

# Statistics Whitepaper



## Web Application Vulnerability Statistics 2010-2011

**Alex Hopkins**

[whitepapers@contextis.com](mailto:whitepapers@contextis.com)

**February 2012**



---

## Contents

<b>Findings Summary</b>	<b>4</b>
<b>Sector Analysis</b>	<b>6</b>
Sector Comparison	7
Individual Sector Analysis	8
<b>Detailed Analysis</b>	<b>12</b>
Server Configuration Analysis	12
Information Leakage Analysis	14
Authentication Analysis	15
Session Management Analysis	16
Authorisation Analysis	17
Input Validation Analysis	18
Cross-Site Scripting Analysis	20
“Other” Input Vulnerabilities	21
Encryption Analysis	22
<b>OWASP Top 10 Analysis</b>	<b>23</b>
<b>Conclusions</b>	<b>25</b>
<b>Dataset Restrictions</b>	<b>26</b>
<b>About Context</b>	<b>27</b>
<b>Works Cited</b>	<b>28</b>
<b>Glossary of Terms</b>	<b>29</b>



## Abstract

Over the past two years Context have amassed statistics on a range of IT security activities based on the output of real-world IT security consultation engagements. One of the most common activities performed during this period has been web application penetration tests. This whitepaper will provide a unique insight into the state of web application security, presenting penetration test analysis from a dataset containing nearly eight thousand confirmed vulnerabilities found in almost six hundred pre-release web applications during the period January 2010 and December 2011.

This dataset has been generated using the output from manually guided penetration tests and not through the use of fully automated vulnerability scanners. As all vulnerabilities have been identified and confirmed manually, the dataset provides a credible and high-quality resource with which to review the current state of web application security.

This review represents the first analysis of the dataset and seeks to identify trends currently affecting the security of web applications. This analysis will help the industry to improve security by highlighting problem areas and to identify areas where specific classes of vulnerability are on the increase.

## Conclusions

- On average, the number of issues identified during each web application penetration test increased during the course of 2011
- Government, Finance, Law and Insurance sectors have seen the largest increases in vulnerabilities identified within their web applications
- Server misconfiguration and information-leakage vulnerabilities are the most prevalent category of issues identified
- All categories of vulnerability are increasingly being identified within web applications, with the exception of input validation weaknesses
- Cross-Site Scripting affected two thirds of applications in 2011
- SQL Injection affected nearly one in five applications in 2011
- In general, the proportion of issues identified during both 2010 and 2011 remain consistent, indicating that developers continue to make the same mistakes.



## Findings Summary

On average, the number of security issues affecting each web application increased from approximately 12.5 in 2010 to 13.5 in 2011.

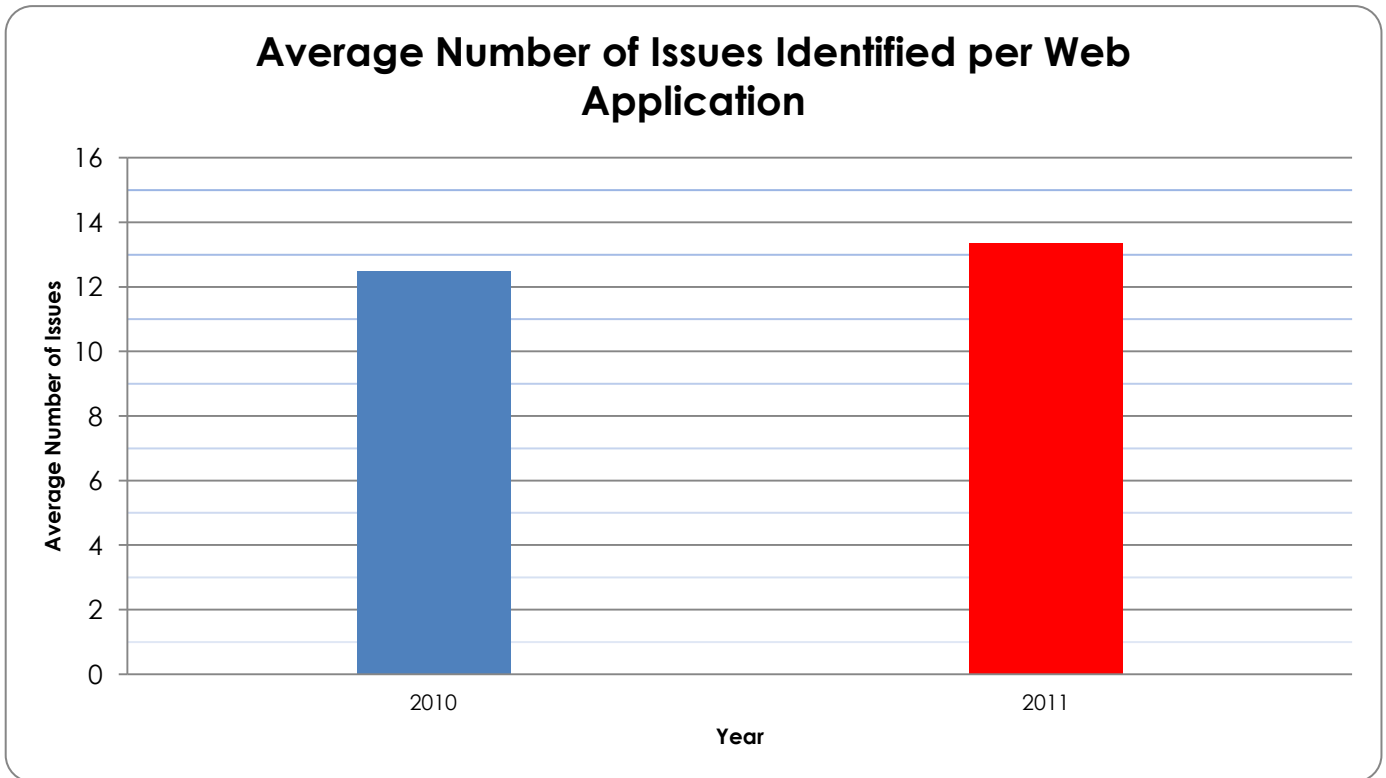


Figure 1

Context grouped each of the identified vulnerabilities into one of eight broad categories for further analysis:

Category	Description
<b>Server Configuration</b>	Insecure server configuration settings that result in security vulnerabilities
<b>Information leakage</b>	Information leaked by the application that could be used by an attacker to help mount an attack
<b>Authentication weaknesses</b>	Issues concerning the application's authentication mechanism that could be exploited by an unauthenticated attacker to either gain or assist in the gaining of authenticated access
<b>Session Management weaknesses</b>	Session management issues that could allow an attacker to either hijack or assist in the hijacking of other users' sessions
<b>Authorisation weaknesses</b>	Issues concerning access controls that could allow an attacker to perform either horizontal or vertical privilege escalation



Category	Description
<b>Input validation weaknesses</b>	Issues concerning any type of weaknesses in the validation of user-supplied input that could be exploited by an attacker to perform some useful purpose
<b>Encryption Vulnerabilities</b>	Issues that concern the confidentiality of data during transport and in storage
<b>Other</b>	Any other issues that cannot be neatly categorised above

Using the broad categories defined above it was possible to identify areas within web applications that caused the highest number of vulnerabilities. In 2010 and 2011, insecure server configuration and information leakage accounted for the highest number of vulnerabilities identified. Both categories also saw some of the largest increases between 2010 and 2011. All other categories saw an increase in the number of vulnerabilities identified, particularly the category of authentication weaknesses. The only exception to this trend was input validation. The decrease in input validation vulnerabilities is likely to be due to the increased use of frameworks which offer built-in input validation features.

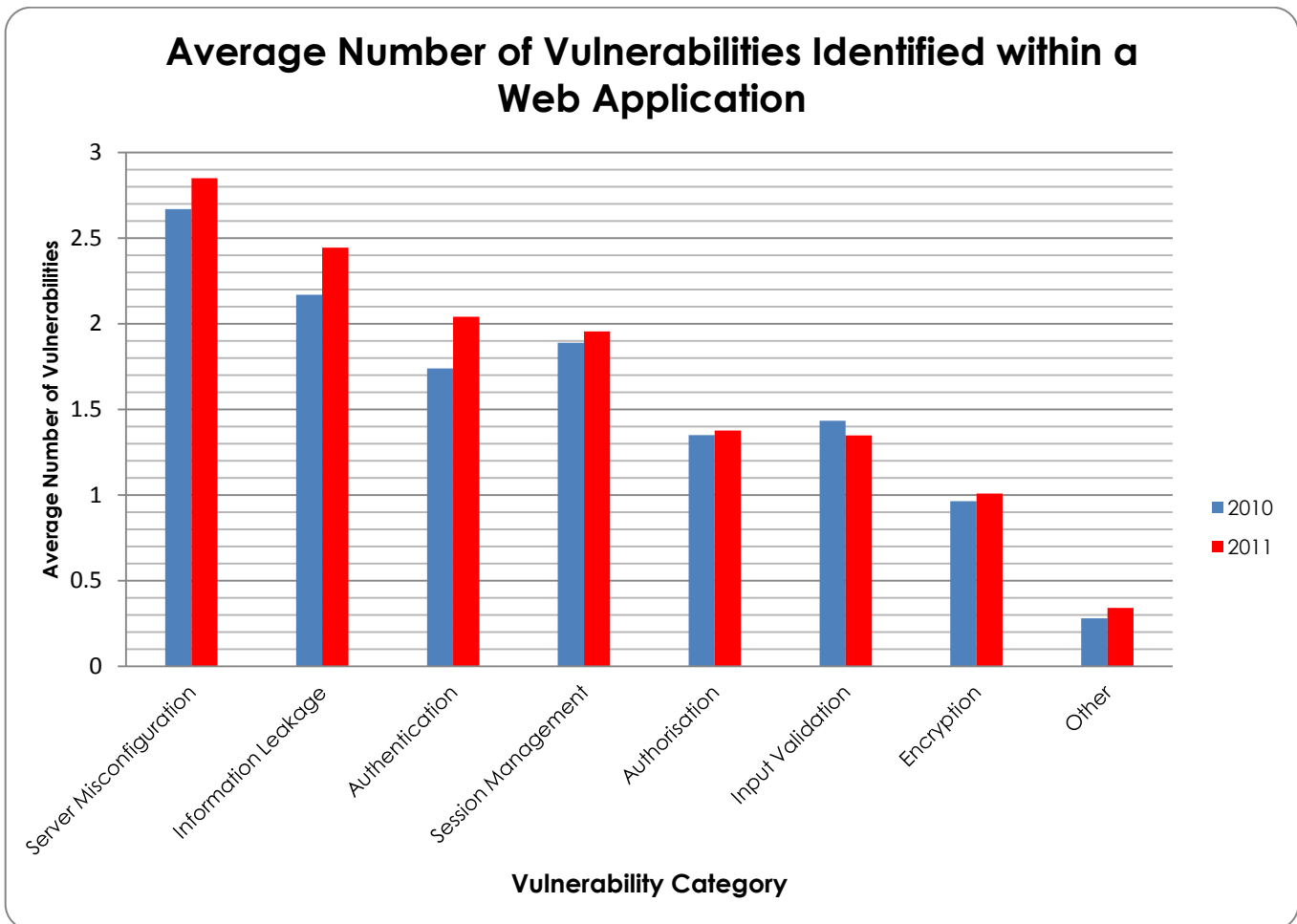


Figure 2



## Sector Analysis

In 2010 and 2011 Context performed penetration tests for companies operating across a broad range of business sectors. To determine if any one particular industry produced web applications that were more susceptible than others, the following industry groupings were created:

- Media and Advertising
- UK Government (the dataset does not include classified government findings)
- Healthcare
- Technology and Telecommunications
- Insurance and Law
- Financial Services (Europe)
- Other (includes Charity, Not for Profit, Gambling, Recruitment, Extraction)

Web applications within the Government (UK) sector were found to contain the highest and second highest number of vulnerabilities in 2011 and 2010 respectively (not including the multi-sector "other" category). Web applications within the financial services sector contained one of the lowest vulnerability counts in 2010; however, this situation changed in 2011 with an average increase of roughly 1.5 issues per web application tested. The "Law and Insurance" sector also saw similar results, seeing an average increase of roughly 2.5 issues per web application penetration test between 2010 and 2011.

Vulnerabilities within the "Media and Advertising" sector contained a particularly high number of vulnerabilities during 2010; however this number dramatically decreased in 2011. The high issue count identified in 2010 is likely to be a result of a substantial increase in the number of penetration tests performed against user-driven websites not traditionally subject to full manual penetration testing.

Despite the intriguing data it is likely that analysis over a longer time period would be required to accurately predict future trends affecting the sectors.



**Sector Comparison**

The graphs shown below demonstrate the average vulnerability count across the various sectors in 2010 and 2011.

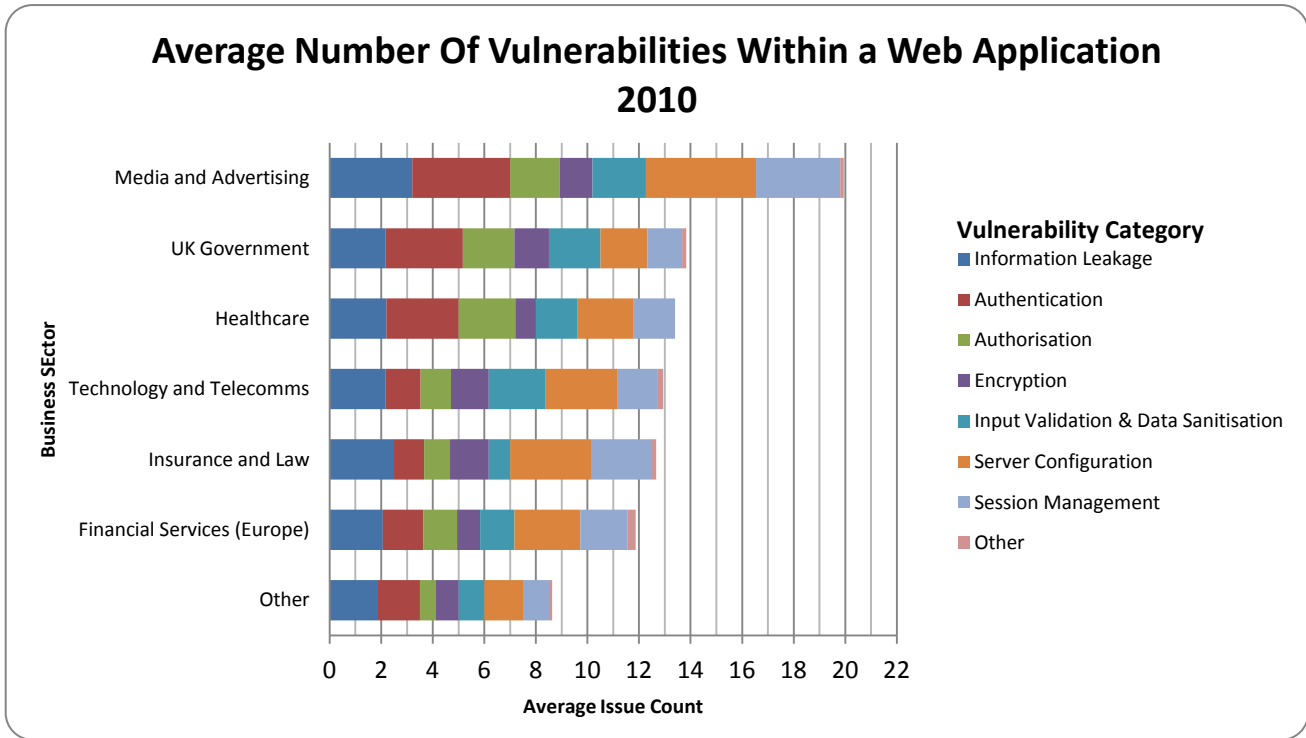


Figure 3

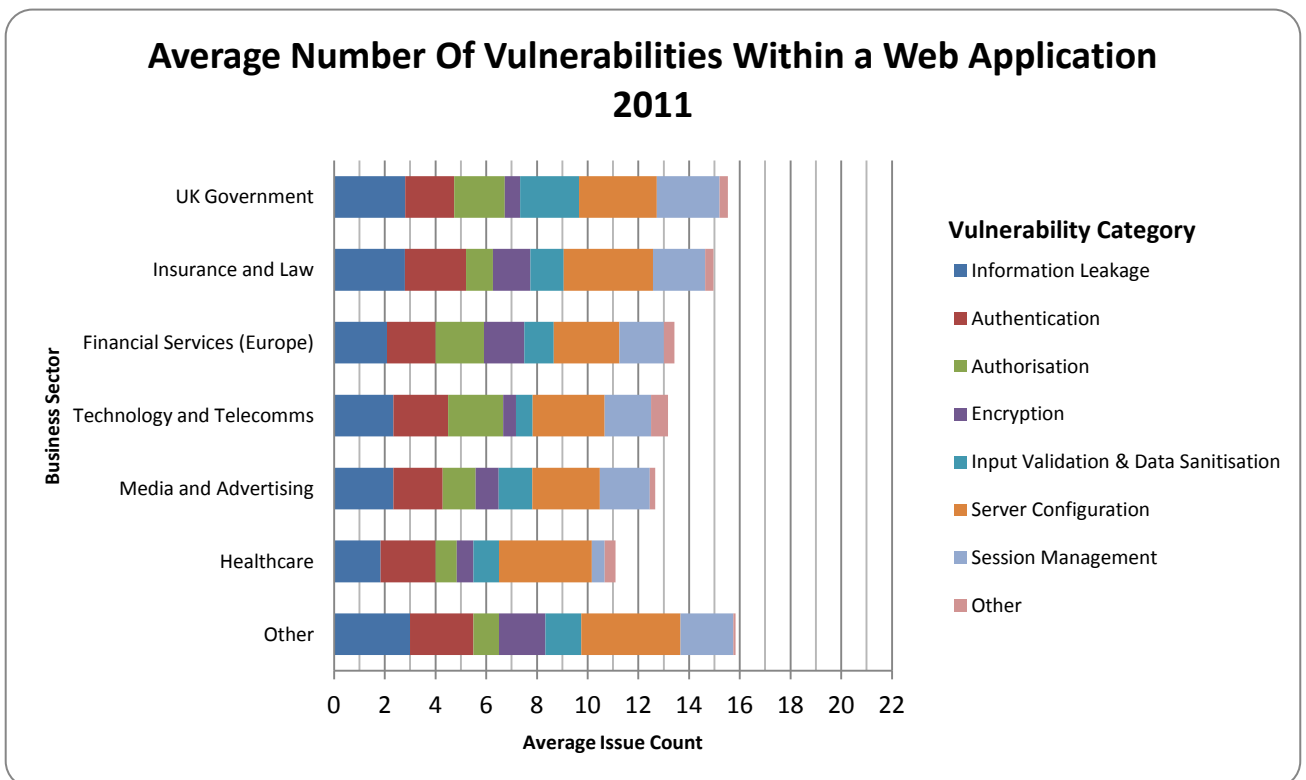


Figure 4



**Individual Sector Analysis**

The following graphs highlight the changes in the average number of vulnerabilities identified for the sectors between 2010 and 2011. The scale represents the average number of vulnerabilities identified within a particular vulnerability category. A larger area signifies a higher number of vulnerabilities being identified. A larger red area (than blue) signifies an increase in the number of vulnerabilities identified for a particular sector in 2011 over 2010.

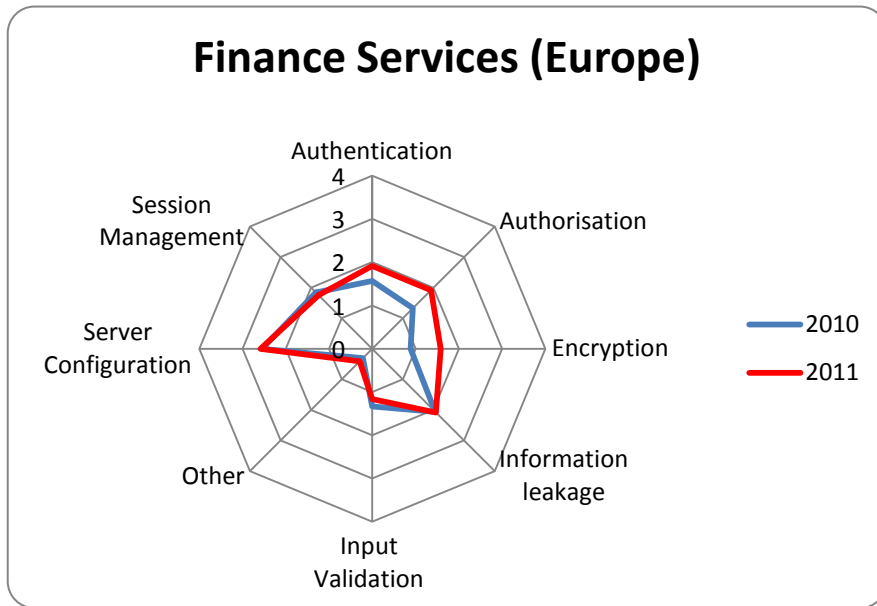


Figure 5

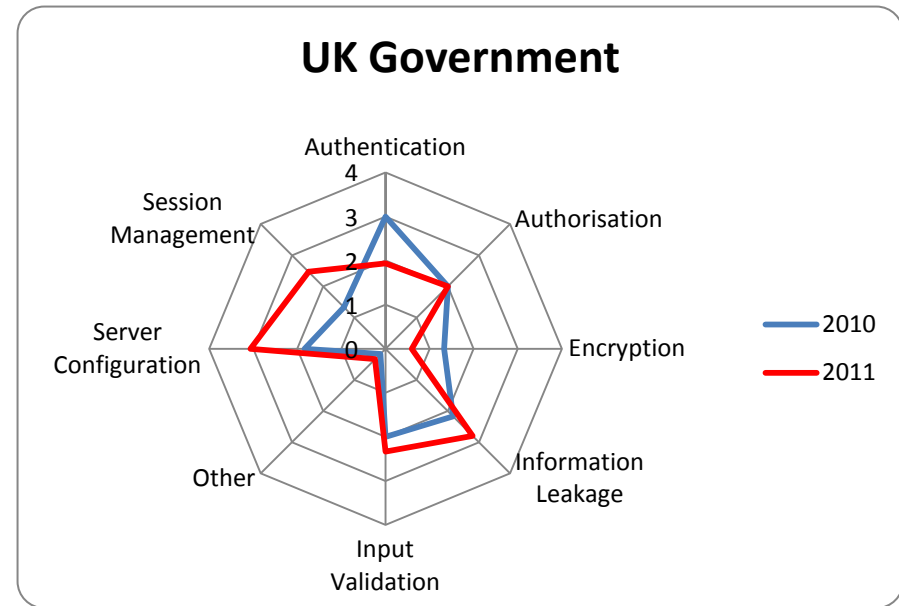


Figure 6



### Media and Advertising

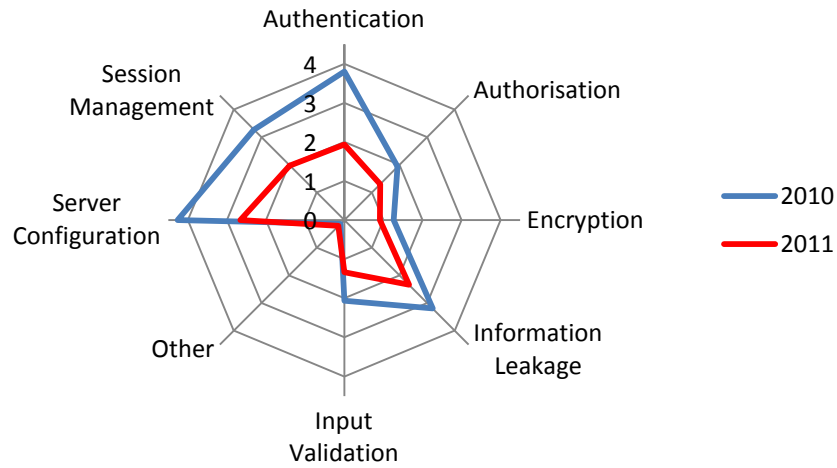


Figure 7

### Technology and Telecommunications

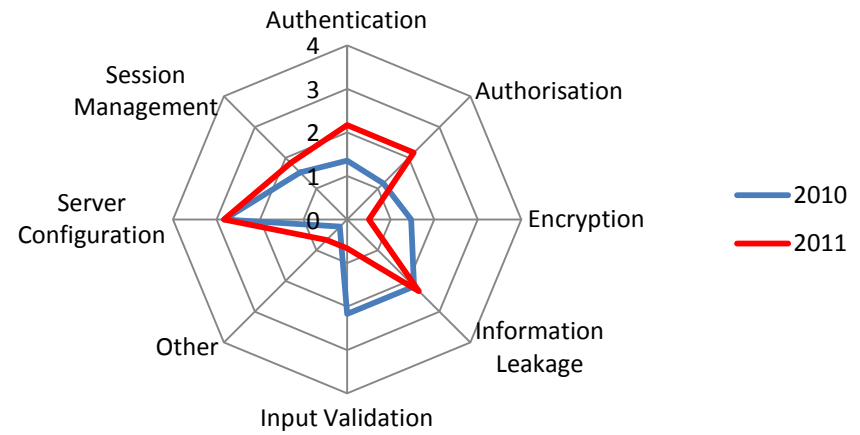


Figure 8

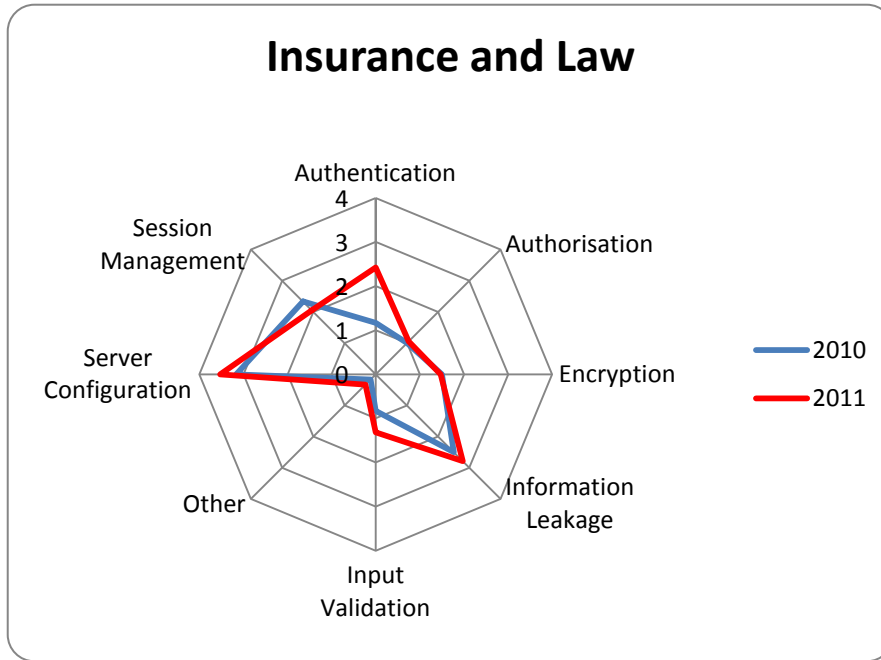


Figure 9

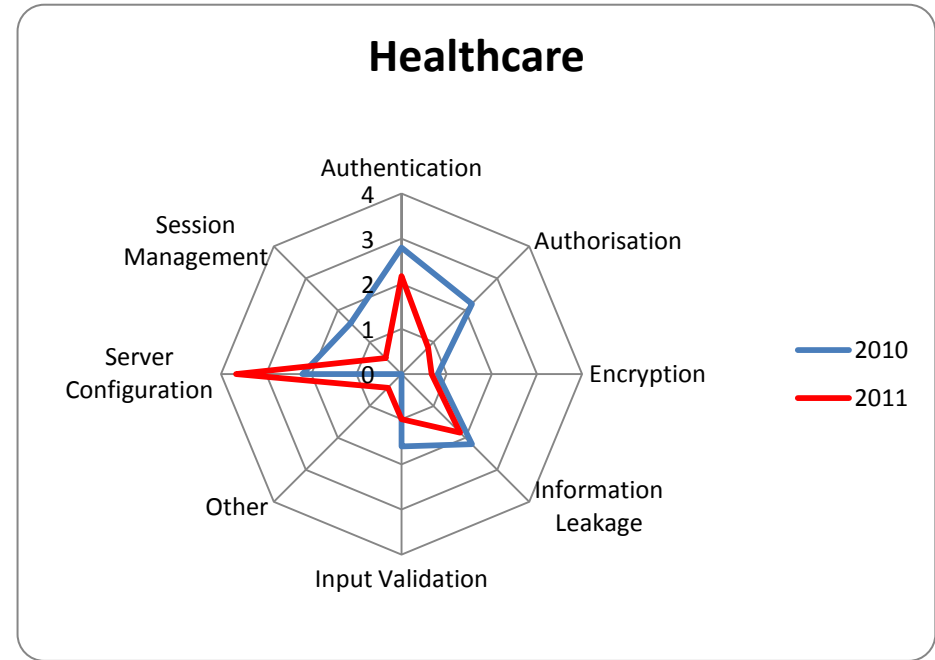


Figure 10

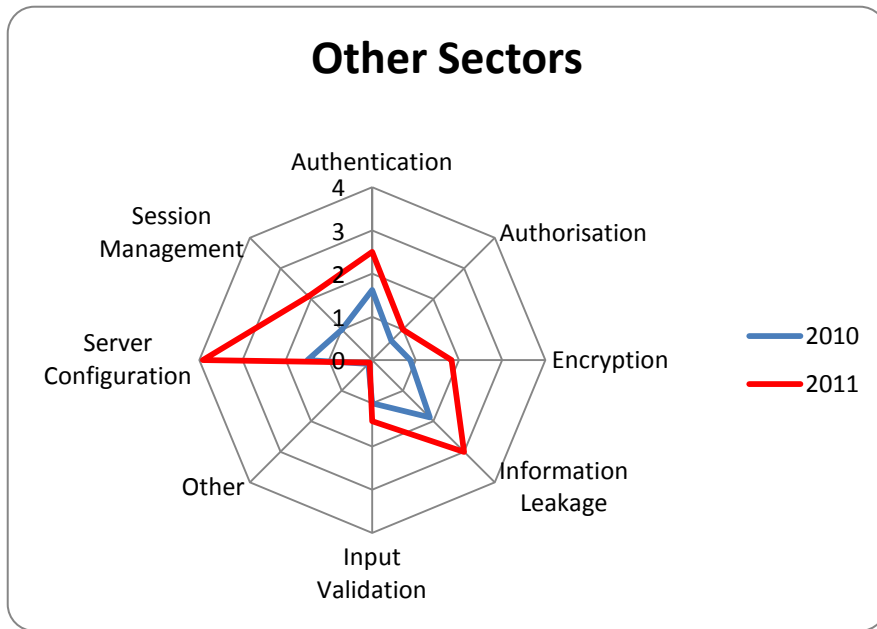


Figure 11



---

## Detailed Analysis

It was possible to further subcategorise the broad vulnerability categories described above to provide a more granular view of the vulnerabilities identified. In the majority of cases, the percentage of applications on average saw a slight increase from 2010 to 2011. Overall, a similar proportion of issues were consistently identified during both 2010 and 2011, indicating that developers continue to make the same mistakes when developing web applications.

## Server Configuration Analysis

A number of server configuration vulnerabilities were found to affect a large proportion of web applications tested. On average, just fewer than three server configuration weaknesses were identified within each web application tested in 2011. Four subcategories affected between 40 and 69 percent of the applications tested in 2010 and 2011. These include:

1. Revealing HTTP headers
2. System error messages
3. Lack of search index protection
4. Vulnerable software versions

These represent some of the most prevalent issues affecting web applications today. However, whilst the first three are likely to be classified as low impact issues, the fourth category – vulnerable software versions – could potentially have more serious consequences depending on the vulnerabilities within the affected software.



## Percentage of Web Applications Containing Server Configuration Vulnerabilities

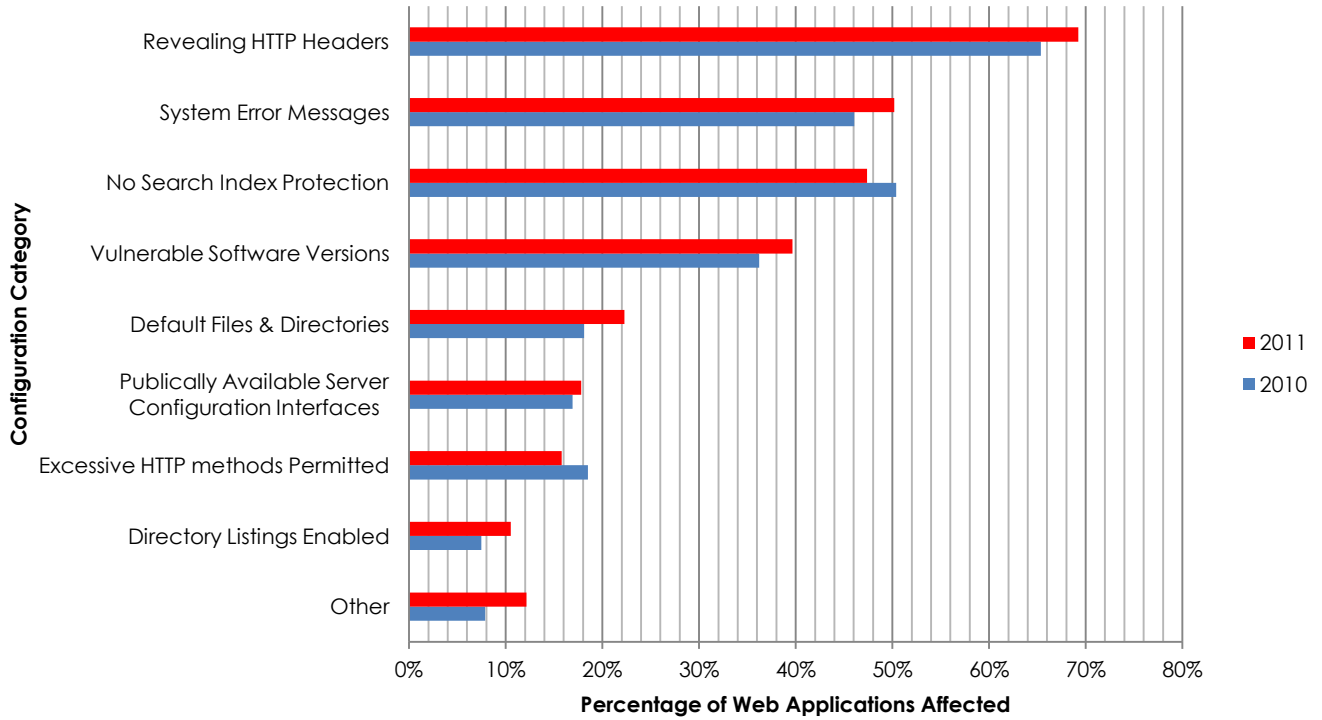


Figure 12



## Information Leakage Analysis

A large proportion of web applications were found to leak information that an attacker could use to help mount further attacks. On average, just fewer than 2.5 information leakage vulnerabilities affected each web application tested in 2011. The use of the term “leakage” here describes leakage of information by the application itself rather than by the server software (in which case it has been categorised as a server configuration issue). The most common leakage vulnerability identified was that of “Revealing User Error Messages”. A commonly-occurring example of this type of vulnerability would be that of an error message returned by a login page stating that a particular user does not exist. An attacker could exploit this knowledge to enumerate valid usernames. Other leakage vulnerabilities affecting a large proportion of applications include those significant within a shared computer environment, such as “auto completion” not being disabled for sensitive form fields, and the caching of authenticated pages.

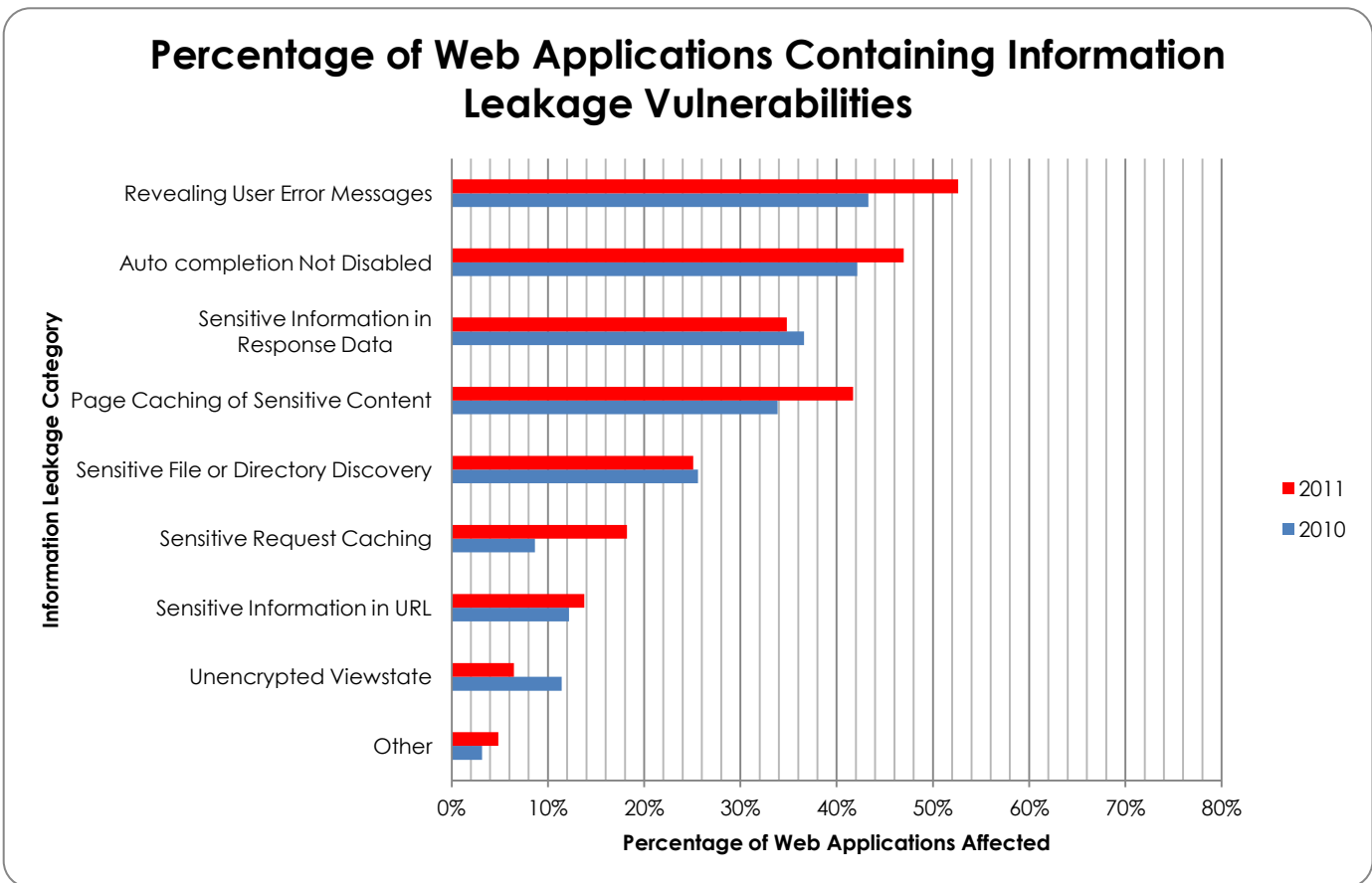


Figure 13



## Authentication Analysis

Nearly all types of authentication issues were found to be on the increase. On average, just over 2 authentications related vulnerabilities were identified within each web application tested in 2011. Serious issues can result where an application contains multiple authentication weaknesses, where in the worst cases an attacker can gain authenticated access by using a multi-stage attack process (statistics for this are not contained within the dataset). Two of the most common issues to affect web application authentication are those of concurrent sessions and weak password policy. These affect approximately 50 and 47 percent of applications respectively. A large increase was seen in the number of applications containing weak password change mechanisms, where the users were not verified by requiring the old password when making a password change, or where the application did not provide the functionality at all.

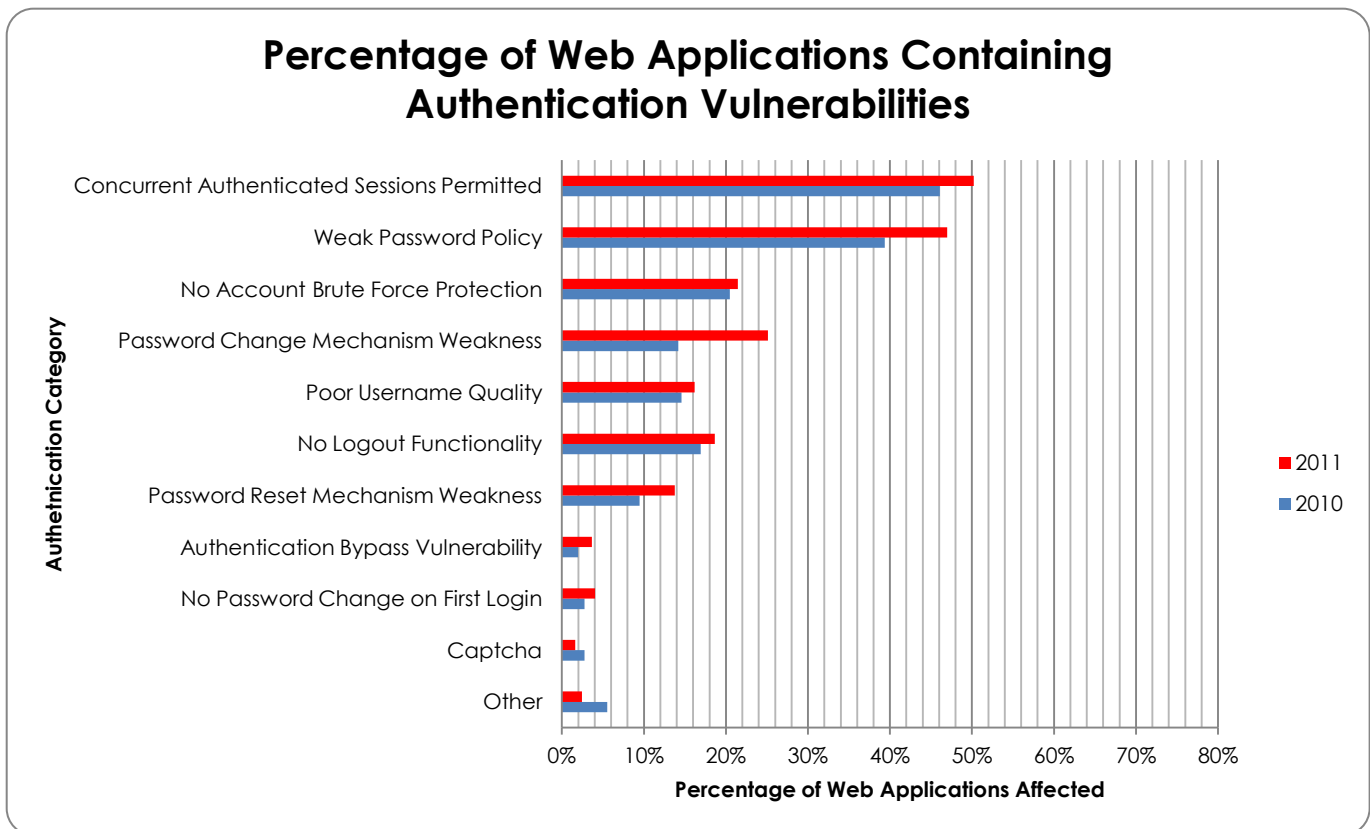


Figure 14



## Session Management Analysis

Session management vulnerabilities saw only a slight increase in 2011, with just fewer than 2 issues identified per web application tested. Classic errors are still being made, including a substantial number of applications not setting the appropriate cookie flags (HTTPOnly and Secure) when required. Additionally, many applications do not prevent sessions from being ported from one computer to another, a security measure which could help protect against session hijacking.

With frameworks providing session management functionality for a number of years now, it can be seen that a number of the so-called 'old-school' vulnerabilities continue to decline; examples include the use of Cookie-less sessions and weak session token content (not random, not large enough, use of a small character class space, etc.).

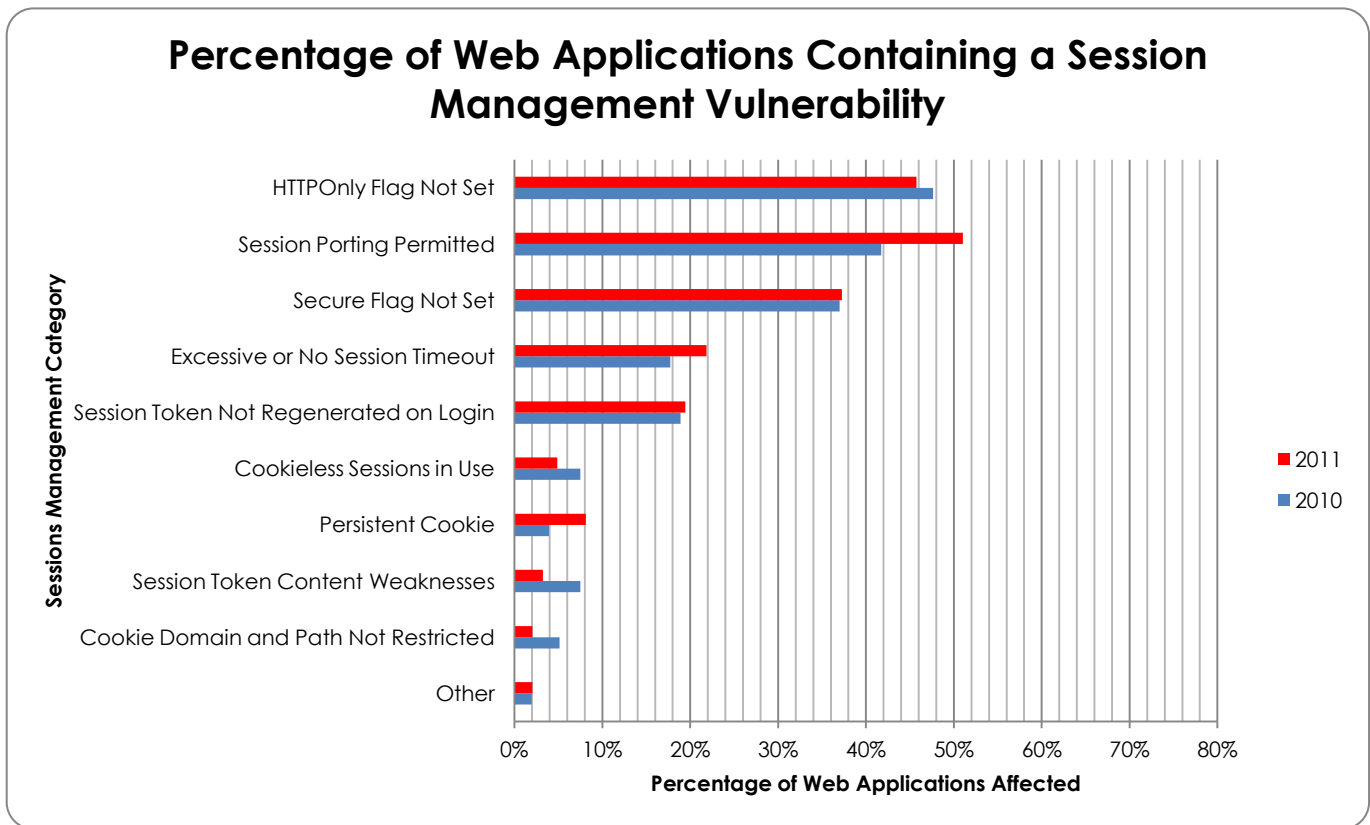


Figure 15



## Authorisation Analysis

Authorisation vulnerabilities saw a moderate increase in 2011, with just fewer than 1.5 authorisation vulnerabilities affecting each web application tested. The ability to gain unauthorised access to resources through Parameter Manipulation or Forced Browsing was possible in 29 percent of all web applications tested in 2011. This is an increase over 2010, with Forced Browsing seeing an increase of 7 percent.

Vulnerabilities that would require an attacker to perform social engineering, such as clickjacking or cross-Site Request Forgery (CSRF) still affect a high proportion of web applications tested; however, both saw a decrease in 2011. Note that only applications where some useful function could be achieved by an attacker were counted as being vulnerable. The 5 percent decrease in the number of applications affected by CSRF is likely due to an increase in the use of frameworks offering prevention mechanisms (e.g. CSRF Tokens). Clickjacking has also seen a small decrease, which is probably due to an increase in publicity surrounding the issue (Stone, 2010) increasing the awareness of developers.

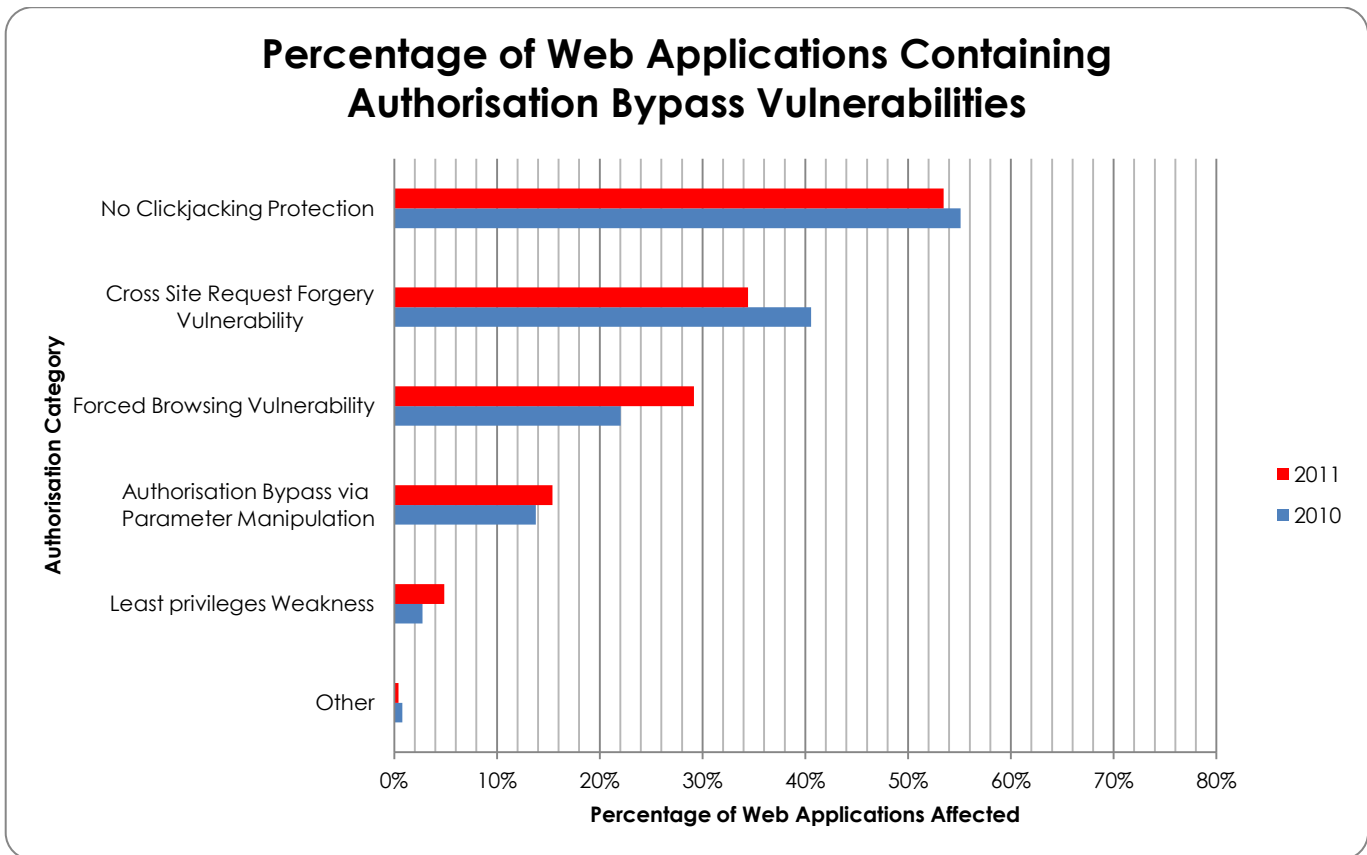


Figure 16



## Input Validation Analysis

Input validation was the only category to see a drop in the number of instances identified within each web application. Just under 1.5 Input Validation vulnerabilities were identified within each web application tested. By far the most prevalent vulnerability identified within this category is that of cross-site scripting. Cross-site scripting is a class of vulnerability that could be abused by an attacker to perform a number of attacks, including the hijacking of authenticated sessions belonging to other users. Roughly 66 percent of applications were found to be vulnerable to cross-site scripting in 2011. However, this represents a slight decrease from 69% in 2010. Further discussion of cross-site scripting vulnerabilities will follow shortly.

One of the most infamous web application vulnerabilities - SQL Injection - bucked the input validation trend with a slight increase. Just less than one in five applications were found to be affected in 2011, showing that the SQL injection threat is still not receding. An attacker could typically exploit an SQL injection vulnerability to extract data directly from a database, or in certain situations access the underlying operating system to run commands or read arbitrary files.

A surprising finding is that just over 11 percent of applications were identified as not performing anti-virus scans on user-uploaded files. Unfortunately the dataset does not contain statistics on how many of the applications actually provided file upload functionality (as many would not have). It is therefore not possible to identify the actual percentage of file upload functions that were implemented do not perform anti-virus scans.

Actual file upload weaknesses saw a decrease to only 8 percent of applications in 2011. This is a positive development as weaknesses within file upload mechanisms can often allow attackers to upload web Trojans to the application. In turn, these can allow an attacker to execute commands on the remote operating system or provide access to the underlying database. The caveat regarding the dataset not containing statistics on the percentage of applications providing file uploads also applies to this issue. It is likely that in both cases, when file upload functionality is provided, the percentage of functions that contain insecurities will be higher than the statistics can demonstrate here.

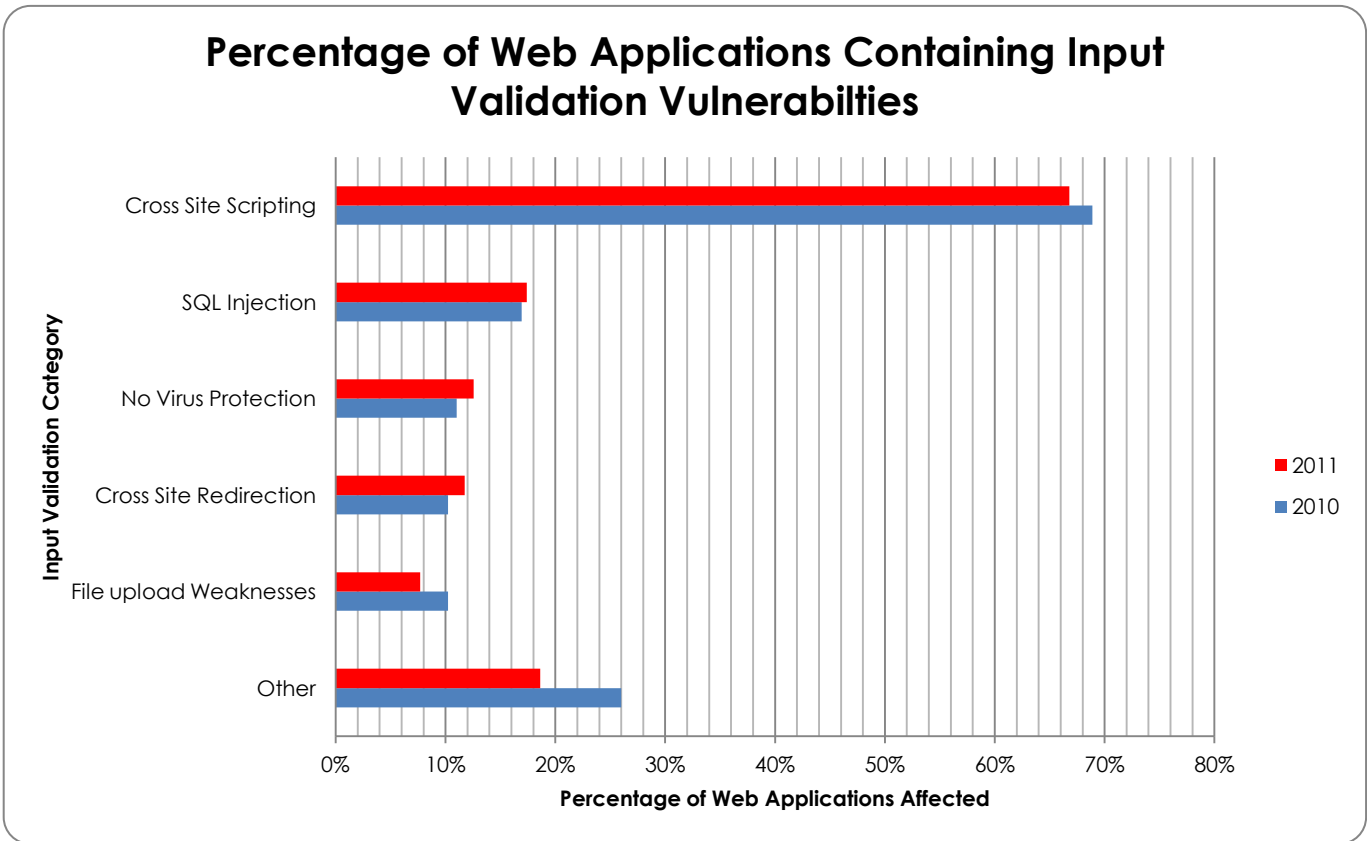


Figure 17



### Cross-Site Scripting Analysis

Cross-site scripting vulnerabilities can be broadly categorised into two varieties; persistent and non-persistent. A persistent cross-site scripting vulnerability occurs when it is possible for an attacker to inject HTML/JavaScript code into a webpage which is then stored by the application. When a victim visits an affected webpage, the injected HTML/JavaScript is retrieved from storage, embedded into the webpage HTML, and rendered by the victim's browser. A non-persistent cross-site scripting vulnerability on the other hand requires the attacker to craft a malicious link using HTML/JavaScript and then entice the victim into following this link. When the victim follows the link to the affected page, the HTML/JavaScript then executes. A non-persistent payload generally requires some level of social engineering on the attacker's part in order to execute.

Overall, the ratio of cross-site scripting vulnerabilities can be seen to be approximately two-thirds non-persistent to one-third persistent over both years. The ratio saw a four percent increase in the persistent variety when compared against non-persistent in 2011:

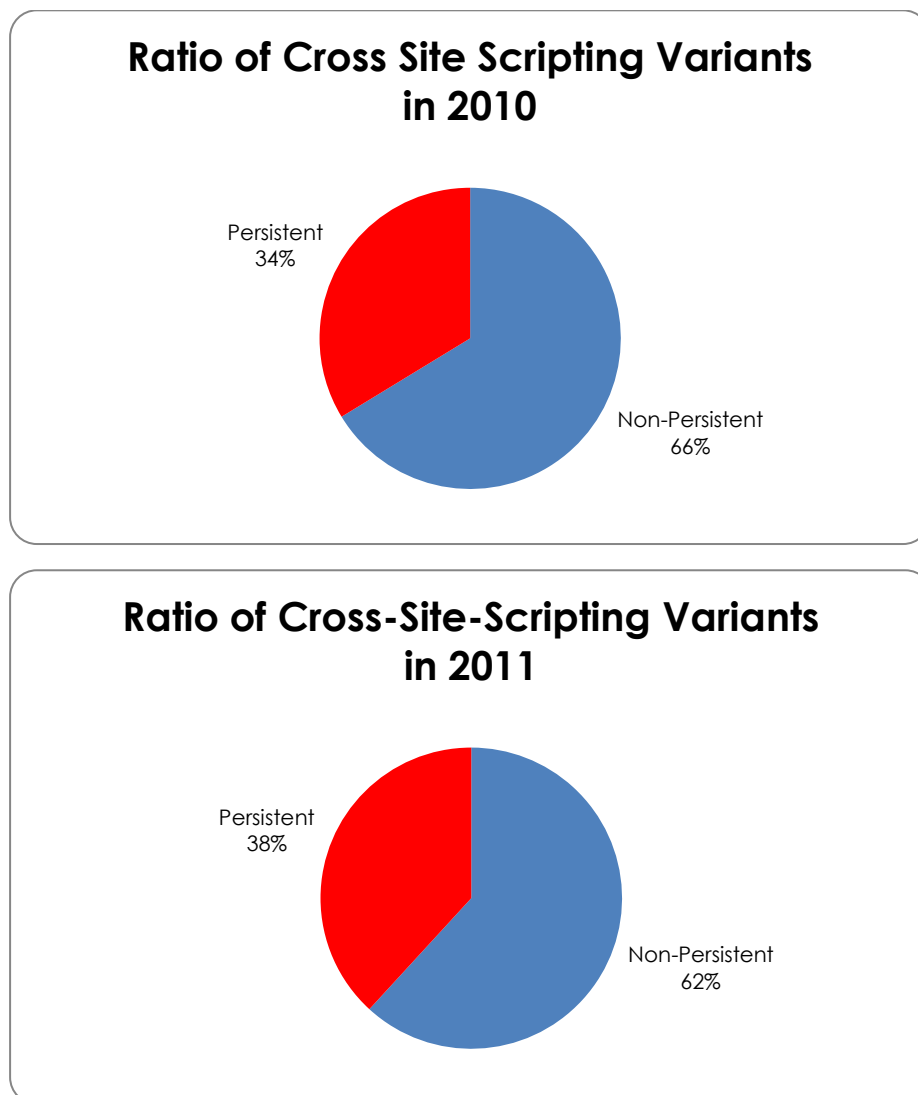


Figure 18



**“Other” Input Vulnerabilities**

The “other” input validation category shown in the figure 17 contains a number of vulnerabilities significant when referring to the OWASP Top 10 (see Section OWASP Top 10 Analysis). These include XPATH, LDAP and OS command injection vulnerabilities. The following graph provides a breakdown of the “other” input validation category to highlight these vulnerabilities. In general, these injection weaknesses appear to be in decline:

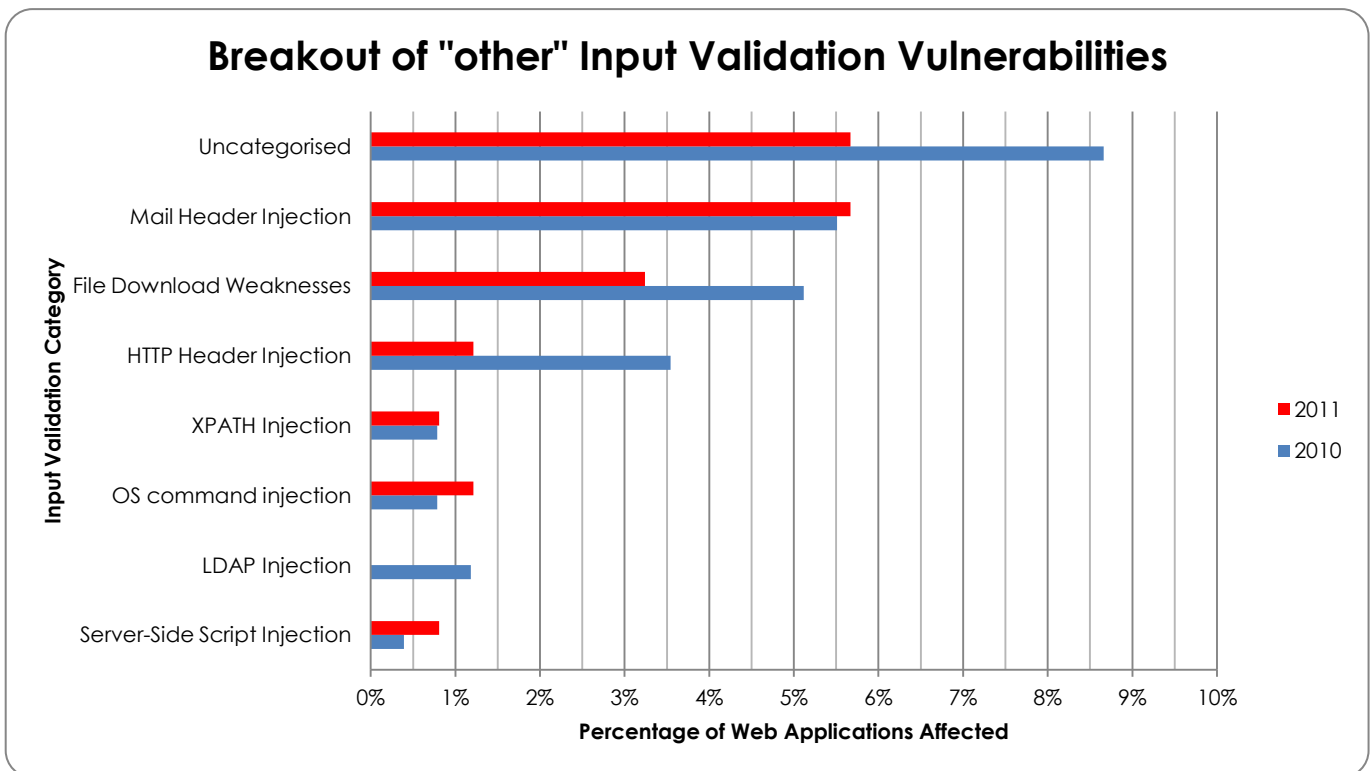


Figure 19



## Encryption Analysis

In 2011, 30 percent of the web applications tested by Context were found to send sensitive data (or permit sensitive data to be sent) over an unencrypted channel (e.g. HTTP), a value comparable to that seen in 2010.

Approximately 22 percent and 28 percent of web applications were found to have insecure SSL configurations supporting connections using weak ciphers and protocols respectively. Unfortunately, statistics on how many of the web applications used SSL were not available within the dataset as not all applications tested would have required SSL. Static “brochure-ware” websites are good examples of such sites. SSL statistics were only recorded where security issues were encountered, the actual percentage of insecure SSL configurations where SSL is used is therefore likely to be higher than indicated by these statistics.

It should also be noted, that some of these issues are likely to only affect users of archaic web browsers (Wallis, 2011).

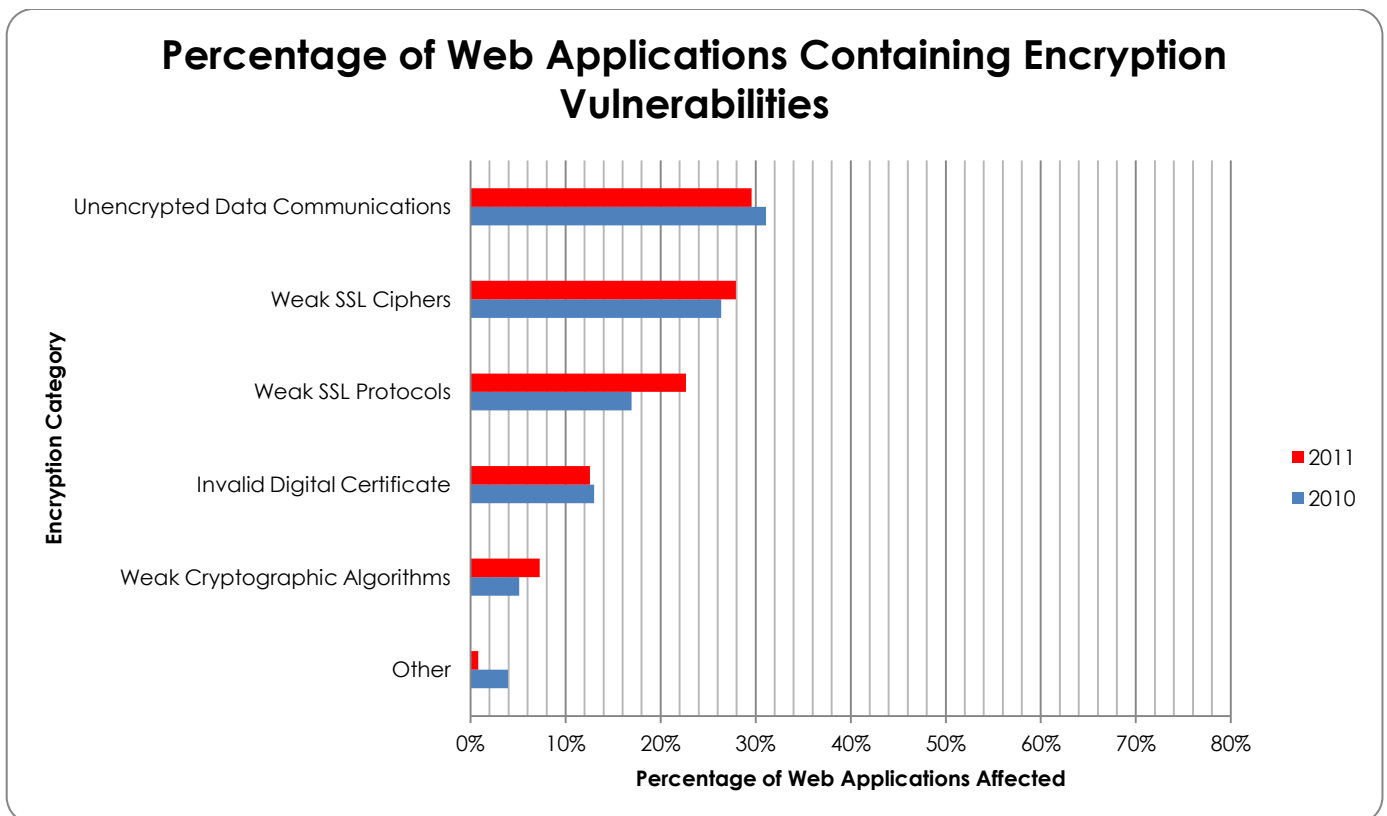


Figure 20



## OWASP Top 10 Analysis

A document commonly used by the industry to categorise web application vulnerabilities is that of the OWASP Top 10 (OWASP, 2010). The OWASP Top 10 is a broad consensus of what are regarded as the most critical security flaws currently affecting web applications. Using the dataset it was possible to identify which OWASP Top 10 issues are most commonly encountered. The OWASP Top 10 categories can be seen below and are not categorised by prevalence but by criticality:

- A1: Injection (SQL, LDAP, XPATH, OS command)
- 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: Insecure Cryptographic Storage
- A8: Failure to Restrict URL Access
- A9: Insufficient Transport Layer Protection
- A10: Unvalidated Redirects and Forwards

The graph below shows the average number of OWASP Top 10 issues identified per web application penetration test. It can be seen that the most prevalent vulnerability category identified during web application penetration tests are due to “Broken Authentication and Session Management” issues (4 per web application) as well as “Security Misconfiguration” (just over 2.5 per web application). However, it should also be noted that a number of issues can be categorised within these two groups. When determining the highest specific variety of vulnerability defined by the OWASP Top 10 it can be seen that cross-site scripting is found to affect two thirds of all applications tested.

With the exception of the cross-Site scripting and cross-Site Request Forgery OWASP categories, the average web application was found to contain vulnerabilities from categories whose prevalence had either increased or remained constant from 2010 to 2011. The largest increases applied to the vulnerabilities of type “Broken Authentication and Session Management” and “Server Misconfiguration”.



## Average Number Of OWASP TOP 10 Issues Identified Per Web Application

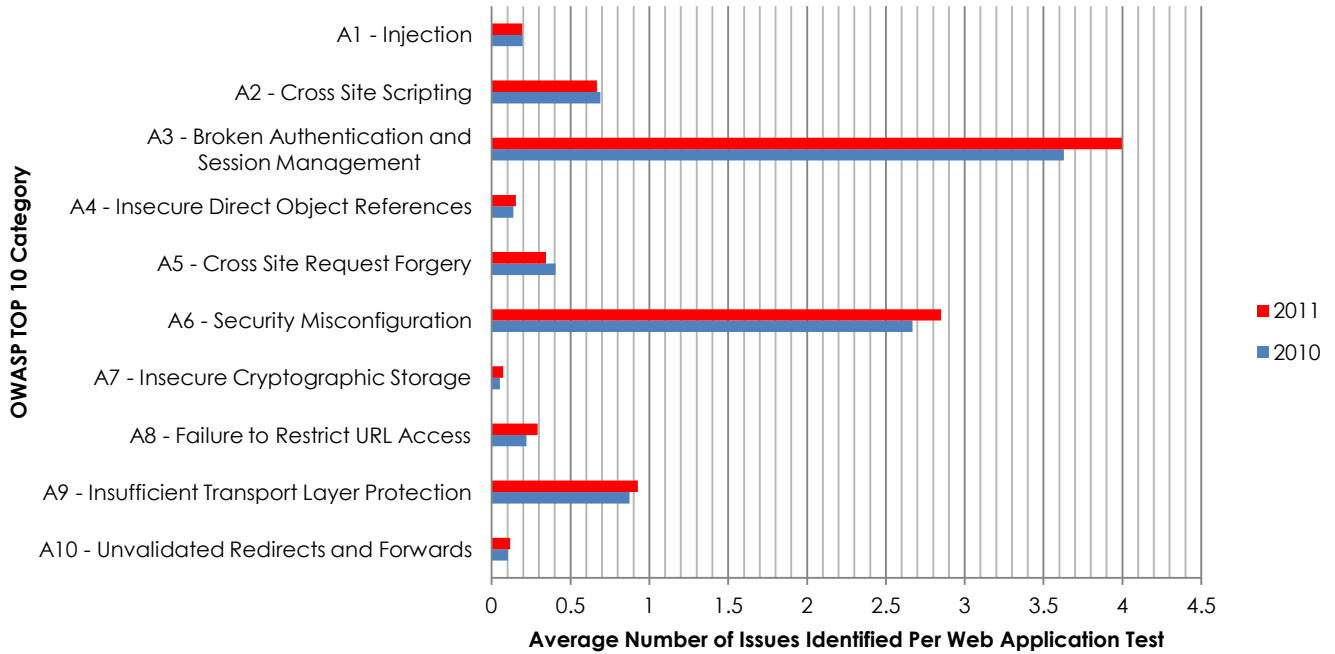


Figure 21



## Conclusions

Having reviewed the data it can be seen that the average level of security applied to web applications is still far from desirable. Overall, the average application tested contained an increased number of vulnerabilities in 2011 when compared with 2010.

The proportions of the vulnerability classes being identified in 2011 remain approximately the same as those being identified in 2010, although the majority of vulnerability instances were found to have increased for each category.

Some positive news can be drawn from the progress that appears to be being made in reducing the number of input validation vulnerabilities. However, it should be noted that despite the overall category decreasing slightly, SQL injection vulnerabilities were actually found to have increased between 2010 and 2011. In addition, whilst the number of cross-site scripting vulnerabilities decreased in 2011, it was still found to affect two thirds of applications tested, making it one of the most prominent vulnerabilities within web applications today.

Identifying future trends across the different sectors is difficult, but it can be seen that of the web applications produced by the various sectors, Government, Financial Services and Law and Insurance sectors saw a significant increase in the average number of vulnerabilities being identified within each web application.

The aim of this paper is to highlight the problem areas affecting web application security today and to demonstrate how things have changed over the last two years. It can be seen that many of the problems faced in 2010 were the same as those faced in 2011 without any area seeing a significant year-on-year improvement. Despite this overall impression, some signs of tentative improvement have been seen, and hopefully this document will provide help as a source of guidance, allowing developers and security professionals to prioritise and focus their web-application security efforts in 2012.



---

## Dataset Restrictions

The following restrictions apply to the dataset:

- UK Government work does not include statistics from classified engagements
- The dataset does not contain data specifying whether credentials were supplied by the client for authenticated testing. However, in the vast majority of engagements credentials will have been supplied. In a small number of engagements testing was performed from a purely black box perspective (where credentials were not supplied by client)
- Analysis of data storage could only be achieved where full access to the database had been achieved through exploitation of other vulnerabilities



## About Context

Context Information Security is an independent security consultancy specialising in both technical security and information assurance services.

The company was founded in 1998. Its client base has grown steadily over the years, thanks in large part to personal recommendations from existing clients who value us as business partners. We believe our success is based on the value our clients place on our product-agnostic, holistic approach; the way we work closely with them to develop a tailored service; and to the independence, integrity and technical skills of our consultants.

Context are ideally placed to work with clients worldwide with offices in the UK, Australia and Germany.

The company's client base now includes some of the most prestigious blue chip companies in the world, as well as government organisations.

The best security experts need to bring a broad portfolio of skills to the job, so Context has always sought to recruit staff with extensive business experience as well as technical expertise. Our aim is to provide effective and practical solutions, advice and support: when we report back to clients we always communicate our findings and recommendations in plain terms at a business level as well as in the form of an in-depth technical report.





## Works Cited

OWASP, 2010. *OWASP Top 10 Project*. [Online]

Available at:

[https://www.owasp.org/index.php/Category:OWASP\\_Top\\_Ten\\_Project](https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)

Stone, P., 2010. *Clickjacking Paper - Black Hat 2010*. [Online]

Available at: <http://www.contextis.co.uk/research/white-papers/clickjacking/>

Wallis, C., 2011. *Server Technologies - SSL2: Should it keep you awake at night?*.

[Online]

Available at: <http://www.contextis.co.uk/research/blog/inf2/>



## Glossary of Terms

Main Category	Sub-Category	Description
Server Configuration	<b>Revealing HTTP Headers</b>	HTTP headers that return server version information
	<b>System Error Messages</b>	Error messages generated by the server revealing information that may be useful to an attacker. In a production environment generic error messages should be used.
	<b>No Search Index Protection</b>	When no search index protection is provided then the web application may have sensitive information indexed by public search engines
	<b>Vulnerable Software Versions</b>	Server software versions that contains publically known security vulnerabilities.
	<b>Default Files and Directories</b>	Default content provided by the server that has not been removed. In some situations this content may contain vulnerabilities or provide some form of information disclosure.
	<b>Publically Available Server Configuration Interfaces</b>	Administrative interfaces that can be accessed by all on the Internet. An attacker could attempt to login to the interface or identify vulnerabilities in order to access administrative content.
	<b>Excessive HTTP Methods</b>	HTTP Methods (PUT, DELETE, TRACE, etc) permitted by the server (but which are most likely not required) exposing the server or application users to various potential threats of varying levels of impact. It is often the case that a production web server will only require a small number of HTTP methods (e.g. GET, POST).
Application Information Leakage	<b>Directory Listings Enabled</b>	Directory listings that reveal filenames and subdirectories within the affected directory. An attacker could use this information to identify functionality or sensitive content that may aid other attacks.
	<b>Revealing User Error Messages</b>	Application error messages that reveal something useful to an attacker. A typical example would be a login function returning an error message indicating a particular username does not exist. An attacker could then use this information to enumerate application usernames.



---

**Auto-Completion Not Disabled**

A server side setting can prevent a browser from caching sensitive fields.

---

**Sensitive Information in Response Data**

Sensitive information may be contained within the server responses. An example could be credentials that are "hidden" within HTML comments.

---

**Page Caching of Sensitive Content**

Authenticated content cached by a web browser could be retrieved by other users revealing sensitive information in a shared computing environment.

---

**Sensitive File or Directory Discovery**

Sensitive files that are not necessary for business operation but are accessible to users who do not require access.

---

**Sensitive Information in URL**

URLs can be cached at various locations including; browser logs, web server logs, proxy logs, etc. A malicious user with access to one of these logs could obtain the sensitive information contained within (e.g. session tokens, usernames, passwords, etc).

---

**Unencrypted ViewState**

ViewState can be used by applications to maintain state on the client side and is passed between the client and server. The ViewState can be encrypted to prevent inquisitive users from inspecting its contents.

---

Authentication

**Concurrent Authenticated Sessions**

Concurrent sessions permit the same user to be logged in multiple times from different locations. This causes problems as it allows for repudiation during auditing.

---

**Weak Password Policy**

Various weaknesses within a password policy can include the creation and use of weak passwords (e.g. short length, use of only characters) as well as permitting password reuse, etc.

---

**No Account Brute Force Protection**

When no account brute force protection is provided an attacker can make an unlimited number of username/password attempts against the login functionality.

---



---

**Password Change Mechanism Weaknesses**

Applications can have a number of weaknesses within their password change mechanisms. This can include the functionality not being provided at all or because a user is not verified by checking the old password. Password change functionality should generally be provided so that a user can change their password on a regular basis (e.g. when they believe their password has been compromised).

---

**Poor Username Quality**

It should not be possible to guess the majority of usernames with a high degree of accuracy. An attacker can harvest this information to perform brute force attacks or shotgun style (limited number of passwords against a large number of usernames) attacks where brute force protection is provided.

---

**No Logout Functionality**

Logout functions should be provided to allow a user to limit access to their session once they have finished.

---

**Password Reset Mechanism Weaknesses**

Applications can contain a number of weaknesses within their Password reset functionality, such as the ability for an attacker to specify the username and to then select an arbitrary email address for which to send the new password.

---

**Authentication Bypass Vulnerability**

This includes vulnerabilities specifically within the authentication mechanism allowing for authentication mechanism to be bypassed directly. Various vulnerabilities could exist, for example weaknesses within the RSA implementation that could be exploited to render its use within the authentication mechanism redundant.

---

**No Password Change on First Login**

When passwords are provided by an administrator or are insecurely delivered it is recommended that password change on first use is enforced.

---

**Captcha**

Captchas are used by applications to protect against automated attacks. Various weaknesses within captchas could be identified such as the implementation of machine readable values.

---

**Session Management****HTTPOnly Flag Not Set**

Applications can restrict client side JavaScript from accessing the session cookies through the use of the HTTPOnly flag being set on cookies.

---



---

**Session Porting Permitted**

Session porting is where it is possible to continue a session by porting the session token from one system to another. This typically happens during session hijacking attacks.

---

**Secure Flag Not Set**

It is possible to set the secure flag on cookies to ensure that cookie values are only ever transmitted over secure channels (e.g. HTTPS)

---

**Excessive or No Session Timeout**

When an application has an excessive timeout it increases the window of opportunity for an attacker to access a legitimate user's session.

---

**Session Token Not Regenerated on Login**

The session token should be set after a successful login has occurred to prevent session token persistence which can be useful for an attacker when performing session hijacking.

---

**Cookieless Sessions in Use**

Cookieless sessions typically place the session token within the URL. This increases its exposure, as the token will appear in a number of logs (including browser history, server, etc), and cannot use the many security mechanisms provided by cookies.

---

**Persistent Cookie**

A cookie that is stored to disk for a period of time using the "expires" cookie directive is known as persistent cookie. It will continue to be sent to the respective server until the "expires" time has been reached. This can have implications when an application is used in a shared computing environment.

---

**Authorisation Clickjacking**

An attack whereby the target application is transparently layered above an innocent looking malicious page. This allows the attacker to manipulate a user into performing actions unbeknownst to them, such as stealing sensitive information or hijacking mouse clicks to perform malicious actions

---

**Cross-Site Request Forgery**

An attack whereby malicious links or web pages can be constructed that will perform some action within the target application should the attacker be successful in enticing the victim into following the link or viewing the web page. Typical exploits would include; changing a victim's password or transferring money between accounts.

---

**Forced Browsing**

It is often possible to directly browse to certain resources or functionality that should not be available to the user requesting it due to poorly defined access controls.

---



---

<b>Authorisation Bypass via Parameter Manipulation</b>	An attack similar to that of forced browsing where it is possible for users to access resources and functionality through manipulation of parameter values for which they should not be allowed. This is again due to poorly defined access controls.
<b>Least Privileges</b>	The application and underlying technologies should be configured to allow the least amount of privilege for a particular user to perform a task. It is not always possible to evaluate all components as access to the underlying operating system is required.
Input Validation	<b>Cross-Site Scripting</b> A class of attack where HTML and/or JavaScript is injected into the application. This content is then rendered in the victim's browser, leaving the user vulnerable to a number of attacks such as session hijacking, theft of credentials and theft of sensitive data.
<b>SQL Injection</b>	A technique used to change the semantics of database queries that are run by the application. They can be used to extract or change sensitive information contained in the database, as well as potentially run system level commands.
<b>Cross-Site Redirection</b>	A class of attack where an attacker exploits the way in which the application acts on a user controllable parameter to redirect the application to another exploit. The attack typically requires some level of social engineering and exploits the user's trust of the website to redirect to "safe" sites.
<b>File Upload Weaknesses</b>	Weaknesses within the functionality that can be abused by the attacker to perform some useful purpose, typically, this would include some form of directory traversal to upload files to arbitrary locations (which may help further weaken the system).
<b>File Download Weaknesses</b>	Weaknesses within the functionality that can be abused by the attacker to perform some useful purpose, typically, this would include some form of directory traversal to download arbitrary files from the server.
<b>No Virus Protection</b>	Where no (or inadequate) anti virus scanning is performed against user supplied file uploads which can then be subsequently downloaded.

---



---

<b>Mail Header Injection</b>	An injection technique used against email functionality provided by the application to allow arbitrary content or destinations to be defined. The control over the mail server can be abused for various purposes (e.g. spamming, phishing, social engineering, etc)
<b>HTTP Header Injection</b>	A technique where attacker supplied input is directly inserted into the HTTP response header. An attacker can attempt to exploit the vulnerability to perform attacks such as HTTP response splitting, session fixation, and cross-site scripting.
<b>XPATH Injection</b>	A technique similar to SQL injection but where XPATH semantics are injected to manipulate the XPATH query in order to retrieve data from XML.
<b>OS Command Injection</b>	A technique where operating system commands are inserted into the application which are then executed by the web application, allowing the attacker to control the underlying server.
<b>LDAP Injection</b>	A technique used to inject LDAP semantics of LDAP queries run by the application against an LDAP database. This type of attack is most common where user information is stored within an LDAP database.
<b>Server Side Script Injection</b>	Server-side script injection is where a user can inject source code (script) into the application. This source code is then executed as part of the application.

---

Encryption	<b>Unencrypted Data Communications</b>	Where sensitive information is transported over an unencrypted channel (e.g. HTTP and not HTTPS)
	<b>Weak SSL Ciphers</b>	Certain ciphers are potentially more susceptible than others against cryptanalysis attacks and are therefore support by servers is not recommended.
	<b>Weak SSL Protocols</b>	Certain SSL Protocols (v2) have insecurities associated within their design such that support by servers is no longer recommended

---



---

**Invalid Digital Certificate**

A number of elements on the certificate are checked for validity, including the date, time, certifying authority, signature algorithm and signature. If these elements are not valid then the users will always accept the security issues when visited the site and therefore will be more likely to fall for a man in the middle attack should an attacker attempt one against the application.

---

**Weak Cryptographic Algorithms**

This includes weaknesses with any hashing or cryptography used by the application particularly in storage. It also includes situations where no sensitive information is stored in clear text.

---



## Context Information Security

### London (HQ)

4th Floor  
30 Marsh Wall  
London E14 9TP  
United Kingdom

### Cheltenham

Corinth House  
117 Bath Road  
Cheltenham GL53 7LS  
United Kingdom

### Düsseldorf

Adersstrasse 28  
1. Obergeschoss  
D-40215 Düsseldorf  
Germany

### Melbourne

Level 9  
440 Collins Street  
Melbourne  
Australia