# Poster: Let History not Repeat Itself (this time) — Tackling WebAuthn Developer Issues Early On

Aftab Alam
s8afalam@cs.uni-saarland.de
Saarland University

Katharina Krombholz
krombholz@cispa.saarland
CISPA Helmholtz Center for Information Security

Sven Bugiel
bugiel@cispa.saarland
CISPA Helmholtz Center for Information Security

## ABSTRACT

The FIDO2 open authentication standard, developed jointly by the FIDO Alliance and the W3C, provides end-users with the means to use public-key cryptography in addition to or even instead of text-based passwords for authentication on the web. Its WebAuthn protocol has been adopted by all major browser vendors and recently also by major service providers (e.g., Google, GitHub, Dropbox, Microsoft, and others). Thus, FIDO2 is a very strong contender for finally tackling the problem of insecure user authentication on the web. However, there remain a number of open questions to be answered for FIDO2 to succeed as expected. In this poster, we focus specifically on the critical question of how well web-service developers can *securely* roll out WebAuthn in their own services and which issues have to be tackled to help developers in this task. The past has unfortunately shown that software developers struggle with correctly implementing or using security-critical APIs, such as TLS/SSL, password storage, or cryptographic APIs. We report here on ongoing work that investigates potential problem areas and concrete pitfalls for adopters of WebAuthn and tries to lay out a plan of how our community can help developers. We believe that raising awareness for foreseeable developer problems and calling for action to support developers early on is critical on the path for establishing FIDO2 as a de-facto authentication solution.

## CCS CONCEPTS

• **Security and privacy** → *Multi-factor authentication*; **Usability in security and privacy**.

## KEYWORDS

WebAuthn; FIDO2; Usable Security for Developers

**ACM Reference Format:**
Aftab Alam, Katharina Krombholz, and Sven Bugiel. 2019. Poster: Let History not Repeat Itself (this time) — Tackling WebAuthn Developer Issues Early On. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3319535.3363283


*CCS '19, November 11–15, 2019, London, United Kingdom*

ACM ISBN 978-1-4503-6747-9/19/11.
https://doi.org/10.1145/3319535.3363283

## 1 INTRODUCTION

FIDO2 is an open-source authentication standard that succeeds the Universal 2nd Factor (U2F): it provides strong security and privacy for user authentication to web-services by relying on a public-key cryptographic challenge-response protocol—WebAuthn—and hardware-based authenticators that store user credentials (e.g., YubiKey, TPM, or an Android phone). WebAuthn, being a W3C standard, found rapid adoption among the major web browsers as well as among the top web services, like Google, Microsoft, Dropbox, or Github. As its adoption at web-scale is still in the early stages, we are now at a critical point at which we can observe how well web developers adopt WebAuthn for their services and which security-critical issues they face. Web developers are responsible for WebAuthn deployment and history has shown that they are function-oriented [9], often paying too little attention to security. In common scenarios, such as SSL/TLS deployment [3–6], password storage [7], or applying cryptographic APIs [2], usability issues and incorrect mental models of developers have been discovered that undermine security—unfortunately long after flaws have been deployed at scale and insecure solutions have become the standard. In order to not repeat history for the deployment of WebAuthn, we want to call for action to investigate and tackle developer issues with WebAuthn early on.

In this poster, we report ongoing work on investigating pitfalls in the WebAuthn specification and existing WebAuthn libraries for *secure* deployment of WebAuthn and on identifying key usability issues from the developers' perspective, based on qualitative data from discussions on developer forums. These results should lead to more focused studies and ultimately to tangible solutions, such as fail-safe default libraries and configurations, tool support, or "golden guidelines" for WebAuthn adopters.

## 2 PRIMER OF WEBAUTHN

We first introduce technical details about WebAuthn. For the sake of brevity, we focus on the responsibilities of the web server (i.e., the *Relying Party*) and abstract parts of the client steps. For a complete overview, we recommend, for instance, articles by Yuriy Ackermann [1]. In a nutshell, with the help of the client application (e.g., browser), FIDO2 with WebAuthn facilitates communication between two components—web applications and hardware cryptographic authenticators built-in to or attached to the client platform—enabling strong user authentication using public-key cryptography. To ensure compatibility with a wide range of hardware authenticators and make sure the implementation works properly and securely, the Relying Party (RP) is responsible for implementing various features. Those include support for strong cryptographic algorithms, key attestation formats and modes, or restrictions on

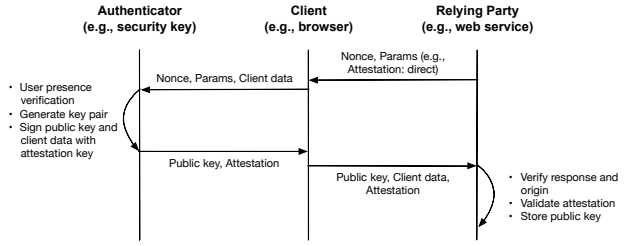


**Figure 1: Abstract WebAuthn client registration**

authenticator selection by the user. Generally, no operation should be performed without user consent. Following, we explain the basic registration and authentication with WebAuthn.

*Registration.* A client that wants to register a public key for future authentication to the RP proves possession of the private key in a challenge-response protocol (see Figure 1). Here, the RP carries responsibility in a) choosing appropriate parameters that reflect its security policies, which the client has to show to be fulfilling, and b) verifying the client response. In particular, in addition to the randomly generated challenge, the RP selects parameters that include **i)** a suitable set of cryptographic algorithms; **ii)** the required user verification that determines how the client application (e.g., browser) will try to enforce user consent, either with PIN or biometrics, or only with a simple physical interaction like a button press; **iii)** the authenticator attachment, i.e., if a built-in platform authenticator (e.g., TPM) or an external roaming authenticator (e.g., YubiKey) is required from the client; **iv)** the authenticator attestation (where *None* is the default attestation mode and the RP needs to explicitly mention what type of attestation they need, e.g., *Direct* for a full attestation).

If the client consents to those parameters, it creates a new public key-pair with the hardware authenticator (under the required constraints, such as user verification) and responds with the signed challenge plus additional meta-data, such as the requested attestation (if not *None*). The RP now has to verify the signed response parameters. For instance, to avoid man-in-the-middle attacks (e.g., online phishing), it has to correctly verify the origin and SSL channel ID the client reports in its response—only if matching to the RP's connection parameters, the RP should proceed. The RP has to verify the attestation object, if present, in order to verify that the private key is protected by, e.g., a dedicated hardware chip. Moreover, the RP has to be aware of several types of attestation formats depending on the authenticator device, such as Packed Attestation, TPM attestation, U2F attestation, or Android key attestation.

A full overview of validation by the RP can be found in the WebAuthn specification [8] and is comprised of 18 individual steps. In summary, the developer of the RP has to be aware of a various critical parameters, their meaning, and how to verify them.

*Authentication.* Authentication follows the same challenge-response protocol using a previously registered public key for the client—a client might have several keys registered for their account—and similar validation steps by the RP (e.g., checking the origin and SSL channel ID to thwart man-in-the-middle attacks). Additionally, the RP has to implement clone detection of authenticators using a monotonic counter that is kept in sync between the authenticator and the RP (i.e., the RP has to securely store and manage the counter). If counter values between client and RP mismatch, this might indicate a cloned authenticator that should not be trusted.

## 3 WEBAUTHN PITFALLS AND ISSUES

We take a systematic approach to identify and understand developer issues with WebAuthn. First, we study the WebAuthn specification and use an expert assessment to pinpoint potential pitfalls for developers (e.g., the various steps and involved parameters). Second, we investigate existing open-source libraries for WebAuthn (e.g., their default configurations and support for features), since current developers will likely base their implementation on top of those. Third, we analyze the topics of discussions around WebAuthn on StackOverflow and other developer forums. As a result, we were able to identify potential problem areas for WebAuthn deployment, which we will briefly introduce in the remainder of this section.

### 3.1 Mental Models

To gain a better picture about the current concerns of developers, we studied the discussions about WebAuthn and FIDO2 posted by developers on StackOverflow and, even more so, on Google's developer forums. Generally, the questions were in the range from big-picture or project idea questions to inqueries about specific technical details. Although the WebAuthn developer community is still in its infancy and steadily growing, we could already identify three themes of questions that were most frequent among the discussions and that should raise concerns.

*Need for secure deployment-ready solutions.* A number of developers inquired about how to start FIDO2 development (*"I need build a web site with FIDO2, but, what are the steps? I know that It's needed a FIDO Server first but, which is the way to implement it?"* or *"From a developer point of view, what is the best resource to start implementing FIDO (Client and server)?. I have been checking the FIDO alliance website for some time but for a funcional[sic] understanding."*) and in many cases they are referred to existing open-source projects or they directly ask about existing solutions. That third-party code is a double-edged sword is well-known and we took a separate look at existing FIDO2 projects (see Section 3.2).

*Wrong mental models.* We also found first indications that developers have or create wrong mental models about FIDO2 and WebAuthn. For instance, they connect it with the wrong attacker model (*"can FIDO2 protect against Ransomware attacks?"*) or are unclear about the form of available authenticators (*"Is FIDO and WebAuthn just about physical security keys???!!!"*). Also questions regarding general workflows indicate a misalignment between what developers expect and what the specifications define (for instance, *de*-registration of users: *"How can a user de-register their authenticator? Assume, there can be multiple authenticators for one user for one RP, and he should be able to deregister one of them. How can we do it in FIDO2. and does webauthn has any API to do this?" Answer: "There's no API for this in WebAuthn, since the RP just needs to delete the public key from the users table (or wherever the RP chooses to store users' public keys) to de-register the credential."*). In some cases, even the most rudimentary picture about FIDO2 deployment was lacking

(e.g., *"what is FIDO server, is it optional?"* or *"To implement FIDO2 in my web application, it's needed the FIDO2 server?"*). Clearly, developers need more comprehensive information and better instructions on FIDO2, and in Section 3.3 we report about the status-quo we found.

*Confusing technical details.* The FIDO2 specifications are not simple and developers are confronted with a number of options and responsibilities. This is also reflected in concrete questions by developers regarding security-relevant parameters (e.g., *"what does the 'user presence' and 'user verification' mean in CTAP?"*) and frustration with backwards compatibility to U2F (e.g., selection of the authenticator: *"Question: I am getting different attestation format in chrome and firefox browser."* Answer: *"This could happen with authenticators that support both CTAP1 (U2F) and CTAP2"*), which can lead to favoring an insecure-but-functioning solution over a secure one (e.g., avoid requesting an attestation in this case).

### 3.2 Insecure and Incomplete Libraries

We observed that developers resort for the integration of WebAuthn into their web applications to off-the-shelf solutions, like libraries. These libraries must provide core functionality and developers are responsible for enforcing the security policies and session management in an actual deployment. After analysing the top 10 WebAuthn libraries on GitHub, we found that they are incomplete, outdated, and have poor documentation. As an example, we found a case where the signature verification failed as a result of misreading the specification. If such insecure libraries are included in actual web applications, it could defeat the security of the system.

### 3.3 Developer Support and Education

The FIDO Alliance supports developers through articles, webinars, tutorials, and technical knowledge on how to implement and deploy FIDO2 Authentication. There are demos, proofs-of-concept, and seminars that are developed and presented by members of the Alliance, but we discovered that all these materials on realizing a FIDO2 server were outdated or non-existent. Further, W3C provides technical details on how to implement WebAuthn securely. According to the W3C specification, web application and framework developers should follow a roadmap that is comprised of nine sections, which, however, focus on functionality and do not include security- and privacy-related issues. A security reference document by the FIDO Alliance to cover such issues is currently in draft, but has not received an update since the beginning of 2018. Additionally, third-party sites like Duo Labs, Yubico, Microsoft, and Google also support developers through articles, demos, and proofs-of-concept. Lastly, the Mozilla foundation developer documentation helps the developers to understand how WebAuthn works. Unfortunately, their documentation is mostly related to very specific use-cases or tailored to their products, and does not give a guideline about security best-practices and pitfalls.

We find it troublesome that in our investigation, only poor documentation is provided for developers. The best documentation we found were blog articles by FIDO2 veterans, such as Yuriy Ackermann, or involved providers, like Yubico and Microsoft.

### 3.4 Security And Privacy Concerns

Lastly, we try to forecast some potential future security issues[1], which we derived from the observed discussions by developers.

*Enrollment from multiple devices.* Integration of FIDO2 can also disrupt the usual strategies for user enrollment and web application developers have to adapt. For instance, if a user registers using a device with a platform authenticator but would also like to use the service on another device with a different authenticator. Web apps have to support a secure enrollment of the additional device to the user account without allowing attackers to surreptitiously gain access to that account. This leads generally to the question about how to securely manage multiple authenticators per user.

*Secret backdoor access to accounts.* Since FIDO2 servers have to potentially register multiple authenticators per user, an attacker with write access to the account database could silently append a new authenticator, resulting in backdoors to user accounts without the need to compromise or replace the original credentials.

## 4 CONCLUSION

WebAuthn is a rapidly adopted solution for user authentication on the web. To not repeat the history of wide-spread insecure SSL/TLS configurations, insecure password storage, or cryptographic API misuse, we believe that our community needs to tackle developer issues with WebAuthn early on and devise adequate solutions and developer support. For instance, in the form of tools, fail-safe default libraries, or "golden guidelines." To raise awareness for this issue and hopefully engage our community, we report in this poster on the current state of studying the FIDO2 specifications, developers forums, and open-source libraries for potential developer issues and pitfalls when adopting FIDO2.

## REFERENCES


[1] Yuriy Ackermann. 2019. Introduction to WebAuthn API. https://medium.com/@herrjemand/introduction-to-webauthn-api-5fd1fb46c285

[2] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. 2013. An Empirical Study of Cryptographic Misuse in Android Applications. In *Proc. 20th ACM Conference on Computer and Communication Security (CCS '13)*. ACM.

[3] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. 2012. Why eve and mallory love android: an analysis of android SSL (in)security. In *Proc. 19th ACM Conference on Computer and Communication Security (CCS '12)*. ACM.

[4] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. 2012. The most dangerous code in the world: validating SSL certificates in non-browser software. In *Proc. 19th ACM Conference on Computer and Communication Security (CCS '12)*. ACM.

[5] Katharina Krombholz, Karoline Busse, Katharina Pfeffer, Matthew Smith, and Emanuel von Zezschwitz. 2019. "If HTTPS Were Secure, I Wouldn't Need 2FA" - End User and Administrator Mental Models of HTTPS. In *Proc. 40th IEEE Symposium on Security and Privacy (SP '19)*. IEEE Computer Society.

[6] Katharina Krombholz, Wilfried Mayer, Martin Schmiedecker, and Edgar Weippl. 2017. "I Have No Idea What I'm Doing": On the Usability of Deploying HTTPS. In *Proc. 25th USENIX Security Symposium (SEC' 17)*. USENIX Association.

[7] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. 2017. Why Do Developers Get Password Storage Wrong?: A Qualitative Usability Study. In *Proc. 24th ACM Conference on Computer and Communication Security (CCS '17)*. ACM.

[8] World Wide Web Consortium. 2019. Web Authentication: An API for accessing Public Key Credentials Level 1 — W3C Recommendation, 4 March 2019. Retrieved 05/07/2019 from https://www.w3.org/TR/webauthn/

[9] Glenn Wurster and P. C. van Oorschot. 2008. The Developer is the Enemy. In *Proc. 2008 New Security Paradigms Workshop (NSPW '08)*. ACM.


[1] There would be many more functional issues to predict, such as RPs that serve multiple origins.