HTML5 SECURITY
THE MODERN WEB BROWSER PERSPECTIVE

Doug DePerry – doug@isecpartners.com

iSEC Partners, Inc
123 Mission Street, Suite 1020
San Francisco, CA 94105
https://www.isecpartners.com

December 4, 2012

Abstract

The purpose of this paper is to serve as a current analysis of HTML5 on modern web browsers and mobile platforms and as a reference for related testing methodologies. A select number of new features introduced by HTML5 are examined and their security implications discussed; but this is not an exhaustive survey. It is meant to raise awareness of the risks or shortfalls of HTML5 as implemented by different browsers and environments among average users as well as enterprise administrators and policy implementers.

1 INTRODUCTION

Many aspects of technology evolve quickly. This is true for hardware as well as software and Application Programming Interfaces (APIs). The HTML5 specification is not a single specification but a collection of assorted, loosely related specifications covering a host of different technologies. This makes it difficult, if not impossible, to create a definitive analysis of the standard until it is complete. Additionally, each major web browser or mobile platform decides which set of specifications it will implement, making this topic a moving target. For an enterprise administrator to determine the compatibility of all possible web browsers and platforms within the company would be a difficult and time-consuming task.

Due to the rapid progression of HTML5, it is sometimes difficult to rely on public information as its timeliness can be questionable. There are many blog articles and websites dedicated to HTML5 that are targeted at application, web browser and game developers. There have been whitepapers written by security researchers and compatibility websites that will determine if a browser is capable of using specific HTML5 features. These are useful tools without a doubt, but if the blog article or website being used is just a few months out of date the information could be inaccurate with potential security implications. This paper is of course no exception. However, by providing a concrete testing methodology, the results presented here can be more easily verified against future specifications and browser versions.

Mobile web browsers are one of the main targets of the specification, mostly due to the appeal of HTML as a cross-platform solution. With many new platforms and devices frequently released, a standard can be beneficial to all. Game developers are increasingly interested in HTML5 for its cross-platform capability and ability to provide a rich experience without native code support. For the purposes of this study an effort was made to test against the most popular mobile platforms to cover the largest set of the population.

While HTML5 is meant to be a standard in the strictest sense of the term, application and hardware vendors are often constrained by their current platforms or the version of the specification they develop against. Oftentimes too, different developers will interpret the same specification differently. For this reason, different applications (web browsers are no exception) will often implement the same functionality in a slightly different way. This research will highlight those differences, if they exist, on the features and platforms tested.

2 A (VERY) BRIEF OVERVIEW OF HTML5

Stated simply, HTML5 is a set of new features for presenting content on the web. The need for a new standard grew partially out of the desire to move beyond the veritable hodgepodge of various specifications, browser-specific features, hacks and mashups that are the web today.

The two standards bodies involved with the specification are the Web Hypertext Application Technology Working Group (WHATWG) and the World Wide Web Consortium (W3C) HTML Working Group. The two groups are working together on the specification, which the WHATWG likes to refer to as "the latest work on HTML", and not necessarily as a specific version number[1]. The history of HTML5 and how two separate groups came to be involved can be somewhat confusing. For a light-hearted explanation, Mike Wilcox explains the history of HTML5 with a bit of humor[2].

The HTML5 specification is often described as a "Living Standard." The WHATWG FAQ[3] has the definition:

> *The WHATWG specifications are described as Living Standards. This means that they are standards that are continuously updated as they receive feedback, either from Web designers, browser vendors, tool vendors, or indeed any other interested party. It also means that new features get added to them over time, at a rate intended to keep the specifications a little ahead of the implementations but not so far ahead that the implementations give up.*

The finalized specification or "Recommendation" is tentatively scheduled for the year 2022 or later. This obviously does not mean that vendors cannot implement HTML5 features in the meantime. Many new tags such as <canvas> are considered stable enough to be used in current implementations.

Security is considered throughout the specification with many elements containing a section dedicated to the topic. While any new feature carries with it the opportunity for abuse, the specification has shown an interest in preventing major security flaws such as cross-domain abuse. Fortunately, due to the "living" nature of the specification, any flaws uncovered during testing and development can be addressed.

3 ATTACK VECTORS

Exploitation scenarios have not changed too drastically in HTML5. With some exceptions, an attacker must still rely on Cross-Site Scripting (XSS) or luring a user to a malicious website in order to execute a malicious payload. JavaScript is still the main scripting language used in HTML5 websites, so the vulnerabilities and abuses inherent to that language still apply. HTML5 does introduce new XSS vectors using some of the new tags and elements. For instance, to exploit an XSS flaw in a page without user interaction, an attacker can make use of the *autofocus* attribute. This attribute is designed to be used with elements such as text input fields. As soon as the page loads, the cursor focus will be directed to an element of the developer's choosing, such as the first element of a contact form. An attacker can use this capability to bypass user-interaction and automatically direct focus to the element containing the attacker's malicious payload. As soon as the page is loaded, the *onfocus* event is triggered, and the payload will execute. Websites that blacklist current attributes such as *onload* could be vulnerable to this new vector. The website html5sec.org contains an extensive list of these attributes along with the specific browsers that are vulnerable.

For new concepts such as Cross Origin Resource Sharing[4] and Web Sockets[5], there is still an implicit trust that the other end of the connection is not a malicious site. The specification attempts to address this weakness with same-origin protections and secure versions of these protocols[6].

Another new HTML5 element that has the possibility of abuse is the <iframe sandbox> attribute. This element works by allowing a developer to load an untrusted site within an iframe (such as an advertisement) but restrict that site's abilities within the iframe. For instance, one of the values that can be configured is *allow-scripts*. If this option is used with the <sandbox> attribute, then the potentially malicious page loaded in the iframe will not be able to run JavaScript. The downside is that many sites employ frame busting JavaScript code that prevents that site from being loaded in an iframe. If that site were to be loaded in an iframe sandbox such that it is prevented from running script, then the frame busting script will not execute and the page will be loaded in that iframe successfully.

Many security vulnerabilities are often caused by improper configuration and insecure coding. In light of this, HTML5 may be considered to expose new vulnerabilities if only because developers and administrators often make mistakes even when using technology that is decades old. Adding new sets of rules to remember and configurations to deploy may cause HTML5 implementations to be deployed insecurely.

4 TESTING METHODOLOGY

The table below lists the platforms and browser versions that were tested as part of this research. The focus was on the latest versions of three popular web browsers - Chrome, Firefox and Internet Explorer. The production versions were used as opposed to "nightly builds" or developer editions in an effort to mirror what the average user or enterprise employee would use. No browser add-ons were installed with the exception of FireBug on Firefox for debugging and monitoring purposes. On Android phones, the latest version of the operating system was tested as well as an older version. The older version running on the Nexus S was chosen since Android version 2.3.6 is one of the most widespread versions deployed to date[7].

The various OS versions were tested due to the sometimes subtle differences within the same browser version running on different platforms. All operating systems were tested in Virtual Machines on a MacBook Pro running VMware Fusion 4.1.3. Testing was completed in July of 2012. Although newer versions of these browsers have been released since testing was completed, these results can be used to guide policy and hardening guidelines for your organization.

Platform	Browser Version
OSX 10.7.3 (64-bit) CPU: Intel Core I7 RAM: 8GB	Chrome 19 Firefox 13 Safari 5.1.5
Ubuntu Linux 10.04 (64-bit) VM CPU: dual-core RAM: 2GB	Chrome 19 Firefox 13
Windows 7 (64-bit) VM CPU: single-core RAM: 2.5GB	Chrome 19 Firefox 13 Internet Explorer 9
Windows 8 Release Preview (32-bit) VM CPU: dual-core RAM: 1.5GB	Internet Explorer 10
iPhone 4 iOS 5.1.1	Safari
Galaxy Nexus Android 4.0.4	Browser Chrome Beta 0.18.4531.3636
Nexus S Android 2.3.6	Browser

Table 1 - Test Environment

The Google Chrome and Safari web browsers use the WebKit rendering engine, so it is expected that the two browsers will behave in a somewhat similar fashion. Mozilla Firefox uses the Gecko engine while Microsoft's Internet Explorer uses Trident. All browsers offer different levels of HTML5 feature support.

Testing was performed in a variety of ways – utilizing various websites that provide HTML5 examples or compatibility tests as well as custom, purpose-written code.

5 HTML5 FEATURE DETAIL

5.1 WEB STORAGE

The Web Storage feature encompasses two distinct types of local browser storage. Although WebSQL and IndexedDB are referred to under the Web Storage umbrella, they are separate technologies with their own specifications. These storage techniques will be discussed in subsequent sections.

Web Storage defines the *sessionStorage* and *localStorage* attributes as means of storing data in a client's browser, similar to cookies. The specification attempts to fix some of the failings of cookies and also adds new concepts to enhance usability.

The Web Storage specification summarizes the basic properties as follows:

- User agents should limit the total amount of space allowed for storage areas.
- User agents should prevent sites storing data under the origins of other subdomains, e.g. storing up to the limit in *a1.example.com*, *a2.example.com*, *a3.example.com*, etc., circumventing the main *example.com* storage limit.
- User agents may prompt the user when quotas are reached, allowing the user to grant a site more space. This enables sites to store many user-created documents on the user's computer, for instance.
- User agents should allow users to see how much space each domain is using.
- A mostly arbitrary limit of five megabytes per origin is recommended. Implementation feedback is welcome and will be used to update this suggestion in the future.

The data is stored as a string in a key=value pair. Anything that can be serialized or converted to a string can be stored, such as the base64 representation of an image. As mentioned in the third bullet point above, the specification states that browsers may prompt a user to increase storage, although none of the browsers tested currently provide this functionality. The same-origin policy[8] applies; each domain has its own separate Local Storage area that cannot be accessed by subdomains or completely different domains. Additionally, unlike browser cookies, there are no path restrictions; Web Storage is site-wide.

5.1.1 SECURITY IN THE SPEC

The specification mentions a number of security concerns. Same-origin security considerations are maintained, as web browser user agents are required to throw security exceptions on same-origin violations. A script is not allowed to access Local Storage set by a script with a differing origin. Because of the potential for DNS spoofing attacks, pages can use TLS to ensure that same-domain restrictions are upheld on storage areas. Cross-directory attacks are a potential issue, since Web Storage cannot be restricted to a specific path. Multiple websites sharing a single domain will share the same Web Storage container; developers of sites where this is the case are discouraged from using Web Storage.

5.1.2 TESTING DETAIL

Session Storage, as the name implies, is only valid for the session - as long as the browser window or tab is open. Any values stored are deleted once the session ends. Session Storage access is limited to the tab or window that created it. In some situations, it may be possible for Session Storage to persist if the web browser utilizes a session resume/restore feature.

Local Storage is more permanent than Session Storage, persisting indefinitely until it is removed by the site or by the user. Local Storage is available to any window or tab open on the same domain.

Each browser implements different limits for the amount of Session Storage available to a domain, described in the table below.

Browser	Platform	Session Storage	Local Storage
Chrome	All	5MB	5MB
Firefox	All	unlimited	5MB*
Safari	OSX	Unlimited	5MB
	iPhone	5MB	5MB
Internet Explorer 9	Win7	4.75MB	4.75MB*
Internet Explorer 10	Win8	4.75MB	4.75MB*
Android Browser	All	unlimited	5MB

Table 2 - Browser Web Storage

* Firefox and Internet Explorer store characters as UTF-8 as opposed to UTF-16. Depending on the character encoding, it may be possible to store more characters in these browsers.

5.1.3 ABUSE

Because Session Storage is stored in the browser process memory, it may be possible to perform Denial of Service (DoS) resource exhaustion attacks if there is no storage limit. This attack was confirmed with Firefox running in Ubuntu and Win7 VMs as well as the Android Browser on both devices. The Chrome, Safari and IE web browsers were not found to be vulnerable. The resource exhaustion can be accomplished with a script that attempts to insert ~1GB of data into Session Storage. The script will continue to run and eat up system memory until it causes the browser to crash.

A JavaScript file was created with a variable containing a one-megabyte[i] string. This variable was then inserted into Session Storage via a loop running a number of times. Setting the loop to 1,000 iterations will cause Firefox to crash in the Ubuntu VM. An instance of Chrome that was also running (though not actively being used) also throws the out-of-memory "He's dead, Jim!" error. Overall system stability does not seem to be compromised, as after Firefox crashes the memory is released.

It can take more than 1,000,000 loop iterations to cause the same crash on the Win7 VM[ii]. This attack is likely dependent on the amount of memory available to the system.

Because Session Storage is an attribute of the Window object, it is accessible via the DOM, which opens up the data to compromise by XSS. For this reason it is not recommended to store sensitive or session data in Local Storage. It is especially important to not store JSON or JavaScript data using this feature. If a site were to store script files in Local Storage the script could be accessed and even edited via XSS. This could create a persistent, DOM-based XSS condition, compromising every user that visits the site.

The security implications of Web Storage are unique and are bound to become more prevalent as browser and developer adoption increases.

[i] Approximately 1034551-characters = .98MB

[ii] While demonstrated multiple times, this attack was inconsistent on Win7 and did not always cause a browser crash.

5.1.4 MITIGATION

Different browsers all have different ways to disable Local Storage. Unfortunately, for some browsers there is a tradeoff. Firefox and Chrome for Android require the user to prevent cookies from being set by webpages in order to disable Local Storage. Many sites will not function properly without cookies, so the ability to view and utilize certain websites will be severely hampered if Local Storage is disabled. For Safari on the desktop and iPhone, Private Browsing must be turned on in order to prevent Local Storage. The Android browser doesn't seem to have any way to prevent Local Storage, short of turning off JavaScript completely.

5.1.4.1 DISABLING LOCAL AND SESSION STORAGE

The table below describes how to *disable* Local and Session Storage in the various browsers tested.

Browser	Configuration Steps
Chrome Desktop	Settings→Advanced→Privacy→Content settings→Block sites from setting any data
Chrome Android	Settings→Content settings→Accept cookies *(uncheck)*
Firefox	Preferences→Privacy tab→ Accept cookies from sites *(uncheck)*
Safari Desktop	Safari→Private Browsing
Safari iPhone	Settings→Safari→Private Browsing→on
Internet Explorer 9/10	Internet Options→Advanced tab→Security heading→enable DOM storage *(uncheck)*
Android Browser	*There does not seem to be any setting for preventing Local Storage. Turning off cookies does not work. Turning off JavaScript is the only option.*

Table 3 - Disabling Web Storage

5.1.4.2 CLEARING LOCAL STORAGE

The table below describes how to *clear* Local Storage data in the various browsers tested.

Browser	Configuration Steps
Chrome Desktop	Settings→Advanced→Privacy→Clear Browsing Data→Delete cookies and other site and plug-in data
Chrome Android	Settings→Privacy→Clear Browsing Data
Firefox	Tools→Clear Recent History→Cookies *This will only clear storage when time range is "Everything"*
Safari Desktop	Safari→Reset Safari→Remove all website data
Safari iPhone	Settings→Safari→Clear Cookies and Data
Internet Explorer 9	Tools→Delete Browsing History→Cookies *This will clear site data for all sites that are not marked as Favorites. To include every site, uncheck the Preserve Favorite Site Data.*
Internet Explorer 10	Internet Options→General Tab→Browsing History→Delete→Cookies and website data *Note 'Favorite' caveat as for IE9*
Android Browser	Settings→Privacy & security→Clear cache

Table 4 - Clearing Local Storage

5.2 WEB SOCKETS

Web Sockets typically get a lot of attention due to the novelty of the feature. The Web Sockets API provides an alternative to HTTP polling for two-way communication from a web page to a remote server[9]. Without the typical overhead involved with XMLHttpRequests (Web Socket packets are framed with just two bytes) a web browser can create a bi-directional communication path with a remote server. This connection is left open for either side to transmit and/or receive. Real-time applications such as stock tickers and games are typical examples. Web Sockets perform a handshake over HTTP followed by basic message framing layered over TCP, so they cannot be used to connect to arbitrary ports or provide functionality like nmap[10] to scan for open ports and services.

The Web Socket protocol is designed to run over the standard HTTP/S ports (80/443) so that there is minimal disruption to existing infrastructure. If the protocol used non-standard ports, it is likely that firewalls and IDS devices would need to be reconfigured to prevent blocking the communication path. The Web Socket protocol can be understood as a very simple and non-disruptive way to expose a raw TCP connection over the web.

The Web Socket connection is created via a brief handshake process initiated by the client web browser. The client sends a request to the server requesting to "upgrade" to a Web Socket connection:

GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13

The server will respond to the handshake request if it supports Web Sockets:

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat

The previous HTTP connection is now replaced by a Web Socket connection, allowing full-duplex data frames to be passed between client and server.

5.2.1 SECURITY IN THE SPEC

The same origin model applies as usual; the Web Socket server must validate a request's origin as presented in the initial Web Socket request. If it is not valid, then the request should be rejected. Of course, this relies on the fact that the server is properly configured to only allow requests from specific domains. If the Web Socket connection transmits any kind of sensitive data, a secure communication path (protected by TLS) should be created.

The server is required to prove that it read the handshake, which it can only do if the handshake contains the appropriate parts. For instance, currently, fields starting with |*Sec-*| cannot be sent by an attacker from a web browser using only HTML and JavaScript APIs such as XMLHttpRequest[11].

To prove that the handshake was received, the server has to take two pieces of information and combine them to form a response. The first piece of information comes from the |*Sec-WebSocket-Key*| header field in the client handshake, *Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==*.

For this header field, the server has to take the value as present in the header field (the base64-encoded string minus any leading and trailing whitespace) and concatenate this with a specific Globally Unique

Identifier[12] *258EAFA5-E914-47DA-95CA-C5AB0DC85B1.* This concatenation is then hashed using SHA-1, base64-encoded, and returned in the server's handshake. The GUID is used because it is unlikely to be used by network endpoints that do not understand the WebSocket Protocol.

As a complete example, if the |*Sec-WebSocket-Key*| header field had the value *dGhlIHNhbXBsZSBub25jZQ==*, the server would concatenate the string *258EAFA5-E914-47DA-95CA-C5AB0DC85B11* to form the string *dGhlIHNhbXBsZSBub25jZQ==258EAFA5-E914-47DA-95CA-C5AB0DC85B11*. The server then takes the SHA-1 hash of this and base-64 encodes it to get the value *s3pPLMBiTxaQ9kYGzzhZRbK+xOo=*. This value is then echoed in the |*Sec-WebSocket-Accept*| header field.

5.2.2 TESTING DETAIL

There are multiple Web Socket server implementations in a variety of languages including .NET, Java, C++, Python and Ruby. Many include client libraries as well. For a comparison of various WebSocket server implementations see http://en.wikipedia.org/wiki/Comparison_of_WebSocket_implementations.

The table below describes the browsers and platforms tested for Web Socket support. Out of these, the only browsers immune to abuse are IE9 and the Android Browser because the protocol is not supported.

Browser	Platform	Web Socket Support
Chrome	All	Supported
Firefox	All	Supported
Safari	All	Supported
Internet Explorer 9	Win7	Not Supported
Internet Explorer 10	Win8	Supported
Android Browser	All	Not supported

Table 5 - Browser Web Socket Support

5.2.3 ABUSE

Web Socket vulnerabilities are likely to be centered on the server implementation. If an attacker is able to execute code due to an injection flaw, they may be able to log or intercept messages or perform DoS attacks on the server.

Perhaps more importantly, Web Sockets may be able to be abused to exfiltrate data from a user's machine. A Web Socket client does not necessarily have to be in the form of a web browser, it could be a standalone application. If a determined attacker is able to install a malicious program on a user's computer, a Web Socket connection could be used to send files to an attacker-controlled Web Socket server. Because the traffic would appear on standard HTTP ports, it could possibly bypass infrastructure firewalls and intrusion detection systems.

The JS-Recon[13] tool from Attack & Defense Labs[14] is an excellent example of using Web Sockets for malicious purposes. The tool is used for scanning networks by timing the status transition of port connections made by Web Sockets and/or Cross Origin Resource Sharing (CORS) connections. Using this information it is possible to determine whether the scanned port is open or closed. The tool does have some limitations, and practical testing showed mixed success. Still, JS-Recon serves to demonstrate the potential influence Web Sockets can have on the web.

5.2.4 MITIGATION

If an organization wishes to stand up a Web Socket server, it should ensure that it is properly configured, especially against cross-domain abuse. Requests violating the same-origin policy are potentially the biggest threat to a Web Socket server implementation. The Web Socket standard includes provisions for secure communication over HTTPS. Implementing only the secure version of this protocol may prevent some abuses, or at least make it more difficult for potential attackers. As with all HTML5 features, the specification and API are still in flux. Careful consideration of Web Socket server implementations and thorough testing are crucial to a secure deployment.

5.3 GEOLOCATION

The geolocation API is not technically part of HTML5, but it is often referred to as such. The API provides a means to determine a user's location via JavaScript, returning latitude, longitude and accuracy. If run on a mobile device, the internal GPS along with cell tower triangulation is used because it is more precise; a desktop computer would use an IP-geolocation service and/or local Wi-Fi hotspots.

Typical use-cases for this API include mapping functions such as navigation assistance and points-of-interest applications. Breaking local news events, marketing and advertising are also common examples. Another important use-case is that of social networking. A "Show Friends Near Me" type of functionality could potentially give users access to a collection of locally-stored GPS data. A common implementation practice is to cache data to improve resolving speed, especially on mobile devices. This could lead to the storage of a trail of user positions over a period of time.

For the most part, the geolocation API is not as invasive as it may seem. Users must be prompted for their consent by the website or service that is attempting to use their location. Privacy concerns are well-noted in the specification, as there is no doubt this information could be easily abused.

Different platforms handle the consent request prompt and the resulting saving, clearing or blocking of permission slightly differently. Users should be aware of these subtle differences when using different platforms if the need for location-based services is occasionally desired. Of course, if there is never a need for a user to share their location, all platforms offer the ability to turn this feature off permanently.

5.3.1 SECURITY IN THE SPEC

Privacy is the main issue, as there is only one piece of meaningful data: the user's location. The specification devotes a small section to discuss security and privacy as it relates to the API[15]. The authors realize the need to protect user privacy when using such a relatively powerful piece of data. The first sentence in the privacy considerations section states:

> *User agents must not send location information to Web sites without the express permission of the user.*

It goes on to discuss how an implementation must respect the permission of the user and only request location information when it is necessary and to not persist the information any longer than is needed to perform the function required. The last section is basically a recommendation to implementers that they attempt to protect the user from themselves as much as is feasible and to promote user-awareness of the technology.

5.3.2 TESTING DETAIL

Testing on the geolocation API centered on determining which platforms supported geolocation and if any user-agent failed to properly request permission. Revocations of the permission as well as methods to deny or block location requests are also documented.

5.3.2.1 CHROME (OSX/UBUNTU/WIN7)

When visiting a website requesting location services, the following prompt will pop up right below the URL bar: *<domain name> wants to track your physical location*. This prompt will ask once and then keep the selected permission unless it is explicitly granted or revoked at a later time. It will persist thru cache-clearing, browser reboot and appears to share the permission with Incognito mode. If location permission is granted while browsing in Incognito mode, the permission will be valid in regular browsing mode and vice-versa. Opening a new tab or window for the same page that requested permission will not re-prompt the user once they've accepted.

REMEDIATION

To clear permissions that have already been granted to a site, click the "target" icon in the URL bar of a non-Incognito window and choose *Clear these settings for future visits*. Location settings can be managed by going to `Settings→Show advanced settings→Privacy→Content settings→Location`. Permission settings can then be set to *Ask me when a site tries to track my physical location*, *Allow all sites to track my physical location* or *Do not allow any site to track my physical location*. Furthermore, by selecting *Manage exceptions* location permission can be viewed and managed on a per site basis.

When browsing in Incognito mode, it does not appear to be possible to clear the location settings via the "target" icon in the URL bar and the *clear these settings for future visits* option. The button will appear to click but will not actually clear the location permission cache. Choosing to clear the permission from a regular window will clear the location permission and force a new request from the Incognito window if the page is refreshed.

5.3.2.2 CHROME (ANDROID 4.0.3)

When accessing a site requesting location permissions in Chrome for Android, the prompt will state: *<domain name> wants to use your device's location* and display *Deny* and *Allow* buttons. Both options are all or nothing – the user has either permanently allowed or permanently denied the website from accessing location data. Predictably, these settings survive device reboot. The permission granted is also available when browsing using an Incognito tab. If location permission is granted while browsing in an Incognito tab, the permission will persist for the same site accessed in normal browsing mode.

REMEDIATION

To permit or deny location services after the initial website prompt, navigate to `Chrome Settings→Content settings→Website settings`. Locate the individual site and select it. To allow previously denied permissions, check the *Location access* checkbox. To deny previously allowed permission, uncheck *Location access*. Once again, the page currently displayed will continue to use its originally granted permission level; refresh the page for the new settings to take effect. To globally allow or deny location services in the Chrome browser go to `Settings→Content settings` and check or uncheck the *Enable location* checkbox.

5.3.2.3 FIREFOX (OSX/UBUNTU/WIN7)

Firefox presents more fine-grained control for location permissions. The default prompt asks a user: *Would you like to share your location with <domain name>?* The default option is set to *Share Location*. It is not explicitly stated, but this is on a per-request basis, so every time the application requests the user's location (even on the same page) it will display a prompt. Clicking the small arrow in the permission prompt will display a menu that provides *Always Share Location*, *Never Share Location* and *Not Now* permission options.

If *Always Share Location* or *Never Share Location* is chosen, the permission will persist across multiple tabs, windows and Private Browsing sessions. If location permission is permanently allowed or denied while in a Private Browsing session, that permission will also persist when Private Browsing is disabled.

A caveat with Private Browsing is that the initial request for permission does not have as many choices as non-private browsing. The user is presented only with *Share Location* or *Not Now* on a per request basis. Every time the website requests the user's location they will be prompted whether or not they denied or allowed the previous request.

REMEDIATION

Removing or granting permissions after the prompt is not as obvious as it perhaps could be. Clearing the cache or cookies does not affect location permission data. If the user decides to change their mind after the initial prompt, they must follow a number of steps. The user must be viewing the website and then go to `Tools→Page Info→Permissions`. Under *Share Location* is the ability to select *Always ask*, *Allow* or *Block*. This setting will be initially populated based on the user's response to the initial prompt as described

above. Page Info permissions are the same between private and non-private browsing, setting one affects the other.

There is no global allow/deny as in Chrome, the user must choose to "Never Share Location" for each site that requests it if they desire their position to remain unknown.

5.3.2.4 SAFARI (OSX)

Safari also prompts the user for each request, even if it is within the same page, but with a slight caveat. If, for instance, there is a button that a user must click to generate the location services, the initial button click will prompt the user for permission, but subsequent button presses will not. Reloading the page resets this behavior. Opening new tabs or windows on the same site will cause a re-prompt for permission. Safari's prompt states: *The website <domain name> would like to use your current location* with the added option of a *Remember my decision for one day* checkbox related to the *Allow* or *Don't Allow* buttons. This feature will store the permission for 24 hours and will persist across multiple tabs and windows as well as application restart.

As with the previously discussed browsers, Private Browsing does not have its own permission set – they are shared between private and non-private browsing sessions.

REMEDIATION

Safari is hooked into the OSX System Preferences. The user has the ability to enable or disable Location Services on a system-wide basis by accessing `System Preferences→Security & Privacy→Privacy→Enable Location Services`. Note that OSX Location Services are on by default. An OSX support article[16] states that unchecking an application in the list of services allowed has the effect of denying the application the use of location services. Testing proved this to be not entirely accurate. With the Safari.app checkbox deselected, Safari will still prompt the user for their location, accept the decision and display the content. If Location Services is completely disabled here, Safari will silently deny all requests. Completely turning off Location Services will of course prevent other applications from utilizing any location-based services as well.

If Location Services are enabled system-wide, Safari can be manually configured to block location permission. Accessing the `Safari Preferences→Privacy` tab allows the user to *Limit website access to location services*. The user can globally set location services permissions here with the following options: *Prompt for each website once each day*, *Prompt for each website one time only* and *Deny without prompting*. If a user desires to completely block location services within Safari, this last option is the proper configuration. The *Prompt for each website once each day* option is slightly misleading because once it is set the logical assumption would be that each website will ask once and not again for 24 hours. After selecting this option and accessing a location services website, the standard prompt is displayed. If the user neglects to select the *Remember my decision for one day* checkbox, they will continue to be prompted for each subsequent request. This selection within Safari preferences does not appear to be a truly global setting. If a user selects *Prompt for each website one time only* and access a location services website, the prompt will appear but is slightly different. Instead of the *Remember my decision for one day* checkbox, it is replaced with a *Remember my decision and don't ask me again* checkbox which will permanently allow the current site access to location information. As with the previous example, if the user neglects to select this checkbox they will be re-prompted at the next permission request.

5.3.2.5 SAFARI (IPHONE)

In the mobile version of Safari, every website will prompt for permission for each request for location services. The only options presented to the user are *Don't Allow* or *OK* buttons in the *<domain name> Would Like To Use Your Current Location* popup. Subsequent requests within the page granted access will succeed without user intervention. Denying the initial request will cause subsequent requests to also fail, until the page is refreshed.

REMEDIATION

Location settings can be managed by accessing `Settings→Location Services`. In this menu the user is allowed to turn off location services globally or per-application. By default, Location Settings are turned on. If the setting is turned off, globally or per-application, the user will not be prompted for permission and the request will fail silently. If location services are turned off, subsequent requests within a page that has already been granted access will fail, even if the page has not been refreshed. Once location services are re-enabled, the user is prompted for permission on subsequent requests, the permission is not saved.

5.3.2.6 INTERNET EXPLORER 9/10 (WIN7/WIN8)

The Internet Explorer location services permission prompt pops up at the bottom of the screen, as opposed to the other browsers that display the prompt right below the URL bar. In this location it can be somewhat easy to miss. The prompt request states: *<domain name> wants to track your physical location* and presents *Allow once* and *Options for this site* buttons. The options button contains a small arrow indicating multiple choices, *Always allow* and *Always deny and don't tell me*.

The easiest button to click is the *Allow once* button, which will give permission once per site, per tab/window. Permission will last for the session, until the browser is closed. Multiple requests from the same site will be allowed without further prompts. Opening a new tab in the same window to the same site causes a re-prompt. Choosing *Always allow* will cause the site to not prompt the user again. The *Always deny and don't tell me* option performs as expected.

Granting permission in a regular window and then opening the same site in an InPrivate mode window causes the site to request permission again, even if the user requested the *Always allow* option. Requests for location services in an InPrivate window only display the *Allow once* button. Opening a new tab in the same InPrivate window to the same site causes a re-prompt. To deny permission during InPrivate browsing, the user must click the "x" to close the prompt pop-up.

REMEDIATION

To clear previously accepted or denied permission, go to `Internet Options→Privacy→Clear Sites`. On opening a new tab/window to a website requesting location services, the user will again be prompted. There is one caveat when clearing location permissions; it does not take effect until the window with the site that has been granted the permission has been closed. If a user selects *Clear Sites*, the current website will still retain its allowed permission. Refreshing the site does not make a difference, subsequent requests on that page are still allowed. This caveat affects InPrivate browsing as well. Select the checkbox to the left of the Clear Sites button labeled *Never allow websites to request your physical location* to globally deny location prompts in Internet Explorer.

Note: In Windows 8 there is a control panel option called *Location Settings* that allows an administrator to enable or disable the *Windows Location Platform*, which is enabled by default. This will set location permissions system-wide. If the *Turn on Windows Location Platform* checkbox is unchecked, all subsequent requests for location services are instantly denied, even if IE10 is open to a page that has already been granted permission. Refreshing this page or making requests for location services are all silently denied. Interestingly, once location services are re-enabled via the system setting, the permissions for the website revert back to whatever option was previously chosen. For instance, if a user chooses *Always allow* for a particular website while Windows Location Services is enabled, the website processes the location information without issue. If the Windows Location Services are then disabled, all requests are denied. Once Windows Location Services are re-enabled, the website does not request permission again – it will process location requests under the previous permission setting which was *Always allow*.

Win7 IE9 has something called *Location and Other Sensors* but it is not the same.

5.3.2.7 ANDROID BROWSER 2.3.6 AND 4.0.3

When a website requests location information from the Android Browser, a *<domain name> wants to know your location* window pops up with *Share location* and *Decline* buttons as well as a *Remember preference*

checkbox that is checked by default. A separate request on the same page will cause the website to request permission again, but not for multiple requests of the same control (i.e. multiple presses of a "what's my location button") or page refresh. If the *Remember preference* checkbox is unchecked, the permission setting will not survive reboot; if it is checked, it will. When a user grants a site permission, a popup displays briefly at the bottom of the screen *This site can access your location. Change this in Settings →Website Settings*.

REMEDIATION

Navigating to `Browser Settings→Clear location access` will clear the permissions for all sites. The option directly above it is *Enable location*. Uncheck this to deny all requests for location data. If *Enable location* is disabled after a site has already been granted location permission, that site will continue to function with the granted permission until the page is refreshed. The *Clear location access* option must be selected prior to disabling *Enable location* to prevent this behavior.

Navigating to `Browser Settings→Advanced→Website settings` will show all individual sites that have been granted location permission.

To prevent all applications from prompting for location information in Android 2.3.6, navigate to `Device settings→Location & security settings→My Location` and uncheck *Use wireless networks* and *Use GPS satellites* to turn off location services device-wide.

The settings are similar in Android 4.0.3, navigate to `Device Settings→Location services` and uncheck *Google's location service* and *GPS satellites*.

Note: When turning on *Use wireless networks* or *Google's location service* the user must agree to the *Location consent* popup that states: *Allow Google's location service to collect anonymous location data. Collection will occur even when no applications are running!*

5.3.3 ABUSE

For the most part, the browsers tested performed well in regards to properly prompting the user and allowing them to grant or revoke permissions after the initial prompt. It is perhaps a logical assumption that any data set during a "private" browsing session would persist only for that session, but that is not the case in Chrome or Safari (with *Remember my decision* checked). In Internet Explorer and Firefox, because the user is not permitted to make a permanent "Allow" decision, this is not an issue.

The security implications of this feature are centered on the UI and interface design/notification but the concern is a strong one. If a user does not clearly understand what they are accepting and are not provided an easy means to change their mind after the fact, their privacy could be at risk. This relates more to mobile than desktop browsers because unless a user has GPS hardware installed, the location will typically not be terribly accurate. At the very least, it is not as accurate as a mobile device with GPS that is always on as a user moves. Besides discovering a user's immediate location, the larger concern may be tracking the movement of a user. If a user is not aware of how long an application stores the permission granted to it, it may continue to request and receive updated location information without the user's express consent - effectively tracking the user as they move throughout the environment.

5.3.4 MITIGATION

In an environment such as the enterprise where corporate administrators have control over a user's computer, location services can be turned off at the browser or system level if there is no business need for them. The situation becomes more difficult when dealing with Bring Your Own Device (BYOD) policies, as corporate IT may not have the ability to disable location services on a user's personal device. In this circumstance, user-education (though unreliable) may be the only solution.

For the average user, disabling location services on a per-application or per-device level would obviously prevent surreptitious tracking, although this may be undesirable from a functionality point of view. Understanding the browsers and platforms being used and the intricacies of each will go a long way

towards using location services safely. Above all, the user must carefully read the display when they are prompted to share their location and manually clear the permission when it is no longer needed.

Note that content loaded in iframes can also ask for a user's location. For example, a malicious advertisement is loaded in an iframe in a familiar website. This ad requests the user's location, which the browser will display in a popup prompt. The user, believing they are being prompted for their location from the trusted website, unknowingly allows the ad access to their location information. Therefore it is important to read the prompt carefully and make sure the domain name asking for location data is the one that the user has purposefully navigated to and has a need for that data.

5.4 SCALABLE VECTOR GRAPHICS (SVG) ELEMENT

The SVG specification has been around since 1999, so it is hardly new. However, with the introduction of HTML5, it is gaining widespread support as a full-fledged standard. SVG is a language for describing two-dimensional vector graphics in XML combined with raster graphics and multimedia[17]. It is intended to be used instead-of or in-addition-to the `<canvas>` element. At a very high level SVG and canvas perform the same function (draw graphics) but they are different technologies that are each suited to certain jobs[18].

The SVG standard gets especially interesting when taking into consideration the `<foreignObject>` element[19]. Using this element, HTML and script can be embedded in a SVG "image." In this manner, SVG capabilities such as object transformation can be performed on HTML content. Of course, the use of HTML and script content is not limited to only legitimate uses; an attacker can easily construct an SVG file containing malicious content. If this file is loaded by a web page, the attacker's content will execute.

An SVG object can be embedded in a webpage using a variety of tags, including `<img>`, `<canvas>`, `<iframe>`, `<object>` and `<embed>`. However, while the `<img>` and `<canvas>` tags will display an SVG image, HTML and script content will not execute as these tags will only render the file as a static image. Furthermore, cross-origin restrictions with the object and embed tags prevent loading of cross-domain content. So while it is possible to include these tags in an SVG file, the content they reference must belong to the same domain. A file upload functionality that accepts any file type is likely the best attack vector as the SVG file must be hosted in the same domain for this attack to be effective.

5.4.1 SECURITY IN THE SPEC

Security is not addressed in the specification.

5.4.2 TESTING DETAIL

To test the ability to display HTML and JavaScript in an SVG image, an SVG file was created containing a <foreignObject> element. Within that element various HTML tags were used to display content, such as <input> <script> <a> <object> and <iframe>. This file was then embedded in an HTML file in multiple ways, using <img>, <canvas>, <object>, <iframe> and <embed> to determine how or if the HTML content (and especially the script) embedded within the file would execute. Also within the file a URL to a different domain was embedded within an <iframe>, <object> or <embed> tag to determine how the content would behave when loaded within an SVG file. As the results will show, it is possible to embed JavaScript in an SVG file that will have access to the parent page's DOM. The table in Appendix A outlines the methods tested on each platform and their success or failure.

The only platform and tag combination that this exploit will not execute on is Android Browser 2.3.6 using the <embed> and <object> tags in the main HTML page. All other platforms and browsers tested will execute the script no matter how the SVG file is embedded and coded. It may be more difficult to exploit this vulnerability on certain platforms, depending on the options available to an attacker on the main page and the level of misdirection/obscurity desired.

Pages loaded by the iframe within the SVG file are still held to the same origin policy of the parent page; pages with the *X-Frame-Options* HTTP header enabled will not load. Even if the site has no such option enabled, it will still not be able to access the DOM of the parent page or the SVG file.

5.4.3 ABUSE

The options for abuse are limited by the vectors in the parent page. If it is possible to upload a SVG file somewhere on the same domain as a page that has an injection vulnerability that will support an `<iframe>`, `<embed>` or `<object>` tag, the exploit will be successful. If a website's file upload functionality places the file in the DOM in something other than an `<img>` or `<canvas>` tag the exploit also may be successful.

5.4.4 MITIGATION

At the present time there is no easy way to actually prevent an SVG file from containing the `<foreignObject>` element and HTML/JavaScript. It may be possible to inspect the content of uploaded files and remove unsafe tags or reject the file if it appears to contain malicious code. A better way to prevent this exploit is to prevent SVG files from being accepted by file upload functionality, but this may not be desirable. Placing SVG files in non-executing tags like `<img>` or `<canvas>` will also prevent scripts from executing. Finally, placing user-generated content on a separate subdomain is a technique used by many companies to leverage the same origin policy for security purposes.

5.5 WEB WORKERS

Web Workers provide the ability for a web browser to run scripts in the background, thereby freeing the user interface. In this manner, long-running JavaScript code can continue to run and will not freeze the current webpage, providing an enhanced user experience. According to the specification:

> *Generally, workers are expected to be long-lived, have a high start-up performance cost, and a high per-instance memory cost*[20].

There are two different types of worker objects, each with certain capabilities. A *dedicated worker* is linked to its creator but can only be accessed by the script that created it. *Shared workers,* once created, can be used by any script running in the same origin.

Web workers have limited access once they are running; a list of the available JavaScript features follows:

- XMLHttpRequest
- Application Cache
- create subworkers
- navigator object
- location object
- setTimeout method
- clearTimeout method
- setInterval method
- clearInterval method
- importScripts method

Workers also do not have access to the DOM of the parent page, although a properly configured script could pass data back to the parent page that could then place the data in its own DOM. Typical uses of web workers include:

- Number processing - Performing complex mathematical calculations, including applications such as cryptography
- Input/output - Fetching real-time stock quotes
- Image Processing - Very high-resolution pictures, game graphics

- Processing data from XmlHttpRequests - Updating content for very large files

5.5.1 SECURITY IN THE SPEC

Security is only touched upon briefly in the specification, with cross-origin concerns chief among them. Browsers must be careful to deny cross-origin requests by workers, as with any other technique.

> *Note: Thus, scripts must be external files with the same scheme as the original page: you can't load a script from a data: URL or javascript: URL, and an https: page couldn't start workers using scripts with http: URLs*[21].

Firefox had a bug in version 3.6 where the *importScripts()* method was allowed to read and parse JavaScript cross-domain, but it has since been fixed[22].

5.5.2 TESTING DETAIL

The web browsers and platforms that support web workers appear to coincide with published data as depicted in the table below.

Browser	Platform	Web Worker Support
Chrome	All	Supported*
Firefox	All	Supported
Safari	All	Supported*
Internet Explorer 9	Win7	Not Supported
Internet Explorer 10	Win8	Supported
Android Browser	All	Not supported+

Table 6 - Browser Web Worker Support

* Neither Chrome nor Safari support subworkers at the moment. Subworker support is a known issue in Chrome[23]

+ The Android Browser in both platforms tested does not support web workers. In addition, the error thrown to the JavaScript console *Uncaught ReferenceError: Worker is not defined* is very similar to the one thrown by Safari and Chrome, indicating a lack of support of subworkers. If and when the Android browser gets support for web workers, it will be interesting to see if subworkers are supported as well.

5.5.3 ABUSE

The biggest case for abuse of web workers (besides implementation flaws) is that JavaScript is now running in the background for a potentially long period of time without a user's knowledge. Before web workers, if a script ran for too long, a popup window would alert the user (usually something along the lines of "a script has become unresponsive") and allow the user to stop the script or allow it to continue. Even the most uneducated user may become suspicious or frustrated if too many of those popups occur and navigate away from the page or quit browsing entirely.

With scripts being moved to the background, developers may be tempted to create larger, slower scripts because they will no longer affect the user experience. This new functionality is not without a cost however, web workers may appear invisible to the user but they are still using system resources. A poorly written web worker will consume system memory unchecked until the browser tab or window is closed or the browser crashes. Even if the browser does not crash, a process eating up 80-90% of system memory or CPU is sure to degrade the user experience system wide, not just for a particular website.

Of the platforms tested, it was simple to cause mobile Safari to crash by creating a web worker infinitely looping over an XMLHttpRequest. In a typical page, if a script were to run for too long, a popup would appear alerting the user to the unresponsive script and allow them to terminate it. With scripts running in the background that is no longer the case – the browser will just crash or continue to eat system memory until the tab or window is closed. Since most users keep tabs open for a long time, this could allow a script to continue running for some time without a user's knowledge.

If an attacker can lure users to a page (such as an online game) that creates a malicious web worker, as long as the user stays on that site (or keeps the tab/window open) the script will continue to execute. Browser-based botnets can be a concern, as well as Bitcoin generation or distributed password cracking.

Because it is technically possible to create web workers inline (within a single webpage instead of separate JavaScript files) it may be possible for an attacker to exploit a Cross-Site Scripting vulnerability on an otherwise friendly site that would create and run their malicious web worker. Imagine a stored XSS on a popular site many users often keep open in a tab or window for extended periods of time like CNN or Google. The malicious script would continue to execute, performing actions unbeknownst to the user.

5.5.4 MITIGATION

There is currently no way to prevent individual web workers from executing, short of turning off JavaScript altogether. This is unfortunate, since due to the lack of user interaction the potential for abuse can be high. The saving grace, as it were, is that the worker will stop executing once the tab or window is closed. Educating users to this threat may help to prevent any long term damage done by unknown sites if the tab or window is immediately closed after viewing. Of course, there are many other ways to get compromised by visiting a malicious site and the "close the tab" rule is unrealistic to enforce on legitimate websites, although the chances of popular websites hosting malicious code may be slightly lower.

6 CONCLUSION

HTML5 brings a number of refinements and new features to the web, and with it, new security concerns. The concepts and features presented in this paper are only a small subset of the many new offerings described in the W3C and WHATWG specifications. Due to time constraints, a full analysis was not performed of all HTML5 features. In the interests of completeness, a few of the more popular are briefly outlined below.

6.1 CROSS ORIGIN RESOURCE SHARING (CORS)

The CORS specification defines specific HTTP headers to be used by a web server to allow access to its resources by a web page from a different domain. The basic functionality of CORS is rather simple. A web server or application that supports CORS sends an extra HTTP header in its response that dictates the domain or domains allowed to access information. All modern browsers currently support CORS.

> *Access-Control-Allow-Origin: http://AllowedDomainName.com*
> *Access-Control-Max-Age: 3628800*
> *Access-Control-Allow-Methods: PUT, DELETE*

The CORS specification is one of the more security-intensive features because it has a high potential for abuse. As it would be too lengthy to repeat here, the following links will direct the reader to the appropriate sections.

- http://www.w3.org/TR/cors/#security
- http://www.w3.org/TR/cors/#resource-security
- http://www.w3.org/TR/cors/#user-agent-security
- http://www.w3.org/TR/cors/#cors-api-specification-security

The site at http://code.google.com/p/html5security/wiki/CrossOriginRequestSecurity contains a list of potential security issues with the use of CORS, summarized in the table below.

Issue	Explanation
Universal allow	The Access-Control-Allow-Origin header has the ability to take the wildcard value (*) which will accept CORS requests from any domain. This option should be used very carefully, and should only be used for information that is normally publicly accessible.
Site-level cross-origin access	CORS requires access control on every page that is allowed to honor cross-domain requests. Care should be taken to ensure that a misconfiguration does not allow all pages to be requested by foreign domains.
Access-control based on Origin header	The Origin header does not guarantee that a particular request actually originates from the domain it specifies, and can be easily spoofed. Implementations should check to ensure that the Origin header matches the Host header field.
Prolonged caching of preflight response	If the preflight response is cached for an extended period of time, the client will continue to honor the policy, even if it is changed on the server. The Access-Control-Max-Age header should set a value no longer than absolutely necessary, such as 30 minutes or less.
Misplaced trust	As previously stated, there is a certain amount of trust inherent with CORS requests. All COR requests and responses should be validated by both the server and the client.
Processing rogue CORS requests	If a server attempts to process COR requests even though it is not configured to accept such requests, the possibility exists for an application level Denial of Service. To prevent this attack server implementations should not process these rogue requests.

Table 7 - CORS Security Issues

6.2 NOTIFICATIONS API

The Notifications API provides the ability to display notifications or alerts to users outside of the web browser window. Desktop Alerts are a common example, alerting the user to a new email or chat message. The security implications of this feature fall into the social engineering category. A relatively savvy user will often dismiss a "your computer is infected!" popup from the web browser, but what happens when said popup appears to originate from the OS? Even an experienced user could mistake a desktop popup for an actual message from the operating system, depending how cleverly the attacker has designed the notification.

Much like geolocation, the API requires implementations to prompt the user for permission prior to displaying desktop notifications. This affords some security to the user, although of course a relatively benign-looking site could be compromised. The initial alert could appear obvious and safe while subsequent events could be more malicious. Currently, Chrome desktop is the only browser that supports this API.

6.3 WebSQL and IndexedDB

WebSQL and IndexedDB are alternatives to web storage, because session and local storage are not designed to store large amounts of structured data. Web storage also does not provide in-order retrieval of keys, efficient searching over values, or storage of duplicate values for a key. To address this need, and to support offline web applications, a web storage database standard was needed.

The WebSQL specification is no longer maintained, despite being implemented by the Chrome desktop, Chrome for Android, Safari desktop, Safari for iPhone and Android Browser web browsers. WebSQL is based on SQLite, but without at least two implementations (something other than SQLite) the specification's path to standardization stalled. Mozilla was an outspoken critic of this specification, so it was never implemented in Firefox. A discussion of Mozilla's reasoning behind this decision can be found here: http://hacks.mozilla.org/2010/06/beyond-html5-database-apis-and-the-road-to-indexeddb/

IndexedDB has replaced WebSQL for all intents and purposes, although the web is full of debate on the topic. IndexedDB provides methods similar to SQL, but does not actually have a structured query language. The API uses JavaScript and an "object store" to store data that can then be enumerated, much like SQL. IndexedDB is currently supported by Firefox and Chrome desktop, Chrome Android and Android Browser.

7 Final Thoughts

Adherence to standards does not always guarantee the same behavior among different implementations. The HTML5 standards track is one of the more ambitious attempts at standardization in the history of the web, but only time will tell how successful it is. The specifications have a long way to go before becoming accepted standards; and it is quite likely that dozens (if not hundreds) of browser versions will be released before that happens in 2022 or later. During the course of testing and writing this paper several versions of Firefox and Chrome were released. A brand new Chrome app for iPhone and a new version of Firefox beta for Android were also made available; clearly browser development is not slowing down anytime soon. The HTML5 specification received countless edits during this time, although none appeared to be major changes.

The relationship between HTML5 and mobile platforms is constantly evolving as well. Much of the current buzz centers around gaming, due to the cross-platform graphical and offline storage capabilities the specification offers. The geolocation API may be of particular concern to user-privacy, but barring a colossal bug in a major browser that allows location tracking without asking for consent, a vigilant user can protect themselves from surreptitious monitoring.

While the security implications of new specifications and browser versions should not be overlooked, secure coding practices and defense in depth strategies, including user-education, can help to prevent security incidents. Enterprise administrators must be prepared for these changes and stay current on new developments, modifying security procedures and infrastructure as needed.

8 APPENDIX A

Browser	Method (parent HTML file / element in SVG file)	Results
Chrome (all platforms)	iframe/iframe	Success
	iframe/embed	Success
	iframe/object	Success
	embed/iframe	Success
	embed/embed	Success
	embed/object	Success
	object/iframe	Success
	object/embed	Success
	object/object	Success
Firefox (all platforms)	iframe/iframe	Success
	iframe/embed	Partial*
	iframe/object	Success
	embed/iframe	Success
	embed/embed	Partial*
	embed/object	Success
	object/iframe	Success
	object/embed	Partial*
	object/object	Success
Safari (OSX and iPhone)	iframe/iframe	Success
	iframe/embed	Success
	iframe/object	Success
	embed/iframe	Success
	embed/embed	Success
	embed/object	Success
	object/iframe	Success
	object/embed	Success
	object/object	Success
Android Browser (4.0.3)	iframe/iframe	Success
	iframe/embed	Success
	iframe/object	Success
	embed/iframe	Success
	embed/embed	Success
	embed/object	Success
	object/iframe	Success
	object/embed	Success
	object/object	Success
Android Browser (2.3.6)	iframe/iframe	Success
	iframe/embed	Success
	iframe/object	Success
	embed/iframe	Fail
	embed/embed	Fail
	embed/object	Fail

	object/iframe	Fail
	object/embed	Fail
	object/object	Fail
Internet Explorer 9 and 10 (Windows 7 and 8)	iframe/iframe	Partial[+]
	iframe/embed	Partial[+]
	iframe/object	Partial[+]
	embed/iframe	Partial[+]
	embed/embed	Partial[+]
	embed/object	Partial[+]
	object/iframe	Partial[+]
	object/embed	Partial[+]
	object/object	Partial[+]

Table 8 - SVG Testing Detail

* The JavaScript will execute in this example, reading the cookie set by the parent page, but the HTML content within the SVG file will not load. An *Additional plugins are required to display all the media on this page* prompt pops up, but no missing plugins are available to download.

[+] The JavaScript will execute in this example, reading the cookie set by the parent page but the HTML content within the SVG file will not load.

REFERENCES

SPECIFICATIONS

Web Storage:
http://www.whatwg.org/specs/web-apps/current-work/multipage/webstorage.html#webstorage

Web Sockets:
http://www.whatwg.org/specs/web-apps/current-work/multipage/network.html

Geolocation:
http://dev.w3.org/geo/api/spec-source.html

SVG:
http://www.w3.org/TR/SVG/
http://www.whatwg.org/specs/web-apps/current-work/multipage/the-map-element.html#svg-0

Web workers:
http://www.whatwg.org/specs/web-apps/current-work/multipage/workers.html

Cross Origin Resource Sharing (CORS):
http://www.w3.org/TR/cors/

Notifications API:
http://dvcs.w3.org/hg/notifications/raw-file/tip/Overview.html

WebSQL and IndexedDB:
http://dev.w3.org/html5/webdatabase/
http://www.w3.org/TR/IndexedDB/

FOOTNOTES

[1] http://wiki.whatwg.org/wiki/FAQ
[2] http://www.slideshare.net/anm8tr/the-history-of-html5
[3] http://wiki.whatwg.org/wiki/FAQ#What_does_.22Living_Standard.22_mean.3F
[4] http://www.w3.org/TR/cors/
[5] http://www.whatwg.org/specs/web-apps/current-work/multipage/network.html
[6] http://blog.kaazing.com/2012/02/28/html5-websocket-security-is-strong/
[7] http://developer.android.com/about/dashboards/index.html
[8] http://en.wikipedia.org/wiki/Same_origin_policy
[9] http://tools.ietf.org/html/rfc6455
[10] http://nmap.org
[11] http://www.whatwg.org/specs/web-apps/current-work/multipage/network.html
[12] http://www.ietf.org/rfc/rfc4122.txt
[13] http://www.andlabs.org/tools/jsrecon/jsrecon.html
[14] http://www.andlabs.org
[15] http://dev.w3.org/geo/api/spec-source.html#security
[16] http://support.apple.com/kb/HT5009
[17] http://www.w3.org/TR/SVG/
[18] http://wiki.whatwg.org/wiki/SVG_and_canvas
[19] http://www.w3.org/TR/SVG/extend.html#EmbeddingForeignObjects
[20] http://www.whatwg.org/specs/web-apps/current-work/multipage/workers.html
[21] http://www.whatwg.org/specs/web-apps/current-work/multipage/workers.html
[22] http://www.mozilla.org/security/announce/2010/mfsa2010-42.html
[23] http://code.google.com/p/chromium/issues/detail?id=31666