An Analysis of Attack Vectors Against FIDO2 Authentication

Alexander Berladskyy and Andreas Aßmuth
Kiel University of Applied Sciences, Kiel, Germany
e-mail: alexander.berladskyy@stu.haw-kiel.de, andreas.assmuth@haw-kiel.de

***Abstract*—Phishing attacks remain one of the most prevalent threats to online security, with the Anti-Phishing Working Group reporting over 890,000 attacks in Q3 2025 alone. Traditional password-based authentication is particularly vulnerable to such attacks, prompting the development of more secure alternatives. This paper examines passkeys, also known as FIDO2, which claim to provide phishing-resistant authentication through asymmetric cryptography. In this approach, a private key is stored on a user's device, the authenticator, while the server stores the corresponding public key. During authentication, the server generates a challenge that the user signs with the private key; the server then verifies the signature and establishes a session. We present passkey workflows and review state-of-the-art attack vectors from related work alongside newly identified approaches. Two attacks are implemented and evaluated: the Infected Authenticator attack, which generates attacker-known keys on a corrupted authenticator, and the Authenticator Deception attack, which spoofs a target website by modifying the browser's certificate authority store, installing a valid certificate, and intercepting user traffic. An attacker relays a legitimate challenge from the real server to a user, who signs it, allowing the attacker to authenticate as the victim. Our results demonstrate that successful attacks on passkeys require substantial effort and resources. The claim that passkeys are phishing-resistant largely holds true, significantly raising the bar compared to traditional password-based authentication.**

Keywords-Passkeys;FIDO2;CTAP;WebAuthn.

I. Introduction

Passwords have been the predominant authentication method for decades. A password is a user-chosen string that is hashed and stored server-side. Upon authentication, the user's input is hashed and compared with the stored hash to verify its validity. This method is simple to implement and remains the most widely used authentication mechanism.

However, passwords have significant drawbacks. They are vulnerable to phishing attacks, and in the event of a server-side data breach, the hashed passwords can be exposed and potentially cracked. The Anti-Phishing Working Group (APWG) reported 892,494 phishing attacks in the third quarter of 2025 alone [1], and data breaches continue to be a prevalent threat.

In response to these vulnerabilities, passkeys, based on the Fast Identity Online (FIDO2) standard, provide a more secure alternative. These authentication methods are designed to be resistant to phishing, ensuring that even if a data leak occurs, no sensitive information that could compromise the user's security is exposed.

This work investigates various attack vectors targeting FIDO2 to assess the feasibility and cost of potential attacks, and to evaluate whether it is truly resistant to phishing. The structure of the paper is as follows: Section II provides an overview of how FIDO2 works, Section III reviews previous research on attacks against passkeys, Section IV outlines the attack vectors examined, Section V demonstrates two specific attacks, Section VI discusses the findings, and Section VII presents the conclusions and directions for future research.

II. Background

The following sections describe the concepts of FIDO2 and introduce the threat model.

A. Client To Authenticator Protocol

The Client To Authenticator Protocol (CTAP) [2][3] is a communication protocol that allows a client, such as a browser, to communicate with an authenticator. An authenticator can be a hardware device, like a phone or a hardware key [4] or software-based authenticators that communicate using the same protocol. Examples include Windows Hello or applications that implement security mechanisms. CTAP, a core component of FIDO2 (cf. project's website [5]), enables passwordless and secure authentication on the Internet. Unlike password-based systems where authentication occurs server-side through hash comparison, authentication with CTAP is performed on the devices which belong to the user. After the authentication on the local device, e.g., by fingerprint or a PIN, the secret key can be accessed and used for authentication by signing a challenge received by the Relying Party (RP). The RP is the service which a user is trying to authenticate to. This could be any web service, providing a way to authenticate to a user account. The RP saves the public key of the owner and thus can verify if the person is the real owner of the key. The private key, which is generated by the authenticator, remains inaccessible to the end users – they only need a way to unlock the authenticator, e.g., a PIN or a fingerprint, and a device that is able to use CTAP. A key pair is also called *passkey* in this context.

B. WebAuthn

The second part of FIDO2 is WebAuthn [6][7] (Web Authentication API). CTAP covered the communication between the client and the authenticator. The communication between the client and the RP is done by WebAuthn. WebAuthn implements two workflows, also called ceremonies, namely signup or registration and the login or authentication workflow. The signup process begins by the user choosing a passkey login instead of a password login. The RP receives the request, containing user information, and generates a challenge, which is random data, e.g., an array of random numbers. The challenge and the user information is stored temporarily on the server, until the ceremony is finished. Then, the server responds with the challenge, user information and server information back

to the client. Now the client tries to create the necessary credentials (public-key pair), by first authenticating the user via CTAP. After the generation of the keys, the private key is used to sign the challenge – this process is also called assertion. Attestation on the other hand, is a certificate, verifiable by the RP, that proves the key being generated by a specific authenticator device like a Hardware Security Key (HSK). The signed challenge, user information, server (RP) information, device information and the public key are sent to the server. Finally, upon verifying the challenge and comparing the user information the process is done.

Similarly, the login workflow starts by the user sending a passkey login request and the RP generating a challenge. The RP responds with the challenge and user information and the client requests a signature on the challenge from the authenticator through CTAP. The challenge is signed and is sent with user and device information to the RP. The RP locates the public key, through the sent information and verifies the signed challenge - the authentication is done. The authentication workflow is visualized in a more simplified way in Figure 1, using a browser as the client: ⓪ the client starts passkey authentication, ① the RP responds with a challenge, ② the challenge gets forwarded to the authenticator, ③ the authenticator signs the challenge with the correct private key, ④ then the authenticator sends the signed challenge to the client, ⑤ & ⑥ the challenge gets forwarded to the RP, gets verified and the authentication workflow is finished.

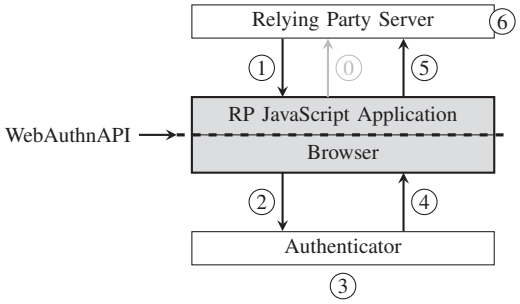

Figure 1. WebAuthn authentication workflow using a browser as client, cf. [6]

C. FIDO2

FIDO2 now incorporates both CTAP and WebAuthn.

> *FIDO2 standards use standard public key cryptography techniques to provide phishing-resistant authentication with cryptographic key pairs called passkeys. FIDO2 is designed from the ground up to protect user privacy and prevent phishing. Every passkey is unique and bound to the online service domain. The protocols do not provide information that can be used by different online services to collaborate and track a user across the services. Biometric information, if used, never leaves the user's device.* [5]

The entire workflow is already described in the sections above. In a more high level perspective:

1) A user is visiting a website and registers.
2) The device of the user is generating a passkey.
3) The passkey is used to login into the webservice.

This workflow is passwordless and also called Universal Authentication Framework (UAF). Another option is the Universal Second Factor (U2F), which incorporates a physical second factor, such as a (physical) security key.

D. Threat Model

The threat model assumes an attacker being able to compromise a victim's device with malware and perform Adversary in the Middle (AITM) attacks, such as a combination of AITM and Domain Name System (DNS) Spoofing [8]. Additionally, the victim may follow phishing links provided by the adversary. The user may use a synchronized authenticator, meaning that the passkeys are stored in the cloud, or a device-bound authenticator (local storage). The threat model focuses on attacks against end users rather than the FIDO2 protocol itself, which is assumed to be cryptographically secure. The main goal of the adversary is to gain access to the user account or to pull out personal information.

For the AITM attack, a scenario would include the adversary to be in the same network as the victim or have malware installed which performs an AITM against, e.g., an HSK.

The malware scenario includes the victim having malware installed on their device. This could be through a malicious download, a malware installation by the adversary locally (physical access) or a local malware installation by the victim via, e.g., a bad USB. In general, there are several possibilities.

A phishing scenario, includes the attacker being able to trick the victim to click on a link and to trust the phishing website through visual deception. The attacker is also able to forward Multi Factor Authentication (MFA) requests, like a text message verification on a login.

III. Related Work

This section summarizes attacks on FIDO2 which have been shown in previous publications.

Li et al. [9] proposed an Authenticator Rebind Attack on the UAF protocol, targeted at Android phones. The lack of effective entity authentication in the UAF protocols allows an attacker to bind the victim's identity to the attacker's authenticator. This attack requires pre-installed malware on the victim's device, which redirects some of the traffic to the attacker or the attacker has to have some access prior to the attack. When the victim tries to enable fingerprint authentication, the request is forwarded to the attacker's authenticator. The attacker performs their own fingerprint verification initiated by the misused authenticator and the victim performs a fake fingerprint verification initiated by the malware. The authors demonstrated the attack's effectiveness against real-world applications, providing a real threat under the given circumstances.

Barbosa et al. [10] showed two possible attacks on FIDO2, one being an impersonation attack and the other one being a rogue key attack. On the impersonation attack, the adversaries

capture the *pinToken* by using an AITM attack. The pinToken, a cryptographic key, is used by the client to send Message Authentication Code (MAC)-authenticated credential and assertion requests to the authenticator. The adversary intercepts active communication between the client and authenticator during the key agreement phase. During this phase, the adversary agrees with their own generated keys to the key agreement protocol. By doing that, the adversary, knowing the shared secret, creates a session with the client and authenticator. Now the adversary is able to create assertion requests to any relying party. Similar to Li et al. [9], the second attack includes registering a rogue key on the RP, but through the USB HID mechanism of the latest FIDO2 version.

Kumar Yadav et al. [11], present local attacks on the FIDO2 protocol. The attack methods are similar to the attacks presented previously [10], since one of the goals is to impersonate the victim. Here, browser extensions are used for performing an AITM attack between the authenticator (a hardware security key) and the RP through WebAuthn. The other goal is to bypass the cloning detection algorithm of an HSK by FIDO2. The authors underline, that the RPs are lacking detailed error messages, to notify victims that they are being attacked.

Mahdad et al. [12], propose an attack which includes running malware in the user space and uses a browser extension. The malware detects keystrokes and mouse movements - the primary function is to capture username and password in an MFA scenario. A hidden browser is launched in the background, navigating to the same service the user is trying to authenticate to. The browser extension, on the victim's browser, redirects the page to a similar-looking attacker-controlled page, prompting the victim to access their security key. Next, the hidden browser attempts to login, receives the challenge and forwards it to the attacker-controlled page. After the user authenticates with the key, the attacker is able to establish a hidden session.

Donghyun et al. [13], present a different kind of attack, on which the victim does not have to be preloaded with some kind of malware. This attack exploits the Bluetooth capabilities of CTAP and uses spoofing to reroute the victim to a phishing server. Upon visiting the phishing website and trying to log in, the adversary goes to the real RP and attempts to login. The RP responds with the relevant data needed to generate a QR-Code and the challenge. The adversary generates a legitimate QR-Code out of the received data and presents it to the victim. Next, the victim scans the QR-Code and tries to establish a Bluetooth connection to the device that generated the QR-Code. After the connection establishment, the challenge gets signed, forwarded to the adversary and thus logging the adversary in.

IV. Methodology

This section enumerates attack vectors against FIDO2 and outlines the requirements for successful exploitation. It is important to note that the following attacks do not target FIDO2 directly, as it is assumed to be secure (Section II-D); rather, they illustrate potential attack vectors, which further demonstrate the robustness of FIDO2. Each attack vector is classified according to its complexity level and setup effort of the attack; with easy, medium or hard/impossible. A key observation is that all attack scenarios require substantial effort. In all cases, an attacker must either compromise the victim's system or be within close physical range. Notably, none of the described attacks offer a method for remotely (high range) exploiting FIDO2 without first compromising the victim's system.

As already mentioned in the threat model (Section II-D), the main goal of an attacker is to gain access to the user account or to gather personal information. Personal user information could then be used to open new attack vectors on the victim; this won't be further discussed. For accessing the account, several paths exist like the ones mentioned in Section III - those will be included below.

A. Getting Victim's Key

One path, which will be practically impossible, especially on synchronized authenticators (due to cloud storage), is to get the private key for an existing account. With malware, it is possible to extract private keys, depending on where they are stored. Password managers like KeePassXC [14] store the passkeys in an encrypted database on the file system. To pull out the private keys, the encryption of the database has to be cracked; this requires malware to be installed or local access to the target device, which can be hard. Extraction methods vary from different authenticators, of course; e.g., HSKs have to be locally accessed since the keys are generated and stored on the device. The private key does not leave the HSK but is only used for signing, making it significantly harder to extract.

B. Victim Impersonation

Other paths are more feasible, like impersonating the victim. For this to work, the authentication process of FIDO2 has to be hijacked. One option would be to create a session with the RP and the client in the key agreement phase, which requires an AITM between the authenticator and the client. Barbosa et al. [10] showed this practically with the USB HID transport mechanism; due to lack of authentication on Elliptic Curve Diffie Hellman (ECDH) in CTAP, an attacker is able to create a session to the RP. The AITM attack requires malware to be installed on the victim's device. If the attacker is positioned between the authenticator and the client, a session can be established or the attacker's own key can be registered on the RP. It is important to mention that the success of this attack depends on the user being present and on malware being installed. Due to this, this path of the impersonation attack would be hard.

C. Bluetooth AITM

The aforementioned Bluetooth attack [13], where the victim does not have to be preloaded with malware, operates in a similar way. This attack adds the use of a phishing page, which is only there for visual deception. The authentication is performed on an attacker's remote device. Further, the attack requires the attacker to be in range of a user, because the Bluetooth authentication of CTAP has to be hijacked. This path depends on a user wanting to authenticate with QR-Code

scanning, the attacker being in range and a successful phishing attack. This attack will be classified as medium, because the user does not have to be preloaded with malware but still has to interact with a phishing page.

D. Authenticator Deception

Passkeys are phishing resistant [5], but still phishing websites can be utilized to gain access to the victim's account. To login as the victim, the attacker has to own a website, with the same domain as the RP. Since this is not possible, the attacker performs a DNS spoofing attack [8], forwarding requests to the phishing website. A problem which arises is the SSL certificate, which the attacker does not own. Two sub paths arise, mainly using an insecure connection or forcing the victim to trust a self signed certificate. An HTTP connection might technically work, most browsers don't explicitly warn you as they would with self-signed certificates - for inattentive users this would go unnoticed. However, the in-browser WebAuthn API only allows HTTP access when running on localhost, making it impossible to use WebAuthn over an HTTP connection in a real attack scenario. When using HTTPS the victim has to trust the self signed certificates, because it is not possible to gain the real certificate of the RP. This requires malware or physical access to the victim's device to install the certificate, either into the browser or into the OS certificate authorities. The latter option would be even harder, because this path usually needs administrative rights. First, a DNS spoof is performed and the victim is lured to the phishing website of the RP. Since the interface and the domain is the same as in RP, the authenticator will recognize it and start the process. In the background, the attacker starts a session with the real RP and forwards the requests, including the challenge, to the phishing website. After the victim's authenticator has signed the challenge, it is forwarded back to the attacker, allowing login access. To remove any suspicions, the phishing server shows an error and recommends reloading the page, so the victim will be forwarded to the real RP. This requires the attacker to turn off DNS spoofing, after the successful login. The setup of this attack is easy to realize but access to the target's device is required, making the attack medium to hard.

E. Registering Attacker's Passkey

Similar to the victim impersonation attacks, an attacker can try registering their own passkey, on behalf of the victim. One way would be to hijack the process of adding passkeys on the RP. To motivate the victim, a fake email can be sent, which recommends that the user add passkeys. When the victim now tries to add passkeys, the attacker has to sit in between the authenticator and the client, thus needing an AITM. Since the goal is to register the keys on the real RP, a simple AITM over the network and DNS spoofing is not sufficient. The AITM has to be run locally, on the victim's device, which again requires malware. When a request to add passkeys is observed, the challenge is signed with the attacker's own private key, and the victim's public key is replaced. This attack is not too stealthy, since the whole FIDO2 operation fails on the victim's side. To optimize this, the attacker can make it look as if the operation was successful by forwarding and modifying packets, marking the authentication as successful from the RP's side. Obviously, the victim won't be able to login with their newly created passkey, due to the public key not matching. The described method will be ranked as hard, because malware has to be installed. The second option for registering own passkeys is to login as the user via traditional methods like email and password. For this, a phishing website is used which captures the login data. After the victim enters their credentials, the attacker logs in and registers attacker-controlled keys. This attack is easy to realize and will also be ranked in this category. This attack succeeds only if the victim uses email and password authentication. If a passkey is used as a second factor, this attack fails, because the phishing domain is not matching. Here, a phishing website as proposed in Section IV-D, can be used to hijack the CTAP communication. This attack focuses on the attacker registering their own keys – the phishing website is used as a tool to gain initial access. In case of password change, the key remains valid. Unlike conventional phishing attacks, which lose effectiveness once the victim changes their password, this attack establishes persistent access through a registered passkey that remains valid independently of any subsequent credential changes.

F. Passkey Reduction

Many websites [15] provide the option to register a passkey instead of typical password authentication. But password authentication is not gone and is still used. This enables a user to authenticate via multiple methods. Since one of the main goals was to gain access to the account, an attacker can utilize a passkey reduction attack. First, the victim is lured to a phishing page and goes to the login page. The passkey option will either not be shown or upon choosing *passkey authentication*, the fake RP signals an error and prompts the victim for password authentication. This attack depends on the user believing that passkey authentication is currently unavailable. The setup, on the other hand, is easy to realize and thus the attack is ranked in the easy category. When passkeys are configured as a second factor in MFA, the attacker has to spoof the real RP's domain to sign the challenge as described in Section IV-D.

G. Infected Authenticator

The final attack on passkeys, where the goal is to gain access to the victim's account, uses an *infected authenticator*. The attacker modifies the underlying source code of an authenticator, making it generate known key pairs. When a user creates accounts on the RP, the attacker will be able to login, since the private key is known. Either the application's source code or the used cryptographic libraries, which are used to generate the keys, can be modified. Manipulating system libraries can be more challenging because it typically requires superuser permissions. When a library is modified, all applications using the library will generate known keys. The complexity in this attack lies in the installation of malware (infected authenticator) or malware which modifies the target library or application. This

attack is limited to local authenticators, meaning the key pairs must be generated locally on the victim's device. This attack path is ranked medium to hard, because of malware installation but the easy realisation. The complexity increases even more when system libraries get modified. However, implementing malicious code into applications is straightforward. Alternative approaches include exfiltrating generated keys over the network. A downside is that such behaviour can be detected by anti-malware systems [16]; e.g., an anti-malware system can detect and flag the use of sockets, because it can associate it to network activity. Other malware-reliant attacks described in this section, such as local AITM or certificate store manipulation, do not require outbound network connections to attacker-controlled infrastructure and are therefore less likely to trigger such detection mechanisms.

H. RP Impersonation

In the final attack scenario, an attempt is made to extract personal information from the victim by causing the victim to believe that authentication is being performed with a legitimate RP. To achieve this, an attacker must spoof both the RP's website and the CTAP interaction. As a result, the victim is led to believe that a successful login has occurred, even though the entire process is a visual deception. This attack requires the attacker to replicate the genuine user interface of a logged in RP session. Further, the FIDO2-related library on the victim's device must be modified so that the authentication process appears successful regardless of the actual input. Once the victim is "logged in", the attacker can display prompts requesting personal information under the guise of security checks. Of course, there are numerous strategies of how information can be extracted. A simpler variant is to force the victim to bypass passkey authentication entirely and directly request personal information, similar to Section IV-F. This variant only requires a successful phishing attack, making it relatively easy to execute. Note that this simpler variant does not rely on FIDO2 at all and would be equally effective against any authentication method; it is included here because it represents a degraded form of the full RP impersonation attack, which requires the installation of malware and the manipulation of RP libraries, making it significantly more complex and thus ranked hard.

Table I provides an overview of the attacks characteristics, which will be analysed in the next section. In the first row of the table, IE is used as an abbreviation for *Infected Authenticator* and AD for *Authenticator Deception*.

TABLE I. OVERVIEW OF THE ATTACKS

Criterion	IE	AD
Stealthiness	High	Medium
Feasibility	Medium	High
Victim Interaction	Low	High
Time Consumption	Low	Low-High
Privileges	Low-High	Low-High

V. EXPERIMENTS

Two experiments were conducted to demonstrate: first, that executing an attack on FIDO2 is highly resource-intensive, and second, that the attacks closely resemble phishing methods. Since passkeys cannot be phished directly, Section V-A describes an attacker being able to possess the same passkey. Similarly, Section V-B mirrors a typical phishing scenario, where a legitimate website is displayed, but the attacker's goal is to steal the proof of authentication.

A. Infected Authenticator

The first experiment follows the method described in Section IV-G. In this attack, KeePassXC, a local authenticator, is used, and only the source code of the binary is modified; no changes are made to the system libraries responsible for FIDO2. As previously mentioned, the source code of the authenticator is altered so that an attacker can obtain the same key that the victim generates during the registration process. KeePassXC relies on the Botan cryptographic library, which is used to generate asymmetric keys. Specifically, KeePassXC requires the generation of Elliptic Curve Digital Signature Algorithm (ECDSA), Rivest–Shamir–Adleman (RSA), and Edwards-curve 25519 Digital Signature Algorithm (Ed25519) keys.

The attack proceeds as follows: the attacker first generates their own public-private key pairs using Botan, which are saved in PEM format for later import into the KeePassXC database. This key is then embedded directly into KeePassXC's source code as a string variable. In the modified code, the private key in PEM format is loaded into the appropriate object type, while the rest of the code remains unchanged. This ensures that every time a new passkey is registered, the same private key is used, leading to the same public key being generated by Botan.

Depending on the RP, the public key (used to verify the challenge) is associated with a unique credential-ID. KeePassXC generates these credential IDs, which are then stored on the RP. Therefore, the attacker must also know the correct credential-ID, which is also easily embedded into the source code. The binary is then placed on the victim's computer, essentially replacing the old KeePassXC binary. After the victim registers their passkey on the RP, the attacker can authenticate and log in using the previously generated key pair. This experiment was conducted on a Firefox browser, specifically for logging into a Google account.

B. Authenticator Deception

The second experiment realizes the Authenticator Deception attack, which was described in Section IV-D. The attack setup was as follows. The victim was using a Windows 10 machine, while the attacker operated from a Kali Linux system. The victim had a registered test account on the target domain: *linear.app* – in general, any domain can be targeted, provided it implements FIDO2. The attacker also possessed a valid account on linear.app, which was required in order to modify authentication requests dynamically during the attack.

First, the attacker generated a valid TLS certificate for the fraudulent frontend server that would later be presented to the victim as a result of DNS spoofing [8]. Since modern browsers do not trust self-signed certificates by default, the attacker requires prior access to the victim's machine. Specifically, the attacker created a custom Certificate Authority (CA) and signed the frontend certificate with this CA, allowing the certificate to be valid. The custom CA certificate was then imported into the victim's browser (Firefox) trust store, effectively forcing the browser to trust the attacker's frontend certificate. Additionally, the frontend certificate contained a Subject Alternative Name (SAN) extension as well as a Common Name (CN) matching the target domain, in this case linear.app. Only after these preparatory steps was the victim's machine considered compromised and the actual attack could begin.

For the attacker-controlled infrastructure, an NGINX web server was used as the frontend, combined with an Express-based Node.js application acting as the backend. This backend handled incoming requests from the victim's browser and proxied or modified them as necessary before forwarding them to the legitimate RP.

The attacker initiated an ARP spoofing attack [8], thereby poisoning the victim's ARP table and replacing the legitimate router's MAC address with the attacker's own. In parallel, a DNS spoofing attack was launched that mapped all DNS requests for linear.app and *.linear.app to the attacker's local IP address. As a result, all traffic intended for linear.app was redirected to the attacker-controlled frontend server.

When the victim navigated to linear.app using the compromised browser, the request was routed to the attacker's NGINX frontend ①. The cloned website provided the option to authenticate using passkeys, which the victim selected. Upon initiating passkey authentication, a challenge was required. To obtain a valid challenge, the attacker simultaneously initiated an authentication attempt with the legitimate linear.app RP ②. Using Burp Suite as an AITM proxy, the attacker intercepted the authentication request and extracted both the challenge and the associated authenticationId ③. This identifier is a temporary value that uniquely binds a specific authentication session and is required to complete the passkey authentication flow.

The attacker's backend then forwarded the captured challenge to the cloned frontend, prompting the victim to sign it using their passkey ④ . In this setup, the authenticator software used by the victim was Bitwarden [17]. Once the victim approved the request ⑤, the signed challenge was produced and returned to the attacker-controlled backend ⑥.

At this stage, the attacker authenticated to the legitimate relying party using their own passkey ⑦. Before completing the authentication, the attacker manually modified the final request by replacing their own signed challenge and user identifier with those obtained from the victim. By submitting the victim's signed challenge together with the victim's user ID, the attacker was able to successfully authenticate as the victim on the real linear.app service.

All steps described above were performed manually. The attacker manually transferred the challenge, initiated authentication flows, and edited the final authentication request.

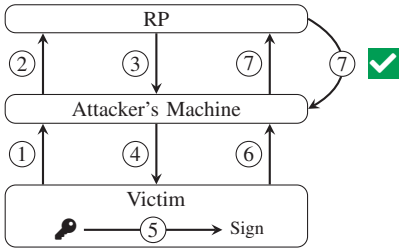

Figure 2. Authenticator Deception Flow

VI. Results and Discussion

Both experiments succeeded as described; however, several important nuances were observed.

For the authenticator deception attack, it was important that the DNS cache on the victim's machine was flushed. If stale DNS entries were present, the victim's browser would continue to resolve the domain to the legitimate RP, bypassing the attacker-controlled server. Since compromising or gaining access to the victim's machine was already a prerequisite for the attack, the attacker was able to flush the DNS cache manually. In a realistic malware scenario, this step is straightforward to automate, as flushing the DNS cache requires only a simple system command that typically does not require elevated privileges. This ensured that subsequent DNS queries were forwarded to the DNS server where the attacker was positioned as an AITM and could respond with their own IP address, pointing to the malicious frontend server. Clearly, in a real-world attack scenario, such an attack would not be executed in such a manual fashion. Instead, automation tools would be employed, such as Selenium for browser automation and a programmable AITM proxy capable of modifying packets dynamically. For demonstration purposes, we did not create a visual clone of the legitimate website; rather, the malicious frontend only provided an option to authenticate via passkeys. In a realistic attack scenario, however, the website would need to be visually indistinguishable from the legitimate RP's website to avoid raising suspicion on the victim's side. Finally, it is important to emphasize that this attack is not trivial to carry out. A critical requirement is the compromise of the victim's machine, which either involves tricking the user into installing malware that modifies the browser's trust store, flushes the DNS cache, and performs related actions, or requires physical access to the system in order to apply these modifications manually. Additionally, this attack is highly targeted at a specific individual, as the attacker must know in advance which website to clone, be located within the same network as the victim and successfully perform an ARP spoofing attack [8] against the target. The attack succeeds only if all these conditions are met. Moreover, the attack will fail if either the router or the victim's machine implements anti-ARP-spoofing mechanisms, which would effectively prevent DNS spoofing.

The infected authenticator attack targeted a specific local authenticator, namely KeePassXC. This attack does not apply to cloud-based authenticators such as Bitwarden. Consequently, the victim was required to use an authenticator that generates cryptographic keys locally. Under these conditions, malware can hijack the key-generation process and deliberately produce attacker-controlled or predictable keys. The described method demonstrated that previously generated keys were hardcoded as strings directly into the source code of the authenticator. This approach causes the authenticator to always generate the same keys and the same credential-ID during passkey registration. As a result, a victim using an infected authenticator would only be able to generate a single passkey per RP. A more effective approach from an attacker's perspective would be to use a predictable algorithm that generates keys and credential-IDs following a known pattern, or to rely on a predictable seed value from which keys are derived. Alternatively, multiple keys and credential-IDs could be embedded into the program, although this approach remains limited. Another option discussed would be to leave the authenticator's behavior unchanged while sharing the generated keys and credential-IDs over the network to the attacker. This would allow the attacker to obtain valid credentials without immediately breaking the expected functionality of the authenticator.

The demonstrated attacks showed that executing an attack on FIDO2 is time-consuming and therefore resource-intensive.

VII. Conclusion and Future Work

While passwords remain a significant security weakness in modern systems, this work demonstrates that attacking FIDO2-based authentication is far from trivial. Our experiments and analysis show that successfully compromising such systems is complex and time-consuming. In addition, multiple strict prerequisites must be fulfilled, such as physical proximity to the victim or prior compromise of the victim's device. These constraints significantly raise the bar for attackers and result in a threat model that is considerably more robust than traditional password-based authentication.

This work analysed possible attack vectors against FIDO2 authentication and demonstrated that FIDO2 largely fulfills its security promises. In particular, passkeys cannot be phished using conventional techniques since the private key never leaves the authenticator. As a result, many well-known attacks that are effective against password-based systems are rendered ineffective.

Future work includes the automation of the authenticator deception attack, as well as research into attack techniques that could target FIDO2 authentication without relying on device compromise or requiring an attacker to be within a specific physical range of the victim.

Acknowledgements

This research was conducted by Alexander Berladskyy as part of a Computer Science Research Project at Kiel University of Applied Sciences.

References

[1] APWG, *Phishing activity trends report*, APWG Report Q3 2025, Activity: July-September 2025, Published: 9 December 2025, Dec. 2025.

[2] C. Brand et al., 'Client to authenticator protocol (CTAP)', Jan. 2019. [Online]. Available: https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html.

[3] N. Bindel, C. Cremers and M. Zhao, *FIDO2, CTAP 2.1, and WebAuthn 2: Provable security and post-quantum instantiation*, Cryptology ePrint Archive, Paper 2022/1029, 2022. [Online]. Available: https://eprint.iacr.org/2022/1029.

[4] Yubico, *Yubico: Security keys and authentication solutions*, https://www.yubico.com/, Accessed: 2026-03-14, 2026.

[5] FIDO Alliance, *Fido specifications*, https://fidoalliance.org/specifications/, Accessed: 2026-03-14, 2026.

[6] J. Hodges et al., 'Web authentication: An API for accessing public key credentials level 2', Apr. 2021. [Online]. Available: https://www.w3.org/TR/webauthn-2/.

[7] Mozilla Developer Network, *Web authentication api (webauthn)*, https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API, Accessed: 2026-03-14, 2026.

[8] P. R. Babu, D. L. Bhaskari and C. Satyanarayana, 'A comprehensive analysis of spoofing', *International Journal of Advanced Computer Science and Applications*, vol. 1, no. 6, 2010.

[9] H. Li, X. Pan, X. Wang, H. Feng and C. Shi, 'Authenticator rebinding attack of the UAF protocol on mobile devices', *Wirel. Commun. Mob. Comput.*, vol. 2020, pp. 1–14, Sep. 2020.

[10] M. Barbosa, A. Cirne and L. Esquível, 'Rogue key and impersonation attacks on fido2: From theory to practice', in *Proceedings of the 18th International Conference on Availability, Reliability and Security*, ser. ARES '23, Benevento, Italy: Association for Computing Machinery, 2023, ISBN: 9798400707728. DOI: 10.1145/3600160.3600174. [Online]. Available: https://doi.org/10.1145/3600160.3600174.

[11] T. K. Yadav and K. Seamons, *A security and usability analysis of local attacks against fido2*, 2023. arXiv: 2308.02973 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2308.02973.

[12] A. T. Mahdad, M. Jubur and N. Saxena, 'Breaching security keys without root: Fido2 deception attacks via overlays exploiting limited display authenticators', in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24, Salt Lake City, UT, USA: Association for Computing Machinery, 2024, pp. 1686–1700, ISBN: 9798400706363. DOI: 10.1145/3658644.3690286. [Online]. Available: https://doi.org/10.1145/3658644.3690286.

[13] D. Kim, J. Shin, G. Ryu and D. Choi, 'Hipass: Hijacking ctap in passkey authentication', *IEEE Access*, vol. 13, pp. 92 086–92 101, 2025. DOI: 10.1109/ACCESS.2025.3570377.

[14] KeepassXC Team, *Keepassxc: Cross-platform password manager*, https://keepassxc.org/, Accessed: 2026-03-14, 2026.

[15] Passkeys.io, *Who supports passkeys*, https://www.passkeys.io/who-supports-passkeys, Accessed: 2026-03-14, 2026.

[16] P. Szor, *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, 2005, ISBN: 0321304543.

[17] Bitwarden, *Bitwarden password manager*, https://bitwarden.com, Accessed: 2026-03-14, 2026.