Single Page Web Applications Security

Bogdan BEDA
IT&C Security Master
Department of Economic Informatics and Cybernetics
The Bucharest University of Economic Studies
ROMANIA
bogdan.beda@hotmail.com

Abstract: With the constant spread of internet access, the world of software is constantly transforming product shapes into services delivered via web browsers. Modern next generation web applications change the way browsers and users interact with servers. A lot of word scale services have already been delivered by top companies as Single Page Applications. Moving services online poses a big attention towards data protection and web application security. Single Page Application are exposed to server-side web applications security in a new way. Also, having application logic being executed by untrusted client environment requires close attention on client application security. Single Page Applications are vulnerable to the same security threads as server-side web application thus not making them less secure. Defending techniques can be easily adapted to guard against hacker attacks.

Key-Words: SPA, JavaScript, HTML, AJAX, XSS, CSRF, Sensitive Data Exposure, Vulnerability, Injection

1. Introduction

Businesses around the globe are overwhelmed by the evolving market magnet developed around people and their potential. Few years ago, software market was built of big clients with big cash potential, willing to pay for a corporate services or applications. Nowadays, a simple, interactive and user oriented piece of software can make millions. How is that possible?
The engine of the big informational revolution is the mobile (accessible) internet, powered by Web 2.0 and AJAX (Asynchronous JavaScript Technology and XML). Web technologies are continuously becoming more and more rich and portable, enable users to interact with applications form any kind of device at any moment in time. The next generation of web applications already started to emerge. Interaction between users and modern web applications has reached new levels.
With ports 80 and 443 open to web traffic, and web pages accepting input and commands from external users, web servers have always been targets for hackers. It is very important for web application developers to pay close attention to their application security as hackers not only can harm the functionality of an application but can steal valuable user information like credit card information.
The problem of web security is increasing in complexity. Is not only about how the code is written but about the whole process of managing user data and ensuring that the data is safe and guarded against unauthorized access. There are a lot of areas in which a developer needs to exercise extra caution when building web applications, technically or at the user interaction level.

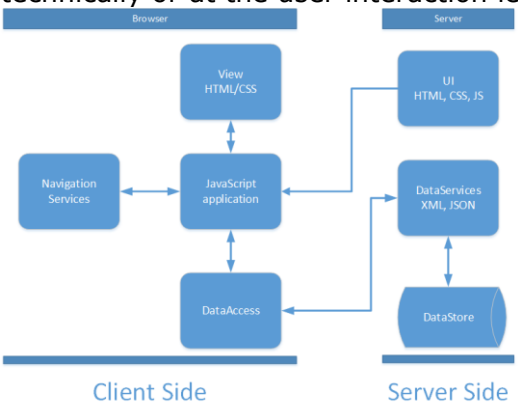

Figure 1. SPA Architecture

The focus of this article is on the security issues modern web applications developers and managers have to pay attention to, suggesting ways to guard

ISSN 2067 – 4074

against these issues. It won't talk about web server or related security issues.

2. Problem Formulation

Modern web applications use new techniques to deliver content to their users. Some old but less used technologies have like JavaScript and AJAX have become first class citizens in the modern web applications landscape being the pillars of every modern web application. Web application architecture has evolved to a more decoupled model where the client browser receives the application, runs it and the application communicates with one or more servers to get the data via API's.

All these changes expose developers to new challenges to protect their applications against attacks. Modern applications must guard against classic web attacks like XSS and SQL Injection and to new attacks like JavaScript code security, information protection, application state protection.

The new web converges towards massive personal data manipulation. Everything becomes social, engaging more people than ever. Valuable web sites to hack aren't just those owned by state institutions but those where people spend most of their time and put their personal data in.

Developers must be aware of the new security issues they face when implementing modern web applications and must create their applications with security in mind. Most of the times, in practice, the time and money do not allow for serious security attention on most applications and they are easy targets for web attacks.

3. Application Vulnerabilities

3.1 Cross-Site Scripting (XSS)

By taking advantage of XSS holes in a website, a hacker can steal user information like passwords, personal IDs, bank account information or credit cards. There is a very high percentage of public websites on the internet which are open to XSS attacks. In fact, any website that redisplays untrusted information can be attacked using an XSS attack.

XSS is the most prevalent web application security flaw. An website opens itself to an XSS attack when it displays user supplied data without properly validating or escaping the content. There are three types of XSS flaws: 1) Stored and 2) Reflected and 3) DOM Based. While the Stored and Reflected attacks can occur on the Server or the Client, the DOM Based attack can occur on the client only as it is based on client DOM modifications.

The attacking vector consists of text-based attack scripts send by an attacker which exploit the interpreter in the browser. Almost any source of data can be considered an attack vector.

In the XSS type of attack, malicious scripts are injected into otherwise trusted site. Generally, the attacker will use a browser site script sent to a different user, to obtain any kind of information in the name of that user. The malicious script sent by the attacker is not visible to the user and the browser has no way to know that the script should not be trusted. As a consequence it will execute the script as a trusted script. The malicious script will then gain access to any kind of sensitive information the browser is using for that site, like session tokens, cookies and other kind of sensitive information. The script can also rewrite the contents of the HTML page defacing the web application or asking the user for private information in the name of the website.

Stored XSS (also named Persistent or Type I) occurs when the user input is saved by the backend (like database) via user input forms like comments, messages etc. When the victim will request the stored data from the web site to display it, if the web site does not make the content safe for display, the malicious code will run on the client. For modern web application, the storage can be the client itself as they heavily relay on client side persistency. HTML 5 Storage is one of the storage targets.

Reflected XSS (also named Non-Persistent or Type II) occurs when the attacker immediately send the malicious code to the victim using the response the user

has been given as a result to an action. This happens when the web applications does not make the data it displays safe to the interpreter.
Another type of XSS attack is the DOM Based XSS. As it has been said previously, modern web applications are built around JavaScript. This means they rely on heavy DOM manipulation by the application code. DOM Based XSS is different from the other two types of XSS because the malicious code does not have to travel to server or back but will be executed directly on the client as a result of the DOM manipulations. Single Page Applications generally use URL hashes to implement content navigations. An attacker can easily inject some malicious code using URL hashes if these are executed or rendered by browser without escaping. This attack only works if the target browser does not modify the URL characters. Modern browsers usually automatically encode URL content on *document.URL* or *document.location* properties. This does not mean the user should rely on browser encoding.
Most Server XSS flaws are fairly easy to detect using testing or code analysis. Client XSS is however very difficult to identify as the code execute by client host.
The main action required to prevent XSS consists of separation of untrusted data from interpretable browser content.
The most common option is to escape all untrusted data being displayed on a web page using html escaping techniques.
Another defense option would be to use "whitelist" validation for input fields but this is not considered a complete defense as many website require special characters in their input.
It is highly recommended to use specialized libraries to sanitize untrusted data instead of implementing custom solutions.

3.2 Sensitive Data Exposure

Web applications generally save user's personal information such as passwords, credit card numbers, addresses etc. in a database. Also, the saved data is used by business logic operations or is required in one form or another by presentation layer.
When the system is not protected effectively from unauthorized access there is a very high probability that an attacker will try to exploit the vulnerability and steal the information. This vulnerability is called "Sensitive Data Exposure".
If an website stores sensitive information, its owners should make sure that the website stores and uses the information in a way that it can't fall into the hands of those who would misuse it. Identity theft and online crime have surpassed other types of crimes in terms of profits earned.
There are a several ways in which sensitive data can be exposed to attackers. Some of them relate with server administration and security policy and others refer to web application security.
Developers should pay close attention to what data is being sent to the client application. With more code running on an exposed environment, on client browser, there's no control over what an attacker can do with the application. It very important to send only the data which is not sensitive on the client.
Modern applications fetch data from the server using background web requests and a lot of developers think that those requests are hidden from users, so the application is safe. The real situation is that the application is not safe and those requests can be easily tracked and their result can be easily read by an attacker. This is also called security through obscurity and is a wrong security technique as the impression of the data being hidden is not a valid security measurement. Things like passwords, encryption keys, credit card data should never be sent to the client.
Web applications should use strong encryption techniques to encrypt stored sensitive data and pay close attention on how the data is handled when it decrypted for manipulation.
Another way to steal sensitive data from a user is by using phishing. Phishing is a clever way of extracting information from unsuspecting users. Usually, it is performed using links sent via emails which look like they were sent by a reputable company.

ISSN 2067 – 4074

Social engineering is another way to gather sensitive information from users but it relies entirely on user weaknesses.
One important principle in keeping sensitive data secure is avoid keeping it. Sensitive data should be kept only if it is necessary and should be discarded as soon as it is not needed any more. Missing data can't be stolen.
If the application needs sensitive data, it must be encrypted using strong encryption algorithms at rest or when in transit.

3.3. Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF) attacks rely on the fact that an application token (session cookie) is predictable for a specific amount of time (until the user closes the browser). The CSRG exploits the trust that a server application has in a user's browser, unlike the cross-site scripting (XSS) which exploits the trust a user has for the application.
The cross-site request forgery works by executing an action from an external application, to an application in which the user is trusted or is supposed to be trusted. This attack relies on the fact that a browser will send web-application related information back to the web-application, regardless of where the request came from. This means that a browser will send an authentication cookie or token with a HTTP request even if the request didn't originate from a page of the application. This way, a malicious web site can send commands to another website which relies on browser stored tokens for user authorization.

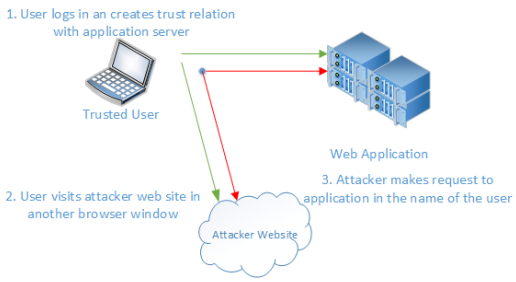

Figure 2. Cross Site Request Forgery

CSRF involves web applications which rely on user's identity and exploits the application's trust in that identity. Attackers can do on target application, whatever the victim is allowed to do.
In classic web applications, a common way to guard against CSRF is to add a random token into a hidden field on pages executing commands. Same version of the token should be added to a cookie. When a command is being sent from page to server, both the cookie and the input field value will be received by the server. If the values are different, then the command is rejected.
Another way to check the request would be to check the referrer header which identifies the address of the web page that linked to the resource being requested.
In single page web applications, the CSRF attack can't be prevented using the same method as server-side applications. In a single page application, the HTML forms are not generated on the server but are created dynamically on the browser. A solution for preventing CSRF attacks for SPA is to pass the anti-forgery token in an Ajax header instead of a hidden form field. Also, the developers should take advantage of how different development frameworks implement CSRF guarding.
For some commands and scenarios, the user can be required to re-authenticate or prove they are a user.

3.4 SQL Injection

SQL Injection attack is based on attacker sending a text based exploit in a form of a command for the targeted SQL interpreter. Almost any source of data can be a gate for potential attackers. SQL injection can be used by attackers to introduce code into a web application with the purpose of modifying or stealing the data. The results of this attack can be very disastrous.
SQL Injection flaw occurs when an application interprets untrusted data as it is. SQL Injection is a very easy to exploit vulnerability as the attacker. Any piece of code that constructs SQL statements should be carefully reviewed for injection vulnerabilities as database servers execute all syntactically valid queries that they receive. Even parametrized data can be a source of SQL Injection attack.
The direct SQL injection attack consists in directly concatenating SQL code into user-

input variables which are later concatenated with commands and executed. An alternate attack method is to insert malicious code into strings that are destined for storage in a table. When the strings are later concatenated into dynamic SQL commands, the malicious code is executed.
The injections process exploits the syntax accepted by the interpreter. It uses the instruction separator to separate the command server appends at the start of user input and the comment indicator to specify that what will be added after user input by server is to be considered a command, thus ignored by the interpreter. All code that executes dynamic SQL commands must be carefully reviewed by the developers.
SQL injection is one of the top web application vulnerabilities.
A parametrized query builder in which SQL parameters are strongly bound to their types is the preferred way to guard against SQL injection. Assigning user input data which is about to be used in an SQL query to typed variables ensures that user input has the required type and length and they are not some malicious code. However, if the variables are strings themselves or there is no way to use parametrized query construction and variable binding, user input should be properly escaped to exclude special characters used by the interpreter.
White or black lists can be used to filter and validate input data but these are not considered complete defenses as many applications require special characters in their input.
User rights policy on the database is also a very important aspect of guarding against SQL injections. Database user used to execute application code should have only the rights she needs to execute application actions only. Any additional right combined with unsafe input validation could open severe injection holes.

3.5 JavaScript Code Security

As it has been said before, JavaScript is the main building block of modern single page web applications. The way JavaScript is being written and how it stores and accesses data is an important aspect in securing a modern web application.
The first aspect developers should have in mind when developing JavaScript driven web applications is that the code will run inside a client browser with the user having access to read, interpret and modify any portion of the code. The client browser is a completely unsafe environment. Actions executed by client JavaScript must not disclose any sensitive data sensitive data nor try to hide it. It must not use it at all. Having a more complex logic it is a high temptation to pass to the client a lot of information, to make things happen but that data should be carefully considered. Things like encryption keys and passwords should never be used or stored on the client.
A technique which is mistakenly considered a security technique is *security by obscurity*. Single Page Applications use AJAX behind the scenes to execute commands and fetch data. These requests are generally executed without noticing the user directly (are not result of user actions). This may lead to the assumption that those requests can execute sensitive operations as they are not visible to users. This assumption is clearly wrong as any user can monitor and access all request and running instructions of a JavaScript application running inside a browser.
Another technique which shouldn't be considered a strong form of security is JavaScript obfuscation. JavaScript obfuscation is just another type of security by obscurity. Obfuscation is efficient if someone wants to hide code from normal users and make it hard for them to understand the code but is fairly easy for hackers to reverse engineer the process and understand the code. Even the most advanced obfuscation technique to date should not be considered safe enough to be used as a way to properly secure client JavaScript code.

4. Conclusion

Although the anatomy of a web application has evolved during the past years and new technologies emerged,

ISSN 2067 – 4074

modern web applications must be guarded against same time of attack classic web applications were vulnerable to. There are differences between how attacks target modern web applications and classic web applications but the attacks are the same, the prevention techniques slightly differ. Also, heavy usage of JavaScript and client side code requires even more attention on how the code is architected and written.
Modern Web Applications are not inherently less secure than server-side applications. For any kind of applications, regardless if it is a Single Page Application or a traditional server-side application developers must constantly guard against vulnerabilities.

Acknowledgement

Parts of this paper were presented at The 7th International Conference on Security for Information Technology and Communications (SECITC 2014), Bucharest, Romania, 12-13 June 2014.

References

[1] Paco Hope, Ben Walther, Web Security Testing Cookbook, O'Reilly , 2008;
[2] Michael Mikowski, Josh Powel, *Single Page Web Applications: JavaScript end-to-end*, Manning, 2013;
[3] Bryan Sullivan, Vincent Liu, Web Application Security, A Beginner's Guide, McGraw-Hill Osborne, 2012;
[4] Hacking Web Apps, Mike Shema, Syngress, 2012;
[5] The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, Dafydd Stuttard, Marcus Pinto, John Wiley & Sons, 2011;