

Essential IT Security Testing

Application Security Testing for
System Testers

By Andrew Muller
Director of Ionize

Who is this guy?

- IT Security consultant to the stars
- Member of OWASP
- Member of IT-012-04 Security Techniques
- Member of IT-015-26 Software Testing

What's he talking about?

- Background
- Issues
- Methodologies
- Tools
- Common vulnerabilities



Why security testing?



AS/NZS ISO/IEC 27001:2006
Information technology – Security techniques
– Information security management systems
– Requirements

Some of us test to satisfy compliance requirements, the most popular being ISO 27001, with the ISM and PCI applying to specific industry sectors.



**Payment Card Industry (PCI)
Data Security Standard**

Requirements and Security Assessment Procedures

Version 2.0
October 2010

Why security testing?



Australian Government

Office of the Australian Information Commissioner

Protecting information rights – advancing information policy

About us >

Information law

Publications and resources >

News and events >

Freedom of information >

Privacy >

Information policy >

You are here: [Home](#) > [Publications and resources](#) > [Reports](#)

Own Motion Investigation Report

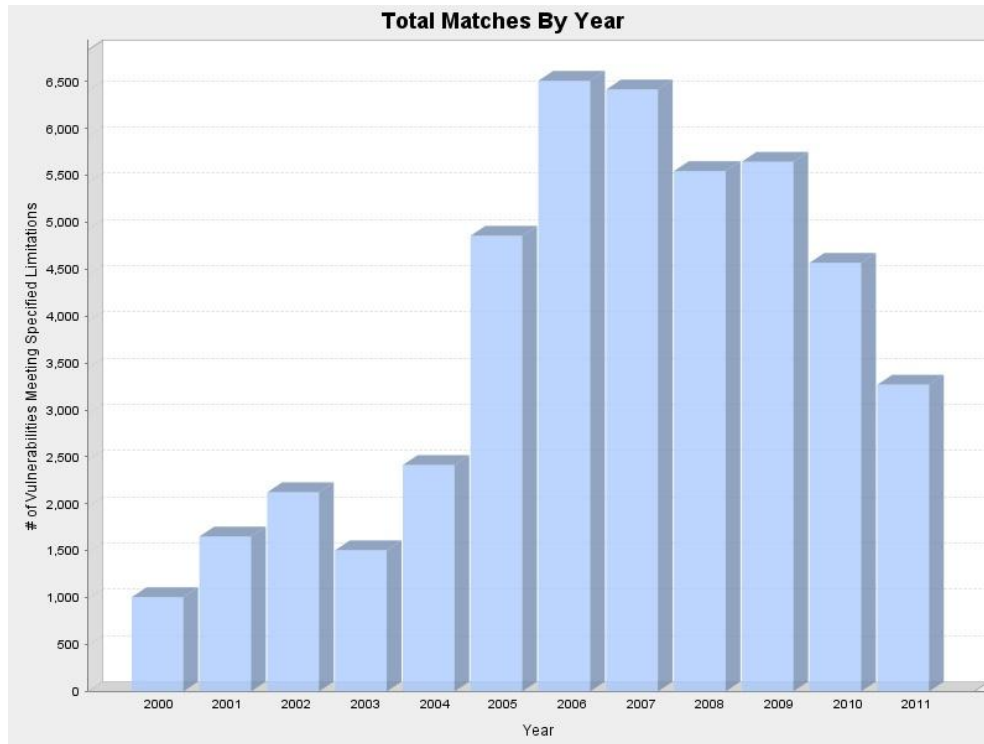
Sony PlayStation Network / Qriocity

Background

On 27 April 2011, the Australian Privacy Commissioner cc
1988 (Cth)[1] following media reports that an unauthorise

Some of us test so as to not run foul of the law.

Why security testing?



This graph indicates the number of vulnerabilities, but not the number of breaches resulting from each of these vulnerabilities.

<http://web.nvd.nist.gov/view/vuln/statistics>

**Some of us test to make software better.
Long history of devastating software flaws.
Number of vulnerabilities discovered each year is growing
Plenty of high profile disclosures. Why?
Are there more or smarter bad guys?**

Why is this happening?

- Functional requirements are **KING**
- Security is a non-functional requirement
 - and diminishes usability
- Some of the dysfunction is cultural

Focus on positive test cases

Believe that our own guys are OK

Most compromises are due to insiders

But what about higher classified enclaves with the network?

What about unintentional DoS?

We are highly connected with highly complex software, based on aging and unsecure technologies: HTTP, SMTP, DNS

We can still relate security to key business requirements: Availability, Reliability

Perception of Security



Lack of security knowledge by developers

Lack of development knowledge by security

Lack of knowledge of both by the business

Lack of communication

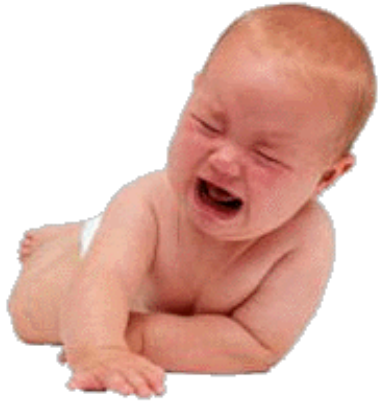
“Security are all sheriffs”

Perception of Developers



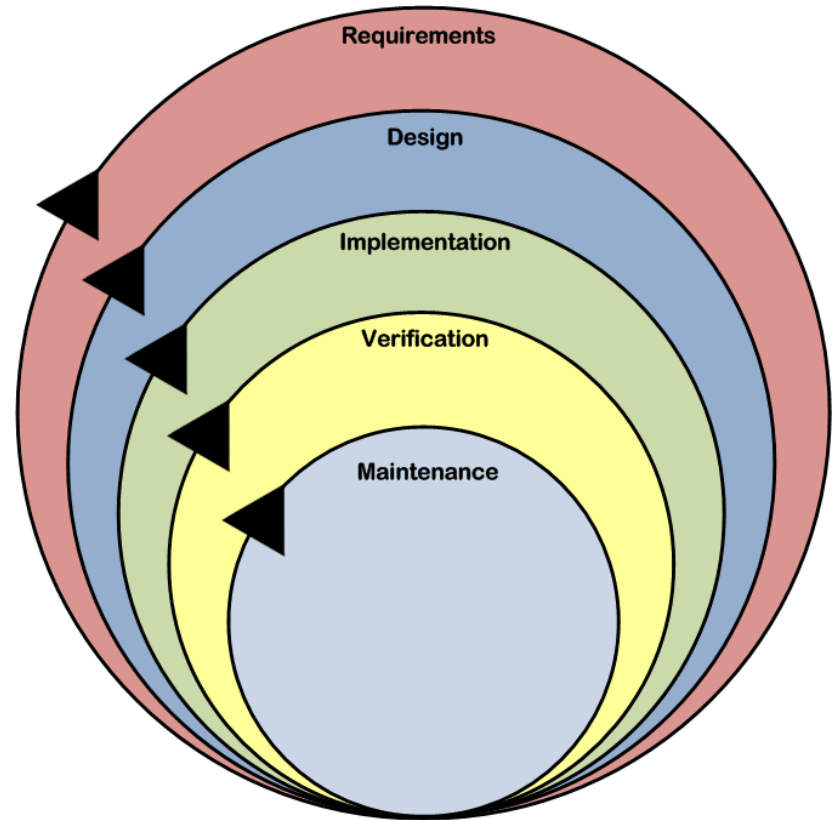
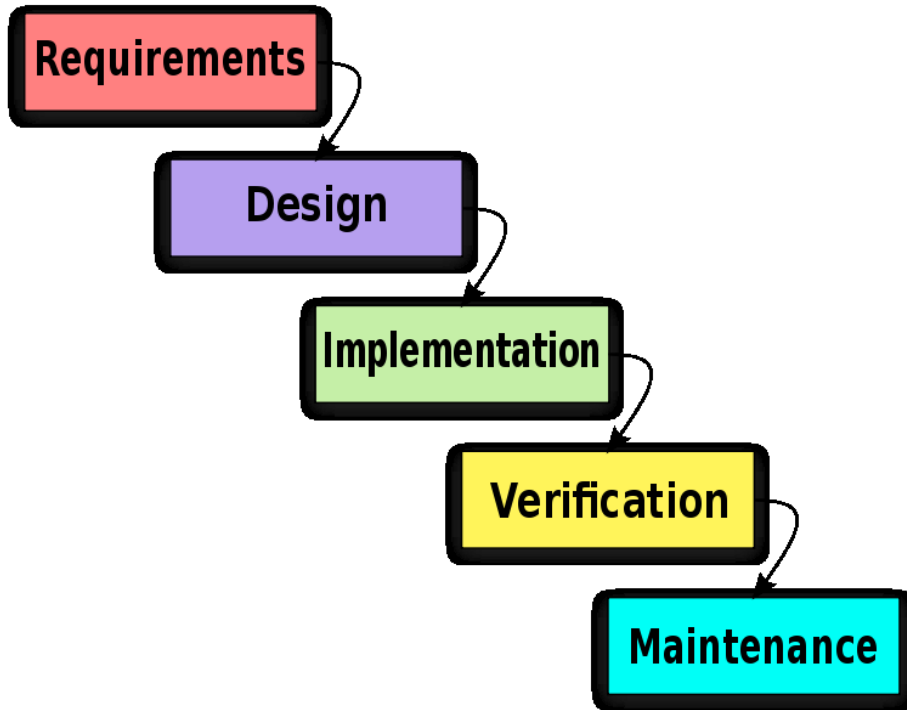
**Lack of communication
“Developers are all cowboys”**

Perception of Business



**Lack of communication
“Business don’t know what they
want and are unhappy with
everything”**

How do we fix it?



An SDLC is a process to produce software or systems that satisfy requirements

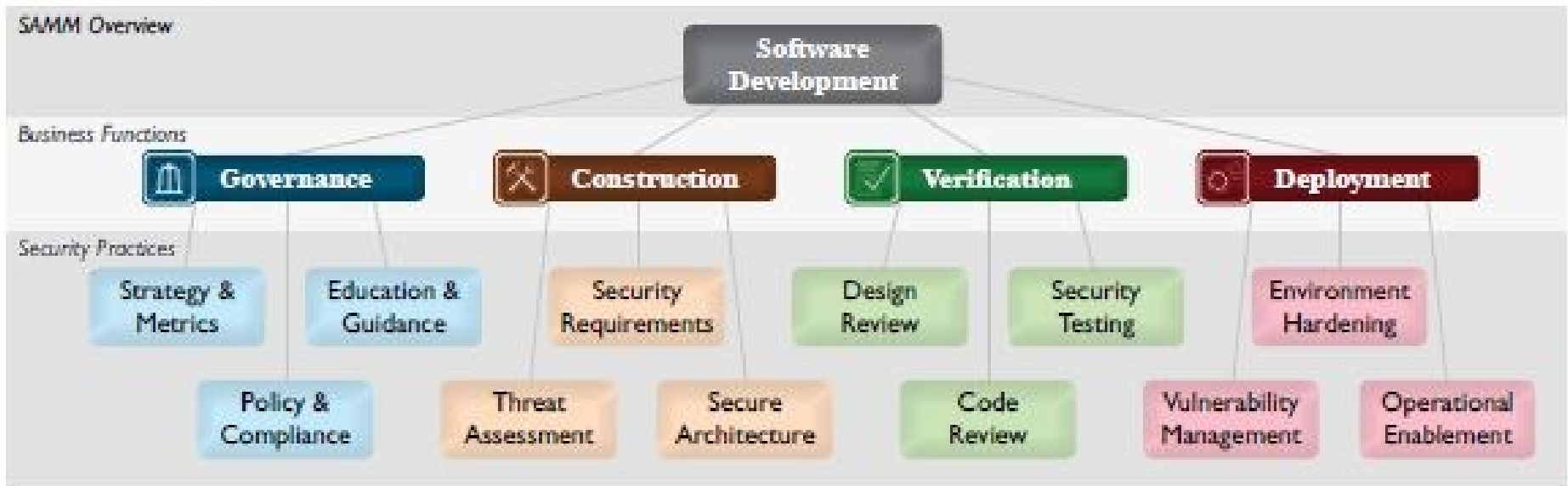
How software is made is critical to understanding how to test it and how to fix it

Where does security fit?



OpenSAMMM

- Software Assurance Maturity Model



- <http://www.opensamm.org>

Okay, well now we know where security testing might fit, how do we undertake the testing?

Security testing methodologies

- Most of us have a robust test cycle
- System and integration testing is methodical
- Security testing hasn't been so rigorous
- Security is just another set of test cases

Security testing methodologies

- What do we want from a methodology?
- Accountability
 - What was tested? Where are the results?
- Repeatability
 - Will the next test be the same as this one?
- Thoroughness
 - How do I know everything has been tested?



Secure OS and software configuration checklists for:

- **Secure configuration**
- **Security vulnerabilities**
- **Security weaknesses**

Supports Security Content Automation Protocol (SCAP) to automate enumeration of insecure software and platform configuration.

National Checklist Program (<http://checklists.nist.gov>)

OSSTMM 3

The Open Source Security Testing Methodology Manual
Contemporary Security Testing and Analysis



Created by Pete Herzog
Developed by ISECOM

ISECOM
INSTITUTE FOR SECURITY AND OPEN METHODOLOGIES

A complete set of test cases for all areas and levels of security:

- Human
- Physical
- Wireless
- Telecommunications
- Digital network

From policy to implementation

**Open Source Security Testing
Methodology Manual**

(<http://www.isecom.org>)



OWASP Testing Guide v3.0

release



Creative Commons (CC) Attribution Share-Alike
Free version at <http://www.owasp.org>

**Open Web Application Security
Project
Documentation
Testing Guide (integrates with
OSSTMM)
Code Review Guide
Development Guide
Tools
WebScarab**

Tools

- Several HTTP intercepting proxies
 - Webscarab
 - Paros
 - BurpSuite
 - Tamperdata (Firefox plugin)

Test Automation

- There are bunch of commercial tools
 - HP's WebInspect
 - Aspect's Acunetix
- And free tools
 - Paros
 - Nikto
- So which one is the best?

Test Automation

- SAMATE - Software Assurance Metrics And Tool Evaluation (<http://samate.nist.gov>)
- Found that the best tools find 33% of software vulnerabilities. All of the tools together could detect 50% of software vulnerabilities.

OWASP Top 10 - 2010

Injection Flaws	Injection occurs when user-supplied data is sent to an interpreter as part of a command or query.
Cross Site Scripting (XSS)	XSS flaws occur whenever an application takes user supplied data and sends it to a web browser without first validating or encoding that content.
Broken Authentication and Session Management	Account credentials and session tokens are often not properly protected.
Insecure Direct Object Reference	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter.
Cross Site Request Forgery (CSRF)	A CSRF attack forces a logged-on victim's browser to send a pre-authenticated request to a vulnerable web application, which then forces the victim's browser to perform a hostile action to the benefit of the attacker.
Security Misconfiguration	Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform.
Insecure Cryptographic Storage	Web applications rarely use cryptographic functions properly to protect data and credentials.
Failure to Restrict URL Access	Frequently, an application only protects sensitive functionality by preventing the display of links or URLs to unauthorized users.
Insufficient Transport Layer Protection	Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications.
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

For this presentation we're going to focus on application security issues. We're doing this because it boasts the best developed framework.

OWASP Testing Guide

- The testing guide provides a set of test cases
- Practical guidance for execution

Authentication Testing

- The keys to the kingdom

User enumeration

- Common/default usernames
- Guessable username structure
- Error responses from login application
 - Invalid user
 - Invalid password
- Mitigation
 - Generic error messages
 - Non-standard usernames

Guessable user account

- Common account names
 - admin, test, guest
- Developer/tester accounts in production

Bypass authentication schema

- Forced browsing
 - Only the login page verifies login status
- Parameter modification
 - `http://server/login.asp?authenticated=no`
- Session identifier prediction
- SQL injection

Remember password and password reset

- Identity verification
 - Secret questions
- Password change
 - Immediate interactive
 - Email

Demonstration: Authentication Flaws

- Forgot password
- Basic authentication

Welcome to WebGoat. A deliberately insecure web application used to demonstrate common web application vulnerabilities.

Web application security training simulator

Demonstration: Parameter tampering

- Exploit hidden fields
 - Change Price field in HTTP header

Session management

- HTTP being stateless requires a mechanism to provide state
 - VIEWSTATE
 - PHPSESSID
 - JSESSIONID

Bypass session management schema

- Session is most typically implemented using cookies
- Cookie collection
 - What generates and consumes cookies?
- Cookie analysis
 - Session token structure and predictability
 - Cookie structure and lifetime
- Cookie manipulation

Demonstration: Session management

Spoof an authentication cookie

Reverse and decrement by one:

aspect auth cookie = 65432udfqt**b**

webgoat auth cookie = 65432ubphc**fx**

Session fixation

- Session cookie set upon accessing application
- New session cookie not set upon successful authentication
- Session cookie can be fixed using other vulnerabilities

Cross Site Request Forgery

- Trick an authenticated to execute a malicious action
 - Login as a valid user
 - Have user execute pregenerated request to perform function

Demonstration: Cross Site Request Forgery

- Unauthorised bank transfer

```
<IMG
```

```
SRC="http://127.0.0.1/WebGoat/attack?Screen=9&menu=900&transferFunds=4000" width="1" height="1">
```

Authorisation Testing

- Controlling access to resources

Path traversal

- Identify parameters that may be used for file operations
- Input validation
 - ../../../
 - %2e%2e%2f %2e%2e%2f %2e%2e%2f
 - Many other encoding options

Bypass authorisation schema

- “Horizontal” authorisation bypass
- Forced browsing by unauthenticated user
 - http://www.site.com/admin/stop_server.asp

Privilege escalation

- “Vertical” authorisation bypass
- Tamper with group, profile or role parameters

Demonstration: Access Control

- Bypass a path based access control scheme

Data validation testing

- The root of all evil

Reflected Cross Site Scripting

- An attack needs user interaction
- Identify parameter inputs that are reflected to the browser
 - `<script>alert("xss")</script>`
 - Lots more (CAL9000)

Stored Cross Site Scripting

- Same as reflected but persistent
 - Usual suspects are guestbooks and user comments

Demonstration: Cross-site Scripting

- Reflected XSS attacks
- Stored XSS attacks

SQL injection

- Identify probable inputs to database queries
 - Login pages
 - Search pages
- Detection is as simple as '
 - The inclusion of this query delimiter throws an exception

Standard SQL injection

Consider a typical authentication database query

```
SELECT * FROM Users WHERE Username='$username' AND  
    Password='$password'
```

Username input is ' or 1=1--

Query becomes

```
SELECT * FROM Users WHERE Username=' ' or 1=1-- ' AND  
    Password='$password'
```

Query is always true, usually authenticating as the first record in the Users table

Demonstration: Injection flaws

- Numeric SQL injection
- 101 OR 1=1

Blind SQL injection

- Doesn't provide any visible response
- Use inference to determine query structures
- Lot of guesswork (lots of requests)

Code injection

- Can lead to Remote File Inclusion (RFI) vulnerability
 - User accessible variables in include() statements
- Accept user input in the construction of application code within the application

OS Commanding

- Accept user input in the construction of a system call within the application
- Nuke the server
 - `rm -f /*`

Thank you!



- In this session we have only been able to explore the tip of the iceberg
- I would encourage you to continue to broaden your knowledge of security testing.