A web tool for analyzing FIDO2/WebAuthn Requests and Responses

Athanasios Vasileios Grammatopoulos*
SSL, University of Piraeus, Greece
avgrammatopoulos@ssl-unipi.gr

Ilias Politis
InQbit Innovations SRL, Romania
ilias.politis@inqbit.io

Christos Xenakis
SSL, University of Piraeus, Greece
xenakis@ssl-unipi.gr

ABSTRACT

Passwords are a problem in today's digital world. FIDO2, through WebAuthn, brought alternative password-less authentication that is more usable and secure than classic password-based systems, for web applications and services. In this work, we give a brief overview of FIDO2, and we present WebDevAuthn, a novel FIDO2/WebAuthn requests and responses analyser web tool. This tool can be used to help developers understand how FIDO2 works, aid in the development processes by speeding debugging using the WebAuthn traffic analyser and to test the security of an application through penetration testing by editing the WebAuhn requests or responses.

CCS CONCEPTS

• **CCS Description: Security and privacy, Security services, Authentication**;

KEYWORDS

FIDO, WebAuthn, Password-less, Authentication

ACM Reference Format:
Athanasios Vasileios Grammatopoulos, Ilias Politis, and Christos Xenakis. 2021. A web tool for analyzing FIDO2/WebAuthn Requests and Responses. In *The 16th International Conference on Availability, Reliability and Security (ARES 2021), August 17–20, 2021, Vienna, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3465481.3469209

1 INTRODUCTION

The use of passwords is problematic due to the security risks they present, not only to users but also to organizations. Popular password policies require the use of relatively long length and complex passwords, while some of them suggest frequent changes of old passwords. Although such policies may increase security, at the same time compromise the usability of the passwords. Moreover, such policies contribute to the users' tendency to use the same passwords – or similar passwords by modifying them just a bit – 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 huge problem exploited usually by email and website phishing [1], [2].

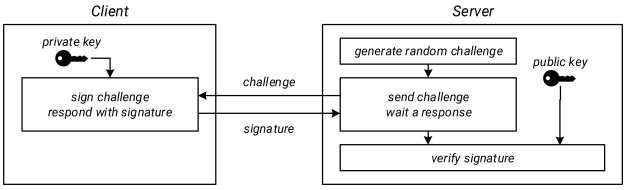

Figure 1: Client authentication using challenge-response and public-private key cryptography.

These disadvantages of password authentication methods led to the deployment of multifactor authentication mechanics and the research and development of alternative password-less authentication methods, like the Fast ID Online (FIDO) framework by FIDO alliance [3]. 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 [4]. Since the release of the first version of W3C's WebAuthn [5] – a specification part of FIDO2 that brings it into our web browsers – in September 2019, FIDO has gained popularity and currently it is 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. Most notably, Microsoft's Corporate Vice President referenced 2020 as "A breakthrough year for passwordless technology" in an article with the same name [6]. In this effort to kill the password problem, libraries, servers, and tools are needed 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 d) secure authentication across multiple services with a single authenticator. Although based on an easy-to-understand concept – that of challenge response –, FIDO defines a large ecosystem of components across multiple domains, which makes it difficult for developers to understand its internals. This complexity is usually hidden from developers behind an API, such is the case of WebAuthn javascript API.

*Corresponding author.

ARES 2021, August 17–20, 2021, Vienna, Austria

ACM ISBN 978-1-4503-9051-4/21/08…$15.00
https://doi.org/10.1145/3465481.3469209

FIDO's ecosystem stretches from secure hardware FIDO authenticator devices that clients possess, all the way up to full scale cloud-based FIDO servers offering registration and authentication services. 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.

Due to the relative recent standardization of WebAuthn, the scientific literature lacks papers and works on the field of developing and testing FIDO2 implementations. Up until now, related work in the field, comes in the form of i) demo WebAuthn websites and ii) "offline" decoder tools. Demo WebAuthn websites (such as [20]) mostly showcase the FIDO2/WebAuthn processes, while some of them reveal parts of their own parameters or the returned response (like the interactive WebAuthn debugger [19] and the work of [17][18]). 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 demos, some developer tools to analyse 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 [21][22] or the fido2viewer [23]), that can be used by developers to unpack, decode or even validate WebAuthn traffic, though the capturing and analysis has to be done manually.

The paper aims to introduce a novel methodology to capture and analyse FIDO2/WebAuthn requests and responses, as well as a technical implementation of our approach, providing the developers with the right tools to deep inspect WebAuthn traffic. Our work allows for instant inspection of the WebAuthn parameters passed to the authentication (essential for validation of the information given by the FIDO2 server), deep decoding and analysis of the WebAuthn response from the authenticator (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), as well as a WebAuthn playground with the ability to generate custom WebAuthn requests (for experimentation and getting familiar with WebAuthn API).

The rest of the paper is organized as follows. In Section 2 the foundations of the FIDO2 and WebAuthn protocols is summarized, while in Section 3 details the proposed WebAuthn analyzed web tool. The capabilities of the tool and its user interface is presented in Section 4. The paper concludes with Section 5, along with a brief overview of the future work.

2 BACKGROUND

FIDO2 at its core uses a challenge-response scheme based on public key cryptography. The server (relying party) prepares a challenge in the form of a random value and forwards it to the client. The client has to sign this challenge with his private key and send the signature back to the server, to prove his identity. Then, the only thing left to be done is for the server to verify the authenticity of the signature using the public key of the user. Obviously, prior to the execution of the challenge response scheme, the server would need to be in possession of the client's public key. A brief overview of the described scheme is shown in Figure 1. The concept idea is quite simple, though a lot more are hidden behind the scenes (uncovered at a later section) to ensure the security of the scheme, compatibility, and ease of use of FIDO.

2.1 FIDO2 in web application

Through the WebAuthn javascript API, web applications are able to request, from the browser and the underlying operating system, credentials creation (public key pair generation) as well as credentials retrieval (proof of secret key possession). The credentials creation method (accessible through window.navigator.credentials.create), from the Credentials Management draft spec, and the public key options, defined at the WebAuthn spec [7], allow the creation of asymmetric cryptography keys (e.g. ECDSA key-pairs) bonded to the caller web application's domain (replay party id) and a user identifier (user handle) linking the credentials with an account. On the other hand, through the corresponding credential get method (accessible through window.navigator.credentials.get) and its public key options, web applications can verify the client's possession of previously created credentials (key-pairs) by requesting the generation of a random challenge's signature and thus verify the identity of a user (through a challenge response scheme). The procedure is illustrated in Figure 2.

A typical use case of FIDO2, using the javascript methods mentioned above, is online password-less authentication, or in other words, the secure login of a user into a website without the use of a secret password. To start the process, the user loads the website through a WebAuthn compatible browser and selects to login with FIDO. The website's backend generates a random high entropy challenge (usually 128 bits or more, as suggested by the specification) and communicates it with the website's frontend. The frontend is then able to invoke the WebAuthn window.navigator.credentials.get javascript method to request from an authenticator to sign the challenge. The browser then sends the challenge along with the website's domain name (replying party id) to the available authenticator devices for signing if an authenticator has a stored key for the website in question. Depending on the system and the support, the browser may contact the authenticator devices directly through the CTAP2 protocol or call custom OS WebAuthn methods. 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. The website's frontend then sends the response data to its backend (the web application server) for verification. Upon successful verification the user is logged in and his session (usually implemented using cookies) is updated. Figure 2 shows the described process as a diagram.

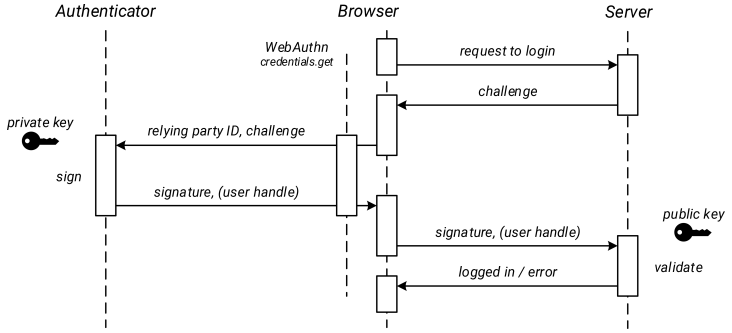

Figure 2: FIDO2/WebAuthn client authentication flow using credentials.get.

The use case described above assumes that authenticator supports resident keys (also called discoverable credentials) and can report back the user identifier (user handle). To support older U2F authenticators or avoid storing information on the authenticator device, web pages may store previously used user identifiers as cookies or at the local storage of the browser and include them at the initial challenge creation request, then the server will return along with the challenge, a list of credential ids registered for this user. This list 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 is already known through the user's session. Thus, a true password-less authentication without need to provide a username or a password, requires information to be stored on the authenticator device which may increase the cost of the authenticator, limits the maximum number of keys that can be generated and needs key management utilities (to be able to remove stored keys that are not needed any more). On the other hand, password-less authentication that requires the knowledge of an account's identifier (e.g., username), does not share such limitations, as the information can be wrapped securely and stored at the relying party's server (through server-side credential storage modality).

Before being able to use FIDO2's authentication process, clients would need to register an authenticator device with the relying party server. During the registration process, the client can generate a public-private key pair supported by the replying party server and send the public key as well as a credentials identifier to the server, to be saved linked with the client's account. Usually, to register an authenticator the client should already be logged into the website, thus a session linked to an account would already be set up. The registration process starts with a request for a challenge generation by the server for credentials creation. In most cases the server will return not only a random challenge, but also 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 (in order to exclude already registered authenticators) and the server's preference for authenticator attestation (whether to request information about the authenticator device used). 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 creation of an authenticator device finished successfully, the browser will return the created credentials to the javascript handler. 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. The described process as a diagram can be seen in Figure 3.

2.2 Security Mechanics

WebAuthn is phishing resistant as its methods do not allow cross domain credentials access unless they are created under the same top domain. 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) except for localhost for development purposes. By requiring HTTPS, the browser can verify the authenticity of the server and thus mitigate man-in-the-middle attacks at the network traffic level.

The server has to verify a number of information apart from the signature, in order to validate the correct execution of the process. First, the challenge returned should match the one generated by the server. To ensure the use of a correct key, the credential identifier returned should be already registered to the user 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 saved with the challenge and cross checked after the response. Additionally, since challenges are generated for use within a limited time frame, the server should expire unused active 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 authenticator requested an interaction by a user. Authenticator attestation data can be used to check the authenticator devices in various ways, for

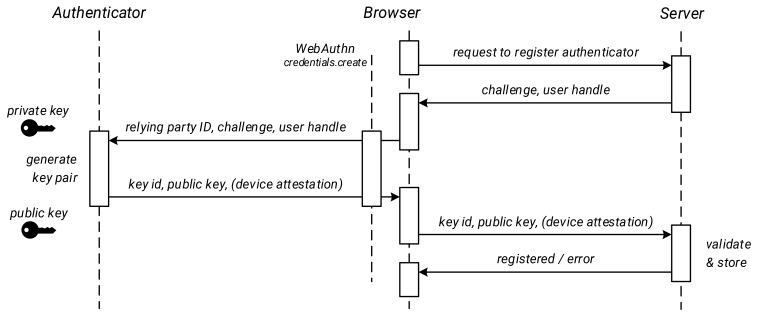

Figure 3: FIDO2/WebAuthn authenticator device registration flow using credentials.create.

example, to identify and unregister authenticators that were found to be vulnerable or to whitelist only trusted authenticator devices.

As one can see, 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. It is thus essential for the developers to deeply understand how FIDO2 and WebAuthn works internally, to deploy such solutions even if a 3rd party library or a server is handling the registration and authentication flows.

3 WEBAUTHN ANALYSER WEB TOOL

In this section, we will describe our novel tool to capture FIDO2/WebAuthn requests and responses, for analysis and human inspection. Through our implementation, one can get familiar with the WebAuthn javascript API as well as to test and analyze responses from real FIDO authenticator implementations. The WebDevAutn tool, showcased in Figure 3, consists of the following main modules:

- the Hijacker in the form of an extension or a developer script,
- the External Communication Manager, used to pipe data between the hijackers and the analyzers,
- the Credentials Manager module, tasked to manage credentials needed across the other modules,
- the Credentials Creation analyser module, that allows the creation of credentials creation requests and the analysis of creation requests and responses,
- the Credentials Get analyser module, thought which one may invoke custom credentials get requests and inspect the analysed get requests and responses,
- the Virtual Authentication, able to create and retrieve virtual keys for cross domain use and authenticator simulation.

3.1 Hijacker

To analyze the WebAuthn requests at the browser level, developers normally must implement such a functionality manually for each webpage, by editing the appropriate part of the code that initiates the WebAuthn requests and display or print variables for inspection. As this approach directly involves editing the application's code, as shown in Figure 4, makes it difficult, impractical and time

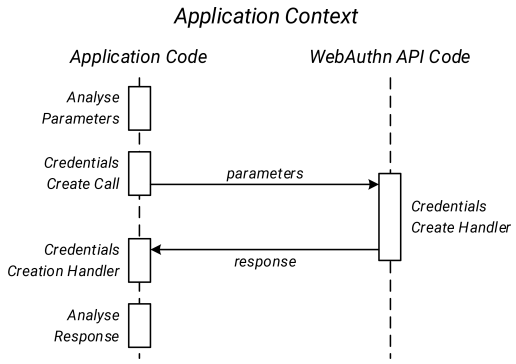

Figure 4: Code flow of an application calling the credentials create method. Any analysis will have to be implemented on the application's main code.

consuming increasing the debugging time and complexity. On top of that, such a manual debugging method lacks advanced analysis features (e.g., values decoding and unpacking), and to make matters worse, the debugging code in most cases is not portable and has to be adjusted or ported if one wants to reuse it on other web applications. To solve the above-mentioned problems, we propose an alternative, practical approach that does not require editing the application's main logic code (avoid affecting the application's functionalities) and speeding up the overall debugging and analysis of WebAuthn. Our approach can be deployed as a browser extension or a javascript script included on the web application of interest.

Through hijacking one can intercept the function calls, process the information passed to the methods, forward (if needed) the request to the real handler (by calling the original function) or even process the returned results before returning them to the caller. In our case, we are looking into hijacking the two (2) WebAuthn methods of interest, credentials create, and credentials get. We must note that these methods are asynchronous and will return a javascript promise, meaning that the WebAuthn response will not be returned instantly, thus we will have to bind a function to handle the response when the promise is fulfilled. Since our aim also includes the meaning less interaction with the already in place application code, our hijacking code will have to simulate the behaviour of the original methods. To analyse the responses, our hijacking code will have to return a new promise and handle

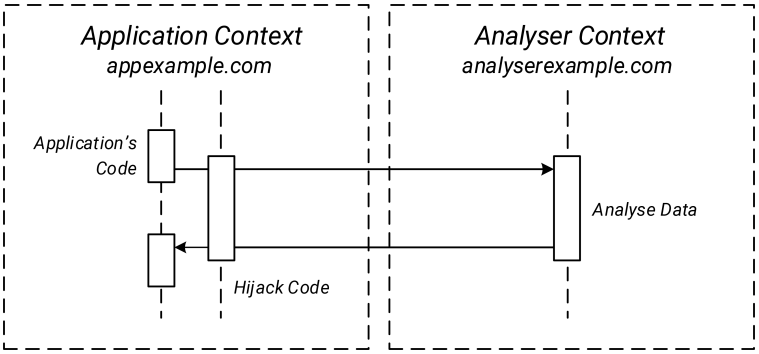

Figure 5: Analyse data through method hijacking code injection and cross domain web page communication.

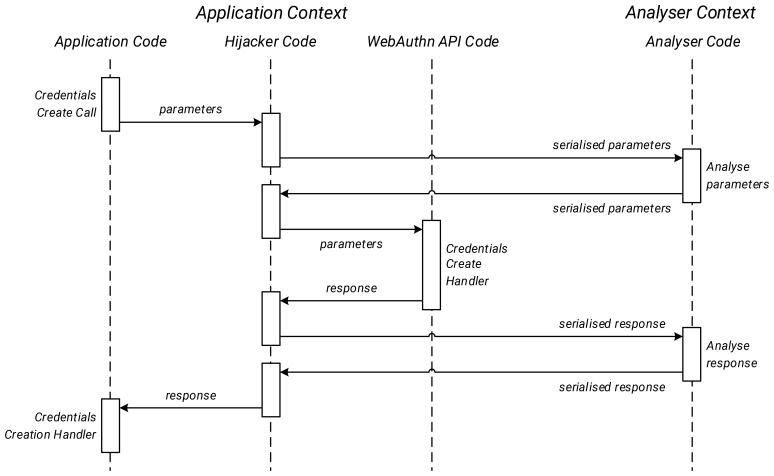

Figure 6: Proposed code flow of an application calling the credentials create method through hijacking. The analysis of the traffic can be implemented independently from the application code flow on a separate website.

the promise returned by the original method call, which then will forward through fulfilling the created promise.

In our implementation, the Hijacker module is responsible for intercepting WebAuthn calls and redirecting them towards the appropriate Analyser module. To bypass cross domain restrictions and be able to analyse requests of 3rd party WebAuthn implementations hosted under different domains of our tool, we developed a browser extension that injects the Hijacker's custom javascript code, that once deployed on the target website hijacks and forwards WebAuthn credentials create and get requests and/or responses to our web tool. Alternatively, the Hijacker's code can be injected manually on the web page through the web browser's developer tools javascript console or be inserted as a web page script on development instances of a web application (if the developer can edit the applications front end code).

3.2 External Communication Manager

To be able to present the traffic and the analysis on a graphical interface without affecting the underlining web application that we are inspecting, the captured information is piped to our analyzer located on another web site context, as shown in Figure 5. The External Communication Manager ensures this cross-website communication using appropriate methods (message posting [8]) and appropriate data serialize/deserialise. The overall flow of the code, based on our suggestion is shown on Figure 6.

3.3 Analyser Modules

The analysis of the traffic, as shown in Figure 5, can be executed both on the supplied parameters (request) and on the returned data (response) and for both credentials creation and credentials get processes. An analysis of the request parameters can give an insight into what exactly the web application is going to request from the WebAuthn method. The analyser code may try to decode the supplied WebAuthn options (public key credentials creation [9] or get options [10]) based on the WebAuthn specification. The analyser may print information such as the relying party id, the challenge generated by the server, as well as any timeout option passed. Specifically, for credential creation requests, the analyser may also print any authenticator selection criteria (e.g., platform authenticators only), the user identifier, attestation preference (e.g., indirect attestation), the ids of the already generated credentials, or even the credentials properties defined by the server (e.g., accepted

credentials algorithms). Similarly, for credentials get requests, the analyser could print any allowed credentials id or any user's verification preference (e.g., user verification discouraged).

Although the analyser may be helpful for inspecting the request parameters, it can play an even more essential role in analysing the request responses. Through the analysis of responses, it could process the packed information and decode them into a human-readable form. To start with, for both types of responses (creation and get), the analyser will be able to decode and present the client data that wraps the server's challenge and the request's type and origin. For credential creation responses, it will be able to unpack the attestation and return the generated credentials public key as well as the provided attestation data (e.g., the device's certificate).

3.4 Credential Manager

The Credentials Manager module stores and retrieves credentials between the analyser's modules. Additionally, it lets the user view information of credentials generated through the tool and allows the user to delete information of credentials that are not needed any more. An example of the presented information, for a key pair is displayed on Table 1. The tool stores the key id – an identifier linked to the created public private key pair – returned by the authenticator (this piece of information is needed in many cases, depending on the authenticator implementation, for the credentials get options generation), the user handle – an identifier linked to the user's account – defined on the passed options by the relying party and the origin under which the credentials were created (as we will explain later, our tool can generate credentials for cross domain application debugging). This credential information is saved on the browser's local Storage and are retrieved by javascript when needed.

3.5 Credentials Creation Analyser Module

The credentials creation analyser module presents a friendly user interface – featuring commonly used interface elements such as text inputs, dropdowns, 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 analysed 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 analysed 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 analysed 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).

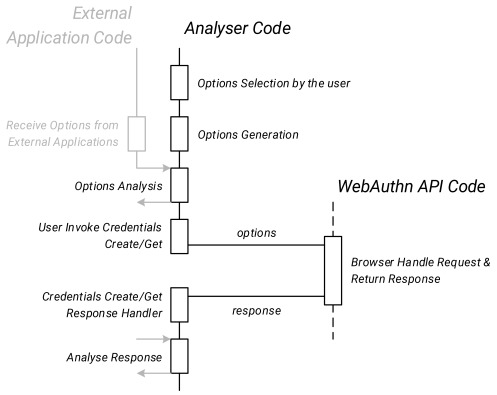

Figure 7: Technical implementation code flow for credentials create & get modules.

3.6 Credentials Get Analyser Module

The credentials get analyser module follows the same approach to the credential's creation page, as shown in Figure 7 for both modules. This module features a similar interface through which users may craft custom WebAuthn credentials get request options according to their needs. Just like the 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 analysed by the tool and it is presented to the user, following the successful retrieval of the credentials get assertion. Just like before, 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).

Our 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 our tool exposes all the information exchanged between a relying party and an authenticator device. On top of that, the tool lets developers tweak the credentials get options and live test their authenticator devices as well as the underlying client system's behaviour.

3.7 Virtual Authenticator

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 [11]). Whichever 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.

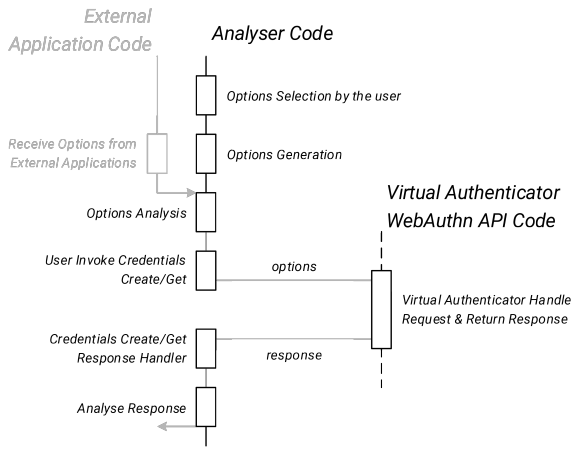

Figure 8: Technical implementation code flow for credentials create & get modules by using a virtual authenticator embedded on the analyser.

To allow our web tool to create credentials for requests originating from different domains, we needed a software authenticator device, accessible through our tool's website without restrictions, and able to answer WebAuthn (including cross origin) requests with valid responses. We needed the responses to be WebAuthn compatible so that we could forward them to 3rd party FIDO2 registration and/or authentication services as if they were crafted by a WebAuthn compatible web browser, as described in our methodology. The implementation of such a virtual authenticator will enable full handle or external applications requests and new code flows as shown in Figure 8.

To overcome any limitation placed either by the browser's security or the operating system's type, we based our virtual device on javascript, making it able to run seemingly on our analyser's web page context provided that the browser supported the Web Cryptography API [12]. Thus, our implementation can run on any modern browser. The use of the web browser's cryptography API allows the fast operation of the authenticator without any noticeable delay.

The virtual authenticator was initially developed as a supported mobile for WebDevAuthn, additionally, due to the portability of the code, it can also be used for malicious purposes, for example after exploiting a XSS vulnerability on a web application, to generate and bind additional custom authenticator devices under victim account, that could later be used as a backdoor or for 2nd factor bypassing.

3.7.1 Credentials. In terms of public keys generation, our virtual authentication implementation is currently able to generate and use Elliptic Curve Digital Signature Algorithm (ECDSA) with Secure Hash Algorithm (SHA) 256 keys (ES256, registered in the IANA COSE Algorithms [13] registry with the code -7) for signing. It can be easily upgraded to support other algorithms – since several asymmetric encryption algorithms are supported by Web Cryptography API – such as the ES256 for compatibility with legacy services, or with bigger hash size as the ES384 and the ES512 for increased security.

3.7.2 Key Wrapping. To support an unlimited number of keys and also eliminate the need for large storage resources on the authenticator device, authenticators may secure wrap information that could be used to recover the generated credentials and return it to the FIDO2/WebAuthn server as a key id. During authentication, the relying party server will return several key ids registered and thus the authenticator will be able to recover the private key needed to sign the challenge. This approach cannot be used for discoverable credentials (through resident keys creation), as for those the relying party server will not return any key id. Many security FIDO2/FIDO-U2F key manufacturers are using this approach so that the keys do not run out of storage. This is mostly preferred for second factor authentication (FIDO U2F) since the user is already logged in and the relying party can easily retrieve the key ids associated with the user's account.

For our virtual authenticator implementation, to eliminate the need for storing the private key, we are wrapping it (along with other authenticator data) in an AES-GSM with a 256-bit key size. To ensure the recovery of the private key, the recovery of the virtual authenticator state and the future compatibility of the generated credentials, the following information are wrapped to generate the key 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 Algorithm Code
- Private Key Values
- Credentials Creation Date

3.7.3 Signature Counter Simulation. To generate a valid attestation 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 1 value 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.

3.7.4 Authenticator's Master Key. Our implementation accepts a custom secret during initialization that is used to generate the authenticator's master key. To generate this strong master key, the given secret is passed through a Password-Based Key Derivation Function 2 (PBKD2) using SHA-256 and enough iterations. The master key is used by the authenticator as a key to the AES-GSM for the private key wrapping. Additionally, initialising the authenticator with another custom secret will result in generating a new authenticator device, able to be used on the same accounts as other instances of our virtual authentication.

4 PROPOSED TOOL EVALUATION

We tested our tool by using it in many cases for our research on FIDO. We found it to be helpful, especially during the development

Table 1: Compatibility of our virtual authenticator implementation with popular demo FIDO2/WebAuthn web applications.

Compatibility	Web Application	Comment
✔ YES	https://webauthn.io	
✗ NO	https://webauthn.org	Demo doesn't support packed attestation
✔ YES	https://www.passwordless.dev	
✔ YES	https://webauthn.me	
✔ YES	https://demo.yubico.com/webauthn	
✔ YES	https://webauthn.bin.coffee	
✔ YES	https://webauthn.lubu.ch	
✔ YES	https://debauthn.tic.udc.es	
✗ NO	https://magicauth.net	Demo doesn't support packed attestation

of FIDO2/WebAuthn solutions (whether they are just an application featuring WebAuthn authentication or a fully featured FIDO2 server), when testing existing solutions as well to demonstrate and explain the FIDO/WebAuthn internals to fellow developers and students.

4.1 Use Case A: Getting familiar with WebAuthn API

We used our web tool as a playground for developers interested in password-less authentication, to create custom WebAuthn requests and understand the authenticator responses. Since our implementation features a virtual authenticator, it can be used by developers without FIDO2, FIDO U2F or platform authenticators.

The new members of the team were able to select and configure various request options from the tool's UI, as shown on Figure 9. The options of preference could then be used to automatically generate the corresponding WebAuthn request API call code [14].

The generated code can then be used to invoke WebAuthn, and request response from an authenticator device. For example, a custom credentials create request can be launched and the tool will retrieve and analyse the newly created credentials, as presented on Figure 10. Since our web tool features both a credentials creation playground and a credentials retrieval one, which both features a simple user interface and appropriate analysis of the requests and responses, the developers were able to play with both credentials' creation and get.

4.2 Use Case B: Testing Analyser & Virtual Authenticator

We used our web tool during the development of our experimental demo FIDO2 web application, for debugging the implementation. We found our tool's virtual authenticator to work with the certified Strongkey FIDO2 server [15] and with an experimental FIDO2 server we are developing based on Yubico's FIDO2 python library [16]. To further validate our web tool and its virtual authenticator device, we also tested it on a number of demos webauthn applications publicly available online. Table 1 lists the publicly available demo websites that we tested our virtual authenticator on.

Our analyser in combination with our virtual authenticator was found to work with most implementations. During the development of our FIDO implementations, we found our tools ability to analyse the WebAuthn traffic helpful for debugging and validating the correct behaviour or the system. Our tool's virtual authenticator was proved to be handy for testing implementations even without a FIDO2 authenticator. By tweaking the configuration of the tool's authenticator, we were also able to reproduce bugs and test our systems response under specific inputs. The only incompatibilities found are associated with the server's backend not supporting our virtual authenticator's self-signed packed attestation.

5 CONCLUSIONS

The paper presented a methodology to analyse WebAuthn requests and responses by injecting javascript code into the web application to be debugged, hijacking the WebAuthn methods, and forwarding it to our analyser, in a non-invasive way without affecting the application code flow. The introduced methodology, and thus our technical implementation, can be used by developers to analyse the WebAuthn traffic of their applications and possibly speed up the debugging process as it will provide direct insight into the information which is packed inside the WebAuthn requests and responses. Thus, we stress that the work presented will help developers get familiar with and understand deeply the FIDO2/WebAuthn javascript requests and responses. The presented technical implementations and the associated analysis proved that our tool is compatible with every implementation of FIDO2/WebAuthn we tested. The prototype virtual authenticator implementation was found to work with many FIDO2/WebAuthn backends and any incompatibility was associated with the server backends' support of FIDO2/WebAuthn attestation formats and policies.

The tool presented in this paper is an ongoing work. Future developments and testing will be focused on more attestation formats decoding and more attestation certificates. Additionally, the tool will be upgraded to allow on the fly edits of the requests and the responses through an easy interface, as now this functionality is only available by accessing the data objects from the javascript console. This would allow the easier testing of FIDO2 implementations as testers will be able to alter the requests and the responses sent

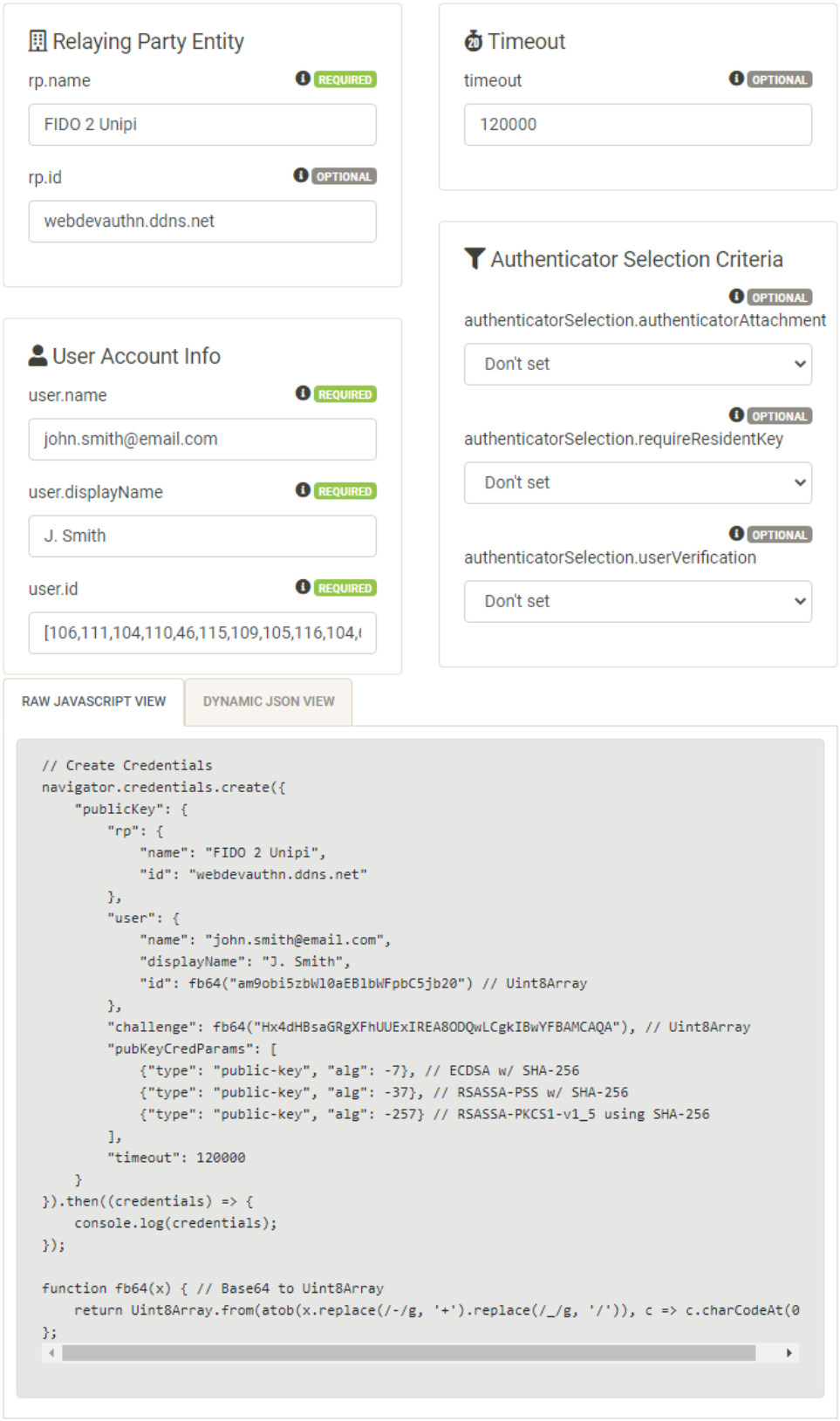

Figure 9: Credential's creation options user interface (left) and Credential's creation options generated through user interface (right).

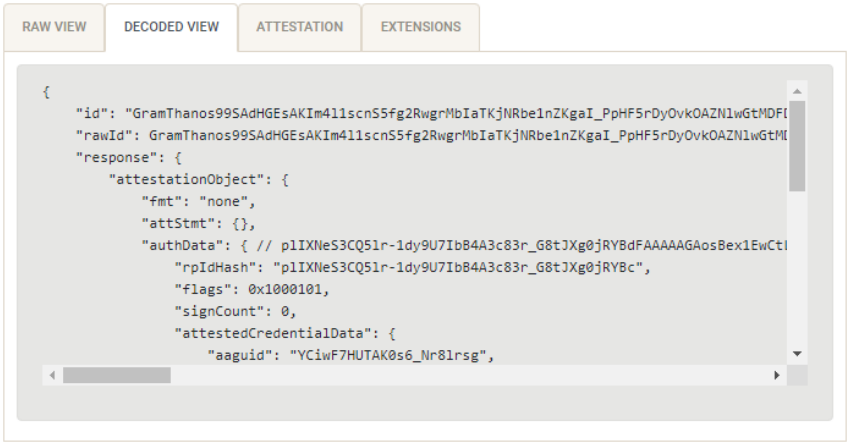

Figure 10: Credentials creation response generated through user interface on virtual authenticator.

to the server. For the same reasons, we are planning to upgrade our virtual authenticator implementation, extending the support of asymmetric cryptography algorithms, including quantum resistant algorithms to provide a future-proof proof-of-concept FIDO2 authenticator.

ACKNOWLEDGMENTS

This research has been partially funded by the European projects: the Greek national project NetPHISH (Operational Programme Competitiveness, Entrepreneurship, and Innovation 2014-2020 (EPAnEK) - T1EΔK-05112) and INCOGNITO (Horizon H2020 Framework Programme of the European Union under Grant Agreement ID 824015). Furthermore, we would like to thank all the members of the Systems Security Laboratory (SSL) of the University of Piraeus, for their testing and feedback on improving the technical implementation featured in this paper.

REFERENCES

[1] .M. Bromiley, “Bye Bye Passwords: New Ways to Authenticate,” SANS Report, July 2019, https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RE3y9UJ
[2] A. Angelogianni, I. Politis, F. Mohammadi and C. Xenakis, "On Identifying Threats and Quantifying Cybersecurity Risks of Mnos Deploying Heterogeneous Rats," in IEEE Access, vol. 8, pp. 224677-224701, 2020.
[3] FIDO Alliance - Open Authentication Standards More Secure than Passwords, https://fidoalliance.org/
[4] K. Papadamou *et al.*, "Killing the Password and Preserving Privacy With Device-Centric and Attribute-Based Authentication," in IEEE Transactions on Information Forensics and Security, vol. 15, pp. 2183-2193, 2020, doi: 10.1109/TIFS.2019.2958763.
[5] M. Jones, R. Lindemann, A. Kumar, J. Hodges, J.C. Jones, H. Liao, A. Czeskis, E. Lundberg and D. Balfanz, “Web Authentication:An API for accessing Public Key Credentials Level 1,” W3C Recommendation, March 2019, https://www.w3.org/TR/2019/REC-webauthn-1-20190304/
[6] A. Simons, “A breakthrough year for passwordless technology,” Microsoft Article, December 2020, https://www.microsoft.com/security/blog/2020/12/17/a-breakthrough-year-for-passwordless-technology/
[7] M. West, “Credential Management Level 1,” W3C Working Draft, January 2019, https://www.w3.org/TR/2019/WD-credential-management-1-20190117/
[8] Window.postMessage() - Web APIs | MDN, https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage
[9] PublicKeyCredentialCreationOptions - Web APIs | MDN, https://developer.mozilla.org/en-US/docs/Web/API/PublicKeyCredentialCreationOptions
[10] PublicKeyCredentialRequestOptions - Web APIs | MDN, https://developer.mozilla.org/en-US/docs/Web/API/PublicKeyCredentialRequestOptions
[11] Microsoft, “Win32 APIs for WebAuthN standard”, GitHub Repository, October 2018, https://github.com/microsoft/webauthn
[12] Mark Watson, “Web Cryptography API,” January 2017, https://www.w3.org/TR/2017/REC-WebCryptoAPI-20170126/
[13] CBOR Object Signing and Encryption (COSE), https://www.iana.org/assignments/cose/cose.xhtml
[14] Dirk Balfanz, Alexei Czeskis, Jeff Hodges, J.C. Jones, Michael B. Jones, Akshay Kumar, Angelo Liao, Rolf Lindemann, and Emil Lundberg. 2019. Web Authentication: An API for accessing Public Key Credentials Level 1 . Technical Report. https://www.w3.org/TR/webauthn
[15] StrongKey, “Open-source FIDO server, featuring the FIDO2 standard”, GitHub Repository, October 2019, https://github.com/StrongKey/fido2
[16] Yubico, “Python FIDO2 - Provides library functionality for FIDO 2.0, including communication with a device over USB.”, GitHub Repository, October 2018, https://github.com/Yubico/python-fido2
[17] M. R. Dourado, M. Gestal, and J. M. Vázquez-Naya, “Implementing a Web Application for W3C WebAuthn Protocol Testing,” Proceedings, vol. 54, no. 1, p. 5, Aug. 2020 [Online]. Available: http://dx.doi.org/10.3390/proceedings2020054005
[18] M. Rivera, “WebAuthn Authenticator Debugging Tool,” DebAuthn. [Online]. Available: https://debauthn.tic.udc.es/. [Accessed: 06-Jun-2021]
[19] Auth0 Inc., See your WebAuthn config in action. [Online]. Available: https://webauthn.me/debugger. [Accessed: 06-Jun-2021]
[20] N. Steele, “A demonstration of the WebAuthn specification,” WebAuthn.io. [Online]. Available: https://webauthn.io/. [Accessed: 06-Jun-2021]
[21] M. Miller, “MasterKale/webauthn-previewer,” GitHub. [Online]. Available: https://github.com/MasterKale/webauthn-previewer. [Accessed: 06-Jun-2021]
[22] M. Miller, “WebAuthn Debugger,” SimpleWebAuthn. [Online]. Available: https://debugger.simplewebauthn.dev/. [Accessed: 06-Jun-2021]
[23] S. Weeden, “sbweeden/fido2viewer,” GitHub. [Online]. Available: https://github.com/sbweeden/fido2viewer. [Accessed: 06-Jun-2021].