MVC Architecture enables rapid and secure Zend Framework Web based Application Development

Vladislavs Marisevs

School of Science & Computing
Galway-Mayo Institute of Technology, Ireland.

Introduction

Modern software development world has several design patterns, one of them is *Model View Controller* it is also called as *Presentation Abstraction Control* (or *PAC*) (Coutaz 1987). The key idea of this pattern is to divide the systems architecture into three different layers. The following is an extract from Wang's publication:

> The pattern isolates "domain logic" (the application logic for the user) from the user interface (input and presentation), permitting independent development, testing and maintenance of each (separation of concerns).
>
> (Wang 2011)

According to Guanhua Wang (Wang 2011) *MVC* is not a new software architecture design concept; it was described in 1979, by Trygve Reenskaug, then working on Smalltalk at Xerox PARC.

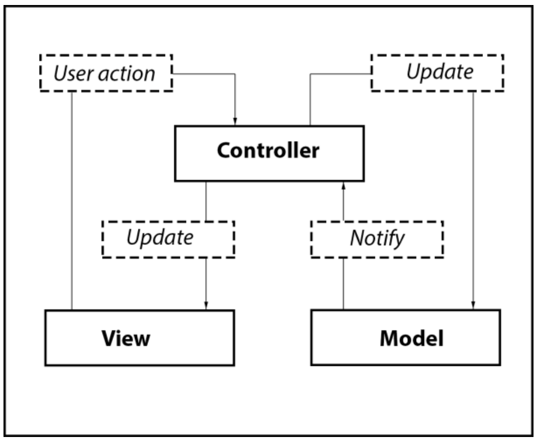


Figure 1. Mechanism of MVC

Model View Controller design divides a development process so that each part can be developed, maintained and tested independently without affecting other parts. The Model manages the application data and the core functionality. The Views displays the user interface and represents program's data. And Controller interprets the user inputs and handles events for views. This structure is shown in Figure 1. The application's front end can be modified without changing business logic. It also can handle different interfaces, like tabular information representation, charts or graphs. Also user's access permissions may vary, and each of access groups could have their own view.

According to Selfa's, Carrillo's and del Rocio Boone's publication (Selfa 2006) MVC's advantages are:

- Less coupling
- Higher cohesion
- Views provide high flexibility
- More design clarity
- Large scalability

According to example taken from Leff's and Rayfield's publication, Web applications also can be designed using Model View Controller architecture:

> "Java approach uses Entity Enterprise JavaBeans as the Model, constructs the View through HTML and JavaServer Pages, and implements the Controllers through Servlets and Session EJBs."
>
> (Leff, Rayfield 2001)

But this literature review will be concentrated on PHP programming format with the MVC architecture. By analysing the statistic report posted by Craig Buckler (Buckler 2015) it is easy to determine, that PHP is in the top 5 category. Moreover it has a good position as the web scripting language. This programming language went through the transformation since it was created in 1994. After 3.0 upgrade, PHP became an Object Oriented Programming language. This paradigm uses data structures and methods, it allows to use features like abstraction, encapsulation, polymorphism and inheritance. OOP side of PHP programming combined with Model View Controller structure improved code efficiency and reusability, but it also have a defect that Guanhua Wang is describing in his publication (Wang 2011) it is lack in operating system efficiency.

To prove this, Wang was using the programs Intel core2 Duo 2.00GHz, Apache 2.2.12, PHP 5.3.0, Chrome 7.0.517.44 Browser. Traditional PHP is a plain HTML page with combination of PHP scripts that are executed on the server side, so user would receive dynamically generated HTML formatted page. The MVC framework was Zend Framework 1.10 (ZF), it was preloading all default functions. And the last test was based on Lightweight MVC-like, this application doesn't have any of default validations or security against hacking attacks. The result of this page request test is shown in Table 1.

TABLE I. PAGE REQUEST TIME

Traditional	**MVC**	**Lightweight MVC-like**
24ms	174ms	46ms

According to the table above we can determine that ZF response time is slower than lightweight MVC-like and traditional PHP format. However, the provided functionality of ZF should be taken into account.

Body

The fast growth of the Internet also increases the amount of information available. Tim Berners-Lee invented the *World Wide Web* in 1989. Based on Vikas K. Malviya et al. (Malviya, Saurav, Gupta et al. 2013) publication, the motivation of *WWW* was to share information between scientists. With the growth and commercialization of *World Wide Web*, plain *HTML (Hypertext Markup Language)* was very limited. New scripting languages were invented to build dynamic websites. On the server-side languages like *PHP, ASP, JSP* and client side *JavaScript*.

Communication between web browser and server is implemented using HTTP (Hypertext Transfer Protocol). When user sends a request to a web server browser receives *HTML* page as response, it also can receive some scripts that will bring dynamic behaviour. Won Kim in the publication "The dark side of the Internet: Attacks, costs and responses" states that web technologies were invented to improve the world, but with threats such as malware, hacking, denial of service attacks, invasion of privacy, etc started taking place. As from Mukesh Kumar Gupta publication:

> "In 2013, Open Web Application Security Project (OWASP) and Common Vulnerabilities and Exposures (CWE) reported Cross-Site Scripting (XSS) as one of the most serious vulnerability in web applications."
>
> (Gupta et al. 2015)

Cross-site Scripting attack Definition and Classification II

Web application's vulnerabilities are the main reason behind the Cross-site Scripting attacks. Weak or lack of validation of user input data provides the chances of XSS (Malviya, Saurav, Gupta et al. 2013). This application gap allows hackers to insert malicious scripts, that browser couldn't identify as a malware and will execute. In this case scenario attacker can hijack victim’s session, cookie, deface websites, insert hostile content or conduct phishing attacks (Wang et al. 2011). Browser can't identify that this malware code is not a part of original page, because it is comes as HTTP response from origin server. There are 4 types of XSS attacks:

Stored or Persistent Cross-site Scripting Attacks

Bazara Barry (Elhakeem, Barry 2013) argues that Stored XSS is the most powerful kind of Cross-site scripting attack. This attack stores malicious script, which has been injected in the database as forum's message, visitor log, comment field or any other text type field. Victim receives the code when tries to access page with attacker's comment or message page.

Figure 2 illustrates the sequence how stored Cross-site Scripting attack is performed; this example is taken from Mukesh Kumar Gupta papers (Gupta et al. 2015).

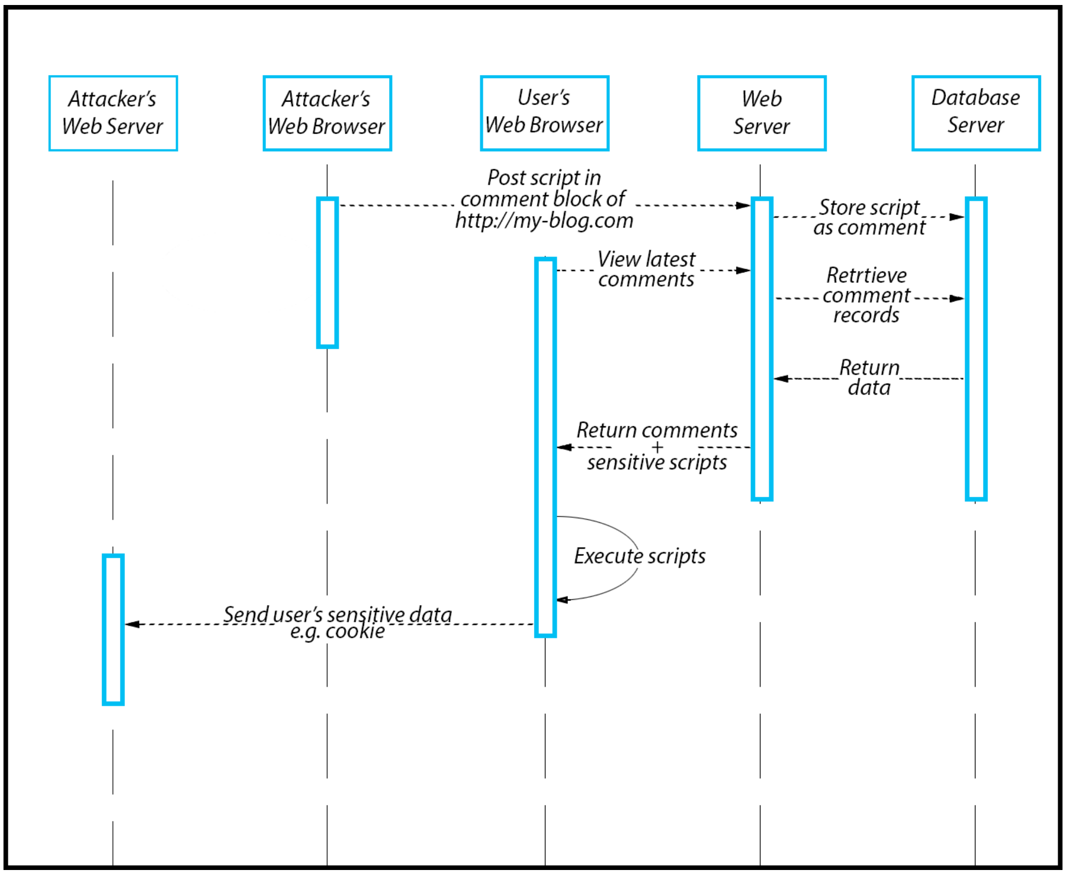


Figure 2. Sequence Diagram to Represent Stored XSS Attack Scenario

The attacker uses a blog website with vulnerability in the input type field, and he passes the malware scripts in the comment box. The comment box doesn't strip the tags and stores this script in the database. When legitimate-user tries to view comments and his browser requests the page. It receives malicious scripts as response and browser processes them like a part of the page. This piece of code can send HTTP request with sensitive user data (e.g. session id, cookie) to a malicious-user's server.

Reflected or Non-persistent Cross-site Scripting Attacks

Reflected XSS attacks code is not stored on the web server. In this type of XSS attacks malicious links are sent to victims using email or embedding the link in hacker's web server page (Malviya, Saurav, Gupta et al. 2013).

Figure 3 illustrates the sequence how malicious-user performs their attack on legitimate-user. Hacker is looking for a web application which returns input without proper validation as an error message, search result or any other response. Using this link, attacker crafts his own link by inserting some malware scripts that will post HTTP request into hacker’s server. And this crafted link will be shared with victims via email, ad or etc.

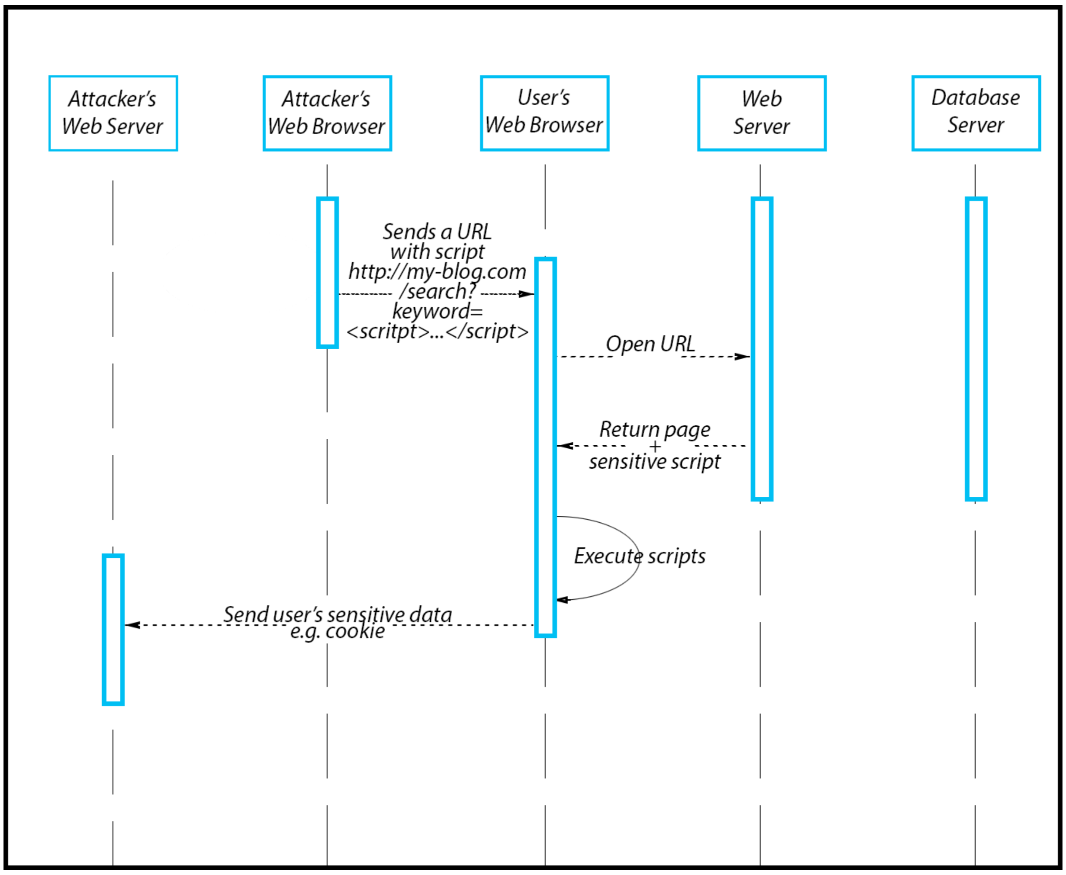

Attacker’s Web Server
Attacker’s Web Browser
User’s Web Browser
Web Server
Database Server
Sends a URL with script http://my-blog.com /search? keyword= <scritpt>...</script>
Open URL
Return page + sensitive script
Execute scripts
Send user’s sensitive data e.g. cookie

Figure 3. Sequence Diagram to Represent Reflected XSS Attack Scenario

DOM-based Cross-site Scripting Attack

XSS vulnerability occurs when client access malicious-user's crafted link, which has invalidated input that dynamically, obtained from *DOM* structure (Gupta et al. 2015). Di Paola and Fedon have found a vulnerability in the browsers that is described in their publication (Di Paola, Fedon 2006). Victim's browser allowed hijacking information, when they requested link with malicious script. For example *JavaScript* code that can be executed in the browser *"http://site.com/file.pdf#FDF=javascript:alert(“Test Alert”)"*, taken from Di Paola pages (Di Paola, Fedon 2006).

Induced Cross-site Scripting Attacks

According to Malviya papers this attack can take a place only when web application has *HTTP Response Splitting vulnerability* (Malviya, Saurav, Gupta et al. 2013). In this case attacker can manipulate the *HTTP header* of the server's response.

Let’s say the request for *index* page returns in a *302 redirect* (*HTTP/1.1 302 Moved Temporarily*) to http://www.abc.com/index.php?lang=en. Another user wants to display the page in German and he changes the language, sends a request and receives *302 redirect* http://www.abc.com/index.php?lang=german . Comparing these two responses we can determine that only parameter *lang* has been changed, it is a target property for an attacker using *HTTP Response Splitting*. Malicious-user makes a link which contains 2 responses:

http://www.abc.com/index.php?lang=german%0d%0aContent-Length:%200%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0aContent-Length:%2041%0d%0aHello, you have been phished

These responses separated by *%0d%0a* and the idea that he can infect the users behind his proxy server. The first response is mapped to first request, but second response hangs as there is no matching request. When user tries to access this page, he gets a response from proxy server, because proxy caches responses for requests often made. This example is taken from Arvind Doraiswamy publication (Doraiswamy 2011)

Traditional web proxy principles III

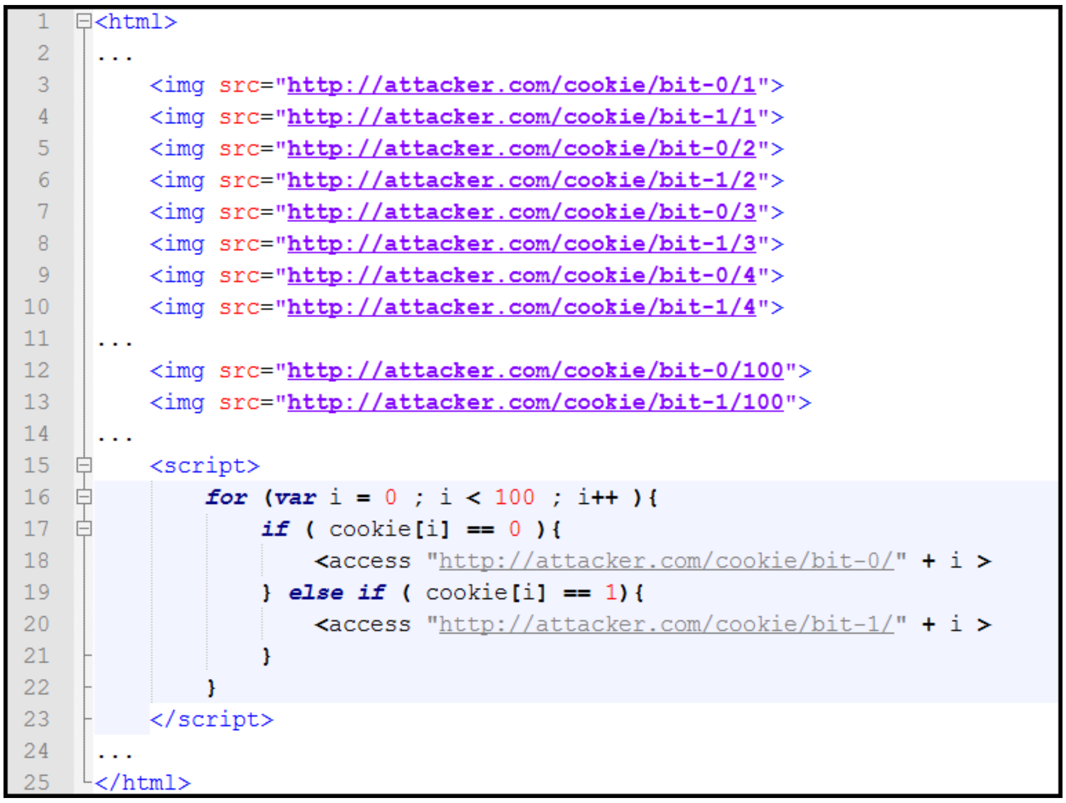
Ding Lan states that:

> "a traditional Web proxy provides the user an additional protection that unsuported by the existing personal firewalls, which allows users to control all the connections related with the browsers."
>
> (Lan et al. 2013)

After in-depth analysis by security experts, they found a solution, that the attacker couldn't use a static link for stealing the user's sensitive information, but only dynamically generated links could be used to submit data to third party domain (Lan et al. 2013). In this case Proxy will have a table with rules, where it temporary stores all static links that can be accessed by user without warning. Dynamically generated links should not be stored in the table, because they might allow transferring data to another party. Next section will discuss binary encoded Cross-site scripting attack, that has been taken from Ding Lan and Wu ShuTing publication (Lan et al. 2013).

Binary-encoded XSS attack

Embedded JavaScript in Web pages can be used also to edit *Document Object Model (DOM)* nodes in *HTML* dynamically (Nadji, Saxena, Song et al. 2009). Figure 4 presents the example of pseudo code based binary-encoded Cross-site scripting attack (Lan et al. 2013).



```
<html>
...
    <img src="http://attacker.com/cookie/bit-0/1">
    <img src="http://attacker.com/cookie/bit-1/1">
    <img src="http://attacker.com/cookie/bit-0/2">
    <img src="http://attacker.com/cookie/bit-1/2">
    <img src="http://attacker.com/cookie/bit-0/3">
    <img src="http://attacker.com/cookie/bit-1/3">
    <img src="http://attacker.com/cookie/bit-0/4">
    <img src="http://attacker.com/cookie/bit-1/4">
...
    <img src="http://attacker.com/cookie/bit-0/100">
    <img src="http://attacker.com/cookie/bit-1/100">
...
    <script>
        for (var i = 0 ; i < 100 ; i++ ){
            if ( cookie[i] == 0 ){
                <access "http://attacker.com/cookie/bit-0/" + i >
            } else if ( cookie[i] == 1){
                <access "http://attacker.com/cookie/bit-1/" + i >
            }
        }
    </script>
...
</html>
```

Figure 4. The pseudo code of binary-encoded XSS attack for stealing the user's cookie

Assume that legitimate-user have a 100 bit size cookie, and attacker aiming to steal that. In this case hacker uses *DOM* functionality to modify *HTML* Web page to add static links and for 100 bits = 200 links so each bit can be represented as 1 or 0 and each of them are unique link. And attacker also specifies the loop which will run through cookie and each bit will be submitted via dynamic link. As we discussed earlier proxy will pass through only static links, but in this case they are equals. Attacker can reconstruct the cookie based on accessed links which identifies bit position and value (Lan et al. 2013).

Analysis

Based on Mukesh Kumar Gupta (Gupta et al. 2015) publication, they build a public *GIT repository* (Stivalet, Delaitre 2014) that contains a synthetic test case generator. This application contains 9408 PHP samples. 5600 are safe and 3808 unsafe that are categorised. Gupta argues that evaluation of the proposed methods are performed on this dataset, as it provides mostly all the cases required for Cross-site scripting vulnerability prediction (Gupta et al. 2015). It is a free open source dataset with PHP source code with their vulnerability labels.

To avoid XSS script vulnerabilities developer should follow *The Open Web Application Security Project (OWASP)* guidelines (Keary at al. 2015). Malviya states that following these approaches XSS vulnerabilities can be prevented; the difficulty with these approaches is that they are totally dependent on developers (Malviya, Saurav, Gupta et al. 2013). The Cross-site scripting vulnerabilities are caused by improperly sanitized user input and this input finally reaches one of the sensitive sinks (i.e. saved in database, or 'echo').

There are several nodes in the proposed security model (Elhakeem, Barry 2013).

1. One of them is Security Awareness and it is based on developer expertise because they either do not know what cross-site scripting is, or they do not take security issues into consideration while developing Web applications and websites states Bazara Barry (Elhakeem, Barry 2013).
2. Another node is Server Security that is very important. Because this is a key point where application connects with the rest of the Internet.
    - Web application files should be always stored on a separate partition or drive other than system files. If attacker could get an access to servers root directory, he could also get access to other files on this partition.
    - To secure Server application they should use mechanisms and protocols that are specialized in particular area. For example Pretty Good Privacy (PGP) is specialized software that provides privacy. Commonly used for authentication and encryption/decryption of messages to increase the protection. Or Secure Electronic Transaction (SET) is a set of encryption used to protect data. It was supported initially by MasterCard, Visa, Microsoft, and others (Elhakeem, Barry 2013).
3. Client Security is also very important. Because they should think, before accessing any untrusted links. Browser must be up to date and it must support the greatest possible security and privacy.
4. Developers should follow Design Guidelines to make their Web applications safe. To reduce the websites vulnerabilities and present a new and efficient way of coding, it was referred to as Web Application Framework states Barry (Elhakeem, Barry 2013).

> "Web Application Frameworks are groups of program libraries, components and tools organized in an architecture system which helps the developers to build a complex Web application projects."
>
> (Elhakeem, Barry 2013)

Zend Framework is an open source framework for developing Web applications and services. It was written on PHP and it is loosely coupled architecture, which allows developer to code each component independently and designed with Model View Controller structure. This MVC paradigm we discussed earlier in this paper.

Barry and Elhakeem (Elhakeem, Barry 2013) argues that Zend Framework enables simple, rapid and agile web application development process, and it also offers AJAX support to convert XML data into JSON format and integrates the most widely used APIs and Web Services of third-party companies such as Google, Microsoft, Amazon, Flickr and Yahoo. ZF provides many options for validation such as Dojo tools for validation and filtering of inputs.

This is quite flexible web application framework that brings open source modules into application. Nowadays web application URL doesn't specify the actual file on the server, and ZF is also designed like that. It has a runner file that is responsible for whole application and all business logic is loosely coupled to provide more efficient development.

Conclusion

Detailed definition of Model View Controller paradigm underlines why it is useful for agile development. This approach inherits more important advantages than lack in operating system efficiency.

This paper has well defined information about Cross-site scripting attacks including examples combined from different sources. The publication concentrates on XSS attacks and how they are dangerous for legitimate-user that their personal information can be stolen. And usually it is under developers' responsibility to create a safe web application or service. To do that they must follow proposed security model and *The Open Web Application Security Project* standards to make it secure.

The proposed model uses an open source web application framework that adopts *Model View Controller* design pattern and allows using default filtering techniques for user input and it is called Zend Framework. Based on the tests described in introduction we can determine that it has longer latency comparing to Traditional PHP and Lightweight MVC-like, but the secure functionality that it is doing can cover that disadvantage. Various tests using XSS attacks on this model can certify its design quality.

References:

Buckler, C. (2015). "What's the Best Programming Language to Learn in 2015?". Viewed 1th November 2015 at *http://www.sitepoint.com/whats-best-programming-language-learn-2015/*

Coutaz, J. (1987). "PAC, an Object-Oriented Model for Dialog Design". Elsevier Science Publishers, Proceedings of Human-Computer Interaction - INTERACT, pp. 431-436, viewed 1th November 2015 at http://mvc.givan.se/papers/PAC_an_Object_Oriented_Model_for_Dialog_Design.pdf

Di Paola, S., G. Fedon (2006). "Subverting Ajax". pp. 1-8, viewed 5th November 2015 at https://events.ccc.de/congress/2006/Fahrplan/attachments/1158-Subverting_Ajax.pdf

Doraiswamy,A. (2011). "HTTP Response Splitting Attack". Viewed 5th November 2015 at http://resources.infosecinstitute.com/http-response-splitting-attack/

Elhakeem, Y. F. G. M., Bazara I. A. Barry (2013). "Developing a security model to protect websites from cross-site scripting attacks using ZEND framework application". pp. 624-629, viewed 5th November 2015 at http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=6634012

Gupta et al. (2015). "Predicting Cross-Site Scripting (XSS) security vulnerabilities in web applications". pp. 162-167 Viewed 4th November 2015 at http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=7219789

Keary at al. (2015). "XSS (Cross Site Scripting) Prevention Cheat Sheet". Viewed 6th November 2015 at https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

Lan et al. (2013). "Analysis and prevention for cross-site scripting attack based on encoding". pp. 102-105 Viewed 5th November 2015 at http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=6835463

Leff, A., J. T. Rayfield (2001). "Web-Application Development Using the ModelNiewlController Design Pattern", Enterprise Distributed Object Computing Conference, pp. 118-127 viewed 1th November 2015 at http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=950428

Malviya, V. K., S. Saurav, A. Gupta et al. (2013). "On Security Issues in Web Applications through Cross Site Scripting (XSS)" pp. 583-588 Viewed 4th November 2015 at http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=6805456

Nadji, Y., P. Saxena, D. Song et al. (2009). "Document Structure Integrity: A Robust Basis for Cross-site Scripting Defense". Viewed 6th November 2015 at http://www.cs.berkeley.edu/~dawnsong/papers/2009%20dsi-ndss09.pdf

Selfa, D. M., M. Carrillo, Ma. del Rocío Boone (2006). "A Database and Web Application Based on MVC Architecture", Proceedings of the 16th IEEE International Conference on Electronics, Communications and Computers, p. 48. Viewed 1th November 2015 at http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=1604744

Stivalet, Delaitre (2014). "Php vulnerabilities test suite". Viewed 6th November 2015 at https://github.com/stivalet/PHP-Vulnerability-test-suite

Wang, G. (2011), "Application of lightweight MVC-like structure in PHP",
IEEE Conference Publications, pp. 74-77 vol 2, viewed 1th November 2015 at
http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=5917846

Wang, Yi, Zhoujun Li, Tao Guo et al. (2011). "Program Slicing Stored XSS Bugs in Web Application" pp. 191-194
viewed 5th November 2015 at
http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=6041609