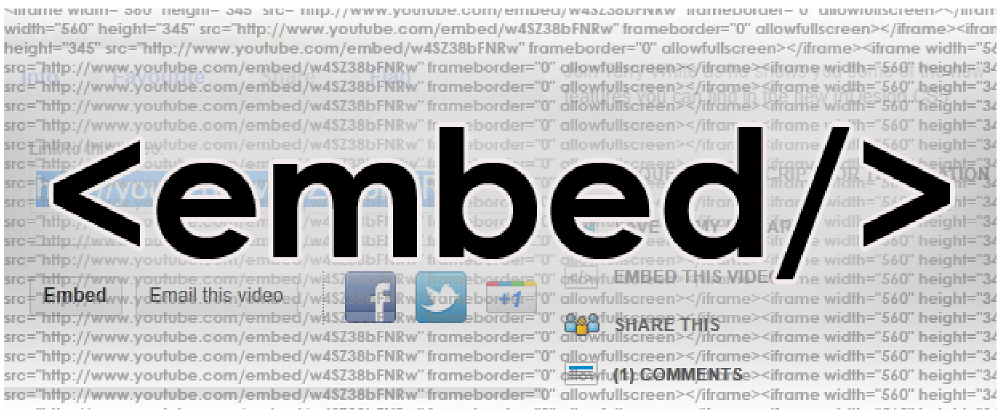
Embedding of External Content from Non-trusted Sources

Research Project II



Author:

Alexandre Miguel Ferreira
alexandre.miguelferreira@os3.nl

Supervisor:

Steven Raspe
steven.raspe@nl.abnamro.com

8 August 2012

Universiteit van Amsterdam

Master System & Network Engineering

Contents

Abstract

On many large websites the content of a website is not provided by a single server or even a single company. It is becoming more and more common to have websites republishing content from third parties, such as search engines or advertising networks. Some of these parties are reliable but other sources might not have the same level of security as an internet banking system. Using these sources may bring severe security problems. Examples are republished content running malicious scripts which can expose the visitors to cross site request forgery (CSRF), cross-site scripting (XSS) or phishing attacks, among others. This study gives an insight on how content provided from non-trusted sources can be securely embed on a website.

Chapter 1

Introduction

Inevitably e-banking and e-commerce websites must be secure. People rely on them to make transactions, check their account information, change their personal information, do payments, etc. These companies have to ensure confidentiality, integrity and availability of the data and services to their customers. The inclusion of content coming from third parties (e.g. partner advisers or social networks) can be a big threat to the trust that the users have in these websites.

Moreover when we reer to security, we cannot forget cybercrime and forensics are just around the corner. Having non-trusted content embed on a website can also be considered a global threat. The importance of having content embedded on a secure way grows and grows everyday. These threats can only be exceeded making sure that this content is coming from trusted sources.

Other studies have been done, some related to web browsers protection against specific attacks [11] [10] [8] others related to server-side protections [9] [4]. However, none of them explain which are the best solutions to protect against the risks that embedded content from third parties can arise nor how much user intervention is required from these solutions.

1.1 Research Question

From the above reasoning, the research question for this project is:

How to securely embed content from non-trusted sources on a website?

To answer this question, the following sub-questions are answered along this project:

1. Which vulnerabilities have to be secured?
2. How do different browsers handle the problem?
3. What can be secured by the bank server?
4. How much user intervention is required for the different solutions?
5. How to create trusted content from untrusted content?
6. What can the bank do to secure third parties' servers?
7. What can be done to have a third party being considered trusted?

Additionally a reflection will be performed about the cybercrime and forensics' consequences that can rise from the insecure embedding of content.

1.2 Scope

In this project only web browsers built-in security and server-side protections will be considered. There are also client-side solutions, such as *Noxes*, a Microsoft Windows-based personal web firewall solution against XSS, or *RequestRodeo*, a client-side solution against CSRF attacks that identifies http requests that are suspicious to be CSRF attacks. These are valid alternative routes but will be disregarded.

1.3 Report Structure

The report is structured as following:

- Chapter 2 will reflect on details related to the possible attacks performed when including content from third parties, how to embed this content and the forensics methods that can be adhered to;
- Chapter 3 will describe the approach and the practical steps executed;
- Chapter 4 will present the results yielded from the followed approach;
- Chapter 5 will give a conclusion and will also reflect on the outcome of the results. The cybercrime and forensics' consequences will also be covered here.

Chapter 2

Background

In order to dive into the project, a literature study has been conducted to better understand the underlying technologies which prepare the foundation of the practical approach performed in Chapters 3 and 4.

Before start discussing the test methodologies and results, a brief identification of the tested vulnerabilities, an explanation about the target web applications and how third parties content can be included in a website will be given in this chapter.

2.1 Target Applications

The main targets of these project are e-Commerce (see section 2.1.1) and e-Banking (see section 2.1.2). The introduction of content from third parties on these websites is becoming more and more common which make these websites targets of innumerable attacks (see section 2.3).

2.1.1 e-Commerce

e-Commerce [19] or Electronic Commerce allows the buying and selling of products over the Internet. It is drawn on technologies such as electronic funds transfer, Internet marketing or automated data collection systems. It also facilitates the financing and payment aspects of business transactions.

2.1.2 e-Banking

e-Banking [20] or Internet Banking allows bank's customers to perform financial transactions. Among others, customers can check their bank account details, make payments to third parties or invest their money in new products. All these actions are done on a (ideally) secure website operated by the bank.

2.2 Inclusion of Content by Third Parties

The inclusion of content provided by third parties can be done by means of inline frames (Iframe) (see section 2.2.2) or scripts (see section 2.2.1). Both have advantages and disadvantages that will be discussed on this section.

2.2.1 Scripts

Script [14] is a program that can accompany an HTML document or be embedded directly in it. The scripts are executed on the client's machine when the document is loaded or when a link is activated. Scripts can extend HTML documents in highly active and interactive ways, such as accompany a form to process input as it is entered or be triggered by events that affect the document (e.g. loading, mouse movement, etc.), among others.
Cross-side scripting usually refers to the inclusion of a malicious script from an external website by an attacker.

2.2.2 Inline frames

Inline frames [16] or Iframe is an HTML document embedded inside another HTML document on a website (see image 2.2.2). Iframes behave as an inline images, however they can be configured independently from the nester HTML content, since they have their own Document Object Model (DOM).
Iframes are commonly used to perform attacks such as Clickjacking, where the user is requested to click an apparently legitimate button that will guide them to malicious coding or content, by an invisible iFrame.

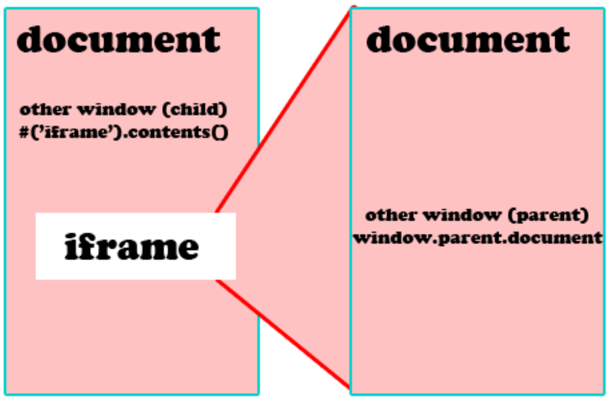


Figure 2.1: Iframe example

2.3 Possible Attacks

There are a lot of attacks performed against these web applications. The Open Web Application Security Project (OWASP) created a list, *OWASP Top 10 Project 2010* [13], where these attacks are explained. However, for this project, only three attacks were considered, they are Cross Site Scripting (XSS), Cross Site Request Forgery (CSRF) and Phishing. These vulnerabilities were chosen due to the fact they are the most relevant attacks performed against the techniques discussed in this paper (see section 2.1).

2.3.1 Cross Site Scripting (XSS)

XSS flaws [18], as they are known, allow attackers to execute malicious JavaScript code, pretending that the application is sending the code to the user. When a website is vulnerable to cross site scripting, an attacker is able to execute scripts in the victims' browser which can be used to hijack users sessions, spoil web sites or possibly introduce worms, among others. It was ranked number 2 on the *OWASP Top 10 Project 2010.*

2.3.2 Cross Site Request Forgery (CSRF)

A CSRF attack [17], also known as one-click attack, allows an attacker to send requests on behalf of a client without knowledge or interaction from the client. Subsequently the attacker can force the victims' browser to perform a hostile action, benefiting from this. These attacks can be as powerful as the web application that is being attacked. It reaches the position number 5 on the *OWASP Top 10 Project 2010.*

2.3.3 Phishing

Phishing [21], is a good example of social engineering. It is an attack where the attacker attempts to obtain informations about the user, such as usernames, passwords and in this specific case credit card details, by misleading the user. This is done by masquerading as a trustworthy entity, such as a bank website.

2.4 Cybercrime and Forensics

Cybercrime and Forensics is a continuously evolving topic that addresses the increasing use of computer technology in criminal activity. Attacks on computers occur more frequently than ever before and such attacks can be time-consuming and expensive to recover from. The possible suspects are rich in technical skills and have so much knowledge about technology as the technology's creators, therefore they know how to use technology against technology.

Chapter 3

Approach

In this chapter the approach taken and the testing methodologies followed during the project are discussed.

The research question, and consequent sub-questions, arise two main difficulties. First, which vulnerabilities are the most dangerous and important to be protected against and secondly, what can be done to be successfully protected against these attacks. That can only be solved by creating a testing environment to evaluate the possible solutions.

3.1 Target Website

The approach to this subject (see section 1.1) was to simulate an e-banking website with some flaws. These flaws were based on the biggest threats against e-commerce and e-banking websites. This way the selected attacks for these tests were cross-site scripting, cross-site request forgery and phishing (as explained on section 2.3).

To embed the content on the simulated e-banking website iframes have been chosen. Unlike scripts, iframes do not have access to the Document Object Model (DOM) [24] of the parent page, the representation of HTML document layout, which make them more secure than the use of scripts.

3.2 Testing Methods

The e-banking website created consisted only of a simple login page, username and password forms, and three iframes. Through these iframes three different the attacks were performed against the simulate e-banking website. One iframe with a cross-site scripting attack, other with a cross-site request forgery attack and another with a phishing attack.

3.2.1 Cross-site Scripting Attack

As said above, to recreate the XSS attack an iframe has been embedded on the website. This iframe had a script (see listing 3.1) running. This script does a session hijacking by stealing the cookies stored with the username and password informations, every time the user accesses the website, i.e. every time the user does a valid login.

```
[...]
<script>
  document.domain = "http://melon.studlab.os3.nl/";
  document.write('http://melon.studlab.os3.nl/homebanking.html?'+escape
      (document.cookie));
</script>
[...]
```

Listing 3.1: XSS attack

3.2.2 Cross-site Request Forgery Attack

As the XSS attack, also the cross-site request Forgery attack has been performed by including an iframe. In this iframe is presented to the costumer a link that, when clicked, will do a POST request with a new transaction on behalf of the user (see listing 3.2).

```
[...]
<form
  action="http://melon.studlab.os3.nl/homebanking.php"
  method="post" id="data">
  <input type="hidden" name="value" value="100000"/>
</form>

<script>
  document.getElementById('data').submit();
</script>
[...]
```

Listing 3.2: CSRF attack

3.2.3 Phishing Attack

The third iframe included on the website add a phishing attack. This attack is performed by using some social engineering skills. This way a form is presented to the costumer requiring him/her to change his/her password.

Since this is a social engineering attack no examples will be provided, however, on a “real world” phishing attack, the attacking website looks like a legitimate organization site, doing it by mimicking the HTML code.

Chapter 4

Possible Solutions

To make sure that the content is securely embed on a website various solutions have been tested. Web browsers' security (see section 4.1), server-side solutions (see section 4.2) and the use of automated scanners (see section 4.3) will be discussed on this chapter.

4.1 Web Browsers' Security

Nowadays, web browsers already have some built-in security features [24]. In order to check how reliable these features are, three web browsers were tested. They are *Firefox*, *Google Chrome* and *Internet Explorer 8*. These three web browsers were chosen due to the fact that, alike the rest of the world [15], the majority of *ABN AMRO* customers use them.

For an easier comparison between the various web browsers security features table 4.1 has been created. For an individual analysis refer to the correspondent section.

Web browser/Attack	**XSS**	**CSRF**	**Phishing**
Firefox (see section 4.1.1)	Same-origin policy protection	Use of add-ons: - CsFire* - RequestPolicy* - NoScript*	Phishing Protection feature*
Google Chrome (see section 4.1.2)	Same-origin policy protection	HTML5 JavaScript Sandbox	Enable phishing and malware protection option*
Internet Explorer 8 (see section 4.1.3)	Same-origin policy protection		SmartScreen Filter*

***** User intervention required

Table 4.1: Table with comparison on how the different tested web browsers protect against the tested attacks

4.1.1 Firefox

Firefox is a free and open source web browser used by 34.4% of the world's population (data from June, 2012 [15]).

As shown on the above table, Firefox[1] provides solutions for all the tested attacks.

Provide Solutions Against Cross-site Scripting

Firefox uses the *same-origin policy* [22] to protect against cross-site scripting. This feature allows scripts running on pages with the same origin to have access to their methods and properties (with no restrictions), however pages from different sites have this access prevented.

Provided Solutions Against Cross-site Request Forgery

Related to cross-site request forgery, Firefox makes use of add-ons to prevent it. Add-ons such as *RequestPolicy*, *NoScript* or *CsFire* [5] are used. Unfortunately these methods require user intervention. Apart from this fact, these add-ons can interfere with the normal operation of many websites.

Provided Solutions Against Phishing

To prevent phishing attacks, Firefox has a *Phishing Protection* feature. This protection is based on known phishing sites. However it does not include zero-day attacks [23]. This means that attacks performed for the first time are not protected by this feature. Once again user intervention is required to have this feature enabled.

4.1.2 Google Chrome

Google Chrome is a freeware web browser developed by Google that uses a WebKit layout engine. It is the most used web browser worldwide having 41.7% of the world's population using it (figures from June, 2012 [15]).

Such as Firefox, also Google Chrome[2] provides solutions against all the attacks performed.

Provide Solutions Against Cross-site Scripting

As Firefox (see section 4.1.1), Google Chrome makes use of *same-origin policy* to secure cross-site scripting attacks.

[1]**Tested version:** Firefox 13.0.1
[2]**Tested version:** Google Chrome 18.0.1025.151

Provided Solutions Against Cross-site Request Forgery

In order to protect against cross-site request forgery, Google Chrome uses an *HTML5 JavaScript Sandbox* [2]. This sandbox allows control over what can be executed within an iframe (see section 2.2.2), such as scripts or use of forms. Although it is used to protect against cross-site request forgery, there are some reports [1] about how it can help working around other attacks such as Clickjaking and Phishing.

Provided Solutions Against Phishing

To protect against phishing, Google Chrome provides a *"Enable phishing and malware protection"* option that is enabled by default, however it can be considered that user intervention is required here too. This protections is based on a list of suspected malware-infected websites [7]. Zero-day attacks are considered by Chrome. It is done by analysing the content on the site and, if it seems suspicious, the user will be warned.

4.1.3 Internet Explorer 8

Internet Explorer 8 is a web browser developed by Microsoft. From the three tested web browser is the one with less people using it, around 17% (June, 2012 [15]) of the world's population.

Unlike the previous web browsers, solutions provided by Internet Explorer 8[3] have not been found for all the attacks tested.

Provide Solutions Against Cross-site Scripting

Such as the other two tested web browsers, also Internet Explorer 8 draws its protection against cross-site request forgery by means of the *same-origin policy.*

Provided Solutions Against Cross-site Request Forgery

As mention above, some solutions provided by Internet Explorer 8 have not been found. Security against cross-site request forgery could not be found. It does not mean these protections do not exist, they were just not found during the execution of this project.

Provided Solutions Against Phishing

Internet Explorer 8 protects against phishing recurring to the *SmartScreen Filter.* This filter is based on a list of spam emails. Alike Firefox this list does not include zero-day attacks and also here user intervention is required.

[3]**Tested version:** 8.0.6001.18702

4.2 Server-side Security

Being required user intervention to most of the web browser security solutions, which can arise other problems, some server-side security solutions have been investigated too. However, not all of the attacks have been tried to be protect server-side.

4.2.1 Cross-site Script Solutions Investigated

Due to the fact that all the tested web browsers handle cross-site scripting attacks well, there was no need to find server-side solutions. It is important to notice the fact that, although this attack is handle by the tested web browsers it does not mean that previous versions of the tested web browsers or other web browsers perform the same way.

4.2.2 Cross-site Request Forgery Solutions Investigated

In order to protect against cross-site request forgery on the server-side the use of *Filtering Proxies* or *Double submit identification* can be done.

Double Submit Identification

Double Submit is a variation of the token identification. With this option, for every single form that the user has to fill a token is generated and stored on a cookie. Finally a validation is made to make sure that the field value (on the server) matches the cookie value. This means that a form generated by the attacker will not have the same value (same policy protects against cookies stealing). (See section 4.1.1)

- *Apache mod_security* is a module that analyses client requests before they are processed by Apache and, furthermore, by analysing server responses after a request has been processed. It is also known as web application firewall. (See Figure 4.2.2)

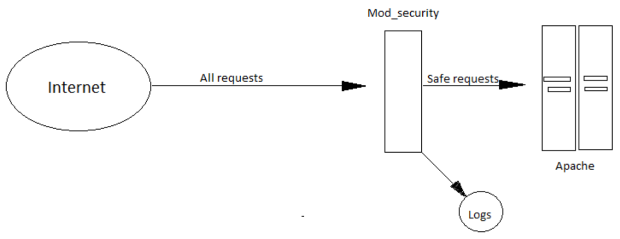


Figure 4.1: Apache mod_security example

Filtering Proxy

A proxy is a service that accepts requests made by web browsers for web pages. It requests and receives the requested pages from the respective web servers and finally transfers those pages to the web browser. A filtering proxy (see Figure 4.2.2) is a service that modifies either the requests or the returned information.

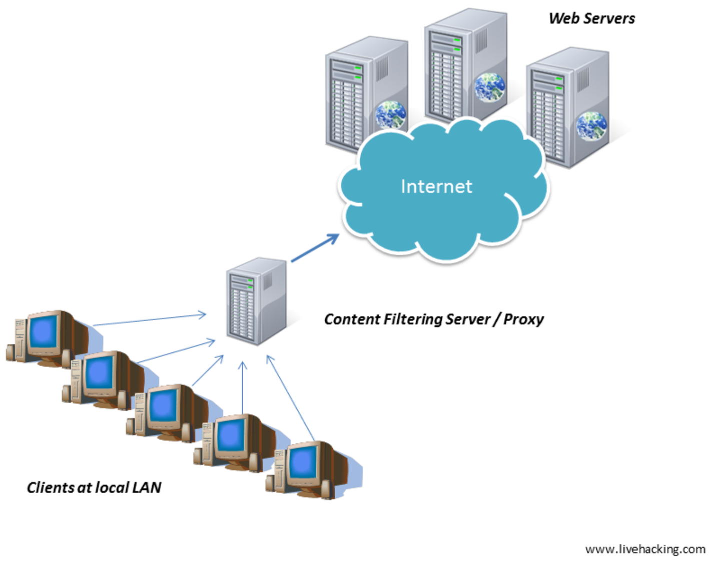


Figure 4.2: Filtering proxy example

Filtering proxies can be a good solution to protect against cross-site request forgery on the server-side. One application of this would be to deny access to a particular web server by simply not requesting pages from that server (based on the server's IP address or name).

4.2.3 Phishing

Concerning to Phishing, nothing can be server-side. Since the moment the user is misled by the attacker all the information sent will never reach the servers of the falsified entity. The only thing that can be done is alert the customers for the hazard of this kind of attacks.

4.3 Automated Scanners

Automated scanners, also known as penetration testing tools, are software developed to assist professionals during a penetration test. This could either mean tools that accomplished specific automated tasks or fully automated point-and-shoot solutions that, without human intervention, crawls the functionality in an application before trying to detect vulnerabilities.

The use of software has been considered, however the attacks previously mentioned (section 2.3) can be performed in such a way that the automated client can be mistaken, returning false-positive or false-negative results, and after all not detect the attacks [6].

Since automated scanners can be mistaken, their use can be rather considered problem detection instead of a protection.

Chapter 5

Conclusion

How to securely embed content from non-trusted sources on a website?

Throughout this research multiple sub-questions have been answered in order to conclude on the above question. It can be concluded that embed content from third parties in a secure way is possible.

Ideally all the vulnerabilities should be secured. Notwithstanding there are vulnerabilities that are more common and easier to explore than others, such as cross-site scripting, cross-site request forgery or Phishing. These vulnerabilities are largely related to Microsoft's security laws [12].

Law #1: If a bad guy can persuade you to run his program on your computer, it's not your computer any more.

5.1 Web Browsers Security & User Intervention

The web browsers tested had different solutions for the different problems explained in this project, however all of them use the same protection against cross-site scripting, the *same-origin policy.* It is important to mention the fact that, although the tested web browsers proved to be protected against cross-site scripting it does not mean that previous versions of these web browsers also use the same way to protect against these attacks. Also to notice that there are a big variety of cross-site scripting attacks that, unfortunately, were not tested during the execution of this project.

Related to cross-site request forgery, it can be conclude that it is difficult to be protected by the tested web browsers. While *Firefox* requires user intervention, *Google Chrome* makes use of a protection that can arise other problems and bring up different attacks. About *Internet Explorer 8* protections against this attack no solutions were found, but once again it does not mean they do not exist.

Talking about phishing, it can be conclude that this is probably the most difficult vulnerability to prevent. The majority of web browsers' features require user intervention and do not cover all the possibilities. Nevertheless Microsoft's security laws are applicable once again (with the appropriate differences).

Law #8: An out of date virus scanner is only marginally better than no virus scanner at all

5.2 Server-side protection & More Trustable Third Parties

Although cross-site scripting server-side protections were not tested, there are some solutions available, such as the one presented by Jovanovic et al. [9]

Concerning server-side solutions against cross-site request forgery, it is important to notice, once again, that not all the possible attacks have been tested, however the tested ones give a good indication that these kind of attacks can be protect by using filtering proxies or double submit tokens. The same protection methods used by to protect against these attacks can be used to protect third parties' servers. This way, and having third parties being audited by the embedding company, should be enough to consider third parties more trustable too.

In order to transform untrusted content into trusted content the use of automated scanners could be considered, although filtering proxies might do a better job since, as explained before, automated scanners can be eluded by the attackers.

Finally the server-side. Nothing can be done against phishing attacks, this social engineering attack cannot be protected, but can be alerted by making the costumer aware of this fact. And yet Microsoft's security laws prove to be right again.

Law #10: Technology is not a panacea

5.3 Cybercrime & Forensics Aspects

Cybercriminals continue to use deceptive social engineering techniques to undermine fragile new web users. Whether it is done via an email that appears to be from a trustworthy bank or by an advertisement or a link in an embed iframe, everything has been tried. The solutions proposed on this report can be important to refrain the attackers' thirst of new victims.

Having the entities logging every single request made to their servers and inspecting the content coming from the malicious sources can be really useful in a forensics investigation too. And as RFC3227 says [3] it has to be transparent:

The methods used to collect evidence should be transparent and reproducible. You should be prepared to reproduce precisely the methods you used, and have those methods tested by independent experts.

Chapter 6

Future Work

6.1 More Web Browsers Tested

During this project only Internet Explorer, Firefox and Chrome were tested (as explained before 4.1). It would be interesting to see how other web browsers, such as Safari, Opera or Android, behave under the same tested situations.

6.2 More Attacks Tested

As explained in section 2.3 only XSS, CSRF and Phishing attacks were tested, but there are other attacks such as Pharming and Man-in-the-Browser that can and should be tested due to their up growth and hazard.

6.3 Use of DNSsec

The use of DNSsec was not covered during the execution of this project. So further research on this field would be interesting.
However, as a mere opinion, the use of DNSsec would not influence the security of the website against iframe attacks, since the only thing that would happen here is the fact that the content of the iframe is secured by DNSsec, which does not mean the content is not malicious. This way we would only have malicious content secured by DNSsec.

Acknowledgments

I would like to thank my supervisor Steven Raspe for his time, effort and insight put into this research project. As well I would also like to Sander Vos for his valuable feedback and help when needed.

Bibliography

[1] Security in depth: Html5's @sandbox. `http://www.idontplaydarts.com/2011/05/clickjacking-and-phishing-with-help-from-the-html5-javascript-sandbox/`, 2012. [Online; accessed 04-July-2012].

[2] Web app security. `http://blog.chromium.org/2010/05/security-in-depth-html5s-sandbox.html`, 2012. [Online; accessed 04-July-2012].

[3] Brezinski, D., and Killalea, T. RFC3227: Guidelines for Evidence Collection and Archiving. *RFC Editor United States* (2002).

[4] Burns, J. Cross site reference forgery: An introduction to a common web application weakness. *Security Partners, LLC* (2005).

[5] De Ryck, P., Desmet, L., Heyman, T., Piessens, F., and Joosen, W. Csfire: Transparent client-side mitigation of malicious cross-domain requests. *Engineering Secure Software and Systems* (2010), 18–34.

[6] Ferreira, A. M., and Kleppe, H. Effectiveness of Automated Application Penetration Testing Tools. `http://staff.science.uva.nl/~delaat/rp/2010-2011/p27/report.pdf`, 2011.

[7] Google. Phishing and malware detection. `https://support.google.com/chrome/bin/answer.py?hl=en&answer=99020&p=cpn_safe_browsing`, 2012. [Online; accessed 11-July-2012].

[8] Jim, T., Swamy, N., and Hicks, M. Defeating script injection attacks with browser-enforced embedded policies. In *Proceedings of the 16th international conference on World Wide Web* (2007), ACM, pp. 601–610.

[9] Jovanovic, N., Kirda, E., and Kruegel, C. Preventing cross site request forgery attacks. In *Securecomm and Workshops, 2006* (2006), IEEE, pp. 1–10.

[10] Louw, M., and Venkatakrishnan, V. Blueprint: Robust prevention of cross-site scripting attacks for existing browsers. In *Security and Privacy, 2009 30th IEEE Symposium on* (2009), IEEE, pp. 331–346.

[11] Maes, W., Heyman, T., Desmet, L., and Joosen, W. Browser protection against cross-site request forgery. In *Proceedings of the first ACM workshop on Secure execution of untrusted code* (2009), ACM, pp. 3–10.

[12] Microsoft. 10 Immutable Laws of Security. `http://technet.microsoft.com/library/cc722487.aspx`, 2011. [Online; accessed 11-July-2012].

[13] OWASP. Open Web Application Security Project OWASP: OWASP Top Ten Project. `http://www.owasp.org/index.php/Top_10_2010-Main`, 2010. [Online; accessed 12-June-2012].

[14] W3C. Scripts. `http://www.w3.org/TR/html4/interact/scripts.html`, 2012. [Online; accessed 25-June-2012].

[15] w3schools. Browser Statistics. `http://www.w3schools.com/browsers/browsers_stats.asp`, 2012. [Online; accessed 12-July-2012].

[16] WhatIs.com. Iframe (Inline Frame). `http://whatis.techtarget.com/definition/IFrame-Inline-Frame`, 2012. [Online; accessed 25-June-2012].

[17] Wikipedia. Cross-site request forgery — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Cross-site_request_forgery`, 2012. [Online; accessed 25-June-2012].

[18] Wikipedia. Cross-site scripting — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Cross-site_scripting`, 2012. [Online; accessed 25-June-2012].

[19] Wikipedia. Electronic commerce — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Electronic_commerce`, 2012. [Online; accessed 30-June-2012].

[20] Wikipedia. Internet banking — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Internet_banking`, 2012. [Online; accessed 30-June-2012].

[21] Wikipedia. Phishing — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Phishing`, 2012. [Online; accessed 25-June-2012].

[22] Wikipedia. Same origin policy — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Same_origin_policy`, 2012. [Online; accessed 02-July-2012].

[23] Wikipedia. Zero-day attack — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Zero-day_attack`, 2012. [Online; accessed 04-July-2012].

[24] Zalewski, M. Browser Security Handbook. `http://code.google.com/p/browsersec/`, Undefined. [Online; accessed 28-May-2012].