UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Testing security of html5: automated scanning vulnerabilities

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Autor: Javier Gil Pascual

Tutor: Jose Ismael Ripoll Ripoll

Hugo Jonker (externo)

2016-2017

Resumen

HTML5 tiene varios nuevos componentes como XHR-Level2, DOM, Storage. Con esta introducción de nuevas tecnologías, HTML5 también lleva consigo potenciales riesgos de seguridad. Algunos originados de los elementos del estándar en sí, otros de la implementación particular del estándar en cada navegador, y otros del cuidado que pongan los desarrolladores a la hora de escribir código. . . En esta tesis vamos hablas de estas nuevas estrategias de ataque y posibles amenazas. También cubriremos cómo detectar estas vulnerabilidad automatizando el proceso. Esta tesis describe una serie de vulnerabilidad web, sobre las que hemos construido unos test para probar las capacidad de algunas herramientas de pentesting. Basándonos en los resultados observados, discutiremos futuros resultados.

Palabras clave: HTML5, pentesting, web, seguridad.

Abstract

HTML5 has several new components like XHR-Level2, DOM, Storage. With any major introduction of new features, HTML5 also brings with it potential security vulnerabilities. It allows crafting stealth attack vectors and adding risk to end client. Some originate from elements of the standard itself, some from implementations of the standard in each browser, and some from the care that developers do (or do not) take in building their HTML5 code. . . In this thesis we are going to talk about this new attack surface and possible threats. We are also going to cover how to automatically detect these possible vulnerabilities. This thesis describes a set of HTML5 vulnerabilities, we build a test web and use this for test the capabilities of several open source pen-testing tools. Based on the observed results, further work is also discussed.

Keywords : html5, pentesting, web, security.

Tabla de contenidos

1. Introducción

The web of today has adopted HTML5 in order to give power to web applications and take them to the next level. HTML5 is improving browser capabilities with several new technologies. With the interest emerged towards html5, new security issues have also begun to come up and novel attack point for malicious users [1]. These vulnerabilities are hard to detect due to the complexity of the new interactions between browsers and web servers. Furthermore, doing this analysis automatically adds an additional level of challenge. Even though there are some companies that offer solutions specially designed for test web vulnerabilities, the vast majority of them cannot deal with modern web technologies, like the dynamic web apps loaded with JavaScript. It is necessary evaluate all new threats and adapt or create new security tools for further improvement of our security as internet users.

1.1. Motivation

In 1989 Tim Berners-Lee invented the World Wide Web, based on three main technologies: a system of globally unique identifiers for resources on the Web and elsewhere, the Uniform Resource Locator(URL); a way to send data between browser and server, the Hypertext Transfer Protocol (HTTP); the publishing language Hypertext Markup Language (HTML). HTML dictates the structure of a web page semantically. The web started simple and with a lack of standard, in the browser wars of the 90s each competitor launched their implementation of JavaScript As the years passed, some companies like Microsoft and Adobe expanded their vision for an Internet of closed source software, powered by their plugins Silverlight and Flash. But in 2008, the World Wide Web consortium, published the HTML5 working draft. Steve Jobs said in 2010: "HTML5... lets web developers create advanced graphics, typography, animations and transitions without relying on third party browser plug-ins like Flash"[2]. Under the new rules of HTML5, audio, video and complex graphics can be added or created directly on a web page.

Web applications have travelled a significant distance in the last decade. Looking back, it all started with CGI scripts and now we are witnessing the era of RIA[3] and Cloud applications. HTML 5, DOM (that now with the DOM level 3 is integrated in the HTML standard) and XHR (Level 2) are new specifications being implemented in the browser, to make applications more effective, efficient and flexible. Hence, now we have three important technology stacks in the browser[4] and each one of them has its own security weaknesses and strengths.

However, new elements of HTML5 have expanded the range of possible attacks by third parties and new web security vulnerabilities have been discovered in web applications and sites that support HTML5[1].

The typical web vulnerabilities, like SQL injection and XSS, are very widespread. We can see that currently this is ones of the main problems on the internet, taking into account the number of vulnerabilities reported to the database Common Vulnerabilities and Exposures[5]. According to reports of Symantec, three quarters of websites contain major security vulnerabilities in 2015 .

Nowadays this common vulnerabilities with HTML5 can be exploited through new ···· ways. HTML5 security issues have drawn the attention of the European Network and Information Security Agency (ENISA), which studied 13 HTML5 specifications, defined by the World Wide Web Consortium (W3C), and identified 51 security threats.

In this scenario, guaranteeing and testing the website security is a must. A common approach for testing web sites is to use a Dynamic Application Security Testing tool, also known as “black box” testing tool(from now on DAST tools). These tools crawl over web applications in order to enumerate all the existing pages they can reach and their associated input vectors. Afterwards generate different crafted input values that are submitted to the app. Then observe the response to determine if a vulnerability has been triggered.

1.2. Research Problem

HTML5 is a relatively new standard in the field of web development where standard, techniques and trends continuously evolve. Even though HTML5 security is better than Flash and other third party plugins, there are no technologies that cannot be misused from a security perspective. Security weaknesses can appear in any technology when developers use it for reasons other than those for which it was originally intended. %This could be explained by the fact that there are developers who are not so security oriented and thus do not pay attention to security.

Lavakumar Kuppan[6], a security researcher explains: "HTML5 brings a lot of features and power to the Web. You can do so much more [malicious work] with plain HTML5 and JavaScript now than it was ever possible before. That said, many of the new features constitute threats on their own, due to how they increase the number of ways an attacker could harness the user's browser to do harm of some sort.

The new web applications are constructed in a completely different way from what has been seen so far, often as SPA (Single page application). These websites are the present and possibly the future of internet, so it is essential to be able to guarantee a correct evaluation of their security. This is where the DAST tools that we are going to evaluate come into play. It is expected that the latest versions have added the functionalities or extensions necessary to deal with web pages accompanied by extensive libraries in javascript. Although, the reality in the world of web security is not always as expected. Many of the tools we can count on are traditionally more geared towards finding highly recognised vulnerabilities with greater fame, as it is the best way to prove their worth against competitors (see owasp top 10). These vulnerabilities, for example XSS or SQLinjection are basically based on forcing the inputs that the web sends to the server to look for possible injections of malicious code or to alter the behaviour of the server.

Something for which DAST tools are highly optimized. But it is not the same in new web apps where communication does not have to be always client-server. New vulnerabilities may manifest themselves in communications or actions on the part of the client, which does not have to be transmitted to the server.

1.3. Research Objectives

Web security from the client side is becoming an important field of research that emerged since the mid 2000s. Besides a broad overview of client-side Web security, this thesis proposes a study of the new threads introduced by HTML5. As finding bugs in software is better and cheaper the earlier it occurs, we focus strongly on autonomous mechanisms to detect the vulnerabilities, like DAST tools. We provide a review of the maturity of those tools solution.

The present thesis proposes a study of the new arising vulnerabilities in HTML5 and its JavaScript API technologies. After researching and analysing a state of art example of this issue we show what kind of defense mechanism exist for defending against different threats. However, because the web is so complex some vulnerable points will always exist, so we also provide a review of the most advanced web application vulnerability scanners, focusing on what tools are there for detecting the different threats and how these tools evolved to detect and deal with new technologies as single page applications built with Ajax a rest APIs.

The present thesis proposes a study of the arising questions by new threats and web technologies. The research question is:

Can we still use DAST open source tools for security evaluation of modern web apps?

And the Questions derived are:

- What are the new vulnerabilities in HTML5 technologies?
- Can these tools be used in a Single Page Application?

The main contribution of this thesis are:

1. We detected the most used HTML5 technologies.

2. We performed the most extensive evaluation of vulnerabilities regarding HTML5.

3. We choose the best 3 open source DAST tools base on input vectors support and HTML5 features supported.

4. We perform a comparison of the crawling capabilities of this tools against a RIA web.

While the topic of how these new technologies can be exploited is a good research topic, there are a growing list of new act scenarios and threats. Because of this we are unsure if this will apply to new attack types or ways to confront older types of attacks.

1.4. Thesis outline

The first question will be answered in this work in Chapters 2 and 3. We will introduce the new technologies in the second chapter and in Chapter 3 we will discuss the threats, at least for the main new technologies. The list of news in html5 is long and we can not cover all possible vulnerabilities.

The second question will be partially answered in Chapter 4. The tools and characteristics of these aimed to discover the new html5 related threats are explained in Chapter 4. We will prepare some test sets with the new HTML5 vulnerabilities and build a realistic web application in Chapter 5 in order to put the tools to the test. More precisely our web uses features of HTML5 but misconfigured to make easy a possible attack. Learning how those vulnerabilities can be used for attack scenarios is important to understand how to test them.

To answer the third question, we will launch the tools against some test of crawling. In order to see how the tools deal with a Single Page Application a website with this characteristics is chosen in Chapter 5.

The evaluation of the tools selected with several tests results is performed in Chapter 6. Finally, we also conclude this thesis in Chapter 6.

2. Technological Background and Related Work

In this Chapter we introduce the new technologies and background work related to DAST tools.

2.1. Technological background

Modern web apps are growing in features, size and number of users. Newer features require newer technologies explained in this section to comply with the growing needs. In this chapter we present some background on the related technologies like HTML5, web applications and dynamic testing tools scanners. We also briefly discuss related work.

2.1.1. HTML5

HTML stands for HyperText Markup Language, and it is the authoring language used to create documents on the World Wide Web. HTML is used to define the structure and layout of a Web page, how a page looks and any special functions. HTML does this by using tags that have attributes.

On 28 October 2014, the W3C officially published HTML5 as a Web standard (or recommendation of HTML5). HTML5 is the fifth major revision of the format used to build Web pages and applications.

HTML5 contains powerful capabilities for Web-based applications with more interaction, video support, graphics, more styling effects, and a full set of APIs. HTML5 adapts to any device, be it a desktop, mobile, tablet, or television device. HTML5 is an open platform developed under royalty free licensing terms.

People use the term HTML5 in two ways:

- to refer to a set of technologies that together form the future Open Web Platform. These technologies include HTML5 specification, CSS3, SVG, MathML, Geolocation, XmlHttpRequest, Context 2D, Web Fonts (WOFF) and others. The boundary of this set of technologies is informal and changes over time.
- to refer to the HTML5 specification, which is, of course, also part of the Open Web Platform.

According to recent statistics published by Powermapper[7], more than 70% of the web pages analysed use the HTML5 DOCTYPE. This means that they are HTML5 web applications.

HTML5 let developers to move more and more code towards the client side. This has an effect on increasing power on the client side, providing more user features and reducing the server load at the same time. But at the same time this new rich client side introducing some attack vectors. Before analysing the possible vectors of attack, we must understand the logical layers of modern browsers, depicted in figure 1.1.

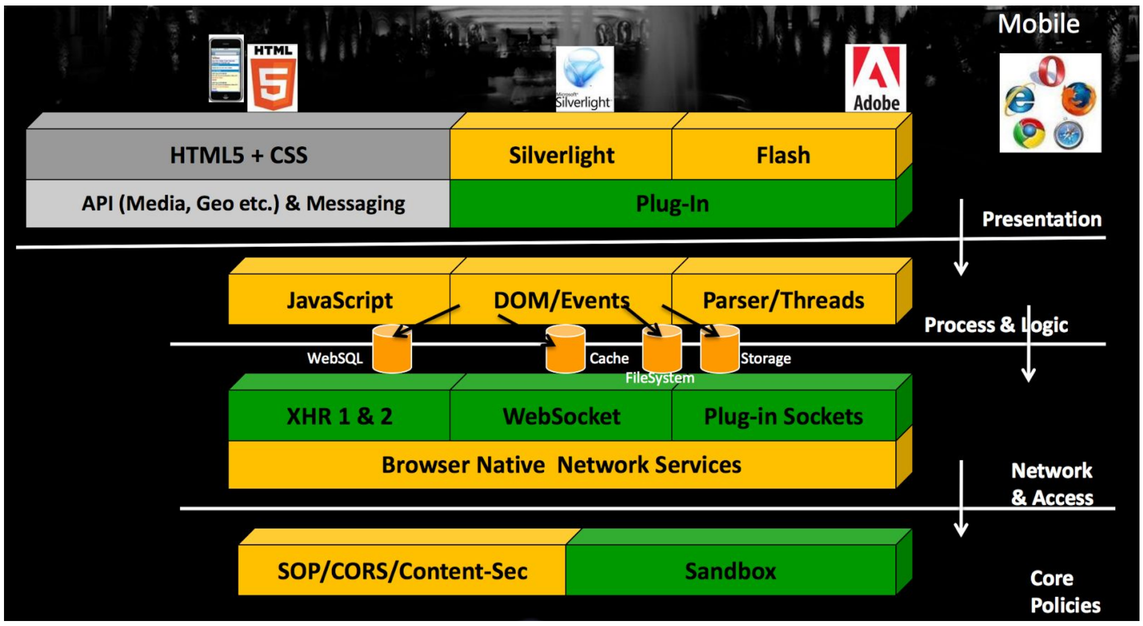


Figure 1.1: Logic layers web browser

We have 4 layers:

- Presentation layer, witch contains the native implementation of HTML5
- Logic layer, where the core of the browser resides which allows DOM, threads, and Javascript
- Network layer, which allows connectivity to the Internet and Domain/Cross Domain calls across networks. HTML 5 supports WebSocket and XHR(Level-2) calls.
- Policies layer, that controls and allows overall security to end users. Policies like SOP and CORS are defined here.

HTML5 within the Open Web Platform featured technologies than are being introduced in all www by de developers are detailed below. New communication mechanism are:

CORS: Cross Origin Resource Sharing or CORS is a mechanism that enables a web browser to perform "cross-domain" requests using the XMLHttpRequest L2 API in a controlled manner. In the past, the XMLHttpRequest L1 API only allowed requests to be sent within the same origin as it was restricted by the same origin policy.

WebSockets: Traditionally the HTTP protocol only allows a TCP connection request/response. AJAX technology provides connections to the server without the page having to be refreshed, however AJAX requires the client to initiate the request and wait for the server response (half-duplex).

HTML5 WebSockets introduce the possibility that the client or server can create a full-duplex communication channel, which gives the two parties total freedom to communicate asynchronously without waiting. The initial request of the Websocket protocol is http, is an upgrade because after the affirmative response leaves opens the communication channel over TCP, the Websocket channel.

To extend client functionality HTML5 brings to scene several new JavaScript API. Modern web browsers have a good support for these APIs nowadays\cite{5}. Further security concerns and threats of this specification is given in section 3. The new JavaScript APIs are:

WebStorage: Before HTML5 web applications could only store data on the client using cookies. But cookies are very limited in space and quantity. To address these limitations and support offline applications, the concept of local storage as web storage appears. Web storage has the possibility of storing data in the client and having access to them later through javascript. There are two types of local storage:

- Local storage: you can store any type of data in tuples. The data is stored until it is explicitly deleted. Closing the browser does not erase the data. Access is protected with SOP.
- Storage session: similar to a local storage but data is deleted when the browser is closed.

Another difference that local storage has to the cookie system is that the local storage variables are not sent to the server in each request, they do not have an expiration date like the cookies and that the local storage objects are separated by the SOP policy. Values saved through HTTP can not be accessed by HTTPS. Cookies, although established by HTTP, are also sent with HTTPS, if the domain is the same.

WebMessaging:New feature-rich websites typically include javascript applications embedded in iframes. These iframes are secure but are isolated from the other domains. In HTML5 a secure communication mechanism through domains is called Cross Document Messaging. This API introduced in the WHATWG HTML5 draft specification, allows documents to communicate with one another across different origins.

To use the messaging api, HTML5 has introduced a postMessage function. With this function, plain text messages can be sent from one domain to another. First the author obtains the Window object of the receiving document. Therefore, messages can be posted to the following:

- other frames or iframes within the sender document's window
- windows the sender document explicitly opens through JavaScript calls
- the parent window of the sender document
- the window which opened the sender document

WebWorkers: WebWorkers give you the ability to run Javascript in the background. They are similar to known threads in programming languages. With Web Workers you can run a heavy JavaScript task, such as accessing network resources, while the web page continues to respond to the user, without blocking.

A fact that we were interested in was the popularity of Web technologies among the recent year showed by developers. \ref{fig:stack} shows the relation between the recent years and the interest/number of searches of some topic within stackoverflow.com, the website where Web developers discuss about programming. The diagram clearly shows that popularity of new web technologies are arising, and more likely to utilize Web technologies than not so popular ones.

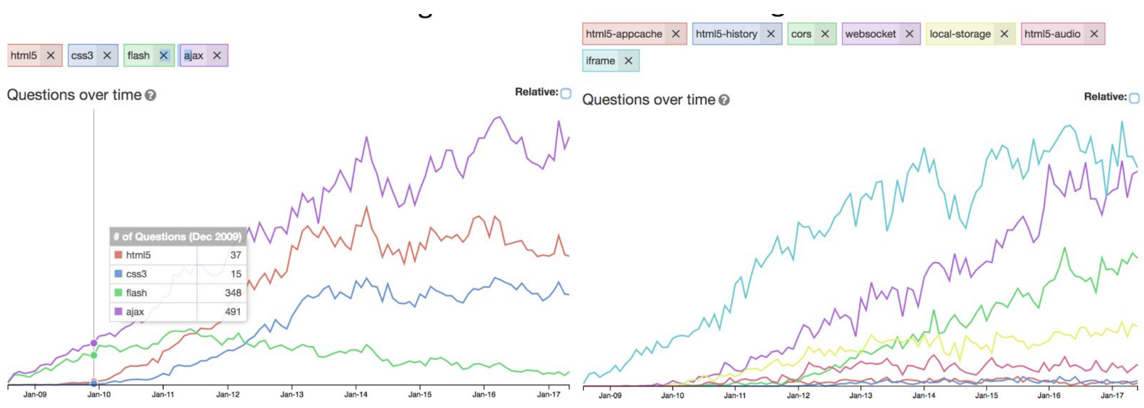


Figure 1.2: Interest/questions related to new APIs

From the graphics we can figure out that HTML5 is being used extensively in web development. The first graph show the comparison with other relevant web technologies. In the second graph we can se that the most important technologies within HTML5 are CORS, Websockets, local-storage and the iframes.

2.1.2. Single Page Application

A single-page application (SPA) is a web application or web site that fits on a single web page with the goal of providing a user experience similar to that of a desktop application. SPAs aren't a fundamentally different model, just a new way for allowing users to interact with a web application in a manner that takes advantage of modern browser capabilities to provider a better user experience.

SPA applications use framework for render the html pages in the client that usually filter and validate all inputs, but this does not mean that are not immune to CSRF, XSS, or other normal attacks. The interaction between the client and the server still dealing with HTTP requests and responses that usually are formatted as JSON.

However you cannot rely on JavaScript in the browser for things like input validation and function authorization. If it is on the client, it doesn't belong to you. Any validation and authorization that happens on the client is UX, not application security. Your endpoints on the server are responsible for ensuring the security of the data and the application.

2.1.3. Front-end and Java script Frameworks

In software design the front-end is the part of the software that interacts with the user (s) and the back-end is the part that processes the input from the front-end. The separation of the system in front-ends and back-ends is a type of abstraction that helps to keep different parts of the system apart.

In the front, the developers have to integrate html, JavaScript and css in their web application to generate an interface with which the user can interact. One of the challenges associated with front end development is that the technologies used are continuously changing and these should be alert that possible security consequences may have to use a certain library or framework. The truth is that on the client side, JavaScript is ubiquitous. Anywhere you look there is something created with javascript. This is due to ease of use and distribution since JavaScript is an open source. In addition, modern browsers that support HTML5 allows you to move more logic from the application and the interface to the client side, for which JavaScript reduces the amount of code to write.

Some examples of more popular open-source libraries are jQuery, Prototype, YUI, ExtJS, MooTools, which allow developers to create shortcuts to work with elements in the Document Object Model (DOM). Later frameworks such as AngularJS, EmberJS or BackboneJS also have frameworks like Backbone.js, Angular.js, and Ember.js, which ensure that the content will be viewed correctly regardless of the device in which they are executed, and facilitate the development of new SPO that can often be a challenge. In a Single Page application, all the code needed to render the page is included in a single application load. This includes all CSS, HTML and JavaScript. The page is not reloaded so you will need dynamic interactions with the web server. Ajax, a very widespread technique for downloading content from the server allows these dynamic uploads through JavaScript With the development of AJAX (Asynchronous JavaScript + XML) a new way of creating interfaces in web applications was introduced. With AJAX you can send and receive information asynchronously without having to refresh the page that is shown to the user.

This paradigm shift is known as Web 2.0, and became famous when major Internet products (such as GMail or Google Maps) adopted it. These new techniques extend the frontiers of web vulnerabilities by opening the possibility to perform serious attacks as seen in Chapter 3.

2.1.4. Related Work with HTML5 issues

The background related with this thesis come from three main points, the security concerns of HTML5, new novel tool for testing new vectors and articles of traditional comparisons of pen testing tools. In this section we present some work from other paper related with HTML5 Javascript API and communication mechanisms.

For several of the new capabilities added to modern Web browsers, insecure usage scenarios or security issues have been identified. We list a selection of them.

In [8] the author tested a local storage vulnerability, illegitimate use of storage capacity of clients to store compromised data. With a modelled web that takes a file, encrypts it, and slices it up into 26 parts to distribute it to as many client systems as possible show how to achieve a reliable recovery rate of the whole file afterwards. Moreover also regarding to local storage in [9] the author identify the problem of caching code into local storage as a practice that creates new attack vectors for adversaries. Traditional Anti-XSS mechanisms are not applicable for cached code because they make cached code unusable . Instead of filtering and validating the code inserted, the author proposes a lightweight integrity preserving mechanism for Web Storage transparently using API wrappers, that implies only a small overhead.

Furthermore, [10] have shown that the postMessage API can be used insecurely, if a JavaScript that accepts postMessage-events does not verifies the origin of the incoming data carefully.

In addition [11] studies if there are new vulnerabilities in the WebSocket protocol/implementation and the defense mechanism against different arise threats.

This paper [12] proposes Threat Detector for WebSocket and Web Storage (TD-WS), a dynamic taint tracking- based threat detector for WebSocket and Web Storage. TD-WS tracks the unsafe data flows and detects potential privacy leaks and XSS vulnerabilities in the client-side WebSocket and Web Storage applications.

%After reviewing the available defenses, we propose a JavaScript-based defense to use until browser support for a solution such as X-FRAME-OPTIONS is widely deployed.

There are some other articles that show some tools dealing with new XSS faults [13–15]. Abuseada [16] proposes an alternative methodology to detect DOM-XSS by building-up on the existing approach used by web scanners in detecting general XSS.

In [17] Doupe et al propose a novel way of inferring the web application's internal state machine from the outside—that is, by navigating through the web application, observing differences in output, and incrementally producing a model representing the web application's state. They utilize the inferred state machine to drive a black-box web application vulnerability scanner. We implemented our technique in a prototype crawler and linked it to the fuzzing component from an open-source web vulnerability scanner.

2.2. Pen testing Tools Background

In this section we present an introduction to a kind of pen-testing tools and some work from other paper related with traditional comparisons of pen testing tools.

2.2.1. Dynamic testing tools

Web Applications nowadays have become extremely complex with new features being added on daily basis. It's almost impossible to achieve complexity and Security at the same time. Developers and security experts have been aware of this situation and need to focus to focus more security resources on web applications, so 79 percent of developers and security are now more concerned and apply security resources to public-facing web applications as we can see in the 2015 sans whitepaper [18]. Moreover application security (AppSec) is maturing for most organizations

To be effective, application security has to be included throughout the complete development life cycle; Use dynamic analysis (DAST), static analysis (SAST), are creating a new way to build applications, also called the application Secure Software Development Life Cycle (SSDLC).

Dynamic application security testing (DAST) technologies are designed to detect conditions indicative of a security vulnerability in a application in its running state. Most of these solutions can test only the exposed HTTP and HTML interfaces of Web-enabled applications, and many also test Web services such as Simple Object Access Protocol (SOAP) or representational state transfer (REST).

However, the main advantage of the dynamic scanners is the fact that they form a general analysis solution. As indicated, the dynamic scanners can operate on any language and framework.

There are a wide range of DAST tools on the market with both commercial and open-source solutions. Unfortunately, the evolution of web application into rich client interfaces have left behind most application security scanners, dynamic scanners have some limitations and drawbacks, which we discuss in chapter 4.

Automated Scanners won't necessarily protect your Web applications. As both of them do not understand Business Logic of the Application. In depth defence principles should be followed to ensure security in all layers.

2.2.2. Related Work DAST tools comparison

Here are two research papers that summarize the state of the art in this area:

[19] Adam Doupe et al evaluates 11 black-box web vulnerability scanners, both commercial and open-source, and compares their effectiveness on a sample application.

[20]Jason Bau et al in his paper evaluates 8 black-box web vulnerability scanners, and tests their effectiveness on several web applications.

Additional information on previous benchmarks, comparisons and assessments in the field of web application vulnerability scanners are in [21], where the author published the results of his own Benchmark.

3. New HTML5 threats

The previous section introduced several new technologies of modern Web apps. This chapter lists the vulnerabilities we figure the tools could test. For each vulnerability an overview will be given, followed by one example or way to exploit these new technologies especially crafted for the convenience of the tools.

Misuse of new HTML5 elements and features may cause vulnerabilities to be built into web sites. Besides, web applications that had been considered secure in the past may become vulnerable since new enhanced featured may make older mitigation measures useless or inconvenient. We want to select some features where DAST tools could help.

Moreover many times vulnerabilities appear in browser specific implementations. According to Symantec 2.4 browser vulnerabilities were discovered per day in 2016, more than 800 in the 4 major browsers[22].

The list of news in HTML5 is long and we can not cover all possible vulnerabilities so we review the ones we consider most suitable for testing according to the most popular technologies among developers showed in Chapter 2.

3.1. Threat Models

Before going into depth with the vulnerabilities let's define our threat model. This section presents relevant threat models for client-side security, also explaining its general capabilities. We can distinguish two kinds of malign users willing to exploit any vulnerability exposed. These models are the most relevant for the discussion of client side Web security, and as simpler as any user can become an active attacker with only a bit of knowledge of the victim system

Forum Poster: This model represents a user of an existing Web application. This user can supply limited kinds of content, can issue requests from the honest site's origin

Web Attacker: A web attacker is a malicious principal who owns a domain name, e.g. attacker.com, has a valid HTTPS certificate for attacker.com, and operates a web server. These capabilities can all be obtained for 10 euros. If the user visits attacker.com, the attacker can mount a CSRF attack by instructing the user's browser to issue cross-site requests using both the GET and POST methods.

For clarify is interesting define what is a threat vs vulnerability. The former generally can not be controlled, rather threats need to be identified. It may be an expressed or demonstrated intent to harm an asset or cause it to become unavailable. The latter are weaknesses we should take proactive measures against and correct. A vulnerability is a flaw in the measures you take to secure an asset.

3.2. CORS

Prior to HTML5 websites were limited to making a http or Ajax request, the Same Origin Policy restricts the communications within their origin domain. An origin is defined by the combination of the following three properties: protocol, domain, and port. If all of these are same for two hosts, communication is allowed. This is especially problematic for web applications which are loading data from different origins.

As stated in section 2.1 with HTML5 this changed, a new standard called CORS was introduced. CORS makes it possible to send cross domain http requests, and more importantly, XHR (XMLHttpRequest) across domains using a new http header called "Access-Control-Allow-Origin". With this header a domain can specify which origins can process its data. The UA makes the decision if the JavaScript is allowed to show the response or not. For example a header "like Access-Control-Allow-Origin: http://webA.com" means that only the website with origin webA.com is allowed to access the web providing this header.

3.2.1. Threats and security issue

HTML5 and CORS give new ways to bypass the Same-Origin Policy. The central security problem is that XHR are sent without the user's permission. This can be used to break the security of an authenticated session, and make requests on behalf of the victim. Another problem with CORS is that now the origin of data can came from anywhere bypassing server validations. This untrusted data needs to be validated on client.

For example, if the response returns a wildcard mask in the http header as it is in the following example, this indicates that access from all origins is permitted.

```
Access-Control-Allow-Origin: *
```

Assume that in a intranet an internal web site defines the access control header wrongly, with the *. Accessing internal websites from the internet is possible. If a internal user accesses to this website and in the same session is tricked to open an attacker controlled malicious website in the internet with malicious JavaScript code, this JavaScript code makes a background XHR request in the UA to the internal website. Because the intranet website has the access control header is set to *, JavaScript can parse the result and send the content back to the attacker controlled web server.

Other ways to exploit CORS basically are due to misconfiguration of the http headers. In some cases this is because of the web trust the null origin. This origin is for the origins from the filesystem (load call come from file:// URL). This is great for attackers, because any website can easily obtain the null origin using a sandboxed iframe:

```
<iframe
sandbox="allow-scripts allow-top-navigation allow-forms"
src=''data:text/html,<script>*cors stuff here*</script>>
</iframe>
```

A severe security vulnerability found on Facebook uses this kind of technique [23].

In other cases websites reflect the origin header without attempting to validate the origin or make wrong validations, for example a domain called somedomain.net trusts all origins that start with *https://somedomain.ne*t, including *https://somedomain.net.evil.net [24].*

Cross Site Request Forgery (CSRF) is defined in the Owasp CSRF Prevention Cheat Sheet as "occurs when a malicious web site, email, blog, instant message or program causes a user's web browser to perform an unwanted action on a trusted site for which the user is currently authenticated"[25].

In another example a CSRF attack is possible. The page apart from return the CORS header with a wildcard(*) or another misconfigured header, has a CSRF token in the response. Any domain can perform XHR request and fetch the required data. To extract the CSRF token, we can craft a web to send the website an AJAX request that will crawl the code and copy the token.

```
xmlhttp.open("GET","https://site.com/",false);
xmlhttp.send();
if(xmlhttp.status==200)
{
      var str=xmlhttp.responseText;
      var n=str.search("csrf-token");
      var c=str.substring(n+30,n+74);
      var url = "http://badSite.com/grab.php?c="
      + encodeURIComponent(c);
      xmlhttp.open("GET", url, true);
      xmlhttp.send();
}
```

A more realistic example of stealing the CRSF-token can be found in [26], where the author discovered this vulnerability on a real web site and shows a proof of concept where successfully exploited it to grab the users authenticity token.

We found also that an attacker can exploit old websites which load files using XMLHttpRequest creating a page on a remote server that allows cross-domain requests from the particular origin of the victim website, or all origins using CORS. It is a new kind of Remote File Inclusion. If there is no url validation we can inject a remote script passed after the \# character in the URL and create a Content injection. In this example some web load content using the fragment, originally from the same domain:

http://website.com/#/a/page

```
xhr.open("GET", "/a/page");
```

But an attacker can place content in a domain whith CORS configured incorrectly, like the following PHP scrip, name it remotexss.php:

```
<?php header('Access-Control-Allow-Origin: *');
?>
<div id="main">
<imgsrc=x onerror=alert(document.domain) /> </div>
```

Now if you enter to the link "*http://website.com/\#http://example.com/remotexss.php*" the content was successfully loaded, triggering an alert.

3.3.1. How to test

Access-Control-Allow-Origin is a response header added by a server to allow some domains to read the response, but it is up to the client to determine whether the client has to show the response data based on this header. From a penetration testing perspective we can look for insecure configurations as for example using a "*" as value of the Access-Control-Allow-Origin header. Another insecure example is when the server generate the origin header based on your input and returned back without proper validations.

We can use the dynamic tools to check the http headers to understand how CORS is used, more particularly what Origin headers are allowed. Also check for the new possibility of client side remote file inclusion is important. The tools can try to inject some crafted file with ACAO * header.

3.3. Sandboxed Iframes

A normal iframe loads all the contents from the destination, this include HTML CSS and JavaScript HTML5 to introduce the attribute. This new feature allows us to specify what content should be loaded to the iframe, forbidding the execution of JavaScript or popup windows. The aim of the of the html5 attribute sandbox is to allow embed the required third party component and give it only the minimum level of capability necessary to do its task. In addition it is possible to give back the privileges with:

- **allow-forms**: allows form submission.
- **allow-popups:** allows popups (window.open(), showModalDialog(), target="_blank", etc.).
- **allow-pointer-lock**: allows pointer lock.
- **allow-same-origin**: allows the document to maintain its origin; pages loaded from https://example.com/ will retain access to that origin's data.
- **allow-scripts:** allows JavaScript execution, and also allows features to trigger automatically.
- **allow-top-navigation:** allows the document to break out of the frame by navigating the top-level window.

3.3.1. Threats and security issue

Although sandbox iframe was implemented as an innovative feature to address security problems, there are some security concerns bound to it.

Clickjacking [27] is a typical attack that allows an evil page to click on a "victim site" on behalf of the visitor, which the attacker's page overlays the target application's interface with a different interface provided by the attacker. One old defence, Frame busting, it is one of the recommended defense against clickjacking\cite{Clickjac57:online}. It use a frame busting code such as:

```
if (top != self) {
 top.location = self.location;
}
```

If the window finds out that it's not on the top, then it automatically makes itself the top. This code prevent the page to be loaded into an iframe. However that's not reliable anymore, because in sandboxed iframes we can prevent the execution of Javascript so the above code would not work. Even if we include allow-scripts permission the code will still not work, the navigation is forbidden and the change of top.location won't work, "the framed website can still execute JavaScript - but has no privileges to modify the top frame's location."[28].

Modern web browser prevent websites from being loaded into an iframe checking the X-frame-options header. If a page fails to set an appropriate X-Frame-Options %or Content-Security-Policy HTTP header it might be possible for a page controlled by an attacker to load it within an iframe in order to enable a clickjacking attack.

As the x-frame options are set to same-origin it still may be vulnerable to clickjacking attacks due to it only checks the top level frame. This means that if you have nested frames, i.e. frames within frames, it is still possible for another origin to include a site with a X-Frame-Options: SAMEORIGIN header. In this regard the header Content-Security-Policy: frame-ancestors 'self' is better, because it checks all frame ancestors[29].

Also another vulnerability appear due to a security misconfiguration. Setting both allow-scripts and allow-same-origin is not recommended, because it effectively enables an embedded page to break out of all sandboxing. Avoid the usage of allow-same-origin and allow-scripts at the same time.

```
<iframe src="link "
sandbox="allow-top-navigation allow-same-origin allow-scripts\">
</iframe>
```

3.3.2. How to test

Described the two ways to protect a Webpage in front of Clickjacking with iframes, the JavaScript code and the http header, we are going to consider only the http header. Naturally, both defenses should be implemented. The header defense is aimed towards modern browsers, while the JavaScript defense protects legacy browsers.

The lack of the X-frame-options header is considered a CWE-693 vulnerability: Protection Mechanism Failure. We can use the dynamic tools to check if the http x-Frame-Options is present, and inform of the possibly embedding capability of the current tested website.

Also is important to check that an iframe doesn't allow the attributes allow-same-origin and allow-script at the same time, the tools can inform us of this misconfiguration in the http response.

3.4. Based on XSS

The vulnerabilities in this section are strongly related with XSS, where JavaScript is getting more and more important.

3.4.1. XSS through HTML5 tags

HTML5 has several new tags and attributes that can brake XSS filters. Before to HTML5 if you found a XSS hole within a input that has some filers for < > symbols you could not exploit it automatically. However HTML5 lets us execute expressions, for example:

<input type="text" AUTOFOCUS onfocus=alert(1)>

Here with the "autofocus" feature we focus on the element and then the onfucus event to execute our XSS. This works with a lot of tags. You can use this method in any form based element[30]:

```
<input autofocus onfocus=alert(1)>
<select autofocus onfocus=alert(1)>
<textarea autofocus onfocus=alert(1)>
<keygen autofocus onfocus=alert(1)>
```

Also new tags introduced with HTML5 broke XSS filters. In case we can not inject common tags such as <script>, <iframe> because they are blocked by a blacklist filter, HTML5 has introduced lots of new tags that can be used to bypass the blacklists. Here are couple of examples:

Vectors	Browsers
<img src="a" onerror='eval(atob("cHJvbXB0KDEpOw=="))'>	all
<a href="\&\#1;javascript:alert(1)">CLICK ME<a>	all
<video src=_ onloadstart="alert(1)">	all
<input onfocus=write("pepe") autofocus>	all
<svg><animate href=\#x attributeName=href	all

In other cases, with the introduction of the "formaction" attribute with button in HTML5, we can use it in order to execute Javascript.

<button formation='javascript:alert(1)">text</button></form>

In order to protect against all types of XSS, secure input handling must be performed in both the server-side code and the client-side code.

Since this thesis is more focused on HTML5 attacks rather than XSS attacks, we don't go into more depth with this topic. In html5sec.org lists a dozen new XSS vectors are listed.

3.4.2. DOM-based Cross-Site Scripting

DOM Based XSS or also called "type-0 XSS" is an XSS attack wherein the attack payload is executed as a result of modifying the DOM, so in order to understand DOM XSS, we need to describe a bit what DOM is, and why is it relevant to this context. The DOM is a convention for working with objects in an HTML document. When a web page is loaded, the browser creates a Document Object Model of the page with which JavaScript can manipulate and create dynamic HTML [31].

DOMXSS first was documented in a paper by Amit Klein in 2005 and has risen in relevance over the last years due to affecting specifically the client side.

DOM xss happens when untrusted data is added to DOM (or eval'd) through unsafe JavaScript call. Either can be Stored or Reflected, untrusted data can come from client or server. Some of these attacks never reach the server side, and thus cannot be detected by server side code.

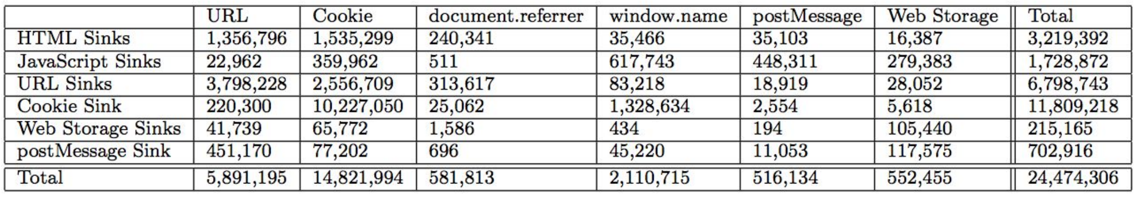
Sources and Sinks are two terms that give meaning to this attack, defined hereafter.

Sources: The source is the injection point for the malicious JavaScript. With DOM XSS, the sources are on the client. Payloads including malicious JavaScript code injected into sources with or without some processing could then reflect in the DOM or execute.

Examples of DOM XSS sources are document.URL, cookies, referer header.

Sinks: The sink is the reflection point that eventually executes (or helps with execution of) the malicious JavaScript injected through the source. These are usually locations on the DOM or Browser Object that can change and invoke code, or they are JavaScript routines that allow direct JavaScript execution.

Examples of easy-to-exploit sinks are eval, document.write, setTimeout.



	URL	Cookie	document.referrer	window.name	postMessage	Web Storage	Total
HTML Sinks	1,356,796	1,535,299	240,341	35,466	35,103	16,387	3,219,392
JavaScript Sinks	22,962	359,962	511	617,743	448,311	279,383	1,728,872
URL Sinks	3,798,228	2,556,709	313,617	83,218	18,919	28,052	6,798,743
Cookie Sink	220,300	10,227,050	25,062	1,328,634	2,554	5,618	11,809,218
Web Storage Sinks	41,739	65,772	1,586	434	194	105,440	215,165
postMessage Sink	451,170	77,202	696	45,220	11,053	117,575	702,916
Total	5,891,195	14,821,994	581,813	2,110,715	516,134	552,455	24,474,306

Figure 3.1: Common sinks/sources[32]

But at the same time this new rich client side introduces some attack vectors. The common flows of DOM XSS are depicted in \ref{fig:sinks}. HTML5 lets attackers do persistent DOM XSS storing not validated data inside local Storage. This kind of attack are described in the next subsections.

A basic example of DOM Xss:

```
var hash = document.location.hash //source
firstName=hash.slice(1)
document.write(firstName) //sink
```

Few more examples on different contexts of this type of XSS can be found on [1]

3.4.3. WebStorage

The main security concern with Local Storage is that the user is not aware of the kind of data that is stored in Local Storage. The user is not able to control Storage respectively access to data stored in Local Storage. The whole access is performed through JavaScript code and, therefore, it is sufficient to execute some JavaScript code in the correct domain context to access all items stored in Local Storage transparently for the user.

Some concern related with the security are derived from how developers could misuse the Webstorage:

- The data in the web storage is not encrypted. This means that any sensitive information stored inside the web storage, such as cookies and code, does not guarantee any integrity of the data.
- If the web application is vulnerable to XSS, the attacker can steal information from the web storage of the website user.
- Web storages do not have HTTPOnly and SecureFlag like cookies. An HTTPOnly flag instructs the browser that only "http" requests should be able to access the cookies, hence preventing cross-site scripting vulnerabilities since the XSS attack vectors heavily rely upon JavaScript. Since web storage is a JavaScript API, it would, by design, allow JavaScript to access the web storage, raising security concerns. Along with HTTPOnly flags, the web storage also does not support SecureFlag.
- If the data stored inside the web storage is being written to the page by using a vulnerable sink, it will result in DOM based XSS vulnerability. A similar issue happens with web messaging.

One example of an insecure implementation of local storage could be the code below, where assignment from localStorage could lead to XSS:

[1] https://github.com/eoftedal/writings/blob/master/published/owasp_top_10_for_js_-_xss.md

```
function action(){
var resource = location.hash.substring(1);
localStorage.setItem("item",resource);
item = localStorage.getItem("item");
document.getElementById("div1").innerHTML=item;
}
</script>
<body onload="action()">
<div id="div1"></div>
</body>
```

In the example above the item extracted without any validation is passed to the innerHTML value.

Other ways to exploit web storage is through a flaw in browsers implementation, that lets unlimited storage using sudbomains upon disc full[33]. Nowadays, the browser patches this attack to negate it.

Further real examples of domXSS vulnerabilities leveraging with localStorage are exposed in[34,35]

Here is an example of illegitimate use of storage capacity of clients to store compromised data [8].

3.4.4. WebMessaging

HTML5 PostMessages (also known as: Web Messaging, or Cross Domain Messaging) is a method of passing arbitrary data between domains. The Messaging API introduced the postMessage() method, with which plain-text messages can be sent cross-origin. It consists of two parameters, message and domain.

However if not implemented correctly it can lead to sensitive information disclosure or cross-site scripting vulnerabilities as it leaves origin validation up to the developer. If the origin is not being checked, any window which can interact with the window will be allowed to send a message regardless of whether it's trusted or not.

Another problem with Cross Window Messaging is the use of insecure DOM methods to display the data, which would obviously result in DOM XSS[36]:

```
//Listener on http://www.examplereceiver.com/
```

```
window.addEventListener("message", function(message){
      //Inexistent or inefective message.origin check
            document.getElementById("message").innerHTML =
message.data;

});
```

To conclude this section, WebMessaging have created a new attack vector for exploiting XSS. Further examples of how harmful could be a bad use of postMessage can be found on [37].

3.4.5. How to test

Blackbox testing for DOM-Based XSS is not usually performed since access to the source code is always available as it needs to be sent to the client to be executed.

The JavaScript web applications differ significantly from other types of applications because they are often dynamically generated by the server. We need to crawl the web to determine all the instances of JavaScript being executed and where user input is accepted. Some inputs obtained from client-side JavaScript objects are window.location, window.referer, etc. If some function was a sink and used one of the former inputs, then the exploitability would depend on the encoding done by the browser.

Automated testing has had only very limited success at identifying and validating DOM-based XSS as it usually identifies XSS by sending a specific payload and attempts to observe it in the server response. Due to their nature, DOM-based XSS vulnerabilities can be executed in many instances without the server being able to determine what is actually being executed. This may make many of the general XSS filtering and detection techniques impotent to such attacks.

The tools need to have knowledge of what is going on with the page on the JavaScript Engine level, static analysis is not reliable anymore for DOM Xss detecting.

To detect DOM XSS automatically, is necessary instrumentation of the browsing engine, especially the JavaScript execution engine, to detect the passage of strings from sources to sinks. This is known as the taint propagation method. All the strings passed from possible sources have to be tracked to see if they land on sinks. Much tools do not implement those techniques what makes impossible for them to detect DOM vulnerabilities.

For our test we are going to consider only cases of DOM XSS that include some of the HTML5 features we are discussing in this chapter. Hence our test cases include examples of DOM XSS where the sink are either local storage or postmessage.

Even though we consider that test the other variant of XSS are also important in our research. The exploitation of some HTML5 features depends directly in the presence of a XSS vulnerability. This XSS might leverage an attacker to get a chained attack poisoning local storage for example. Hence a good evaluation of the evolved capabilities of detecting XSS in general is worth for the testing section.

3.5. WebSocket hijacking

An important concern with WebSockets is that WebSockets are not restrained by the same-origin policy. Thus an attacker can easily initiate a WebSocket request. The user is not able to control or block a request from a malicious webpage targeting the ws:// or wss:// endpoint URL of the attacked service. Due to the fact that the request starts with a HTTP(s) handshake, if the user is authenticated with some cookie the browsers send these cookies and authentication headers along, even cross-site.

Hence, new threat concerning WebSockets are possible. Christian Schneider describes that during the handshake operation, where the handshake request is sent using HTTP-protocol, the hijacking of the WebSocket connection is possible. Hence the name for the attack: Cross-site WebSocket Hijacking (CSWSH). In CSWSH, it is possible to establish a WebSocket connection to a legitimate service from outside the original application, whether or not it uses HTTPS.

A good and quick way to see what is happening within the WebSocket protocol is use through the use of the chrome devtools [2], that in the recent previous have been adapted and redefined for best support.

Some mitigations that modern browsers introduce like chrome are detections of insecure connexion with WebSockets. The web browser blocks insecure connections with ws:// unencrypted protocol within a web using the protocol https due to a mixed content fault, and prompt the next message in console:

```
"websocket:17 Mixed Content: The page at 'https://example.com' was
loaded over HTTPS, but attempted to connect to the insecure
WebSocket endpoint 'ws://echo.websocket.org/'. This request has been
blocked; this endpoint must be available over WSS".
```

3.5.1. How to Test

Commonly available web security testing tools such as Burp Suite or OWASP ZAP are unsuitable for testing the security of a WebSocket implementation automatically[38].

This leaves it as a manual task requiring extensive knowledge of how the implementations work, what the common security issues in them are, and how they can be security tested.

[2] http://octogence.com/blog/websocket-security/

3.6. Other vulnerabilities

Apart from the vulnerabilities detailed above, other vulnerabilities with different degrees of severity emerge which, although not derived directly from an html5 technology, have been expanded or are notorious because of them. They will be mentioned and given a solution.

As in other technologies, frameworks javascripts are exempt from vulnerabilities. As we can see some of them have different CVEs[3][4] Failure to update or use one of these frameworks that contains vulnerabilities may lead to one of the risks of OWASP top 10, more specifically A9, Using Components with Known Vulnerabilities.

On the other hand a well-established CSP guarantee code separation and can help us mitigate widespread vulnerabilities like XSS. But if we do not take into account the guidelines or recommendations when setting the header we may be exposing ourselves to an A5 Security Misconfiguration. Furthermore, the specification for this header has evolved over time. It was implemented as X-Content-Security-Policy in Firefox until version 23 and in IE until version 10, and was implemented as X-Webkit-CSP in Chrome until version 25. Both of the names are deprecated in favor of the now standard name Content Security Policy.

The following are misconfiguration scenarios:

- If default-src is not set or set to wildcard and/or other directives are set to wildcard we have a policy that is too permissive.
- Multiple instances of this header are allowed in same response. Depending on the version of the browser this will understand ones and ignore others. Hence, in order to achieve desired support it is essential that the response include an identical policy with all three names.
- If a directive is repeated within the same instance of the header, all subsequent occurrences are ignored.

I would argue that SPAs are more vulnerable to JSON Hijacking[5](which is a form of CSRF) because many of these applications rely on JSON to pass information. This is an attack that allows a malicious site to retrieve and process data instead of the one way communication you typically see with CSRF. But nowadays browser improvements to do not allow this kind of specific attack.

3.7. Mitigations

Until now we have been showing the bad face of html5 but, as the author states in his research we can improve Modern Web Application Defenses Using HTML5. There are big efforts within html5 to improve the overall security of the web, and in contrast with

[3] https://www.cvedetails.com/vulnerability-list/vendor_id-6538/Jquery.html
[4] https://www.cvedetails.com/vulnerability-list/vendor_id-6541/Prototypejs.html
[5] http://haacked.com/archive/2009/06/25/json-hijacking.aspx/

the last sections is this one we are going to show this new security features, that can prevent some of the previous attacks.

We define the main ways to defend HTML 5 webs as follows :

- **Strong solutions:**

  Clickjacking -- X-Frame-Options

- **Mitigating solutions:**

  HTML injection -- Content Security Policy

  Mixed-Origin content -- CORS, CSP, <iframe> sandbox

  Sniffing -- HSTS

- **Implementation-specific solutions:**

  CSRF -- specific implementations with tokens

This means we defeat exploits rather than vulnerabilities [6]. The mitigations include some secure headers that are getting more and more common. In regards to the adoption of secure headers, the owasp have a great tool for getting usage statics [7]. Use this headers enforce some policies in the browser. Some security headers that if used properly enhance the security of HTML5 applications are:

X-XSS-Protection

Internet Explorer and Google Chrome has inbuilt XSS protection framework, which prevents XSS attack from being executed. These features are enabled in the default state. The X-XSS-Protection header asks the browser to enable or disable those plugins.

- X-XSS-Protection: This prevents XSS from being executed
- X-XSS-Protection: This allows XSS to execute

X-Frame-Options

In the "Sandboxe Iframes" section, we show the possibility of bypassing clickjacking protections af JavaScript framebusting. X-Frame-Options header countless clickjacking vulnerability. The X-Frame-Options header has been made obsolete by the frame-ancestors directive from Content Security Policy Level 2. If a resource has both policies, the frame-ancestors policy SHOULD be enforced and the X-Frame-Options policy SHOULD be ignored [8].

Strict-Transport-Security

[6]https://deadliestwebattacks.files.wordpress.com/2013/09/using-html5-to-make-javascript-mostly-harmless.pdf
[7] https://github.com/oshp/headers
[8] https://www.w3.org/TR/CSP2/\#directive-frame-ancestors

HTTP Strict-Transport-Security (HSTS) enforces secure (HTTP over SSL/TLS) connections to the server. This reduces the impact of bugs in web application leaking session data through cookies and external links and defends against Man-in-the-middle attacks. HSTS also disables the ability for user's to ignore SSL negotiation warnings.

Simple example, using a long (1 year) max-age. This example is dangerous since it lacks includeSubDomains.

Strict-Transport-Security: max-age=31536000

Content-security-policy

HTML 5 introduces the concept of Content Security Policy (CSP). CSP is a HTTP header with directives delivered from the server. Using these directives the authors establish a baseline about what can, and should not, be trusted by the client. In this case, even if an attacker can inject some malicious script, unless it conforms to the defined CSP then the browser will ignore that script.

On the one hand, one problem of CSP is the case that an attacker alter the stream (such as with a classic man in the middle attack) they can simply eliminate, or alter, HTTP headers that define the CSP. By using HTTPS applications this problem can be avoided to a great extent, but communications over HTTP may still be vulnerable to attack.

On the other hand, a CSP disallows any inline JavaScript from executing. This is a good architectural policy to separate instructions from data, making HTML more readable and maintainable.

One advantage of using the new CSP is that you can not only prevent XSS, but you can also detect it[9].

HTTP header enforce, in the client, a least-privilege environment for script and other content using a series of policy directives. The directives below are available as of CSP version 1.0. Each one of them limits the origins from where you can get such content.

The frame-ancestors directive of Content Security Policy (introduced in version 1.1) can allow or disallow embedding of content by potentially hostile pages using iframe. Setting this directive to 'none' is similar to X-Frame-Options: DENY (which is also supported in older browers).

A Best Practice for CSP[10] implementation is establishing a default-src directive. It is always better to start with full lockdown and then start allowing resources that are absolutely needed. To do that set the default-src directive to ‘self’ or ‘none’ and then start adding other directives like script-src, media –src etc. as needed. This would help in making the web application secure at the roots with additional directives serving as exceptions added in order to maintain functionality. This also help in achieving

[9] http://www.madirish.net/556
[10] https://www.html5rocks.com/en/tutorials/security/content-security-policy/

flexibility because if any new source location has to be added/removed, it can be done very easily.

Google has a tools to improve CSP protection for Web Apps[11]. This online tool is very useful for developers because it validates the CSP headers and warns you about bad practices.

3.7. HTML5 recap OWASP Top 10

OWASP top 10 is by far the leading application security standard or guideline followed by builders who took this survey \cite{SANS_Survey2015-io}, \cite{SANSSurv17:online}. In this section we present a classification of the vulnerabilities described referring to Owasp top 10.

Looking through the OWASP list we can see a few interesting security risks. We could discard A1 (Injection) (the WebSql is deprecated), A2 (Broken authentication and session management), A4 (Insecure direct object references) and A7 (Missing function level access control). These risks contains backend implementations and we could not find any usage of these risk with HTML5 functionality. Even though we did not find any usage of these risk this does not mean that they can't be exploited using HTML5.

The classification of the vulnerabilities exposed before referencing the owasp top 10 are the following:

- **A3-XSS**

  TestCors1 (Cors rfi)

  TestDomXssWMessaging

  TestDomXssWStorage

- **A5-Security Misconfiguration**

  TestCors2 (ACAO headers misuses)

  TestIframe2 (To permissive sandboxing directives)

  Content security policy misconfiguration

- **A6-Sensitive Data Exposure}**

[11] https://cspvalidator.org

TestCors2 (ACAO headers misuses)

- **A8-CSRF**

  TestWebsocketsHijacking

- **A9-Using Known Vulnerable Components**

  The use of vulnerable javascripts frameworks

- **A10-Unvalidated Redirects**

  TestIframe1

4. Penetration testing tools

This chapter presents the options available for security web analysis, which dynamic scanners there are, which ones we have chosen and why. Also for each to we provide extra information about what features or plugins provide the tools for test the test cases we expose in chapter 5.

4.1. Open Source Prevention Tools

W3af: w3af is an open-source program written in Python. All functionalities within w3af are implemented as plugins. These functionalities are divided into three different types of plugins. Firstly, crawl plugins are responsible for finding new URLs, forms, and other injection points. The web spider plugin is one of the crawl plugins. It is a classic web spider designed to navigate through the application and extract all URLs from web pages it encounters.

Arachni: Open source pen testing framework written in Ruby and is highly extensible. One of Arachni's most lauded attributes is its scalability and modularity; the tool can be used as a simple command line scanner utility or configured in a high performance scanner grid to support large-scale application security testing routines.

Arachni can handle complex modern web applications providing internally a headless browser-cluster (for apps with lots of JS). Support for JavaScript/DOM/HTML5/AJAX. Also detection of DOM-based vulnerabilities tracing of data and execution flows of DOM and JavaScript environments. Extra tracing optimizations for common JavaScript frameworks

Zap Proxy: Zap is an open source project within the OWASP community. One of the extras of zap is the REST api accessible whenever the java program be working. For extract the results You can access this information via the ZAP API, which has a (basic) HTML interface. Point your browser to the host/port your instance of ZAP is listening on and select: "Local API" / "spider" / "fullResults" Then enter '0' for the scanId and press the 'fullResults' button. You may also need to supply your API key, which is available from the ZAP Options / API screen. You can also change the format of the results if you want - HTML, JSON and XML are all supported.

ZAP provide several plugins that make the tool able to detect and handle with modern web technologies.

4.2. Licensed Tools

Netsparker: Full HTML5 Support The Netsparker scanners have a state of art Chrome based crawler that enables them to simulate JavaScript and DOM on pages, thus they can fully understand and crawl all types of modern Web 2.0, HTML5 web applications and Single Page Applications and identify vulnerabilities and security flaws in them.

Netsparker is also able to identify web application vulnerabilities which are typically associated with modern HTML5 web applications, such as DOM based cross-site scripting. The vulnerabilities that detect or warn are Misconfigured Access-Control-Allow-Origin Header, DOM based Cross-site Scripting (XSS), Misconfigured Frame

Acunetix: Is a commercial tool that provide a Web Interface and windows program. It has support for AJAX, WSDL and their results can now be imported into a Web Application Firewall (WAF). Acunetix DeepScan also supports scanning of Single Page Applications (SPA). They claim to be able to crawl and then scan this technologies, thanks to its DeepScan technology, Acunetix can crawl any website and web application, even modern Single Page Application (SPAs) developed using HTML5, JavaScript and RESTful APIs. Acunetix DeepScan crawls even the most advanced web applications by replicating user actions and executing JavaScript just like a real browser does. Acunetix also includes a fully automated web browser that can understand, and interact with complex web technologies. This internal browser crawl and scan HTML5 web applications, and execute JavaScript like a real browser. Interacts with AJAX, SOAP/WSDL, SOAP/WCF, REST/WADL, XML, JSON, Google Web Toolkit (GWT) and CRUD operations. Analyses web applications developed in Ruby on Rails and Java Frameworks including Java Server Faces (JSF), Spring and Struts.

Burp Suit: Burp is a graphical tool for testing Web application security. The tool is written in Java and developed by PortSwigger Security. Similar to Zap, burp offer a proxy to intercept to request in the free version. A full version also have crawling and automating testing capabilities.

List contains the definitions of all issues that can be detected by Burp Scanner can be found on[12] . This list include several HTML5 related vulnerabilities.

4.3. Other tools

DOMinator Pro[13] is a great semi-automated tool for identifying DOM XSS. Is a tool or rather a runtime analyzer written with Nodejs that use a chrome browser to discover DOM Xss issues. Is worth mention this tools because is the only one that can deal with complex JavaScript environment that contain DOM Xss.

[12] https://portswigger.net/knowledgebase/issues
[13] https://www.blueclosure.com/

Despite being used in the most popular websites that we frequent, JavaScript does, unfortunately, come with its own unique risks. Dynamic code analysis is the best way to ensure that security vulnerabilities don't make it into your Javascript code. % and that security is a top priority in every element of the software development life cycle.

4.4. Limitations

This research is subjected to several limitations. First of all, the experiments we conducted with several dynamic vulnerability scanners were done with no prior knowledge or experience with those scanners. We cannot guarantee that our use of the scanners is the optimal way, and therefore all findings are merely ours and do not necessarily reflect findings of other developers.

In addition, if you generate results for a commercial tool, be careful who you distribute it to. Each tool has its own license defining when any results it produces can be released/made public. It is likely to be against the terms of a commercial tool's license to publicly release that tool's score against your own tests or Benchmark.

In the Owasp web it is not recommended Black box testing for issues within the new HTML5 technologies since access to the source code is always available as it needs to be sent to the client to be executed.

Furthermore, crawling some web app are becoming a problem for some tools. This tools in most cases do no support the navigation method of recent JavaScript frameworks.

Web scanners general approach is to inject payload in the web page inputs and check the received HTML response for possible cross-site scripting vulnerabilities, but for instance in a DOM based XSS this does not works because of the fact that the vulnerability take place during the rendering of the page, injected data appears only in the rendered response.

TABLE 4.1: Input vectors support

	OWASP Zap	W3af	Arachni
HTTP Query String Parameters	Yes	Yes	Yes
HTTP Body Parameters	Yes	Yes	Yes
HTTP Cookie Parameters	Yes	Yes	Yes
HTTP Headers	Yes	Yes	Yes
HTTP Parameters Name			Yes
XML Element Content	Yes		Yes
XML Attributes	Yes		Yes
XML Tags			
JSON Parameters	Yes		Yes
Flash Action Message Format			
Custom Input Vector	Yes		Yes
Google Web Toolkit	Yes		
Summary	9	4	9

4.5. List of Chosen Web Application Scanners

The following open source scanners were covered in the benchmark and tests. These scanners have been chosen for further analysis due to the easy access of their source code and because the wide range of supported input vectors as we can se on table \ref{table:inputVectors}. The input delivery method (a.k.a the input vector) is the method used by the HTML/Flash/Applet/Silverlight application to deliver user-originating input from the client to the server. In the investigating field of DAST tools, Is considered the scanner's to support as much as application input delivery method for testing. Consequently this is a significant aspect in the selection process of any scanner.

The selected tools are all free, without any restriction or limitation what make easy the task of install the tools. Also all this tools have multiplatform support:

- Zed Attack Proxy (ZAP) v2.6.0 , changed to ZAP Multi-2017-06-05 (OWASP)
- W3AF v1.7.6 revision 27b1516a3f (The W3AF team)
- arachni v 1.5.1 webUI 0.5.12(Tasos Laskos)

5. Websites for tests

After defining the limitations of the tools with respect to the new technologies in this section we are going to find, compare and define a set of tests to evaluate the reliability and improvements made to the tools during the past years to face the vulnerabilities. Testing the tools consist of several tests. We have created a small test to show the support of HTML5 features by the tools, two existing benchmarks, and two web applications.

5.1. HTML5 Web Test

From the extracted test sets that we presented in chapter 3 we have implemented a website. The vulnerabilities are implemented in its simplest form in order to guarantee some basic principles:

- Each test case focuses on a specific property, thus having a clear test goal which is more convenient in evaluation.
- The test cases omit all vulnerability-irrelevant language features and external dependencies, making them easier to deploy.
- The test cases scale are not realistic applications, which reduces the cost of detecting and facilitates rapid testing

Some other vulnerabilities reviewed like CRSF with CORS are not implemented because it depends on an specific conditions that rather in any web are satisfied. Even trying to implement this deliberately are difficult, the browsers don't allow ACAO * header with allow credentials for secure reasons, so if the user is authenticated, this session cookie or header wouldn't be send to the server. CORS can't lead to a real CRSF attack unless this attack take place in a non authorised web or the server would be misconfigured echoing the origin of any CORS request submitted.

5.2. Other Web for Testing

This sections provides an updated list of vulnerable web applications which you can test the pen-testing tools. The vulnerable web applications have been classified in three categories: offline, VMs/ISOs, and online. For each web we also show the technologies that implement. Another category are the case of the websites for benchmarking purposes.

Wackopickto: is a website written by Adam Doupé. It contains known and common vulnerabilities (XSS vulnerabilities, SQL injections, command-line injections, sessionID vulnerabilities, file inclusions, parameters manipulation, ...). It is intended to be a realistic website.

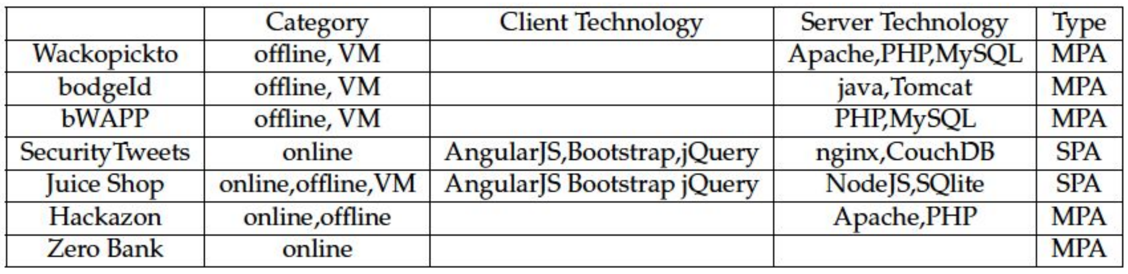
bodgeId: The bodgeId is an open source vulnerable web designed as a simulation of a webshop to help people learn about penetration testing.

bWAPP: free and open source deliberately insecure web application. Written in PHP uses a MySQL database. It has over 100 web bugs including some related to HTML5 but only for CORS. The source code is accessible for downloading. Another possibility is to download the bee-box, a custom Linux VM pre-installed with bWAPP[14].

SecurityTweets: web application was built as a Single Page Application (SPA) using modern web technologies such as AngularJs, Bootstrap, CouchDB, Flask and Nginx. Is the web test of acunetix.

Juice Shop: is written in Node.js, Express and AngularJS. It was the first application written entirely in JavaScript listed in the OWASP VWA Directory. There is an online version[15] but also the source code is disposable in github for local deploying. The web imitates a real online shop with some challenges for security learning.

Hackazon: Hackazon is a vulnerable test site that is an online storefront built with the same technologies used in modern client and mobile applications. Hackazon has an AJAX interface and RESTful API's used by a companion mobile app. Hackazon enables users to configure each area of the application in order to change the vulnerability landscape to prevent well known vulnerabilities. The application have a online version and also the code are disponible in github[16] in order to install it locally.



	Category	Client Technology	Server Technology	Type
Wackopickto	offline, VM		Apache,PHP,MySQL	MPA
bodgeId	offline, VM		java,Tomcat	MPA
bWAPP	offline, VM		PHP,MySQL	MPA
SecurityTweets	online	AngularJS,Bootstrap,jQuery	nginx,CouchDB	SPA
Juice Shop	online,offline,VM	AngularJS Bootstrap jQuery	NodeJS,SQlite	SPA
Hackazon	online,offline		Apache,PHP	MPA
Zero Bank	online			MPA

Figure 5.1: Summary of webs classified by technologies

[14] http://www.itsecgames.com/
[15] https://juice-shop.herokuapp.com
[16] https://github.com/rapid7/hackazon

Other web apps might be webgoat or multillidae 2, but all of these have the problem that are designed for training and they don't implement vulnerabilities from new of HTML5.

5.3. Beachmark frameworks

In order to see the level of performance of each tool is interesting compare these with some kind of test suite designed to evaluate the speed, coverage, and accuracy. There are already several Beachmarks, we also review this benchmark projects in order to show the most recent results for the tools selected.

WIVET: [39]an open-source benchmark for web link extractors includes a set of tests which test different methods that can be used to access a website. When scanners are mainly used in a point-and-shoot scenario, where is preferible as much automation as possible, a high WIVET score will be the second most important feature you should follow. WIVET tests Link Extraction capabilities of the tools.. WIVET contains 54 tests and assigns a final score to a crawler based on the percent of tests that it passes. The tests require scanners to analyze simple links, multi-page forms, links in comments and JavaScript actions on a variety of HTML elements. There are also AJAX-based tests as well as Flash-based tests. In our tests, we used WIVET version number 129.

WAVSEP: is vulnerable web application designed to help assessing the features, quality and accuracy of web application vulnerability scanners. This evaluation platform contains a collection of unique vulnerable web pages that can be used to test the various properties of web application scanners[21].

OWASP Benchmark:The OWASP Benchmark for Security Automation (OWASP Benchmark) is a free and open test suite designed to evaluate the speed, coverage, and accuracy of automated software vulnerability detection tools and services. You can use the OWASP Benchmark with Static Application Security Testing (SAST) tools, Dynamic Application Security Testing (DAST) tools like OWASP ZAP and Interactive Application Security Testing (IAST) tools. The benchmark includes false positives test cases that used cookies as a source of data that flowed into XSS vulnerabilities. The Benchmark treated these tests as False Positives because the Benchmark team figured that you'd have to use an XSS vulnerability in the first place to set the cookie value, and so it wasn't fair/reasonable to consider an XSS vulnerability whose data source was a cookie value as actually exploitable

5.4. Limitations and Websites Chosen

A considerable number of vulnerable web applications already existed before. We have give a list of these applications. Most of them are only server-side rendered applications. But Rich Internet Application (RIA) or Single Page Application (SPA) style applications were already a commodity at that time. Juice Shop is constructed as SPA as was meant to fill that gap.

We have not chose Juice Shop because at the moment Zap proxy POST Data field supports only application/x-www-form-urlencoded. You cannot authenticate this web that expect a post with JSON.

As far as zap is concerned, automatic crawling of web typo ecommerce applications, where there are lists of products that are updated based on ajax requests (such as the hackazon testing website) is not feasible. Having no clear way to add redundant urls, the application iterated infinitely by the different url of the products (url example).

In addition, the rules have been limited to look for vulnerability to XSS types only because when activating all rules, the process could take hours or even not finish. This decision of only executing a test for XSS will be maintained for other applications, and although it limits the capacities of the tools very much. We assume that for this thesis it is a correct decision, as it has been mentioned before, the XSS vulnerabilities continue being the most exploited today and are key to obtaining information from the client.

Also we do not use the hackazon page because it has some vulnerabilities that have to be exploited in a specific workflow. They cannot be directly attacked without respecting the order of execution.

For test the crawling capabilities we have chosen WIVET because it is easy to deploy. Also for see the effective crawling in a more realistic web we use SecureTweets. Lastly we using OWASP benchmark to get an idea of the overall accuracy of the tools.

6. Conclusion

In this Chapter we show the results of the test and give a conclusion.

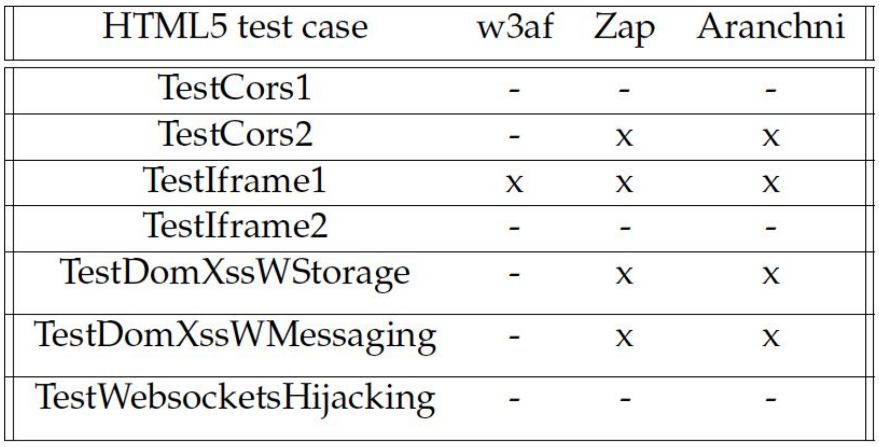
6.1. Test Conclusions

In this section we are going to show the test result. Of the we only talk about the most significant ones. Other test are performed also but without the expected results.

6.1.1. HTML5 test results

HTML5 Test

In this test the three tools have been executed against the HTML5 test cases designed in section 5.1. This evaluate several test case of HTML5 vulnerabilities. As we can see from the table 6.1 of obtained results, w3af cannot discover any of the vulnerabilities related with manipulations of the DOM. Any tool are able to execute the Javascript code to reveal the client file inclusion of the TestCors1. Even though Zap and Arachni have reported the Dom vulnerabilities, does not mean those can discover any DOM xss. The test cases implemented only consider a basic flow of this vulnerability taking as source the url query parameter. In another more complex attach the vulnerability might not have been detected. As regards the detection of websocket hijacking, any tool can automatically detecting it. w3af even though having a plugin for this purpose, only report the presence of websockets link. Zap allows you to interact manually with websockets using a plugin.



HTML5 test case	w3af	Zap	Aranchni
TestCors1	-	-	-
TestCors2	-	x	x
TestIframe1	x	x	x
TestIframe2	-	-	-
TestDomXssWStorage	-	x	x
TestDomXssWMessaging	-	x	x
TestWebsocketsHijacking	-	-	-

Figure 6.1: HTML5 test features results

6.1.2. Crawling results

WIVET Test

We tested the scanners on WIVET. For each connection received, WIVET will create a session, which needs to be maintained throughout the test.

There are 2 ways that a session can be disconnected:

*Click on the Logout link (/logout.php)

*Browse to /pages/100.php

So each tool should be setup in a specific way. For instance w3af can see a cookie session using a file that contains the value of the cookie. Arachni has a argument if you execute this from the command line "-http-cookie-string="PHPSESSID=".

In the WIVET test the clear winner is arachni. Thanks to its internal browser, arachni can monitor the DOM to see changes or new events, execute them and find hidden links in them.

Following is Zap Proxy, although in our own test we found a bug in the ajax spider that has been reported[17], we take as a reference the results of the page [18]where zap developers publish for each new version some updated results. The ajax Spider crash while crawling a given a url where an alert box is displayed.

We started using the app specific installation for mac os, but after several problems regarding about connecting the app with web browsers and out of memory problems, we get the multiplatform java executable. This is lauched using a script that tune the memory levels and contain all the latest selenium driver to automate external browser direct control.

It has very complex interface and I have encountered some accessibility problems while working with it.

The last one would be w3af, that without having a spider that interprets JavaScript, is only able to extract static links of the http response code.

Therefore in the graph\ref{fig:coverage}, the tool that dominates all would be aranchni, and links extracted by Zap would be a subset of its parent node, just as it occurs with w3af.

[17] https://github.com/zaproxy/zaproxy/issues/3412
[18] http://zapbot.github.io/zap-mgmt-scripts/scans.html

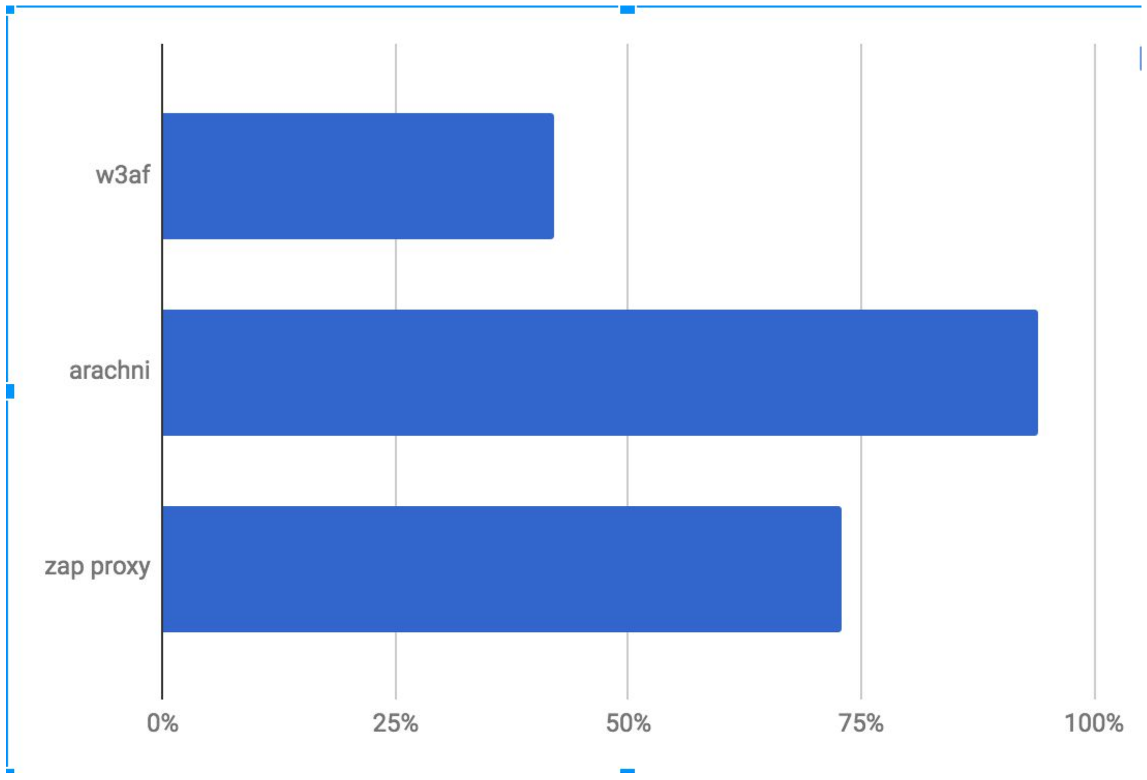


Figure 6.2: Coverage wivet

SPA Test

Today a large number of web applications are template-driven, meaning the same code or path generates millions of URLs. For a security scanner, it just needs one of the millions of URLs generated by the same code or path.

The test HTML 5 website, SecureTweets, was built as a SPA web application. Its URLs are designed to look like:

- http://testhtml5.vulnweb.com/\#/popular
- http://testhtml5.vulnweb.com/\#/latest
- http://testhtml5.vulnweb.com/\#/latest/page/1
- http://testhtml5.vulnweb.com/\#/carousel
- http://testhtml5.vulnweb.com/\#/archive
- http://testhtml5.vulnweb.com/\#/about

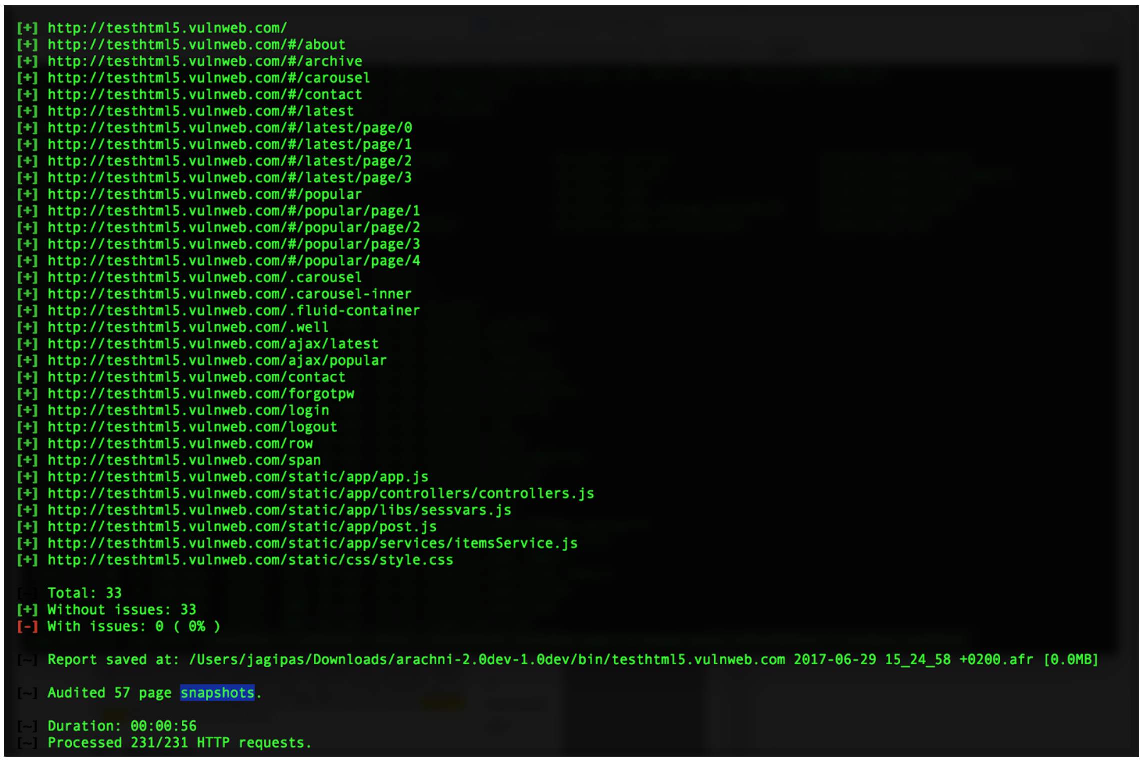
All the URLs shown above are using the location hash to determine the target page. There is only one real page (/) and this is page is loading various sections of the website by using the value of the location hash parameter. The web server doesn't see any of the URLs above, everything is happening only in the client's browser and the page is not reloaded.

Zap Proxy results:

```
http://testhtml5.vulnweb.com/
http://testhtml5.vulnweb.com/favicon.ico
http://testhtml5.vulnweb.com/login
http://testhtml5.vulnweb.com/logout
http://testhtml5.vulnweb.com/static/app/app.js
http://testhtml5.vulnweb.com/static/app/controllers/controllers.js
http://testhtml5.vulnweb.com/static/app/libs/sessvars.js
http://testhtml5.vulnweb.com/static/app/post.js
http://testhtml5.vulnweb.com/static/app/services/itemsService.js
http://testhtml5.vulnweb.com/static/css/style.css
http://testhtml5.vulnweb.com/static/img/logo2.png
```

Crawling SPA is failing in zap proxy. Zap does not recognize the fragment links that are not a complete page in html as another page or state. The ajax spider Zap is unreliable, through a browser automatically managed by the drivers selenium, it tries to click and fill forms to reach all possible pages. This implementation is more like a program to test the usability of the U.I. than a crawler.

Arachni results:



```
[+] http://testhtml5.vulnweb.com/
[+] http://testhtml5.vulnweb.com/#/about
[+] http://testhtml5.vulnweb.com/#/archive
[+] http://testhtml5.vulnweb.com/#/carousel
[+] http://testhtml5.vulnweb.com/#/contact
[+] http://testhtml5.vulnweb.com/#/latest
[+] http://testhtml5.vulnweb.com/#/latest/page/0
[+] http://testhtml5.vulnweb.com/#/latest/page/1
[+] http://testhtml5.vulnweb.com/#/latest/page/2
[+] http://testhtml5.vulnweb.com/#/latest/page/3
[+] http://testhtml5.vulnweb.com/#/popular
[+] http://testhtml5.vulnweb.com/#/popular/page/1
[+] http://testhtml5.vulnweb.com/#/popular/page/2
[+] http://testhtml5.vulnweb.com/#/popular/page/3
[+] http://testhtml5.vulnweb.com/#/popular/page/4
[+] http://testhtml5.vulnweb.com/.carousel
[+] http://testhtml5.vulnweb.com/.carousel-inner
[+] http://testhtml5.vulnweb.com/.fluid-container
[+] http://testhtml5.vulnweb.com/.well
[+] http://testhtml5.vulnweb.com/ajax/latest
[+] http://testhtml5.vulnweb.com/ajax/popular
[+] http://testhtml5.vulnweb.com/contact
[+] http://testhtml5.vulnweb.com/forgotpw
[+] http://testhtml5.vulnweb.com/login
[+] http://testhtml5.vulnweb.com/logout
[+] http://testhtml5.vulnweb.com/row
[+] http://testhtml5.vulnweb.com/span
[+] http://testhtml5.vulnweb.com/static/app/app.js
[+] http://testhtml5.vulnweb.com/static/app/controllers/controllers.js
[+] http://testhtml5.vulnweb.com/static/app/libs/sessvars.js
[+] http://testhtml5.vulnweb.com/static/app/post.js
[+] http://testhtml5.vulnweb.com/static/app/services/itemsService.js
[+] http://testhtml5.vulnweb.com/static/css/style.css

[~] Total: 33
[+] Without issues: 33
[-] With issues: 0 ( 0% )

[~] Report saved at: /Users/jagipas/Downloads/arachni-2.0dev-1.0dev/bin/testhtml5.vulnweb.com 2017-06-29 15_24_58 +0200.afr [0.0MB]

[~] Audited 57 page snapshots.

[~] Duration: 00:00:56
[~] Processed 231/231 HTTP requests.
```

By utilizing client-side code through its built-in browser, Arachni is able to crawl not only static links in the HTML code, but also DOM links created by arbitrary client-side code. This form of link can not be detected by just statically analyzing received HTML. Since Arachni also analyses changes to the DOM, it is able to detect these. As we see in the arachni results, arachni get the best result crawling SecureTweets.

Automatic scanning of SecureTweets with w3af is not feasible. Their crawling spider only has the ability to find links in static HTML. It does not handle dynamic links created through Javascript events or ajax requests. It also does not understand pages created with templates. For example, in SecureTweets through the index received after the \# a request is made that downloads a partial HTML that angular renders. W3af does not interpret these partial changes as new pages.

Benchmark OWASP Test

In this test the three tools have been executed against the benchmark designed by Owasp. This benchmark evaluates several categories of vulnerabilities, in our case we have evaluated only those related to xss. More specifically the page has a total of 455 vulnerabilities reflected XSS.

There are four possible test outcomes in the Benchmark:

- Tool correctly identifies a real vulnerability (True Positive - TP)
- Tool fails to identify a real vulnerability (False Negative - FN)
- Tool correctly ignores a false alarm (True Negative - TN)
- Tool fails to ignore a false alarm (False Positive - FP)

Formulas:

True Positive Rate (TPR) = TP / (TP + FN)

False Positive Rate (FPR) = FP / (FP + TN)

False Positive Rate (FPR) = FP / (FP + TN)

It should be noted that w3af was not included for evaluation in this test because the benchmark do not provide a parser for its results. In figure 6.4 we can see a graphical representation that describes how close is the tool to the ideal XSS detection rate[40].

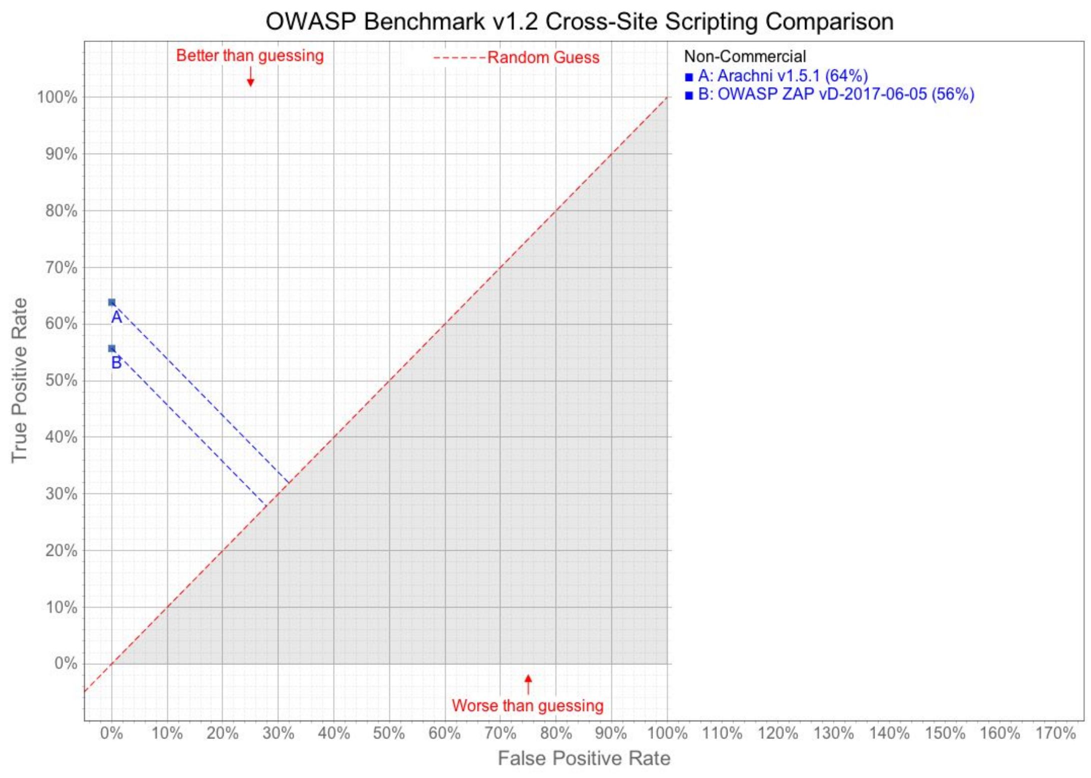


Figure 6.4: XSS accuracy results OWASP benchmark

In table 6.4 we can see the detailed values of true positives, false negatives, false positives, true negatives and the corresponding rates.

TABLE 6.4: Detailed benchmark results data

Tool	TP	FN	TN	FP	Total	TPR	FPR	Score
Arachni v1.5.1	157	89	209	0	455	63.82%	0.00%	63.82%
OWASP ZAP vD-2017-06-05	137	109	209	0	455	55.69%	0.00%	55.69%

6.2 Main Conclusions

What are the new vulnerabilities in HTML5 technologies?

Unless there are a lot of information regarding of security in HTML5 this is very widespread and difficult to search. The results were somewhat were discouraging. We could not find a good vulnerable code example and most security commentaries boiled down to either “there is no major risk” or “if developers use HTML5 properly there is no risk.”.

Moreover When I started this work, I did not realize that new technologies like APIs and frameworks are maybe more important than new HTML5 features. As we see the nowadays modern webs are getting more and more dependent of APIs. The APIs are the foundation of a secure web so is more important to understand the risks of those.

Nevertheless, HTML5 brings new risks and preserves old vulnerabilities in new and interesting ways, but a large responsibility for those weaknesses lies with developers who would misuse an HTML5 feature in the same way they might have misused XHR and JSONP in the past. From our research we can figure out some new possible attacks like stealing credentials with CORS request, or more sophisticated XSS exploits using WebMessaging and WebSockets to scour data from a compromised browser.

The vulnerabilities detected by the tools are based on the well-known OWASP Top Ten, or CWE. We have seen that currently some specific HTML5 vulnerabilities like the ones related with websocket, still without a good testing support within the different scanners.

Can the tools crawling new javascript frameworks aka SPA webpages?

This thesis has shown that none of the web scanners are perfect doing his task in a SPA. Traditionally, crawlers use Unified Resource Locators (URLs) to navigate through the web. But in a Single Page Application JavaScript manipulates de DOM and crate new pages dynamically. The states of the web page are not based on the url anymore. To do effective crawling, DAST tools should adopt features like the DOM Crawling of Arachni. One disadvantage of this approach is that consume a lot more time.

It has been observed that the main limitation of these tools is crawling web application with javascript frameworks like angular

We realized that the DAST tool use relatively highs amounts of process power due to in some webs the rate of request/response sent can become of more than 20 per second. Hence this tools need a better environment to work well, desktop pc with at least 8 gigas of RAM and multicore CPU.

Another finding is the lack of effective automated tools client side. There currently are not any automated tools that effectively scan client-side and server-side JavaScript, performing dataflow analysis and not just pattern-matching, and understanding the thousand and one frameworks.

Some tools like Zap have already a good support of pluggins for check some of the threads exposed in Chapter 3. But from the perspective of a inexperienced user, the process to manage this plugins are not user friendly and requires a little of knowledge. Other tools like arachni have more client side technologies support but any of the tools can testing in a automated way Websockets technology

Overall discussion for the research question

The tools have limitation understanding modern web apps. There are simply too many vulnerabilities in modern HTML5 webs which requires a certain logic. DAST tools are recommended as an initial check for a modern web app but together with a manual penetration test.

Application technologies overlooked by most web application scanners include JSON and REST. The application language is one of the factors when choosing a web scanner. For example, if the application uses extensive AJAX code, a scanner that cannot crawl AJAX will not be a good choice. Dynamic technologies, such AJAX, generally challenge DAST tools. DAST tools send requests to the application and analyse responses. Since AJAX is used on the client-side to create asynchronous web applications, a DAST tool have to execute

AJAX scripts before it can send a request. In classic web application, when a vulnerability is discovered, DAST tools would reference the page and parameter where the issue was found. This cannot be applied to AJAX application because everything is often presented as a single page with many possible user events. The vulnerability may rely on a certain combination of steps to occur before it exists, which makes automated scanning a challenge. To detect a vulnerability hidden behind AJAX/JSON code, a scanner needs to execute AJAX code and process and analyse JSON parameters.

6.3. Future work

One future work is a evaluation of the tools in terms of some requirements specification. Different metrics can be used to compare the web scanners. For example, only comparing the amount of false positives. Another way is by comparing the vulnerabilities found by the scanners with the actual vulnerabilities that exists on the custom built website. A custom built web application is required to run the scanners on. The most interesting would be do an evaluation on a more realistic web application. One idea is make one web app based on some other website for testing, for instance Wackopicko, but Migrating this to a modern web app with more recent technologies.

Regarding the HTML5 vulnerabilities, would be interesting build an extension plugin for some tool like Zap, consisting in a audit plugin for HTML5 features.

Other idea is use well known realistic test webs used in other paper like wackopicko, bodgeit, all of them MPA. Test again this webs with the tools might give an idea of how the tools have improved. Maybe the maturity of scanners has reached the point where they are able to detect almost all of this well-know pages.

Because commercial tools have the latest state of art, features Performing an evaluation of the commercial scanners would be very interesting as these are presumably the most effective web scanners.

Bibliography

1. Shah S. Html5 top 10 threats stealth attacks and silent exploits. BlackHat Europe. 2012;

2. Jobs S. Thoughts on flash, Apple. 2010; Available: https://www.apple.com/hotnews/thoughts-on-flash

3. Fraternali P, Rossi G, Sánchez-Figueroa F. Rich internet applications. IEEE Internet Comput. IEEE; 2010;14: 9–12.

4. WHATWG Living Standard [Internet]. Available: https://html.spec.whatwg.org/multipage/

5. CVE - Common Vulnerabilities and Exposures (CVE) [Internet]. Available: https://cve.mitre.org/

6. Kuppan - Presentation at Black Hat L, 2010. Attacking with HTML5. media.blackhat.com. 2010; Available: https://media.blackhat.com/bh-ad-10/Kuppan/Blackhat-AD-2010-Kuppan-Attacking-with-HTML5-slides.pdf

7. Powermapper. HTML Version Statistics [Internet]. Available: https://try.powermapper.com/Stats/HtmlVersions

8. Bogaard D, Johnson D, Parody R. Browser web storage vulnerability investigation: html5 localstorage object. Proceedings of the International Conference on Security and Management (SAM). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp); 2012. p. 1.

9. Lekies S, Johns M. Lightweight Integriti Protection for Web Storage-driven Content Caching [Internet]. Available: http://www.w2spconf.com/2012/papers/w2sp12-final8.pdf

10. Son S, Shmatikov V. The postman always rings twice: Attacking and defending postmessage in html5 websites. 2013.

11. Karlström J. The WebSocket Protocol and Security: Best Practices and Worst Weaknesses. 2015;

12. Bai J, Wang W, Lu M, Wang H, Wang J. TD-WS: a threat detection tool of WebSocket and Web Storage in HTML5 websites. Security and Communication Networks. Wiley Online Library; 2016;

13. Dong G, Zhang Y, Wang X, Wang P, Liu L. Detecting cross site scripting vulnerabilities introduced by HTML5. 2014 11th International Joint Conference on Computer Science and Software Engineering (JCSSE). 2014. pp. 319–323. doi:10.1109/JCSSE.2014.6841888

14. Kim I-A, Cho K-H, Yim H-J, Kim H-K, Lee K-C. Hadoop-based Crawling and

Detection of New HTML5 Vulnerabilities on Public Institutions' Web Sites. Indian J Sci Technol. 2015;8.

15. Gupta S, Gupta BB. JS-SAN: defense mechanism for HTML5-based web applications against JavaScript code injection vulnerabilities. Security and Communication Networks. Wiley Online Library; 2016;9: 1477–1495.

16. AbuSeada W. Alternative Approach to Automate Detection of DOM-XSS Vulnerabilities.

17. Doupé A, Cavedon L, Kruegel C, Vigna G. Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner. Presented as part of the 21st USENIX Security Symposium (USENIX Security 12). Bellevue, WA: USENIX; 2012. pp. 523–538. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/doupe

18. SANS Survey A. 2015 State of Application Security: Closing the Gap [Internet]. May 2015. Available: https://www.sans.org/reading-room/whitepapers/analyst/2015-state-application-security-closing-gap-35942

19. Doupé A, Cova M, Vigna G. Why Johnny can't pentest: An analysis of black-box web vulnerability scanners. International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer; 2010. pp. 111–131.

20. Bau J, Bursztein E, Gupta D, Mitchell J. State of the art: Automated black-box web application vulnerability testing. Security and Privacy (SP), 2010 IEEE Symposium on. IEEE; 2010. pp. 332–345.

21. Chen S. A Comparison of Prices vs. Features of Web Application Vulnerability Scanners - WAVSEP Benchmark [Internet]. Available: http://www.sectoolmarket.com/price-and-feature-comparison-of-web-application-scanners-unified-list.html

22. Symantec. Internet Security Threat Report [Internet]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf

23. Cynet. Critical Issue Opened Private Chats of FACEBOOK MESSENGER Users up to Attackers [Internet]. Available: http://www.cynet.com/blog-facebook-originull/

24. PortSwigger Web Security Blog: Exploiting CORS Misconfigurations for Bitcoins and Bounties [Internet]. Available: http://blog.portswigger.net/2016/10/exploiting-cors-misconfigurations-for.html

25. Dave Wichers ES Paul Petefish. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet - OWASP [Internet]. Available: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

26. Aboukir Y. Security impact of a misconfigured CORS implementation [Internet]. Available: http://yassineaboukir.com/blog/security-impact-of-a-misconfigured-cors-implem

entation/

27. Clickjacking - OWASP [Internet]. Available: https://www.owasp.org/index.php/Clickjacking

28. Cure93. HTML5 Security Cheatsheet [Internet]. Available: https://html5sec.org/

29. Security SLWA. Headers to block iframe loading [Internet]. Available: https://www.sjoerdlangkemper.nl/2016/07/20/block-iframe-loading/

30. Exploiting XSS - Injecting into Tag Attributes | Burp Suite Support Center [Internet]. Available: https://support.portswigger.net/customer/en/portal/articles/2325988-exploiting-xss---injecting-into-tag-attributes

31. HTML5 DOM w3schools [Internet]. 2017. Available: https://www.w3schools.com/js/js_htmldom.asp

32. Lekies S, Stock B, Johns M. 25 million flows later: large-scale detection of DOM-based XSS. Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. ACM; 2013. pp. 1193–1204.

33. Introducing the HTML5 Hard Disk Filler API » Feross.org [Internet]. Available: https://feross.org/fill-disk/

34. Web Storage Security - WhiteHat Security [Internet]. Available: https://www.whitehatsec.com/blog/web-storage-security/

35. How a Platform Using HTML5 Can Affect the Security of Your Website - Security Musings [Internet]. Available: http://securitymusings.com/article/3159/how-a-platform-using-html5-can-affect-the-security-of-your-website

36. The pitfalls of postMessage [Internet]. Available: https://labs.detectify.com/2016/12/08/the-pitfalls-of-postmessage/

37. postMessage XSS on a million sites [Internet]. Available: https://labs.detectify.com/2016/12/15/postmessage-xss-on-a-million-sites/

38. Kuosmanen H, Others. Security Testing of WebSockets. Jyväskylän ammattikorkeakoulu; 2016;

39. Tatli Eİ, Urgun B. WIVET—Benchmarking Coverage Qualities of Web Crawlers. Comput J. Oxford University Press; 2017;60: 555–572.

40. Benchmark - OWASP [Internet]. Available: https://www.owasp.org/index.php/Benchmark