10/17/2017

Finn Turnbull

Dundee and angus college

Developing Websites

Assessment 1

# Introduction

This report will document and attempt to explain various aspects to developing websites for use on multiple platforms.

This includes but is not limited to potential security issues, the document object model, and the pros and cons of different ways to implement scripting.

## Document Object Model

The DOM or document object model is a W3C specification (Robbie, Wood, & Hégaret, 2004) designed as an API, ie ‘*a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service*’. (What is API, 2017)  
The purpose of DOM is quite well described in its name, in that it gives us a way to visualize and map elements in an html page as objects (or nodes) in a tree.

These objects can subsequently be accessed and modified using potentially any language however by far the most frequently used is JavaScript and offshoots such as jQuery, as well as other scripting languages. (Introduction to the DOM, 2017)  
An example of a piece of JavaScript code accessing the DOM is ‘*document.getElementByID()*’.

Although all popular browsers adhere to the W3C specification of the DOM there can be differences between how some things are interpreted by each browser, which can cause compatibility issues when developing for multiple browsers.  
One such example is the way the popular browsers interpret white space – Firefox treats this white space as an individual node, the same as it would an HTML tag, however IE does not.

Not only this but older versions of browsers can often not support the same functionality as the newer more up to date versions, which when considering backwards compatibility can cause even more issues.  
That said, there are resources that do an excellent job of documenting and testing these differences across different browsers and their versions.

## Security

There are countless ways in which both the developers and the users of web apps can help secure sensitive information.  
For the user their options can be limited – of course they can use common sense, selecting strong passwords, not storing unnecessary personal data, not sending sensitive information over public of insecure connections, but sometimes the users must rely on good development.

So, what can developers do?  
A common and relatively easy to implement step is hashing and salting user passwords, so that the passwords themselves are never actually stored on the database – this means that if an attacker was to get hold of the table data containing the passwords, it’s far from a simple task to then make use of that data.

On that subject then, how could an attacker get hold of sensitive data out of a web apps’ database?  
One of the most common problems even to this day is SQL injection attacks – where a user can insert SQL code into a seemingly harmless form (perhaps a search for some stock in an inventory) and as a result potentially get the web app to print out table data that was never meant to be retrieved.  
To help prevent this, input validation can be used. Either whitelisting – where the input type is restricted to a specific form, or blacklisting, where specific inputs result in an error.

(Cairns & Somerfield, 2017)

## Client Side and Server Side Scripting

Client side scripting is interpreted, you guessed it, on the clients’ machine. This means when a web page is downloaded the all the source code is loaded onto the clients’ computer and subsequently executed by the browser.

As a result, the speed of execution of client side scripts relies entirely on the clients’ machine.

Purely using client side scripting can cause some security issues however, as it is then potentially possible for the user to edit the script once it is loaded on their machine.

Server side scripting is interpreted, again as you would imagine, on the server.  
Thus the script is never sent to the clients’ machine – on request of a page from a user the server interprets any scripts before sending the client the final product, meaning the user must send another request to the server if they wish to change anything.

This can potentially cause a lot of load on the server, if it receives a large number of requests, but the clients’ machine needs only display the result sent by the server.

Obviously this method is a lot more secure, as the client never gets sent the script – and as a result this kind of scripting is used, among many things, to provide the kind of personalization that social media sites require.

Thus, most client side scripts are merely used to add interactivity and some functionality to web pages, which remains secure and shifts some load onto the clients’ machine rather than bogging down the server with relatively menial tasks, while keeping access to databases and other equally sensitive tasks on the server, away from the user.

(SQA, 2007)

#### A server side scripting language comparison: PHP vs ASP.NET

PHP is open source, while ASP.NET is owned by Microsoft – making the former free to use, while the latter is not, although depending on the scale of the project the cost of using ASP.NET may not be that large.

PHP takes the edge in terms of support, due to its massive community of developers and open source nature.  
ASP.NET on the other hand relies more on the development tools provided by Microsoft, as well as better error handling.

In conclusion I think they each have their place – PHP is relatively easy to learn, free, and great for implementing in web apps.  
ASP.NET has a steeper learning curve but once mastered has a wider variety of applications, not to mention a multitude of development tools provided by Microsoft.

(PHP Vs ASP.NET: How to Choose the Right One?, 2017)

#### A client side scripting language comparison: VBScript vs JavaScript

Like the comparison above, VBScript is developed by Microsoft, but is free to use so long as the trade mark is displayed.

JavaScript is by far the most commonly used of the two, being the default scripting language in nearly all browsers, while VBScript was originally only available in Internet Explorer.

There are some benefits to VBScript however, in that it supports subroutines as well as functions.  
Additionally VBScript is quite an easy language to learn, and is not case sensitive.

(Difference between Javascript and Vbscript, n.d.)

## Embedded Scripts

There are three main ways to implement client side scripts.

The first is to have an external document containing the script linked to the HTML page using the script tag and specifying a source.  
This method is tidiest, as well as allowing a single script to be used across multiple documents.

The second is to simply have the script inline, ie situated in the head between an opening and closing script tag, and this script can then be called at the desired point in the body.

The third is using event handlers such as onclick, so when the user clicks on the containing element the script will be executed, which is useful for reactionary scripts responding to the users’ input.

## Event Handling (?)

Event handlers such as onsubmit=’…’ are very powerful features in web development.  
They allow the user to interact with the website, making it responsive.

The example above means that when the user clicks a submit button in a form they will trigger a script which could do any number of things – a very basic example would be for, when the button is clicked, an alert be displayed to say the form was submitted.

In short, event handlers allow for a script or action to be executed when the user triggers the event– this could be a click, or many other actions.

(JavaScript Events, n.d.)

# References

Cairns, C., & Somerfield, D. (2017, January 05). *The Basics of Web Application Security*. Retrieved October 17, 2017, from Martin Fowler: https://martinfowler.com/articles/web-security-basics.html

*Difference between Javascript and Vbscript*. (n.d.). Retrieved October 17, 2017, from Difference Between: http://www.differencebetween.info/difference-between-javascript-and-vbscript

*Introduction to the DOM*. (2017, October 12). Retrieved October 16, 2017, from MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/API/Document\_Object\_Model/Introduction

*JavaScript Events*. (n.d.). Retrieved October 17, 2017, from w3schools: https://www.w3schools.com/js/js\_events.asp

*PHP Vs ASP.NET: How to Choose the Right One?* (2017, September 8). Retrieved October 17, 2017, from PixelCrayons: https://www.pixelcrayons.com/blog/web/php-vs-asp-net-how-to-choose-the-right-one/

Robbie, J., Wood, L., & Hégaret, P. L. (2004, April 07). *What is the Document Object Model?* Retrieved October 16, 2017, from W3C: https://www.w3.org/TR/DOM-Level-3-Core/introduction.html

SQA. (2007). *Differences between Client-side and Server-side Scripting*. Retrieved Octobeer 17, 2017, from SQA.org: https://www.sqa.org.uk/e-learning/ClientSide01CD/page\_18.htm

*What is API*. (2017, October 16). Retrieved from IGI Global: https://www.igi-global.com/dictionary/api/1298