# QWACs Plugin

## Proof of concept browser plugin to support the two-step verification of qualified certificates for web-site authentication

DECEMBER 2017

## About ENISA

The European Union Agency for Network and Information Security (ENISA) is a centre of network and information security expertise for the EU, its member states, the private sector and EU citizens. ENISA works with these groups to develop advice and recommendations on good practice in information security. It assists member states in implementing relevant EU legislation and works to improve the resilience of Europe's critical information infrastructure and networks. ENISA seeks to enhance existing expertise in member states by supporting the development of cross-border communities committed to improving network and information security throughout the EU. More information about ENISA and its work can be found at www.enisa.europa.eu.

### Contact

For queries in relation to this paper, please use isdp@functional.mailbox
For media enquires about this paper, please use press@enisa.europa.eu.

# Contents

Executive Summary 5
1. Introduction 7
1.1 Conceptual background of the project 7
1.2 The trajectory towards “qualified”: A developing market for website authentication 7
2. Requirements 10
2.1 Validation of QWACs (R1) 10
2.2 Facilitate user recognition of QWACs (R2) 10
2.3 Easy installation (R3) 10
2.4 Universal applicability (R4) 10
3. Browsers 12
3.1 Mozilla Firefox 13
3.2 Google Chrome 14
3.3 Microsoft Edge 14
3.4 Other Browsers 14
4. Extension Architectures 16
4.1 Extension types 16
4.2 Mozilla Firefox 16
4.2.1 Legacy extension system 17
4.2.2 Modern add-on types 18
4.3 Google Chrome 19
4.3.1 Themes 19
4.3.2 Extensions 20
4.3.3 Apps 21
4.3.4 Native Client Apps 21
4.4 Distribution models 21
4.4.1 Mozilla Firefox 21
4.4.2 Google Chrome 22
4.4.3 Other Browsers 23
4.5 Summary 23
5. Possible Solutions 24
5.1 Native Messaging 24

5.1.1 Architecture 24
5.1.2 The Native Messaging Host 24
5.1.3 The Native Messaging Application (Extension) 24

**5.2 Providing online validation 24**
5.2.1 Architecture 24

**6. Discussion 25**

**6.1 Advantages and disadvantages of the proposed solutions 25**

**6.2 Facilitating user recognition of QWACs 26**

**6.3 Important industry trends 26**

**7. Roadmap 28**

**7.1 Steps for Chrome Browser 28**
7.1.1 Create the *native host* application 28
7.1.2 Create the *native client* extension 28
7.1.3 Create installation machinery 28

**8. Proof-of-concept browser plugin 29**

**9. References 30**

# Executive Summary

Regulation (EU) N°910/2014 on electronic identification and trust services for electronic transactions in the internal market (hereinafter eIDAS Regulation), which was adopted on 23 July 2014, introduced in the EU legal framework provisions on qualified certificates for website authentication (QWAC). This report considers the ecosystem for these qualified website authentication certificates, but is more explicitly focused on the interaction between browsers and QWACs, and on the user experience which browsers provide users regarding the security of the websites they are visiting. An important objective of the system of qualified trust services is to increase transparency, reduce fraudulent activity and protect personal information and other sensitive data being transferred over the Internet, thereby reducing the risk to all end users, whether they be governments and public agencies, private enterprise or average citizens. Importantly, the system also seeks to streamline administrative undertakings such as the development of easily adopted and enforced rules regarding certificate maintenance, educating users about the graphical indicators of security, and to reduce the long term costs of system maintenance and the mitigation of potentially illegal activities.

Due to its inherent complexity and importance to the emerging world of qualified trust services, the argument for the development of QWACs stands upon three pillars. First, there is a technical argument which involves the differences in how certificates are created, deployed and used to develop chains of trustworthy authenticity. Secondly, there is an administrative argument, given that there are a handful of prominent browsers, the leaders of which wield significant market power and, therefore, influence how protocols are adopted and which rules will be structured, even in the presence of a governmental regulatory environment. The third argument leans on aesthetics and approaches the topic of functionality at a more basic level: that is to say, how basic end-users interact with the software, given the knowledge that they may or may not be bringing to their daily web-surfing experience. Of course, these discussions do not exist in a vacuum, and must be considered as overlapping areas of the same industry dialogue about how to make the Internet a more democratic, transparent and secure place.

## Structure of the report

- Chapter 1 introduces the conceptual background of the study, defining key parameters and the need for QWACs in the context of a developing market for qualified website authentication certificates.
- Chapter 2 describes the four primary requirements for the creation of any potential browser plugins.
- Chapter 3 explains key features and differences between the market leading browsers and provides a brief, yet comprehensive look at how browsers function and sets up the conditions in which the proposed software tool will operate.
- Chapter 4 presents a discussion of extension architectures specifically for the browsers Mozilla Firefox and Google Chrome, in addition to a survey of the different available distribution models for each of these browsers.
- Chapter 5 proposes the two most viable solutions to the plugin debate, and describes in detail the architecture of each solution alone and in in the context of inter-browser operability.
- Chapter 6 discusses the advantages and disadvantages of the two proposed solutions, and provides and analysis of important emerging industry trends which may impact the direction of further supporting actions for the EU QWACs market.

- Chapter 7 makes a recommendation for the most appropriate and viable proposed solution, and provides a roadmap of technical steps to developing prototypes and, eventually, launching commercial versions of the validation plugin. This chapter also delivers projections for the time commitment we estimate necessary for the development, deployment, maintenance of the proposed solution.
- Chapter 8 provides information about the proof of concept software component developed for ENISA

# 1. Introduction

## 1.1 Conceptual background of the project

ENISA has produced several detailed and informative reports in the previous three years since the roll-out of the eIDAS Regulation, which will help the reader understand the most important aspects of functionality and legal purview of "qualified" trust services as defined within the Regulation[1]. "Qualified Website Authentication Certificates – Promoting consumer trust in the website authentication market" (December 2015) and "Guidelines on initiation of qualified trust services – Guidance for supervisory bodies and for TSPs" (September 2016) are especially useful in understanding the technical and regulatory environment which surround QWACs. These reports are of course supported by the guidelines produced by industry groups like ETSI, such as ETSI EN 319 411-2: "Electronic Signatures and Infrastructures (ESI); Policy and security requirements for Trust Service Providers issuing certificates; Part 2: Requirements for trust service providers issuing EU qualified certificates" (June 2015). ETSI TS 102 853, "Electronic Signatures and Infrastructures (ESI); Signature validation procedures and policies" (December 2014) provides a more detailed look at signature validation. The following section is a brief, yet necessary, overview of QWACs to put this report into the appropriate perspective in consideration of developing the certificate validation software to support the emerging market for qualified website authentication.

The proposal in this report is to develop a software component (plugin) that will support and facilitate the validation of qualified certificates for website authentication through the national trusted lists (TLs) via the EU "List of the Lists".

It should be noted that this report uses the term "validate", "validation" and "validator" from the perspective of the proposed software tool. It is meant to signify the process by which the developed software component reads and confirms a certificate's presence on an EU member state's Trusted List or via the EU "List of the Lists" in order to present users with a visual indicator assuring the trustworthiness of the website visited.

## 1.2 The trajectory towards "qualified": A developing market for website authentication

Between 1995 and 2001, organization validated (OV) certificates were the widely available method for encrypting web communication and data transfer (although some commercial vendors that validated business data before issuing SSL certificates already existed). While OV certificates do offer identity confirmation through a simple vetting process performed by the issuing certificate authority, the browser user interface usually only provides a binary indication of whether a site was secured by one of these certificates or not, and does not provide much more indication about the extent of "added" security. When domain validated (DV) certificates were introduced to the market in 2001, the price of encryption began decreasing as the accessibility of SSL protection across the web increased. However, because DV certificates do not provide identity verification information—although they are by now mostly free—the market has become saturated with this less transparent and, arguably, less apparently secure version of SSL protection. Incidentally, the User Interface (UI) provided by browsers offers only two indicators, as with OV certificates. In fact, the two certificate types are not differentiated by this UI in most browsers,

[1] https://www.enisa.europa.eu/topics/trust-services

giving users the impression that there are no functional differences in security. To address the problems arising from the DV-OV binary problem, around 2007 extended validation certificates were brought to the market, rich with identity information, often undergoing a rigorous vetting process by issuing certificate authorities, but often at a steep price. Instead of acquiring a free certificate to protect a website, site owners can now opt to purchase a considerably more expensive product that was itself more apparently secure and are commonly sold bundled with the promise of enterprise level support and other nonessential perks.

Over half of all internet traffic is now encrypted, with current browsers offering incentives such as higher SEO rankings for SSL protected sites, and deterrents like negative UI markers for conventional unsecured HTTP pages. In the past year alone, the market has jumped around ten percent higher in the proportion of websites that are using some sort of SSL protection. The low cost (or no cost) of DV certificates encourages new site owners to acquire and deploy SSL, and the issuance and installation of this type of certificate is often automated and easy to use.

However, DV certificates, while certainly contributing to the overall increase in encryption, make up a bulk of this expanding market. Since they are the least apparently secure (from what they provide and also from a system wide perspective[2]) and do not require appended identification information, they allow malware and fraudulent websites to 'hide' more easily from currently available common security tools. Problematically, according to a report from Infosecurity Magazine, the protection offered by SSL hides not only the data with good intentions but also the bad[3]. In fact, nearly 50 percent of cyber attacks in the past year reportedly arose from sites and services protected by SSL. These attacks also arise from mixed content portions of some websites, in which protected and non protected portions of their content are included on the same page.

Another related critical system flaw is the presence of free public certificate authorities, who despite fulfilling an important need in the market for certificates, are unequipped and unable to provide the essential service of making sure the entities to whom they provide certificates are trustworthy and authentic. Whereas they may be able to vet simple tasks such as the fact that an email address works, they cannot possibly provide the checks necessary to confirm the identity of all holders of their signed certificates, creating a sizeable hole in the fabric of the whole trust services ecosystem.

It is important to understand that the security of a website means, from a technical point of view, that a certificate encrypts the data traveling between web server and browser client, and makes no judgment about how legitimately "secure" the website's content is. All certificates perform a similar cryptographic function – the largest distinctions lie in the identifying information used by CAs to vet the identity of the certificate holder and the assurance to end users that they have been thoroughly vetted. The differences between "security" and "perception of security" are not just a question of semantic nuance, but a real condition that is inherent to the HTTPS ecosystem. This report uses terms like "security" and "secured" to reflect the continuum of the perception of security that comes from these certificates; certificates which offer higher security, therefore, include more identifying information and build a stronger case for trust than those that do not.

[2] See: Qualified Website Authentication Certificates – Promoting consumer trust in the website authentication market, published 16 May 2016 by ENISA

[3] https://www.infosecurity-magazine.com/news/malicious-ssl-traffic-doubles-in/

The discussion about the technical abilities of deployment, functionality and revocation of certificates leads inevitably to the role of humans in the system design. Administratively, vetting identities, for instance, is necessarily done by people, but is also becoming less commonplace as industry players grapple for taking on the responsibility (and costs) of such activities. Ultimately, this is a question which, to date, important industry stakeholders like browsers and certificate authorities are struggling to answer, but for which the eIDAS Regulation for qualified trust services and providers has more or less explicitly laid out for the providers of trust services in all 28 European member states. The transparency and hierarchy put into place by the eIDAS Regulation and supported by industry working groups builds transparency into the process of certificate creation through deployment and revocation. Simply put, QWACs are the best current answer to the holes created by a system characterized by critical design flaws, flooded with substandard products and amidst industry stakeholders unclear about their role in the upkeep of vital security measures.

It is important to remember that, from a technical point of view, DV, OV, EV and QWAC certificates all support the same or similar cryptographic tools allowing for secure transmission of data between client and web server. The salient difference is in the business services provided – which usually go hand in hand with cost – and the resources that certificate authorities employ (or not) to verify identity and therefore the trustworthiness of certificate holders, and to remediate situations in which certificates are mis-issued or used for fraudulent or illegal purposes. Another clear benefit of QWACs is the defined level of legal liability and responsibility that qualified TSPs share in the provision of security which other (non-qualified) TSPs may not have.

QWACs effectively represent a more rigorously controlled environment for trusting identity in a way that even EV certificates cannot yet provide.

# 2. Requirements

In the sections that follow, we describe the requirements (R1-4) for a secure browser add-on resp. extension that will be able to

- validate QWACs, and to
- indicate that a web site is using a QWAC

## 2.1 Validation of QWACs (R1)

The software component will read and confirm a certificate's presence on an EU member state's Trusted List or via the EU "List of the Lists" in order to verify that it has been created and issued with transparency and according to the definition of qualified trust service as laid out in the eIDAS Regulation.

## 2.2 Facilitate user recognition of QWACs (R2)

Visual acknowledgement by browsers that a QWAC has secured a website is a critical element for qualified trust service providers. Without this feature, users may be prompted with a warning from their browsers, which by itself, can trigger trust concerns for users.

While most common browsers have introduced indicators (e.g. the green bar tab) for websites that use EV certificates, in order for web browser users to easily recognise QWACs, it is important to introduce a different scheme. The differentiator should ideally be clearly visible to any user without any action (or with as little effort as possible) on their part.

## 2.3 Easy installation (R3)

The installation of the browser add-on or extension software must require low user effort and must be achievable without the need for administrative rights. The more difficult the installation of the extension, the less likely users will install the software. For this reason, the extension should be installed with as few clicks or as close to automatically as possible.

As extensions for Firefox and Chrome can easily be installed over the Internet with only a few clicks, this requirement is unproblematic, if only the extension needs to be installed. However, if additional software (such as an application able to validate QWACs) needs to be installed on the user's PC, further complications may arise. Although the installation of such a program may be relatively simple, this step may be too difficult for users who are less technically savvy. Furthermore, the installation of some programs may require administration rights.

## 2.4 Universal applicability (R4)

In order to fulfil R1-3, the most popular browsers (according to user usage) must be supported. Because two of the most used browsers are Google Chrome and Mozilla Firefox, any possible web extension must support at the very least these two browsers, though a longer term approach would be to support additional browsers such as Microsoft Edge and Apple Safari in addition to the mobile versions of these browsers.

Furthermore, the browser extension should be independent of the operating system. This requirement appears to be a matter of course, as Chrome and Firefox can be used on all common desktop operating systems and their extensions are essentially platform independent. However, extensions may require that additional software is installed on the user's PC. In this case, a universal applicability assumes that this software can also run on common desktop operating systems (notably Windows, macOS and Linux).

Moreover, universal applicability means that the extension would work on all web pages.

# 3. Browsers

Web browsers allow users to request and view websites on the World Wide Web. However, as the web attracts many businesses and websites get access to increasingly powerful APIs (application programming interfaces), browsers have evolved to become feature rich software platforms. Today, most user needs can be solved by web applications, making the browser one of the most essential pieces of software installed on an operating system. Abstractly, a browser functions as follows: After the networking component transparently handles all communication necessary to obtain a website's code, the rendering engine at the client side parses and renders it. If a script is encountered during parsing, the interpreter is given its code. Both scripts and mark-up can trigger new requests to the website to fetch additional resources. Thus, all components of a browser are constantly interacting with each other to display the structure and contents of a website. Of course, a browser consists of more than only those three parts: the user interface (UI) and data persistence layers are just two additional examples in a range of components which make up a modern browser.

As browsers play an essential role in today's lives, different needs arise around their UI and behaviour. In order to satisfy those needs without needing to implement all functionality themselves, all large browser vendors have introduced extensions. With the given extension mechanisms, developers can customize large parts of a browser. This allows for different work flows and behaviour to be implemented independently of the browser core itself.

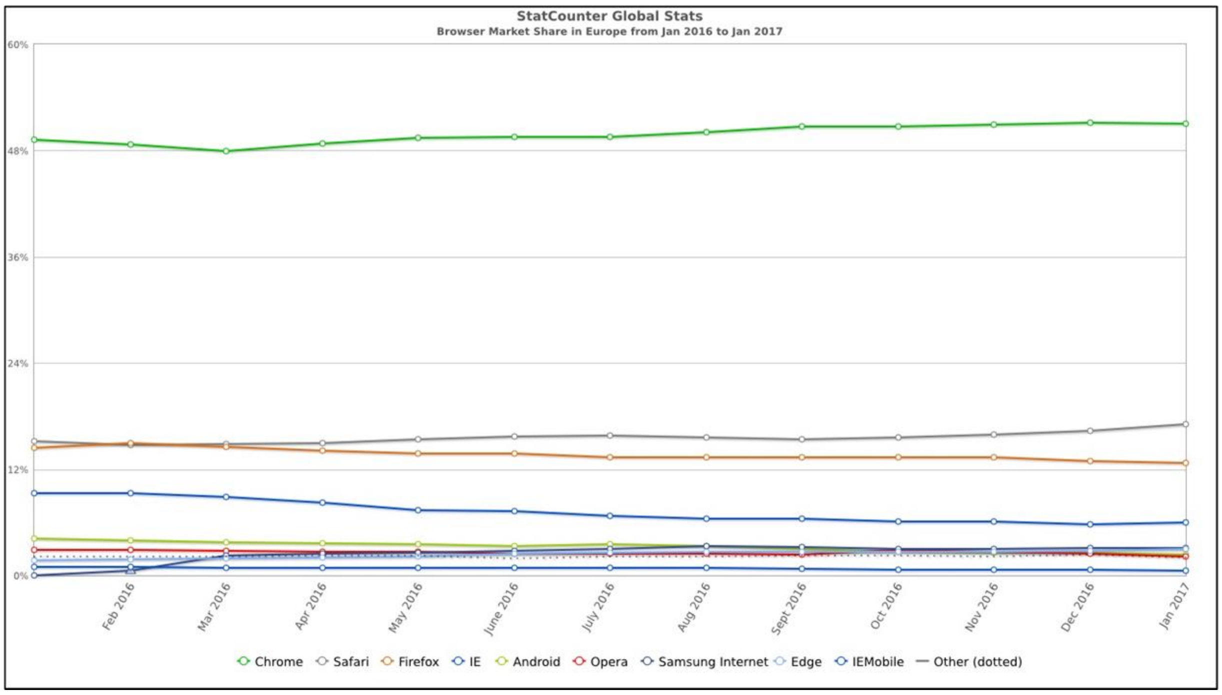

**Figure 1: Browser market share worldwide (courtesy of gs.statcounter.com)**

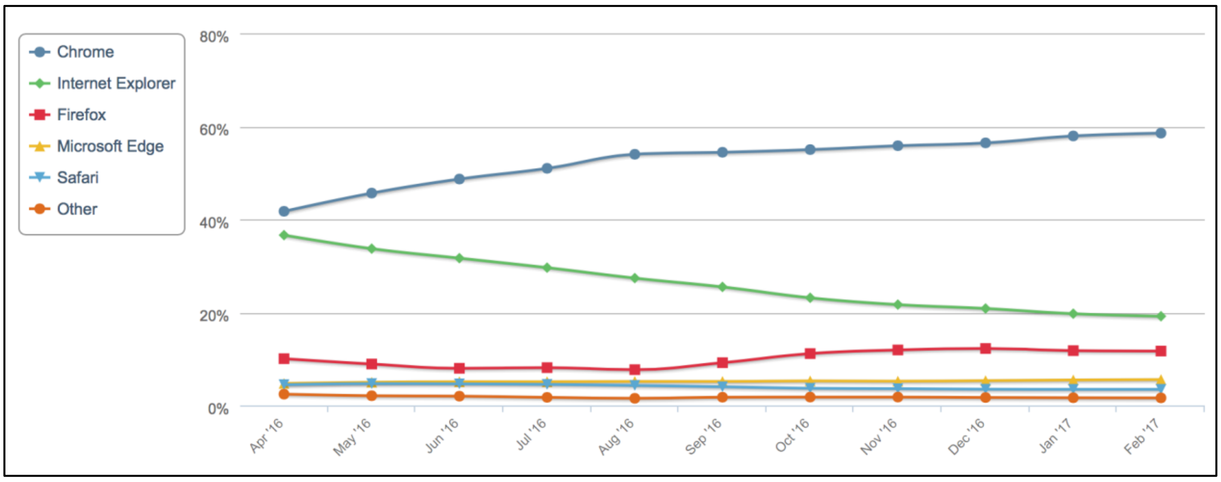

**Figure 2: Browser market share worldwide (courtesy of netmarketshare.com)**

The combined analysis of figures 1 and 2 lead to concluding that the top browsers include Google, Mozilla, Microsoft, and Apple. Perhaps most importantly is the recognition that Chrome occupies a strong share of the market, meaning that:

- any software that is developed will need to work on core components of Google's software,
- Google's protocols for authenticating certificates will need to be monitored in preparation for developing projects, and
- it is likely that other browsers will be using same or similar software components in the machinery of their own platforms, opening potential avenues for interoperability across different versions of the developed extension.

## 3.1 Mozilla Firefox

Firefox is a web browser released by the Mozilla Foundation and its subsidiaries. Firefox is available for Windows (desktop and mobile versions), Apple macOS and iOS, and Linux operating systems, and is also available for Android (formerly Firefox for mobile, it also ran on the discontinued Firefox OS). Its main components are Gecko, a layout rendering engine (s. [MDN2]), and SpiderMonkey (s. [MDN3]), a JavaScript parser and interpreter.

Note: Firefox for iOS was released in late 2015, but this version does not use Gecko due to Apple's restrictions limiting third party web browsers to the WebKit based layout engine built into iOS (s. [WIKI]).

While the available statistics differ slightly in absolute values and significance, the Firefox desktop browser market share can be estimated to around 15 percent globally (s. [STAT]).

Legacy Firefox extensions, sometimes referred to as add-ons, offer many ways of extending the browser's core functionality or changing its appearance. Modern add-on types are strictly limited to a few predefined APIs and settings.

Firefox for Android supports add-ons using the same extension system used by the modern desktop version of Firefox, e.g. WebExtensions (s. [FFANDR]). However, extensions that work with desktop Firefox do not necessarily automatically work in Firefox for Android because the latter is implemented to use the native Android UI.

Apple iOS does not currently support Firefox extensions and add-ons as it is not able to use the Gecko browser engine due to Apple’s restrictions (s. [FFIOS]).

## 3.2 Google Chrome

With roughly 55 percent of the global market share (s. [STAT]), Google Chrome is the most popular and widely used browser at the time of this writing. In contrast to Firefox, the implementation of the browser is not fully available to the public as it contains proprietary components. Large parts of its code base are, however, shared with an open source project called Chromium[4]. The internal Flash implementation, for instance, is not open source whereas other components, like the JavaScript engine and the Blink rendering engine, are.

For its first release in September 2008, Google Chrome employed the WebKit rendering engine which is most prominently known from Apple’s Safari browser. This changed in 2013, when WebKit was forked into Blink in order to speed up development and allow more architectural changes for performance experiments (s. [CHR]). Multiple software projects rely on parts of Chrome’s architecture: recent versions of the Opera browser, for example, leverage the Blink rendering engine in lieu of a former in-house developed engine called Presto.

Chrome extensions have a strong focus on extending the browser’s functionality. Modifications of existing behaviour are strictly limited to a few predefined APIs and settings. In contrast to Firefox, very few additional technologies are introduced for the extension system.

Chrome extensions are currently not supported on Chrome for Android and Chrome for iOS, and there are no current plans to announce at this time (s. [CHRANDR]).

## 3.3 Microsoft Edge

Microsoft Edge (formerly “Spartan”) is a web browser developed by Microsoft with a 4 percent global market share (s. [STAT]) and is included in Windows 10, Windows 10 Mobile and Xbox One, replacing Internet Explorer as the default web browser on all device classes. Edge does not support legacy technologies such as Internet Explorer’s ActiveX or Browser Helper Objects. Browser extension support was developed and added in preview builds in March 2016, and released with the Windows 10 Anniversary Update in August 2016. Edge supports a new HTML, JavaScript and CSS based extension model and is Chrome compatible, which means that existing Chrome extension developers will be able to migrate their extensions to Edge with minimal changes (s. [EDG]).

## 3.4 Other Browsers

Apart from Chrome and Firefox, there are multiple other browsers from which users can choose. While there are too many options to discuss fully, few browsers are conceptually different from the two examined in this study. In fact, a large portion of browsers use the Blink rendering engine and are hence similar to Chrome in a number of relevant aspects. Most importantly, many Blink based browsers have adopted Chrome’s extension system, allowing for easy adjustment of extension techniques. For example, Opera, a browser with a global market share of roughly 2 percent, is based on Blink. Its extension system is almost identical to Chrome’s, with the exception of a few different APIs (s. [OP]). However, the core security model remains unchanged. A similar approach has also been taken by the Vivaldi browser, which, building on top of Chrome, mainly extends its GUI with a pre-installed App.

[4] https://www.chromium.org/chromium-projects

The Chinese 360 Browser, which is the most used browser in China, is also based on the Chromium core engine, but features more options for updating, for example, than Opera.

From the browsers which are uniquely distinct from Firefox and Chrome, Microsoft's Internet Explorer is the most popular with a current market share of about 9 percent. Starting in 1995, it has continuously evolved through the current version 11 with its own rendering engine, script interpreter and extension system. However, it has been recently discontinued to make room for Microsoft's completely rewritten Edge.

Another prominent browser with its own extension system is Apple's Safari. Even though the browser is based on WebKit, which is Blink's predecessor, the extension systems found in Chrome and Safari have few similarities. Safari extensions can also be used on Apple's mobile devices, i.e. on iPhone and iPad (s.[SAF]).

# 4. Extension Architectures

## 4.1 Extension types

This Chapter introduces the extension systems of Mozilla Firefox and Google Chrome. While not as descriptive as the official documentation, the following sections focus on the most important aspects. These two browsers in particular were chosen for their relative market influence and, importantly, the open source nature of Firefox's code. The Secure Information Technology Center of Austria has recently produced a functional beta version of a similar idea plugin, albeit using technology which will be phased out by Mozilla later this year. This initiative serves, however, as an interesting benchmark from which to consider moving forward with a focus on Mozilla in light of other leading browsers such as Microsoft's Edge, whose closed code would make initial efforts more difficult, both technically and economically.

Most browsers feature more than one type of extension. Often, extension types are assigned with different tasks:

- (Modern) Extensions modify the behaviour of existing features of the browser or add new features. The feature could be something in the user interface or a functional feature that manifests itself when a certain action is performed. Search engine definitions are examples of functional extensions.
- Themes are meant to customize the visual appearance of the browser. They strictly modify certain elements of the user interface. Their most prominently featured change is the background image that they add to toolbars, menu bars and status bars of the main application window. They may change the text and background colour as well.
- Localization packages add new languages and location specific display information to the UI.
- Plug-ins render web content that the browser cannot natively render. A plugin declares content types it can handle. When the browser encounters a content type it cannot handle natively, it loads the appropriate plugin, sets aside space within the browser context for the plugin to render and then streams data to it. The plugin is responsible for rendering the data. The plugin runs in place within the page, as opposed to older browsers that had to launch an external application to handle unknown content types. An old framework that enables creation of plug-ins is called NPAPI (Netscape Plugin Application Programming Interface). Due to the age of the API and security issues, as well as the adoption of plugin free web technologies such as HTML5, major web browser vendors began to phase out NPAPI support in 2013. Firefox ended support for NPAPI plugins, except Adobe Flash, in version 52 which was released in March 2017 (s. [FFNAPI]). Oracle Corporation has also announced plans to deprecate the web browser plugin Java Runtime Environment, starting with JDK 9 (s. [ORA]).
- Compiled applications inside the browser use a sandboxing technology (also known as 'native client') for running a subset of Intel x86, ARM, or MIPS native code in a sandbox. It allows safely running native code from a web browser, independent of the user operating system, allowing web based applications to run at near native speeds.

## 4.2 Mozilla Firefox

In the Firefox ecosystem, extensions are commonly referred to as add-ons. Due to its age, the browser has amassed multiple competing ways of writing add-ons, each with their own peculiarities. Broadly speaking, these can be divided into extension types building on top of the legacy extension system and independent ones.

### 4.2.1 Legacy extension system

The group of extension types building on top of the legacy extension system is further separated by the following types:

#### 4.2.1.1 Extensions

Extensions can exercise the full power of the Firefox add-on system by leveraging privileged JavaScript APIs. In order to avoid confusion between the general term extension and this type, add-ons belonging to this group will be called regular extensions in the remainder of this report. Writing a regular extension is possible in three competing ways.

- **Legacy** extensions are the oldest, yet, still working type of add-on in Firefox. Generally, functionality is implemented using Mozilla's Cross Platform Component Object Model (XPCOM) technology. It enables add-ons to invoke the same APIs Firefox uses internally. Furthermore, most legacy extensions use a mechanism called XUL Overlay to customize the browser's UI. Essentially, overlays allow developers to modify elements and attributes in other XUL documents. As the browser's UI is written in XUL, almost all components can be customized or replaced. A major disadvantage of this technique is that it requires a browser restart when a new add-on is installed or uninstalled. Presumably due to optimization reasons, Firefox applies overlays only on browser startup. Unlike a Chrome extension (p.12), legacy extensions based on XUL and XPCOM are not executed in a sandbox. As a result, they have the same rights as a locally executed application. They thus not only have access to the memory of each website, but also from any other extension. The user's hard disk can also be read as far as the software Firefox could. These security concerns may have been one reason why Mozilla has decided to disable XUL and XPCOM for future extensions.
- **Restartless** extensions (also called **bootstrapped** extensions) solve the browser restart issue by disallowing XUL Overlays altogether. Instead, a dedicated JavaScript file (bootstrap.js) provides functions for common events like startup, shutdown or installation. Add-ons are expected to modify the GUI programmatically and undo these changes when being uninstalled. In spite of this modernization, restartless extensions still rely on XPCOM functions and interfaces to provide functionality.
- **Add-on SDK** extensions (formerly called Jetpack add-ons) offer an alternative to the use of XPCOM. Various high and low level APIs are meant to make extension development easier and more accessible. Instead of requiring deep knowledge of Firefox specific technologies, the Software Development Kit (SDK) focuses on providing functionality through standardized web technologies. In contrast to restartless or legacy extensions, no knowledge about XUL, XBL and XPCOM is required. Moreover, the SDK is the first add-on type to introduce isolation between core extension and DOM interaction code through content scripts.

#### 4.2.1.2 Themes

Themes, or complete themes, use CSS to customize the visual appearance of the browser UI. As the UI of Firefox is based on mark-up languages, almost every aspect of the browser can be styled. However, although themes can use all language features of CSS, they are prevented from employing XUL, XBL and privileged JavaScript APIs.

#### 4.2.1.3 Locale packs

Locale packs (or localization packages) contain translations and localization settings for the browser GUI. Mostly consisting of DTD and property files, they have no direct access to privileged JavaScript APIs.

#### 4.2.1.4 Multiple item packages

Multiple item packages bundle multiple add-ons in one package. In contrast to other extension types, multiple item packages implement no functionality on their own. Bundled add-ons, on the other hand, have the privileges they would normally have. This extension type allows distribution of a theme alongside an add-on or similar combinations.

#### 4.2.1.5 Spell check dictionaries

Spell check dictionaries add languages to the browser's spell checking engine. Based on the Hunspell project, Firefox parses one dictionary (.dic) and one affix file (.aff) from each extension of this type. Other than that, the add-on is not able to perform any further actions.

#### 4.2.1.6 Telemetry experiment

Telemetry experiments are described as "specially designed restartless add-ons" and are used to run tests on a wide range of Firefox installations. In terms of technology, they have the exact same advantages and disadvantages as restartless extensions and, thus, belong to the group of regular extensions. However, due to their special purpose and Mozilla's signing process, normal extension authors do not support telemetry experiments.

### 4.2.2 Modern add-on types

Modern add-on types do not follow the distinction of the legacy extension system. Currently, two add-on types belong to this group:

#### 4.2.2.1 WebExtensions

WebExtensions mimic Google Chrome's extension architecture (p.12) and are still in development at the time of writing. Until the end of 2017, legacy and restartless add-ons will be deprecated in favour of this new type of extension (s. [MDN4]). In addition to establishing cross-browser compatibility, Mozilla's implementation of WebExtensions allows for compatibility with a multi-process variant of Firefox. Similar to the Add-on SDK, only standardized web technologies are utilized. Furthermore, WebExtensions use a revised security model, also borrowed from Chrome.

At the time of writing no APIs in Mozilla's (and Chrome's) WebExtensions architecture exist that allow for direct access to SSL/TLS data (s. [SSL]).

Moreover, Mozilla is also beginning to support 'native messaging' (p.14), which enables a WebExtension to exchange messages with a native application installed on the user's computer. This allows native applications to provide a service to add-ons without needing to be reachable over the web. One common example is password managers: the native application manages storage and encryption of passwords, and communicates with the add-on to populate web forms. Native messaging also enables add-ons to access resources that are not accessible through WebExtension APIs, such as some particular piece of hardware.

The native application is not installed or managed by the browser: it is installed using the installation machinery from the computer's underlying operating system. Along with the native application itself, a JSON file called the "host manifest" or "app manifest" will need to be provided and installed in a defined location on the user's computer. This app manifest file describes how the browser can connect to the native application.

The WebExtension must request the "nativeMessaging" permission in its manifest.json file. Conversely, the native application must grant permission for the WebExtension by including its ID in the "allowed_extensions" field of the app manifest. The WebExtension can then exchange JSON messages with the native application using a set of functions in the runtime API. On the native application side, messages are received using standard input (stdin) and sent using standard output (stdout).

Support for native messaging in WebExtensions is mostly compatible with Chrome, with two main differences:

- The app manifest lists "allowed_extensions" as an array of app IDs, while Chrome lists "allowed_origins" as an array of "chrome extension" URLs.
- Firefox and Chrome store their app manifests in different locations.

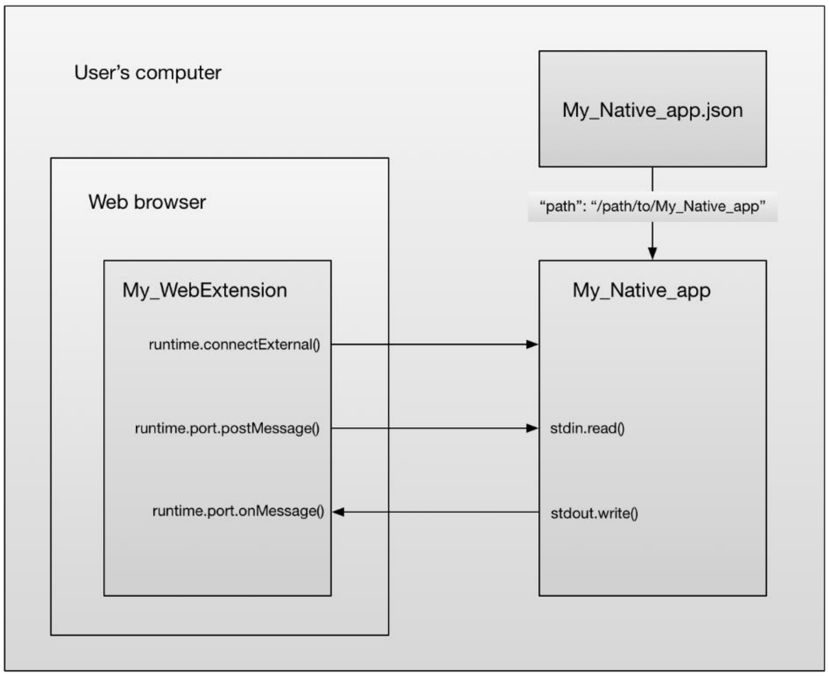

**Figure 3: Mozilla native messaging**

Unfortunately, native messaging is not currently supported by Firefox for Android.

#### 4.2.2.2 Lightweight themes

Lightweight themes merely consist of two images. Separated into header and footer, the pictures are set as the background of the browser's top and bottom part of the window respectively. In Firefox 3.6, the first implementation of lightweight themes appeared under the name Personas (s. [MDN5]).

## 4.3 Google Chrome

Chrome features the following types of extensions: themes, extensions, apps, native client apps:

### 4.3.1 Themes

Themes can modify the appearance of the browser user interface. However, instead of having full control over the visual styling, they can only change predefined elements. For example, while the colour of the browser's toolbar can be altered, its general appearance (e.g. height) is fixed. All modifications are listed declaratively in a manifest file, rendering the use of CSS syntax unnecessary.

### 4.3.2 Extensions

Extensions can request access to a range of APIs, which allow them to perform high-privilege actions. In contrast to Firefox add-ons, however, Chrome extensions do not have the power to execute system commands. Instead, the available APIs are highly specialized and offer a limited amount of control over the browser. Extensions are written using web technologies like JavaScript, CSS and HTML.

Extensions are executed by Chrome in a sandbox, which grant only certain rights. For example, it is not possible for a Chrome Extension to open a pop-up without the user pressing a button. The sandbox borders a Chrome Extension to each website and other extensions. As a result, a Chrome Extension can only interfere with a webpage or other extensions after a certain query has been made. It is also not possible to read the memory (e.g. the local storage) of another web page or extension. This concept is referred to as the 'same origin policy', which is also responsible for the fact that a web page cannot access the memory of another web page, i.e. each extension has its own origin.

When programming a Chrome extension, a central file can be used to define the extension's privilege to leave the sandbox in order to access, for example, the active tab of the browser. Thus, Chrome is able to recognize extensions that need such a right. The user is referred to the permissions required by these extensions, as is the case, for example, for smartphone apps. Only extensions for which the user has granted permissions are able to access web pages.

A Chrome extension consists of different components, which offer different possibilities. For example, content scripts are responsible for interacting with the content of a web page. They have limited access to the main component of the Chrome extensions, the so called 'Background Page'.

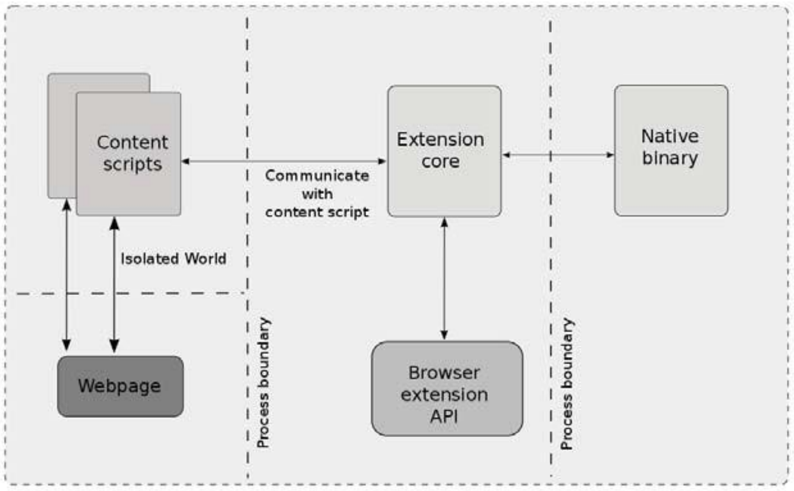

**Figure 4: Google Chrome extension architecture**

Chrome provides various APIs for the extensions, which include both predefined functions as well as ways to communicate with the operating system.

At the time of writing, no APIs in Chrome's (or Mozilla's) WebExtensions architecture exist that allow for direct access to SSL/TLS data. The Chrome dev team even announced that they will not be adding any new APIs to support accessing SSL/TLS information (s. [SSL-C]).

Because a Chrome extension runs in a sandbox, it essentially has no way to start an application on the local computer. The exception to this is 'Native Messaging', in which Chrome receives permission to launch a locally installed application. The browser and the application can then communicate with each other using a defined protocol. The local application must, however, be installed independently of the Chrome extension; there is no way to install local applications from the Chrome Web Store.

### 4.3.3 Apps

Apps attempt to mimic native applications by having access to even more APIs than extensions. In contrast to extensions, they are not meant to modify the browsing behaviour but represent a completely isolated application. Still, apps are not able to execute arbitrary commands on an operating system level. Like extensions, apps can be created with technologies like HTML, CSS and JavaScript. However, Chrome will be removing support for Chrome apps on Windows, Mac, and Linux (s. [CHRAPP]).

### 4.3.4 Native Client Apps

Native Client is a sandboxing technology for running compiled applications within Chrome. Here, programs can be developed in C, C++ or another programming language. The program is translated by a special compiler, which verifies that the program does not execute any prohibited actions. Within Chrome, the program is started in a sandbox, which means that the extension has the same restrictions as a Chrome extension. This technology should allow existing programs to be executed as a secure extension within the browser, with minimal modification effort. By running the programs within a sandbox, they cannot access the local file system and cannot use interfaces of the operating system.

Furthermore, Native Client is not supported by Firefox, so it is not a browser spanning solution. Moreover, it appears that on October 12, 2016, a comment on the Chromium issue tracker indicated that Google's Native Client teams had been de-staffed (s. [DES]).

## 4.4 Distribution models

The distribution of software can be a major hindrance in achieving the goal of easy installation (R3). In all cases, the process is tremendously important to the security of the overall system.

### 4.4.1 Mozilla Firefox

Mozilla requires all extensions to be signed by Mozilla in order for them to be installable in Release and Beta versions of Firefox (s. [ADDSIG]). Signing is done through addons.mozilla.org (AMO) and is mandatory for all extensions, regardless of where they are hosted. Only Mozilla can sign the extension so that Firefox will install it by default. To do so, extensions must pass either an automated or manual code review in order to eliminate disguised malware or potential vulnerabilities.

Listing or distributing extensions through AMO is not required; however, with self-distribution of add-ons, that option may be chosen, and AMO will serve as the way to get the package signed.

Firefox extensions are mainly distributed through the following three channels:

- Whitelisted domains have permission to install extensions by calling the proprietary InstallTrigger API. Remarkably, this list can be modified in the user preferences. However, despite the whitelist, consent is always required for a successful installation. Clean Firefox profiles contain two whitelisted domains, namely the Firefox Marketplace and AMO. The latter is embedded by the Firefox user interface on an internal extension management site (about:addons).
- Third party domains can also access the InstallTrigger API. In contrast to whitelisted domains, they require additional approval of the user. During this process, the browser GUI clearly indicates danger to the user, as the domain is untrusted.

- Local installers may deploy extensions as part of their software bundle. There are two ways to achieve this: Either the high privileges obtained during the installation process are exploited to modify the Firefox root directory or the extension is placed using one of the intentional mechanisms built for this purpose. For instance, on Windows systems, registry keys can be used to enable add-ons for all or only specified users. Furthermore, installers may place a file with the path to the add-on in a profile's extensions directory. This will prompt Firefox to ask the user for consent on the next start.

Both third party domains and local installers cannot distribute arbitrary extensions without Mozilla's signature. Rogue installers, however, are still able to override the imposed security checks by modifying the user's preferences or the Firefox binary itself. (Also, unsigned add-ons can still be installed in Developer Edition, Nightly, and ESR versions of Firefox, after toggling the *xpinstall.signatures.required* preference in about:config.)

While installation of extensions via AMO is intuitive and simple, there may be situation in which further software is needed (e.g. in case of native messaging). The installation of such a program may be relatively complex, even under Windows, macOS or Linux.

### 4.4.2 Google Chrome

Chrome has a complex distribution model, offering different options based on the user's operating system. In general, there are three ways to distribute Chrome extensions:

- The Chrome Web Store is the single authoritative source of extensions for both Windows and macOS operating systems. It requires an upfront fee of five dollars and mandatory reviews to list an extension. While the review guidelines are not public, they most likely attempt to prevent malicious software from being distributed through the store. The Chrome Web Store gives developers the option to carefully distribute extensions only to a selected number of users or in a controlled manner.
- Third party sites have a less important role in the Chrome extension ecosystem. Extensions hosted on a website may only be installed by Linux users. In order to ask the user for an installation, the developer first has to package the extension and then serve it with the correct MIME type (application/x-chrome-extension). Otherwise, if setting the MIME type is not possible, the correct suffix, a regular MIME type and a missing content sniffing prevention header is sufficient, too. While this is not an option for Windows and macOS operating systems, third party sites can trigger an installation from the Chrome Web Store. Using the chrome.webstore API, a user can be prompted to install an extension in this manner.
- Local installers also have different options based on the underlying operating system. On Windows systems, a registry entry can be set. However, it must point at the Chrome Web Store. Similarly, such a link can be placed in a JSON file at a predefined path on macOS systems. On Linux, this file is not required to point at the store, but can also reference other locations.

Chrome employs its own packaging format with the .crx file extension. It uses a custom header which is followed by a standard ZIP file. The header contains information like the packaging format version, a public key and a signature. A public-private key pair is created when first packing an extension locally, or assigned by the Chrome Web Store. Subsequent updates must all bear a valid signature. Inside the ZIP file, all extensions feature a manifest file which describes their type and purpose.

While installation of extensions via AMO is intuitive and simple, there may be situation in which further software is needed (e.g. in case of native messaging). The installation of such a program may be relatively complex, even under Windows, macOS or Linux.

### 4.4.3 Other Browsers

Most other current browsers support similar web stores like AMO or Chrome Web store for distributing extensions.

## 4.5 Summary

The modern extension frameworks of Firefox and Chrome (and therefore also Opera, Edge, Vivaldi, 360 etc.), which mimic Google Chrome's extension architecture, have almost established cross browser compatibility for browser extensions on desktops (i.e. extensions developed for one browser can easily be adapted to another browser). The use of legacy frameworks will be deprecated by end of 2017. Hence, to fulfil requirement R4, any extension that will be able to

- validate QWACs, and to
- indicate that a web site is using a QWAC

should be based on these modern extension frameworks.

While this holds for desktop systems and browsers, for mobile devices the situation is somewhat more restricted; both Chrome for mobile devices (Android and iOS) and Firefox for iOS do not currently support extensions. Only Firefox for Android supports a modern extension framework, namely the same modern extension system used by Firefox for desktops with some minor changes to support the Android UI.

For Apple's mobile iOS devices, the only current possibility to develop extensions would be to develop Safari extensions based on Apple's proprietary extension framework.

In any case, both on desktops and mobile devices, there is currently no way to directly provide information about a given SSL/TLS connection to extensions based solely on these modern extension frameworks (at least at the time of this writing). This would suggest that the evaluation of the corresponding certificates and the validation of QWACs must be implemented outside such an extension.

According to the descriptions within Chapter 3, there are only two possible ways to combine modern extensions with a solution outside the extension, namely:

- native messaging with a native application on the OS of the user's PC, where the native application plays the role of a QWAC validator, or
- providing an additional online service that plays the role of a QWAC validator and that is reached by the extension.

In the following chapter, we will describe these two possibilities in more detail.

# 5. Possible Solutions

## 5.1 Native Messaging

### 5.1.1 Architecture

The proposed solution consists of two major components, the Native Messaging Host and the Native Messaging Application. The Native Messaging Host is the OS native program that implements a QWAC validator. On the other hand, the Native Messaging Application runs in the web browser and has the form of a modern extension. It initiates the validation process by passing the URL of the web site to the Native Messaging Host and retrieves the results of the QWAC validator, which uses the URL to analyse the SSL/TLS-certificate.

### 5.1.2 The Native Messaging Host

The Native Messaging Host is launched by the browser extension and receives a JSON formatted message at its standard input. The message consists of a payload (i.e. information about the SSL/TLS-connection and the visited web site) which is used by the Native Messaging Host to connect to the corresponding web site and validate the corresponding certificate (QWAC). With multiplatform support being a prerequisite, the program should be written in a portable (scripting) language, e.g. Python 3.

### 5.1.3 The Native Messaging Application (Extension)

The Native Messaging Application initiates the validation procedure when the user opens a new web site. The extension then crafts a JSON message containing the URL of the web site and forwards it to the Native Messaging Host. The extension also registers a callback function, triggered when the Native Messaging Host responds with the validation result of the certificate used in the SSL/TLS-connection to indicate the validation result in the UI of the browser, e.g. by a red or green icon, or by showing an icon with the European flag.

In the case of Google Chrome, the extension could be provided for installation through the Google Chrome Web store, and for Firefox through AMO or others. Users could be guided to preinstall the extension and then the native application on the OS. The latter option makes the procedure more difficult for technically less savvy users.

## 5.2 Providing online validation

### 5.2.1 Architecture

The proposed solution consists of a modern extension and an additional online service provided by a trusted third party that plays the role of a qualified online QWAC validator. The extension is built to send the URL of the current SSL/TLS connection to the dedicated online QWAC validator. The online validator checks the certificate and sends information back to the extension. The extension indicates something using a green or red icon, or an icon with the European flag, on the right-hand side of the address panel.

In the case of Google Chrome, the extension could be provided for installation through the Google Chrome Web store, and for Firefox through AMO or others. While, in this case, no additional software must be installed by the users, the challenge to provide a qualified online QWAC validator needs to be handled.

# 6. Discussion

## 6.1 Advantages and disadvantages of the proposed solutions

We have proposed two different ways to implement a modern browser extension that will be able to

- validate QWACs, and to
- indicate that a web site is using a QWAC,

based on native messaging or requiring the availability of a trusted qualified online validation service for QWACS. Both solutions fulfil the requirement R1.

While the first requires the user to install additional software on his or her PC, and thus does not fully fulfil the requirement R3, the second assumes that some trusted third party actually does provide a trusted qualified online validation service for QWACs. However, the establishing and operation of a trusted qualified online validation service for QWACs alone is a challenging task and depends on business cases for the provider. Of course, it is possible but not expected that there will be cost-free online validation services in each member state despite some MS (e.g. Austria) which have enacted laws obliging supervision authorities to offer free validation services. It must also be considered that an online validation service would be an additional link in the validation chain which is more sensitive to attacks, especially to denial-of-service and man-in-the-middle.

| EXTENSION REQUIREMENTS | NATIVE MESSAGING | ONLINE VALIDATION |
|---|---|---|
| [*R1*] Validation of QWACs | + | + |
| [*R2*] Facilitate user recognition of QWACs | + | + |
| [*R3*] Easy installation | +/- | + |
| [*R4*] Universal applicability | +/- | + |
| **Development considerations** | | |
| Ease/cost of development | + | + |
| Requires cooperation of browser vendors | +/- | + |
| Access to browser SSL/TLS information | - | - |
| Expected maintenance | + | +/- |
| Sensitivity to cyber attacks | + | - |

The table above is meant as a visual reference to help simplify the complexity of the decision to more rigorously pursue the development of the native messaging option. Each line item carries with it its own weight of importance, so they are not meant to be read as relative values in comparison with one another, but rather stand alone, indicating where the option lies on the scale from optimal to less than optimal. A

"+" represents a positive outcome with regard to its respective line item. A "-" represents a less optimal outcome and a "+/-" represents a more complex outcome with both positive and less optimal elements.

## 6.2 Facilitating user recognition of QWACs

Regardless of which solution is chosen, the extensions described need to indicate on the browser UI whether a web site is secured specifically by a QWAC or not. The modern extension frameworks allow displaying buttons and/or icons on the browser UI, e.g. a green or red button on the right hand side of the address panel, or even to open a new small browser window that displays some explanatory information. However, the extensions will not be able to assume control of the normal behaviour of the web browser, defined by the browser maker. For example, if the browser (like Chrome) indicates that a user should not trust a SSL/TLS connection because the corresponding SSL/TLS certificate cannot be found in so called "qualified" certificate logs for SSL/TLS end entity certificates, e.g. by a grey or red address bar, this cannot be changed by modern extensions. In the case of a QWAC that cannot be found in the certificate logs, this would trigger trust concerns for users, even if the extension shows that the web site uses a QWAC.

Moreover, if the browser should refuse to render web sites where the corresponding SSL/TLS certificate cannot be found in so called "qualified" certificate logs for SSL/TLS end entity certificates, then the proposed extensions will either be useless or need to open a second window, which would then render the previously refused web site. Unfortunately, we cannot imagine that any customer would put much trust in such a solution.

Hence, in both cases it is difficult or even impossible to guarantee the fulfilment of requirement R2.

## 6.3 Important industry trends

Today, the differences between certificate validation systems can be broadly viewed as fundamental ideological concerns about the responsibility to maintain system wide security by either corporate entities or governmental bodies or, more increasingly, by a transparent open-source community.

In the current model of hierarchical PKI structures, the certificate chain based root concept allows for trust to follow a linear path from a website to a signing certificate's issuing entity; trust is ensured in the integrity of root CAs, leaving significant and well known gaps in security, based on the rules for who can become a CA. Given that the number of certificates is on the rise and nowhere yet near market saturation, revocation lists must be built and maintained to reflect an increasingly larger ecosystem of SSL/TLS certificates. Although certainly market leading CAs have taken steps to address many common security concerns, validating trust chains in real time can be a cumbersome if not impossible task, and certificate revocation lists are also updated and audited too slowly in an environment of increasingly mis-issued certificates.

There are, of course, responses to these complex security issues. For example, OCSP offers an answer to CRLs and can check the revocation status of a certificate more quickly and without requiring the client and network resources needed to check CRLs. Firefox is moving entirely away from checking CRLs and toward a revocation list push mechanism that frequently updates the browser.

Importantly for this report, the European Trusted Lists system seeks to answer these security flaws with a more centralized review responsible for vetting and monitoring issuing entities and their certificates. The proposed QWACs browser plugin would allow the best features of both systems to work together, vetting certificate specific revocation and other important identifying information, and only from signing certificates which originate from strictly vetted issuing entities contained within the member states' lists or in the "List of the Lists". This concept favours organizational trustworthiness in lieu of the certificate and

CA chained hierarchy developed in the root PKI system; trust service providers and the services they provide are granted “qualified” status once they have met the requirements set forth in the eIDAS Regulation, and member states are required to maintain and publish these Trusted Lists.

Amidst the varied methodologies inherent to the trust landscape, however, a new brand of security protocol is emerging with some force. Later this year (by October 2017), Google will be transitioning to a mandatory policy of “certificate transparency”, opening the way for an online community of open certificate logs, monitors and auditors to provide near real time validation of certificates. This push away from the certificate authority based infrastructure relies on so called “qualified” certificate logs for SSL/TLS end entity certificates. If a CA does not include a certificate in the CT log server, it will not be treated as qualified or trustworthy, raising important questions about users receiving conflicting signals of trustworthiness (i.e. a “green light” from the EU side and a red flag from Chrome). How the European efforts led by the eIDAS Regulation respond to CT going forward will determine the direction of qualified trust services provided under it. Certainly, the (lasting) strength of any software tool developed in support of authenticating (in addition, displaying to users) QWACs will need to correspond, at least in part, with the security protocols of current market leading browsers.

It should also be noted that this study reveals that the only possible solutions for the implementation of a modern browser extension able to validate QWACs with currently available techniques are based on surpassing the browser functionality. Unfortunately, extensions do not give access to the browser’s SSL/TLS information. A much more secure route would be an interface in the browsers extension framework by which extensions could gain access to the browser validated certificate. Irrespective of whether online validation or native messaging is ultimately used (the latter can have the same issue, even if it appears less vulnerable), discussions with the browser vendors should start in getting a documented interface so QWAC validation can be developed appropriately.

This study concludes that the most effective use of resources lies in the development of a native messaging application, while retaining consideration for additional support from an online validation service. Through continued dialogue with relevant stakeholders, the most optimal outcome is possible with direct support from the browser and OS vendors.

# 7. Roadmap

The native application is not installed or managed by the browser: it is installed using the underlying operating system's installation machinery. Along with the native application itself, a JSON file called the "host manifest" or "app manifest" will need to be provided and installed in a defined location on the user's computer. The app manifest file describes how the browser can connect to the native application.

The WebExtension must request the "nativeMessaging" permission in its manifest.json file. Conversely, the native application must grant permission for the WebExtension by including its ID in the "allowed_extensions" field of the app manifest.

After that the WebExtension can exchange JSON messages with the native application using a set of functions in an appropriate API. On the native app side, messages are received using standard input (stdin) and sent using standard output (stdout).

## 7.1 Steps for Chrome Browser

### 7.1.1 Create the *native host* application

This is the application in the OS to create which will be started when a Chrome port is instantiated. This application is responsible for validation the QWAC of a given URL for an SSL/TLS connection. To be connected with the browser, both the standard input and output streams are required to be managed, so anything which allows for this ability can be used e.g. C# or Python.

Messages passed between the extension and the application need to be formatted and encoded and respectively decoded using UTF8. Of course, the integrity of the messages also needs to be secured.

### 7.1.2 Create the *native client* extension

The extension will implement the user interface and indicate the validation result of the certificate based on the messages of the native application.

### 7.1.3 Create installation machinery

This step is for the native application and for extension to ensure an appropriate distribution model.

# 8. Proof-of-concept browser plugin

A proof-of-concept software tool, which has the potential to recognize and signal to users the deployment of EU qualified website authentication certificates (QWACs), has been developed.

At present, live testing is not a functional possibility due to the nature of full eIDAS compliant QWACs, namely that no QWACs are currently commercially available and issuing qualified "test certificates" is not possible due to binding policy requirements. More plainly, the software has been developed to specification, but the lack of available certificates for beta testing means the software cannot be functionally tested until further notice.

While 10 companies have been identified via the EU "List of the Lists", which are currently approved as qualified providers of website authentication certificates, none of these TSPs actually provide this particular product in their current product portfolios. The table below provides a list of these TSPs, their national origin and a link to their website. Updates to products and services will be monitored until they become commercially available from these or other (Q)TSPs.

| | PROVIDER NAME | COUNTRY | WEBSITE |
|---|---|---|---|
| **1** | PostSignum | CZ | http://www.postsignum.cz/ |
| **2** | Netlock Sign | HU | https://netlock.hu/ |
| **3** | InfoCert | IT | https://www.firma.infocert.it/ |
| **4** | QuoVadis | NL | https://www.quovadisglobal.nl/ |
| **5** | KIR | PL | https://www.kir.pl/ |
| **6** | TS Authority of Slovenia | SI | http://www.si-trust.gov.si/ |
| **7** | halcom | SI | http://www.halcom.si/ |
| **8** | POŠTA CA | SI | https://postarca.posta.si/ |
| 9 | D-TRUST | DE | https://www.bundesdruckerei.de/ |
| **10** | disig | SK | http://www.disig.eu/ |

# 9. References

- [ADDSIG] Add-ons/Extension Signing. https://wiki.mozilla.org/Addons/Extension_Signing. Retrieved 2017-03-13
- [CHR] The Chromium Projects. Developer FAQ. 2013. https://www.chromium.org/blink/developer-faq. Retrieved 2017-03-13
- [CHRANDR] Chrome for Android. FAQ. https://developer.chrome.com/multidevice/faq. Retrieved 2017-04-27
- [CHRAPP] From Chrome Apps to the Web. https://blog.chromium.org/2016/08/from-chrome-apps-to-web.html. Retrieved 2017-03-13
- [DES] https://bugs.chromium.org/p/chromium/issues/detail?id=239656#c160. Retrieved 2017-03-13
- [EDG] Microsoft Edge extensions. https://docs.microsoft.com/en-us/microsoft-edge/extensions. Retrieved 2017-03-13
- [FFANDR] Extensions for Firefox for Android. https://developer.mozilla.org/en-US/Add-ons/Firefox_for_Android. Retrieved 2017-04-27
- [FFIOS] Add-ons in Firefox for iOS. https://support.mozilla.org/en-US/kb/add-ons-firefox-ios. Retrieved 2017-04-27
- [FFNAPI] Firefox dropping NPAPI plugins by the end of 2016—except for Flash". Firefox Site Compatibility. https://www.fxsitecompat.com/en-CA/docs/2016/plug-in-support-has-been-dropped-other-than-flash/. Retrieved 2017-03-13
- [MDN1] Add-ons. Mozilla Developer Network. https://developer.mozilla.org/en-US/Add-ons. Retrieved 2017-03-13
- [MDN2] Gecko. Oct. 2015. https://developer.mozilla.org/en-US/docs/Mozilla/Gecko. Retrieved 2017-03-13
- [MDN3] SpiderMonkey. Sept. 2015. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/ SpiderMonkey. Retrieved 2017-03-13
- [MDN4] Legacy extensions. https://developer.mozilla.org/en-US/Add-ons/Overlay_Extensions. Retrieved 2017-03-13
- [MDN5] Lightweight themes. URL: https://developer.mozilla.org/en-US/Add-ons/Themes/Lightweight_themes. Retrieved 2017-03-13
- [OP] Dev.Opera. Extension APIs Supported in Opera. URL: https://dev.opera.com/extensions/apis/. Retrieved 2017-03-13
- [ORA] "Oracle deprecates the Java browser plugin, prepares for its demise". Ars Technica. https://arstechnica.com/information-technology/2016/01/oracle-deprecates-the-java-browser-plugin-prepares-for-its-demise/. Retrieved 2017-03-13
- [SAF] Safari Extensions. https://developer.apple.com/safari/extensions/. Retrieved 2017-04-27
- [SSL] Web extensions: SSL (TLS) status API request. https://bug623317.bugzilla.mozilla.org/show_bug.cgi?id=1322748. Retrieved 2017-03-13
- [SSL-C] Provide information about the TLS connections to extensions. https://bugs.chromium.org/p/chromium/issues/detail?id=107793. Retrieved 2017-03-13
- [STAT] StatCounter. http://gs.statcounter.com/browser-market-share/desktop-tablet-console/worldwide/#monthly-201601-201703. Retrieved 2017-03-13

ENISA

European Union Agency for Network
and Information Security
Science and Technology Park of Crete (ITE)
Vassilika Vouton, 700 13, Heraklion, Greece

Athens Office

1 Vasilissis Sofias
Marousi 151 24, Attiki, Greece

PO Box 1309, 710 01 Heraklion, Greece
Tel: +30 28 14 40 9710
info@enisa.europa.eu
www.enisa.europa.eu