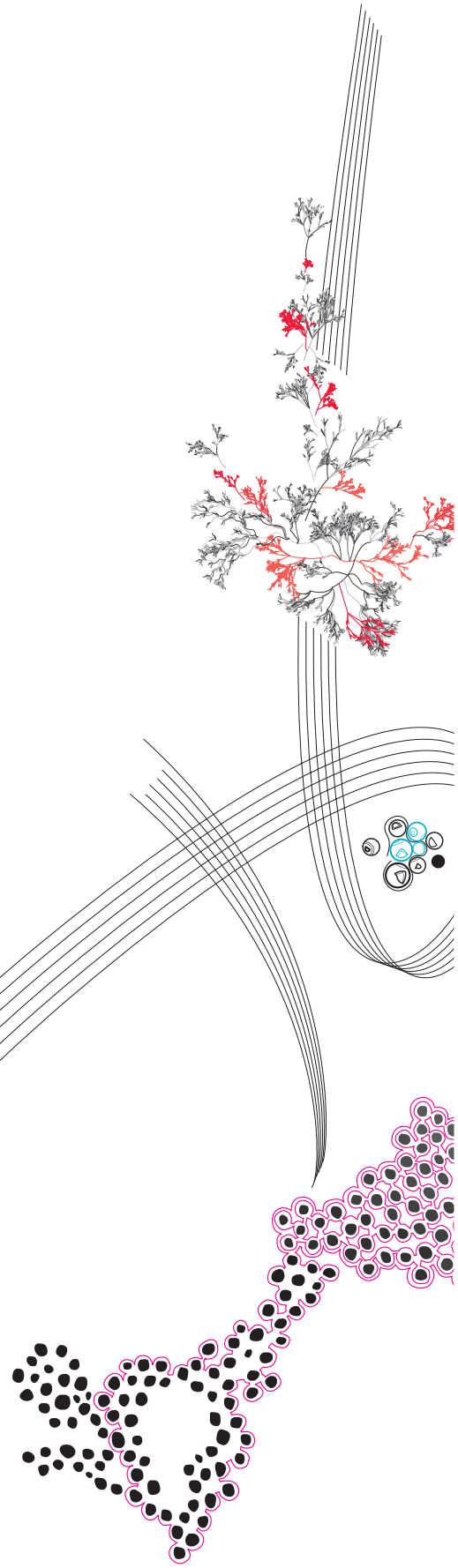

MSc Computer Science
Research Topic

Vulnerability Testing for WebAuthn

Peizhou Chen

Supervisors:
Luca Mariot (SCS, UT)
Matthijs Melissen (Computest)
Roland van Rijswijk-Deij (DACS, UT)

March 2024

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

UNIVERSITY OF TWENTE.

Abstract

Authentication is crucial to maintain confidentiality on the web. Passwords are vulnerable to data leaks, reuse, and phishing attacks. To mitigate these, FIDO2 offers a more secure and easier way to authenticate. However, the complexity of the protocol might leave developers with wrong assumptions, leading to vulnerable implementations. Currently, there are not many tools available to assess server-side misconfigurations of FIDO2 implementations. This thesis proposes a novel methodology to create a Burp Suite extension that can be used in black-box penetration testing engagements. We implement a tool that identifies and addresses misconfigurations, along with exploring potential attack vectors. This can enhance the testing efficiency by automatically scanning for twenty-five test cases. We have conducted tests on five public websites, in which three did not fully comply with the security practices of the official WebAuthn standard. Furthermore, one of these misconfigurations led to a full account takeover. We also introduce a novel Cross-Site Request Forgery attack tailored to exploit the improper session management of web services during WebAuthn operations. This attack can result in a complete takeover of user accounts. Our testing covered thirteen websites, with one being vulnerable to this attack. This work contributes to a better understanding of FIDO2/WebAuthn implementation security. We aim to provide insights into the potential security risks associated with the incorrect implementations of this promising protocol.

Keywords: FIDO2, WebAuthn, Burp Suite, Cross-Site Request Forgery, security, vulnerability.

Contents

Chapter 1

Introduction

Authentication is crucial to maintain confidentiality on the web. From accessing an online service to unlocking a password manager, most of the authentication processes require (or, required) a user to remember a secret, i.e., *something-you-know*, the password.

Due to human nature, people tend to create short and/or guessable passwords, using common words and numbers, to facilitate the authentication procedure [11]. A malicious actor can take advantage of this by using password-guessing techniques facilitated by dictionaries of commonly used passwords to easily guess a user's credentials and gain unauthorized access to one of their account. Moreover, data breaches happen frequently and compound this precarious landscape [22]. For instance, the "Collection #I" data breach [23] exposed nearly 773 million unique email addresses and passwords. Such incidents amplify further the risk of unauthorized access. Because most users re-use passwords across multiple services [9], the outcome poses significant threats to the security of numerous accounts and therefore the confidentiality of sensitive information.

To mitigate easily-guessable passwords and password reuse, password managers are an effective solution. These help users create unique pseudo-random passwords for each service the user signs up for. Nevertheless, previous work shows that, due to poor usability and general low attention towards security, users are inclined not to adopt a password manager in their daily routines [12] [52]. Regardless, passwords are still vulnerable to more advanced attacks [49], such as targeted spear phishing [48] and password-busting [41].

To mitigate unauthorized access, several services suggest or enforce the use of Two-Factor Authentication (2FA). 2FA adds an extra layer of protection, as it adds to *something-you-know* (i.e., the password), a *something-you-are/have*, such as biometrics scanning, push notifications, Short Message Services (SMSes), Time-based One Time Passwords (TOTPs), etc. Should 2FA be adopted by a user, after an initial successful login with credentials, an ulterior verification challenge is sent to their personal device (mobile phone, laptop, etc.). It would be more challenging for an attacker in this case, in possession of only the credentials of a victim, to gain unauthorized access to their account. Unfortunately, 2FA also causes ulterior friction in the authentication process, leading to only 32% of users having it set up in 2021 [8].

Nevertheless, it was only a matter of time before new attacks against the second factor were discovered. SMSes and TOTPs, for example, could be captured also with a real-time phishing [24] or a SIM swap attack [27]. Push notifications can be also tampered with by the push fatigue attack and be compromised with a timing attack [25]. On the other hand, hardware tokens are more resistant to phishing attacks due to their design, but users often do not rely on them, again, due to low usability [8].

The main issues with passwords are that they represent a symmetric secret shared

between a client and a server and that they are not usually unique per account [9]. To address these, in 2013 the FIDO Alliance was founded to reduce the reliance of the authentication process on passwords. Instead of adding another "band-aid", the new protocol(s) should be secure by design. The latest protocol, FIDO2, defines a password-less authentication scheme based on asymmetric cryptography. To be widely adopted, FIDO2 aims to be user-friendly by leveraging the biometrics readers of commonly used devices, such as smartphones and laptops. As these become ubiquitous, they can facilitate users adopting this new technology in their daily authentication routine. Users can access their online account with a single gesture (e.g., face scan, fingerprint scan), which proves both factors *something-you-have* and *something-you-are* simultaneously.

1.1 Motivation

In 2021, there exist more than 4 billion FIDO-supporting devices and around 150 million people using password-less methods each month [17]. The State of Authentication Report 2023 [40] states that around 65% of respondents plan to adopt password-less technologies in the next 24 months. As this technology becomes more adopted, we are interested to see what attack vectors exist that can tamper with the intended behavior of the protocol. The choice of removing passwords from authentication should be carefully understood and handled by developers before implementing FIDO2. Alam et al. [3] show that previous security-critical technologies, such as TLS, password storage, and cryptography APIs, have had pitfalls in their adoption. The authors research the state of FIDO2 specifications, developers' forums, and open-source libraries and outline that the most common issues are:

- The complexity of the protocol leaves developers with doubts and sometimes wrong mental models,
- the libraries are incomplete and poorly documented,
- playgrounds to implement FIDO2 lack and do not include security- and privacy-related issues.

From this, we are motivated to research whether services that implement FIDO2 have misconfigurations that can lead to potential vulnerabilities. In particular, we collect a list of operations a server should or must do in order to be compliant with the security goals of the protocol. We hypothesize that developers who integrate custom FIDO2 authentication into their services may make mistakes when implementing this technology due to wrong assumptions or the complexity of the protocol. In this work, we also hypothesize the reasons behind any design that can lead to potential misconfigurations.

1.2 Research questions

With the increase of services implementing FIDO2, we are interested in seeing the design choices and the security posture of the current implementations by analyzing their conformance to the protocol. We focus only on the client-server interactions of the protocol, which is defined by the Web Authentication (WebAuthn) standard, and leave out the communication between the client and the authenticator. We notice that in the security industry, there is a lack of tools that can test externally the conformance and the security of the FIDO2/WebAuthn implementations. In this work, we aim to create a tool that

can do black-box testing of the FIDO2/WebAuthn implementations and that can be used during practical penetration testing engagements. We choose to use Burp Suite to help us with testing, as it is commonly used for web security testing and it is customizable by means of extensions.

To summarize, our goals are:

- understanding how to implement a comprehensive FIDO2/WebAuthn testing tool,
- using this tool to conduct automated scanning to detect eventual misconfigurations,
- gaining some insights into the security posture of the current implementations of FIDO2/WebAuthn.

We also yearn to find new attack surfaces in FIDO2/WebAuthn by exploiting the wrong security assumptions that developers can make. We propose the following research questions that this work aims to answer:

- **RQ1**: How can we ease and automate the testing process for FIDO2/WebAuthn misconfigurations in web applications?
- **RQ2:** What are the design choices of current implementations of FIDO2/WebAuthn and how do they comply with the security best practices?
- **RQ3:** What types of vulnerabilities in FIDO2/WebAuthn relying parties exist and how can we exploit these?

To approach **RQ1**, we begin with understanding how FIDO2/WebAuthn operations work and what makes a WebAuthn request valid. We then repeat these valid requests with non-ordinary values to see how a server interprets these requests. Based on its response, we can evaluate whether the server is compliant or not with the protocol specifications. Our end goal is to create an automatic scanner that, given a website implementing FIDO2/WebAuthn, can assess the correctness of its implementation based on our test cases. We define a test case as a value that we modify in the data objects sent to the server to see the latter's behavior.

To answer **RQ2**, we plan to use the scanning tool against both demo websites/playgrounds and commercial services that implement WebAuthn. We make sure to target websites that have a responsible disclosure policy. We then collect and show the results, and comment on the design choices of the developers and whether the eventual misconfigurations can lead to vulnerabilities.

We tackle **RQ3** by looking into common web security attacks and designing an exploit that could interfere with FIDO2/WebAuthn's normal execution flow. We plan to target websites that have some WebAuthn misconfigurations and chain these with web security exploits to see if new vulnerabilities can be discovered.

1.3 Contribution

This research contributes to the academic literature in three ways:

- we present a novel methodology to implement a Burp Suite extension that automatically scans for FIDO2/WebAuthn misconfigurations. Our objective is to create a useful tool that can assess FIDO2/WebAuthn's security during penetration testing engagements. With it, security specialists can provide a comprehensive view of the password-less authentication method and be alerted if there is any server-side miss-check.

- we validate our methodology by utilizing the tool against publicly available websites. We give an overview of the current WebAuthn implementations landscape through comprehensive test cases and analysis.

- we comment on whether our findings can lead to security vulnerabilities. In this case, we craft proof of concepts of practical attacks, demonstrating the feasibility and the consequence of a successful exploitation. We also execute Cross-Site Request Forgery attacks, which can potentially lead to a full account compromise in case the session management of WebAuthn operations is not done correctly.

1.4 Computest

This thesis is realized with the collaboration of Computest[1]. Here I would like to thank my colleagues who helped to understand web security in more depth, my supervisor, Matthijs Melissen, who provided me with great insights into every doubt I had, and Thijs Alkemade, who gave me great tips about security research.

Computest was founded in 2005 and today it offers solutions regarding security. It offers 360-degree services, including security assessments, Security Operations Center (SOC) monitoring, research, and business consulting. The company is one of the leading security companies in the Netherlands and its customers include Talpa, AFAS software, and Koninklijke Nederlandse Golf Federatie (NGF). Currently, the company holds around 70 employees. The company is also certified with relevant standards, such as ISO9001 and ISO27001.

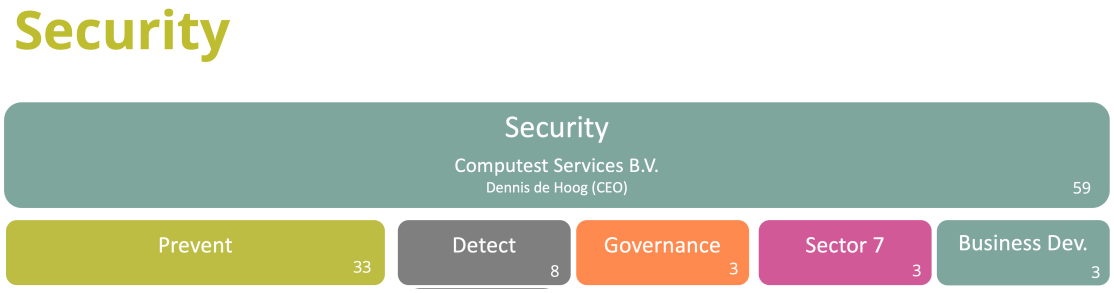

FIGURE 1.1: Computest Security

Computest has 5 business units, as shown in Figure 1.1. For this project, I was assigned to one of the Prevent business unit teams and worked closely with other security specialists.

Other business units include Detect, Governance, Sector 7, and Business Development. The following is a brief description of which activities each business unit is involved in:

- Prevent: this performs white hat activities, such as security assessments (Cloud, IT infrastructure), pentests, red teaming, etc. Its goal is to fortify the security of the client's organization by finding weaknesses before an attacker can.

- Detect: this includes a Security Operational Center (SOC) and an Incident Response team. It allows customers to have a 24/7 updated view of possible threats in their network.

- Governance: this unit's main activity is business consulting.

[1] https://www.computest.nl/en

- Sector 7[2]: this is the research team at Computest. In the past, it found important vulnerabilities in Zoom [1] and MacOS [2]. This shows how the company is up-to-date with the latest technologies and this was one of the reasons why I chose to join them.
- Business Development: this includes the teams that help Computest grow among its competitors. Its teams are recruitment and marketing.

We collaborate on this project because Computest expects its clients to start using FIDO2 protocols soon. With this work, we expect to create a tool that can find misconfigurations in the implementation of WebAuthn and provide a more complete security assessment. This project also provides a general idea about the security standpoint of the protocol.

1.5 Structure

This section gives an overview of the content of this thesis. Chapter 2 presents the necessary low-level background knowledge regarding FIDO2/WebAuthn and Cross-Site Request Forgery attacks. After this, we give the details of our methodology for automated scanning and executing Cross-Site Request Forgery attacks in Chapter 3. Chapter 4 shows the results of the testing. Chapter 5 discusses the results and the limitations of this work. This paper in Chapter 6 concludes with our vision and comments on this revolutionary technology. We include an extra chapter about the ethics of our research in Chapter 7.

[2]https://sector7.computest.nl/

Chapter 2

Background

This chapter explains all the preliminaries necessary for this thesis.

In the first section, we provide details of the FIDO2 protocol and its sub-protocol, WebAuthn. In order, we present the actors involved in the protocol and how the registration and authentication ceremonies work. We denote all the low-level theoretical notions needed for the continuation of this work. Then we present the security and privacy analyses of FIDO2 and WebAuthn.

In the second section, we explain what Cross-Site Request Forgery attacks are and how can these be executed on the FIDO2/WebAuthn ceremonies.

In the final part, we analyze the existing related work about the protocols. In particular, we study the formal verification of the security of the FIDO2 and WebAuthn, the attacks proposed by other authors, and explore work that is similar to ours.

2.1 FIDO2

The Fast ID Online (FIDO) Alliance[1] is an open industry that works on improving interoperability among authentication technologies and on reducing the reliance of these on passwords. The latest standard of FIDO, FIDO2, allows easier password-less authentication by leveraging common devices, such as mobile and desktop environments [14]. With this standard, users can log in to their accounts by using simple gestures, such as a biometric scan or tapping on a security token. FIDO2 replaces passwords with asymmetric keys. During the registration phase, a key pair is created on the user's device, and the public key is sent to the server. The private key is used to sign messages to prove the user's identity and never leaves the user's device.

FIDO2 provides four main benefits on its adoption [14].

1. It enhances **security** as it creates unique, complex, and scoped credentials and does not publish the private key (unless encrypted), to the service. By these means, common attacks such as phishing, replay attacks, and data breaches are fully mitigated.

2. It has a **user-friendly** design, as simple gestures (fingerprint scan, biometrics, etc.) are used to unlock users' credentials.

3. FIDO2 is also **privacy-preserving** since the credentials identifiers are unique for each service, so honest-but-curious servers cannot track users across sites.

[1] https://fidoalliance.org/

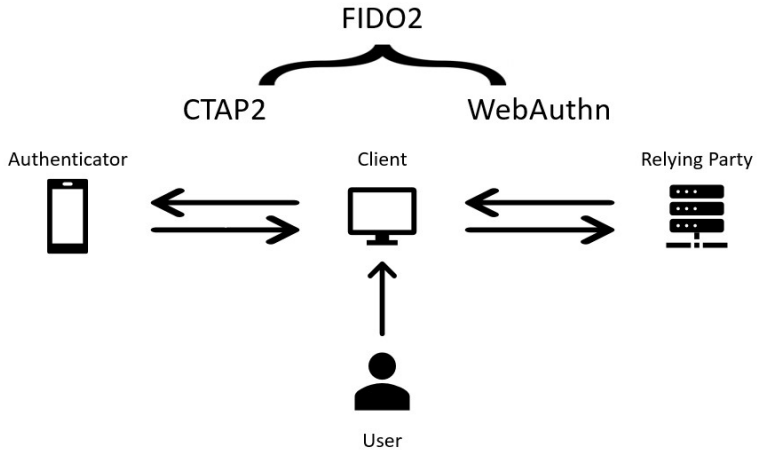

FIGURE 2.1: FIDO2

4. It is **scalable**, as browsers and service providers can easily integrate WebAuthn, and most of the major vendors and platforms already support it.

Figure 2.1 depicts the whole architecture of FIDO2. In the following section, we define and describe the involved parties: the relying party, the authenticator, and the client.

2.1.1 Actors

Before diving into the details of the standard, we define the actors involved in FIDO2, which are the **relying party**, the **authenticator**, and the **client**. We describe in more detail each component.

Relying Party

The relying party (RP) is the service that a user wants to interact with. It is usually a web application and in this context, it is able to process FIDO2 registration and authentication requests on its back-end. Each relying party is identified to the authenticator by its Relying Party Identifier (RP ID). By default, this corresponds to the origin's effective domain [50]. For instance, the origin of the URL `https://login.example.com:1337` is `login.example.com`.

Authenticator

An authenticator is a piece of software or hardware that helps authenticate the user who possesses it. It can also be referred to as an authenticator device, token, or security key. Authenticators create and store credential maps, which map the relying party to the created credential. Its primary function is to provide WebAuthn signatures.

There currently exist two classes of authenticators: **platform authenticators** and **roaming authenticators**. Figure 2.2 and Figure 2.3 show examples of them, respectively. Platform authenticators are usually not removable from the client's device and they

FIGURE 2.2: Platform authenticator

FIGURE 2.3: Roaming authenticators

communicate with the client by platform attachment. Some examples are Windows Hello, iOS TouchId/FaceId, Android's biometrics reader, etc.

A roaming authenticator, on the other hand, is an external hardware device that communicates with the client via cross-platform attachment. Authenticators of this class are removable from and can "roam" between client devices using Universal Serial Bus (USB), Near-Field-Communication (NFC), or Bluetooth-Low-Energy (BLE).

Authenticators store a signature counter value. This value is synchronized between the authenticator and the relying party and is used to prevent cloned authenticators. After each FIDO2 operation, both parties increase their value (by 1). In case an attacker clones the private key(s) of an authenticator, they would not be able to authenticate correctly if their signature counter value is not greater than the relying party's. We note that the WebAuthn standard [44] does not specify how much greater the authenticator's signature counter should be. We comment that this is not entirely secure, as an adversary can simply choose a very high counter value and bypass this check with their cloned authenticator.

In cases where a roaming authenticator has storage limitations, it can use a *key wrapping* technique to "outsource" the generated private keys to the relying party. Briefly, the roaming authenticator holds a master key, which is unique to each one and bounded during the manufacturing phase. When a new key pair is generated, the new private key and the hash of the RP ID are encrypted with the master key and are exported to the relying party for storage. We note that authenticators that use key wrapping are vulnerable to device cloning. Kang [26] provides a proof of concept of a possible attack that can be done. Briefly, an attacker can create a virtual authenticator with the cloned master key and give the original security key to a victim. The victim registers it in one of their online accounts. The attacker can now use the cloned customized authenticator with a large signature counter to log in on behalf of the victim.

Client

We define the client as the interface that a user uses to communicate with the online services. Typically, these are browsers. In FIDO2, clients act as proxies, as they forward communication between the relying party and the authenticator. Specifically, relying parties execute the WebAuthn methods on the client, which prompts the authenticator via the CTAP2 protocol.

2.1.2 Specification

FIDO2 has two integral components: Web Authentication (WebAuthn) and Client-To-Authenticator Protocol v2 (CTAP2). The former is a standard for a JavaScript API, developed by the World Wide Web Consortium (W3C) [2], with which the FIDO alliance collaborates. It defines two JavaScript methods in the client in order to create public key pairs and compute cryptographic signatures with the created private keys. We describe WebAuthn in more detail in Section 2.2, as it is relevant to this work.

The CTAP2 protocol defines communication between the client and the authenticator. CTAP2 communications are all encoded using Compact Binary Object Representation (CBOR) because some transport methods (e.g., Bluetooth Smart) are bandwidth-constrained [16]. CBOR, indeed, is limited to at most four levels of any combination of CBOR maps and/or CBOR arrays. Serialization formats such as JSON do not have length limits and so are not suitable for resource-constrained devices.

Unfortunately, CBOR-formatted data [16] are not human readable and there are limited resources on decoding the data in a way tailored to WebAuthn. We explain in Section 3.1.1 how to create a decoder to display WebAuthn CBOR format in a human-readable way.

2.2 WebAuthn

In this section, we explain the details of one of the two sub-protocols of FIDO2, WebAuthn. Web Authentication, or WebAuthn, is a W3C standard that defines an Application Programming Interface (API) for password-less authentication, based on asymmetric keys. To this day, "Web Authentication: An API for accessing Public Key Credentials - Level 2" [44] is the latest public standard version, while Level 3 [47] is still a public draft. WebAuthn allows web browsers to integrate FIDO2, as it enables web applications to call client-specific functions to create and store credentials. WebAuthn defines two JavaScript methods: `navigator.credentials.create` and `navigator.credentials.get`, which are implemented in the clients. These are invoked by the relying parties during registration and authentication ceremonies, respectively. When invoked, the client then communicates with the authenticator via the other FIDO2 sub-protocol, CTAP2. The former is a method to created *scoped* credentials. This means that the generated key pair can only be used under the origin it has been created on. Indeed, when the client executes `navigator.credentials.get` to authenticate and prove the ownership of the private key, the credential checks if the request comes from the correct origin. In practice, this prevents a private key registered under `example.com` from being used on `attacker.com`. In this way, a phishing website cannot obtain any information about the credentials related to the legitimate one. Furthermore, to ensure integrity and confidentiality, web browsers expose the API only under a so-called secure context, i.e. the use of HTTP over TLS (HTTPS).

Following is a detailed explanation of the two ceremonies in WebAuthn: **registration** and **authentication**.

2.2.1 Registration

Figure 2.4 shows the registration ceremony flow. We assume that a user is registering their authenticator to a service and starts the registration flow. This is composed of the following steps:

[2] `https://www.w3.org/`

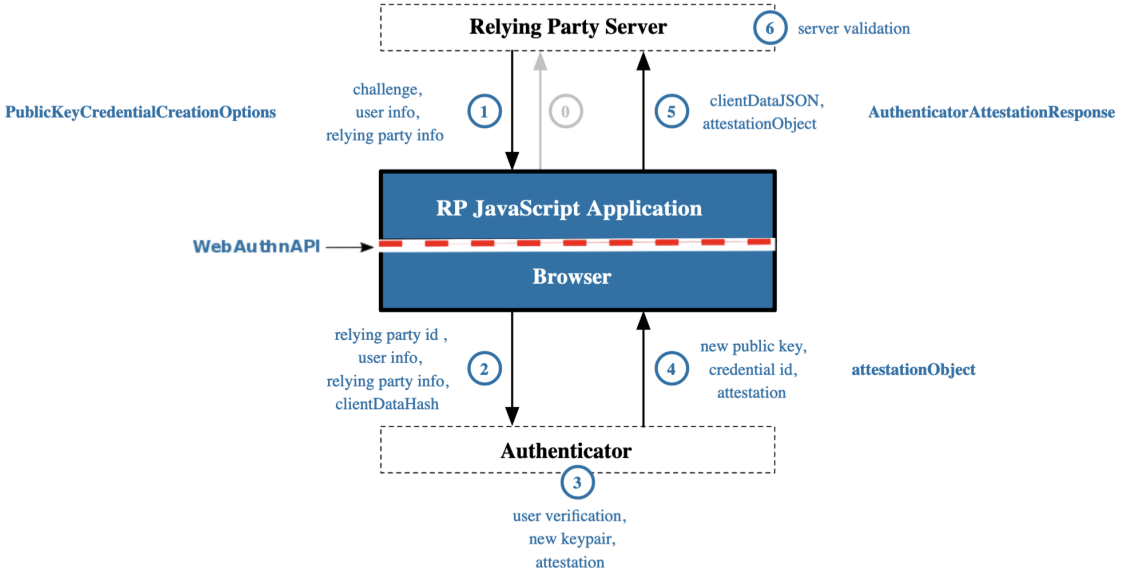

FIGURE 2.4: WebAuthn registration flow [44].

0. The user initiates the WebAuthn registration flow. In this work, we refer to this request as the **setup request**.

1. The relying party invokes the `navigator.credentials.create` method on the user's client, with arguments a high-entropy random challenge, the RP ID, the user ID, and the cryptographic algorithms that it supports. The full list of properties can be found in the Credential API documentation[3].

2. The client creates the `clientDataJSON` object which includes:

   - the received **challenge**
   - the **origin** of the relying party
   - the **type** of the ceremony (in this case, the literal string "webauthn.create")

   The client then sends the user- and relying party-related information, the hash of the `clientDataJSON`, and the RP ID to the authenticator via CTAP2.

3. The authenticator must verify that the user is in its proximity (by using BLE, NFC, or tapping on a button) and optionally verify that they can unlock the authenticator with a gesture (PIN, biometric, etc.) depending on the relying party's preference. It then creates the key pair and scopes the RP ID to it.

   The authenticator creates an authenticator data object, which includes:

   - The hash of the RP ID.
   - The flags, which are bits indicating the authenticator's status:
     - `UserPresence (UP)`: if `true`, it means that the user is near the authenticator.

[3] https://developer.mozilla.org/en-US/docs/Web/API/CredentialsContainer/create#publickey_object_structure

  - `UserVerification (UV)`: if `true`, it means that the user is authorized to use the authenticator (for example, by typing a PIN).
  - `Attested credential data`: if `true`, it means that the authenticator data added attestation.
  - `Extension data included`: if `true`, it means that the authenticator data has extensions.
- The credential ID (or key ID).
- The public key, in CBOR Object Signing and Encryption (COSE) format.
- The AAGUID, an identifier chosen by the authenticator's manufacturer.
- The initial signature counter.
- Optional extensions.

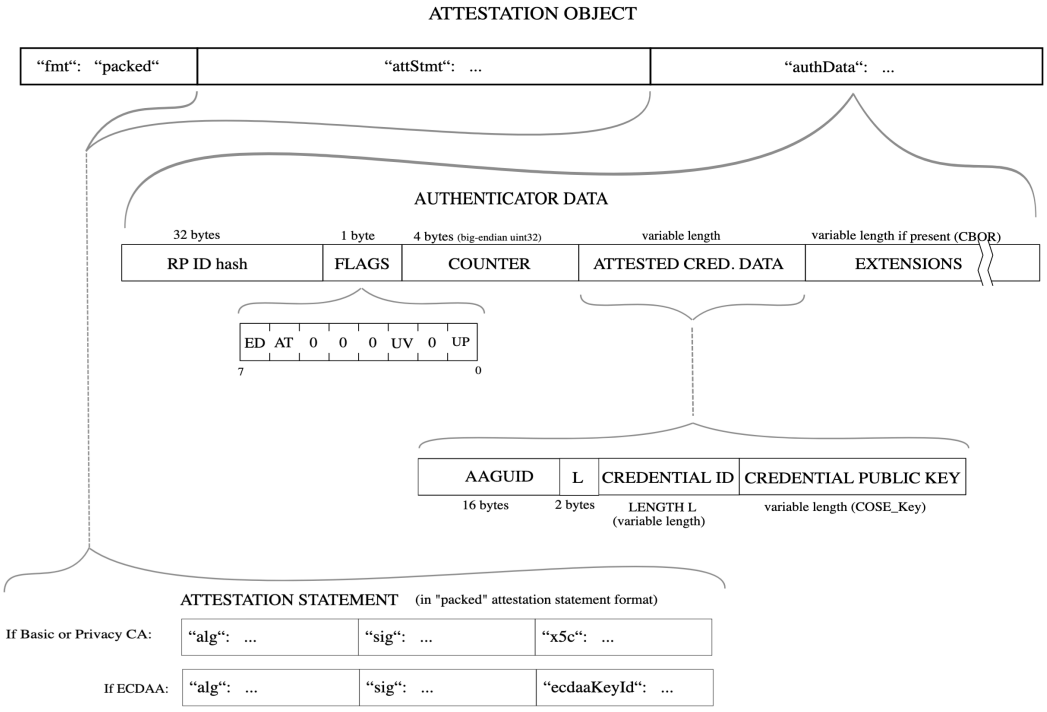

FIGURE 2.5: Attestation object layout [43].

The authenticator can optionally include attestation data to provide proof of its manufacturer depending on the relying party's preference. The whole data structure is then called the `attestationObject`. Figure 2.5 provides a nice overview of the `attestationObject`'s composition.

4. The authenticator returns the `attestationObject` to the client.
5. The client forwards the `attestationObject` and the `clientDataJSON` to the relying party. We mention this as the **assertion request**.
6. The relying party verifies the data according to the specification [45] and if validated, adds the credentials to its storage.

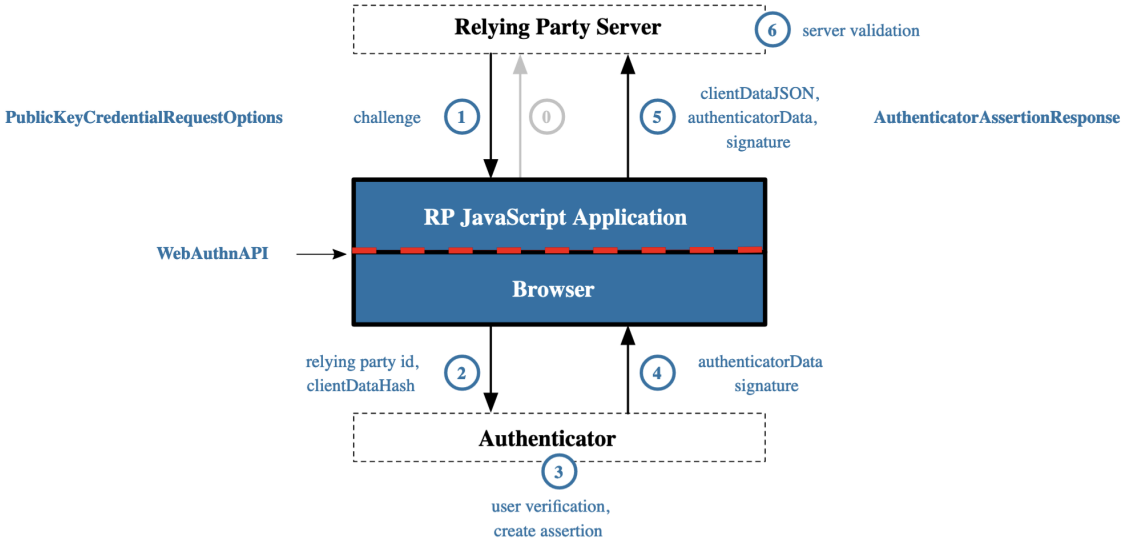

FIGURE 2.6: WebAuthn authentication flow [44].

2.2.2 Authentication

Figure 2.6 shows the authentication ceremony. When a user navigates to a website and proceeds to sign in with one of their authenticators, the authentication flow works as follows:

0. The user initiates the WebAuthn authentication flow. In this work, we refer to this request as the **setup request**.

1. The relying party calls the `navigator.credentials.get` method on the user's client with the argument a high-entropy random challenge. Optionally arguments can include the RP ID, the preference for user verification (i.e., "preferred", "enforced", or "discouraged"), the list of credential IDs registered by the user, etc. The full list of properties can be found in the Credential API documentation[4].

2. The client creates the `clientDataJSON` as in the registration, with the value of type set to "webauthn.create".

   The client sends the hash of the `clientDataJSON` and the RP ID to the authenticator through CTAP2.

3. The authenticator must verify that the user is in its proximity (by BLE, NFC, or tapping a button) and optionally verify that the user does a gesture (PIN, biometric, etc.) to unlock it.

   The authenticator creates a CBOR-encoded `authenticatorData` object, which includes:

   - The hash of the RP ID.
   - The flags.
   - The signature counter.

[4] https://developer.mozilla.org/en-US/docs/Web/API/CredentialsContainer/get#publickey_object_structure

- Optional extensions.

The authenticator then uses the scoped private key and signs the `authenticatorData` concatenated with the hash of the `clientDataJSON`.

4. The authenticator sends the signature and the `authenticatorData` to the client.
5. The client appends the `clientDataJSON` to the `authenticatorData` and forwards them to the server. We mention this as the **assertion request**.
6. The RP verifies the data according to the specification [46] and if validated, lets the user sign in.

In this work, we are curious to see how relying parties validate the data (step 6). In case we manipulate the data in the assertion requests, we want to see if the server accepts or rejects the modified data. In the former case, we consider this mis-check a server misconfiguration. We explain more about how we operate to test for these comprehensively in Chapter 3.

2.3 Security

In this section, we explain what considerations FIDO and W3C make regarding security and privacy.

2.3.1 WebAuthn Security considerations

Section 13 of the WebAuthn standard [44] outlines WebAuthn-specific security considerations. Here, we list the most relevant ones to our work. In particular, we focus on security considerations for relying parties in Section 13.4 and we comment how these can influence the security of the protocol. We also include Section 13.4.8 and Section 13.4.9 from the WebAuthn draft standard [47], as we consider these points relevant to the security of the protocol.

Section 13.2 says that "it is generally assumed that roaming authenticators are physically close to, and communicate directly with, the client."

In Section 13.4.3, the specification mentions that the relying parties SHOULD store the challenge temporarily until a WebAuthn operation is complete. It does not specify how this can cause security issues.

Section 13.4.4 outlines the importance of TLS-protected connections, in order to prevent MitM attacks that could forge the attestation object and change the public key to one that is controlled by the attacker.

Section 13.4.7 describes how relying parties can leak the privacy of users about whether they have registered an authenticator in the service. An attacker can leverage these relying parties to execute username enumeration.

Section 13.4.8 talks about injection attacks and says that relying parties allow the execution of untrusted code in the origin of the credential that it creates.

Section 13.4.9 says that relying parties MUST validate the origin value in the client data. Later, they say that even though the credentials are already scoped, this validation serves as an additional layer of protection in case of a faulty authenticator.

2.3.2 CVEs

In this section, we list vulnerabilities disclosed by the National Institute of Standards and Technology (NIST) that are related to the implementation of FIDO2/WebAuthn.

NIST published several security vulnerabilities regarding FIDO2/WebAuthn. **CVE-2021-38299** [31] exposes a critical vulnerability in the WebAuthn framework version $< 3.3.4$. Hackenjos discovered that the PHP library webauthn-framework did not verify the user presence bit in WebAuthn assertions. A remote attacker can potentially log into a service without passing the user presence test on the victim's authenticator.

CVE-2022-42731 [32] shows a vulnerability in the Django-mfa2 library. This allows a replay attack, which can register another authenticator to a user by replaying a request since the challenge is not invalidated after its first use.

2.4 Web Security

In this section, we give an overview of relevant web application security concepts. We introduce Burp Suite, a modern web security testing tool, and its functionalities, such as extensions. We conclude with the description of a well-known class of client-side vulnerabilities, Cross-Site Request Forgery (CSRF).

2.4.1 Burp Suite

Burp Suite [33] is a "comprehensive suite of tools for web application security testing" [35], developed by PortSwigger. It allows for intercepting HTTP(S) traffic and modifying requests and responses in order to test different application behaviors. Hereby, we describe and summarize the relevant components to our work:

- **Proxy**: The Proxy sits between the client and the server, allowing Burp Suite to capture HTTP(S) requests and responses between the two parties. It decrypts the TLS-protected request content [36] and shows it in Burp Suite.
- **Repeater**: The Repeater allows users to manually manipulate individual HTTP(S) requests and test how the application responds to different kinds of inputs.
- **Extensions**: The Extension tab manages the extensions active in the current Burp session. It permits the extensibility and automation of custom tasks.

Burp Suite is one of the most common web application testing tools, thanks to its versatility. As mentioned before, Burp Suite allows users to extend and customize its functionalities by means of extensions. Any user may personalize Burp Suite's behaviors, such as modifying HTTP requests and responses, customizing its interface, adding extra checks, etc., by developing a custom plugin. Burp Suite provides a comprehensive API that exposes various functions, interfaces, and data structures: Montoya API [34]. This allows developers to manipulate the tool's features.

WebAuthn CBOR Decoder

To date, the only extension available in the BApp Store regarding FIDO2 is a WebAuthn CBOR Decoder[5]. Its purpose is to decode the `attestationObject` from the

[5]WebAuthn CBOR Decoder: https://portswigger.net/bappstore/36b86e134e93444596cc2675a4f0e48d

CBOR format of WebAuthn requests and visualize it in a user-friendly way. This is extremely useful for debugging, but its utility is limited due to its simplicity as it simply loops through the request parameters and looks for the literal string "attestationObject". The extension cannot detect and decode more complicated request parameters like `data=attestationObject:some_attestation_value` because its key is not "attestationObject". Moreover, it is not able to decode the `authenticatorData` parameter. In Section 3.1.1, we show how we extend this decoder to decode more data structures and formats.

2.4.2 Cross Site Request Forgery

Cross-Site Request Forgery (CSRF) is an attack that makes a victim do unintended actions through social engineering. According to PortSwigger [42], three conditions must be satisfied to execute a CSRF attack:

- The action to be computed is relevant, such as changing user-specific data or modifying permissions for other users.
- The session relies solely on session cookies.
- The request parameters must not contain any unpredictable values.

An example of an HTTP request to register an authenticator would be:

```
POST /authenticator/add HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 39
Cookie: session=yvthwsztyeQkAPzeQ5gHgTvlyxHfsAfE

authenticator=attackers_authenticator
```

If a relying party is vulnerable to CSRF, an attacker can construct a webpage that once accessed by a victim, triggers automatically an HTTP request on the latter's behalf. The victim will perform the relevant action with their session cookie(s) being automatically included. For instance, this can lead to the attacker's public key being added to the victim's account. The attacker can then log in to the victim's account with their own private key and take it over.

Common mitigations against Cross-Site Request Forgery are:

- CSRF tokens, which are unique and unpredictable values generated by the server. For each relevant action, it is important to include this token in the request parameters to be considered. An attacker is not able to guess this value and therefore construct a valid request.
- the `SameSite` attribute on cookies, which prevents them from being sent if the request is triggered from a different site than the target one. This does not prevent attacks between different subdomains of the same domain. There are three possible values for the `SameSite` attribute: `Strict`, `Lax`, and `None`. `Strict` allows the cookie(s) to be sent only under the same site. `Lax` allows the cookie sent only in GET requests and from a top-level navigation by the user, e.g., clicking on a link. There are advanced methods to send the cookie with POST requests and we make sure to test them during experiments. `None` allows the cookie to be sent and thus the website might be vulnerable to CSRF.

2.5 Related Work

In this section, we outline the related work regarding the security of FIDO2/WebAuthn. We remark on the difference between the present literature and our work by highlighting the relevance of our contribution. In particular, we summarize previous theoretical formal verification of the FIDO2 and the WebAuthn protocols. Then, we look at the existing attacks against FIDO2. In particular, client-side attacks mainly tamper with the underlying protocol and its security assumptions. Also, we mention previous authenticator attacks which are able to extract private information by exploiting hardware weaknesses. We pay attention to testing server-side misconfigurations, in which only little work has been done. We do not focus on the protocol itself, but rather on how it is implemented.

2.5.1 Formal verification

Previous work has shown that FIDO2 and its sub-protocol, WebAuthn, are theoretically secure. In our work, we do not provide security considerations about the protocol, rather we focus on the practical implementation of it. We study the previous work in order to find potential theoretical vulnerabilities that we can exploit in practice. The following is a summary of the most relevant formal verifications of the FIDO2/WebAuthn protocols.

Firstly, Barbosa et al. [5] introduce a formal model for authentication for FIDO2, based on the first version of WebAuthn and CTAP2 and confirm the authentication security of WebAuthn.

Bindel et al. [6] build on [5] and provide a more fine-grained security model and prove other security guarantees, such as token binding agreement, the None attestation mode, and user presence/verification checks. As we will explain later in Section 2, these properties are significant for the secure workflow of FIDO2. The authors prove that FIDO2 is secure in their security model.

Hanzlik et al. [21] are the first to formalize the privacy notion that FIDO wishes to achieve. The authors extend Barbosa et al.'s work by verifying security for security keys adopting key derivation and key wrapping. Furthermore, Hanzlik et al. do not consider authenticator attestation in their model because it is impractical [6] and it could potentially violate privacy [19]. They show that existing implementations of FIDO2 satisfy the notion of security against impersonation also in the case of externally stored keys.

Guirat and Halpin [19] conduct the first formal verification of the security and privacy of solely the WebAuthn protocol. Their security goals are to prove integrity, i.e., no secret key material can be known by the attacker, and authentication, i.e., the user can only authenticate to the server they have registered to. Their results show that no phishing and man-in-the-middle attacks were found. Thus, WebAuthn itself securely authenticates its users according to their test cases. They also wish to verify the privacy properties by analyzing whether authentication is unlinkable between origins. They propose a setting where only a user and a server are present. If the server can link any information that the client sends, then the privacy property cannot be verified. Their result is that, if a user uses one authenticator, then its public attestation key will always be the same, so the server can check that and link it with previous registrations. We argue that this conclusion assumes that attestation keys are unique to each authenticator, but in practice, multiple authenticators share the same AAGUID (batch attestation) [6].

2.5.2 Client-side attacks

Previous work shows what attacks can be conducted if the client is compromised.

Yadav et al. [51] mention in their work that 115,881 Chrome extensions use *activeTab/tabs* permission, which can override the WebAuthn API. If one of these extensions is compromised, an attacker can launch an attack impacting millions of users. The authors also mention that relying parties usually do not provide specific error messages, leaving the users unaware if actions are their own or spoofed.

Similarly, Kuchhal [28] also implements a proof of concept for an attack using a malicious browser extension. In this case, the authors manage to create a malicious request immediately after the legitimate one, so that the user re-authenticates with a gesture as they are unconscious of the second being a malicious request. The attacker can leverage the Risk-Based Authentication (RBA) systems, which lack transparency and do not specify what exact action is the user authenticating to.

Schrempp [39] performs a formal verification on FIDO2 and she also proposes two attacks that enable an attacker to make the victim's authenticator sign a challenge from the relying party. She also uses a similar technique to the previous ones.

We agree with these authors that client compromise is still a practical and realistic scenario, although FIDO Security Reference [15] assumes that it is trustworthy and not tampered with.

2.5.3 Authenticator Security

Furthermore, previous work shows attacks against the security of authenticators. NinjaLab conducted a side channel attack to reveal the ECDSA ephemeral key [37]. Even if this attack is impractical for an average user, due to its high economic cost and complexity, it still shows that this threat needs to be considered in the security models for formal verification. A CVE has also been assigned to this vulnerability [30].

Roth et al. [38] demonstrate the cloning attack on the Nordic nRF52832 on Apple AirTags with a low-cost setup (5€), showing that hardware cloning is nevertheless a feasible threat to be considered. Anyway, we mention that there are techniques to mitigate authenticator tampering, such as [10].

2.5.4 Server-side misconfigurations

There are not many tools analyzing comprehensively the security of the server-side implementations of FIDO2/WebAuthn. The Fido Alliance offers the "Conformance Self-Validation Testing" tools [4] which help developers test the conformance of their FIDO2/WebAuthn implementation with the specifications during the development. In our work, instead, we create a tool that security professionals can use to test the FIDO2/WebAuthn implementation externally. We would like this tool to be used in penetration testing assessments and be able to support black box testing, i.e., without knowing how FIDO2/WebAuthn is implemented. Grammatopoulos et al. have previously implemented WebDevAuthn, a tool for "testing FIDO2/WebAuthn implementation's conformance" [18]. WebDevAuthn hijacks the communication between the client and the relying party and automatically displays warnings when misconfigurations are detected. Our work takes inspiration from theirs, but we aim to improve the usability of the tool, consider more comprehensive test cases, and provide proof of concepts of feasible attacks. We comment that our implementation can be integrated into Burp Suite, which is a commonly used software in practical security assessments.

Chapter 3

Methodology

In this chapter, we present the methodology to provide answers to our research questions.

To ease and automate the testing process for FIDO2/WebAuthn misconfigurations in web applications, we implement a Burp Suite extension that can analyze server-side implementations automatically. We describe our approach to implement it and outline the design choices and the core functionalities behind this scanner.

To visualize what are the design choices of current implementations of FIDO2/WebAuthn, we manually compute registration and authentication operations to see what and how information is exchanged between client and relying parties. We analyze what makes a WebAuthn request valid and how it is handles on the server side. To assess their compliance with the security best practices, we check with the scanner if the relying parties follow the official specifications and have the proper security configurations in place. To expand on this, we further research whether it is possible to conduct client-side attack, such as Cross-Site Request Forgery, and compromise other user accounts.

After finding potential WebAuthn misconfigurations, we study if possible attacks could arise. We combine the results of the automated scanning, understand previously disclosed vulnerabilities, and manually exploit the misconfigurations by trying out several attacks such as CSRF or JavaScript manipulation.

This chapter is structured as follows. We begin outlining our approach to making an automatic scanner with Burp Suite. We describe in detail our whole setup, composed of the Burp Suite extension, the CBOR Decoder, and the virtual authenticators. We proceed to describe what tests we plan to do and on what services we execute these in order to find server-side misconfigurations. We conclude by explaining how Cross-Site Request Forgery attacks can in theory be executed against the FIDO2/WebAuthn ceremony flows, in case of improper server-side session management.

3.1 Security Configurations

Firstly, we present our setup environment to conduct our tests. We list the components that help us during the testing of the configurations, namely the Burp Suite CBOR Decoder, our novel Burp Suite extension, and the virtual authenticators that we use. Secondly, we list the test cases we are executing and what impact they have. We define a test case as a value that we modify in the data objects sent to the server to see the latter's behavior. We list also the targeted relying parties. We mention that we only select websites on which testing is allowed. These are either demo websites or the ones that have a public responsible disclosure platform so that we can submit our findings to the vendor.

We proceed to describe our approach to building our Burp Suite extension and how

to do automated testing. We give two complementary approaches: the first one facilitates manual work in case of specific test cases and the second gives a comprehensive testing result of all the test cases. We recall that we aim to build a Burp Suite extension that can be used during security assessments. In these cases, the tester can use the second approach to get an initial overview of all the misconfigurations. After a patch of a misconfiguration, the tester can then use the first approach to test that single case, without having to check all of them once again.

3.1.1 Setup

We introduce our setup architecture to conduct automated scanning of FIDO2/WebAuthn server-side misconfigurations. We aim to implement a Burp Suite Extension that can modify WebAuthn parameters in the requests. In particular, we target the data structures `clientDataJSON`, `attestationObject`, and `authenticatorData`. Briefly, our goal is to see whether the server would reject an *invalid* request. In case it does not, we report this as a security issue in Burp Suite. We wish to create a tool that can be used during security assessments in penetration testing engagements.

Our setup is composed of five integral components, namely Burp Suite, a client, a (virtual) authenticator, an IDE, and a relying party.

Burp Suite's CBOR Decoder extension

Examining existing Burp Suite extensions that could facilitate our work, we consider the WebAuthn CBOR decoder [1]. This tool decodes CBOR formatted objects and renders them in a human-readable form. Currently, this extension decodes only the `attestationObject` parameter by simply checking whether a parameter name in the requests matches the literal string "attestationObject". Its simplicity makes it not fully compatible with all requests' content types and formats. Our contribution to improving the performance of this tool is adding more content types and formats to detect CBOR-formatted values. In detail, we include the `application/json`, the `application/x-www-form-urlencoded`, and the `multipart/form-data` content types by creating a function that looks thoroughly in the request for every possible CBOR-encoded value. Moreover, we add the detection and display of the `authenticatorData` object. With this, we can read more CBOR-encoded values and better debug WebAuthn requests during our testing.

Burp Suite Extension

The core of this work is creating a Burp Suite extension and we design to be application in penetration testing engagements. To program it, we use the Montoya API [34] from PortSwigger. Its versatility allows developers to easily customize the behavior of Burp Suite tailored to their needs. With the Montoya API, we create a custom HTTP handler and a context menu item in Burp Suite. In essence, the former is a class that allows us to implement rules to handle the intercepted HTTP requests. The latter is a Java Swing menu item, in which we can insert the option that, when clicked, triggers specific actions (e.g., "Test registration" to start the testing for the registration ceremony). More details will follow as we explain our approach to test WebAuthn misconfigurations.

[1] https://portswigger.net/bappstore/36b86e134e93444596cc2675a4f0e48d

Virtual authenticators

In the testing phase, we opt for virtual authenticators to create and store our FIDO2/WebAuthn credentials We use two types of virtual authenticators:

- **Chrome DevTools virtual authenticator** [13]. This is revealed to be the most useful tool since it offers the functionality to export private keys. These are needed to re-compute the signature when modifying values in an authentication request. We mention that exporting the private key is non-conform with the WebAuthn standard as it "is expected to never be exposed to any other party, not even to the owner of the authenticator" [44].

  Its drawback is that not all relying parties recognize this as a valid FIDO2 authenticator.

- **1Password passkeys**. The 1Password password manager is able to create, store, and synchronize passkeys among different devices. It is extremely useful because it can create a valid passkey for every relying party we are testing.

3.1.2 Test cases

In this section, we present the comprehensive tests we conduct in our analyses. A test consists of changing a specific value to one of the data structures mentioned in Section 2.2 in the request that we send to the server during WebAuthn ceremonies. To create our test set, we follow the list of procedures the relying party should or must follow to correctly handle the ceremonies defined in Chapter 7 of the standard [44]. A misconfiguration arises when the relying party incorrectly handles a test. This happens when a server has not rejected an invalid value that we provided. We refer to Chapter 13 of the standard [44] and previous CVEs to explain what potential attack surface is exposed by misconfigurations.

Following are the data objects and the values we are modifying.

Client data

We modify the `clientDataJSON` object sent from the client to the relying party. It contains client-related data rather than authenticator-specific information. To correctly modify this object, we undertake the following steps: first, we Base64-decode the `clientDataJSON`. Subsequently, we change its value according to the specific test case. After that, we re-encode the data with Base64 and update the data in the request. We perform the following tests:

- **Client data type**: we modify the value associated with key `"type"` in the `clientDataJSON` object. The relying party must make sure that this value is `"webauthn.create"` for registration and `"webauthn.get"` for authentication. We comment that this mis-check does not imply a security vulnerability.

- **Client data challenge**: we modify the value associated with key `"challenge"` in the `clientDataJSON` object. The relying party must make sure that this value is the original challenge value that it has created for this user for this session.

- **Client data origin**: we modify the value associated with key `"origin"` in the `clientDataJSON` object. The relying party must make sure that the received origin value is in a whitelist of acceptable origins. This serves as an additional layer of protection against phishing attacks. If the "origin" is not checked correctly, in case

an attacker has obtained a credential (i.e., a valid assertion request) from a victim because of a faulty authenticator, they can use that to log onto the legitimate website on behalf of the victim.

To test if the relying party correctly checks the `"origin"` value, we perform two types of test:

- **Client data origin**: we modify the `"origin"` value with, e.g., "Test Extension".
- **Client data origin subdomain**: we modify the `"origin"` value with a subdomain value, e.g., "test-extension.domain.com".

We comment that in actual penetration testing engagements, it is more comprehensive to test for instance subdomains with a dictionary of the most common ones.

Authenticator data

We modify the values in the `authenticatorData` object to assess the abilities of relying parties to detect anomalies. We undertake the following steps: first, we URL-Base64 decode the `authenticatorData` to retrieve the CBOR-encoded byte array. From there, we modify the data at a low level using bitwise operations or overwriting bytes. Subsequently, we URL-Base64 encode the modified CBOR array to get the new `authenticatorData` and update it in the request. We perform the following tests following the syntax in Figure 2.5:

- **User Presence**: we set the `UserPresence` flag to 0 and `UserVerification` flag to 1. We comment that even if this case is not realistic as remote authenticators that require user verification do not exist, we still test for this for completeness.
- **User Verification**: we set the `UserVerification` flag to 0 and `UserPresence` flag to 1. Should a relying party allow single-factor authentication, they must check that the `UserVerification` bit is set, which means that the user has confirmed the ownership of the authenticator. If relying parties omit checking the `UserVerification` flag, this would be a serious vulnerability, as an attacker can log in to a victim's account by only obtaining physically their security key.
- **User Presence and Verification**: we set the `UserPresence` and `UserVerification` flags to 0. We comment that even if this case is not realistic as remote authenticators do not exist, we still test for this for completeness.
- **RP ID hash**: in *registration*, we change the `RP ID HASH` value to invalid hash bytes, e.g., to the padded byte-representation of "Test Extension". This is to check if the relying party accepts an authenticator that tells that it is registering with another relying party. We comment that this mis-check does not imply a security vulnerability, but we consider this case for completeness.
- **Public key ID**: in *registration*, we change the `CREDENTIAL PUBLIC KEY` value to an identifier of a random public key algorithm. We want to see if the server checks that we have inputted an algorithm that it has not allowed. We comment that this mis-check does not imply a security vulnerability, but we consider this case for completeness.

- **Credential ID**: in *authentication*, we change the `CREDENTIAL ID` value to an invalid one, e.g., the padded byte-representation of "Test Extension". This is to check if the relying party accepts an authenticator ID that has not been stored in previous registrations. We comment that this mis-check does not imply a security vulnerability, but we consider this case for completeness.
- **Signature counter 1**: in *authentication*, we change the `COUNTER` value to 1 to see if the relying party accepts a low signature count. We first ensure that the relying party stores a signature value greater than 1 by manually signing in a few times. The consequence of this mis-check is that an attacker with a cloned authenticator can authenticate with that on behalf of the victim.
- **Signature counter 999**: in *authentication*, we change the `COUNTER` to 999 to see if the relying party accepts a significantly greater signature count. With this, we want to test if, in the case of a cloned authenticator, the relying party detects if the value is significantly larger than the stored signature counter. If not, an attacker with a cloned authenticator can authenticate with that on behalf of the victim by setting a very high value in the signature counter.

Signature

During authentication, we change the `signature` to random values to see if the relying party checks it. A failure in signature validation could have critical consequences, as an attacker can gain unauthorized access to another account without needing to prove the ownership of any private key.

3.1.3 Test pages

Here we present the test pages that we target, categorized into two types of relying parties: demo websites and commercial websites. Our testing methodology slightly differs between these due to their inherent differences.

Demo websites are WebAuthn playgrounds, in which any user can only register an authenticator and authenticate within. Their purpose is to let users familiarize themselves with password-less authentication and for developers to make sure that the FIDO2 library is implemented and deployed correctly. These websites are easier to test because session management is simpler or non-existent, the HTTP requests are easily readable and modifiable, and there are no defense mechanisms such as rate limiting. Additionally, many of these specify which open-source WebAuthn libraries and frameworks they use. This allows us to also conduct white-box testing, with which we can refine our tests tailored to the application.

Commercial web services present a different set of challenges. To compute WebAuthn ceremonies, we need to have an account within the service and a valid session. Unlike demo websites, the HTTP requests to compute a ceremony are not straightforward to grasp, causing more work in the automation of the testing. Commercial services often have the obstacle of not being open-source as the source code remains the intellectual property of the commercial party. In case we find open-source resources of these, we perform also white box testing by manually auditing code, in order to find possible vulnerabilities.

In order to protect the personally identifiable information of other users, we create dedicated test accounts and act upon these. To ensure ethical testing, We firstly verify that all the target websites have responsible disclosure programs or bug bounties. We

strictly follow these programs and disclose promptly any security findings discovered in this work. More details are in the Chapter 7.

We consider two community-trusted lists of websites that support WebAuthn in their authentication methods: 1Password passkeys list [2] and the YubiKey Catalog [3]. We prioritize relying parties that allow password-less authentication rather than leveraging FIDO2 as a second factor. We exclude services where it is not suitable to do analyses. These include the ones requiring real personal information (Social Security Number, phone number), with non-English language, malfunctioning, requiring to pay to register, etc. We further exclude the ones that do not strictly follow the WebAuthn standard [44]. For instance, we do not consider relying parties that use the standard Base64 encoding (instead of the URL safe mode), as this requires significant effort in crafting valid requests in our tool. From the Yubikey catalog, we sort websites by popularity and select the most well-known ones.

Vendors that have not yet disclosed our findings are anonymized as "RPx" and we refrain from providing any specific information about them to comply with their disclosure policies. We select the following relying parties to test on:

- `webauthn.io`: a demo website that uses the Python library of WebAuthn.
- `webauthn.cedarcode.com`: a demo website that uses the Ruby library of WebAuthn.
- RP3 This is an online co-working space where developers can write code and do software version control.
- RP4 This is a company that provides electronic signing services.
- RP5 This is a widely-used online travel agency used by people to compare prices and make reservations for flights, hotels, and other travel-related services.

We target these as we want to show how popular sites with thousands of users adopt this password-less technology. We believe that in case of a security finding, its disclosure would significantly and safeguard the security of the respective relying parties and their user bases.

3.1.4 General approach

In this section, we describe our approaches and design choices behind the implementation of our Burp Suite extension.

The WebAuthn standard [44] defines the necessary data to compute a WebAuthn ceremony, but it does not specify how these should be transmitted from the client to the relying party's back end. Relying parties have the freedom to choose their preferred way to parse the results of the WebAuthn methods and to format them. For instance, the requests' content type can vary between JSON (`Content-Type: application/json`), URL parameters (`Content-Type: application/x-www-form-urlencoded`), multipart form data (`multipart/form-data`), etc. Therefore, it is challenging to build a tool that generates valid requests out of the blue without knowing in advance what data and in which format the relying party is expecting.

We assume these discrepancies are because commercial services tend to implement their own version of WebAuthn. By comparing request formats from open-source WebAuthn

[2] `https://passkeys.directory/`
[3] `https://www.yubico.com/works-with-yubikey/catalog/?protocol=5`

libraries and commercial websites, we often do not notice similarities among them. Therefore, we infer that commercial services do not adopt the publicly available libraries, but opt for an ad-hoc implementation of WebAuthn tailored to their existing codebase.

Notwithstanding, our general testing approach is understanding what makes requests valid, i.e., which format and which encoding they employ, and applying our test cases accordingly. A test case is a value that we modify in one of the data objects described in Section 2.2, such as `attestationObject`, `authenticatorData`, or `clientDataJSON`. Our objective is to see whether the relying party accepts our modified (invalid) value. In that case, we raise an alert in the Burp Suite's issues panel. We proceed to design our testing process following two approaches:

1. For each test case, we compute manually a WebAuthn operation and make our tool apply the test case on the fly. This approach would work with every relying party and it is less prone to errors as we are essentially executing an ordinary WebAuthn operation. For instance, when computing a registration ceremony manually, we are sure that values such as session cookies, CSRF tokens, and the created public key are valid since they have been created with an ordinary flow. However, this approach requires significant manual effort from the tester. Indeed, for each test case, they must manually select it in the source code, load the Burp Suite extension, and perform manually a ceremony. The main advantage of this approach is that the tester can select which single case it wants to test, without completing a whole comprehensive assessment.

2. We manually compute the WebAuthn ceremonies only once. From Burp Suite Proxy's history, we look for the setup and assertion requests that compose the ceremony and we modify and replay them according to the test case. This approach requires great effort to understand which parameters need to be updated for the requests to be considered valid by the relying party. For instance, a session cookie might be invalidated after the initial ceremony and needs updating (e.g., by logging in again). Despite an initial important manual effort, this approach simplifies significantly the effort of testing a website as it automatically scans through our test cases.

These two approaches complement each other. In scenarios where a developer wishes to test their own WebAuthn implementation, they can execute the second approach to have a comprehensive review of all the misconfigurations and test cases. Once these are patched, the developer can then manually test the individual cases using the first approach.

During authentication, after we change the `clientDataJSON` or the `authenticatorData` objects, it is necessary to recompute a valid signature. Leveraging the DevTools virtual authenticator, we can export the private key associated with the credential and recompute the signature with that. Unfortunately, with other authenticators, we cannot extract the private keys as these are not exportable. For this reason, the relying parties that do not accept the DevTools virtual authenticator cannot be tested for the authentication ceremony (unless they skip the signature check).

After sending the modified requests to the relying party, we need to understand whether the latter has considered them valid and accepted them. To do this, we need to define a way to classify valid ceremonies by looking at the HTTP response of the assertion requests. During the manual steps in our approaches, we observer how relying parties distinguish between valid or invalid operations. For instance, a relying party can return an HTTP status code of 201 to indicate that a new authenticator has been added to an account.

As the standard does not specify what responses should relying parties send, we observe in practise that different HTTP responses are given when a ceremony succeeds or fails. For

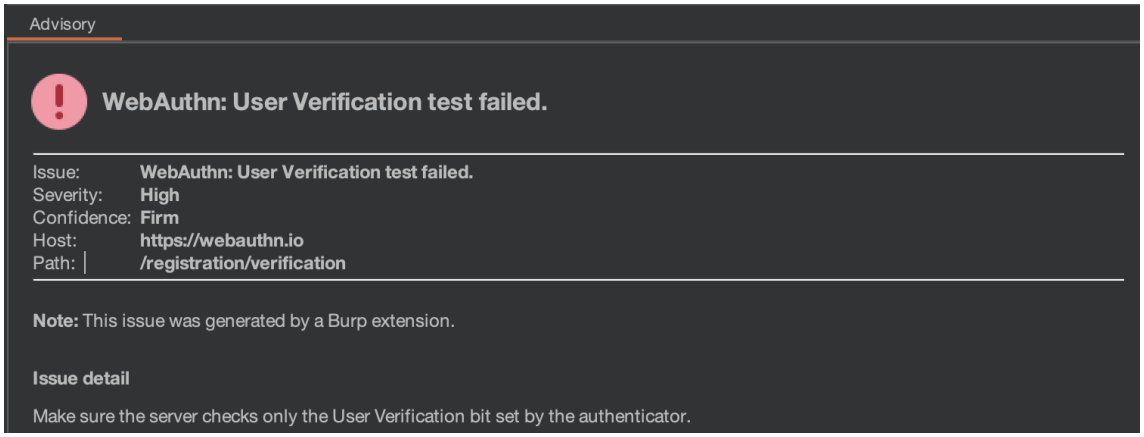

FIGURE 3.1: Issue raised in Burp Suite about the `userVerification` flag not being checked by `webauthn.io`.

example, some applications to return a successful status code of 200 when an operation fails or a status code of 301 when it succeeds. We develop a generic function that, given an HTTP response, can classify it as valid or not. This function examines the response for peculiar characteristics, such as particular failure literal strings or status codes.

Finally, after scanning a targeted relying party, we check each response manually to make sure that it is correctly classified. Once a security issue is detected, i.e., after the relying party has accepted our invalid value, we display it in Burp Suite's issues panel, as shown in Figure 3.1

Following is a detailed explanation of the two approaches.

Approach 1

With this approach, we design the Burp Suite extension as follows. We create methods that look thoroughly for the data objects targeted by our test cases. For instance, if we want to modify the `attestationObject`, we create a method that given a generic request, retrieves that from the request's parameters. In addition, we create a setter method that, after modifying the data object, sets it back into their previous location in the request's parameters.

After loading the tool into Burp Suite, we manually navigate to the target website and initiate a new WebAuthn ceremony. The extension can detect the data objects of interest in the request, change the value according to the test case, set it back, and forward the modified request. With this approach, we can also visualize the result in our client. For instance, Figure 3.2 and Figure 3.3 show the response of `webauthn.io` when modifying the `UserPresence` bit during registration and authentication ceremonies.

Approach 2

With this approach, we begin by navigating to the target website and manually compute a new WebAuthn ceremony. Firstly, we need to understand which requests compose a full valid WebAuthn ceremony flow, i.e., which WebAuthn requests registers our authenticator or authenticates us. Our goal is to repeat these requests and, each time, change their content according to our test cases. Specifically,

1. We take the *assertion request*, i.e., the one containing the `attestationObject` or the `authenticatorData`, and replay it with Burp Suite Repeater. We examine if

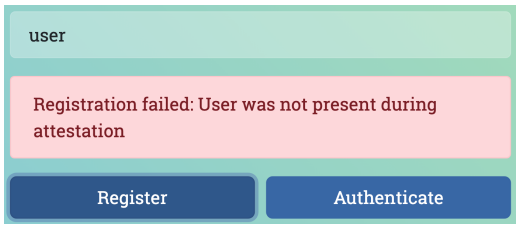

FIGURE 3.2: `webauthn.io` registration error due to missing `userPresence` flag.

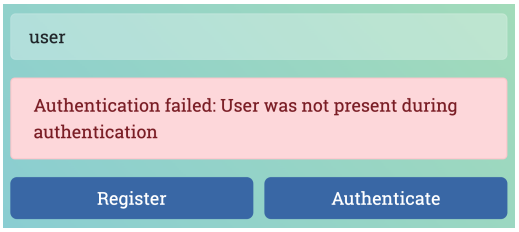

FIGURE 3.3: `webauthn.io` authentication error due to missing `userPresence` flag.

the relying party still considers this request as valid. For instance, for the registration ceremony, we test whether simply replaying the request containing the `attestationObject` would result in registering a new authenticator. If so, it means that the relying party does not invalidate the challenge or other parameters after the initial ceremony, and we consider this a valid request. We manually test this case to also verify if CVE-2022-42731 is still present in some relying parties.

If the repeated request fails, we save this request and proceed with the next step.

2. We take the *setup request* and replay it with Burp Suite's Repeater. We see whether the relying party accepts it by seeing if it responds with a new challenge value. If so, we retrieve the saved *assertion request* in step 1 and replay it with the updated challenge value in the `clientDataJSON` object. We eventually need to update manually other nonces such as the CSRF token or cookies if the request fails. If replaying the *setup request* and the new *assertion request* result in successfully registering a new authenticator during registration, we consider these requests as valid.

   If we are unable to retrieve a valid challenge by relaying the *setup request*, e.g., because we do not have a valid session anymore, we proceed with the third step.

3. At this point, we have to compute the normal authentication flow again to receive a valid fresh session cookie. For example, we can do this by submitting the email and password again to log in again. We then receive a valid session cookie from the server, with which we can do the operations in step 2.

In conclusion, we consider valid requests the ones that, replayed, will successfully complete a WebAuthn ceremony. We send these requests to our Burp Suite extension through our custom context menu item. The extension maintains a list of buffered valid requests and sends them one by one. After each request, the tool parses the response and updates the next buffered request with fresh values, such as challenges, CSRF tokens, cookies, or other nonces, to make it valid and accepted by the relying party.

The main difference from the first approach is that, during registration ceremonies, we are repeating the registration with the same public key. In the first approach, we are manually creating distinct public keys each time by manually calling the WebAuthn methods. In this second approach, if a relying party rejects the registration of an already registered key, we have to make our tool change the credential ID in the `attestationObject` to a random one.

For each test case, we send the modified valid requests according to the test case. This approach can be summarized with the following pseudo-code:

```
for test in tests do
    for request in valid requests do
        if request not assertion then                    ▷ setup request(s)
            send request
        else                                             ▷ assertion request
            test(request)
        end if
    end for
end for
```

3.2 Cross-Site Request Forgery

In this section, we present a novel attack we plan to test on our targeted relying parties - Cross-Site Request Forgery (CSRF). CSRF makes a victim perform unintended actions via a simple social engineering technique, i.e., inducing them to click an attacker-controlled link. In this particular scenario, the goal of the attacker is to illicitly register their public key on the victim's account. In this way, the attacker can log in to the victim's account with the former's corresponding private key. The goal of this attack is to achieve a full account takeover.

Our methodology involves emulating both the attacker and the victim to develop a proof of concept demonstrating the potential success of the CSRF attack. We highlight that this attack does not disrupt the ordinary functioning of the relying parties, compared to the scanning approach.

3.2.1 Test pages

Once again, we select our testbed from the two aforementioned lists: 1Password passkeys list [4] and the YubiKey Catalog [5]. We choose relying parties that allow password-less authentication instead of using FIDO2 as a second factor. As the information considered in this experiment is publicly available, we do not anonymize our results in this particular case.

We select the following relying parties:

- webauthn.io: a demo website that uses the Python library of WebAuthn.
- keycloak.org: a popular open-source identity and access management solution.
- RP3: an online co-working space where developers can write code and do software version control.
- webauthn.cedarcode.com: a demo website that uses the Ruby library of WebAuthn.
- bestbuy.com: a consumer electronics retailer and it is in the Fortune 500 ranking.
- authgear.com: a highly customizable authentication solution to help businesses better manage their users.
- app.hancock.ink: a signing service for documents, quotes, and proposals.

[4] https://passkeys.directory/
[5] https://www.yubico.com/works-with-yubikey/catalog/?protocol=5

- `hanko.io`: an open-source authentication and user management solution with a focus on moving the login beyond passwords.
- RP5: an online travel agency.
- `locker.io`: an online service to store passwords and sensitive data.
- `minter.io`: provides analytics for social network accounts and shows statistics about followers, posts, ads, etc.
- `nintendo.com`: a multinational video game company.
- RP4: a company that provides electronic signing services.

3.2.2 Approach

Firstly, we need to find the session cookie(s) and check the `SameSite` attribute of these on the target relying party. We sign into a service to receive all the cookies. Then, with Burp Suite's Repeater, we manually repeat a request dropping one cookie at a time and observe if it is accepted. If not, it means that we are discarding a session cookie and we take note of its name and `SameSite` attribute.

Moving forward, we see if the WebAuthn ceremonies requests include a CSRF token. If we find a potentially vulnerable website, i.e., that has `SameSite` attributes set to `None` and does not have a CSRF token, we create two accounts to emulate an attacker and a victim. In the case of demo websites, we simply initiate two different sessions.

Firstly, we log in as the attacker and we initiate a WebAuthn **registration** ceremony. As explained in 2.2, this is generally composed of a setup request and an assertion request. We forward the setup request to retrieve the challenge from the server. Then, we intercept and drop the outgoing assertion request containing the attacker's public key in Burp Suite. In this way, we hope that the created challenge remains valid for further assertion requests, even from other users. Indeed, we hope that the challenge is not associated with the attacker's session so that the victim can complete the registration ceremony with their own cookie (or any other session identifier). We mention that the WebAuthn specification [44] does not specify defensive measures against this. In particular, Section 13.4.3 of the specification about the cryptographic challenges mentions only that these must be randomly generated, and not being tied to the user session.

At this point, we craft a phishing page that, when clicked, sends an assertion request with the previously saved challenge and the attacker's public key. We can easily set up this malicious page with Burp Suite's CSRF proof of concept generator.

After this, we log in as the victim and click on this phishing page. The assertion request is now sent on behalf of the victim. If the application accepts this request, the victim has now registered the attacker's authenticator in their account. In this way, this results in an account takeover and a severe violation of the confidentiality of other users. This method provides valuable insights into potential security vulnerabilities within the WebAuthn ceremonies' session management, emphasizing the need for robust protective measures against such attacks like CSRF.

Chapter 4

Results

The previous chapter answers **RQ1** by explaining our approach to making a tool that can ease the automated testing process for WebAuthn misconfigurations. The focus is on minimizing manual labor to enhance efficiency. We have seen the components to create a Burp Suite extension that can assess for server-side misconfigurations.

This chapter now delves into the application of our methodology and the presentation of the results. To answer **RQ2**, we also provide insights on the design choices of the FIDO2/WebAuthn implementations of each tested relying party. For each identified misconfiguration, we analyze its impact and propose a proof of concept to exploit the potential vulnerability. Furthermore, we show the results of our Cross-Site Request Forgery (CSRF) attacks and see if these are successful. To execute the attacks, we need to learn how relying parties do session management and what are their mitigations against CSRF. For this, we look at the `SameSite` attribute of the session cookies and the presence of an eventual CSRF token. Then, we aim to see if executing the described attack in Section 3.2 would lead to an account takeover. Furthermore, we explain the reasons why the attack succeeds or fails.

Based on the results of the found misconfigurations and the outcomes of CSRF attacks, we respond to **RQ3**. This comprehensive examination provides an understanding of the security posture of the assessed systems.

4.1 Misconfigurations

Implementing a tool capable of automatically testing for WebAuthn misconfigurations across arbitrary relying parties is a challenging task. This is due to relying parties behaving significantly differently compared to each other and we have to extend our tool to each case. For instance, an application may assign a new session cookie after each request. In this case, we have to create a specific function that updates this value repetitively. Due to this customization being significantly time-consuming, we are not able to deliver a large number of results and make comments about the general misconfigurations of FIDO2/WebAuthn.

Despite these challenges, our applied methodology yields insightful results for the five targeted relying parties, gaining interesting insights. To automatically scan for all the test cases, we use the second approach described in our methodology.

Table 4.1 and Table 4.2 show the results for the registration and authentication tests, respectively. To adhere to the responsible disclosure, we promptly disclose the findings to the respective relying parties. More details about the reporting are in Chapter 7. We refer to the relying parties of the findings that are still under responsible disclosure as "RPx".

	Client data type	Client data challenge	Client data origin	Client data origin subdomain	RP ID hash	User Presence	User Verification	User Presence and Verification	Public key ID
webauthn.io	○	○	○	○	○	○	●	○	○
webauthn.cedarcode.com	○	○	○	○	○	●	○	○	○
RP3	○	○	○	○	○	●	○	○	○
RP4	○	○	●	●	○	○	●	○	○
RP5	○	○	○	○					

● Affected ○ Not Affected

TABLE 4.1: Registration tests.

	Credential ID	Client data type	Client data challenge	Client data origin	Client data origin subdomain	User Presence	User Verification	User Presence and Verification	Signature	Signature counter 1	Signature counter 999
webauthn.io	○	○	○	○	○	○	●	○	○	○	●
webauthn.cedarcode.com	○	○	○	○	○	●	○	○	○	○	●
RP3	○	○	○	○	○	●	○	○	○	○	●
RP4	○	○	○	●	●	○	●	○	○	●	●

● Affected ○ Not Affected

TABLE 4.2: Authentication tests

An interesting observation is that registration and authentication share the same misconfigurations in our results. Following is an overview of the most interesting findings about how each relying party implements FIDO2/WebAuthn and the impacts of the identified misconfigurations.

webauthn.io

Upon successful registration, `webauthn.io` stores the credential by retrieving the credential ID from the `attestationObject` parameter [1]. We comment that, on a production website, this would not be a good idea. If an attacker manages to retrieve a credential ID of a victim and register a new credential with the same credential ID but with the attacker's username, the victim's credential gets overwritten. This can cause the legitimate user not to be able to log in to their account with their passkey. We consider this attack to be feasible as some services allow credential ID enumeration, allowing an attacker to obtain other users' credential IDs.

We comment that the `userVerification` finding is not a misconfiguration in this case. Rather, it is a deliberate option offered by the website allowing to omit it.

webauthn.cedarcode.com

`webauthn.cedarcode.com` saves the credential ID from the request parameters `ID` and `rawID` (the second and third element of Figure 4.1), unlike `webauthn.io`, which stores it from the `attestationObject`.


```
{
  "type":"public-key",
  "id":"AxCmlBkco_6_n6y7rrcFQ1BiD-1Dm8hrLX_vRxMEqMk",
  "rawId":"AxCmlBkco_6_n6y7rrcFQ1BiD-1Dm8hrLX_vRxMEqMk",
  "authenticatorAttachment":"cross-platform",
  "response":{
    "clientDataJSON":
    "eyJ0eXBlIjoid2ViYXV0aG4uY3JlYXRlIiwiY2hhbGxlbmdlIjoibEJZSmVZX2k5Z2ZyRFQtN2o2enpnT25xR
    k00WnJXR0daejRRNnJ3TVMzYyIsIm9yaWdpbiI6Imh0dHBzOi8vd2ViYXV0aG4uY2VkYXJjb2RlLmNvbSIsImN
    yb3NzT3JpZ2luIjpmYWxzZX0",
    "attestationObject":
    "o2NmbXRkbm9uZWdhdHRTdG10oGhhdXRoRGF0YVikhpnmnBxgatkgCKL-w79cENZZ53NdvwafSZw3_dI6l55FA
    AAAAQAAAAAAAAAAAAAAAAAAAAAAAAAIAMQppQZHKP-v5-su663BUNQYg_tQ5vIay1_70cTBKjJpQECAyYgASFYIHP
    9KZbxTMv1AvEufkTI2lV3F7Fz1Z4FEIVUfn6uLo3lIlgg00Ey-9Cf2LosNfShAuBepR47k-lZPdSARFmcM14GG
    bA",
    "transports":[
      "usb"
    ]
  },
  "clientExtensionResults":{
  }
}
```


FIGURE 4.1: `webauthn.cedarcode.com`'s registration assertion request.

An important finding we have is that the relying party does not check the case where the `UserPresence` flag is false and the `UserVerification` is true [2], during both registration and authentication. Omitting the `UserPresence` check means that the user does not need to be in proximity and control of the authenticator when computing a WebAuthn ceremony. Figure 4.2 shows the decoded `authenticatorData` in the request (with "flagUP" false and

[1] https://github.com/duo-labs/webauthn.io/blob/c3bb1e30e98bb9b982f4ff9bdab3ba18c1fc6869/_app/homepage/services/credential.py#L116

[2] https://github.com/cedarcode/webauthn-ruby/blob/2-stable/spec/webauthn/authenticator_assertion_response_spec.rb#L106

"flagUV" true) and the confirmation that the ceremony is successful. We comment that even if in practice it is not possible to have this case, as there are no remote authenticators that need user verification, it is still an important misconfiguration as this strictly goes against the WebAuthn standard [44].

FIGURE 4.2: webauthn.cedarcode.com not checking the `userPresence` flag.

Despite the limited impact, the vendor, Cedarcode, has confirmed our finding. Additional details can be found in Chapter 7.

RP3

Similar to webauthn.cedarcode.com, RP3 does not check the `UserPresence` flag in case the `UserVerification` flag is set to true. We mention again that this case is impractical in a realistic scenario, as remote authenticators do not exist yet. We report that the vendor has acknowledged our finding and awarded us a bounty. Further details can be found in Chapter 7.

RP4

RP4 does **not** check the `origin` value in the `clientDataJSON` object in both registration and authentication ceremonies. This misconfiguration allows an attacker to take over a victim's account, under some conditions. First, the victim has already a passwordless authentication method set up on RP4 using an authenticator that does not check the origin on the client side. Second, the attacker succeeds in making the victim visit a website under the attacker's control and perform an assertion request. They can then retrieve and forward that request to RP4, which accepts it without checking that the origin is the attacker's website. Under these conditions, this attack might lead to an account take-over.

We assume for the proof of concept that the adversary has a valid email of the victim.

1. The attacker sends the victim's email to RP4 to perform the setup request. They retrieve the challenge from the relying party.

2. The attacker creates a phishing webpage with the following JavaScript script to leverages a faulty authenticator to create a valid signature.

```
let credential = await navigator.credentials.get({ publicKey: {
```

```
  challenge: new Uint8Array([ ... challenge ... ])
}});
```

We assume that the victim's authenticator is faulty and computes the signature without checking the RP ID. The attacker then retrieves the valid signature and forwards it to the relying party. The latter does not check the origin in the `clientDataJSON` and accepts the assertion, resulting in an account takeover.

We comment that the faulty authenticator assumption is non-practical. To verify this, we buy two relatively cheap security tokens that adopt key wrapping (Key-ID FIDO2 and HyperFIDO Pro Mini) and conduct the experiment. We hypothesize that security keys using key wrapping could be mistakenly implemented by only wrapping the private key and not the RP ID. The attack did not succeed as the security keys correctly checked the RP ID.

Nevertheless, this goes against the WebAuthn specification [44] and RP4 also confirms our finding after the responsible disclosure.

From the results, we also see that RP4 does not check the `UserVerification` flag during authentication. The impact of this is an account takeover. In this case, we can craft a proof of concept attack as follows. We assume that the victim uses password-less authentication in RP4 by means of a PIN-protected security key and the attacker physically gains access to it. Then the attacker starts the authentication flow by inserting the victim's email. This step can also be brute-forced, for instance, if the attacker collects the security key in an office, they can loop through the emails of the employees. Before proceeding with the authentication, the attacker sets a breakpoint at the JavaScript method `navigator.credentials.get` on the client. The attacker then starts the authentication flow, which hits the breakpoint and pauses its execution. Next, they modify the `userVerification` value in the argument of the method to `"discouraged"`. They resume the execution flow and now the security key does not ask for the PIN to the attacker. As mentioned before, since the `userVerification` flag is not checked, the attacker can successfully log in without proving the *something-you-know* factor.

We tested this proof of concept successfully with the Security Key NFC by Yubico. We comment that this attack is not possible when the user presence and verification check coincide on the authenticator. This is the case for platform authenticators and some biometric roaming authenticators, such as the Yubikey Bio.

RP5

The testing results for RP5 are limited due to the use of standard Base64 encoding instead of Base64 URL encoding on the `attestationObject`. This does not conform with the WebAuthn specification [44]. As a result, we are not able to test for misconfigurations in the `attestationObject` with our Burp Suite extension.

Despite not finding any misconfiguration, RP5 offers some interesting insights on its implementation of FIDO2/WebAuthn. Firstly, it allows for user enumeration. During registration, in the setup request, it is possible to specify an email address to be sent to the relying party. When providing the logged-in user's email or an email that was previously not registered, RP5 replies back with a `PublicKeyCredential` object, containing also the `excludeCredentials` array. But when giving as input another user's email, this would give an error. We comment that this is not best practice, as this can enumerate the users who have previously registered a security key. A better approach would be to follow

Section 14.6.2 of the specification[44], which is to give an indistinguishable response to every provided email.

Now, we provide some observations about how the relying party stores the credentials of the user. In the assertion request's body, there are three parameters: `credentialId`, `attestationObject`, and `clientDataJSON`. The server-side checks if both the `credentialId` parameter and the `credentialId` value in the `attestationObject` match one of the entries in the `excludeCredentials` array. If so, the registration is rejected. If it is a valid request, then the relying party stores the `credentialId` value in the `attestationObject` parameter and ignores the first `credentialId` parameter. We do not understand why the stored ID is different than the checked one. This shows how WebAuthn is not mature enough and developers find difficulties implementing this protocol.

4.2 CSRF

To assess the feasibility of Cross-Site Request Forgery (CSRF) attacks, we first look at the defensive mechanisms that websites adopt, such as session cookie(s)'s `SameSite` attribute and the presence of a CSRF token. In case the `SameSite` attribute of the session cookies are `None` and the CSRF token is not present, we can build a social engineering attack and send it to the victim.

Table 4.3 provides an overview of the CSRF prevention methods used by the targeted relying parties. Under the first column, we write the `SameSite` value of the session cookie(s). In the parenthesis, we list the names of the session cookies. In the second column, we write the name of the CSRF token and we append a "?" to non-explicit token (e.g. Bearer tokens). In case of a CSRF attack, these are not automatically sent on behalf of the victim so this mechanism mitigates CSRF.

We note that these defense mechanisms are not exhaustive and should to be complemented with other security components. For instance, the `SameSite` attribute protection can be bypassed if a website is vulnerable to Cross-Site Scripting (XSS). An attacker can exploit it and make the victim send requests, in this case, without "crossing" websites. This leads to the cookies being sent even if they have `SameSite` value of `Lax` or `Strict`. Furthermore, CSRF token protection can be bypassed in case it is not tied specifically to a user's session. In our experiments, we tried different techniques to exploit the latter (by using the attacker's CSRF token on the victim's session, removing the CSRF token, leaving the CSRF token empty, etc.). We do not look for XSS vulnerabilities, as we consider that out of the scope of this work.

The results of our conducted attacks are limited, indicating that the majority of the targeted relying parties defend correctly against CSRF. Our attack worked only against the demo website `webauthn.io`.

RP3

When testing for RP3 misconfigurations, we notice that by only repeating the assertion request, we can register a new authenticator within RP3. This implies that the challenge is not invalidated after its first use and therefore this exhibits the same issue as CVE-2022-42731 [32]. Since it takes only one request to compute a WebAuthn ceremony, this could lead to a CSRF attack. Now, we want to test whether it is possible to let another user repeat this assertion request with our public key using their session. For this attack to work, it is needed that the challenge in the assertion request is not tied to the user session; otherwise, the challenge in the victim's request will be invalidated as it is created by the

	SameSite	CSRF token
webauthn.io		
RP3	>=Lax (redacted)	(redacted)
webauthn.cedarcode.com	None (_webauthn_app_session)	X-Csrf-Token header
bestbuy.com	Lax (at, ut)	
authgear.com	Lax (session)	gorilla.csrf.Token
app.hancock.ink		Bearer token?
hanko.io	Strict (hanko)	
RP5	None (redacted)	(redacted)
locker.io		Bearer token?
minter.io	Lax (session)	
nintendo.com	None	csrfToken in JSON body
RP4	(empty) (redacted)	

TABLE 4.3: SameSite cookies and CSRF tokens of the targeted relying parties.

attacker.

As described in Section 3.2, we emulate the adversary and the victim. We make a registration ceremony manually with the attacker noting of the challenge used. We then register an authenticator with the victim's account, replacing the challenge with the attacker's in the assertion request. This request was not successful. Moreover, the session cookie(s) have the `SameSite` attribute set at least as `Lax`. Indeed, by constructing a CSRF attack on Burp Suite and emulating the victim by clicking on it, we note that the request does not contain any cookies.

To conclude, RP3 rewarded us with a bounty even though they commented that this case does not present a significant security risk.

webauthn.io

Here we present our successful CSRF attack against `webauthn.io`. This demo website does not implement session management. From the `webauthn.io` source code[3], we notice the `@csrf_exempt` decorator in the `registration_verification.py` and `authentication_verification.py` files. This removes the CSRF protections for the registration and authentication ceremonies.

To craft a valid CSRF proof of concept, we need to send the setup request with the username of the victim. When intercepting the subsequent assertion request, we need to pay attention to a few details:

- the forged request cannot be of `Content-Type application/json`, so we change it to be `text/plain`. In this case, the server does not check the content type, so we can proceed by constructing the payload.
- the body of a text/plain payload must contain an "=" character. We append it at the end of any Base64 encoded value (as this would be considered as padding and

[3] https://github.com/duo-labs/webauthn.io

not cause decoding errors).

We execute the CSRF attack and receive a confirmation from the server. At this point, our authenticator stores a credential ID which is tied server-side with the username of the victim. We confirm the attack was successful by being able to log in with the victim's username. Limitations are that no cookie-based session management is used in `webauthn.io`. In that case, we would have not been able to execute the attack, because `webauthn.io` cookies have the `SameSite` attribute set as `Lax`.

Chapter 5

Discussion

In this chapter, we review the FIDO2/WebAuthn technology in tested environments, comment on the decisions made in the process, and highlight some key findings in our analysis results. We outline our contribution to the progress of the FIDO2/WebAuthn technology through a comprehensive design, review, and validation of our methodology.

In the first section, we discuss our contribution to creating a tool to ease the automated testing of FIDO2/WebAuthn misconfigurations. We conclude by discussing the difficulties we have found in the creation of the took and in the scanning process.

In the second section, we reason the successes and failures of the CSRF attacks. We discuss whether this new type of attack could break the security of FIDO2/WebAuthn. Our aim is to provide valuable insights that can further enhance the understanding and mitigation of potential risks associated with the FIDO2/WebAuthn technology.

In the final part, we discuss the limitations of our work. We comment on whether these can affect the applicability and reliability of our methodology in real-world scenarios.

5.1 Security misconfigurations

Our results show that it is possible to create a tool for scanning for FIDO2/WebAuthn misconfigurations. Despite an initial manual effort, the scanning part is automated through a comprehensive analysis of requests and responses exchanged between the client and the relying party. Creating a completely automated scanner for WebAuthn ceremonies is not trivial. While the WebAuthn specification [44] outlines data exchange between clients and relying party servers, the diversity in how relying parties implement these flows presents a challenge for automated tools. For instance, Figure 5.1 and Figure 5.2 clearly show the differences between two assertion requests' bodies. From what we have observed in our tests, relying parties have the freedom to design the execution of the WebAuthn execution flow by choosing in which format, encoding, capitalization, or content type the communication with the client should occur. It is therefore not easy to program a one-size-fits-all tool that automatically parses the needed values and updates them in the right format. Moreover, WebAuthn ceremonies flows differ slightly according to how relying parties manage sessions and so the number of valid requests to be repeated.

Nevertheless, we provide a comprehensive methodology and starting point for future implementations. We make our tool such that it is extensible for future work so that developers can integrate it with more test cases and relying parties. This work can help security professionals get a better understanding of FIDO2/WebAuthn and the implementation of these.

We comment that extending the extension WebAuthn CBOR decoder is an important

contribution because it helps the testing of WebAuthn flows significantly. By supporting more formats and the `authenticatorData` structure, we can read the CBOR-encoded content of the test cases and make sure that our tool is modifying the values properly.

Regarding the results, while certain misconfigurations were identified, it is still difficult to find practical attacks against most of them. The `UserPresence` misconfiguration has been discovered before in other applications (CVE-2021-38299 [31]) and other technical report [20] also mentions that this is not a critical issue. Despite being one of the mandatory checks in the WebAuthn standard [44], its mis-check does not have a significant security impact.

Another misconfiguration in our results is the `origin` value mis-check, found in RP4. Even if this is enforced in Section 7 of the standard [44] and Section 13.4.9 of the draft [47], there are no practical attacks that can exploit it. Section 13.4.9 indeed mentions that this validation serves as an additional layer of protection in case of a faulty authenticator, which is not practical as discussed in the results section.

The most impactful result is the `userVerification` mis-check. This results in a practical attack which leads to a potential account take-over. After responsibly disclosing this finding, RP4 accepted this vulnerability and assigned it a medium priority. This is because the prerequisites of this attack are not easy to meet. These are that the victim must have set up a password-less authentication with a roaming authenticator and the attacker needs to have physical access to it, as explained in Section 4.1.

Despite the limited quantity and impact of our results, we remark on the validity and effectiveness of our approach in automating the testing process of WebAuthn misconfigurations. After responsibly disclosing out findings, we got confirmation from the respective vendors that these were valid. In particular, `webauthn.cedarcode.com`, RP3, and RP4 all confirmed at least one of our findings and rewarded us with a bounty.

We evaluate our tool's performance by estimating how much it increases the efficiency in a comprehensive test of the WebAuthn misconfigurations. We have created 9 test cases for the registration ceremony and 11 test cases for the authentication one. Despite an initial effort to extend the tool to be tailored to the targeted websites, we are able to automate 20 test cases. The exact effectiveness depends on the tester's ability to understand the target relying party's WebAuthn ceremonies and extend our tool accordingly. For this, we make sure to program our tool to be easily extendable.


```
{
  "username":"a",
  "response":{
    "id":"WZOvkeVPUTrkRElA06ai_rZuKMP_9pHzSo2DbgVOs9Y",
    "rawId":"WZOvkeVPUTrkRElA06ai_rZuKMP_9pHzSo2DbgVOs9Y",
    "response":{
      "attestationObject":
      "o2NmbXRkbm9uZWdhdHRtdG1oGhhdXRoRGF0YVikdKbqkhPJnC90siSSsyDPQCYqlMGpUKA5fyklC2CEHvB
      FAAAAAQAAAAAAAAAAAAAAAAAAAAAAIFmTr5HlTlE65ERJQNOmov62bijD__aR8OqNg24FdLPWpQECAyYgASF
      YIMlfWZYo9sCuqPrPmZYHSBWkAAxolLIy_9RZOQnjaytIlggMzbIcljXF3kMJms18u2kWQtZvyVgfUHvjM6
      UInKTBlA",
      "clientDataJSON":
      "eyJ0eXBlIjoid2ViYXV0aG4uY3JlYXRlIiwiY2hhbGxlbmdlIjoiOEVoYjczTjdTNHQ4WGxVMUdGeUxRZXF
      KbWN3SzhEOERoZlRjVlFKVWJ5ekiLX0txTnRoUmh5aU5jW9hORGZ6dDhzSC1mQVd0a05odlEzQjJmSFhOdFE
      iLCJvcmlnaW4iOiJodHRwczovL3dlYmF1dGhuLmlvIiwiY3Jvc3NPcmlnaW4iOmZhbHNlfQ",
      "transports":[
        "usb"
      ],
      "publicKeyAlgorithm":-7,
      "publicKey":
      "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEzV92lij2wK6ok_M-bZgdIFaQADGiUsjL_1Fk5CeNrK0zNsh
      yWNc_eQwmayXy7aRZClm_JWB9Qe-MzpQicpMGUA",
      "authenticatorData":
      "dKbqkhPJnC90siSSsyDPQCYqlMGpUKA5fyklC2CEHvBFAAAAAQAAAAAAAAAAAAAAAAAAAAAAIFmTr5HlTlE
      65ERJQNOmov62bijD__aR8OqNg24FdLPWpQECAyYgASFYIMlfWZYo9sCuqPrPmZYHSBWkAAxolLIy_9RZOQ
      njaytIlggMzbIcljXF3kMJms18u2kWQtZvyVgfUHvjM6UInKTBlA"
    },
```


FIGURE 5.1: `webauthn.io` registration assertion request.


```
f.req=
%5B%5B%5B%22B4hajb%22%2C%22%5B1%2C1%2Cnull%2C%5B53%2Cn
ull%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2
Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull
%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2Cnu
ll%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2C
null%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%
2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2Cnull%2C%5B
null%2C%5B%5C%22PiizA7Al%2BoDWaMjRBDBfUXBct48kYCdKHaoT
uN%2Fyo%2FU%3D%5C%22%2C%5C%22MEUCIBa2cB2dHE9ik63yhW8Yf
MH8Mch18BLsxbdAWflJy2N3AiEA4i3kCEx%2BLr38jwtXh3lqNh%2B
```


FIGURE 5.2: `google.com` registration assertion request.

5.2 CSRF attacks

Apart from `webauthn.io`, all other websites tested are not vulnerable to Cross-Site Request Forgery (CSRF) attacks. We notice that the challenge value can serve as a form of CSRF token since it is randomly generated by the relying party. On top of finding websites that do not use CSRF tokens during WebAuthn ceremonies and do set the `SameSite` attribute to `None` to session cookies, it is also needed that the challenge is not tied to the session. As expected, none of the relying parties misconfigured all three of these checks. The testing process for CSRF vulnerabilities included difficulties such as identifying and manipulating CSRF tokens. This resulted in a more manual effort and limited results in this regard.

5.3 Limitations

This work has limitations. With our tests, we could not cover some aspects of the FIDO2/WebAuthn protocol. Firstly, we do not consider authenticators' attestation modes. Their function is for relying parties to prevent the use of weak or uncertified authenticators, that could compromise the security of the protocol. Bindel et al. [7] provide a security and privacy overview of the attestation modes. We comment that we do not include attestation modes testing in our work due to the absence of concrete attack in this area.

Moreover, we do not consider WebAuthn extensions [29] due to time constraints. These extensions are extra properties in the `PublicKeyCredential` object, passed as an argument to the methods `navigator.credentials.create` and `navigator.credentials.get`. For example, the `credProps` extension informs the relying party about whether the created credential is resident/discoverable.

We also do not consider resident keys. These are credentials that store the username along with the public key information. With these, during the authentication process, the user does not need to manually input the username. This might allow a potential adversary to skip the username enumeration phase.

Chapter 6

Conclusions

In this work, we conducted a thorough study of FIDO2 and its sub-protocol WebAuthn. We started with a comprehensive analysis of the communication between the client and the relying party. According to the WebAuthn specification, in order to complete successfully a ceremony, the latter needs to perform certain verifications on the client's input data. Our goal was to observe the behaviour of relying parties once we presented with invalid or faulty data. In particular, we targeted three WebAuthn data objects: `clientDataJSON`, `attestationObject`, and `authenticatorData`. We created nine test cases for the registration ceremony and eleven for authentication.

Moreover, we proposed a methodology that helps automate the black-box testing of WebAuthn configurations. To achieve this, we created a Burp Suite extension capable of sending our test cases and detecting whether relying parties accepted our faulty input. If so, we considered this as a misconfiguration. Once a misconfiguration is found, we tried to craft proof of concepts of possible attacks against the vulnerability and executed them.

We also considered Cross-Side Request Forgery (CSRF) attacks with a potential impact of an account takeover. We crafted a malicious webpage that, if accessed by a victim, would register the attacker's public key into the victim's account as a password-less authentication method. If successful, the attacker can log in to the victim's account as they possess the correct respective private key and prove their identity.

This chapter is composed of two sections. In the initial one, we address the research questions stated in the beginning. Subsequently, we present our concluding thoughts on this promising protocol.

6.1 Research answers

We recall the proposed research questions:

- **RQ1**: How can we ease and automate the testing process for FIDO2/WebAuthn misconfigurations in web applications?

To answer **RQ1**, we have introduced a novel methodology to scan for WebAuthn vulnerabilities using a custom Burp Suite extension. Given a relying party, we can program to extend our Burp Suite extension to target that and execute our test cases on it. Despite the initial programming effort, we still consider our methodology to be a significant improvement over manual testing. Creating a tool that can create valid WebAuthn requests without prior knowledge of what valid WebAuthn requests look like is challenging. For this reason, we designed our tool to be easily expandable for future use.

- What are the design choices of current implementations of FIDO2/WebAuthn and how do they comply with the best security-related practices?

To answer **RQ2**, our results show that current implementations of five FIDO2/WebAuthn relying parties do not fully comply with the best security practices. Our test cases consist of server-side checks specified by the WebAuthn standard [44] and previously disclosed vulnerabilities (CVE-2021-38299, CVE-2022-42731). Our most impactful results are:

- `origin` value in `clientDataJSON` mis-check.
- `userPresence` flag mis-check.
- `userVerification` flag mis-check for password-less authentication.

Our results show that of the five targeted relying parties, three exhibited security misconfigurations, with one vulnerability posing a tangible risk of account takeover.

We have reported our security findings to the respective vendors and had confirmation of them as security vulnerabilities.

- What types of vulnerabilities in FIDO2/WebAuthn relying parties exist and how can we exploit these?

To answer **RQ3**, we attempted to exploit the previously found misconfigurations to see if these could lead to any practical attacks. We have successfully found one concrete attack against the `UserVerification` mis-check in the WebAuthn authentication flow. This attack allows an attacker to compromise a victim's account under specific circumstances. These are that the victim must have set their roaming security key as a password-less WebAuthn authentication method and that the attacker has physical access to it. These make the attack less practical in practice, its impact could be an account takeover if exploited.

Furthermore, we investigated leveraging web security attacks such as Cross-Site Request Forgery. Our goal was to achieve an account takeover. Our result shows that only `webauthn.io` was vulnerable to this attack. We believe that this type of attack was previously unknown; our contribution lies in uncovering and addressing it.

6.2 Final remarks

After extensively studying the behaviors and request flows of WebAuthn in numerous services, we conclude the implement of technology in replying parties is not mature enough. We notice many inconsistencies among relying parties in terms of how WebAuthn is deployed. Every browser and operating system presents the password-less experience in a different-looking pane. The terminology, the format, and the encoding each service uses are inconsistent and confusing. A remedy to this is putting pressure on browsers and relying parties to standardize terms.

Despite these inconsistencies, the FIDO2 protocol is highly secure and we definitely recommend its adoption. All parties involved - relying party, client, and authenticator - implement security measures, making it practically difficult to compromise the authentication and registration processes.

Looking towards the future of password-less authentication, we comment that despite more services adopting FIDO2, it is still very difficult to completely remove passwords from the authentication process. For instance, there are currently no practical remedies

for recovering the account after losing an authenticator. FIDO suggests either letting users register multiple authenticators or re-running the authentication using traditional methods.

Our last comments regard Passkeys syncing. Password managers can now sync passkeys across devices. On MacOS, Google Chrome allows created passkeys to be saved and synced on the iCloud Keychain to better support usability. We comment that syncing security keys among devices improves significantly the user's experience and usability, but it is a bad practice for security as it opens up more potential attack surfaces. For best security, we suggest storing the private keys in the Trusted Platform Modules than exporting them for syncing purposes.

Chapter 7

Ethics

In this chapter, we discuss the ethical aspects behind this research work. During our experiments, we make sure to not interfere with the normal execution of the targeted applications. In particular, we are careful to preserve the confidentiality, integrity, and availability of the services. To address confidentiality, we create test accounts on purpose and execute attacks within accounts under our control. We do not involve any sensitive data or personally identifiable information of other users. To address integrity, we do not modify any server-side data. In our tests, we only target the registration and authentication flows of the services and we do not modify any critical data. Furthermore, the faulty input we provide does not contain any malicious payload and is chosen not to produce any low-level issues such as buffer overflows. To address availability, our max requests rate is around 30 requests every 2 minutes. We consider this to be not problematic also because we test against large companies which receive a significant amount of traffic at every moment.

Our work has been reviewed and approved by the Ethics committee of the Computer & Information Sciences (CIS) of the University of Twente on date 13/03/2024.

7.1 Responsible Disclosure Programs

The following is a summary of responsible disclosure interactions with the vendors.

7.1.1 Cedarcode

We responsibly disclose the `userPresence` mis-check by contacting security@cedarcode.com, which suggested creating a Pull Request on GitHub.

Timeline:

- On 16/01/2024, we corrected the `userPresence` mis-check through a Pull Request.
- On 02/02/2024, we got the confirmation that the issue is being fixed and that we may publish this finding.

The full conversation is publicly available at https://github.com/cedarcode/webauthn-ruby/pull/418.

7.1.2 RP3

RP3's Responsible disclosure is hosted on a public bug bounty platform. We make sure that all our requests are targeting in-scope domains. We responsibly disclosed our findings, which include the `userPresence` mis-check and the repeatable challenge.

We have responsibly disclosed the `userPresence` mis-check through the bug bounty platform. Timeline:

- On 16/01/2024, we submitted the `userPresence` finding.
- On 23/01/2024, the issue was validated and RP3 started working on a fix.
- On 13/02/2024, RP3 confirmed the finding and rewarded us with a bug bounty.

We have also responsibly disclosed the repeatable challenge finding through the bug bounty platform. Timeline:

- On 01/02/2024, we submitted the repeatable challenge finding.
- On 01/02/2024, the issue was validated and RP3 started analyzing its impact.
- On 05/03/2024, RP3 considered the finding not a security vulnerability, but still rewarded us with a bug bounty and closed the issue.

7.1.3 RP4

We have responsibly disclosed the `userVerification` mis-check through RP4's public bug bounty platform. Timeline:

- On 19/01/2024, we submitted the `userVerification` finding.
- On 31/01/2024, we got confirmation of RP4 acknowledging the vulnerability and working on a fix.

We have also responsibly disclosed the `origin` mis-check through RP4's public bug bounty platform. Timeline:

- On 08/02/2024, we submitted the `origin` finding.
- On 01/03/2024, we got confirmation of RP4 acknowledging the vulnerability and working on a fix.

Bibliography

[1] Sector 7. Zoom RCE from Pwn2Own 2021. https://sector7.computest.nl/post/2021-08-zoom/, 2021. Accessed: 20/11/2023.

[2] Sector 7. Process injection: breaking all macOS security layers with a single vulnerability. https://sector7.computest.nl/post/2022-08-process-injection-breaking-all-macos-security-layers-with-a-single-vulnerabili 2022. Accessed: 20/11/2023.

[3] Aftab Alam, Katharina Krombholz, and Sven Bugiel. Poster: Let history not repeat itself (this time) - tackling webauthn developer issues early on. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2669–2671. ACM, 2019. doi:10.1145/3319535.3363283.

[4] Fido Alliance. Conformance Self-Validation Testing. https://fidoalliance.org/certification/functional-certification/conformance/. Accessed: 24/01/2023.

[5] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, and Bogdan Warinschi. Provable security analysis of FIDO2. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 125–156. Springer, 2021. doi:10.1007/978-3-030-84252-9_5.

[6] Nina Bindel, Cas Cremers, and Mang Zhao. Fido2, CTAP 2.1, and webauthn 2: Provable security and post-quantum instantiation. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 1471–1490. IEEE, 2023. doi:10.1109/SP46215.2023.10179454.

[7] Nina Bindel, Nicolas Gama, Sandra Guasch, and Eyal Ronen. To attest or not to attest, this is the question - provable attestation in FIDO2. *IACR Cryptol. ePrint Arch.*, page 1398, 2023. URL: https://eprint.iacr.org/2023/1398.

[8] Dave Childers. State of the Auth: Experiences and Perceptions of Multi-Factor Authentication. https://duo.com/assets/ebooks/state-of-the-auth-2021.pdf, 2021. Accessed: 27/12/2023.

[9] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The tangled web of password reuse. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014. URL: https://www.ndss-symposium.org/ndss2014/tangled-web-password-reuse.

[10] Emma Dauterman, Henry Corrigan-Gibbs, David Mazières, Dan Boneh, and Dominic Rizzo. True2f: Backdoor-resistant authentication tokens. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 398–416. IEEE, 2019. doi:10.1109/SP.2019.00048.

[11] Alessia Michela Di Campi. Password guessing: learn the nature of passwords by studying the human behavior. *Università Ca'Foscari Venezia*, 2021.

[12] Michael Fagan, Yusuf Albayram, Mohammad Maifi Hasan Khan, and Ross Buck. An investigation into users' considerations towards using password managers. *Hum. centric Comput. Inf. Sci.*, 7:12, 2017. doi:10.1186/s13673-017-0093-6.

[13] Sofia Emelianova Fawaz Mohammad, Jecelyn Yeen. WebAuthn: Emulate authenticators. https://developer.chrome.com/docs/devtools/webauthn/, 2020. Accessed: 25/09/2023.

[14] FIDO. FIDO2. https://fidoalliance.org/fido2/. Accessed: 04/12/2023.

[15] FIDO. FIDO Security Reference. https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-security-ref-v2.0-id-20180227.html, 2018. Accessed: 10/12/2023.

[16] FIDO. Client to Authenticator Protocol (CTAP). https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html, 2019. Accessed: 07/01/2024.

[17] FIDO. The State of Strong Authentication. https://fidoalliance.org/the-state-of-strong-authentication/, 2021. Accessed: 27/12/2023.

[18] Athanasios Vasileios Grammatopoulos, Ilias Politis, and Christos Xenakis. Blind software-assisted conformance and security assessment of fido2/webauthn implementations. *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, 13(2):96–127, 2022. doi:10.22667/JOWUA.2022.06.30.096.

[19] Iness Ben Guirat and Harry Halpin. Formal verification of the W3C web authentication protocol. In Munindar P. Singh, Laurie A. Williams, Rick Kuhn, and Tao Xie, editors, *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security, HoTSoS 2018, Raleigh, North Carolina, USA, April 10-11, 2018*, pages 6:1–6:10. ACM, 2018. doi:10.1145/3190619.3190640.

[20] Timon Hackenjos. FSA-2021-1 Missing User Presence Check in webauthn-framework. https://www.fzi.de/wp-content/uploads/2022/02/doku-missinguserpresence.pdf, 2021. Accessed: 11/01/2023.

[21] Lucjan Hanzlik, Julian Loss, and Benedikt Wagner. Token meets wallet: Formalizing privacy and revocation for FIDO2. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 1491–1508. IEEE, 2023. doi:10.1109/SP46215.2023.10179373.

[22] HaveIBeenPwned. HaveIBeenPwned Pwned Websites. https://haveibeenpwned.com/PwnedWebsites. Accessed: 28/12/2023.

[23] Troy Hunt. The 773 Million Record "Collection #1" Data Breach. https://www.troyhunt.com/the-773-million-record-collection-1-data-reach/, 2019. Accessed: 28/12/2023.

[24] Amnesty International. When Best Practice Isn't Good Enough: Large Campaigns of Phishing Attacks in Middle East and North Africa Target Privacy-Conscious Users. https://www.amnesty.org/en/latest/research/2018/12/when-best-practice-is-not-good-enough/, 2018. Accessed: 27/12/2023.

[25] Mohammed Jubur, Prakash Shrestha, Nitesh Saxena, and Jay Prakash. Bypassing push-based second factor and passwordless authentication with human-indistinguishable notifications. In Jiannong Cao, Man Ho Au, Zhiqiang Lin, and Moti Yung, editors, *ASIA CCS '21: ACM Asia Conference on Computer and Communications Security, Virtual Event, Hong Kong, June 7-11, 2021*, pages 447–461. ACM, 2021. doi:10.1145/3433210.3453084.

[26] Wang Kang. GPN18:U2Fishing: Potential Security Threat Introduced by U2F Key Wrapping Mechanism. https://entropia.de/GPN18:U2Fishing:_Potential_Security_Threat_Introduced_by_U2F_Key_Wrapping_Mechanism, 2018. Accessed: 08/01/2023.

[27] Myounghoon Kim, Joon Suh, and Hunyeong Kwon. A study of the emerging trends in SIM swapping crime and effective countermeasures. In Van Hung Trong, Jongwoo Park, Vo Thi Thanh Thao, and Jongbae Kim, editors, *7th IEEE/ACIS International Conference on Big Data, Cloud Computing, and Data Science, BCD 2022, Danang, Vietnam, August 4-6, 2022*, pages 240–245. IEEE, 2022. doi:10.1109/BCD54882.2022.9900510.

[28] Dhruv Kuchhal. *Building Trust In The Online Ecosystem Through Empirical Evaluations Of Web Security And Privacy Concerns*. PhD thesis, Georgia Institute of Technology, 2023.

[29] Mozilla. Web Authentication extensions. https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API/WebAuthn_extensions, 2023. Accessed: 03/01/2023.

[30] NIST. CVE-2021-3011. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3011, 2021. Accessed: 11/12/2023.

[31] NIST. CVE-2021-38299. https://nvd.nist.gov/vuln/detail/CVE-2021-38299, 2021. Accessed: 16/02/2024.

[32] NIST. CVE-2022-42731. https://nvd.nist.gov/vuln/detail/CVE-2022-42731, 2022. Accessed: 20/12/2023.

[33] PortSwigger. Burp Suite - Application Security Testing Software - PortSwigger. https://portswigger.net/burp/. Accessed: 13/12/2023.

[34] PortSwigger. Creating Burp extensions. https://portswigger.net/burp/documentation/desktop/extensions/creating, 2023. Accessed: 15/12/2023.

[35] PortSwigger. Getting started with Burp Suite. https://portswigger.net/burp/documentation/desktop/getting-started/, 2023. Accessed: 14/12/2023.

[36] PortSwigger. Installing Burp's CA certificate. https://portswigger.net/burp/documentation/desktop/external-browser-config/certificate, 2023. Accessed: 17/12/2023.

[37] Thomas Roche, Victor Lomné, Camille Mutschler, and Laurent Imbert. A side journey to titan. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 231–248. USENIX Association, 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/roche.

[38] Thomas Roth, Fabian Freyer, Matthias Hollick, and Jiska Classen. Airtag of the clones: Shenanigans with liberated item finders. In *43rd IEEE Security and Privacy, SP Workshops 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 301–311. IEEE, 2022. doi:10.1109/SPW54247.2022.9833881.

[39] Larissa Schrempp. Formal verification of fido2 with human interaction. Master's thesis, Swiss Federal Institute of Technology (ETH) Zurich, 2023.

[40] SecureAuth. CVE-2022-42731. https://secureauthcorp.wpenginepowered.com/wp-content/uploads/2023/04/2023-state-of-authentication-report-3.pdf, 2022. Accessed: 27/12/2023.

[41] Mark Stockley. The sound of you typing on your keyboard could reveal your password. https://www.malwarebytes.com/blog/uncategorized/2023/12/the-sound-of-you-typing-on-your-keyboard-could-reveal-your-password, 2023. Accessed: 03/01/2023.

[42] Port Swigger. What is CSRF? https://portswigger.net/web-security/csrf. Accessed: 11/12/2023.

[43] W3C. Attestation object layout. https://www.w3.org/TR/webauthn-2/#fig-attStructs, 2021. Accessed: 13/12/2023.

[44] W3C. Web Authentication: An API for accessing Public Key Credentials Level 2. https://www.w3.org/TR/webauthn-2/, 2021. Accessed: 11/12/2023.

[45] W3C. Web Authentication: An API for accessing Public Key Credentials Level 2. https://www.w3.org/TR/webauthn-2/#sctn-registering-a-new-credential, 2021. Accessed: 10/12/2023.

[46] W3C. Web Authentication: An API for accessing Public Key Credentials Level 2. https://www.w3.org/TR/webauthn-2/#sctn-verifying-assertion, 2021. Accessed: 10/12/2023.

[47] W3C. Web Authentication: An API for accessing Public Key Credentials Level 3. https://www.w3.org/TR/webauthn-3/, 2021. Accessed: 16/11/2023.

[48] Jingguo Wang, Tejaswini C. Herath, Rui Chen, Arun Vishwanath, and H. Raghav Rao. Research article phishing susceptibility: An investigation into the processing of a targeted spear phishing email. *IEEE Trans. Prof. Commun.*, 55(4):345–362, 2012. doi:10.1109/TPC.2012.2208392.

[49] Alex Weinert. Your Pa$$word doesn't matter. https://techcommunity.microsoft.com/t5/microsoft-entra-blog/your-pa-word-doesn-t-matter/ba-p/731984, 2019. Accessed: 11/01/2023.

[50] whatwg. HTML Standard. https://html.spec.whatwg.org/multipage/browsers.html#concept-origin-effective-domain, 2. Accessed: 10/12/2023.

[51] Tarun Kumar Yadav and Kent E. Seamons. A security and usability analysis of local attacks against FIDO2. *CoRR*, abs/2308.02973, 2023. URL: https://doi.org/10.48550/arXiv.2308.02973, arXiv:2308.02973, doi:10.48550/ARXIV.2308.02973.

[52] Shikun Zhang, Sarah Pearman, Lujo Bauer, and Nicolas Christin. Why people (don't) use password managers effectively. In Heather Richter Lipford, editor, *Fifteenth Symposium on Usable Privacy and Security, SOUPS 2019, Santa Clara, CA, USA, August 11-13, 2019*. USENIX Association, 2019. URL: https://www.usenix.org/conference/soups2019/presentation/pearman.