July 2011

# A Security Analysis of Next Generation Web Standards


enisa
European Network and Information Security Agency

## About ENISA

The European Network and Information Security Agency (ENISA) is an EU agency created to advance the functioning of the internal market. ENISA is a centre of expertise for the European Member States and European institutions in network and information security, giving advice and recommendations and acting as a switchboard of information for good practices. Moreover, the agency facilitates contacts between the European institutions, the Member States, private business and industry actors.

## Contact details:

**Editors:** Dr. Giles Hogben (giles.hogbenQenisa.europa.eu) and Dr. Marnix Dekker (marnix.dekkerQenisa.europa.eu)

**Authors:** Philippe De Ryck, Lieven Desmet, Pieter Philippaerts, and Frank Piessens, Katholieke Universiteit Leuven

**Media Inquiries:** Ulf Bergstrom (ulf.bergstromQenisa.europa.eu).

## Credits

The security analysis was performed by the DistriNet Research Group, K.U.Leuven, Belgium, and in particular by Prof. Dr. ir. Frank Piessens, Dr. ir. Lieven Desmet, Dr. Pieter Philippaerts and Philippe De Ryck.

## Acknowledgements

We would like to thank Thomas Roessler and Michael Smith from W3C for interesting discussions and additional pointers while reviewing the security analysis.

# Executive summary

The web browser is arguably the most security-critical component in our information infrastructure. It has become the channel through which most of our information passes. Banking, social networking, shopping, navigation, card payments, managing high value cloud services and even critical infrastructures such as power networks – almost any activity you can imagine now takes place within a browser window.

This has made browsers an increasingly juicy target for cyber-attacks: the volume of web-based attacks per day increased by 93% in 2010 compared to 2009, with 40 million attacks a day recorded for September 2010 (Symantec Threat Report, 2010). Many more complex threats such as DDoS attacks using botnets rely on flaws in web browsers, which allow the installation of malware. Even if the root cause is elsewhere, the browser is often in a position to protect the user – e.g. in combatting phishing and pharming etc.

The standards which govern the browser – and hence its security - are currently undergoing a major transformation. In order to accommodate innovations in web applications and their business models, a raft of new standards is currently being developed. These include an overhaul of HTML (HTML5), cross-origin communication standards such as CORS and XHR, standards for access to local data such as geo-location, local storage and packaged stand-alone applications (widgets).

Many of these standards are reaching a point of no return, beyond which opportunities for security-by-design will be lost. ENISA is seizing a unique chance to make detailed recommendations for improvements to browser security before the current suite of new standards reaches recommendation status within W3C - at which point they will be non-negotiable for several years to come (HTML 4.0 has been unchanged since 1999).

The report analyses 13 W3C specifications covering HTML 5, Cross-origin communication Interfaces (e.g. Web Messaging and Cross-Origin Resource Sharing), Device Application Programming Interfaces (APIs) (e.g. Geo-location, Web Storage and Media Capture), and Widgets. For each specification, we perform a threat analysis using a consistent methodology which is explained in [4.1]. A general observation of the security analysis is that the security quality of the studied specifications is reasonably high, especially given the number of introduced capabilities and amount of specifications. In total 51 security threats and issues are identified and detailed in this report.

Most of the threats (25) identify security-relevant capabilities in the individual specifications which are not well-defined or insecure. In the analysis, we identify cases of unprotected access to sensitive information, new ways to trigger form submission to adversaries, adversary-controlled cross-domain requests, granularity problems in specifying and enforcing least-privilege policies, potential mismatches with Operating System permission management, and underspecified capabilities, potentially leading to conflicting or error-prone implementations.

We also identify 8 issues and potential threats dealing with isolation properties. This includes new ways to escape origin separation and click-jacking protection, as well as under-specified behaviour in a restricted browsing context (such as a sandboxed i-frame). We also identify permission inconsistencies and under-specifications, as well as issues in dealing with user-involvement. The following is a selection of the most important threats identified:

- **CORS: Interaction with legacy servers:** The CORS specification allows a simple POST request to send arbitrary data in contrast to the key=value format imposed by form elements. This enables an attacker to trigger cross-domain APIs.
- **Web Messaging: Origin delegation:** In case a channel is set up to securely pass messages between two frames, one of the endpoints can easily be passed on to a less-trusted third origin without the original sender noticing (or being able to prevent it).
- **HTML5: Content Handlers:** Content and protocol handlers allow delegation to remote service providers for specific protocols or content types. The use of HTTPS URLs with content/protocol handlers is discouraged, but the specification does not explain why. There is also no specification

of how to inform end-users that a content/protocol handler is registered. Therefore it may remain active indefinitely without the user realizing.

- **HTML5: Disabling Click-jacking Protection:** A common technique to mitigate click-jacking attacks is to check via JavaScript if a window is framed and if so, break out of the frame. However, if an attacker applies the i-frame sandbox option that disables top-level navigation, this technique is prohibited, and the framed window is vulnerable to click-jacking.
- **HTML5: Form Tampering:** Due to the newly-provided capability to introduce buttons outside a form, an attacker can now , using simple HTML injection, more easily lure an end-user in submitting filled-in form values to an attacker-controlled destination.
- **Geo-location: Cache Polling:** The use of the geo-location cache API allows the explicit retrieval of the latest geo-location entry from a cache, according to freshness criteria. This allows an attacker to retrieve a previous location of the end-user, as well as timing information on that particular location.

We make recommendations on:

- **Controlling functionality:** E.g. through enhanced access control policies such as the Content Security Policy specification currently in development.
- **Permission system design:** Permissions systems are currently either underspecified or inconsistent across specifications. Ideally this could be resolved by a separate specification for permissions systems referenced by all related specifications. Failing this, co-ordination between specifications is recommended. We also recommend co-ordination with permissions systems in lower layers (e.g. smartphone OS). This would also help to manage threats which exploit features in multiple specifications such as ever-cookies which can exploit HTML5 features to create extremely persistent 3rd party identifiers.
- **More detailed user interface requirements:** Permissions specifications should require certain features in user interfaces such as information about the document origin, if the permission is for one-shot or monitoring access etc.
- **End-user policing:** The ability of users to make correct decisions on security has its limits. Users are often unable to assess the implications of granting permissions to certain origins. Involving the user too often leads to blindly approving permissions. Specifications should require permission awareness indicators, so the user knows when a site is using a granted permission. Users should also have a way to select predefined security profiles.
- **Restricted contexts:** Only one of the specifications studied takes into account the possibilities offered by restricted contexts such as private browsing or sandboxes. Private browsing should be included in a W3C specification which defines behaviours such as whether permissions should be shared outside of private browsing mode.

We also make recommendations targeted at specific stakeholder groups:

- **Standards working groups and browser vendors:** Here we recommend changes to each of the specification documents to address each threat. We have also produced targeted descriptions of these recommendations and sent them to the relevant working groups.
- **Web application developers:** We draw the web developer's attentions to the implications of certain specification elements such as the possibility to put form buttons outside the form element, the implications of sandboxing for click-jacking protection and the importance of client-side validation of locally stored data.
- **End-users:** We recommend the use of separate browser profiles for different types of task. Everyday browsing should be separated from more sensitive activities such as online banking. Different contexts should be sandboxed, with different settings and permissions.

An important conclusion of this study is that significantly less security issues were found in those specifications which have already received the most attention from academic study and review (e.g. CORS). This demonstrates the value of in-depth security review across all up-coming specifications, as we have performed in this work.

## Table of contents

# 1. Scope

The focus of this report is a security analysis of newly offered capabilities in upcoming web standards and specifications, including a threat enumeration and a set of recommendations.

In particular, this study focuses on the use of web standards in multi-tenant applications (such as web mashups). In such applications, multiple stakeholders provide content and/or functionality with different trust levels and different security characteristics. Due to the fragmentation of ownership in such aggregated applications, there is a clear need for both isolation as well as collaboration between components (either cross-domain/origin, or within the same domain/origin). The report then makes recommendations for more secure isolation and collaboration. It also identifies features of standards which represent risks unlikely to be addressed within specifications because they underpin vital usability considerations. It is still important to raise awareness of such "necessary evils" so that users can make informed choices.

For the analysis, the following system use case scenarios were the primary drivers:

1. Inter-component communication (i.e. inter-page messaging)
2. Cross-application web communication
3. Least-privilege integration of third-party web components
4. Use of newly-proposed Human Computing Interfaces (media capture, geo-location, …)
5. Storing and retrieving client data in local Web Storage

This multi-tenant focus (illustrated by the system use case scenarios) and the identified categories of security issues scope the assessment to the following specifications:

1. HTML 5 specification [1]
2. Cross-origin messaging specification
    a. XML Http Request levels 1 and 2 [2,3]
    b. Uniform Messaging Policy [4]
    c. Cross-Origin Resource Sharing [5]
    d. HTML5 Web Messaging [6]
3. Device API specifications
    a. Media Capture API [7]
    b. System Information API ( systems info and events) [8]
    c. Permissions for Device API Access [9]
    d. Device API Privacy Requirements [10]
    e. Web Storage [11]
    f. Geo-location API Specification [12]
4. Widget specifications
    a. Widget Access Request Policy [13]
    b. Digital Signatures for Widgets [14]

In later follow-ups of this security assessment, it would be interesting to extend this scope to the other promising and emerging specifications and technologies (which were not considered mature enough at the time of writing) such as the Content Security Policy (CSP) [28], the Do Not Track header [31], alternative client-side storage techniques (e.g. Indexed DB [32], File API [33]), and the X-Frame-Options response header [27].

## Terminology

Throughout this document, we consistently use the following terminology:

**Capability**: a piece of functionality, available to a user or a script.

**Origin**: a triple (scheme, host, port)

**Document Origin or Origin:** the origin of a document in the browser

Effective Script Origin: the origin of a script in the browser. This origin can be relaxed with document.domain to a suffix of the current origin.

**Unique Origin:** a globally unique identifier, which is used internally

**Document Address:** the address of the document, typically the URI. The address when a document was opened can differ from the current address (e.g. by adding a fragment identifier or using the history interface)

**Restricted Context**: a context where certain restrictions apply. Examples are a sandboxed context with a unique origin, or a private browsing context as found in all modern browsers

**Private Browsing Context:** a separate browsing mode, in which no history or site information will be stored. Firefox uses the term private browsing mode, Chrome has incognito windows and IE has InPrivate.

# 2. Analysis results

In this section, we briefly summarize the results of the analysis, and discuss some of the more severe threats identified during the security analysis. In addition, we distill several higher-level issues from this overview, which cover numerous capabilities and threats. The recommendations discussed later, are largely based on these higher-level issues.

## 2.1. Overview

An important observation resulting of this study is that the security quality of the studied specifications appears generally high, especially given the number of introduced capabilities and amount of specifications. The main goal of this assessment is to further identify and reduce remaining issues, as well as identifying trade-offs made between security and usability.

By way of disclaimer, we note that with the large amount of newly introduced functionality, which reflects the transformation in the use-cases for the web, it is impossible to analyse all possible security implications (e.g. implications of combinations of new capabilities). The real-world will no doubt throw up many unforeseen threats.

In our security analysis, we identified **51 issues and potential threats**, as summarized, per security question, in the table below.

| | Well defined / Secure | Isolation Properties | Consistency | User Involvement |
|---|---|---|---|---|
| **HTML5** | 8 | 3 | 2 | 2 |
| **Web Messaging** | | 1 | 2 | |
| **XHR 1 + 2** | 1 | | | |
| **CORS** | 2 | 1 | | |
| **UMP** | | | | |
| **Web Storage** | 3 | 1 | 1 | |
| **Geo-location API** | 5 | 1 | 1 | 1 |
| **Media Capture API** | | | 3 | |
| **System Information API** | 3 | 1 | 1 | 2 |
| **Widgets - Digital Signatures** | | | | 2 |
| **Widgets - Access Request Policy** | 3 | | | 1 |
| **Total** | **25** | **8** | **10** | **8** |

Most of the threats (25) were triggered by the first security question, namely if the security-relevant aspects of the newly introduced capabilities are **well-defined and secure**. In the analysis, we identified cases of unprotected access to sensitive information, new ways to trigger form submission to adversaries and adversary-controlled cross-

domain requests, granularity problems in specifying and enforcing least-privilege policies, potential mismatches with OS permission management, and underspecified capabilities, potentially leading to conflicting or error-prone implementations.

Next, we identified 8 issues and potential threats dealing with **isolation properties**. This includes new ways to potentially escape the origin separation and existing click-jacking protection, as well as underspecified behaviour if executed in a restricted browsing context (such as a sandboxed iframe).

Finally, we also identified 10 **permission inconsistencies** and under-specifications across the various specifications dealing with permission management. In addition, 8 issues were raised concerning **user-involvement**. This includes ambiguity about the permission nature, the way user consent is asked, and awareness feedback towards the end-user.

### 2.2.Selection of more severe threats

In this section, we briefly review some of the more severe threats identified in the security analysis. For a more elaborate overview of all the identified issues and potential threats, we refer to [Security analysis].

***CORS-SECURE-1.Legacy Servers*** The CORS specification allows a simple POST request to send arbitrary data as part of the body, in contrast to the key=value format imposed by form elements. This enables an attacker to trigger cross-domain APIs (e.g. REST), or potentially facilitates cross-channel scripting [26].

***WEBMSG-ISOLATION-2.Port Endpoint Origin*** In case a channel is set up to securely pass messages between two frames, one of the endpoints can easily be passed on to a less-trusted third origin without the sender noticing. This could happen in a benign scenario where the receiver would like to delegate the message processing, but this could as well be the result of an XSS attack. In contrast to a direct invocation of `postMessage` on a window, a sender cannot explicitly scope the origin attribute of the destination when sending over a channel, and as a result has no control over the receiving party.

***HTML5BROWSER-SECURE-1/2/3.Content/Protocol Handlers*** The content and protocol handlers allow delegation towards service providers for specific protocols or content types. This could, for instance enable a webmail service to register itself to handle all mailto:// URLs, or a document processor to edit word documents, based on consent by the end-user. In case a handler is registered, the URL is not processed at the client-side, but the URL is passed to the content/protocol handler, and is fetched as part of this service.

Although these are valid scenarios in the evolving service landscape and emerging Web Operating Systems, the impact of registering a malicious content/protocol handler is huge. At this moment, the specification only list potential security issues with the use of handlers, but leaves it up to the browser vendors and end-users to control the leakage of confidential data, or the registration for default protocols such as HTTP or HTTPS. The specification even discourages the use of HTTPS URLs if used in combination with content/protocol handlers, but does not provide argumentation why insecure communications should be used. In addition, there is no specification of how to indicate to end-users that a content/protocol handler is registered and in use (user awareness). Because of this, it is quite probable that a content handler may be approved by an end-user, and remain active indefinitely without the user remembering or realizing.

***HTML5ATTR-SECURE-1.Form Tampering*** Due to the newly-provided capability to introduce buttons outside a form, and override form properties (such as the action or method), an attacker can now , using simple HTML injection, more easily lure an end-user in submitting

filled-in form values (either filled in by the end-user, or prefilled by the browser or server) to an attacker-controlled destination. This means that existing input validation approaches that focus on script injection can easily be circumvented by injecting HTML, and validation techniques should be adapted appropriately.

***GEOLOC-SECURE-3.Cache Polling*** The use of the geo-location cache API allows the explicit retrieval of the latest geo-location entry from a cache, according to the freshness criteria specified in the query. While this feature reduces the power-intensive geo-location queries, it also allows an attacker to retrieve a previous location of the end-user, as well as timing information on that particular location. The specification does not provide upper limits on the lifetime of geo-location cache entries.

***HTML5ATTR-ISOLATION-1.Disabling Click-jacking Protection*** A common technique to mitigate click-jacking attacks is to check via JavaScript if a window is framed, and if so, navigate the top-level window to the window being framed [17]. However, if an attacker applies the i-frame sandbox option that disables top-level navigation, the abovementioned mitigation technique is prohibited, and the framed window is vulnerable to click-jacking. Alternative techniques proposed in [17] to use the X-FRAME-OPTIONS [27], to use the Content Security Policy (CSP) [28], or to render a window by default useless via JavaScript in case it is framed, seem not to be affected by the sandbox attribute, but are not widespread at this moment.

### 2.3.Extensive Functionality Growth

In the past few years, the available functionality on the client-side has seen an extensive growth with the introduction of new APIs. The permission to use this extended functionality is typically granted to a certain origin and stored persistently, until revoked by the user. This situation can make sites that have acquired such privileges highly interesting targets for attackers. The related APIs are:

- HTML5 Browser Features
- Web Storage
- Geo-location API
- System Information API

### 2.4.Dealing with Multiple Browsing Contexts

A lot of the functionality defined in the specifications is available in multiple browsing contexts, including restricted contexts such as a sandbox or a private browsing context. Unfortunately, the specifications are not always clear on the exact behaviour of this functionality in such a restricted context. Some example problems are

- Are permissions stored in a normal browsing context also valid in a restricted context or vice versa?
- Can data be stored under one browsing context and retrieved under the other?

The related threats are (see [Security analysis]):

- HTML5CACHE-ISOLATION-1.Restricted Context
- STORAGE-ISOLATION-2.Restricted Context
- GEOLOC-ISOLATION-1.Restricted Context
- SYSINFO-ISOLATION-1.Restricted Context

### 2.5.Additional Permission Systems

Several specifications introduce potentially sensitive functionality; hence they also introduce or mention a permission system. Unfortunately, the permission systems are very inconsistent between specifications, in multiple ways such as:

- Mention of potential policy violations, but no mention of how this system might/should work
- Different bases for permission (e.g. for framed documents: document origin vs document origin + top-level origin)
- Lack of requirements for the permission UI, both for granting and for managing permissions

The table below offers an overview of all new permissions that are introduced by the analyzed APIs. Next to these APIs, there are undoubtedly other newly introduced APIs that also require some form of permission.

The columns of the table mean the following:

- Consent: is permission consent required for this functionality? Required either means that the specification explicitly requires them or that an existing implementation of the spec requires permissions.
- Multiple Types: are there different types of operations that can require permissions (e.g. one-shot access versus continuous monitoring)
- Based On: on which basis is the permission set (e.g. Origin, Host, ...)
- Storable: can the permission be granted for multiple uses (i.e. more than one time)
- Manageable: can the permissions or other data be managed (e.g. revoke granted permissions, remove stored data, ...)

| | Consent | Multiple Types | Based On | Storable | Manageable |
|---|---|---|---|---|---|
| **Offline Applications** | yes | no | unknown | unknown | yes |
| **Scheme/content handlers** | yes | no | Scheme / Content Type | yes | yes |
| **Web Storage** | no | - | Origin | - | yes |
| **Geo-location API** | yes | yes | Origin | yes | yes |
| **Media Capture API** | yes | no | unknown | no | no |
| **System Information API** | yes | yes | Origin | yes | yes |

The related threats are:

- HTML5CACHE-CONSISTENCY-1.Permission System
- HTML5BROWSER-CONSISTENCY-1.Handler Permissions

- STORAGE-CONSISTENCY-1.Permission System
- GEOLOC-CONSISTENCY-1.Permission Management
- GEOLOC-USER-1.Permission Nature
- MEDIACAPTURE-CONSISTENCY-1.Requesting Consent
- MEDIACAPTURE-CONSISTENCY-2.Consent UI Content
- MEDIACAPTURE-CONSISTENCY-3.Consent UI Security
- SYSINFO-CONSISTENCY-1.Permission Management
- SYSINFO-USER-1.Permission Nature

### 2.6.Conflicting Security Controls

The devices on which web applications run are very diverse, ranging from classic desktop systems to smartphones or embedded devices, such as gaming consoles or television sets. Each of these devices runs an operating system, which may already contain security controls for specific operations, such as determining the location of the device. Stacking several security controls on top of each other may be problematic and can confuse the user. Additionally, the security controls defined in the specification are typically more fine-grained than the underlying security controls.

The related threats are:

- GEOLOC-SECURE-5.OS Conflicts
- SYSINFO-SECURE-3.OS Conflicts
- WARP-SECURE-3.OS Conflicts

# 3. Recommendations

In this section, we present concrete recommendations to improve the security of the analyzed specifications. The recommendations are based on the threats identified in the security analysis, as well as the higher-level problems identified in the conclusions of the analysis.

## 3.1. Controlling Functionality

One conclusion of the security analysis is that a web application potentially has a large set of privileges on the client-side. This can be problematic with the presence of well-known attacks, such as script injection. These attacks are already discussed by the HTML5 Security Cheat Sheet, but the newly added functionality in HTML5 further increases this attack vector, e.g. with the **`autofocus`** or **`formaction`** attribute. In this recommendation section, we investigate restrictive countermeasures.

Current restrictive systems (i.e. the **`sandbox`** attribute *[HTML5ATTR-SECURE-2]* or the Widget Access Request Policy *[WARP-SECURE-1.Coarse-Grained Approach]*) are very coarse-grained and only offer the option of enabling or disabling a certain functionality. It is currently impossible to integrate third-party functionality into a web application (e.g. in a mashup scenario) while applying the least-privilege security principle. In such a scenario, either access to all security-sensitive APIs is given while relying on the same-origin policy and user-consent mechanisms, or scripts are completely disabled.

One approach is the use of additional fine-grained policies. Such policies are typically enforced at the client-side, but are often provided by the server. There are several types of policies available, either in the form of additional response headers page (e.g. Content Security Policy [27] and X-Frame-Options [26]) or embedded in the page. The latter can be achieved by using an inline reference monitor technique (e.g. ConScript[19], Safe JavaScript Wrappers[20,21], WebJail[22]) or by ensuring the safety of the language, based on the capability security model (e.g. Caja [23]).

Of these fine grained policies, the recently proposed *Content Security Policy* (CSP) is on its way to standardization and already offers extended control over scripts and embedded objects. Rigorously applying CSP certainly addresses several of the mentioned threats, although others remain still valid. Therefore, extending CSP to deal with script capabilities such as domain relaxation (i.e. by using **`document.domain`**) or restricting the power of form attributes is recommended. However, a key challenge in finalizing CSP and deploying it on a wide scale will be the level of granularity.

Another approach to limit the available functionality to scripts or documents is to define subset(s) of available functionality. This can be achieved in various ways, for instance using some form of strict mode, as present in some languages, such as ECMAScript 5. Another way is to distinguish sensitive APIs (e.g. geo-location) from general functionality, and restrict access on this basis.

In addition, a thorough or formal analysis of the available capabilities and potential restrictions can offer a high level of confidentiality that external content can be integrated with a minimal level of privileges, and that there are no undetected attack vectors present.

## 3.2.Consistent Permission Systems

The security analysis also shows that permission systems are very inconsistent or incomplete. The specifications suffer from a number of problems, which will be addressed below.

### Permission System Design

Several specifications include a permission system, without any consistency between these systems. This is troublesome, since some systems are fairly weak compared to other systems. One good example is the Geo-location API versus the System Information API, where the latter is much more extended (e.g. it requires permission for a pair of origins for a framed document and distinguishes between one-shot vs. monitoring permissions).

One recommendation is for W3C to co-ordinate these permission systems across specifications. This would enable a strong permissions system for all involved APIs and makes it less complicated for browser vendors.

An alternative recommendation is to create a separate permission specification, which describes how permission systems should work across all the specifications. The specification should describe the necessary types of permissions (e.g. persistent vs one-time, one-shot permission vs a watching process, ...) and what they are based on. The specifications describing the APIs can then simply reference the permission specification with a certain type of permission, and require the implementation of that permission system. This brings consistency across all permission systems and decouples improvements to permission systems from functionality developments. Recent work on *Feature Permissions* [29] already points in this direction, but does not yet cover all aspects raised in this analysis. This would also help to manage threats which exploit features in multiple specifications such as "evercookies" which may use HTML 5 local storage, session storage, canvas data etc… to create extremely persistent 3rd party identifiers.

A third recommendation is to investigate potential conflicts of the permission systems with underlying security controls. Integration of API permissions systems and underlying security controls not only avoids potential conflicts between systems, but can also be useful in delivering an integrated and consistent experience to the end user.

### User Interfaces

The specifications do not include much detail about the user interfaces for giving permissions and managing permissions. Typically, they require that the origin of the document is displayed. One major missing item is the nature of the permission (one-shot vs watching process) as well as the actual document requesting it (e.g. is it sandboxed or framed? is it visible?). For managing permissions, the specifications typically state that it should be clear and easily usable. This is very vague and is not even a strong requirement (should instead of must).

One recommendation is to extend the specifications to include more UI requirements. At least all the aspects of the requested permission should be present. For managing the permissions, the specifications should describe how it should happen (not what it should look like). For example, they could state that "A user must be able to get an overview of all assigned permissions, grouped per origin".

A second recommendation is to include these UI requirements into the abovementioned separate permission specification, which is then referenced from all other specifications

requiring a permission system. This ensures consistency and facilitates improvements or extensions of the permission system.

### 3.3.End User Policing

Several specifications depend on the end user to ensure security, typically by means of a permission system. With the rapid development of new APIs offering new functionality, this approach is reaching its limits. Typical web users are unable to assess the ramifications of granting certain permissions to certain origins. Additionally, involving the user too often might quickly lead to blindly approving permissions. The threats directly related to this recommendation are:

- HTML5EL-USER-1.Overriding Sandbox
- SYSINFO-USER-2.Awareness
- WGDIGSIG-USER-1.Certificate Management
- WGDIGSIG-USER-2.Unsigned/Unchecked Packages
- WARP-USER-1.User's Role

A first recommendation is to require an awareness indicator, so the user knows when a site is using a granted permission (e.g. when a site is locating the user). Currently, this is only included as a suggestion in the Geo-location API and is missing from other APIs, such as the System Information API.

A second recommendation is to offer the users a way to select predefined security profiles or to create a security profile (e.g. by means of a wizard, which could explain the ramifications of sharing your location and then provide the option to allow this or not.). Once a security profile is chosen, it is used to determine the appropriate actions when a site requests permission for an action.

### 3.4.Restricted Contexts

In the analysis, we identified two types of restricted context: a sandbox environment and a private browsing context. None of the specifications, except for HTML5, takes these restricted contexts explicitly into account, leading to several problems with under-specification, as concluded in the analysis.

Our recommendation is to include the private browsing context, a feature available in all major browsers, explicitly into a W3C specification. This enables the possibility of defining context-specific behavior, such as whether permissions should be shared across contexts, and whether isolation works in both directions between a private browsing context and a normal operation context. In this context, it can be interesting to study the requirements of other anonymity and privacy systems, such as the TOR anonymity button [30] or the Do-Not-Track initiative [31].

### 3.5.General Recommendations

Apart from the selected topics above, a lot of smaller threats can be addressed with a wide range of recommendations. This section makes some general recommendations towards a specific target audience. At the end of each recommendation, the relevant threats from the security analysis are referenced.

### To the standardization bodies and browser vendors

A few of the identified threats have a considerable impact on security or privacy. We recommend that these issues are studied and addressed, in order to obtain a secure specification. If these issues cannot be fixed in a reasonable timeframe, warnings should be added to the specification, clearly indicating the security or privacy risks. NB these recommendations will be sent separately to the respective working groups. This applies to the following threats:

- WEBMSG-ISOLATION-2.Port Endpoint Origin
- GEOLOC-SECURE-3.Cache Polling
- HTML5EL-SECURE-1.Media Usage
- HTML5BROWSER-SECURE-1.Content/Protocol Handlers
- HTML5BROWSER-SECURE-2.Sleeping Content/Protocol Handlers

Future versions of the specifications should address issues that are not easy to solve, but improve security/privacy in the long run:

- HTML5BROWSER-USER-1.Drag and Drop Target
- GEOLOC-SECURE-2.Technology Agnostic
- GEOLOC-SECURE-4.Changed Behavior
- SYSINFO-SECURE-2.Changed Behavior

The specifications often leave some functionality or details unexplained. It is recommended to address these items, in order to obtain a correct and complete specification. This applies to the following threats:

- HTML5EL-SECURE-2.Menu Integration
- HTML5EL-SECURE-3.Keygen Scenarios
- HTML5ATTR-ISOLATION-2.Srcdoc URI
- HTML5BROWSER-SECURE-3.Content Handler URLs
- HTML5BROWSER-USER-1.Drag and Drop Target,
- HTML5BROWSER-SECURE-4.Infinite History
- WEBMSG-SECURE-1.Origin Determination
- XHR-SECURE-1.Inconsistent Method Checks
- CORS-SECURE-2.Unnecessary Processing
- CORS-ISOLATION-1.Unique Origins
- STORAGE-SECURE-1.Subdomain Storage
- STORAGE-SECURE-3.Shared Frame Storage
- GEOLOC-SECURE-1.Monitoring Lifetime
- SYSINFO-SECURE-1.Monitoring Lifetime

Finally, it is remarkable that for some specifications, considerably less threats were found. Examples are the HTML5 navigation policy and the cross-origin communication protocols (CORS and UMP). One reason for this higher quality is the academic attention that has been given to these specifications [24], including a formal analysis [25]. Next to these research papers, additional documentation (e.g. the flowchart of XHR constructed in this security analysis) can help browser vendors to have a better overview of the specification and thus lead to a cleaner and more correct implementation.

### To Web Application Developers

A web developer designing and implementing exciting new applications using the latest technology has to rely on the specifications. Some of the specifications contain subtle details or potential consequences which are not documented, and thus can easily escape a developer's attention. This applies to the following threats:

- HTML5ATTR-SECURE-1.Form Tampering
- HTML5ATTR-ISOLATION-1.Disabling Click-jacking Protection
- WEBMSG-ISOLATION-1.Port Handler's Origin
- CORS-SECURE-1.Legacy Servers
- STORAGE-SECURE-2.Client-side Validation

We recommend amending the specifications with notes towards developers, to warn them of the potential consequences when using one of these specifications.

### To End Users

In order to guarantee the best privacy protection, the use of separate browser profiles for different tasks is recommended. Everyday browsing should be separated from highly sensitive browsing (e.g. online banking), as well as highly risky browsing (e.g. searching cracks or warez). This can be achieved by explicitly creating multiple browser profiles and runing these profiles concurrently as different instances of the browser (e.g. as can be done in Firefox), by using multiple browsers concurrently (e.g. Opera and Chrome next to each other), or probably more convenient (and not available yet) by extending the concept of private browsing mode already available in mainstream browsers to support multiple well-isolated, concurrent contexts within one browser.

The use of private browsing mode as simple mitigation technique turns out to be inadequate in some of the cases. This applies to the following threats:

- HTML5EL-USER-1.Overriding Sandbox
- HTML5ATTR-SECURE-1.Form Tampering
- HTML5ATTR-ISOLATION-1.Disabling Click-jacking Protection
- HTML5CACHE-ISOLATION-1.Restricted Context
- HTML5BROWSER-SECURE-1.Content/Protocol Handlers
- HTML5BROWSER-USER-1.Drag and Drop Target
- STORAGE-ISOLATION-2.Restricted Context
- GEOLOC-SECURE-3.Cache Polling
- GEOLOC-ISOLATION-1.Restricted Context
- SYSINFO-ISOLATION-1.Restricted Context

# 4. Security analysis

This section shows the results of the security analysis, performed as described in section 2. For each of the specifications in scope, we provide a brief summary of the relevant functionality, distill capabilities, the involvement on the user and the assumptions made. The enumeration of threats is guided by the security questions defined earlier, and is categorized as such.

One general threat is the well-known injection attack vector. Examples are cross-site Scripting (XSS) or HTML injection. Such threats are considered in scope during the security analysis, but we only include them in the list of threats if they have not yet been documented in the existing "HTML5 security cheat sheet" [15]. This threat class is also revisited in the recommendations section.

For the security analysis, we focused on the newly added features in the specifications. In particular, for all of specifications except HTML5 we consider the entire specification to be in scope (since they are new). For the HTML5 specification (which covers numerous topics, such as structuring documents, navigation policies, offline applications), the difference between HTML4 and HTML5 [16] was used a de facto basis for the analysis. Guided by this list of differences containing all new or changed elements and attributes, we conducted a first selection of relevant sections of the HTML5 document. In a second screening of the entire specification, we selected several additional sections documenting new browser features or policies. Sections about rendering and document structure are not directly included in the analysis, unless security-relevant. The selected HTML5 sections have been divided into 5 topics:

1. **Elements**: analysis of elements with relevant changed or new functionality in HTML5
2. **Attributes**: analysis of attributes with relevant changed or new functionality in HTML5
3. **Browser Features**: analysis of relevant behavior part of HTML5, not specific to an element or attribute
4. **Navigation**: analysis of the navigation policy as specified in HTML5
5. **Application Cache**: analysis of the application cache, supporting offline applications

The HTML5 topics are discussed in sections 4.2 to 4.6, the rest of the HTML5 APIs in sections 4.7 to 4.17. Section 2 provides a summary of the security analysis.

## 4.1. Approach

### Technical Approach

The technical approach used to assess the security of browser standards consists of 7 different steps. Each of the steps is discussed in more detail in the following subsections. The 7 steps are executed sequentially, although a few small iterations of step 1 and 2 are needed to converge to a small but sufficient model of next-generation browser standards.

**Step 1: Definition of next generation browser standards**

In the first step, an abstract model of next generation browser standards is built to form the basis for the asset-centric threat analysis. In this model, we consider the use of web technology in multi-tenant applications in which multiple stakeholders provide content and/or functionality with different trust levels and different security characteristics. Such applications have typical isolation requirements to guarantee the confidentiality and

integrity of application-specific data as well as end-user data, and to limit the use of security-sensitive browser APIs. In addition, the power of multi-tenant mashups lies in the lightweight composition of components and their inter-component collaboration.

With this focus in mind, we identify the relevant building blocks from the specifications listed in the previous section, and build an abstract model of browser standards, suited for the threat analysis. In this model, we include the five identified system use case scenarios (as discussed in the previous section). The hypertext application model, developed in this step, incorporates the various HTML5 APIs and includes an informal description of the provided functionality towards next-generation browser standards.

**Step 2: Asset identification**

In the second step, we identify the different assets within the hypertext model, including end-user assets, as well as server-side application assets. These high-level technical assets are largely independent of specific usage scenarios.

**Step 3: Attacker model**

Within the analysis, the following threat model is applied. We consider an honest user employing a standard web browser viewing content from an honest web site. The web browser can either support the newly offered (and studied) web standards, or adhere to the state-of-practice of current mainstream browsers. In this model, we assume the following attackers to be in scope[1] :

**Web attacker**: The web attacker is a malicious principal who owns one or more machines on the network, and hosts malicious websites. We assume that the browser fetches and renders content from the attacker's web site. In doing so, malicious websites (i.e. under control of attacker) are visited during a browsing session.

**Gadget attacker**: The gadget attacker is a web attacker with one additional ability: the page visited by the user embeds a gadget of the attacker's choice in the multi-tenant application (such as a 3rd party javascript library, an advertisement frame or a mashup component). This allows the analysis to focus on isolation from as well as collaboration with untrusted third party gadgets in multi-tenant applications.

We assume the network attacker to be out-of-scope for this analysis. In addition, we also do not cover identification and mitigation of classic (i.e. not introduced by the new standards) web application security vulnerabilities and attacks (such as XSS, SQL injection, code injection, session management). These vulnerabilities and their possible countermeasures are well-known and well-documented (e.g. in OWASP Top10 and SANS Top 25 reports) and are not specific to the newly proposed standards and specifications. However, within the security assessment, we do analyze to what extent the newly proposed functionality increases the potential impact of classical injection vulnerabilities.

[1] This identification of attacker types (web attacker and gadget attacker) is based on the threat model used in: Adam Barth, Collin Jackson, and John C. Mitchell. 2008. *Securing frame communication in browsers*. In Proceedings of the 17th conference on Security symposium (SS'08). USENIX Association, Berkeley, CA, USA, p.17-30.

**Step 4: Threat enumeration**

In the fourth step, we perform an asset-centric threat analysis on the hypertext application model, developed in step 1. This threat analysis concentrates on the impact of the proposed attacker models on the five system use case scenarios (as described in section 2.1).

The outcome of this threat analysis is an enumeration of the different identified threats on the browser security model. This threat enumeration is used in the next steps to select and evaluate in more technical details the most relevant subsets of this abstract model.

**Step 5: Selection of most relevant threats and technologies**

In the fifth step, we use the results of the threat enumeration of step 4 to prioritize the identified threats, and select an appropriate subset of threats and technologies for further detailed analysis and documentation. To do so, different subsets of the abstract model are put forward with a selection of important threats and potential room for improvement.

The main criteria for identifying and selecting these subsets are:

1. Importance of the identified threats – as defined by their likelihood and impact
2. Feasibility and pragmatism of the potential security improvements
3. Impact of the potential improvements and recommendations

Step 5 is an important synchronization point within the assessment. The most promising subset of the abstract model is selected in this step, and further refined in the next steps.

**Step 6: Detailed analysis and threat documentation of selected subset**

In this step, the selected subset of the abstract model is further refined to a concrete model taking into account all the relevant specifics of the specifications in scope. Each of the identified threats within this subset is assessed in more detail within this concrete model.

This results in an assessment of the security benefits of the newly proposed standards and specifications, as well as remaining or newly introduced security risks within the chosen concrete model.

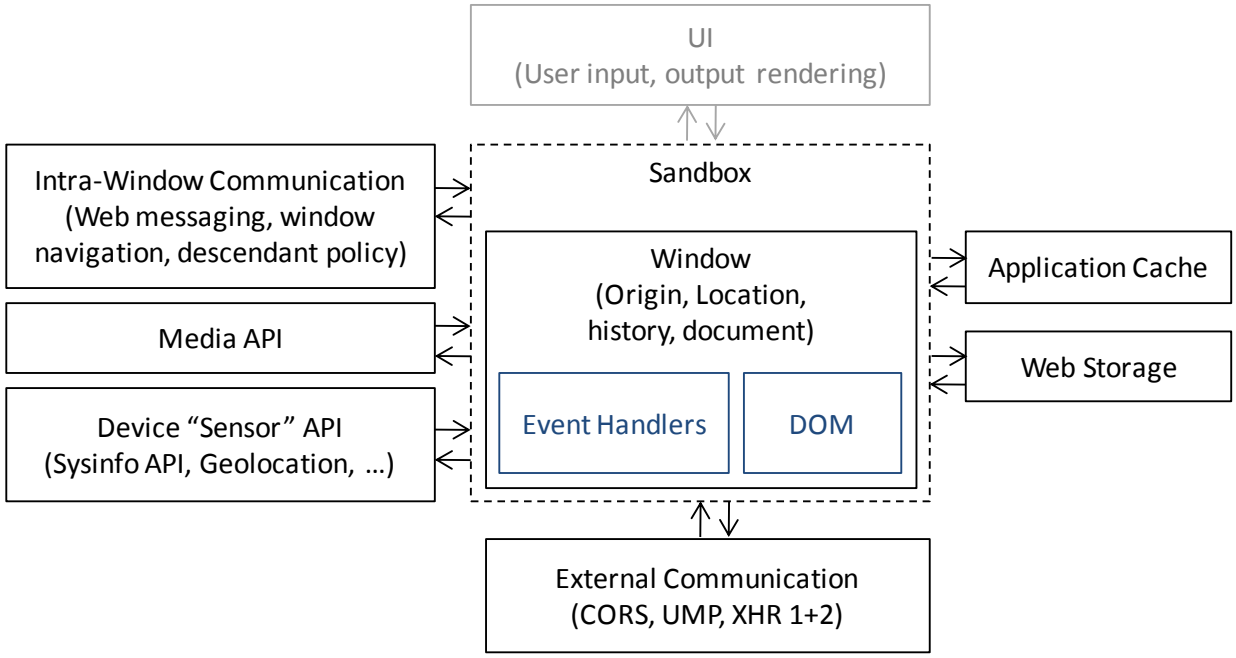
**Step 7: Recommendations**

In this step, we enumerate and discuss realistic and pragmatic recommendations for the threats detailed in step 6. This ranges from advice on amendments to the standards, advice to browser developers (and browser extension developers), or website developers implementing and applying the standards. To prioritize the recommendations in scope, we take into account the countermeasure complexity, the threat severity, the feasibility for broad acceptance.

## The next-generation hypertext model

### Model

Based on a first screening of all the specifications in scope, we use the following abstract model of next-generation browser standards. The model is shown in the figure below, and




shows how the different specifications in scope can be grouped together.

The centre-piece of the model is the browser concept of a window containing a document. Visually, such a window occurs as a single browser window, a tab, a popup or a frame. This window is represented by a window object ([1], section 5 5.2). Through the window object, web pages and scripts gain access to internal properties (the URL, navigation history, ...), event handlers, the document and its associated DOM tree and numerous client-side APIs.

The functionality offered by the analyzed specifications is grouped into blocks based on available functionality. We discuss the window and sandbox in more detail and briefly introduce the other functional blocks below.

#### *Window and Sandbox*

As mentioned before, the browser window and its associated window object enclose a document with a specific origin and location (a URL). A window can contain multiple documents (i.e. a browsing history) but only one of these documents can be active at any given time. Since the relation between window and document at one moment in time is one-to-one, we do not separate a window and a document when this is not relevant.

New functionality introduced in the HTML5 specification allows the sandboxing of an iframe. This sandbox imposes restrictions on all the content in the iframe, as shown by the

dotted line in the model. The specific features and consequences of the sandbox will be part of the security analysis.

The two functional blocks inside the window (Event Handlers and DOM) represent two cornerstone pieces of functionality for dynamic web pages. Event handlers are used extensively to register handlers for a specific event, such as receiving messages from other windows or being notified of mouse clicks. Access to the DOM enables a script to read or modify the document's structure on the fly. Within the security analysis, we will only explicitly assess the newly-added security-relevant handlers.

### *Inter-window Communication*

An important aspect of composed applications is communication between several windows (e.g. sending messages to embedded iframes). This functional block covers window navigation and the corresponding policy, the descendant policy (HTML5). Additionally, this block includes message passing as defined by the Web Messaging specification.

### *Device "Sensor" API*

Several specifications introduce new APIs to retrieve client-side information. In the scope of this analysis, we investigate the Geo-location API, which allows the retrieval of the physical location of the device, and the System Information API, which offers access to device properties, such as battery level, CPU information, ambient sensors (light, sound, movement ...).

### *Media CaptureAPI*

Capturing audio and video fragments through a microphone or webcam, often integrated in mobile devices, becomes possible with the Media API. This API offers scripted access to these (highly sensitive) devices.

### *External Communication*

An interactive client-side web page often requires communication with remote parties, for example to load contacts from an address book. This block covers the specification of XMLHttpRequest, the communication mechanism, and new policies to enable secure cross-origin communication (CORS and UMP).

### *Client-side Storage*

Web Storage, one of the specifications covered introduces client-side storage, which applications can use to temporarily or persistently store data. One example is a calendar application which stores a copy of your calendar at the client-side.

### *Application Cache*

With the newly introduced Application Cache (HTML5), applications can be downloaded and run offline. This is particularly useful for mobile devices, such as notebooks or smartphones.

### *UI*

Naturally, the loaded documents need to be rendered to the user. Additionally, users interact with the browser and the contained pages in numerous ways (mouse movement,

entering text with the keyboard, entering new URLs, using the back and forward button, ...). For the security analysis, only newly-added security-relevant interactions are taken into account, such as interactions that impact the state of the window or page (e.g. submitting a form or clicking a link).

### *Others (Not In Model)*

The security analysis also includes specifications about widget security. These are not present in the abstract model, because they are on a higher level of abstraction than the model. A widget is an instantiation of a web application, which uses windows and documents as presented in the abstract model. The widget-specifications we include in the security analysis are Digital Signatures for Widgets, which discusses how a widget can be signed by authors and distributors (and the signature verified), and Widget Access Request Policy, specifying how rights to access network resources (URIs) can be assigned to a widget.

## Assets and Methodology

Within the hypertext model, we can identify a set of technical assets relevant for this security analysis:

**A1** Application-specific data and functionality (page contents, DOM elements, JavaScript code, ... )

**A2** End-user data and identity (credentials, cookies, local storage, user/media input, location, ...)

**A3** Contextual data (device/sensor information, location, referer, document or effective script origin, ...)

Examples of potential security issues involving these assets are injection of unauthorized code, making requests in the user's name or manipulating cookies, and a breach of privacy through local storage, sensor information, location information ...

The specifications in the scope of this analysis are very extensive. Therefore, we conduct a security analysis focused on the new attack surface only (new functionality and features with respect to previous versions of the standards) and its effect on the user (usability/burden). The analysis is based on the assets mentioned above, guided by four security questions:

**SQ1 - Well defined / Secure**: Are the security-relevant aspects of the newly introduced capabilities well-defined and secure? (e.g. privacy problems, unprotected security-sensitive features, ...)

**SQ2 - Isolation Properties**: Do the new specifications violate isolation properties between origins or restricted contexts? (e.g. sandboxes or private browsing mode)

**SQ3 - Consistency**: Is the new specification consistent with other specifications? (e.g. permission management, ways to access information, ...)

**SQ4 - User Involvement**: How do the security measures of the specification rely on the user making correct security decisions? (e.g. which decisions does the user have to make)

An analysis based on these questions will identify poorly-protected functionality (A1) or data (A2/A3), either on a technical level or on the end-user level. Additionally, any

problems with unintended access to data (A2/A3) across origins or contexts will be discovered. The analysis is conducted in three steps:

1. A study of the specification leads to a brief summary of relevant functionality (description). We distill the presence of the following properties in the specification:
    - **Capabilities:** which functional capabilities are offered by the specification (e.g. retrieving the user's location)
    - **User Involvement:** in which way does the specification involve the user (e.g. approve location retrieval once per site)
    - **Security/Privacy assumptions:** which security/privacy considerations and assumptions are explicitly listed in the specification (e.g. location can only be retrieved with explicit permissions) and which are implicitly present in the specification and need to be derived (e.g. the user can not be tricked into giving permission to access the location)? NB the validity of these assumptions is analysed as part of the threat analysis – see point 2.
2. Based on the capabilities, level of user involvement and both explicit and implicit security/privacy assumptions, we analyze the specifications for potential threats, issues and underspecified behavior (e.g. the requirements for browser behavior when asking permissions) We also assess the validity of the assumptions made by the specification as part of the threat analysis. This step of the analysis is guided by the previously defined security questions.
3. Based on the detailed analysis of all specifications, we investigate potential issues with the combination of multiple specifications (e.g. the behavior of permissions in a restricted context). Identified issues will be listed under the most relevant specification.

After the security analysis, the identified issues and threats are addressed in the recommendations section. For different categories of issues and threats, we will discuss the importance of this kind of problems and formulate appropriate recommendations.

### 4.2.HTML5 - Elements

#### Description

***media element (audio, video)***: A media element can be used to embed a video or audio object in the web page. Pages can script their own controls for the stream, but if they do not offer controls (e.g. by explicit indication or disabled scripting), the user agent should expose a user interface to control the stream. A media element allows multiple alternative streams to be specified with the source element. Text tracks can be added using the track element (for example subtitles, captions, chapters, descriptions). The programmatic interface of media elements offers origin-independent access to the controls and the stream's properties, such as playback information (e.g. the timeranges of the video that have actually been rendered).

***canvas***: A canvas can be used to draw 2D or 3D images. Pre-existing (e.g. JPG, PNG) images can be drawn on a canvas, as well as frames from a video element. Additionally, a drawing context can offer a rich set of JavaScript controls to draw images. The image data

of a canvas can be requested (`getImageData`) or it can be exported to a data URL, which represents a PNG image (`toDataURL`).

***embed***: The embed element supports the integration of external (typically non-HTML) content. The src attribute defines the location of the external content. Parameters can be passed by specifying them as attributes of the embed element. Embedded content is disabled if the content is sandboxed, although the user agent can allow the user to explicitly enable the disabled items.

***menu***: Using the menu element, a web application can construct menus. Both context menus (e.g. when right clicking) as toolbar menus (e.g. always accessible top bar) are available. A sample visual representation is provided, but no concrete implementation criteria are given.

***command***: A command element represents a user-invocable command. It can be part of a menu/toolbar or it can define a keyboard shortcut. Commands can be a checkbox, radio button or a generic command. The actions are implemented using onclick handlers.

***keygen***: A form control that can be used to generate a public/private keypair. The private part is stored in the local keystore, the public part is submitted to the server. The specification does not mention any way to store, access, or use the private key value, but suggests that the server creates a client certificate from the public key, which can then be installed in the browser.

### Capabilities

The specification enables the following capabilities:

- Embed audio/video objects in a page, potentially overlaying other items
- Control audio/video objects through the controls or via a script
- Access properties of audio/video objects, including playback information
- Draw custom graphics using a canvas
- Embed non-HTML content into a document
- Add menus and menu items to a web application
- Create commands within a page, invocable by the user
- Generate a private/public key, submit the public part and store the private part

### User Involvement

The specification involves the user in the following way:

- Control embedded media elements
- Explicitly enable plugin content in a sandboxed environment
- Interact to activate a menu item
- Interact to execute a command
- Interact to generate a keypair

### Security/Privacy Considerations

The specification explicitly mentions the following security/privacy considerations:

- Embedded Media Elements: to avoid security issues with embedded media elements, the specification imposes the following two measures:
    - Embedded content must be treated as if it were in its own unrelated top-level browsing context (e.g. no access to embedding page)
    - Malicious pages embedding cross-origin content have restricted access to its properties (e.g. no access to text tracks)
- Canvas Security: information leakage through a canvas element can occur if content from multiple origins is displayed on the canvas. A canvas can therefore only be exported (with the `getImageData` or `toDataURL` operation) if it is *origin-clean*. A canvas' origin-clean flag becomes false if:
    - An image from an image of video element with a different origin than the canvas' document's origin is used (example use: `drawImage`, `fillStyle`, ...)
    - A font whose origin is different than the canvas' document's origin is considered (considering the font suffices, it does not actually have to be used)
- Plugins: in a sandboxed environment, plugins are disabled by default because they might not honor the imposed restrictions. User agents should warn the user of the consequences if an enable option is provided.

The following considerations are assumed by the specification and are derived from its contents:

- By isolating embedded media in its own unrelated top level browsing context, all potentially malicious behavior is mitigated
- Properties of media elements contain no private information, since they are accessible from other origins than the origin of the media element
- The output of a canvas (PNG data) produces trusted data (e.g. no vulnerabilities or exploits)
- The input of a canvas is validated properly
- The application's menu items are separated from the user agent's menu items
- The private key is stored securely
- New HTML5 elements (such as the canvas and the media elements) might provide an additional and interesting way to trick users into click-jacking, and should be taken into account as well

## Threat Enumeration

### SQ1 - Well defined / Secure

***HTML5EL-SECURE-1.Media Usage*** A media element contains a played attribute, which offers access to the time ranges that have been rendered (thus listened to or watched) by the user. Access to this attribute is not protected, so a page embedding a video from another origin can still access this information

***HTML5EL-SECURE-2.Menu Integration*** A web application can define contextual and toolbar menus. The specification does not mention many implementation details. A user agent may implement integrate these menus with its own user interface, especially on small

displays such as smartphones. This may confuse a user and may present malicious or erroneous menu items

***HTML5EL-SECURE-3.Keygen Scenarios*** The specification does not provide details on how the keygen element can be used in practice. No concrete usage scenarios or implementation requirements are provided.

**SQ4 - User Involvement**

***HTML5EL-USER-1.Overriding Sandbox*** Sandboxed content is not allowed to load plugin content. The specification of the embed element however states that a user agent may allow the user to override this for a specific content item, but the user agent should warn the user that this could be dangerous. The override option is only briefly mentioned as part of the description of the embed element, instead of being incorporated in the entire spec (e.g. with the sandbox attribute)

### 4.3.HTML5 - Attributes

#### Description

***autofocus (form controls)***: Indicates that a control is to be focused as soon as the page is loaded. A document must only specify this attribute for one element. Receiving focus is not possible if the current focus lies in a cross-origin document or if the current document has disabled automatic features in a sandbox

***form (form controls)***: Specifies the form this control belongs to (by default, this is the nearest ancestor form). This allows the use of nested form controls or arbitrary placement of form controls.

***form... (submit buttons)***: A submit button can use **`formaction`**, **`formenctype`**, **`formmethod`**, **`formnovalidate`** and **`formtarget`** to override the attributes defined by the form itself.

***async / defer (script)***: If a script defines async, it is executed asynchronously as soon as it is available; if async is not present but defer is, the script is executed after the page has finished parsing; if none is specified, the script is fetched and executed immediately, before continuing parsing the page.

***sandbox (iframe)***: The sandbox attribute enables a set of restrictions on the iframe content. By default, sandboxed content:

- is considered to be from a unique origin
- is prevented from navigating browsing contexts other than the sandboxed browsing context itself
- is prevented from submitting forms
- is prevented from running scripts
- is prevented from running automatic features (e.g. autofocus, refresh through **`meta`** tag, automatically starting playback, ...)
- prevents links from targeting other browsing contexts
- is prevented from loading plugin content (e.g. **`embed`**, **`object`**, **`applet`**)
- is prevented from using the seamless attribute on nested iframes

Relaxations can:

- Remove the unique-origin restriction, thus reverting to the default same-origin policy
- Allow the navigation of the top-level browsing context
- Allow form submission
- Allow the use of scripts and automatic features (no point in blocking these since they are easily enabled from scripts). Popups however are still blocked.

Next to the sandbox attribute, the content can be served with a **`text/html-sandboxed`** MIME type, which also offers protection in non-iframe contexts.

***seamless (iframe)***: Enables the seamlessintegration (i.e. with no visual indications) of the content into the embedding document. This applies to the visual aspects (e.g. applying the same stylesheets) as well as content retrieval aspects ( e.g. if the user agent supports listing all the links in a document, links in "seamlessly" nested documents would be included in that list without being significantly distinguished from links in the document itself.). The attribute only applies if the embedding document and embedded document have the same origin, or if the embedding document's address (which is the absolute URL the moment the document was created) and the embedded document have the same origin, or if the embedding document is an iframe'ssrcdoc document.

***srcdoc (iframe)***: The srcdoc attribute allows the specification of HTML content that needs to be rendered in the iframe. The srcdoc attribute always has priority over the src attribute, that can be used to specify fallback content in case srcdoc is not supported. This can be used together with the sandbox attribute and does not require the fetching of an additional document. The URL of such an iframe is **`about:srcdoc`** and the origin is the origin of the document containing the browsing context of the srcdoc document.

***manifest (html)***: Allows the specification of a manifest file, that specifies which parts should be cached for offline use (not included here, but in separate section "Application Cache").

***rel (link, a, area)***: The rel attribute allows the specification of special link types, such as a help file, a pingback link, ... If a rel attribute is not defined for an a or area element, the element just creates a normal link with no special meaning. For link elements, the type must always be specified.

## Capabilities

The specification enables the following capabilities:

- Automatically focus an input element, potentially triggering focus handlers
- Specify form controls that are connected to a form, but occur outside of its natural scope (between the tags)
- Override form settings from the submission button's element
- Apply content restrictions with a sandbox
- Seamlessly incorporate a frame and its content into a webpage of similar origin
- Specify inline iframe content, without loading a separate document
- Create links with special types

### User Involvement

The specification involves the user in the following way:

- The user may submit a form using an associated form control

### Security/Privacy Considerations

The specification explicitly mentions the following security/privacy considerations:

- Assigning a document in a sandbox a unique origin also prevents access to `document.cookie` and `localStorage` *[11].*
- A document can break out of its sandbox if both the `allow-same-origin` and `allow-scripts` keywords are given as sandbox parameters.
- A document that is loaded in a sandboxed iframe should also be served with the sandboxed MIME type, otherwise the attacker can trick the user in loading the document directly (not in an iframe).
- A srcdoc document has no explicit URI.

The following considerations are assumed by the specification and are derived from its contents:

- Out-of-element form controls cannot compromise the form's integrity.

### Threat Enumeration

**SQ1 - Well defined / Secure**

***HTML5ATTR-SECURE-1.Form Tampering*** A form's properties can be overridden using the new attributes on a submit button and such a button can occur anywhere in the page (including outside the <form></form> node), while still being associated with a form. This makes it easier to insert a button into a page since the button can be anywhere in the page. By applying a simple HTML injection, an attacker may thus override the form's destination and trick the user into clicking the new submit button, thus stealing the entered form data. This means that existing input validation approaches that focus on script injection can easily be circumvented by injecting HTML, and validation techniques should be adapted appropriately.

***HTML5ATTR-SECURE-2.Coarse-Grained Control*** The sandbox attribute offers a set of restrictions, with some relaxations. Both the restrictions and relaxations are very coarse-grained and typically only offer an all or nothing option. Examples are enabling or disabling scripts, enabling or disabling forms, ... While this valuable feature supports very simple cases of least privilege, more complex cases are not supported. An example of a complex case is a same-origin document that requires scripts, but should not have access to previously authorized sensitive APIs (e.g. Geo-location, System Information).

**SQ2 - Isolation Properties**

***HTML5ATTR-ISOLATION-1.Disabling Click-jacking Protection*** The sandbox behavior of disabling top-level navigation by default disables numerous popular click-jacking countermeasures. This means that the sandbox attribute facilitates click-jacking attacks *[17]*.

***HTML5ATTR-ISOLATION-2.Srcdoc URI*** A srcdoc document has `about:srcdoc` as URI. The specification explicitly states that the origin of an srcdoc document is derived from the container of its browsing context. Additionally, resolution of relative URLs or fetching resources is documented for srcdoc documents. URI determination is not, meaning that `about:srcdoc` is the URI of the document. This can cause problems in cases where the URI is used (e.g. for permission management). Currently, no specifications suffer from this problem (although a previous version of the Geo-location API did), but future specifications should be aware of this potential issue.

## 4.4.HTML5 - Navigation

The HTML5 specification defines how navigation across documents/browsing contexts can occur.

### Description

Navigation in HTML5 involves two browsing contexts, where the source browsing context initiates the navigation of a navigated browsing context. The rules determining the navigation policy are spread over multiple sections (5.1.4 and 5.5.1). The former section (5.1.4) specifies when a source browsing context is permitted to navigate another browsing context. The latter (5.5.1) discusses the actual navigation algorithm, which takes additional attributes, such as sandbox or seamless, into account.

A source browsing context *A* is permitted to navigate another browsing context *B* if one of the following conditions is true:

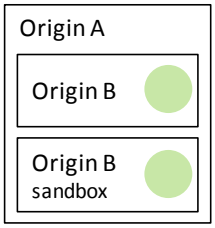
- The origin of the active document of *A* is the same as the origin of the active document of *B*
- *A* is a nested browsing context and *B* is its top-level browsing context
- *B* is an auxiliary browsing context (= related to a top-level browsing context, but not nested through an element (e.g. with target)) and *A* is allowed to navigate the opener browsing context of *B*
- *B* is not a top-level browsing context, but has an ancestor browsing context whose active document has the same origin as the active document of *A*

The navigation algorithm imposes the following rules:

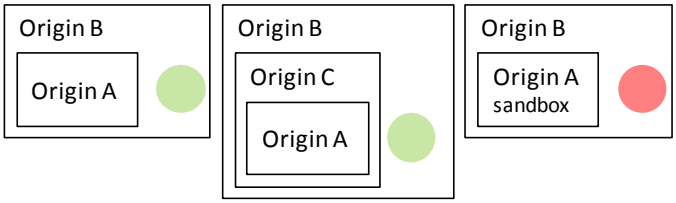
- Abort the algorithm if all of these conditions hold:
  - *A* is not the same as *B*
  - *A* is not an ancestor of *B*
  - *B* is not both a top-level browsing context and an ancestor of *A* (taken care of by the *sandboxed top-level navigation browsing context* flag)
  - *A* had *its sandboxed navigation browsing context* flag set when the active document was created (this flag is always set when the sandbox attribute is specified)
- Abort the algorithm if all of these conditions hold:
  - *A* is the same as *B*
  - *B* has the seamless browsing context flag set

- *B* is not navigated through an explicit self-navigation override (`target="_self"`)

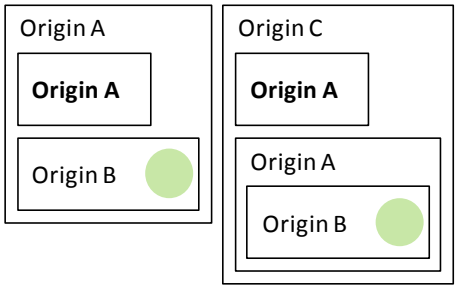
The table below clarifies the navigation possibilities. Navigation always goes from the context with origin A to the context with origin B. In case there are multiple contexts with origin A, the bold one initiates the navigation. **A green circle means that navigation is possible, a red one means that it is not.**



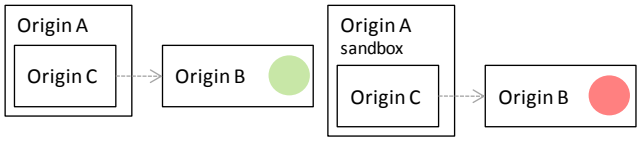

A browsing context A can always navigate its children. This is not restricted by the sandbox attribute




A browsing context A can navigate its top-level browsing context. The sandbox attribute restricts this, but allows overriding




A browsing context A can navigate a sibling, if the sibling has an ancestor of the same origin as A




A browsing context A can navigate an auxiliary browsing context B, if A can navigate the opener of B. If A is sandboxed, this is restricted

## Capabilities

The specification enables the following capabilities:

- Navigation of a selected set of browsing contexts

## User Involvement

The specification does not involve the user, although navigation can be triggered by users (not relevant for this analysis).

### Security/Privacy Considerations

The following considerations are assumed by the specification and are derived from its contents:

- The nested relationship of documents cannot be manipulated

### Threat Enumeration

The analysis did not reveal any threats.

## 4.5.HTML5 - Application Cache

The application cache enables the persistent caching of specified resources. These resources will be served when requested by the user agent in offline mode.

### Description

To enable the offline usage of a web application, the author needs to define which resources are needed, by listing them in the manifest file. The manifest file has to be referenced using the manifest attribute on the main HTML document of the page, after which the browser will store the required files in the application cache. At the end of the caching process, the browser will re-fetch the manifest to verify that the latest version has been cached. At this point, the application is available offline.

The specification defines an application cache API, which offers information on the status of the application cache, as well as an operation to update the cache. If a newer cache is available, the swap-operation allows switching to this version. The update process can be run in the background using Web Workers.

**Application Cache Structure**

All resources stored in an application cache are stored with their out-of-band metadata, such as HTTP headers. The application cache itself consists of the following items:

- Master entries: documents where a browsing context was navigated to and that included a manifest file
- Manifest: the manifest file
- Explicit entries: the resources occurring in the `CACHE` sections of the manifest
- Fallback entries: the resources occurring in the `FALLBACK` sections of the manifest
- Fallback namespaces: the namespaces pointing to a fallback entry
- Online whitelist namespaces: the namespaces that always need to be fetched from the network
- Online whitelist wildcard: if set to open, all URLs not in the cache will be fetched online, otherwise, all URLs not in the cache are treated as unavailable.

Explicit and fallback entries can be foreign, which means that they specify their own manifest, which differs from the manifest where they are specified (other manifest URL).

An *application cache* is part of *an application cache group*, which is a set of chronological application caches for one manifest file. Each time a new manifest file is detected, a new application cache is added to the group. The *relevant application cache* is the newest

application cache in the group that is marked as completed. Only the most recent application cache can be incomplete, because it is still downloading.

Resources can occur in multiple application caches in different application cache groups. In that case, the user agent needs to decide which application cache to use, based on:

- Which application cache was most recently updated
- Which application cache was being used to display the resource leading up to this request
- Which application cache is preferred by the user

**Manifest Format**

The cache manifest can contain three sections:

- **`CACHE`**: resources that explicitly need to be cached
- **`NETWORK`**: resources that are not cached. If such a resource is requested, the request always bypasses the cache. A match here takes priority over fallback namespaces.
- **`FALLBACK`**: resources that need to be used in case no cache entry was found. Fallback resources can be bound to a namespace (a path prefix)

**Manifest Constraints**

The manifest is subject to the following constraints:

- The manifest is always of the same origin as the master document (the document linking to the manifest)
- Explicit entries:
  - The specified resources need to have the same scheme as the manifest
  - If the specified scheme is HTTPS or equivalent, the specified resources need to have the same origin as the manifest
- Fallback entries:
  - Fallback namespaces and resource URLs need to have the same origin as the manifest
- Online whitelist:
  - Entries in the online whitelist need to have the same scheme as the manifest

An additional constraint is that the cache download process does not follow redirects. This way, several potential issues are prevented:

- Captive portals blocking downloads (e.g. Wifi authentication points)
- Adding of resources under URLs that are not allowed by the networking model, thus leaving orphan entries
- Adding of resources under URLs that differ from their true URL

### Capabilities

The specification enables the following capabilities:

- Persistently store resources offline (both same-origin and cross-origin)
- Load a previously cached version of a resource

### User Involvement

The specification involves the user in the following way:

- Authorize the caching of an application (suggestion)
- Purge cached data (suggestion)

### Security/Privacy Considerations

The specification explicitly mentions the following security/privacy considerations:

- The specification suggests that the caching process can ask the user for permission. This is especially important on space-constrained devices or in highly privacy-sensitive environments.
- The specification further suggests treating the application cache the same way as cookie data. For instance, the user should be able to purge all data for a specific origin, which includes cookies, local storage and application caches.

The following considerations are assumed by the specification and are derived from its contents:

- The cached version of the application offers the same protection (e.g. against unauthorized access to data) as the online version.

### Threat Enumeration

#### SQ2 - Isolation Properties

***HTML5CACHE-ISOLATION-1.Restricted Context*** The specification does not mention the behavior of offline resources in a restricted browsing context. For a sandboxed application, the behavior is indicated by the sandboxed MIME type, specified in the headers. In a private browsing context however, nothing is specified. This may be problematic for cached data, such as a JSON file, which can be stored in normal mode and then accessed in private browsing mode.

#### SQ3 - Consistency

***HTML5CACHE-CONSISTENCY-1.Permission System*** The specification mentions that optionally, permission to start the download process can be obtained from the user. There is no mention of a concrete permission system or any constraints on obtaining such permissions.

## 4.6.HTML5 - Browser Features

### Description

***Custom scheme and content handlers***: using `registerProtocolHandler` and `registerContentHandler`, sites can register (only) themselves as possible handlers for a particular content type or scheme (e.g. `sms:`). Every URL matching the criteria is sent to the URL of the registered content handler. Notice that the URL is sent to the content

handler, and not the content itself. This means for instance that an external handler (outside the company network) cannot access intranet files. Additionally, for content handlers this should not apply to content retrieved by a non-idempotent request (e.g. another method than GET), since the external service cannot retrieve the same content. The actual implementation is up to the browser vendor, but asking user confirmation is recommended (e.g. Firefox 4.0 does this).

***Editable Content***: **`contentEditable`** allows an element to be editable, which allows the user to add text or remove text, move components around, ... An entire document can be made editable with the **`designMode`** attribute. A document or element becomes non-editable if its state changes (external manipulation) or if it is destroyed.

***Drag and Drop***: elements can use the **`draggable`** attribute to indicate they can be dragged and the **`dropzone`** attribute to indicate they can receive drop events. The appropriate actions can be implemented using handlers for the different drag/drop events. The data involved in the drag and drop operation is stored in the drag data store. Drag and drop is also supported from external sources (e.g. dragging files from your desktop to a web page).

***History Interface***: Each browsing context, including nested browsing contexts has a distinct session history. A session history consists of a sequence of documents. One of the documents of the session history is known as the *current entry*. The history interface provides the following programmatic operations:

- **`go`**: Allows to jump to an entry, indicated by a delta (+ or -). If no delta is given, the current entry is reloaded.
- **`back/forward`**: Allows to go 1 step back or forward
- **`pushState`**: Adds an entry to the history, using the given state information and suggested title. Optionally, a URL can be provided, allowing to store a different address than the document's current address. The restrictions discussed below apply. After updating the history, the document's current address is also updated. This is a property change, not a reload or navigation event.
- **`replaceState`**: Replaces the last entry with a new entry, using the same procedure as **`pushState`**.

To store a different URL than the document's current address, the following restrictions apply:

- If the URL compared to the document's address differs in any part, except for the path, query (i.e. GET parameters) or fragment, a security exception is raised. This effectively prevents a document from storing a URL for a different protocol, host or port.
- If the origin of the URL differs from the origin of the script's document and either the path or query parts from the previous step differ, a security exception is raised. This prevents a sandboxed document from spoofing documents in the same origin: a sandboxed document has a URL within the origin as address, but is assigned a globally unique origin.

### Capabilities

The specification enables the following capabilities:

- Install custom handlers for protocols or document types
- Let the user edit rendered HTML content
- Drag and drop html elements within the browser
- Drag and drop system files to the browser
- Navigate the document's history
- Add entries to the document's history

### User Involvement

The specification involves the user in the following way:

- Approve the installation of a new scheme or content handler (suggestion)
- Initiate and terminate a drag and drop event

### Security/Privacy Considerations

The specification explicitly mentions the following security/privacy considerations:

- Content and Protocol Handler Security
  - User agents *should* not allow the overriding of existing schemes such as HTTP or HTTPS, to avoid hijacking of all traffic
  - User agents are *strongly recommended* to not overwrite defaults and new handlers should never automatically be used
  - User agents *should* gracefully prevent registration spamming, where a large number of requests are made
  - User agents *should not rely* on the *title* property given when registering a handler, but instead use the handler's domain
  - User agents *should* protect against interface attacks by malicious strings in visible handler data (e.g. the URL)
  - Intranet URLs *can* be leaked (Although the data will not be accessible from outside the intranet). User agents *might* wish to provide some form of whitelist/blacklist
  - User agents *should* not send secure URLs (HTTPS) to third party content handlers
  - User agents must never send username or password information in the content URLs
- Drag and drop security risks
  - User agents must not make the data from the **`dragstart`** event available until the drop event occurred
  - User agents must consider a drop to be successful if the user specifically ended the drag operation
  - User agents *should* take care not to start drag and drop operations in response to script actions
- History Security
  - Using **`pushState`**, a malicious page can fill up the session history with its own entries, effectively crippling the back-button. The specification advises implementers (i.e. browser vendors) to take this into account, for example by

offering to return to the previous page (instead of the previous state for the current page), or to disallow the use of `pushState` from a timer or event listener.

The following considerations are assumed by the specification and are derived from its contents:

- The drop zone (and the associated origin) can be unambiguously identified by the user
- Some of the security measures against the abuse of `pushState` assume that the history has an infinite size (thus the previous relevant entry cannot be pushed off the stack)

## Threat Enumeration

### SQ1 - Well defined / Secure

***HTML5BROWSER-SECURE-1.Content/Protocol Handlers*** The specification mentions an extensive list of potential security issues with the registration of custom content or protocol handlers. The most important ones are excessive use of a content/protocol handler (e.g. HTTP traffic, ...) and leakage of confidential information through URLs. Unfortunately, no real solutions, except for "the user agent should to take care of it", are given.

***HTML5BROWSER-SECURE-2.Sleeping Content/Protocol Handlers*** Once a content handler is registered, the user has to remember this himself. This can be problematic if the user registers a handler which is almost never used. After quite some time, the user forgets this, and is not reminded of the fact. Even if the content handler is used, there is no explicit notification or indication required (awareness to the user).

***HTML5BROWSER-SECURE-3.Content Handler URLs*** Sending URLs to content handlers can suffer from different problems. One is leaking confidential information (e.g. username/password, embedded authentication token, unique or intranet filenames, ...). Additionally, accessing an URL can require an access token (e.g. a session cookie), which means that the content handler cannot access the document. Preventing HTTPS URLs to be loaded via a content handler also seems unnecessary and unjustifiable. If the document is publicly accessible (e.g. does not require a session cookie), the handler cannot access it. Otherwise, there is little reason to prevent this specific behavior, which might even discourage the use of HTTPS.

***HTML5BROWSER-SECURE-4.Infinite History*** The countermeasure proposed by the specification to prevent history abuse, assumes that a history is infinite in size. It suggests that instead of offering the option to return to the previous state, the user agent can offer to return to the previous page. If a malicious page adds a large amount of new states, the last relevant page will probably be pushed off the history stack.

### SQ3 - Consistency

***HTML5BROWSER-CONSISTENCY-1.Handler Permissions*** The specification says very little about the approval of installing a new content/protocol handler. The specification suggests that user approval can be asked, but does not mention anything about warning the user, easy revocation lists, etc...

**SQ4 - User Involvement**

***HTML5BROWSER-USER-1.Drag and Drop Target*** When a user drags an item to a certain area, the user is expected to know which element of the website (and more important which origin) will actually handle the drop event. There are numerous ways to confuse the user. One (legitimate) way is embedding an iframe that is not visually distinctive from the rest of the page. A second (malicious) way is to overlay the drop zone with an invisible element, effectively stealing the drop event and its data.

## 4.7.Web Messaging

The Web Messaging specification defines two mechanisms for communicating between browsing contexts in HTML documents.

### Description

The specification offers two communication mechanisms: *cross-document messaging* and *channel messaging*. The former sends a message in the form of a single asynchronous event. The latter opens a channel between two contexts, allowing the asynchronous posting and receiving of messages through the channel. Both mechanisms allow the passing of messages of any type, although the transferred message is a clone of the original message, as defined by the structured clone algorithm in the HTML5 specification [1]. In essence, the algorithm duplicates the values of each field into a new object, and does this in a recursive way.

Cross-document messaging allows scripts to post messages to another window. The posting of a message with the `postMessage` operation requires the specification of the target origin, which can either be a URI, a wildcard (`*`) or the current origin (`/`). The target origin is compared against the origin of the document of the window on which the method was invoked. If a message is sent to the wrong origin, it is silently discarded. Receiving messages is possible by handling the message event.

Channel messaging creates two ports, of which one needs to be transferred to the remote browsing context. The `postMessage` mechanism (used in cross-document messaging) allows the passing of ports between two browsing contexts. Message ports belonging to the same channel are entangled. The message ports also offer the `postMessage` operation and the message event. The `postMessage` operation accepts a message and optionally another array of ports. After use, ports should be closed explicitly to avoid needless resource consumption.

### Capabilities

The specification enables the following capabilities:

- Send messages to other browsing context (opt-in by means of an event handler)
- Establish a message channel with another browsing context

### User Involvement

The specification does not involve the user.

## Security/Privacy Considerations

The specification explicitly mentions the following security/privacy considerations:

- Cross-document messaging: Security guidelines for authors
  - Check the origin attribute, to ensure that messages are only accepted from trusted domains
  - Check the data format
  - Do not use the wildcard for confidential data
- Cross-document messaging: Security guidelines for user agents
  - Do not allow scripts to trigger arbitrary events (e.g. a message event) on cross-origin objects

The following considerations are assumed by the specification and are derived from its contents:

- The origin of a message is correct and unambiguous
- The origin of an established message port does not change
- An established message port cannot be passed to other origins

## Threat Enumeration

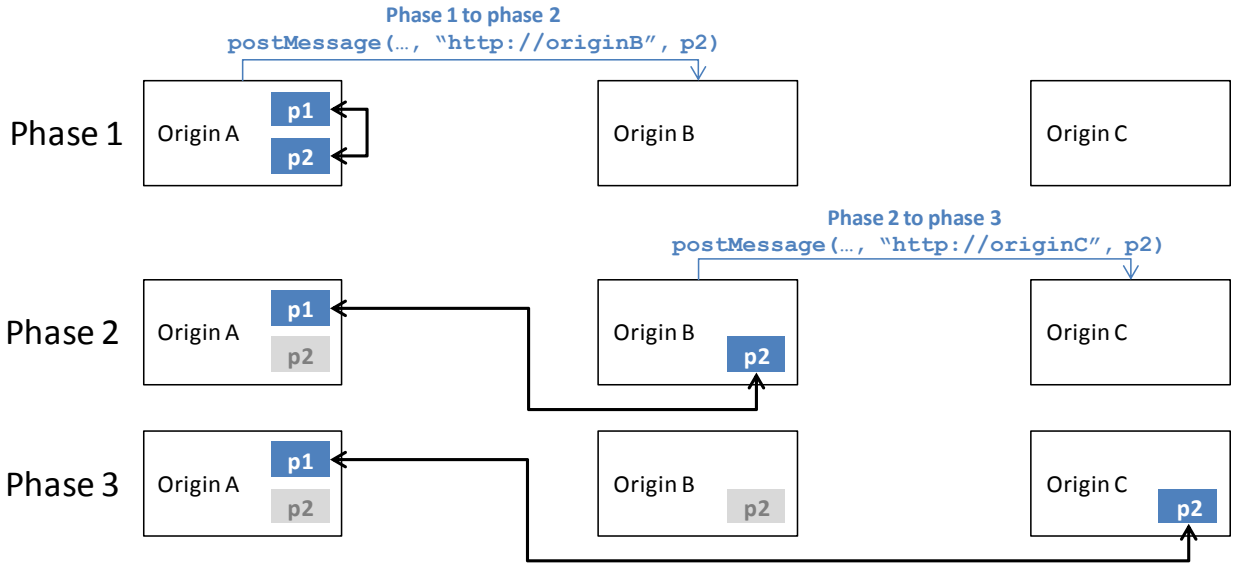
### SQ1 - Well defined / Secure

***WEBMSG-SECURE-1.Origin Determination*** The specification uses document origins to check the destination origin. In the next step (Algorithm 4.3 step 7), the source origin is specified as "the script's origin" (which may be different from the script's document's origin). This probably should be the script's document's origin. All major browsers do implement it this way.

### SQ2 - Isolation Properties

***WEBMSG-ISOLATION-1.Document/Port Handler's Origin*** A window or message port uses an event handler to receive messages. The effective script origin of this handler is never checked. This can lead to potential issues, for instance if the script relaxes its domain (using `document.domain`), its handler can be overwritten by another document with the same `document.domain` value.

***WEBMSG-ISOLATION-2.Port Endpoint Origin*** This threat is illustrated in the figure below. When a script in a document with origin A creates a pair of ports (p1 and p2) (figure: phase 1) and shares p2 with another origin B, using `postMessage`, the destination origin B is explicitly stated (figure: going from phase 1 to phase 2). Any messages sent using p1 and p2 no longer include the source or destination origin (figure: phase 2). It is however possible for party B to send port object p2 to another party C (figure: going from phase 2 to phase 3). This will cause p2 to become untangled (no longer usable) and will link p1 to the port owned by party C (figure: phase 3). This means that no origin guarantees can be given on any messages sent over a message channel. As in this scenario, party A can perform the same action with p1.




### 4.8.XMLHttpRequest Level 1 and 2

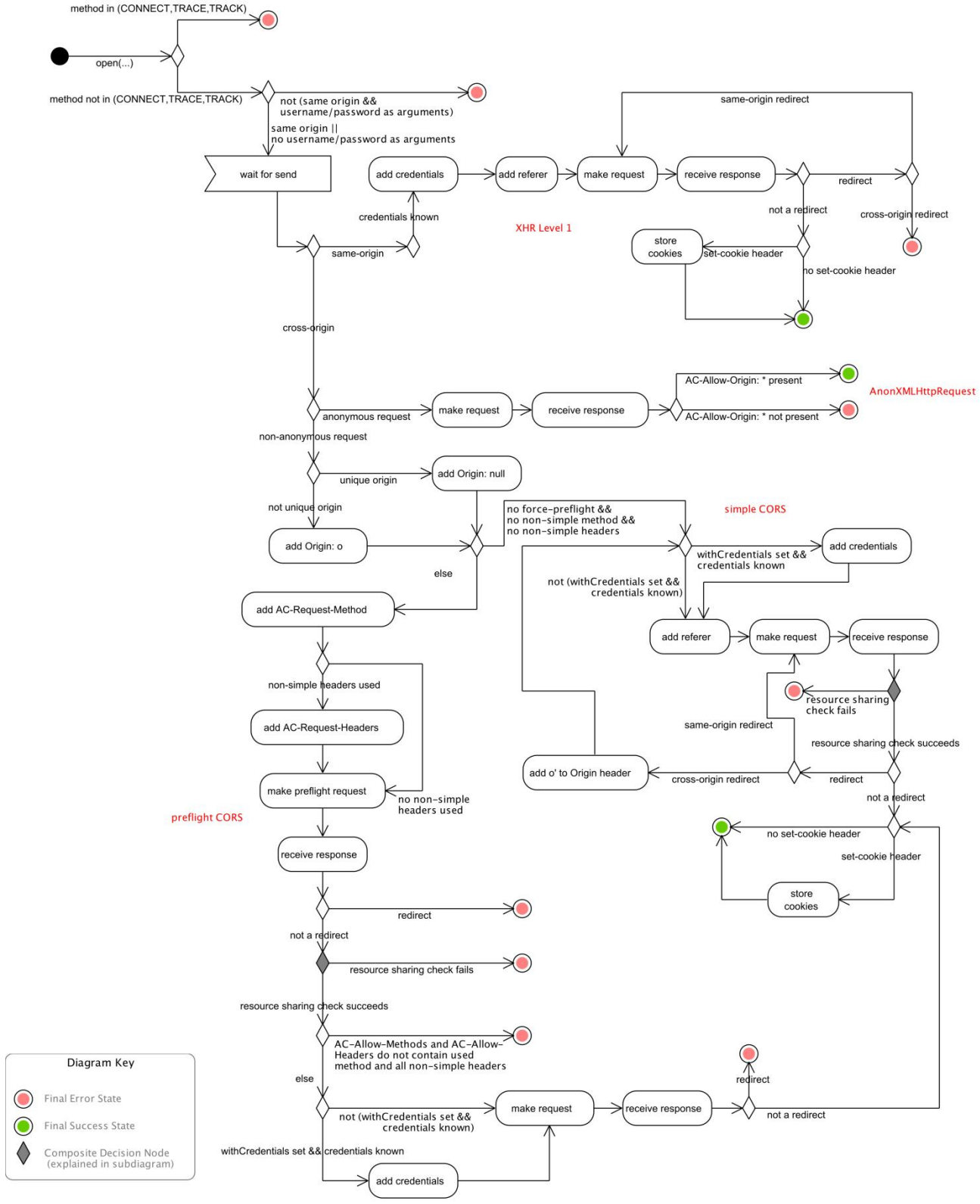
The XMLHttpRequest specification defines an API that provides scripted client functionality for transferring data between a client and a server.

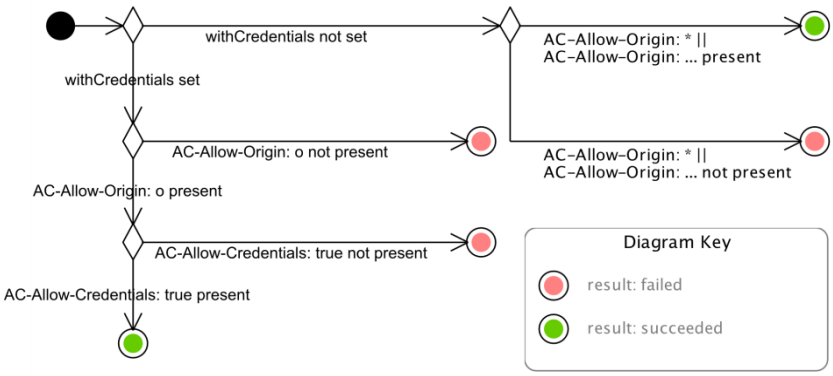
#### Description

The XMLHttpRequest specification is the HTTP API for ECMAScript. It offers script the possibility to initiate HTTP requests and process the responses. The XHR specification has evolved over time, which is why there is a Level 1 and Level 2 version. Level 1 offers same-origin communication, while level 2 extends the functionality towards cross-origin communication.

We have constructed a UML activity diagram that shows how an XMLHttpRequest is processed according to the specifications. The following notes are important when reading the diagram:

- Credentials includes cookies, HTTP authentication and client-side SSL certificates (as stated in the specification)
- A globally unique origin identifier is always considered to be cross-origin
- The diagram only contains steps important to this analysis (i.e. remote communication). Other features, including the ones listed below, are omitted for clarity reasons.
    - Internal bookkeeping of the object (states, events, ...)
    - Caching
    - Redirect loop detection
- The activity *add referer* is not explicitly mentioned in the spec, but for an anonymous request, the referer is explicitly left out. For clarity reasons, we chose to model the inclusion instead of the exclusion.


method in (CONNECT,TRACE,TRACK)
open(...)
method not in (CONNECT,TRACE,TRACK)
not (same origin && username/password as arguments)
same origin || no username/password as arguments
wait for send
add credentials
add referer
make request
receive response
same-origin redirect
redirect
credentials known
XHR Level 1
not a redirect
cross-origin redirect
store cookies
set-cookie header
no set-cookie header
same-origin
cross-origin
AC-Allow-Origin: * present
AC-Allow-Origin: * not present
AnonXMLHttpRequest
anonymous request
make request
receive response
non-anonymous request
unique origin
add Origin: null
not unique origin
add Origin: o
no force-preflight && no non-simple method && no non-simple headers
simple CORS
withCredentials set && credentials known
add credentials
not (withCredentials set && credentials known)
else
add AC-Request-Method
add referer
make request
receive response
non-simple headers used
resource sharing check fails
add AC-Request-Headers
same-origin redirect
resource sharing check succeeds
add o' to Origin header
cross-origin redirect
redirect
make preflight request
no non-simple headers used
not a redirect
preflight CORS
no set-cookie header
set-cookie header
store cookies
receive response
redirect
not a redirect
resource sharing check fails
resource sharing check succeeds
AC-Allow-Methods and AC-Allow-Headers do not contain used method and all non-simple headers
else
redirect
not (withCredentials set && credentials known)
make request
receive response
not a redirect
withCredentials set && credentials known
add credentials
Diagram Key
Final Error State
Final Success State
Composite Decision Node (explained in subdiagram)




The sub-diagram below shows the resource sharing check from the main activity diagram.

## Capabilities

The specification enables the following capabilities:

- Same-Origin communication
  - HTTP Requests to the origin of the enclosing document
  - Use of any HTTP method except for CONNECT, TRACE or TRACK
  - Setting of custom HTTP headers, except for specific protected headers (e.g. host)
  - Automatic inclusion of cached HTTP credentials unless custom username/password is supplied
  - Automatic inclusion of cookies for the destination domain
  - Retrieving custom HTTP headers, except for specific protected headers (e.g. **Set-Cookie**)
  - Retrieving the response as plain text or as a parsed XML document
- Cross-Origin communication
  - HTTP requests to other origins (by including the Origin header and restricting access to the response)
    - Including custom username/password credentials is not allowed
  - Anonymous requests (no referer, no `Origin` header, no credentials, no cookies) – allowing the browser to make requests without disclosing potentially identifying or sensitive information.
  - Additional data types for formatting data: `ArrayBuffer`, `Blob`, `File` and `FormData`

## User Involvement

The specification does not involve the user.

### Security/Privacy Considerations

The following considerations are assumed by the specification and are derived from its contents:

- The origin of the enclosing document is correct and unambiguous
- The destination URI of the request is correct and unambiguous
- The callback handler and its script environment are not compromised

### Related Specifications

The following specifications are closely related:

- Cross-Origin Resource Sharing
- Uniform Messaging Policy

### Threat Enumeration

Since this specification implements *Cross-Origin Resource Sharing* and supports *Uniform Messaging Policy*, the issues revealed by the analysis of those specifications also need to be taken into account.

**SQ1 - Well defined / Secure**

***XHR-SECURE-1.Inconsistent Method Checks*** The specification performs two method checks, of which one is case-insensitive and the other is case-sensitive. The inconsistency does not seem logical and is probably a mistake.

## 4.9.Cross-Origin Resource Sharing

This specification defines a mechanism for cross-origin communication between client and server, which requires opt-in by the server.

### Description

CORS defines a mechanism to enable client-side cross-origin requests. CORS only defines algorithms which can be followed by an implementing API, such as XMLHttpRequest Level 2.

The main idea behind the specification is that a client provides the server with adequate information to decide whether the requesting origin has access to the requested data or not. This decision is sent to the client, which effectively enforces this decision by either granting or denying access to the requested resource. CORS identifies *simple* requests, which use the HEAD, GET or POST method and use only headers and content types from a specific list, and *actual* requests which are *non-simple* requests. To prevent CSRF-like attacks, actual requests (which are requests that you cannot make using common HTML elements) are required to have a pre-flight request, which acquires authorization from the server to make the actual request. Only when the server grants this permission will the actual request be executed.

The CORS mechanism depends on the following headers:

- Request headers

- Origin
- Access-Control-Request-Method (preflight)
- Access-Control-Request-Headers (preflight)

- Response headers
  - Access-Control-Allow-Origin
  - Access-Control-Allow-Credentials
  - Access-Control-Allow-Expose-Headers
  - Access-Control-Allow-Max-Age (preflight)
  - Access-Control-Allow-Allow-Methods (preflight)
  - Access-Control-Allow-Allow-Headers (preflight)

### Capabilities

The specification enables the following capabilities:

- Making *simple* cross-origin requests, regardless of whether they are allowed or not
- Accessing *simple* cross-origin responses if the server allows this
- Making *actual* cross-origin requests if allowed by a preflight request

### User Involvement

The specification does not involve the user.

### Security/Privacy Considerations

The specification explicitly mentions the following security/privacy considerations:

- Resource authors are:
  - strongly encouraged to ensure that a simple request cannot have server-side effects
  - to check the Host header, to prevent DNS rebinding attacks

- User Agents:
  - Can take additional precautions (e.g. strict cache management, refuse to connect to certain URLs)
  - Are encouraged to limit **max-age**, to prevent items from staying the preflight cache too long
  - Can terminate the algorithm when initiating a request (e.g. blacklisting, local policy, ...)
  - Are encouraged to apply the cross-origin security rules to other elements as well (e.g. an **img** tag)

The following considerations are assumed by the specification and are derived from its contents:

- The origin of the enclosing document is correct and unambiguous
- The destination URI of the request is correct and unambiguous

- The *simple* communication mechanisms offer no additional capabilities over traditional HTML elements combined with JavaScript
- An *actual* request is not possible without the server's consent (by responding to a preflight request)

### Related Specifications

The following specifications are closely related:

- XMLHttpRequest level 1 and 2
- Uniform Messaging Policy

### Threat Enumeration

**SQ1 - Well defined / Secure**

***CORS-SECURE-1.Legacy Servers*** One of the assumptions of CORS is that there is no additional attack surface towards legacy servers, which have no knowledge of the CORS spec. This is largely true, except that the spec enables two unprecedented scenarios: (i) the body of a POST request is no longer constrained to the key=value format imposed by form elements, which may enable malicious cross-origin REST or JSON calls, or facilitate a cross-channel attacks [26]; (ii) the use of the HEAD method was previously not possible using forms, which may be problematic since PHP processes a HEAD request similarly to a GET request, but aborts execution at the first output (a HEAD response cannot have body).

***CORS-SECURE-2.Unnecessary Processing*** The CORS specification states that if a CORS-aware server receives a simple request from an origin, which cannot get access to the response, no headers should be included. A question about this decision is why the server should process the request at all? Can it not just return an empty response, without taking any server-side action? Obviously, the client-side checking mechanism still remains in place to prevent unauthorized access to responses coming from legacy servers.

**SQ2 - Isolation Properties**

***CORS-ISOLATION-1.Unique Origins*** When run in a document with a globally unique identifier for an origin, the Origin header specification requires that null should be sent as the value of the Origin header. The algorithms listed in the CORS specification do not explicitly take the null value into account, leading to some risk scenarios. It is for instance valid that a request sends origin null and the server responds with an Allow-Origin header with the value null.

## 4.10. Uniform Messaging Policy

UMP enables cross-site messaging while attempting to minimise attacks abusing cookies and other credentials.

### Description

UMP allows the resource owner to consent to cross-origin retrieval and enforces origin independent messaging. This is realized using a *uniform* request and a *uniform* response.

A non-*uniform* response as an answer to a uniform request must be met by an error. It must not be made available to the requesting node and if it is a redirect, it must not be followed.

A *uniform* request:

- Must not contain a username/password
- Must use the GET or POST method
- If a body is provided, it must be of the type
    - `application/x-www-form-urlencoded`
    - `multipart/form-data`
    - `text/plain`
- Must not be sent over a client-authenticated connection (e.g. client-side SSL certificates)
- Must not add any identifying data to the request (e.g. Referer header, Origin header, cookies)
- May be an Origin header with a null value

A uniform response:

- Must have a single `Access-Control-Allow-Origin: *` header if the resource can be accessed
- All headers except for a whitelist must be filtered from the response by the user agent

Because a uniform request does not contain any identifying information, attacks such as cross-site request forgery are not possible. Authentication and authorization should be managed using appropriate protocols that are not susceptible to attacks.

The specification also offers advice for application authors:

- A resource not useful to other sites (e.g. a login page) should not return uniform responses. It should also still incorporate protection against CSRF and click-jacking.
- Publicly available resources should always return a uniform response
- A resource that requires authorization should check permission tokens and return a uniform response if access is allowed
- A GET response with a JavaScript body (scripts or JSON) should be a uniform response, since the same content is already accessible through script inclusion (via the script tag), which is exempted from the Same Origin Policy protection.

### Capabilities

The specification enables the following capabilities:

- Making cross-origin uniform requests
- Accessing cross-origin uniform responses

### User Involvement

The specification does not involve the user.

### Security/Privacy Considerations

The specification makes no explicit or implicit security/privacy considerations

### Related Specifications

- The following specifications are closely related:
- XMLHttpRequest Level 1 and Level 2
- The AnonXMLHTTPRequest of the XHR Level 2 spec is very similar to UMP. The major difference is that the spec allows other methods than GET and POST.
- Cross Origin Resource Sharing

### Threat Enumeration

The analysis did not reveal any threats.

## 4.11. Web Storage

The Web Storage API provides persistent client-side storage of key-value pair data.

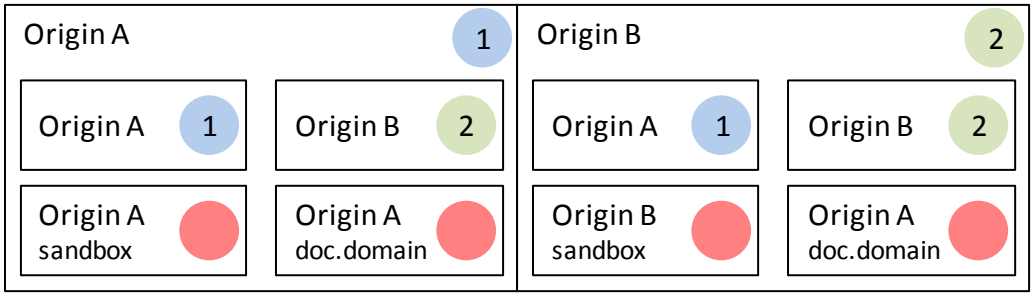
### Description

Web Storage offers each origin its own separate client-side storage area for key-value pairs (keys as in indices, not cryptographic keys), similar to the cookie mechanism. A valid key is any string, including the empty string. A valid value is any piece of data supported by the structured clone algorithm: each value is cloned when storing it and is cloned when retrieving it (preventing indirect updates by reference).

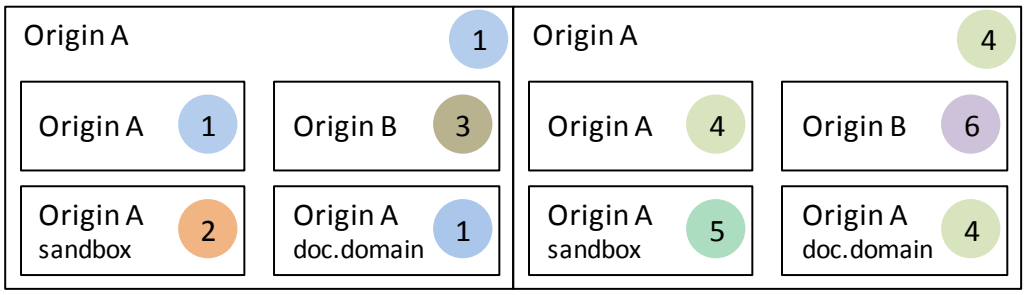
Modifying the storage (**`setItem`**, **`removeItem`**, **`clear`**) also triggers a storage event to each document that has a storage object that is affected (e.g. `window.localStorage`). The event contains the information about the change.

Web Storage supports two different storage areas: *localStorage* and *sessionStorage*.

***localStorage***: Local storage is persistent storage with no limited lifetime. The user agent manages a set of local storage areas, one for each origin. The local storage is accessible from any document, as long as the effective script origin matches the origin of the document that accessed the storage area. The data in a local storage area is only erased upon user demand or in case of space constraints. The figure below shows in which cases the local storage area for an origin is shared.



***sessionStorage***: Session storage is meant to support independent (same-origin) concurrent transactions in separate windows, a case that cookies do not support. Each top level browsing context has a set of storage areas, one per origin. The lifetime of a session storage area is equal to the lifetime of the browsing context (which can be longer than the browser process, due to session restore mechanisms). The storage area assigned to a document never changes, not even if an iframe is relocated to a different top-level browsing context. If a top level browsing context is cloned, the session storage areas must be copied to the new context. The figure below shows in which cases the session storage area for an origin is shared.




The specification proposes a limit of 5MB per origin. It also suggests that user agents allow the user to enlarge the available space. Additionally, the specification states that a user agent should prevent sites from using subdomains (`a1.example.com`, `a2.example.com`) to circumvent the storage limits (it does not state how to prevent this).

### Capabilities

The specification enables the following capabilities:

- Persistent client-side store data within an origin
- Session-specific storage, even after a resume
- Separate storage area for each top-level browsing context, regardless if they share the same origin or not
- Add (limited amounts of) data to the local disk

### User Involvement

The specification involves the user in the following way:

- Enlarge storage space (suggestion)
- Pruning the locally stored data, similar to managing cookies (suggestion)
- Whitelisting or Blacklisting sites (suggestion)

### Security/Privacy Considerations

The specification explicitly mentions the following security/privacy considerations and advice:

- localStorage security

    - Whenever a script accesses a storage object returned by the localStorage attribute and the script's effective origin differs from the origin of the document of the window on which the localStorage object is accessed, an error is thrown
    - This means that after using document.domain, a script can no longer access previously obtained storage objects
- Privacy: user tracking
    - Block third party storage: do not allow a document in an iframe access to the local storage of that origin
    - Expiring stored data: expire data after a fixed amount of time (makes the API less usable/valuable)
    - Treat persistent storage as cookies: from a user perspective, local storage is treated the same as cookies (e.g. erasing on user demand)
    - Site-specific whitelisting: allow access to session storage by default, but require user authorization for accessing local storage
    - Origin-tracked storage: store the origins of sites that contained third-party content that stored data. This eases pruning by the user
    - Shared blacklists: use community forces to blacklist tracking origins
- Sensitive data
    - User agents should process stored data as potentially sensitive
    - When data is deleted, it should promptly be removed from the underlying storage
- Security
    - DNS spoofing attacks can occur, but this can be mitigated using TLS
    - Cross-directory attacks: sites using shared hosting (e.g. geocities.com) all share the same storage areas, since they are hosted on the same origin. Such sites should not use this functionality.
- Implementation risks: user agents should take extra care to avoid one origin to read data from another origin, when it's not supposed to

The following considerations are assumed by the specification and are derived from its contents:

- Stored data is only available to the origin that stored it
- The effective script origin is sufficient to distinguish all origin changes (e.g. document.domain, sandbox, ...)]
- Every origin automatically has access to storage functionality, without explicit permissions

## Threat Enumeration

### SQ1 - Well defined / Secure

***STORAGE-SECURE-1.Subdomain Storage*** The specification mentions that using subdomain storage could be used to circumvent storage limits. User agents are expected to prevent this, but no practical guidelines are given. How can legitimate use be distinguished from abuse?

***STORAGE-SECURE-2.Client-side Validation*** The specification does not mention the need for client-side validation. Since data is stored locally, but is accessible to all scripts loaded in within the document (also third-party scripts included with a `script` tag), the stored content should not be trusted, and therefore should be validated on every use.

***STORAGE-SECURE-3.Shared Frame Storage*** The goal of the specification is to support multiple transactions in separate windows, a case that cookies fail to support. This can be achieved using session storage, although this storage area is still shared across frames in the same top-level browsing context. This case however is not documented.

**SQ2 - Isolation Properties**

***STORAGE-ISOLATION-2.Restricted Context*** The specification does not mention the behaviour of storage in restricted contexts.

**SQ3 - Consistency**

***STORAGE-CONSISTENCY-1.Permission System*** The specification does not mention an explicit permission system. However, one of the steps in an algorithm is that the user agent may throw an error if certain storage policies apply.

## 4.12. Geo-location API

The geo-location API provides scripted access to geographical location information associated with the hosting device.

### Description

The geo-location API offers access to the current location of the device. Geo-location offers both "one-shot" location information or repeated monitoring of location changes. When monitoring location changes, the user agent determines when a location has changed significantly and only then sends an event. When requesting the location, the option `enableHighAccuracy` indicates the best possible result is needed, even if it is slower or consumes more power. The API also explicitly offers access to cached locations, configurable by specifying a `maximumAge`.

The API is agnostic of the underlying location mechanism (e.g. GPS, network-based) and provides no guarantees about the accuracy of the location. Since the underlying mechanism is not defined, geo-location is applicable to handhelds, notebooks, netbooks, desktop machines, ...

### Capabilities

The specification enables the following capabilities:

- one-time permitted access to the current location, either as one-shot or as a monitor (one-time permission)
- continuous permitted access to the current location, either as one-shot or as a monitor (stored permission)
- retrieval of a previously cached location, which is potentially entirely different from the current location

### User Involvement

The specification involves the user in the following way:

- give consent for an origin to access location information
- manage long-term permissions

### Security/Privacy Considerations

The specification explicitly mentions the following security/privacy considerations:

- Accessing the location requires explicit consent, where the UI must show the URI of the document's origin. The "Permissions for Device API Access" specification states that the user's permission has to be explicitly asked, but can be retained for later use in the same session.
- Privacy considerations
    - No location information can be disclosed without explicit user permission
    - The permission UI must show the URI of the document's origin
    - Long-term permissions must be easily revocable
    - Permission is not needed for pre-established trust relationships (e.g. emergency 911 services)
    - Recipients
        - must only request location information when necessary
        - must only use it for the task they requested it for
        - Must dispose of the info after using it
        - Should allow users to update/delete stored information
        - Must not retransmit the information
        - Must disclose purpose, storage terms, security measures, …
    - Implementers are advised to add location-sharing-awareness and provide easy permission-revocation mechanisms

The following considerations are assumed by the specification and are derived from its contents:

- All sites within an origin have the same trust level
- Once a persistent permission is granted to an origin, it remains valid until the user revokes it
- Recipients of location data are honest
- The callback handler and its script environment are not compromised
- The user can not be tricked into giving consent to access location data
- The user unambiguously knows which (document) origin and which part of the page wants to know the location
- An implementer is fit to determine when a location is significantly different
- There are no conflicts between the API and the underlying OS (e.g. access control to location information)

## Threat Enumeration

### SQ1 - Well defined / Secure

***GEOLOC-SECURE-1.Monitoring Lifetime*** The specification discusses a way to launch a background monitoring process, that invokes a callback handler if the location has changed. It does not explicitly specify the lifetime of a `watchPosition` process, except when it is cancelled by the caller. Such a process should terminate when the associated document no longer exists.

***GEOLOC-SECURE-2.Technology Agnostic*** The specification is agnostic to the underlying location technology. In practice, location providers such as Google Location Services or SkyHook Wireless are used to determine the location of the device. By doing so, these services can -- and do -- store the requested information, leading to potential privacy issues [18].

***GEOLOC-SECURE-3.Cache Polling*** The specification supports the use of cached positions and even supports the explicit retrieval of a cached position, instead of a fresh one. By querying the cache with increasingly larger `maximumAge` values until a hit occurs, the timestamp of the last cached location can be determined, which is a potential privacy issue.

***GEOLOC-SECURE-4.Changed Behaviour*** The specification allows that permissions are stored for a certain origin, avoiding prompting the user every time the site requests location information. The specification does not take into account that the origin's intent/permission usage can change over time, thus using the location information for completely other purposes than before. Even if the site would like the user to re-confirm the permission, there is no mechanism to do so [18].

***GEOLOC-SECURE-5.OS Conflicts*** The specification employs a separate permission system for accessing location information. There is no mention about potential integration with the underlying OS. At least, the OS should be aware of the availability of more fine-grained controls for accessing location information. Additionally, having a uniform access control model and UI avoids potential confusion about location-sharing decisions.

### SQ2 - Isolation Properties

***GEOLOC-ISOLATION-1.Restricted Context*** The interactions of permissions and restricted contexts are underspecified. Should permissions granted in the normal context also be valid in a private browsing context and vice versa? How are permissions granted in a sandboxed context with a unique origin?

### SQ3 - Consistency

***GEOLOC-CONSISTENCY-1.Permission Management*** Long-term permission management is only specified as a "*should*", leaving too much room for implementers. For example, the statement that "implementors are advised to enable user awareness of location sharing, and to provide easy access to interfaces that enable revocation of permissions" is not sufficient. In addition, the lack of awareness to the end-user that geo-location is being used complicates long-term permission management. This is evidenced by current implementations:

- Firefox: this setting is in the page information dialog, which is only accessible after loading the page (and thus resubmitting location information)

- Chrome: this setting is located in preferences -> under the hood -> privacy content settings -> location, where exceptions can also be managed
- Opera: same as Firefox
- IE9: only allows clearing of the entire list of sites. Moreover, the list cannot be consulted. This is located under internet options -> privacy.

**SQ4 - User Involvement**

***GEOLOC-USER-1.Permission Nature*** The specification imposes a requirement on the permission UI, stating that the origin of the document must be shown. However, it does not impose that the nature of the permission (one-shot or monitoring) must be made clear. The difference between permission for a one-shot location retrieval or launching a monitoring process is quite important. Additionally, continuous one-shot permissions are very similar to the monitoring process.

## 4.13. Media Capture API

The Media Capture API provides programmatic access to audio, image and video capabilities of the device.

### Description

Scripts can use the Capture interface to capture audio clips, video clips and images. Capture operations are invoked asynchronously and when finished, they return their result using a success or error callback handler. The actual capturing is handled by an external application that is part of the browser or the underlying platform. Captured data is available as a `FileList`, as defined in the File API [33]. The lifetime of captured data is equal to the lifetime of the top-level browsing context, unless files are removed due to space constraints.

The API allows the specification of configuration data for capturing images/audio/video (e.g. number of items, duration of clips). The API does not allow live capturing, as needed in interactive applications (e.g.video chat).

### Capabilities

The specification enables the following capabilities:

- capturing audio/visual information
- providing access to files as specified by the File API [33]

### User Involvement

The specification involves the user in the following way:

- give consent to start a capture

### Security/Privacy Considerations

The specification explicitly mentions the following security/privacy considerations:

- The specification mentions the importance of privacy and states that capturing cannot happen without the user's consent. There are no details on how to implement these consent mechanisms.

The following considerations are assumed by the specification and are derived from its contents:

- the user cannot be tricked into consenting with a capture operation
- the user is aware when a capture operation takes place
- the user unambiguously knows which (document) origin and which part of the page wants to capture data
- the callback handler and its script environment are not compromised

### Threat Enumeration

SQ3 - Consistency

***MEDIACAPTURE-CONSISTENCY-1.Requesting Consent*** The specification does not mention when exactly user consent is required. The related HTML Media Capture specification does mention that consent is required every time a capture is started. This is an underspecification in this API.

***MEDIACAPTURE-CONSISTENCY-2.Consent UI Content*** The specification mentions that user consent is needed before any data can be captured. It does not impose any requirements on the consent mechanism. One important item for the user is to know which origin requests the capture (since it can come from a hidden or embedded iframe). Additionally, the UI should indicate if the requesting document is sandboxed or not.

***MEDIACAPTURE-CONSISTENCY-3.Consent UI Security*** The specification does not mention any details about the implementation of the consent UI. The related HTML Media Capture specification does mention that the UI should be created with an HTML element under control of the user agent. The UI should also be protected against UI re-dressing attacks.

## 4.14. System Information API

The System Information API provides web applications with access to various properties of the system on which they are running.

### Description

The API offers programmatic access to all kinds of system properties typically available to an operating system. Examples are CPU properties, audio/video codecs or different sensors (ambient light, ambient noise, temperature, ...). The API offers one-shot access and continued monitoring, which can be canceled by the user. Currently, only reading information is supported, but a future version may offer write-support.

### Capabilities

The specification enables the following capabilities:

- one-time permitted access to all kinds of system information, either as one-shot or as a monitor (one-time permission)

- continuous permitted access to all kinds of system information, either as one-shot or as a monitor (stored permission)

### User Involvement

The specification involves the user in the following way:

- give consent for an origin to access a specific group of properties
- manage long-term permissions (both for origins and for nested origins)

### Security/Privacy Considerations

The specification explicitly mentions the following security/privacy considerations:

- No information is retrievable without the user's explicit permission
- Permission must be acquired through a user interface, unless a prearranged relationship exists (e.g. via a widget)
- The user interface must include the origin of the document in whose context the callback will be invoked
- Long-term permissions must be revocable
- Permissions can be granted for groups of properties, but they only apply to that specific property group
- The proposed groups are: power, CPU, thermal, audio/video codecs, storage, output devices, input devices, sensors and network.
- Separate permissions are required if the callback is executed in a nested context and its origin is different than the top-level context's origin
- Such permission is scoped to the pair of both origins
- For the monitor method, distinct permission must be granted and the user interface must explicitly indicate the continuous monitoring nature
- The monitor method must be limited in time by the user agent

The following considerations are assumed by the specification and are derived from its contents:

- the principle of least privilege is applied through the permission system
- the user can not be tricked into giving permissions
- the callback handler and its execution environment are not compromised
- the user understands why an application might need such permissions
- there are no conflicts between the API and the underlying OS (e.g. access control to device information)

### Threat Enumeration

**SQ1 - Well defined / Secure**

***SYSINFO-SECURE-1.Monitoring Lifetime*** The specification discusses a way to launch a background monitoring process, that invokes a callback handler if the location has changed. It briefly mentions that there is a maximum lifetime. The specification does not

provide a concrete value for the "maximum lifetime", nor does it provide any requirements for the lifetime of a monitor process. For instance, such a process should also terminate when the associated document no longer exists.

***SYSINFO-SECURE-2.Changed Behavior*** The specification allows permissions to be stored for a certain origin, avoiding prompting the user every time the site requests system information. The specification does not take into account that the origin's intent/permission usage can change over time, thus using the system information for completely other purposes than before. Even if the site would like the user to re-confirm the permission, there is no mechanism to do so [18].

***SYSINFO-SECURE-3.OS Conflicts*** The specification employs a separate permission system for accessing device information. There is no mention of potential integration with the underlying OS. At least, the OS should be aware of the availability of more fine-grained controls for accessing this information. Additionally, having a uniform (or co-ordinated) access control model and UI would avoid potential confusion about information-sharing decisions.

**SQ2 - Isolation Properties**

***SYSINFO-ISOLATION-1.Restricted Context*** The interactions of permissions and restricted contexts are underspecified. Should permissions be granted in the normal context also be valid in a private browsing context and vice versa? How are permissions granted in a sandboxed context with a unique origin?

**SQ3 - Consistency**

***SYSINFO-CONSISTENCY-1.Permission Management*** Long-term permission management is only implied, leaving too much room for implementers.

**SQ4 - User Involvement**

***SYSINFO-USER-1.Permission Nature*** Even though clarifying the difference between one-shot and monitoring access would be an improvement, the practical difference between a stored one-shot permission and a monitoring permission is very small.

***SYSINFO-USER-2.Awareness*** The specification does not specify or suggest that browsers should make users aware that a page is accessing system information.

### 4.15. Widgets - Digital Signatures

This specification defines a profile of the general XML Signature Syntax and Processing specification, specifically aimed at widgets.

#### Description

The specification defines two types of signatures: author signatures and distributor signatures. Both types are identified by their associated roles: an author role or a distributor role. A signature is based on a list files in the widget package. For each file, a canonicalization method can be specified.

***Author Signature***: This type of signature covers all non-signature files in the package and is meant to be generated by the author and can be used to determine (i) which entity has authored the widget, (ii) that the integrity of the widget is as the author intended and (iii)

whether two widgets came from the same author. The filename for the author signature is author-signature.xml.

***Distributor Signature***: This type of signature covers all files in the package, except other distributor signatures, and is meant to be generated by a distributor of the widget. The signature can be used to determine (i) that a particular distributor has distributed a widget package and (ii) that the integrity of the widget package is as the distributor intended. The distributor signature includes all files in the package, including the author signature but excluding other distributor signatures. Multiple distributors can add their signature to the same package. The filename convention is signature[X].xml, where [X] is a positive integer without leading zeroes.

The specification includes a recommended signature algorithm (RSAwithSHA256), a minimum key length (2048bit for RSA/DSA) and recommended key length (4096bit for RSA/DSA). The recommended digest algorithm is SHA256.

### Capabilities

The specification enables the following capabilities:

- Sign widget packages to protect their contents
- Detect unauthorized changes to a widget package

### User Involvement

The specification involves the user in the following way:

- Be able to manage public key certificates
- Handle a security warning or error (e.g. for a self-signed cert)

### Security/Privacy Considerations

The specification explicitly mentions the following security/privacy considerations:

- Because the signature is based on packaged content, the attack surface of the decompression and unpacking code still remains
- Denial of Service attacks need to be taken into account when validating signatures (e.g. malformed packages)
- Path components need to be handled carefully, since an attacker might try to point to sensitive operating system files
- Signatures can be attacked by removing a distributor signature or removing all signatures, including the author signature
- The management (and installation) of root certificates needs to be handled with care, since it is the basis of signature validation

The following considerations are assumed by the specification and are derived from its contents:

- Unsigned packages are hard to install
- Users will not ignore security advice/constraints

### Threat Enumeration

**SQ4 - User Involvement**

***WGDIGSIG-USER-1.Certificate Management*** The specification depends on signature validation, which involves X509 certificates. As with all security technologies depending on this infrastructure, the validation mechanism heavily depends on the user's behavior. Unless the user knows how certificates and their validation works, it will be very difficult to provide meaningful integrity protection.

***WGDIGSIG-USER-2.Unsigned/Unchecked Packages*** To boost the usage of digital signatures, the specification should make the installation of unsigned or unvalidated packages significantly harder than signed/checked packages.

## 4.16. Widgets - Access Request Policy

This specification defines the security model controlling network access from within a widget.

### Description

The goal of this specification is to allow widgets to include external resources. The widget platform should not become less powerful than the HTTP web platform, which is why the access policy should not be too restrictive.

The default policy for network resource access is deny all. This policy can be relaxed by means of the access-request list, to which the user agent should provide access. The user agent is allowed to deny access to network resources, even if they are listed in the access-request list (e.g. custom policy, user prompting, ...). Additionally, exceptions are allowed for specific schemes, if they are considered benign (e.g. `mailto`).

An element of the access-request list (the access element) lists an origin to which access is allowed. By using the subdomains boolean value, access to subdomains of the host can be easily enabled. An access element can also use a wildcard (`*`) to enable universal network access.

### Capabilities

The specification enables the following capabilities:

- Customize the default deny all policy by allowing access to specific hosts (and its subdomains)
- Completely open the Deny all policy by using a wildcard in the access element

### User Involvement

The specification involves the user in the following way:

- Allow/disallow certain accesses (very vague suggestion)

### Security/privacy Considerations

The specification does not explicitly mention security/privacy considerations.

The following considerations are assumed by the specification and are derived from its contents:

- An access policy on the level of origins offers sufficient security
- There are no conflicts between the API and the underlying OS

### Threat Enumeration

**SQ1 - Well defined / Secure**

***WARP-SECURE-1.Coarse-Grained Approach*** Allowing the refinement of the deny-all policy can work quite well. The level of refinement however (an origin) is too coarse grained, especially if the goal is to allow the loading of external resources.

***WARP-SECURE-2.Wildcard*** By allowing an easy wildcard, the specification risks that this will become the default policy. A similar effect occurred with the Adobe crossdomain.xml policy file. In CORS, for example, the wildcard can only be used in some cases.

***WARP-SECURE-3.OS Conflicts*** The specification employs a separate permission system for widget network resource access. There is no mention about potential integration with the underlying OS. At least, the OS should be aware of the availability of more fine-grained controls for network access. Additionally, having a uniform/consistent access control model and UI would avoid potential confusion about network resource access decisions.

**SQ4 - User Involvement**

***WARP-USER-1.User's Role*** It is unclear how the user is involved in granting permissions. Does the user have to agree on a set of permissions when installing the widget? Does this agreement process make the user aware of the dangers of a wildcard policy?

# References

[1] W3C. HTML5, A vocabulary and associated APIs for HTML and XHTML, Editor's Draft 25 March 2011. http://dev.w3.org/html5/spec/Overview.html

[2] W3C. XMLHttpRequest, Editor's Draft 4 March 2011. http://dev.w3.org/2006/webapi/XMLHttpRequest/

[3] W3C. XMLHttpRequest Level 2, Editor's Draft 4 March 2011. http://dev.w3.org/2006/webapi/XMLHttpRequest-2/

[4] W3C. Uniform Messaging Policy, Level One, Editor's Draft 15 June 2010. http://dev.w3.org/2006/waf/UMP/

[5] W3C. Cross-Origin Resource Sharing, Editor's Draft 17 November 2010. http://dev.w3.org/2006/waf/access-control/

[6] W3C. HTML5 Web Messaging, Editor's Draft 4 March 2011. http://dev.w3.org/html5/postmsg/#channel-messaging

[7] W3C. The Media Capture API, W3C Editor's Draft 02 December 2010. http://dev.w3.org/2009/dap/camera/Overview-API

[8] W3C. The System Information API, W3C Editor's Draft 16 March 2011. http://dev.w3.org/2009/dap/system-info/

[9] W3C. Permissions for Device API Access, W3C Editor's Draft 30 September 2010. http://dev.w3.org/2009/dap/api-perms/

[10] W3C. Device API Privacy Requirements, W3C Editor's Draft 23 June 2010. http://dev.w3.org/2009/dap/privacy-reqs/

[11] W3C. Web Storage, Editor's Draft 28 February 2011. http://dev.w3.org/html5/webstorage/

[12] W3C. Geo-location API Specification, Editor's Draft 10 February 2010. http://dev.w3.org/geo/api/spec-source.html

[13] W3C. Widget Access Request Policy, W3C Candidate Recommendation 20 April 2010. http://dev.w3.org/2006/waf/widgets-access/

[14] W3C. XML Digital Signatures for Widgets, W3C Working Draft 18 March 2011. http://dev.w3.org/2006/waf/widgets-digsig/

[15] Mario Heiderich. HTML5 Security Cheatsheet Project. http://code.google.com/p/html5security/

[16] W3C. HTML5 differences from HTML4, W3C Working Draft 25 May 2011. http://www.w3.org/TR/html5-diff/

[17] Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson. Busting frame busting: a study of click-jacking vulnerabilities at popular sites. IEEE Oakland Web 2.0 Security and Privacy (W2SP 2010).

[18] Nick Doty, Deirdre K. Mulligan and Erik Wilde. Privacy Issues of the W3C Geo-location API. UC Berkeley School of Information Report 2010-038, February 2010. Available at http://escholarship.org/uc/item/0rp834wf

[19] Leo Meyerovich, Benjamin Livshits. ConScript: Specifying and Enforcing Fine-Grained Security Policies for JavaScript in the Browser. Proceedings of the IEEE Symposium on Security and Privacy, IEEE, 20 May 2010.

[20] Phu H. Phung, David Sands, Andrey Chudnov. Lightweight Self-Protecting Javascript. ACM Symposium on Information, Computer and Communications Security (ASIACCS 2009), Sydney, Australia, March 2009. ACM Press. pp. 47-60.

[21] Jonas Magazinius, Phu H. Phung, David Sands. Safe Wrappers and Sane Policies for Self Protecting JavaScript. The 15th Nordic Conf. in Secure IT Systems (NordSec 2010), Springer Verlag, 2011.

[22] Steven Van Acker, Philippe De Ryck, Lieven Desmet, Frank Piessens, Wouter Joosen. WebJail: Least-privilege Integration of Third-party Components in Web Mashups.

[23] M. S. Miller, M. Samuel, B. Laurie, I. Awad, M. Stay. Caja Safe active content in sanitized JavaScript. October, 2008.

[24] Adam Barth, Collin Jackson, John C. Mitchell. Securing frame communication in browsers. Communications of the ACM 52, 6 (June 2009), pp. 83-91

[25] Devdatta Akhawe, Adam Barth, Peifung Eric Lam, John Mitchell, Dawn Song. Towards a Formal Foundation of Web Security. Proceedings of the 23rd IEEE Computer Security Foundations Symposium, Edinburgh 2010.

[26] Hristo Bojinov, Elie Bursztein, Dan Boneh. XCS: cross channel scripting and its impact on Web applications. Proceedings of the 16th ACM Conference on Computer and Communications Security (New York, NY, USA, 2009), ACM, 420–431.

[27] Mozilla Developer Network. The X-Frame-Options response header. https://developer.mozilla.org/en/the_x-frame-options_response_header

[28] Brandon Sterne, Mozilla. Content Security Policy. http://people.mozilla.com/~bsterne/content-security-policy/

[29] W3C. Feature Permissions, W3C Editor's Draft 31 May 2011. http://dev.w3.org/2009/dap/perms/FeaturePermissions.html

[30] Mike Perry. TorButton Design Documentation. https://www.torproject.org/torbutton/en/design/index.html.en

[31] J. Mayer, A. Narayanan, S. Stamm. Do Not Track: A Universal Third-Party Web Tracking Opt Out. Network Working Group, Internet-Draft, March 7, 2011.

[32] W3C. Indexed Database API, W3C Working Draft 19 April 2011. http://www.w3.org/TR/IndexedDB/

[33] W3C. File API, W3C Working Draft 26 October 2010. http://www.w3.org/TR/FileAPI/