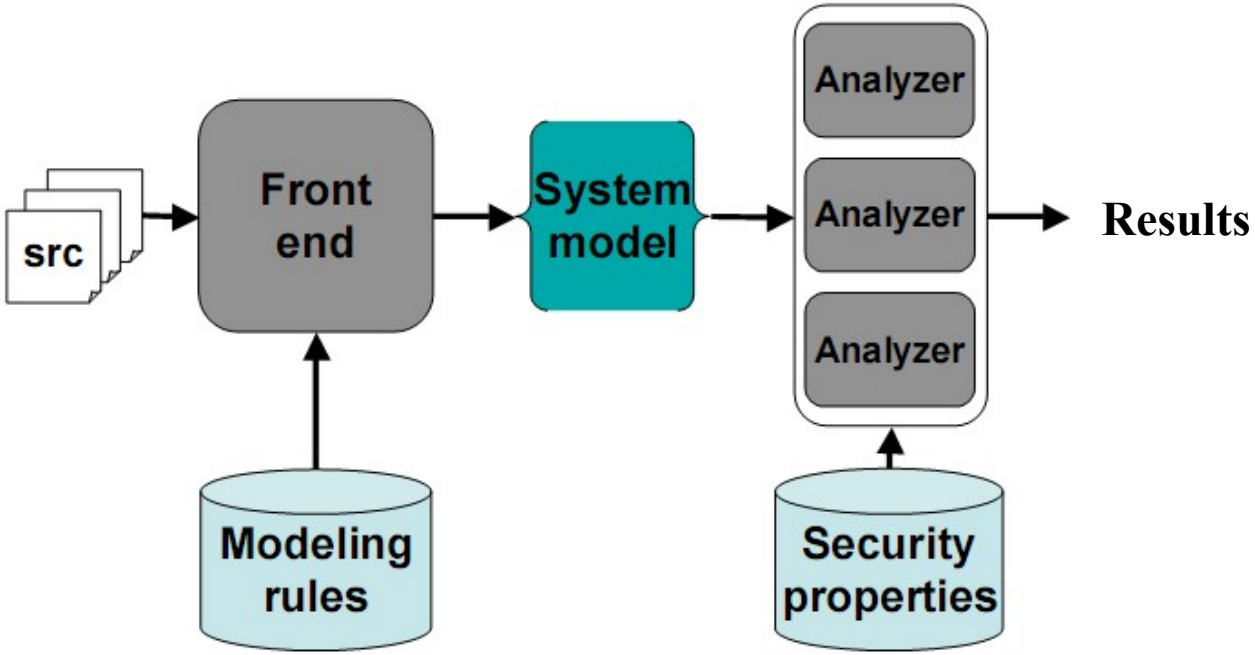
Code Reviews vs. Testing

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

Automated assistance for code reviews

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



Fix Flaws in Critical Applications

Prioritize identified flaws

- Impact (consequences of exploitation)
- Cost to fix (time, people)

Identify a solution

- Ex: change from query to prepared query

Generalize solution

- Create coding standards
- Ex: always use prepared query
- Ex: validate all input

Fixing Flaws Resources

- OWASP Top 10
  http://www.owasp.org/index.php/Top_10_2007
- SQL Injection Prevention Cheat Sheet
  http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
- XSS Prevention Cheat Sheet
  http://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet
- OWASP Code Review Guide
- OWASP Testing Guide
- OWASP Live CD
  http://appseclive.org/node/45

Incident Response Plan

Plan for getting hacked; it will happen.

- Assign appropriate people to CSIRT.

Goals for incident response may include:

1. Determining if a security breach occurred
2. Containing intrusion to prevent further damage
3. Recovering systems and data
4. Preventing future intrusions of same kind
5. Investigating and/or prosecuting intrusion
6. Preventing public knowledge of incident

Incident Response Plan

Availability may be most important goal

- Plan for quick recovery
- Virtual Machines make this easy
- Save VM checkpoint, then click to recover

Need to avoid repeats

- Checkpoint or backup system still has flaw
- Save copies of logs off the server

Monitor your Security

Maintain logs

- Enable logging in web server
- Setup log rotation
- Keep backups of logs

Read your logs daily

- Server, Apache, and mod_security logs
- Alerts and summaries: swatch and logwatch

Conclusions

- Attackers do want your web server.
- You need to protect it:
  - Identify assets, entry points, trust levels
  - Segregate data on diff servers by value
  - Install a WAF and watch your logs
- You will get hacked, so
  - Plan for incident response
  - Plan for quick recovery