Verifying Web Applications

Shai Halevi

July 7, 2023

Abstract

The use of code in web pages is ubiquitous nowadays, but it is virtually impossible for end-users to know what this code is actually doing or to get any assurances about it. The need for better verification of web applications was recently noted in the *Core Verify* project by Meta. That project implements a browser extension that compares the code in the browser to some "ground truth" that the developer posted elsewhere, and displays a warning if it finds a mismatch. This is a good first step, but it doesn't provide the same level of assurance that users get from a mobile app store.

In this short note I put forward a direction for evolving the Code Verify approach, that I think will provide a much better assurance. Specifically, I propose to establish an infrastructure similar to the PKI for TLS, alowing developers to describe the functionality of their web applications and auditors to vouch for the implementation of these applications.

1 Web Pages as Programs

These days, we are all accustomed to programs that are running in the web pages that we interact with. Sometimes these programs only display dancing pigs for our enjoyment, but other times they handle more serious jobs. For example, the WhatsApp Web page is supposed to ensure end-to-end security by encrypting our communications, only sending to the server the encrypted traffic. Another example, web-wallets for cryptocurrencies are supposed to keep our secret-keys safe in memory, and only use them to sign transactions that we authorize.

But a user interacting with these pages has no effective means to check that they are actually doing what they are supposed to. For example, a WhatsApp Web user only sees messages in different colors, and a padlock in the URL line that gives some assurance as to the origin of that page (e.g., `web.whatsapp.com` in this example). The user must take it on faith that `web.whatsapp.com` indeed sends the appropriate code with the page, that this code was inspected (by someone) and found to be doing the right thing, and that the back-end has adequate operational controls to ensure that the code which is delivered to the browser is the one that was inspected. In many cases, this trust turns out to be misplaced, and cryptocurrency users suffered many millions of dollars of losses due to web-wallets with malicious code that sent their secret key to the attackers.

It is instructive to compare this situation with the assurances that users get when they install an app on their mobile device: here too the users must trust the app developer, but in that case the app store serves as an intermediary, subjecting the developers to a somewhat-rigorous process for uploading each version of the app, documenting that process, and letting users see that documentation (as well as reviews by other users). Similar assurances are available for installing browser extensions from browser addon stores. But no equivalent process is available for web

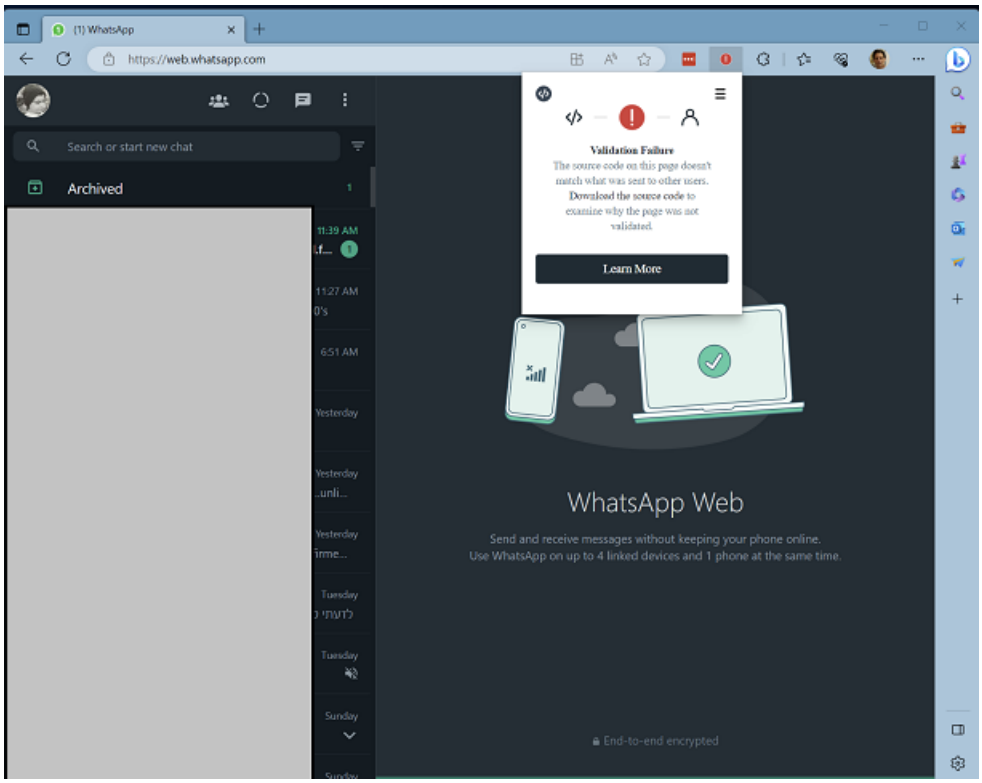
applications, users typically have no information on who developed the code running in the current page, what this code is supposed to be doing, when was it last modified, who examined it, etc.

1.1 The Code-Verify Browser Extension

Recognizing this gap, Meta unveiled in 2022 the Code Verify project for code authenticity on the web [1]. They implemented a browser extension [2] that when installed, lets a user verify the authenticity of certain Meta web applications. This system asks the developer to post (a digest of) the current version of the code on some public registry (specifically one hosted by Cloudflare), and the extension checks the code in the current web page against that registry. As noted on the installation page for this extension:

> *The Code Verify browser extension brings the same protections that mobile apps have to the web. The extension scans the JavaScript code of the web-based app and ensures it matches the source of truth that has been publicly posted on Cloudflare. If there are inconsistencies, the extension will immediately alert you.* [...]

For example, here is how the WhatsApp Web page looks on my Edge browser with the plugin installed:[1]



While this project is a step in the right direction, it still falls short of its ambition to bring "the same protections that mobile apps have to the web". Specifically, it lacks the oversight and documentation aspects of app stores. There is no process for providing and updating the code, and no way for the end user to see what the code is supposed to be doing, when was it updated, etc.

[1] It seems that the WhatsApp Web developers do not keep the registry up to date, hence the warning.

2 A Proposal: PKI for Web Applications

It is clear that to get some measure of assurance in web applications, some trust infrastructure must be set up. My suggestion is to try and mimic the existing PKI that is used in TLS to vouch for the origin of pages, setting up a similar PKI to vouch also for the code running in these pages. This includes in particular:

- Visual elements in the browser that signify a signature on an "approved code", similar to the current padlock in the URL line. Hovering the mouse or clicking these visual elements should show the user context information about the code. At the very least it should display the name of the web application and when it was last updated. More information could include a human-readable description of the intended functionality, version history and release notes, audit information and assurance level, what operational security measures are put in place, etc.

  Initially, such visual information can be provided by a browser extension (such as Code Verify). Later on, as the PKI becomes more widely used and gets standardized, the visual elements can be integrated into the browsers themselves.

- Entities that play a role similar to TLS individual key-holders, vouching for (each version of) the code. These individual key holders could be the application developers, or code-auditing companies and/or third-party consortia that can also provide assurance for the suitability of the code. These signers will sign the code itself and associated context information.

- CA-like entities, that vet individual key holders (and possibly also individual pieces of code). These too can be code-auditing companies or third-party consortia, and can also be higher-level aggregators.

- Root-CA entities, that compile a list of CA's and integrate them into the system. Initially, when using a browser extension, the creators of the extension will play the role of the root CA for it. Later, that role will be assumed by a more traditional CA-like company (or by the browsers themselves).

Hopefully, we can learn from our experience setting up PKI for TLS and avoid many of the pitfalls along the way.

References

[1] Richard Hansen and Vicente Silveira. Code Verify: An open source browser extension for verifying code authenticity on the web, `https://engineering.fb.com/2022/03/10/security/code-verify/`, accessed July 2023.

[2] meta-code-verify, `https://github.com/facebookincubator/meta-code-verify`, accessed July 2023.