Paper VII: Vulnerabilities in E-Governments

Vulnerabilities in E-Governments

Vebjørn Moen, André N. Klingsheim,
Kent Inge Fagerland Simonsen, and Kjell Jørgen Hole

Abstract

This paper shows that more than 80% of the e-governments in the world are vulnerable to common Web-application attacks such as Cross Site Scripting and SQL injection. Industrialized countries were found to be more vulnerable than under-developed countries (90% vs. 50%). The paper also describes some malicious data mining possibilities on the Norwegian e-government, and how information can be combined and used to create other practical attacks.

1 Introduction

E-government is one of the buzzwords of the Internet age, referring to any government function or process that is conducted in a digital form over the Internet. Basically, one or several Web portals supply individuals and businesses with public information, government forms for download, and contact with government representatives. According to most e-government plans, providing services over the Internet will yield higher efficiency and quality, easier access, the possibility of offering individual services, and increased transparency, ultimately leading to a more efficient public sector [1].

An EU press release claims: "A new survey of e-government services prepared for the European Commission has found that EU citizens are saving 7 million hours a year on the time it takes to do their income tax returns, and EU firms are saving about €10 per transaction on their VAT returns by doing them online. Moreover, there is still huge scope for further savings" [2].

However, since e-government projects are provided over an insecure channel, namely the Internet, other important issues surface. In most countries there are no governmental infrastructure that supports authentication, confidentiality, and integrity. The Public Key Infrastructure (PKI) used in E-Commerce is not applicable to e-government without a thorough analysis of what the new trust model should be. The trust calculation for commerce is based on monetary issues, while government solutions involve important infrastructure, society, and privacy issues.

There are also other problems, related to Web solutions, that can give unforeseen consequences when e-government solutions are put forth. We have examined e-governments in the whole world for known Web application issues, and found that a vast majority of them have common vulnerabilities. As a special case study, we have also examined parts of the Norwegian effort to create an e-government and located possibilities for malicious data mining and denial-of-service attacks in the services offered.

Section 2 gives a short introduction to some common Web application vulnerabilities, Section 3 presents the results from the inquiry into the security of e-governments, Section 4 describes why the Social Security Number (SSN) should not be used for identification or authentication, and Section 5 concludes the paper.

2 Web application vulnerabilities

In this section we give a brief overview of the Web application vulnerabilities we have chosen, Cross Site Scripting (XSS) and Structured Query Language (SQL) Injection. These vulnerabilities were chosen because they have been well-known for years. XSS and SQL injection attacks are simple, yet powerful attacks against Web applications that can be carried out with nothing more than a Web browser. The implications of these attacks will be discussed in the following sections.

2.1 Cross Site Scripting

XSS [3, 4] is perhaps the most common Web application vulnerability, where the Web application reflects unvalidated user input on a dynamically generated Web page. This makes it possible to supply Javascripts and HTML code in the user input, which will then be part of the dynamically generated page. E.g., by using Javascript it is possible to steal cookies, and Cascading Style Sheets and the HTML tag *iframe* can be utilized to completely control both the content and the layout of the resulting page. Common attack vectors are search applications which reflect the search string, and parameters supplied in the URL.

Here is an example of an XSS attack string which generates a page with arbitrary content on an XSS vulnerable site:

```
http://VulnerableSite.com/search?q=<iframe style=height:100%;
width:100%;border:none;transparent:none;position:absolute;top:0;
left:0;z-index:100; src=http://AttackerSite.com/ />
```

The attack string can be URL encoded [5] so that the content is unreadable for the average Internet user. An attack is successful if a victim visits an URL containing the XSS attack. This can be achieved by e.g. spoofing an e-mail from a sender in which the user has trust.

2.2 SQL injection

SQL injection [3, 6] is considered to have more severe consequences than XSS, due to the fact that a successful SQL injection can compromise the integrity of a database. A Web application is vulnerable to SQL injection if unvalidated user input is used to generate SQL queries. The following example is a typical SQL query used to generate dynamic Web pages:

```
SELECT * FROM articles WHERE id='<user input>';
```

An attacker can control the user input, and, e.g., enter: `'; DROP 'articles';` This adds a second command to the SQL query, which then becomes: `SELECT * FROM articles WHERE id="; DROP 'articles';';`

These SQL commands will select some data, delete the table "articles" in the database, and then generate an SQL error due to the single quotation mark.

Continent	Only XSS	Only SQL	XSS and SQL	XSS or SQL	None
Africa (61)	14.75 (9)	0.00 (0)	34.43 (21)	49.18 (30)	50.82 (31)
Asia (55)	9.09 (5)	0.00 (0)	76.36 (42)	85.45 (47)	14.55 (8)
Europe (53)	7.55 (4)	0.00 (0)	83.02 (44)	90.57 (48)	9.43 (5)
North America (34)	20.59 (7)	2.94 (1)	52.94 (18)	76.47 (26)	23.53 (8)
Oceania (25)	24.00 (6)	0.00 (0)	28.00 (7)	52.00 (13)	48.00 (12)
South America (17)	17.65 (3)	0.00 (0)	52.94 (9)	70.59 (12)	29.41 (5)

Table 1: Percentages of vulnerabilities in e-governments for each continent. Number of countries enclosed in parenthesis. Note that some countries are counted for more than one continent, e.g. Russia belongs to both Europe and Asia.

In general, SQL injection gives an attacker the opportunity to manipulate the database and in special cases execute arbitrary code on the database server. It is therefore an effective attack on Web applications. Typical attack vectors are logins, search forms, and the URL of dynamically generated pages (e.g. `http://VulnerableSite.com/article?id=42` could result in a SQL query similar to the one in the example). SQL injection can be avoided through user input validation, ensuring appropriate handling of characters with a special meaning in SQL.

2.3 Combining information and data mining

Internet was created for information sharing, and e-governments will hopefully give more efficient services to the public. However, the Web is completely different from the more traditional information channels, such as radio, TV, and postal services. The fact that we have an up-channel gives rise to new attack scenarios which must be taken into account before the e-government concept can replace old government systems. In addition to the previously described attacks on e-government portals, other risks exist. Malicious data mining can be possible when huge databases containing governmental information about the public are made accessible through e-government Web applications.

3 Vulnerabilities in e-government Web portals

In order to investigate the security levels of e-government Web applications we have probed government Web sites for XSS and SQL injection vulnerabilities. An e-government Web application is considered to be all Web pages under a gov.<country code> sub domain, and/or pages from the government, ministries, and parliament.

In addition to searches on Google, the Google directory of countries [7] and Gunnar Anzinger's list of Governments on the WWW [8] were used to locate the e-government Web applications. The investigation was carried out during February and March, 2005.

3.1 Scope of vulnerabilities

In this paper, a Web application is considered vulnerable to XSS if we are able to insert script or HTML code on a dynamic Web page. Furthermore, a Web application is vulnerable to SQL injection if we are able to change the structure of an SQL query run by a dynamic Web page, causing an SQL statement error. Vulnerabilities are not examined further, since both ethical and legal issues then arise. In order to evaluate the severity of a vulnerability we would have to exploit it in the worst possible manner. Disruption of service, loss of data, or bringing the Web application to a complete halt could be the effect of such an investigation. Our mission was not to cause havoc on e-government Web sites, but rather to check the status of their Web application security. Hence, we were forced to use a simple definition of what a vulnerable Web application is.

We suggest defining a country as vulnerable if there exists one vulnerability in an e-government Web application. This simplification is done since there are millions of governmental Web pages on the Internet, so an exhaustive review of all available pages would be impossible without automated tools. Different countries of course have different numbers of vulnerabilities, and the vulnerabilities vary in their severity. In addition, many of the Web pages are subject to constant change, so vulnerabilities may come and go. Hence, an exhaustive review would never give exact results. We therefore argue that our approach is useful, confirming that there are problems in governmental Web applications and that harm may be done when armed with nothing more than a Web browser.

3.2 Methods for finding XSS and SQL vulnerabilities

There exist algorithms and programs that can check Web applications for common vulnerabilities, one example is [9], but we chose to check all the e-governments manually in order to completely control which pages and input vectors that were used.

The basic idea in finding XSS and SQL vulnerabilities is to look for input vectors that are used by some server side script. To test for XSS, an input such as `<script>alert('XSS')</script>` can be used. If this string is included in a dynamically generated page, a Javascript enabled browser will show a Javascript pop-up window containing the text `XSS`. To test for SQL injection vulnerabilities, a single quote (') can be used as input. The single quote is a special character in SQL, and if it is included in the SQL query it will most likely generate an SQL statement error.

To further ease the work of finding vulnerabilities, we used Google to search for file extensions we know are more likely to be vulnerable than others. Consider e.g. the Australian government, which have the gov.au domain. If we search for: `site:gov.au inurl:php` in Google, we will find pages under the gov.au domain generated using PHP. A common practice was to search for ASP or PHP pages, and then other dynamic pages if no vulnerable ASP or PHP pages were found.

3.3 Results

There are 244 entries on the ISO 3166 list [10], but some of these entries are for countries/territories governed by another country. In total, we were able to locate e-government sites for 212 countries, and 173 of these countries were

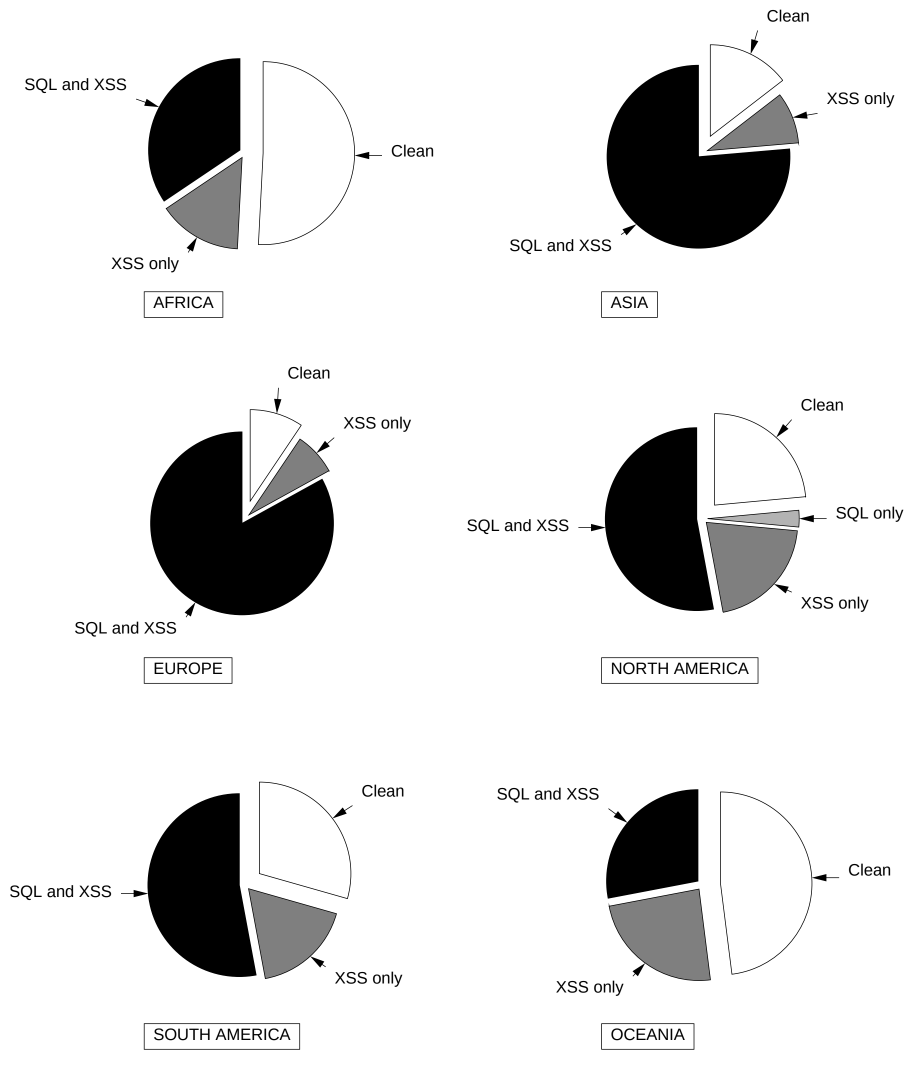


Figure 1: Pie charts showing vulnerabilities for each continent.

Country category	Only XSS	Only SQL	XSS and SQL	XSS or SQL	None
1st World (32)	6.25 (2)	0.00 (0)	90.63 (29)	96.88 (31)	3.12 (1)
2nd World (31)	9.68 (3)	0.00 (0)	80.64 (25)	90.32 (28)	9.68 (3)
3rd World (50)	18.00 (9)	0.00 (0)	32.00 (16)	50.00 (25)	50.00 (25)
G8 (8)	0.00 (0)	0.00 (0)	100.00 (8)	100.00 (8)	0.00 (0)

Table 2: Percentages of vulnerabilities in e-governments for different country categories. Number of countries enclosed in parenthesis. Note that not all of the 244 countries are included in the statistic for 1st, 2nd, and 3rd World, we used the countries listed on [11].

vulnerable to either XSS or SQL injection. That is; 81.6% of the countries with a Web portal were vulnerable to these simple attacks.

Table 1 gives the percentages of vulnerable countries for each continent. Europe have the highest ratio of countries vulnerable to either XSS or SQL injection, closely followed by Asia. More than 90% of European e-governments have a vulnerability, while slightly less than 50% of African e-governments are vulnerable. Figure 1 illustrates the numbers with pie charts.

The same tendency can be seen in Table 2, showing the percentages of vulnerable 1st, 2nd, and 3rd World countries, as well as the G8 countries. More than 90% of 1st and 2nd World countries are vulnerable, along with all of the G8 countries. However, we found SQL injection or XSS for "only" 50% of the 3rd World countries. These results are also shown as pie charts in Figure 2.

While testing the e-government portals we noticed that almost all of the industrialized countries had dynamic pages, based on technologies such as PHP, ASP, or JSP. We also noted that many of the dynamic pages retrieved parts of their content from databases. To our "disappointment" we only located static HTML pages for many African and 3rd World countries. Figure 3 shows the distribution of Web server software based on vulnerability type. Figure 4 shows which technologies were used to generate the vulnerable pages. However, the results are affected by the methods we used to find vulnerable pages, and they give a better picture of which technologies were most frequently used than of the security of any of the technologies.

3.4 "Invulnerable" sites

Some sites did not seem to have any vulnerabilities so we had to look at the characteristics of these sites. According to [12] there is a trinity of trouble which makes software difficult to control: complexity, extensibility, and connectivity. Therefore; our assumption is that sites with no discovered vulnerabilities must be smaller and less complex, and also use fewer technologies.

The 39 "invulnerable" countries had on average 8,126 pages indexed by Google, and 2.75 different technologies had been used to create the pages. In comparison; the G8 countries had an average of more than 12 million pages, and had pages created by 14–15 different technologies.

Our conclusion from these observations is that the "invulnerable" sites are not vulnerable because their complexity is low. The big industrial countries on the other hand use many different technologies to build enormous Web sites, hence increasing the possibility of introducing vulnerabilities in their Web applications.

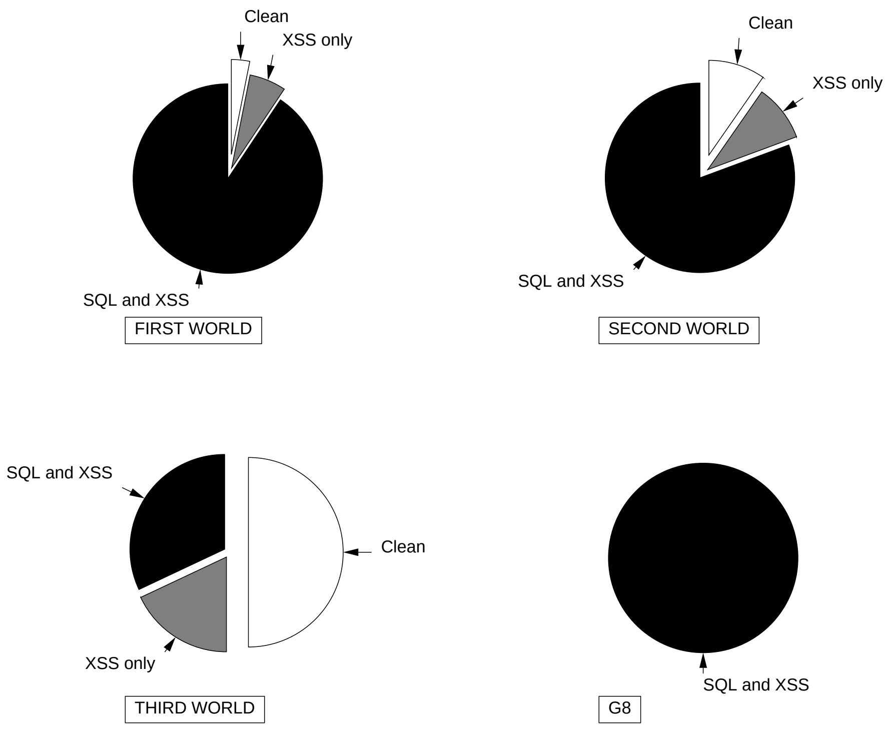


Figure 2: Pie charts showing vulnerabilities for the 1st, 2nd, and 3rd World countries, and also for the G8 countries.

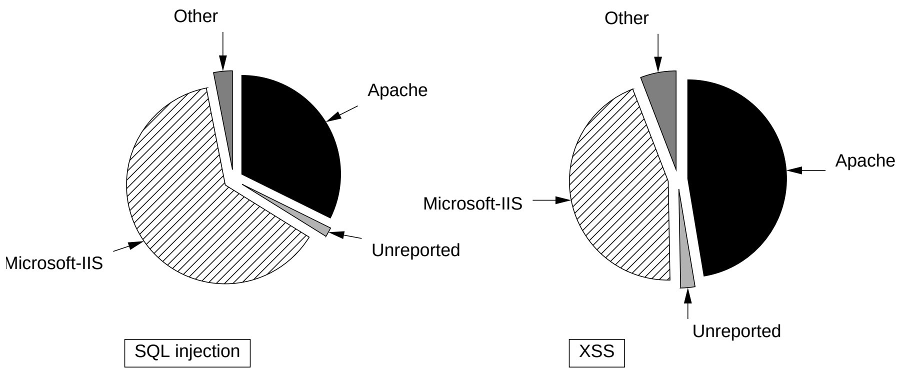
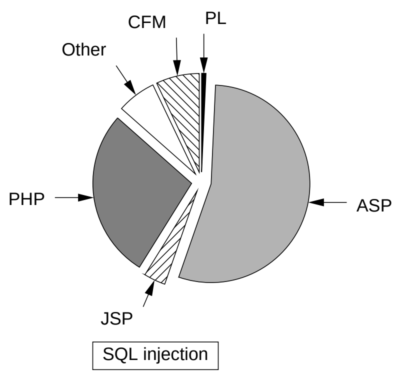


Figure 3: Pie charts for SQL injection and XSS, with regard to which Web server was serving the vulnerable portal. We used the 'Server' HTTP-header field in the HTTP reply to identify the servers, and consider the whole Apache family as Apache, and we do not distinguish between different versions of Microsoft-IIS. The 'Other' category includes Netscape-Enterprise, Oracle, Lotus-Domino, and Sun-ONE-Web-Server.



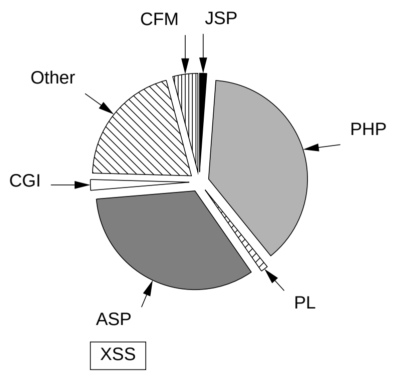


Figure 4: Pie charts showing which technologies generated the vulnerable pages, based on file-endings. We see that ASP and PHP dominates the statistics. The 'Other' category contains the pages we were unable to classify.

3.5 Defenses

Defending a site against these attacks requires input validation. Input can be validated on two different levels: either by the Web application or the Web server. The Web application can implement a function that parses all user input, handling dangerous characters/commands or rejecting the input. At the Web server level a solution such as the ModSecurity [13] module for the Apache server can be installed. This module can be used to validate all input before it is handed over to the Web application.

Successfully defending a site against SQL injection and XSS requires a constant focus on these problems. Whenever a new feature is added to an application there is a risk of introducing vulnerabilities. Even skilled and experienced programmers, who know about these attacks, make vulnerable applications. For a more detailed description of how to create more secure Web applications, [3] is highly recommended.

4 Collecting Social Security Numbers

SSNs are used around the world for purposes they were not initially intended to be used for. This has major implications. Problems arise when SSNs are used for any degree of authentication [14]. In order to investigate these issues further we decided to check how SSNs are used in Norway, and how much information we could obtain from governmental Web applications using our knowledge of the Norwegian SSN structure.

The structure of Norwegian SSNs is shown in Table 3 and described in [15]. Knowledge of the SSN structure makes it possible to generate all valid SSNs for any given day/month/year for both men and women. SSNs are not secret, but access to SSNs connected with personal information is restricted in Norway.

We have located several Web applications that allow us to filter valid SSN numbers belonging to real persons. E.g. all governmental employees are members of a pension fund. On the pension fund's Web site it is possible for members

The Norwegian SSNs consist of 11 digits: $x_1x_2x_3x_4x_5x_6i_1i_2i_3c_1c_2$	
$x_1x_2x_3x_4x_5x_6$	Birth date (ddmmyy)
$i_1i_2i_3$	Individual number. Highest available number for that day is used for each person. For persons born in 1855–1899 the possible numbers are 500–749, anyone born between 1900 and 1999 are given a individual number between 000 and 499, and finally for those who are born in 2000–2054 the numbers 500–999 are used. Girls have an even i_3 while boys have an odd i_3 value.
c_1c_2	Control digits. Used to detect every incorrect SSN with one number wrong or any mixing of two numbers $c_1 = 11 - (3x_1 + 7x_2 + 6x_3 + x_4 + 8x_5 + 9x_6 + 4i_1 + 5i_2 + 2i_3) \bmod 11$ $c_2 = 11 - (5x_1 + 4x_2 + 3x + 2x_4 + 7x_5 + 6x_6 + 5i_1 + 4i_2 + 3i_3 + 2c_1) \bmod 11$ if c_1 or c_2 is 10 mod 11, then the SSN is rejected and the next possible SSN is chosen.

Table 3: Structure of Norwegian SSNs.

to apply for loans, a process that uses SSNs for identification. By supplying an SSN belonging to a member, access is granted to a page displaying the member's full name, home address, and name and address of the member's workplace.

To test the malicious data mining possibilities in such a portal, we wrote a simple Python script with less than 30 lines of code, which collected SSNs and the corresponding names and addresses from the Web application. We started off with the birth dates of all the members of the Norwegian cabinet and used our script to find their SSNs. The script accomplished this by generating all possible SSNs for given dates, posting them to the portal and logging the answers. Note that such a script also makes it possible to build a database containing all the SSNs and names of all the members of the pension fund (approximately 1,000,000). We have notified the pension fund about these issues on several occasions.

It is possible to combine such an attack with information gathered on other portals. For example, the portal for the tax department allows users to enter their SSNs to get a new tax card, but it gives an error message if the entered SSN does not belong to anyone. This fact makes it possible to do a binary search to determine the range of SSNs that belongs to real persons for any given day since SSNs are assigned in chronological order. Consider all persons born in Norway on 01.01.2001. The first girl to be registered that day would be given SSN number 01010199952 (boy: 01010199871), the second girl would be given 01010199790 (boy: 01010199448—birth date 01.01.01 with individual number 996 does not exist since $c_2 = 10 \bmod 11$ and so on. By executing the previously mentioned binary search we can find the last number assigned for that day. Hence, we then know the whole range of SSNs assigned for that particular day.

4.1 Using SSNs to do mischief

Many portals use SSNs to identify visitors. However, since there is no established infrastructure supporting authentication, many services only "authenticate" users based on knowledge of a valid SSN and perhaps a name. There are several examples of this practice in Norwegian Web portals, e.g. you can order a new bank account, apply for loans, order a new tax card, and order a health card by entering a name and an SSN.

This allows several attacks; we can order/apply for a new tax card/health card for all Norwegians by using a simple script that logs on with every possible SSN number or any set of valid SSN numbers. Hence, we can affect service access for other users and create a lot of disturbance and manual work for governmental entities.

In 2003–2004 exact lists of assigned SSNs could be used to brute-force online bank accounts since several Norwegian Internet banks used SSNs for identification purposes. A description of such attacks can be found in [16].

5 Conclusions

The vast majority of dynamic e-government Web applications are vulnerable to XSS or SQL injection. XSS enables an attacker to create URLs to government sites that, if visited, will display arbitrary content controlled by the attacker. Such a malicious URL can further be used in Social Engineering attacks for the purpose of stealing passwords and cookies, or even spreading a false news story. SQL injection can potentially give an attacker the possibility to manipulate databases and in some cases execute arbitrary code on the database server.

These simple attacks have been known for several years, but still e-governments are vulnerable. More and more services are offered to the public through the portals, and more and more government databases are connected with these services, increasing the risk of malicious data mining, identity theft and compromising the integrity of the databases.

In addition to securing the Web applications, further steps are needed to secure the online services. E-government requires at least some kind of infrastructure to authenticate the users of the services, other than that of a username and password. A proper governmental PKI would make it possible to develop governmental Web applications implementing the required level of authentication and integrity.

References

[1] The Digital Task Force, “The danish egovernment strategy 2004-2006,” last visited: October 26, 2005. [Online]. Available: http://e.gov.dk/uploads/media/strategy_ pixi.pdf

[2] European Commission, “egovernment services yield real benfits for eu citizens and businesses,” last visited: October 26, 2005. [Online]. Available: http://europa.eu.int/rapid/pressReleasesAction.do?reference=IP/05/41&format=HTML&aged=0&language=EN&guiLanguage=en

[3] S. H. Huseby, *Innocent Code: a security wake-up call for Web programmers.* Wiley, 2004.

[4] CERT, “Cert advisory ca-2000-02 malicious html tags embedded in client web requests,” 2000. [Online]. Available: http://www.cert.org/advisories/CA-2000-02.html

[5] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)," RFC 1738 (Proposed Standard), Dec. 1994. [Online]. Available: http://www.ietf.org/rfc/rfc1738.txt

[6] C. Anley, "Advanced sql injection in sql server applications," A NGSSoftware Insight Security Research (NISR) Publication, last visited: October 26, 2005. [Online]. Available: http://www.nextgenss.com/papers/advanced_sql_injection.pdf

[7] Google, "Directory over countries," last visited: June 12th, 2006. [Online]. Available: http://directory.google.com/Top/Regional/Countries/

[8] G. Anzinger, "List over governments on the www," last visited: June 12th, 2006. [Online]. Available: http://www.gksoft.com/govt/en/

[9] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo, "Securing web application code by static analysis and runtime protection," in *WWW '04: Proceedings of the 13th international conference on World Wide Web*. New York, NY, USA: ACM Press, 2004, pp. 40–52.

[10] ISO, "ISO 3166 country code lists," last visited: October 26, 2005. [Online]. Available: http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/index.html

[11] Nationsonline.org, "First, second and third world," last visited: October 26, 2005. [Online]. Available: http://www.nationsonline.org/oneworld/third_world_countries.htm

[12] G. Hoglund and G. McGraw, *Exploiting Software: How to Break Code*. Addison-Wesley, 2004.

[13] Modsecurity, "Open source web application firewall," last visited: October 26, 2005. [Online]. Available: http://www.modsecurity.org/

[14] C. J. Hoofnagle and E. Mierzwinski, "U.S. PIRG answers to Chairman Shaw's questions on social security number privacy," 2004, last visited: June 12th, 2006. [Online]. Available: http://www.epic.org/privacy/ssn/ssnanswers7.2.04.html

[15] E. Selmer, "Personnummerering i Norge: Litt anvendt tallteori og psykologi," *Nordisk matematisk tidskrift*, 1964.

[16] K. J. Hole, V. Moen, and T. Tjøstheim, "Case study: Online banking security." *IEEE Security & Privacy*, vol. 4, no. 2, pp. 14–20, 2006.