

Blind software-assisted conformance and security assessment of FIDO2/WebAuthn implementations

Athanasios Vasileios Grammatopoulos^{1*}, Ilias Politis², Christos Xenakis¹

¹Systems Security Lab., University of Piraeus, Greece
avgrammatopoulos@ssl-unipi.gr, xenakis@unipi.gr

²InQbit Innovations SRL., Romania
ilias.politis@inqbit.io

Received: December 19, 2021; Accepted: May 18, 2022; Published: June 30, 2022

Abstract

With passwords being a problem in today's digital world, FIDO2 through WebAuthn brought an alternative password-less authentication model for web applications and services, which is more usable and secure than the legacy password-based systems. The adoption of WebAuthn standard is undoubtedly a step forward in improving and strengthening online services, however it may carry potential risks if not implemented correctly. To minimise the risk of leaving implementations vulnerable to attacks, a more systematic approach has to be followed for testing and securing emerging FIDO2 services. Towards this end, the paper proposes a novel tool for testing FIDO2/WebAuthn implementation's conformance, configuration and security by analysing the WebAuthn requests and emulating the client's WebAuthn responses. The proposed tool and associated tests aim towards empowering application developers and security auditors with the ability to effectively and quickly improve WebAuthn implementations by identifying and resolving flaws and security vulnerabilities in their password-less services. A detailed analysis of various commercial and open source WebAuthn services has been conducted, revealing common security weaknesses and faulty configuration, thus highlighting the significance of the proposed methodology.

Keywords: WebAuthn, FIDO2, Password-less, Authentication, Security, Assessment

1 Introduction

The use of passwords bears security risk for both users and organisations, hence to mitigate such risk popular password policies require the use of relatively lengthy and complex passwords, while others suggest the frequent change of old passwords. Although such policies may increase the security, they compromise the usability of the passwords. Evidently, these policies contribute to the users' tendency to use the same or similar passwords, by just slightly modifying them, across multiple websites and applications. Therefore, the leakage of a single password has the potential to compromise multiple accounts of a single user. The latter is a major security problem which is often exploited through email and website phishing [1, 2] or by cracking weak password hashes [3] leaked online through data breaches.

These disadvantages of password authentication methods led to the deployment of multi-factor authentication mechanisms and the development of alternative password-less authentication methods, such

as the Fast ID Online (FIDO) framework by the FIDO Alliance [4]. With FIDO, users can use secure authenticator devices instead of passwords and authenticate with applications and web services easily, quickly, and securely in a privacy preserving way [5]. Since the release of the first version of W3C's WebAuthn [6] in September 2019, which enabled FIDO2 to be integrate with the web browsers, FIDO has gained popularity and it is currently supported out-of-the-box by several platforms and websites. WebAuthn defines a way to allow web applications to request access and communicate with FIDO authenticator devices (hardware or software based) to provide FIDO2 registration and authentication services to their users. During the last years, due to the increase of the support by operating systems, FIDO became available to a wider range of developers. In the effort to eliminate the password problem, libraries, servers and tools are required to facilitate the adoption of FIDO2 and WebAuthn.

FIDO's goal is to bring easy and secure authentication, while mitigating the traditional password authentication problems. Some of the main advantages of FIDO through WebAuthn are: a) support for biometric authentication, b) mitigation of phishing attacks targeting credentials, c) strong authentication through elliptic curve public key cryptography and d) secure authentication across multiple services with a single authenticator. Although FIDO is based on an easy-to-understand concept (i.e., that of challenge and response), the protocol defines a large ecosystem of components across multiple domains. This complexity which is usually hidden from developers behind an API, (i.e., WebAuthn JavaScript API), often makes FIDO difficult for developers to understand and adopt correctly within their applications and services.

There is a need to ensure the correct implementation of the WebAuthn standard on current and future web based services and improve the security of critical components [7]. The motivation for developing a novel methodology for assessing the conformance and security of FIDO2/WebAuthn implementations springs from the need to reduce the developer's frustration and improve their understanding of the standard, so that fewer mistakes are made during the implementation and/or the configuration of a WebAuthn service. There are several complementary approaches to achieve this end goal. Firstly, the production of in-depth explanatory guides, videos and the development of code playgrounds will lay the basis for more application developers to incorporate WebAuthn in their designs. Additionally, the development of new tools to assist developers during the implementation from both the debugging and the testing side should reduce the occurrence of security flaws in WebAuthn services. Enabling the developers to monitor the communication between WebAuthn client and the server, while allowing them to emulate various authenticator devices, should allow them to identify and address such issues which otherwise remain obscure. Finally, an efficient and user-friendly methodology for assessing new implementations and minimise security faults in production is required. To address all these requirements, new tools and efficient solutions capable to empower pen-testers to conduct security assessment of emerging applications and services is required.

The aim of this paper is to introduce a novel methodology to capture and analyze FIDO2/WebAuthn requests and responses and propose a novel tool to implement this methodology, thus enabling the application and services development community with the to deep inspect WebAuthn traffic. To our knowledge we are the first to present such a methodology for evaluating and testing WebAuthn implementations. Furthermore, we emulated a number of components of the FIDO2 and WebAuthn specification (i.e. FIDO2 Authenticator, WebAuthn Javascript methods) inside the browser environment to overcome the limitations of the web context sandbox, allowing us to put our methodology into practice and released an open-source tool [8] that analyses the WebAuthn requests and conducts assessment tests automatically. The scope of the proposed solution includes:

- the facilitation of instant inspection of essential for the validation of the information provided by the FIDO2 server WebAuthn parameters, passed to the authentication;
- the provision of deep decoding and analysis of the authenticator's WebAuthn response, thus en-

- enabling faster debugging of WebAuthn processes at the browser level;
- the featuring of a novel virtual FIDO2 authenticator, which allows for platform independent WebAuthn response simulation for implementation testing;
- a WebAuthn playground with the ability to generate custom WebAuthn requests for experimentation and familiarisation with the WebAuthn API;
- a modular and extensible web tool, which can facilitate the development of new features and security tests;

Our tool is available on GitHub and as it was developed in components, it can be extended even further. To prove that, we extended our tools' supported algorithms to include the first fully functional WebAuthn post quantum signing algorithm (using FALCON), which we used to develop and test a custom quantum resistant FIDO2 server.

This paper is an extended version of a paper presented in ARES 2021 [9], including additional analysis of the proposed methodology, in-depth look into the developed tool, introduction of the tool's new testing features and enhancements. Furthermore, this paper takes one step further and evaluates FIDO2/WebAuthn implementations using the proposed tool.

The rest of the paper is organized as follows. Section 2 includes an overview of the FIDO2 and WebAuthn protocols and their role in the web application market as well as other related work on the field. Section 3 analyses the proposed WebAuthn Analyser tool providing detailed of the modules and components it incorporates. The testing and evaluation sessions which demonstrate the capabilities and added value of the proposed implementation is included in Section 4, while Section 5 concludes the paper and highlights the future work.

2 Background

2.1 FIDO2/WebAuthn overview

FIDO2 at its core uses a challenge-response scheme based on public key cryptography as depicted in Figure 1. The server (i.e., relying party) prepares a challenge in the form of a random value and forwards it to the client. The client must sign this challenge with a private key and send the signature back to the server, to prove its identity. Then, the server needs to verify the authenticity of the signature given by the client using the public key of the user that the client is claiming to be. Prior to the execution of the challenge response scheme, the server would need to be in possession of the user's public key. This simple conception ensures the security of the scheme, the compatibility, and ease of use of FIDO.

FIDO's ecosystem spans from secure hardware authenticator devices to full scale cloud-based services, embedding a wide range of technologies. Looking at the client side, FIDO's Client to Authenticator Protocols (CTAP1 and CTAP2) define how devices can communicate with FIDO compatible authenticators. FIDO Universal Authentication Framework (UAF) describes how a FIDO UAF server should communicate with client devices (usually a mobile phone with a fingerprint sensor) to offer password-less authentication using biometrics. Lastly, the more recent FIDO2, improves the older Universal 2nd Factor (U2F) authentication, and through WebAuthn JavaScript API and server side WebAuthn libraries or services, brings FIDO to the web.

2.1.1 FIDO2 in web applications

Through the WebAuthn JavaScript API, web applications are able to request from the browser (WebAuthn client) and the underlying operating system credentials creation (i.e., public key pair generation),

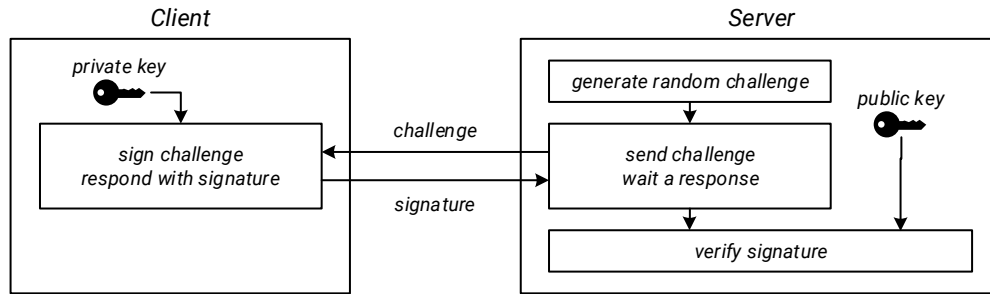


Figure 1: Client authentication using challenge-response and public-private key cryptography

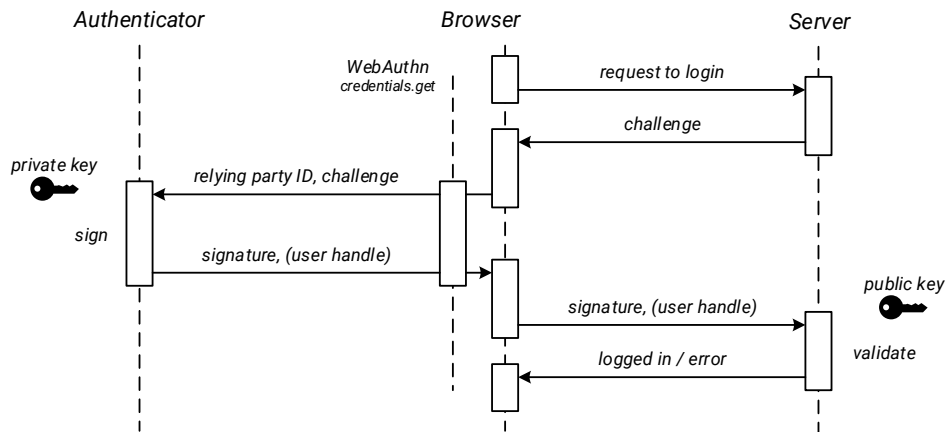


Figure 2: Simplified FIDO2/WebAuthn client authentication flow using `credentials.get`

as well as credentials retrieval (i.e., proof of secret key possession). The credentials creation method, which is accessible through the JavaScript `window.navigator.credentials.create` method and the public key options, defined at the WebAuthn specifications [10], allow the creation of asymmetric cryptography keys (e.g. ECDSA key-pairs). These keys are bind to the caller web application’s domain (relying party id) and a user identifier (user handle) is linking the credentials with an account. Moreover, through the corresponding credential get method, which is accessible through the JavaScript `window.navigator.credentials.get` and its public key options, the web applications can verify the client’s possession of previously created credentials (key-pairs) by requesting the generation of a random challenge’s signature. Thus, the identity of a user can be verified through a challenge-response scheme, as illustrated in Figure 2.

2.1.2 Authentication flow

A typical use case of FIDO2, using the above mentioned JavaScript methods, is an online password-less authentication. That is, the secure login of a user into a website without the use of a secret password. Figure 2 depicts the authentication process as a diagram. To start the process, the user loads the website through a WebAuthn compatible browser and selects to login password-less. The website’s back-end generates a random high entropy challenge (usually 128 bits or more, as suggested by the specification) and communicates it with the website’s front-end (the user’s browser). The front-end is then able to invoke the WebAuthn `window.navigator.credentials.get` JavaScript method to request from an

authenticator device to sign the challenge. The browser then sends the challenge along with the website's domain name (relying party id) and some more data (list of accepted credentials, list of excluded credentials, extensions etc.) to the available authenticator devices for signing, assuming that one of those authenticators possesses a key pair for the website in question (or one of the key pairs requested). Depending on the system and the support, the browser may contact the authenticator devices directly through the CTAP protocol [11] or call custom methods of OS specific WebAuthn Client implementations (such as Android's FIDO2 API [12]). Eventually, the browser gets a response that then parses and forwards to the corresponding JavaScript handler. The response includes the identifier of the key used for generating the signature (credentials id), the actual signature, the data structure used to generate the signature as reported by the authenticator, a user identifier (user handle), a signature counter and several flags. Then the website's front-end forwards the response data to the relying party's back-end (the web application server) for verification. Upon successful verification, the user is logged in and his session (usually implemented using cookies) is updated.

The described use case assumes that the authenticator supports resident keys (i.e., discoverable credentials) and can report back the user identifier (user handle). To support older U2F authenticators, or to avoid storing information on the authenticator device (leveraging key wrapping techniques), web pages may store previously used account identifiers as cookies or at the local storage of the browser and include them at the initial challenge creation request. This way the web pages can provide the account's identifier to the server enabling the latter to return along with the challenge a list of credential IDs registered for this account. This list of credentials could then be added on the invocation of the `WebAuthn.window.navigator.credentials.get`. A similar process can be used for second factor authentication flows, as the user's identifier should already be known through the active session. This later method could also be used when there is a need to re-authenticate an already authenticated user, commonly used to renew sessions of returning users, by requesting a fingerprint or PIN authentication when relaunching a mobile application (usually found on banking related applications).

A true password-less authentication without the need to provide any username or password requires information to be stored on the authenticator device, which may increase the cost of the authenticator, limit the maximum number of keys that can be generated and need key management utilities to be able to remove stored keys not needed any more. On the other hand, the password-less authentication, which requires the knowledge of an account's identifier (e.g., a username) does not share such limitations, since the information can be wrapped securely and stored at the relying party's server, through server-side credential storage modality.

2.1.3 Registration flow

Prior to FIDO2's authentication process, clients would need to register an authenticator device with the relying party server. During the registration process, shown in Figure 3, the client can generate a public-private key pair supported by the relying party server and send the public key along with its credentials identifier to the server, to be saved and linked with the client's account. To register an authenticator, the client should already be logged into the website, hence, a session linked to an account would already be set up. The registration process starts with a request for credentials creation, created by the WebAuthn client, for a challenge generation by the server. The server will return a random challenge and a user handle, linked to the user's account, a list of supported credentials types (e.g., ECDSA or RSASSA-PSS or RSASSA-PKCS1-v1.5), authenticator filtering criteria (e.g., allow only external authenticators), a list of already registered credentials (to exclude already registered authenticators) and the server's preference for authenticator attestation (to request information about the used authenticator device). These parameters can be used as options when calling the `window.navigator.credentials.create` JavaScript method (as defined by the WebAuthn specification) to request credentials creation. After the credentials

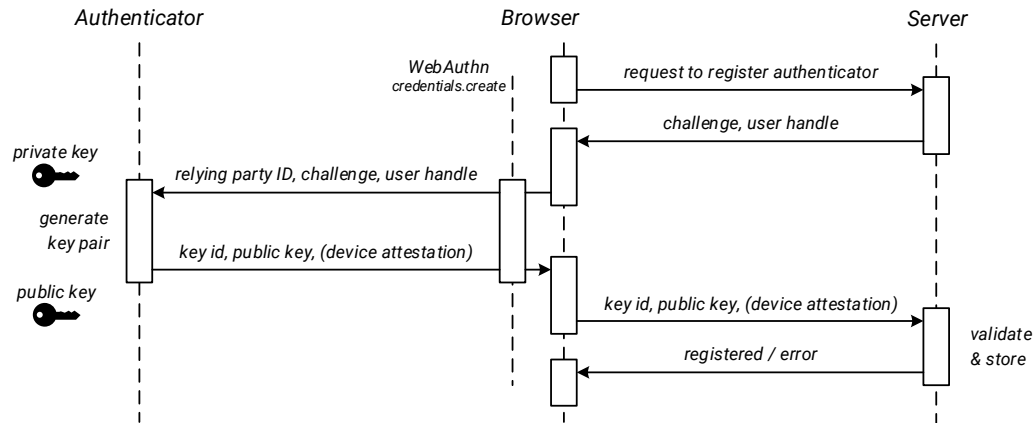


Figure 3: Simplified FIDO2/WebAuthn authenticator device registration flow using `credentials.create`

creation procedure is handled by an authenticator device successfully, the browser will return the created credentials to the appropriate JavaScript handler defined by the method caller. The response includes the generated credentials identifier, the generated public key, the challenge generated by the server and device attestation data (e.g., a device certificate). The received data are forwarded to the relying party server. Upon receiving the data, the server will first validate the provided information and then store at least the credentials identifier and the public key under the account the challenge was generated.

2.1.4 Security mechanics

One of the main features of WebAuthn that makes it unique with respect to other authentication methods is that it is resistant to phishing attacks. By design its methods do not allow cross domain credentials access unless, they are created under the same top domain. Thus fraudulent websites are not able request authentication for another legitimate website. This is achieved through the web browser's validation of the relying party id of the request, which should be the domain name of the website (note that authenticator devices should bind the credentials with the relying party id and the user handle).

To ensure the integrity and the confidentiality of the processes described above, web browsers expose the WebAuthn API only under secure context (web pages loaded under HTTPS), with the exception of "localhost" origins that are commonly used for development purposes. By requiring HTTPS, the browser can assure the authenticity of the server (by validating the server's certificate) and thus mitigate man-in-the-middle attacks at the network traffic level. In simple terms, to secure the schema, FIDO builds a trusted communication channel between the relying party and authenticator device.

Form the relying party's side, in order to validate the correct execution of the authentication process, the server has to verify a number of information returned on the response apart from just the signature. First, the challenge returned should match the one generated by the server. To ensure the use of a correct key used, the credential identifier returned should be already registered to the user account in question and bind under the user identifier received (if one was returned). In the case of challenge generation for a specific user (U2F or second factor), the user id should be temporary stored upon the challenge generation and cross checked after the response. Additionally, since the challenges are generated for use within a limited time frame, the server should expire unused challenges. To protect users against cloned authenticator devices, the server may also save for each credentials the signature counter and ensure that for every authentication procedure this number increases. Lastly, the server may check the flags returned

by the authenticator and check, for example, whether the user interacted with the authenticator device (which may be required for security reasons).

Depending on the relying party's policies, the authenticator's attestation data can also be requested and used to assess the authenticator device information provided by the FIDO metadata service. For example, a server may try to identify whether an authenticator device is among the white-listed devices supported by the service or furthermore save the model information of the device to unregister authenticators if they were to be found vulnerable in the future (e.g., due to an exploit or leak of the manufacture's master keys).

FIDO2/WebAuthn servers must take into consideration several things in order to correctly validate a response. That complexity makes it difficult to develop FIDO2 services without the use of a FIDO2 server or a FIDO2/WebAuthn library. However, even with the use of 3rd party software to handle the verification process, still, the service may be left vulnerable to attacks due to poor or faulty configuration. To give some examples, in many cases developers don't configure the services not to use deprecated algorithms, and don't handle invalid signature counter errors appropriately (by disabling the authenticator device). Hence it is essential for the developers to deeply understand how FIDO2 and WebAuthn work, to deploy such solutions even if a 3rd party library or a server is handling the registration and authentication flows.

2.1.5 Types of authenticator devices

FIDO authenticators can be categorized based on their type into two categories, platform authenticators and cross-platform authenticators. Cross platform authenticators are external devices that connect with the system through USB, NFC, or Bluetooth (e.g., USB Keys or NFC Keys) and communicate through FIDO's CTAP1 and CTAP2 protocols. On the other hand, platform authenticators are embedded into the system (e.g., Android internal authenticator, Windows Hello authenticator) and may communicate with applications directly through the underlined system's calls and libraries (e.g., Microsoft WebAuthN Win32 headers [13]). Independent of the type of an authenticator, the device should be able to protect the private keys so that they cannot be extracted by an adversary that may have physical access to it.

Another practical characteristic that we can use to categorize the authenticator devices is the available methods they support to verify the user presence. Although the authenticator itself is a way the user to prove possession of the authenticator it self, in many cases the authenticator will have to verify the user's identity first before executing an FIDO/WebAuthn operation. Many authenticators (usually mobile or laptop devices) leverage access to bio-metrics sensors (e.g., fingerprint, face recognition, iris scan) to securely verify the user. Other simpler (usually USB cross-platform) authenticator devices features just a button, which the user press to verify its presence. To mitigate the risk of unauthorized use of such a FIDO2 device, an operation system may also ask the user for a PIN to authenticate him.

2.1.6 Authenticator device attestation

The WebAuthn requirement of a secure connection (through HTTPS) not only protects the information exchanged between the client and the server but also verifies the authenticity of the server (managed by the relying party), through the trusted certificate issued to the domain name used by the service. In a similar way, depending on the application needs of a WebAuthn deployment, the relying party may want to verify that the client's authenticator device is compliant with its policies. For example, the relying party may have to verify before registering a new authenticator device, that this new device has the appropriate security level required by the service's security policy. To achieve this, the relying party server may request additional attestation information from the authenticator device, during the registration phase, and assess them before finalising the registration process. The returned attestation statement would ideally prove the original identity of the authenticator device or verify the trustworthiness of the

device. Depending on the attestation conveyance method, the authenticator may return its Authenticator Attestation GUID (AAGUID), exposing the authenticator's maker and model, as well as provide a way to verify its authenticity (e.g., by providing a certificate). Furthermore, using an authenticator's AAGUID, relying parties may query the FIDO Alliance Metadata Service (MDS) [14, 15], to get more information about the authenticator device (e.g., authenticator security level, available user verification methods and combinations) and verify any attestation certificate returned.

Nevertheless, such an attestation of the authenticator device may expose too much information (e.g., Authenticator Model and Number), which may be used to track a user between multiple services. For this reason, an attestation conveyance preference can be defined, stating the relying party's preference to, no attestation ("none"), anonymised attestation through a CA ("indirect"), or authenticator generated attestation ("direct" or "enterprise"). Thus, authenticator devices respond with different attestation responses based on the requested preference and their supported attestations or ignore the suggested attestation opting for user privacy. The relying party from its side, may have to reject the authenticator registration if the needed attestation statement returned is not supported or the given or retrieved information does not satisfy its policies (e.g., due to failure of verifying any given certificate).

To allow the extension of the available attestation information, plug-able Attestation Statement Formats are supported by WebAuthn. Due to the nature of this scheme, the implementation of a relying party may not support all of the attestation statement formats. The latest WebAuthn standard [16] describes the following attestation statement formats: None, Packed, TPM, Android Key, Android SafetyNet, FIDO U2F and Apple Anonymous. The corresponding Attestation Statement Format Identifier values are listed and maintained in the appropriate registry by IANA [17].

2.2 Related work

Due to the relative recent standardization of WebAuthn, the scientific literature is lacking of research papers and studies on the field of developing and testing FIDO2/WebAuthn services implementations. FIDO Alliance offers a certification program [18] for FIDO2 servers, focusing on conformance and self-validation to ensure interoperability testing rather than their security. This is not the case for authenticator devices though, as the equivalent certification [19] not only focused on the conformance but also on the security level of the device.

The most relevant documentation of research activities related to WebAuthn services implementations are either in the form of websites with WebAuthn demo implementations, or "offline" decoder tools. Demo WebAuthn websites [20] mostly showcase the FIDO2/WebAuthn processes, while others reveal parts of their own parameters or the returned response (as the interactive WebAuthn debugger [21] and the work in [22, 23]). Such demo applications, can be used to test the client-side implementations, demonstrating that the FIDO2 technology is currently available, rather than serve as tools designed for developers to test or debug server-side FIDO2 implementations.

Apart from these demos, developer tools for analysing WebAuthn traffic (usually called "debuggers") can be found online. Such tools, take as an input a JSON version of the whole WebAuthn response or just a Base64 representation of the values and decode it (for example the WebAuthn Previewer [24, 25] or the fido2viewer [26]), which can be used by developers to unpack, decode or even validate WebAuthn traffic, though the capturing and analysis has to be done manually.

The proposed solution which is inspired by the above-mentioned demos and the analysers, uses JavaScript to connect all these functionalities and many more directly to third party external implementations. This paper can be considered as the first systematic research work on the assessment of WebAuthn services.

3 WebAuthn Analyzer tool

Following the overview of the FIDO2/WebAuthn operations, it is evident that the appropriate implementation of the standard is not trivial and the application developers could benefit from effective and easy-to-use tools that would shed light into the functionalities of WebAuthn. Since the adaptation of standards does not always follow the appropriate paths, it results in poor implementations with various vulnerabilities. It is important to facilitate a smooth and correct adoption of standard associated with critical modules [7]. For the purposes of this work, the standard involves the authentication of users considering the needs of the developers.

The invocation of credential creation and get JavaScript methods requires the correct configuration of the public key options by the web application. In most cases these options are automatically generated by the web application's back-end. Usually a specialised FIDO/WebAuthn library or server without or minimum configuration from the developers and the service administrators. The generated options are passed to the application's front-end, which uses them accordingly to contact the authenticator devices. Since the output and input data are generated and validated at the back-end, it is quite difficult to monitor, debug or assess the correct execution of the process at the client side (front-end). Due to this automatization and simplification, quite often developers who may not be specialised in WebAuthn, do not have a clear view and understanding of the information passed to and from the client-side JavaScript methods as they see the system as a black box. This lack of understanding and obscurity of WebAuthn may cause miss-configuration problems producing unintended bugs and even security flaws which may be hard to identify and resolve.

The current processes for debugging and testing WebAuthn are complicated and time consuming as they require to retrieve the data given by the server or returned by the authenticator. Furthermore, they need manual effort to unpack the encoded values which requires deep understanding of the WebAuthn specification. The same complexity can be observed during the review of the information passed from the relying party server to validate the correct configuration and implementation of the FIDO2 back-end as well as, the compliance with the WebAuthn. This lack of tools for aiding the developers in conducting essential analysis of the WebAuthn traffic, leave the implementations prone to miss-configuration errors, security vulnerabilities or even policy violations.

Moreover, the lack of appropriate testing tools for developers and security testers makes it difficult to ensure the trustworthiness of FIDO services. To avoid insecure implementations or vulnerable miss-configuration of FIDO2/WebAuthn as well as, to aid developers and increase their productivity, more effective and easy-to-use developer and penetration testing tools targeting WebAuthn are needed. In this work, an innovative methodology used to capture and analyse FIDO2/WebAuthn requests and responses is described. The developed solutions provide the developers with the right tools needed to assess WebAuthn implementations. Specifically, the proposed solution:

- sets up a WebAuthn playground with the ability to generate custom WebAuthn requests (for experimentation and getting familiar with WebAuthn API);
- enables instant inspection of the WebAuthn parameters passed to the authentication (essential for validation of the information given by the FIDO2 server);
- allows the deep decoding and analysis of the WebAuthn response from the authenticators (facilitating faster debugging of WebAuthn processes at the browser level);
- features a novel virtual FIDO2 authenticator (to allow for platform independent WebAuthn response simulation for implementation testing);

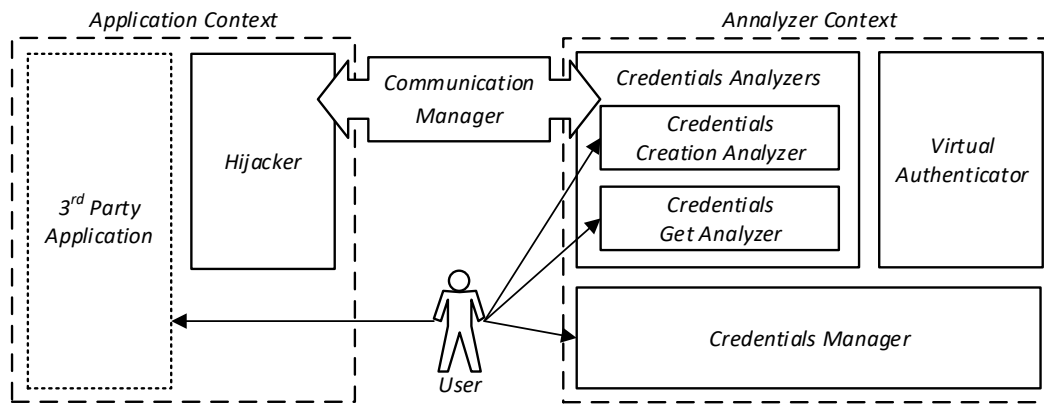


Figure 4: Main modules of WebDevAuthn inside each particular context, indicating user interaction points

- provides the appropriate tools to experiment with new cryptographic algorithms without the need of specialised hardware authenticators;
- provides an automated assessment of the passed WebAuthn parameters highlighting invalid or insecure configurations and compliance to the standard (important for identifying common mis-configuration and security flaws);
- provides various server testing methods to emulate attacks or malicious behaviour (needed to assess WebAuthn implementations);

This section describes the web tool implemented to capture FIDO2/WebAuthn requests and responses, for analysis and human inspection, as well as for conducting several conformance and security tests on the server. The proposed implementation enables users to assess the conformance and security of a WebAuthn service, get familiar with the WebAuthn JavaScript API or test and analyze responses from FIDO authenticators. The WebDevAuthn tool consists of the following main modules, also shown in Figure 4, which will be described in more details in the following sections:

- the **Hijacker** in the form of a browser extension or a website script for development servers,
- the **Communication Manager**, used to pipe data between the hijacker and the analyzer,
- the **Credentials Analyzers** that handles and analyses credentials requests and responses, composed by:
 - the **Credentials Creation Analyzer**, responsible for analyzing credentials creation requests and responses
 - and the **Credentials Get Analyzer**, responsible for analyzing credentials get requests and responses,
- the **Credentials Manager**, tasked to manage all the key information credentials storage used by the rest of the tool's modules,
- lastly, the **Virtual Authentication**, able to create and retrieve custom virtual keys for cross domain use and authenticator simulation.

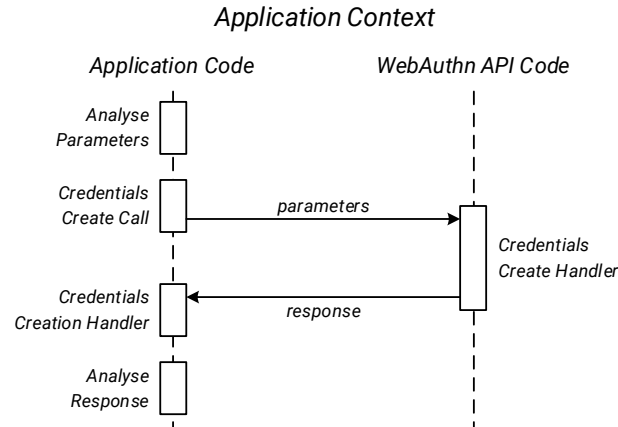


Figure 5: Code flow of an application calling the credentials create method. To inspect the parameters or the response, custom code will have to be deployed on the application’s main code as the WebAuthn API code is implemented by the browser.

3.1 Hijacker

The Hijacker module is responsible for injecting code into the targeted web application and intercepting credentials create and credentials get WebAuthn method calls. If the Hijacker is configured to use the virtual authenticator to handle the request, the intercepted WebAuthn calls parameters will be packed and forwarded directly to the credentials creations or get analyser module depending on their type and wait for a response to complete the request. Otherwise, if it is configured to use a real authenticator device, the parameters will first be forwarded to the original WebAuthn methods of the browser and then packed and passed along with the returned response to the appropriate analyser. For the proposed implementation the Hijacker can be deployed as a browser extension (featuring a control panel user interface) or as a configurable JavaScript script included on the web application (either through the web browser’s developer tools JavaScript console or as a web page script). To hijack the WebAuthn methods, they are overwritten with custom code. Two instances of the original methods are stored in two private variables so as to be able to call them when needed. The Hijacker simulates the behavior of the original methods (returning a promise) and tries to format the response appropriately, so that no change in the application code is needed. Figure 5 shows a normal WebAuthn call flow, while the same flow with the Hijacker injected is shown in Figure 6.

Using the Hijacker, the developers can capture the WebAuthn requests and responses without having to manually edit any part of their application code, thus making the capturing easy and practical by automating, simplifying, and speeding up the process. Furthermore, the captured traffic is now being forwarded to the appropriate analyser module where a more advance inspection can take place. We also have to note that by loading the hijacker as a browser extension, it works out of the box with almost any WebAuthn enabled application.

3.2 Communication Manager

The Communication Manager module is responsible for setting up a cross-origin communication channel between the hijacker and the credential creation or get analysers. Additionally, it is responsible to encode the data appropriately before transmitting them and decoding them upon reception. To set up a connection, the Communication Manager first creates a new instance of the appropriate credentials

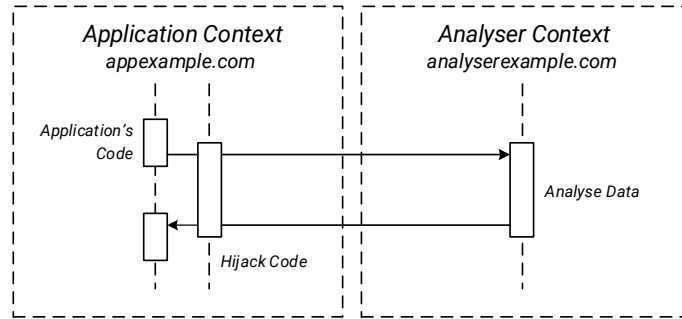


Figure 6: Analyze data through method hijacking code injection and cross domain web page communication

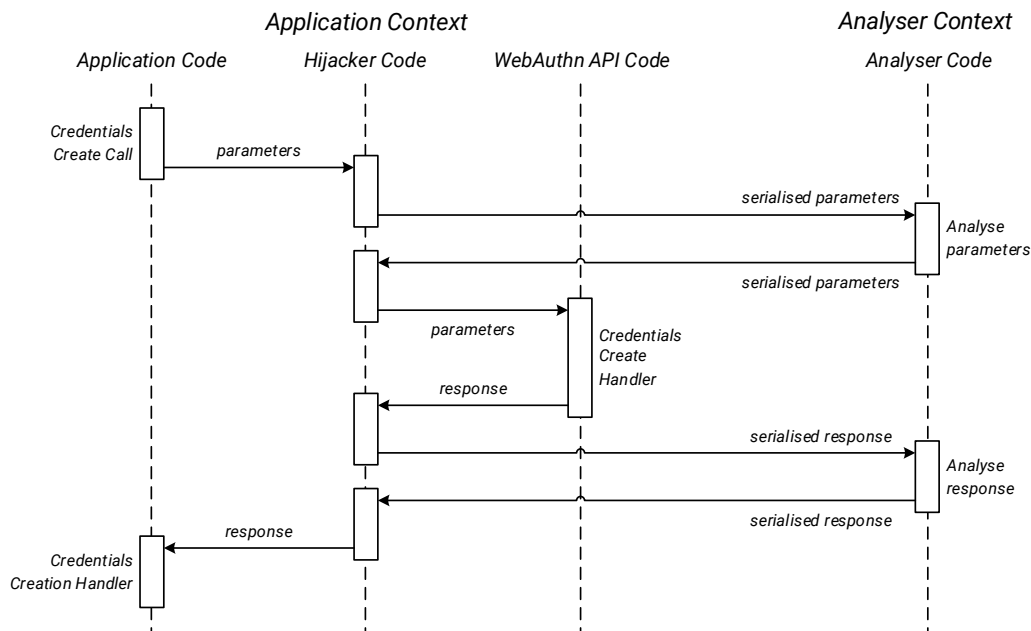


Figure 7: Code flow of an application calling the credentials create method through hijacking. The analysis of the traffic can be implemented independently from the application code flow on a separate website

analyser module by opening a new window, as a conclusion the user may be prompt by the browser to allow popup windows. Following the instance creation, the communication channel is created based on JavaScript cross-origin communication mechanics (message posting [27]). Custom serialisation and de-serialisation mechanics (based on JSON) are used to exchange the data between the modules. The overall flow of the information from the Hijacker to one of the credentials analyser modules through the channel set up by the Communication Manager is shown on Figure 7.

The Communication Manager module enables the transmission of the captured traffic outside the context of the application. This allows for the development of a custom analyser application and user interface to facilitate the analysis of the captured traffic separated from the application code. Furthermore, by separating the analyser context from the application context, the same analyser code can be used to analyse the traffic from various WebAuthn applications. Essentially for the developers this allows the

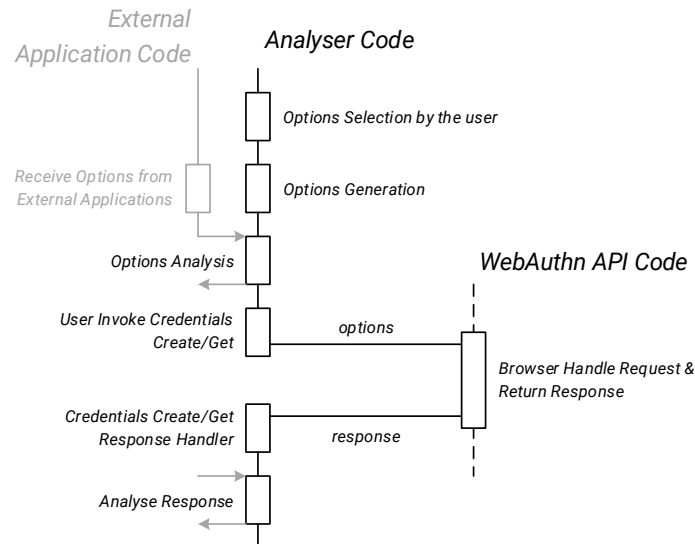


Figure 8: Analyser's code flow for handling credentials create & get requests

automatically extraction of the traffic for inspection and analysis in a separate external tool.

3.3 Credentials Analyzers modules

The Credentials Creation Analyser and the Credentials Get Analyser modules are the main modules of the tool responsible for analysing the server generated WebAuthn parameters (response) and the authenticator returned data (response), as shown in Figure 8. The analysis includes the assessment of the given parameters (request), the unpacking of the response and the display of the information in human readable formats.

Firstly, the supplied WebAuthn options (public key credentials creation [28] or get options [29]) are supplied to the appropriate Credentials Analyzer module and are been decoded based on the WebAuthn specification. The Analyzer then prints all the options information (e.g., the relying party id, the challenge) in a human readable format. Then the Analyzer check whether the passed options are following the WebAuthn standard and prints in the note section any deviation. Furthermore, the Analysers will look for invalid options or insecure configurations. On top of that, tests are conducted on various information such as the challenge to check its randomness and the user handle to determined if it leaks private information (e.g., an email or a phone number is used). Secondly, the Analyzer modules will unpack and decode any authenticator response into a human-readable form. For credential creation responses, they are even able to unpack some attestation formats of the device (e.g., allow the export of the device certificate) and even return the generated credentials public key.

Such an analysis and assessment of the request parameters can instantly pinpoint any invalid implementation of configuration and also give an insight into what exactly the web application is going to request from the WebAuthn method. The Analyzer modules are not only helpful for inspecting the request parameters, as they can also play an even more essential role in analyzing the request responses. Through the analysis of responses, they could process the packed information and decode them into a human-readable form. This allows developers to deep inspect the WebAuthn traffic without editing their application code, in an easy and fast way. Moreover, these modules provide an automated advanced analysis (e.g., values decoding and unpacking) and assessment of the request (e.g., conformance testing of the parameters) available for almost every WebAuthn application, which otherwise was not available

to the developers and penetration testers.

3.3.1 Credentials Creation Analyzer module

The Credentials Creation Analyzer module presents a friendly user interface, featuring commonly used interface elements such as text inputs, drop-downs, tabs etc., through which users can craft custom WebAuthn credentials creation request options. The crafted request options are printed on the page and could be analyzed before forwarding them through the `WebAuthn.navigation.credentials.create` method invocation. After the successful execution of the credential's creation method, the credentials response object returned is analyzed by the tool and it is presented to the user. The analysis of the response includes the unpacking and decoding of packed and encoded fields respectively, to human readable formats. The analysis of both the credentials creation options and the credentials response object can be used by developers to understand how FIDO2/WebAuthn credentials creation and authenticator device registration works internally. Additionally, developers can craft specific creation options for testing a particular authenticator or how a particular system handles the requests. In this concept, developers may also inspect the analyzed response of a specific authenticator and possibly identify problems or incompatibilities (e.g., identify authenticator's attestation format that may not be supported by their backend FIDO2/WebAuthn implementation).

In collaboration with the Hijacker, the module can also let developers analyse request coming from 3rd party websites. During this process, the module is able to conduct automatic conformance testing on the given request options to identify implementation problems such as invalid use of the API, insecure configurations and private information leakage.

3.3.2 Credentials Get Analyzer module

The Credentials Get Analyzer module follows the same approach to the credential's creation page, as shown in Figure 8 for both modules. This module features a similar interface through which users may craft custom WebAuthn credentials get request options according to their needs. Similarly to Credentials Creation module, the generated options are printed on the web page in a human friendly format for inspection by the user. These options can then be used to invoke the `WebAuthn.navigation.credentials.get` method. The credentials response object returned by the browser is analyzed by the tool and it is presented to the user, following the successful retrieval of the credentials get assertion. The in-depth analysis of the response includes the unpacking and decoding of packed and encoded fields into human readable formats for ease of use (e.g., expose the authenticator counter).

The proposed tool's credentials get module's functionalities can be used by developers to understand the FIDO2/WebAuthn credential possession assertion and user authentication process, as the proposed tool exposes all the information exchanged between a relying party and an authenticator device. Additionally, the tool lets developers tweak the credentials get options and live test their authenticator devices as well as the underlying client system's behaviour.

3.4 Credentials Manager

The Credentials Manager module is responsible for the storage, retrieves and management of credentials between the analyzer's modules. It lets the user view the information of the credentials generated through the tool and allows the user to delete information of credentials that are not needed any more. Furthermore, it gives access and shows the information of the resident keys stored inside the virtual authenticator device, which can be used for debugging purposes.

The tool stores various credentials-related information, as shown in Table 1. The `User Handle` is the identifier linked to the user's account, defined by the relying party server on the passed parameters.

Table 1: Authenticator registered key-pair information logged by Credentials Manager

Key	Value	Type
User Handle	bNb60Cv3o8PCgq-HtHHxp2ssYlCQw8ttgLi1e0ElpPs	Base64
KeyID	GramThanos_aDFQfGhSwub0svkKfvE_ebfbho-Xh0_MirRuDsUnxB5mSAkTGGcSxu8FD5Zx8f0jPPWrFYP7SCXz-Y2ahcTKK_MrwyCqeuJcAmokC8GbqePUttNSR9hFUnsT_oateG1vx0x5iyLDtHtTv5B-8ps-DVPcQsIcr9GMRZw3slzjFFvs_Ipwr_enk0YSKLU0GW2U0Ues_3FPcqEggMZ4AQ5qHMkGpBhAL1VksRid1YyeVDqToICaBzwVHW3rPkow30bmrKYzcBnHeciXVr-eN1E-_uS1Yy11vhv2d55sAA5NXMzBKsPUU1TE-ySNfamK3COFBtPpUHML73uw34S5hVt3s0mQW68kZL6FbxWgedF1NbJutGcIOo	Base64
Origin	https://webauthn.ddns.net	URL
Date Created	2021-09-19T18:44:10.547Z	Date
Authenticator	virtual	Text

Code 1: Decrypted credentials information wrapped inside a KeyID value

```
{
  "v": 1, // Version
  "r": "webauthn.ddns.net", // Origin
  "u": "bNb60Cv3o8PCgq-HtHHxp2ssYlCQw8ttgLi1e0ElpPs", // User Handle
  "c": "ktrkjbtb", // Date (encoded)
  "a": -7, // Algorithm code
  "k": { // Formatted Private Key
    "x": "-XzBP4uT5P-iXMDgUCtFtJVfw4VB6ZYzERYyINnDqU8",
    "y": "DZ709D2ovj8G2rB5dVTpZp8XBE4v9jA1TKTLA7NIIbk",
    "crv": "P-256",
    "ext": true,
    "kty": "EC",
    "key_ops": ["verify"]
  }
}
```

The KeyID, is the identifier of the created public-private key pair, returned by the authenticator, and depending on the authenticator implementation, it may be need to be included for the credentials get options generation. The manager also saves the Origin value so that the credentials can be retrieved based on the request origin. Furthermore, the credentials creation date is saved of helping the user distinguish them. This credential information is saved on the browser's local Storage and are retrieved by JavaScript when needed.

For credentials generated by the virtual authenticator, the Credentials Manager module also allows for the decryption of the KeyID values and thus the recovery of the public-private key pair info secure wrapped inside, as shown in Code 1.

3.5 Virtual Authenticator

We developed on novel virtual (software-based) authenticator device scripted in Javascript, embedded inside our tool, in order to allow the creation of custom credentials and WebAuthn response for requests originating outside the domain of our tool. In this way, we bypassed the security restrictions set up by the browser and the underlining system, both at the software (e.g., the Operating System) and hardware level (e.g., a USB authenticator) by simulating all the process from WebAuthn browser API down to the FIDO authenticator. Furthermore, our software authenticator gives us full control of its functionalities, and access to all the information generated, stored, or transmitted (e.g., public-private key pairs) allowing us to tamper with them and conduct custom tests.

We developed the authenticator to take as generate valid WebAuthn responses so that it could be used

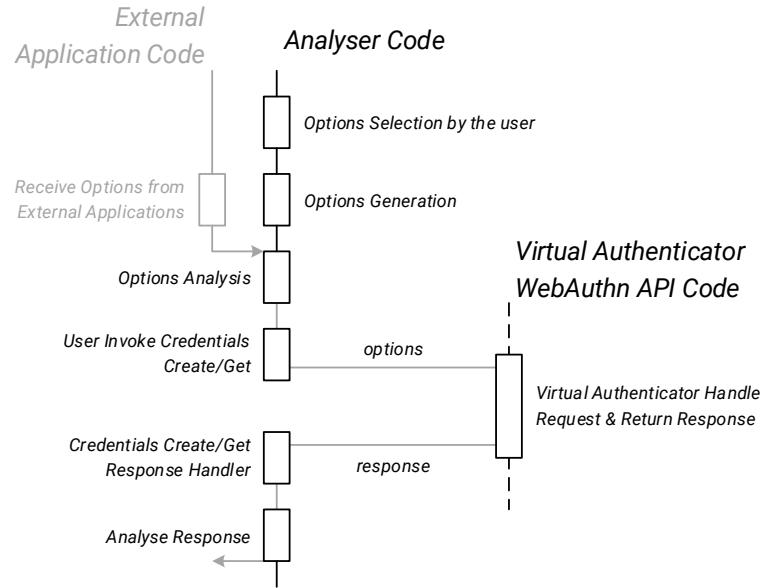


Figure 9: Analyzer’s code flow for handling credentials create & get modules using an embedded virtual authenticator

with 3rd party FIDO2/WebAuthn services (password-less registration and/or authentication services) as if they were crafted by a WebAuthn compatible web browser and a compliant FIDO authenticator device. Such a flow of information sourced from a 3rd party service is shown in Figure 9 where our tool’s virtual authenticator is used to reply to the WebAuthn requests.

The virtual authenticator device runs seemingly on our analyzer’s web page context provided that the browser supported the Web Cryptography API [30] used to conduct most of its cryptographic operations, and thus, the proposed implementation can run on any modern browser. The use of the web browser’s cryptography API allows the fast and secure operation of the authenticator without any noticeable delay.

Our virtual authenticator was primary developed to facilitate the conduction of tests using custom WebAuthn responses, though due to its portability, it can also be used to extend the FIDO2 attack vector as it can be used for malicious purposes. For example, our implementation can be used as a proof-of-concept (PoC) to generate and bind additional custom authenticator devices under a victim’s account, after exploiting a cross-site-scripting (XSS) vulnerability on a web application, essential setting up a backdoor to the account.

In the following sections, we will give insights in the features supported by the latest version of our virtual WebAuthn authenticator device. We will start by listing the cryptography algorithms supported, then move to the mechanics used by the authenticator to generate, handle and store public-private key pairs (credentials) and then analyse the attestation formats that it can generate and return to the server.

3.5.1 Credentials

In terms of public key pairs generation, our virtual authentication implementation is currently able to generate and use various popular public private key signature algorithms. Specifically, the following algorithms and schemes are supported (codes in parentheses are based on the IANA COSE Algorithms [31] registry):

- Elliptic Curve Digital Signature Algorithm (ECDSA) with SHA-256 (-7), SHA-384 (-35) and

SHA-512 (-36);

- RSA Probabilistic Signature Scheme (RSASSA-PSS) with SHA-256 (-37), SHA-384 (-38) and SHA-512 (-39);
- Signature Scheme with Appendix as first standardized in version 1.5 of PKCS #1 (RSASSA-PKCS1-v1_5) with SHA-1 (-65535), SHA-256 (-257), SHA-384 (-258) and SHA-512 (-259);
- Fast-Fourier Lattice-based Compact Signatures over NTRU (FALCON) with 256, 512 and 1024 variants (custom implementation);

The virtual authenticator can be upgraded to support more public private key algorithms and schemes by extending the appropriate section of the code. This can be achieved by using algorithms or schemes already supported by a browser's Web Cryptography API implementation, or extending it to support more using Javascript libraries. Specifically, to demonstrate the possible applications of this and also to kick start the discussion on quantum resistant algorithms on FIDO and WebAuthn, we implemented the above mentioned FALCON [32] algorithm using Emscripten and the source code submitted on the Post Quantum NIST competition round 3 [33]. We used the virtual authenticator's FALCON implementation to test and develop our open-source quantum resistant FIDO2 server [34] implementation and our FIDO2 SDK kit [35]. To our knowledge this is the first working implementation of quantum resistant FIDO2 password-less authentication.

3.5.2 Key wrapping

Through a key wrapping process where private key information is securely encrypted and wrapped inside a value, authenticators can store the generated credentials at the FIDO2/WebAuthn server by passing that value as a KeyID (credentials ID). This method, known as server-side credential storage modality strategy, allow the authenticator to support an unlimited number of keys and also eliminate the need for large storage resources.

Instead of requesting authentication with any credentials generated for that particular relying party, authentication could also be deployed by having the server supply the user's credentials ids in a list (list of wrapped keys values). This method resembles the older U2F functionality of FIDO, where the account for which we are authenticating against, must be known prior to the initiation of the process, so that the list of associated credentials ids can be recovered, hence working as a second-factor authentication. When the server has no knowledge of the account to be authenticated, and starts the authentication process, the authenticator would have to provide this information (as the user handle parameter) through the use of a resident key (also known as discoverable credentials).

Many security FIDO2/FIDOU2F key manufacturers are using the key wrapping approach so that their keys do not run out of storage. We can assume that this also reduce the cost of the authenticator device as it may reduce the hardware and software requirements needed to store and manage credential information. To support an unlimited number of keys without the need for storage, our virtual authenticator implementation, wraps the private key (along with other authenticator data) in an AES-GSM with a 256-bit key size. In this way, our implementation can be used across different machines or browsers without the need to copy data. To ensure not only the the recovery of the private key, but also the recovery of its state and the future compatibility of the generated credentials, the following information are wrapped to generate the a credentials ID:

- Authenticator Format Version (currently 1);
- Relying Party ID (the id given during credentials creation);

- User Handle (the user id given during credentials creation);
- Credentials Creation Date;
- Key Pair Data (depending on the algorithm);

In case the key wrapping is not adequately secure it may be easier for an attacker that knows its schema to crack it (and recover the private key of the credentials) instead of cracking one of our public-private keys. Thus, the key wrapping schemes deployed on production implementation of secure authenticator devices should always be of the same or greater security strength compared to the public-private scheme that it is protecting. Furthermore, manufactures should not assume their implementation's security is based upon the obscurity of their wrapping schema. To add more on the discussion, depending on the relying party server implementation, there may be a limit on the size of a credential id and thus limitations may be applied on the amount of information that can be wrapped inside it, which may increase as the public-private schema's key size increases.

A master key is used by the authenticator as a key to the AES-GCM for the key wrapping. A custom secret is given to the authenticator during initialization that is used to generate the its master key. To generate a strong master key, the given secret is passed through the Password-Based Key Derivation Function 2 (PBKDF2) using SHA-256 and enough iterations. As a result, even if the credentials information is stored on the browser's local storage, our virtual authenticator is able to decrypt and retrieve only the credentials wrapped using the appropriate secret supplied. Users are able to set a new master key secret or be asked for one on the fly, upon an authenticator action. As this secret is protecting not only the keys stored on the local storage but also the wrapped keys stored on the server side, it is suggested to use a long and complex password. Depending on the use case, this may not be of high concern as our authenticator is supported to be used only for development and testing reasons.

We have to note though that our virtual authenticator don't only supports server-side public key credential sources (also known as non-resident keys) with wrapped credentials information inside, but also client-side discoverable credentials (also known as resident keys).

3.5.3 Authenticator attestation

Many WebAuthn services opt to request from authenticators to return the device's attestation object (direct attestation) rather than no attestation (none attestation). Authenticators that do not satisfy the server's attestation dependencies are rejected and their registration process is aborted. Hence, to extend the implemented virtual authenticator's compatibility, as well as to broaden the testing functionalities of the proposed solution, a packed self-signed attestation (named surrogate basic attestation) has been developed. It is using the generated private key which is returned when indirect or direct attestation is requested. In self-signed attestation, the authenticator signs the returned authenticator data using the newly generated private key, proving its possession to the server.

Furthermore, especially for direct attestation, a custom authenticator certificate is also generated and is included in the attestation object, though this functionality is currently only supported for ECDSA keys. To achieve this, since such operations are not currently supported by WebCrypto, parts of the Abstract Syntax Notation One (ASN.1) encoding and the Distinguished Encoding Rules (DER) were implemented in JavaScript, so that our authenticator's responses featured correct formatting and encoding.

Whichever the attestation format, to generate a valid response, our authenticator should be able to generate signature counter values, each one always greater than the last one, otherwise the relying party server may flag the authenticator device as cloned. Since our virtual authenticator does not feature a storage to save its state, the signature counter had to be implemented in another way. For this reason, our

implementation calculates the signature counter based on the client's machine time. Specifically, we set as a point of reference the credentials creation date (therefore wrapped inside the credentials id value) and we assume that the counter increases by a value of 1 every 250 milliseconds (arbitrarily chosen), thus the counter will overflow (and consequently be invalidated) about 34 years after its creation, which we found to be reasonable.

4 Testing and evaluation of proposed tool

To test and evaluate the performance our WebDevAuthn tool, a set of use-cases have been meticulously defined to showcase the entire range of capabilities offered to application and service developers. The sections describes the use cases and presents the evaluation results, emphasizing on the benefits of the proposed tool. Our tool is publicly available on GitHub [8], where someone can get the code and/or experiment with it through GitHub pages functionality.

4.1 Use Case A: Educating on the WebAuthn API

In order to facilitate the faster and smoother adoption of WebAuthn, we have get people familiar with it and educate the developers on how to correctly leverage the technology to build secure WebAuthn services. To achieve that it is important to provide demos which let the interested parties understand what is WebAuthn, experience where they can use it and experiment with it testing their authenticator devices. Furthermore, it is important to provide to developers the appropriate material and tools not only to educate them efficiently on WebAuthn but also to let them debug their services.

Our tool can be used as a playground for individuals (developers and/or stakeholders) interested in password-less authentication using WebAuthn. The users are able to craft custom WebAuthn requests, both credentials creation and credentials get requests, by picking their preferred options through a graphical interface, as shown in Figure 10. The tool provides a description (based on the WebAuthn standard) for each option, explaining its purpose, usage and indicating whether it is required or optional.

Following the selection of the preferred options, the user can generate the corresponding WebAuthn request API call code [6] with those exact selected options. Figure 11 shows the generated Javascript code of the selected options presented on Figure 10. The generated code is fully functional and can be used to launch a WebAuthn request right on the browser.

The user can then invoke the generated WebAuthn code launching a request which will be handled by the browser. The user will then be able to use his authenticator device to respond to the request. Since our tool features a virtual authenticator, it can also be used by users without the possession of a FIDO2, FIDO U2F or any platform authenticators (e.g., Windows Hello).

The tool is then able to analyse the return response. The analysis includes decoding of the embedded parts of the report and analysis of the returned attestation assuming that it is supported. Figure 12 shows the decoded response of a custom request handled by our tool's virtual authenticator.

We leveraged the our implementation's playground to showcase how FIDO works internally so by showcasing all the mechanics working underneath. Users was able to create custom requests, test them on their available platforms and explore their devices FIDO capabilities (e.g., availability of biometric authentication). The playground was particular helpful for developers looking into learning how to use WebAuthn, as it was easier to get them familiar with the API through the practical approach to introduce them to the code defined WebAuthn specification. Finally, we used the tool's as a WebAuthn traffic analyser during the development of WebAuthn services for resolving bugs in the implementation through the inspection of the decoded data.

Based on our observations, we found our tool's playground to be helpful in familiarising interested

The screenshot shows a user interface for creating a credential. It is organized into four distinct panels:

- Relaying Party Entity:** Contains two input fields. The first is labeled 'rp.name' with a 'REQUIRED' tag and contains the text 'FIDO 2 Unipi'. The second is labeled 'rp.id' with an 'OPTIONAL' tag and contains the text 'webdevauthn.ddns.net'.
- User Account Info:** Contains three input fields. The first is labeled 'user.name' with a 'REQUIRED' tag and contains 'john.smith@email.com'. The second is labeled 'user.displayName' with a 'REQUIRED' tag and contains 'J. Smith'. The third is labeled 'user.id' with a 'REQUIRED' tag and contains a list of numbers: '[106,111,104,110,46,115,109,105,116,104,1'.
- Timeout:** Contains one input field labeled 'timeout' with an 'OPTIONAL' tag, containing the value '120000'.
- Authenticator Selection Criteria:** Contains three dropdown menus. The first is labeled 'authenticatorSelection.authenticatorAttachment' with an 'OPTIONAL' tag and is set to 'Don't set'. The second is labeled 'authenticatorSelection.requireResidentKey' with an 'OPTIONAL' tag and is set to 'Don't set'. The third is labeled 'authenticatorSelection.userVerification' with an 'OPTIONAL' tag and is set to 'Don't set'.

Figure 10: Credential's creation options selection user interface

parties with WebAuthn where other simpler demos fail due to the lack of deep analysis of the WebAuthn API the developers seek or due to the user's lack of a FIDO authenticator. The virtualisation our tool offers through the virtual authenticator allow the user's experimentation with WebAuthn independent of the platform they use (even if their platform does not support WebAuthn). Furthermore, the simple yet descriptive interface, fulfills the requirements of both technical and non-technical users.

4.2 Use Case B: Compatibility with 3rd party implementations

In order for our tool to operate at its full potential and enable the conformance and security assessment, including the emulation of the WebAuthn authenticator responses, it has to work seemingly with the 3rd any party service targeted for analysis. For this to be achieved, our tool has to support all the dependencies set by the relying party server and generate valid (for the server) WebAuthn responses. Furthermore, the relying party itself will have to support the attestation formats return by our authenticator.

We used our tool as a debugging mechanism during the development of in-house experimental FIDO2 web applications. Specifically, we deployed the tools while working with the certified StrongKey FIDO2 server [36] and an experimental FIDO2 server based on Yubico's FIDO2 python library [37]. In this way we were able to test and improve our tool (and the underlined virtual authenticator) with various

```

// Create Credentials
navigator.credentials.create({
  "publicKey": {
    "rp": {
      "name": "FIDO 2 Unipi",
      "id": "webdevauthn.ddns.net"
    },
    "user": {
      "name": "john.smith@email.com",
      "displayName": "J. Smith",
      "id": fb64("am9obi5zbWl0aEB1bWFpbC5jb20") // Uint8Array
    },
    "challenge": fb64("Hx4dHBsaGRgXFhUUExIREA8ODQwLCgkIBwYFBAMCAQA"), // Uint8Array
    "pubKeyCredParams": [
      {"type": "public-key", "alg": -7}, // ECDSA w/ SHA-256
      {"type": "public-key", "alg": -37}, // RSASSA-PSS w/ SHA-256
      {"type": "public-key", "alg": -257} // RSASSA-PKCS1-v1_5 using SHA-256
    ],
    "timeout": 120000
  }
}).then((credentials) => {
  console.log(credentials);
});

function fb64(x) { // Base64 to Uint8Array
  return Uint8Array.from(atob(x.replace(/-/g, '+').replace(/_/g, '/')), c => c.charCodeAt(0)
};

```

Figure 11: Credential's creation generated WebAuthn API call code

configurations as we had control over the relying party servers.

To further assess our authenticator's compatibility, we searched for services and/or additional software featuring WebAuthn authentication and compiled a list of 29 WebAuthn services¹ (listed on Table 2) that we could analyse using our tool. The list is composed by various WebAuthn implementations including WebAuthn demos, open-source and commercial solutions and identity management servers. For each WebAuthn service on the list, we tested the compatibility of our tool with the 3rd party service, by first registering our virtual authenticator device and then, assuming that the registration was successful, executing a password-less or second-factor authentication.

As shown on Table 2, we found our tool's features were compatible with all the services that we tested it against. Specifically, all features worked including the virtual authenticator and the WebAuthn requests analysis. Although in our tests we found our tool compatible will all the services, an incompatibility of our virtual authenticator may exists with a services which may reject our virtual authenticator's attestation, considering our authenticator as not trustworthy or insecure. The later maybe an intended security measure based on the usage of the FIDO attestation metadata service or a white-listing functionality allowing only popular authenticator devices.

For some of the services (flagged as partially compatible), due to a credentials id size limit set by the

¹Note that a number of FIDO services (such as Google's authentication services using security keys) were excluded from the list as they are still using the deprecated and not standardized U2F Javascript API and not WebAuthn

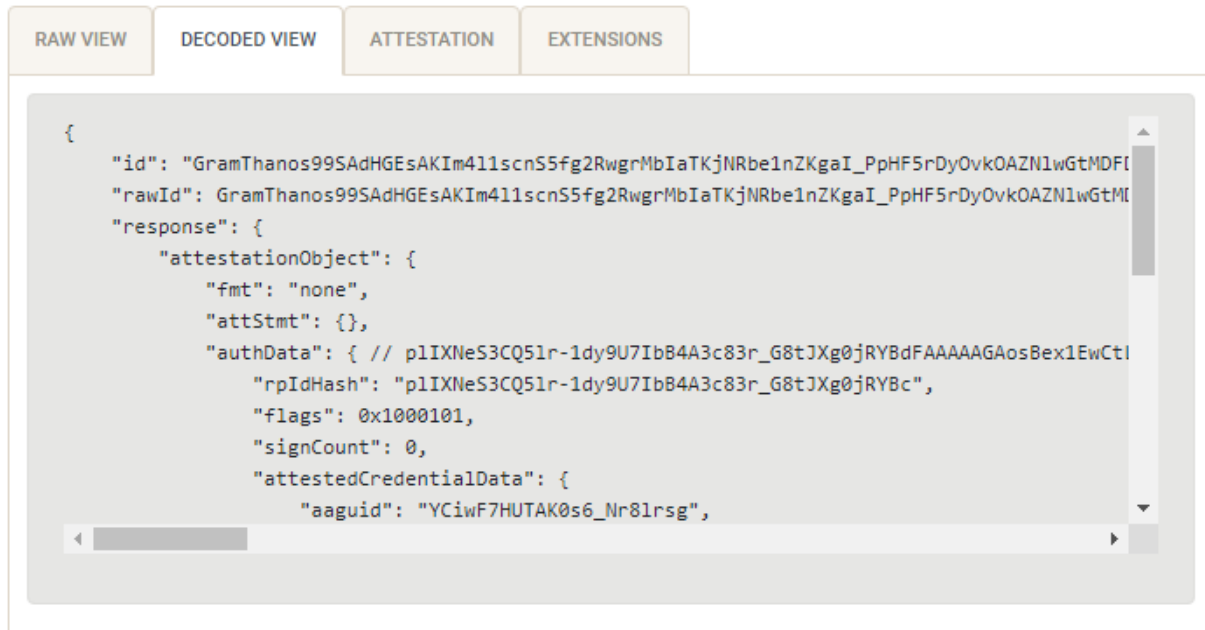


Figure 12: Credentials creation response generated from our virtual authenticator and decoded by the credentials analyser

relying party server, our virtual authenticator registration response was rejected. To bypass the problem, we triggered the "force resident key" testing functionality on our authenticator which reduced the size of the credentials id dramatically, as there is no need to wrap the credentials information inside the credentials id, and thus our authenticator's response was accepted by the server.

We found our tool analysis capabilities to be compatible with all the WebAuthn services we tested, which is essential for assisting developers in conducting conformance and security assessments of WebAuthn services regarding the front-end and the back-end environment. Furthermore we found our tool's virtualisation capabilities to be compatible with the majority of the services we tested, which is important to facilitate extensive testing to the relying party's server back-end.

4.3 Use Case C: Assessing implementations

To assess the performance of the tool various WebAuthn implementation were selected, as shown previously. During these testings and assessment sessions of various websites supporting WebAuthn for password-less or second factor authentication, a number of problems replicated in the majority of the servers were observed.

The features of the our tool are not limited only in inspecting the WebAuthn traffic and generating response. It is extended to include assessment functionalities so that we can easily conduct test on 3rd party implementations. Since the tool is able to interact with the services (through the Hijacker), just like any other authenticator, it is able to perform blind tests handling the WebAuthn server as a black box.

4.3.1 Analyser's assessment functionalities

The first assessment the implemented tool is performing automatically upon receiving a WebAuthn request (supported for both credentials creation and get) is a conformance review on the given options supplied by the WebAuthn service. It checks the correct usage of the options both in terms of data types

Table 2: Compatibility of our virtual authenticator with FIDO2/WebAuthn implementations

Compatibility	Implementation	Type
PARTIAL	1password.com	Commercial
YES	boxcryptor.com	Commercial
YES	cedarcode.com	Demo
YES	cloudflare.com	Commercial
YES	debauthn.tic.udc.es	Demo
YES	demo.yubico.com	Demo
PARTIAL	dropbox.com	Commercial
YES	EU Login	Commercial
PARTIAL	ebay.com	Commercial
YES	github.com	Commercial
YES	Gluu IDM	Open-Source Server
YES	Hanko Apple-Passkey Demo	Demo
YES	KeyCloak IDM Server	Open-Source Server
YES	magicauth.net	Demo
YES	microsoft.com	Commercial
YES	Microsoft WebAuthn Sample	Demo
YES	Nextcloud	Open-Source Server
YES	passwordless.dev	Demo & Service
YES	quado.io	Demo
PARTIAL	Singular Key WebAuthn Demo	Demo
YES	StrongKey Fido2 Server	Open-Source Server
YES	token2.com	Demo
YES	web-auth-n-demo.herokuapp.com	Demo
YES	webauthn.bin.coffee	Demo
YES	webauthn.io	Demo
YES	webauthn.lubu.ch	Demo
YES	webauthn.me	Demo
YES	webauthn.org	Demo

given and by validating the assigned values. Furthermore, it tries to identify problems on the selected parameters and reports them back to the user. All the automated generated comments are available under the tool's notes section.

- Report options schema deviations from the specification;

The analyser automatically detects unknown parameters or missing parameters defined as required by the specification. For example, the analyser will detect and warn user about misspelled parameters passed on the options object.

- Report invalid options value types;

The analyser automatically detects parameters whose value type is not the same as the one defined by the WebAuthn standard. For example, the analyser will detect and warn user about values that where expected to be integers but strings where given.

- Report invalid option values (e.g. values outside range or dictionaries);

The analyser automatically detects parameters whose value is not inside the accepted values defined by the WebAuthn standard. For example, the analyser will detect and warn user about "time-out" values that deviate from the suggested ones.

- Report default values for options that are not defined;

The analyser automatically reports default values of parameters not specified. For example, the analyser will report the default value of the "pubKeyCredParams" parameter if it was not specified on the request.

- Report usage of insecure options;

The analyser automatically detects parameters whose value may leave the service vulnerable to attacks. For example, the analyser will detect and warn user about not setting "userVerification" parameter to "required".

- Report usage of not recommended, deprecated or unknown signature algorithms;

The analyser automatically detects "deprecated" or "not-recommended" public-private signature algorithms reported by the WebAuthn service (based on the recommendations by IANA [31]). For example, the analyser will detect suggestion to use the "RSASSA-PKCS1-v1_5 using SHA-1" scheme and warn the user.

- Suggest priority ordering of signature algorithms based on strength;

The analyser automatically detects if the server prefers the usage of algorithms with lower security level over other with higher security level. Furthermore a suggestion to change the order based on the security level will be given. For example, the analyser will detect if the relying server prefers the usage of "RSAES-OAEP w/ SHA-256" over "RSASSA-PSS w/ SHA-512" and suggest a new order².

- Check id values for personal info leakage;

The analyser automatically detects any possible usage of private data such as email, telephone or name used as a user identifier, to avoid any leakage of personal information. For example, the tool will detect if a service is trying to use a telephone number as a user account identifier.

- Check challenge randomness and security level;

The analyser automatically try to detect whether the challenge supplied is secure enough, by checking its bits length and conducting a number of randomness tests. For example, the analyser will detect challenges that is not at-least 16 bytes long as suggested by the WebAuthn specification.

To assess the randomness of challenges passed from the server, our tool uses a random bitstream tester [38] and passes them through a number of tests. It needs to be underlined that some WebAuthn server implementations use big ASCII values as challenges that may fail to pass randomness tests, but they seem encoded (e.g. in base64) and thus the random value used to generate them may be wrapped inside. In the same way, other implementations may concatenate additional not random pieces of information on the challenge (e.g. a session, a state). No matter the format of the challenge, the initial random value used should be at-least 16 bytes long.

²Note that the suggestions are comparing the security level of the hash functions used by the scheme and assumes that the schemes are secure

4.3.2 Virtual authenticator's assessment functionalities

For more advance testing a number of manual enabled options that manipulate the behaviour of the virtual authenticator have been implemented. Through these advance testing options, developers are able to test functionalities of their service that otherwise are not easily triggered.

- Swap Challenge

This option allows the alternation of the challenge passed to the authenticator device. Developers can use this options to observe a service's behaviour when an invalid challenge was returned.

- Swap User ID

This option allows one to change the User ID passed to the authenticator device. Developers can use this options to bind the generated credentials with another account (inside the authenticator).

- Freeze User Verification Flag;

This option allows the manipulation of the user verification flag. Developers can use it to simulate man-in-the-middle attacks where the attacker removes the requirement of user presence from the request. This option can test whether a WebAuthn service implementation validates the response flags when it requires the user presence.

- Swap Relay Party ID

This option allows the alternation of the Relay Party ID (website's domain name) passed to the authenticator device. Developers can use this options to simulate man-in-the-middle attacks where the authenticator device generated credentials for another Relying Party, thus for another website.

- Override Signature Algorithm

This option can be used to force the virtual authenticator to select a specific algorithm to generate credentials with. This testing option can be deployed to test the behaviour of the server for different signature algorithms, or to return credentials based on an algorithm that the server is not supposed to accept. For the later case, such a testing could simulate security downgrade attacks where the attacker is trying to generate credentials using weaker or deprecated algorithms.

- Clone AAGUID

This option will clone the AAGUID of other (maybe more popular) authenticators. Using this option, the effectiveness of AAGUID whitelist or blacklist filters could be tested. Note that since we don't poses the private keys required to generate valid attestation objects for other authenticator devices, this option will not trick servers that uses the FIDO Alliance's Metadata Service.

- Force Resident Key

This option forces the authenticator to generate a resident key. This reduce the size of the credentials id generated by the authenticator device as the credentials information are saved in the virtual authenticator (inside the browser's local storage). This is often helpful in case the Relying Party service has a limit in the credentials id size.

- Freeze Signature Counter

Table 3: Tests Conducted on FIDO2/WebAuthn services registration.

		Tests on Registration Flow									
		Not Recommended or Deprecated algorithms	No user verification	Bad algorithm ordering	Swap Challenge	Freeze User Verification Flag	Swap Relay Party ID	Swap Origin	Swap HTTP Origin	Swap Subdomain Origin	Override Signature Algorithm
Services	1password.com	○	●	○	○		○	○	●	○	●
	boxcryptor.com	○	●	○	○		○	○	○	○	○
	cedarcode.com	●	●	○	○		○	○	○	○	○
	cloudflare.com	●	●	○	○		○	○	○	●	●
	debauthn.tic.udc.es			○	○	○	○	○	○	○	○
	demo.yubico.com	●	●	○	○	○	○	○	○	○	○
	dropbox.com	●	●	○	○		○	○	○	○	○
	EU Login	●	●	○	○		○	○	○	○	○
	ebay.com	●	○	●	○	○	○	○	○	○	○
	github.com	●	●	○	○	○	○	○	○	○	○
	Hanko Apple-Passkey Demo	●	○	●	○	○	○	○	○	○	○
	KeyCloak IDM Server			●	○	○	○	○	○	○	●
	magicauth.net	●	●	○	○	○	○	○	○	○	○
	microsoft.com	●	○	○	○	○	○	○	○	○	○
	Microsoft WebAuthn Sample	●	●	○	○		○	○	○	○	○
StrongKey Fido2 Server	●	●	●	○		●	○	○	○	○	

● Affected ● Unhanded ○ Not Affected empty Not Applicable

This option gives you full control of the signature counter. By enabling this option one can set the signature counter to any value and simulate clone authenticator attacks. We found this useful for testing the server’s incident handling behaviour.

- Swap User Handle

This option allows one to overwrite the returned user handle value. By using this option, one can impersonate another user and trick poorly implemented WebAuthn services.

4.3.3 Assessments findings

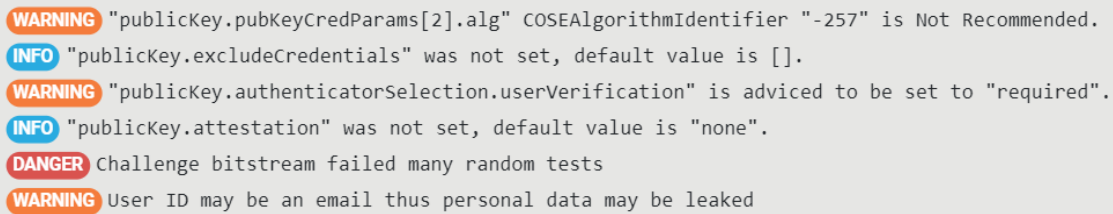
We analysed the assessment results for a number of service and listed the issues identified for the registration and the authentication flow at the Table 3 and Table 4 accordingly. At the tables one can see for each service the issues that affect it as well as the issues that the services is not handling correctly (bad response the the issues or bad handling of the problem).

Table 4: Tests Conducted on FIDO2/WebAuthn services authentication.

Services	Tests on Authentication Flow							
	No user verification	Freeze User Verification Flag	Swap Challenge	Freeze Signature Counter	Swap Origin	Swap HTTP Origin	Swap Subdomain Origin	Swap User Handle
1password.com	●		○	○	○	○	○	◐
boxcryptor.com	●		◐	◐	◐	○	○	◐
cedarcode.com	●		○	◐	○	○	○	◐
cloudflare.com	●		○	◐	○	○	●	○
debauthn.tic.udc.es			○	◐	○	○	○	○
demo.yubico.com	●		○	●	○	○	●	◐
dropbox.com	●		○	●	○	○	○	◐
EU Login	●		○	◐	○	○	○	○
ebay.com	●		◐	●	●	●	●	◐
github.com	●		○	◐	○	○	○	◐
Hanko Apple-Passkey Demo	○	○	○	◐	○	○	○	○
KeyCloak IDM Server		○	○	◐	○	○	○	○
magicauth.net	●		○	◐	○	○	○	○
microsoft.com	○	○	○	◐	●	○	○	◐
Microsoft WebAuthn Sample	○	○	○	◐	○	○	○	●
StrongKey Fido2 Server	●		○	◐	○	○	○	◐

● Affected ◐ Unhanded ○ Not Affected empty Not Applicable

One of the first showcased FIDO2/WebAuthn attacks was that of the "unaware user presence". Specifically, the scenario for this attack is the following: The victim possesses an NFC authenticator, which is placed on his/her pocket. The attacker passes by and scans the authenticator and logs into a service without the victim noticing it. This attack can be mitigated, if the service indicates on the request that a user action is required to prove the user's presence, through setting the "userVerification" option to "required". Throughout our testing sessions, the majority of the servers do not have this option set. This results in setting the default value of "preferred", thus they are unable to reject authentications performed without an interaction between the user and the server. The problem escalates further due to the fact that the WebAuthn specification assumes that the scanning of an NFC authenticator is considered an interaction with the user, which may solve practical implementation problems, at the expense of reduce security of the security keys of the NFC user. The developed Credentials Creation Analyser detects this problem and automatically informs the user throwing a warning, as shown in Figure 13. We recommend all implementation (especially the once that use WebAuthn as single factor authentication) to set the user verification as "required" by default in order to mitigate this attack. As the support for FIDO2 increases,



```
WARNING "publicKey.pubKeyCredParams[2].alg" COSEAlgorithmIdentifier "-257" is Not Recommended.
INFO "publicKey.excludeCredentials" was not set, default value is [].
WARNING "publicKey.authenticatorSelection.userVerification" is advised to be set to "required".
INFO "publicKey.attestation" was not set, default value is "none".
DANGER Challenge bitstream failed many random tests
WARNING User ID may be an email thus personal data may be leaked
```

Figure 13: Notes raised by the credentials analysers after assessing the given WebAuthn options

and the authenticator devices became common, the risk of such attack will also increase. As a user, avoid using authenticator devices that do not require an action (e.g. a press of a button or a touch with your fingerprint) to complete the authentication.

An additional implementation problem identified during the evaluation sessions is due to the wide adoption and use of algorithms which are considered by IANA as “deprecated” and “not recommended” [31]. Most of the servers allow authenticators to register using the “not recommended” algorithm “RSASSA-PKCS1-v1_5 using SHA-256” (number code -257) to support backwards compatibility with older authenticator devices. Furthermore, several servers allow the use of the deprecated “RSASSA-PKCS1-v1_5 using SHA-1”. Our tool manages to detect the usage of such algorithms and warns the user about the issue. This functionality is illustrated in Figure 13. Although this may break compatibility with old FIDO authenticators, all FIDO implementations should remove or disable the support for “deprecated” and “not recommended” algorithms. Furthermore, configuration options should be given to the service administrators so that any algorithm could be disabled manually if needed in the future. It is recommended to the users to use relative new authenticator devices which will most probably make use of the newer recommended algorithms, or check your authenticator’s specifications to ensure that only strong and recommended are supported.

Various minor problems were identified during the evaluation that, to this time, have not been reported in the past. In particular, one of the most prominent is the wrong ordering of algorithms on the credentials creation options. It can be assumed that since the algorithms are supposed to be ordered based on the server’s preference, most servers would list the algorithms based on their cryptographic strength. It has been found during the testing that this is not the case, since in most implementations the order that the algorithms are listed is random or grouped based on the family that the algorithm belongs to. This may force an authenticator device which support multiple algorithms, to generate and register credentials with lower cryptographic strength, hence essentially downgrading the security. Our tool achieves in identifying this problem and proposing a better ordering of the algorithms when algorithms with lower security level are set by the sever on the top of the preference order. It is suggested that all the WebAuthn implementations order their supported algorithms based on their security and not randomly. This problem highlights again the importance of removing deprecated and not recommended algorithms from the supported algorithms list to avoid downgrading the security of an authenticator that opts to use a less secure algorithm while a stronger one is supported.

During this use case a more through testing was performed leveraging the virtual authenticator’s testing options. Therefore the behaviour of services on unusual input were able to be tested. A significant finding involves the emulation of attacks and the observation of the relying party’s server behaviour. In particular, the scenario of a cloned authenticator device trying to authenticate to a WebAuthn service, was tested. To perform the attack the virtual authenticator’s signature was set to a lower value (from the one the server was expected to see) and was instructed to authenticate with the service. The detection of a lower value on the signature counter, most probably indicates that a cloned authenticator attack is been

conducted, thus the server should not only reject the authentication, but also disable the authenticator device inform the user, as well. During the tests, most servers just blocked the authentication without performing any other action and only one service removed the device from the account's authenticators, without informing the user by any means about the incident.

Assuming that the next years we may see TPM and TEE attacks, also affecting FIDO2 authenticator devices (such as the Samsung's TrustZone attack on android [39] which can be used to recover FIDO2 private keys), cloned authenticator attacks may cause a big problem and thus preparing our systems to detect and report such attacks to security key manufactures may be essential for identifying problems and blacklisting vulnerable authenticator devices. We should look into how in the future such a reporting functionality could be established, maybe through the FIDO Alliance Metadata Service, so that vulnerable authenticators (either new ones during registration or already registered ones) can be detected by all systems. It is recommended that FIDO implementations monitor and pay more attention to failed authentication attempts as, unlike to the password authentication method where the human factor may introduce errors, with FIDO all the authenticator errors may be attributed to timeouts, incompatibilities or attacks.

It is evident through the presented test results (and other observations such as unhandled server errors) that although the deployed WebAuthn services are quite secure, the developers lack the tools to test them appropriately.

5 Conclusion

The paper presented a methodology to analyze WebAuthn requests and responses by injecting JavaScript code into the web application to be debugged, hijacking the WebAuthn methods and forwarding it to our analyzer, in a non-invasive way without affecting the application code flow. Our implementation, can be used by developers to analyze the WebAuthn traffic of their applications and possibly speed up the debugging process, since the proposed tool will provide direct insight into the information which is packed inside the WebAuthn requests and responses. One of the key objectives of the developed solution is that it lays the foundation for aiding developers to familiarise with and deeply understand the FIDO2/WebAuthn JavaScript requests and responses. The proposed tool and the evaluation sessions performed with it proved that it is compatible with a large number of FIDO2/WebAuthn implementation available on the market. While the prototype virtual authenticator implementation proved to be working seamlessly with many FIDO2/WebAuthn back-ends. Furthermore, the assessment capabilities of the proposed analysers were successfully tested and proved very helpful in validating implementations and identifying configuration problems.

The work of this study is not exhausted with these use cases. The modular architecture and the extensibility of our tool, allow us to enhance it further and introduce new functionalities. Future developments and testing will be focused on improving the assessment functionalities of the analysers and expanding the testing and assessment sessions of WebAuthn implementations, particularly from the open-source projects. Thus, positively affecting their maturity by indicating vulnerabilities and improving their security, correct implementation and appropriate configuration. Towards this end, the upgrading of the virtual authenticator implementation is currently an ongoing project that would extend the testing options, enabling it to support more innovating attack simulations.

Acknowledgments

This research has been partially funded by the European Union's Horizon 2020 Stimulating innovation by means of cross-fertilisation of knowledge program under the Grant Agreement No 824015 (H2020-

MSCA-RISE-2018-INCOGNITO) and the Greek state funded Operational Programme Competitiveness, Entrepreneurship and Innovation 2014-2020 (EPAnEK) under the Grant Agreements NetPHISH-T1EΔK – 05112. Furthermore, we would like to extend our thanks to the members of the Systems Security Laboratory (SSL) of the University of Piraeus, Greece, for their participation in the testing sessions and the valuable feedback on improving the technical implementations featured in this paper.

References

- [1] M. Bromiley. Bye bye passwords: New ways to authenticate. <https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RE3y9UJ>, June 2019.
- [2] A. Angelogianni, I. Politis, F. Mohammadi, and C. Xenakis. On identifying threats and quantifying cybersecurity risks of mnos deploying heterogeneous rats. *IEEE Access*, 8:224677–224701, December 2020.
- [3] C. Ntantogian, S. Malliaros, and C. Xenakis. Evaluation of password hashing schemes in open source web platforms. *Computers & Security*, 84:206–224, July 2019.
- [4] FIDO Alliance. Open authentication standards more secure than passwords. <https://fidoalliance.org>. [Online; Accessed on Dec. 2, 2021].
- [5] K. Papadamou, S. Zannettou, B. Chifor, S. Teican, G. Gugulea, A. Caponi, A. Recupero, C. Pisa, G. Bianchi, S. Gevers, C. Xenakis, and M. Sirivianos. Killing the password and preserving privacy with device-centric and attribute-based authentication. *IEEE Transactions on Information Forensics and Security*, 15:2183–2193, December 2020.
- [6] D. Balfanz, J. Hodges, E. Lundberg, M. Jones, H. Liao, J. C. Jones, A. Czeskis, A. Kumar, and R. Lindemann. Web authentication: An api for accessing public key credentials level 1, w3c recommendation. <https://www.w3.org/TR/2019/REC-webauthn-1-20190304/>, March 2019. [Online; Accessed on Nov. 17, 2021].
- [7] A. Alam, k. Krombholz, and S. Bugiel. Poster: Let history not repeat itself (this time) – tackling webauthn developer issues early on. In *Proc. of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS’19)*, New York, NY, USA, pages 2669–2671. ACM, November 2019.
- [8] A.V. Grammatopoulos, I. Politis, and C. Xenakis. Webdevauthn: A tool to test & analyze fido2/webauthn requests and responses. <https://gramthanos.github.io/WebDevAuthn/>, 2022.
- [9] A.V. Grammatopoulos, I. Politis, and C. Xenakis. A web tool for analyzing fido2/webauthn requests and responses. In *Proc. of 16th International Conference on Availability, Reliability and Security (ARES’21)*, Vienna, Austria, pages 1–10. ACM, August 2021.
- [10] M. West. Credential management level 1, w3c working draft. <https://www.w3.org/TR/2019/WD-credential-management-1-20190117/>, 2019. [Online; Accessed on Nov. 17, 2021].
- [11] J. Bradley, J. Hodges, M. Jones, A. Kumar, R. Lindemann, and J. Verrept. Client to authenticator protocol (ctap). <https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-20210615.html>, January 2021. [Online; Accessed on Oct. 24, 2021].
- [12] Google. Fido2 api for android. <https://developers.google.com/identity/fido/android/native-apps>, February 2020. [Online; Accessed on Oct. 24, 2021].
- [13] Microsoft. Webauthn: Win32 apis for webauthn standard. <https://github.com/microsoft/webauthn/>, 2018.
- [14] FIDO Alliance. Fido alliance metadata service. <https://fidoalliance.org/metadata/>. [Online; Accessed on Oct. 24, 2021].
- [15] FIDO Alliance. Fido metadata statement. <https://fidoalliance.org/specs/mds/fido-metadata-statement-v3.0-ps-20210518.html>, May 2021. [Online; Accessed on Oct. 24, 2021].
- [16] J. Hodges, A. Kumar, E. Lundberg, M. Jones, and J. C. Jones. Web authentication: An api for accessing public key credentials - level 2. <https://www.w3.org/TR/webauthn-2/>, April 2021. [Online; Accessed on Oct. 24, 2021].
- [17] J. Hodges, M. Jones, and G. Mandyam. Web authentication (webauthn), iana. <https://www.iana.org/assignments/webauthn/webauthn.xhtml>, August 2020. [Online; Accessed on Oct. 24, 2021].

- [18] FIDO Alliance. Functional certification. <https://fidoalliance.org/certification/functional-certification/>, March 2022. [Online; Accessed on Mar. 1, 2021].
- [19] FIDO Alliance. Certified authenticator levels. <https://fidoalliance.org/certification/authenticator-certification-levels/>, March 2022. [Online; Accessed on Mar. 1, 2022].
- [20] A. Simons. A breakthrough year for passwordless technology. <https://www.microsoft.com/security/blog/2020/12/17/a-breakthrough-year-for-passwordless-technology/>, December 2020. [Online; Accessed on Nov. 17, 2021].
- [21] Auth0 Inc. See your webauthn config in action. <https://webauthn.me/debugger>. [Online; Accessed on Nov. 17, 2021].
- [22] M.R. Dourado, M. Gestal, and J.M. Vázquez-Naya. Implementing a web application for w3c webauthn protocol testing. *Multidisciplinary Digital Publishing Institute Proceedings*, 54(1):1–3, August 2020.
- [23] M.R. Dourado. Debauthn: Webauthn authenticator debugging tool, debauthn. <https://debauthn.tic.udc.es>. [Online; Accessed on Nov. 17, 2021].
- [24] M. Miller. Masterkale/webauthn-previewer: A simple website for previewing webauthn attestations and assertions, github repository. <https://github.com/MasterKale/webauthn-previewer>, September 2020. [Online; Accessed on Nov. 17, 2021].
- [25] M. Miller. Webauthn debugger, simplewebauthn. <https://debugger.simplewebauthn.dev>. [Online; Accessed on Nov. 17, 2021].
- [26] S. Weeden. sbweedden/fido2viewer, github repository. <https://github.com/sbweedden/fido2viewer>, 2019. [Online; Accessed on Nov. 17, 2021].
- [27] Mozilla. Window.postMessage() - web apis — mdn. <https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage>, 2018.
- [28] MDN Web Docs. CredentialsContainer.create() - web apis mdn. <https://developer.mozilla.org/en-US/docs/Web/API/CredentialsContainer/create>[Online; Accessed on Oct. 24, 2021].
- [29] MDN Web Docs. CredentialsContainer.get() - web apis mdn. <https://developer.mozilla.org/en-US/docs/Web/API/CredentialsContainer/get>. [Online; Accessed on Oct. 24, 2021].
- [30] M. Watson. Web cryptography api, w3c recommendation. <https://www.w3.org/TR/2017/REC-WebCryptoAPI-20170126/>, January 2017. [Online; Accessed on Oct. 24, 2021].
- [31] F. Palombini, M. Miller, J. Richer, and C. Bormann. Cbor object signing and encryption (cose), iana. <https://www.iana.org/assignments/cose/cose.xhtml>. [Online; Accessed on Oct. 24, 2021].
- [32] P.A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. Falcon: Fast-fourier lattice-based compact signatures over ntru. Technical Report 5, Submission to the NIST’s post-quantum cryptography standardization process, 2018.
- [33] NIST. Post-quantum cryptography - round 3 submissions. <https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions>, October 2021. [Online; Accessed on Dec. 24, 2021].
- [34] A.V. Grammatopoulos, I. Politis, and C. Xenakis. Strongbee: A python fido2/webauthn server (branch with falcon support). <https://github.com/GramThanos/StrongBee/tree/pq-falcon-support>, 2022.
- [35] A.V. Grammatopoulos, I. Politis, and C. Xenakis. Strongmonkey: Sdk for interacting with fido2 server api v3.0.0. <https://github.com/GramThanos/StrongMonkey/>, 2022.
- [36] StrongKey GitHub Repository. Open-source fido server, featuring the fido2 standard. <https://github.com/StrongKey/fido2>, October 2019. [Online; Accessed on Nov. 24, 2021].
- [37] Yubico GitHub Repository. Yubico/python-fido2: Provides library functionality for fido 2.0, including communication with a device over usb. <https://github.com/Yubico/python-fido2>, October 2018. [Online; Accessed on Nov. 24, 2021].
- [38] Z. Molnar. Random bitstream tester. [Online; Accessed on Jan. 24, 2022], December 2020. <https://mzsoltmolnar.github.io/random-bitstream-tester/>.
- [39] A. Shakevsky, E. Ronen, and A. Wool. Trust dies in darkness: Shedding light on samsung’s trustzone keymaster design. Cryptology ePrint Archive, Report 2022/208, 2022. <https://ia.cr/2022/208>[Online; Accessed on Mar. 21, 2022].

Author Biography



Athanasios Vasileios Grammatopoulos received his joined B.S. and M.S. degree in Electrical and Computer Engineering from Technical University of Crete in 2018, and his M.S. degree in Digital Systems Security from University of Piraeus in 2022. Currently he is conducting research in the cybersecurity field in collaboration with the University of Piraeus. Since 2020 he is working as a Cybersecurity Operational Assistant at the European Union Agency for Cybersecurity (ENISA). His research interests include Web-related Technologies and Password-less Authentication.



Ilias Politis received his BSc in Electronic Engineering from Queen Mary College London, UK in 2000, his MSc in Mobile and Personal Communications from King's College London, UK in 2001 and his PhD in Multimedia Communications from the University of Patras Greece in 2009. Currently he is a Senior Researcher at InQbit Innovations SRL. and the Secure Systems Labs of UPRC responsible for the Research and Innovation activities. Dr Politis has previously been working as Senior Researcher at the Wireless Telecommunications Lab. of the Electrical and Computer Engineering at the University of Patras and the School of Science & Technology in the Hellenic Open University, both in Greece. Dr. Politis has been actively involved in all phases of several H2020 projects (ERATOS-THENIS, PHYSICS, 5G-EVOLVED, SPIDER, SECRET, SONNET, EMYNOS) and FP7 framework projects (ROMEO, SALUS, FUTON), as well as several national funded research projects. His research is focused on areas such as, Future Internet and Next Generation networks (5G and beyond), Access management and Trust, where he has published more than 90 journals and conferences. He has been awarded a post-doctoral scholarship under the SIEMENS "Excellence" Program in the field of Telematic Applications by the State Scholarship Foundation (IKY), Greece for his PhD Thesis. He is a member of the IEEE and the National Technical Chamber of Greece.



Christos Xenakis received his B.Sc degree in computer science in 1993 and his M.Sc degree in telecommunication and computer networks in 1996, both from the Department of Informatics and Telecommunications, University of Athens, Greece. In 2004 he received his Ph.D. from the University of Athens (Department of Informatics and Telecommunications). From 1998 – 2001 he was with a Greek telecoms system development firm, where he was involved in the design and development of advanced telecommunications subsystems. From 1996 to 2007 he was a member of the Communication Networks Laboratory of the University of Athens. Since 2007 he is a faculty member of the Department of Digital Systems of the University of Piraeus, Greece, where currently is a Professor, a member of the Systems Security Laboratory, and the director of the Postgraduate Degree Programme, on "Digital Systems Security". He has participated in numerous projects realized in the context of EU Programs (ACTS, ESPRIT, IST, AAL, DGHOME, Marie Curie, Horizon2020) as well as, National Programs (Greek). He is the project manager the CUREX, SECONDO, INCOGNITO and SealedGRID projects, funded by Horizon2020, while he was the project manager of the ReCRED project funded by Horizon 2020 and the technical manager of the UINFC2 project funded by DGHOME/ISEC. He is also a steering committee member of the European Cyber Security Challenge (ECSC) and the leader of the Hellenic Cyber Security Team. His research interests are in the field of systems, networks and applications security. He has authored more than 100 papers in peer-reviewed journals and international conferences.