CTRAPS: CTAP Client Impersonation and API Confusion on FIDO2

Marco Casagrande
EURECOM
marco.casagrande@eurecom.fr

Daniele Antonioli
EURECOM
daniele.antonioli@eurecom.fr

Abstract

FIDO2 is the standard technology for single-factor and second-factor authentication. It is specified in an open standard, including the WebAuthn and CTAP application layer protocols. We focus on CTAP, which allows FIDO2 clients and hardware authenticators to communicate. No prior work has explored the CTAP Authenticator API, a critical protocol-level attack surface that deals with credential creation, deletion, and management. We address this gap by presenting the first security and privacy evaluation of the CTAP Authenticator API. We uncover two classes of protocol-level attacks on CTAP that we call CTRAPS. The client impersonation (CI) attacks exploit the lack of client authentication to tamper with FIDO2 authenticators. They include zero-click attacks capable of deleting FIDO2 credentials, including passkeys, without user interaction. The API confusion (AC) attacks abuse the lack of protocol API enforcements and confound FIDO2 authenticators, clients, and unaware users into calling unwanted CTAP APIs while thinking they are calling legitimate ones. For example, if a victim thinks he is authenticating to a website, they are deleting their credentials. The presented eleven attacks are conducted either in proximity or remotely and are effective regardless of the underlying CTAP transport (USB, NFC, or BLE). We detail the eight vulnerabilities in the CTAP specification, enabling the CTRAPS attacks. Six are novel and include unauthenticated CTAP clients and trackable FIDO2 credentials. We release CTRAPS, an original toolkit, to analyze CTAP and conduct the CTRAPS attacks. We confirm the attacks' practicality on a large scale by exploiting six popular authenticators, including a FIPS-certified one from Yubico, Feitian, SoloKeys, and Google, and ten widely used relying parties, such as Microsoft, Apple, GitHub, and Facebook. We present eight practical and backward-compliant countermeasures to fix the attacks and their root causes. We responsibly disclosed our findings to the FIDO alliance and the affected vendors.

1 Introduction

Fast IDentity Online v2 (FIDO2) is the de-facto standard for single-factor (passwordless) and second-factor (2FA) authentication. Google, Dropbox, and GitHub [44] designed FIDO to offer a practical and scalable solution for authentication. FIDO has been widely adopted by industries and organizations, including Google, Microsoft, and the US government [29]. Market forecasts predict the FIDO market to rapidly grow from USD 230.6 million in 2022 to USD 598.6 million in 2031 [60]. Yubico, a FIDO authenticator market leader, sold more than 22 million YubiKey authenticators [66]. This growth will continue because of the recent industry-wide push towards single-factor passkey-based authentication [21, 30, 56].

FIDO2 involves three entities: an *authenticator* that generates and asserts possession of authentication credentials (e.g., public-private key pairs), a *relying party* that authenticates the user (e.g., challenge-response protocol based on credentials), and a *client* who wants to authenticate to the relying party and manages the communication between the authenticator and the relying party. Typically, the authenticator is a dongle, the relying party is a web server, and the client is a web browser or a mobile app.

The authenticator and the client communicate using the *Client to Authenticator Protocol (CTAP)*. CTAP works at the application-layer and is transported over Universal Serial Bus (USB), Near Field Communication (NFC), or Bluetooth Low Energy (BLE). CTAP exposes to the client the *CTAP Authenticator API*, usable to interact with the authenticator, e.g., credential creation, management, and deletion. These API calls might require User Verification (*UV*) and User Presence (*UP*) authorization. The latest CTAP protocol specification is the version 2.2 [3].

This work focuses on the CTAP protocol and its security and privacy guarantees. There are only a few research studies about CTAP. The authors of [10] performed a provable security analysis on CTAP, highlighting unauthenticated DH key exchange. In a follow-up work [11], they proposed an imper-

sonation attack exploiting CTAP to register an authenticator with an arbitrary relying party. The authors in [34] propose a MitM privacy leak attack on CTAP based on unauthenticated DH. Other works target the authenticator with fault injection and side channel physical attacks [43, 55]. The literature on WebAuthn is extensive, featuring, among others, misbinding, misauthentication, session hijacking, cookie theft, social engineering, and rogue authenticator attacks [38, 42, 46, 50, 65] (see Section 10 for more FIDO2 related work).

No prior work investigated the *CTAP Authenticator API*. This API is a critical protocol-level attack surface as it enables the creation, management, and deletion of FIDO2 credentials and the administration of FIDO2 authenticators. FIDO2 credentials are essential for security and privacy as they authorize access to sensitive online services, including social media, banking, data sharing, e-commerce, etc. A protocol-level attack on the CTAP Authenticator API would enable access and tamper with any FIDO2 credential stored on any FIDO2 authenticator, regardless of the authenticator's hardware and software details. Hence, it is crucial to assess the API's expected security and privacy properties and whether they hold them in practice.

We fill this gap by presenting the first security and privacy assessment of the CTAP Authenticator APIs. We uncover *two classes of protocol-level attacks on CTAP* we call **CTRAPS**. The *client impersonation (CI)* attacks exploit the lack of client authentication to tamper with a victim authenticator. Some of the attacks are zero-click (i.e., not require user interaction), while others are one-click (i.e., require expected user interaction). The *API confusion (AC)* attacks abuse the lack of protocol API enforcements and confound a FIDO2 authenticator, a client, and an unaware user into calling unwanted CTAP Authenticator APIs while thinking they are calling legitimate ones (e.g., the user thinks he is authenticating to a website but he is instead deleting his credentials). In total, we present five CI and seven AC attacks.

The attacks are conducted in *proximity* (malicious FIDO2 device close to the victim) or *remotely* (malicious app installed on the victim's phone). The AC attacks require a MitM position, while the CI attacks target the authenticator. Unlike prior work, they do *not* require physical access to the authenticator, a compromised client, or side channel and fault injection [43, 47]. Moreover, the attacks are *stealthy* because they employ CTAP-compliant API calls and do not require unexpected user interactions (unlike phishing [61] or other deception techniques [50]).

The CTRAPS attacks have a *critical* and *widespread* impact on the FIDO2 ecosystem. They are critical as they enable the violate security, privacy, and availability of FIDO2 devices. For example, a CI or an AC attacker can (remotely) factory reset an authenticator, deleting all FIDO2 credentials and locking out the victim from the related service. Moreover, despite targeting CTAP, the attacks have an impact on FIDO2 relying parties, e.g., they can delete non-discoverable

credentials stored on the relying party using WebAuthn. They are widespread as they exploit protocol-level vulnerabilities in the CTAP application-layer protocol. Hence, they can be conducted against any FIDO2 device regardless of the CTAP transport (USB, NFC, and BLE).

The isolate *eight vulnerabilities in the CTAP specification* enables the CTRAPS attacks. Six of them are novel within FIDO2 and include unauthenticated CTAP clients, trackable FIDO2 credentials, and weak authorization (of destructive API calls). The vulnerabilities are *severe* as they affect any FIDO2 authenticator and client implementing any CTAP version, including the latest CTAP2.2 draft. They also indirectly affect FIDO2 relying parties, as we will explain later. Moreover, we find a *implementation flaw* on Yubico authenticator firmware, allowing the leak of sensitive data and user tracking (CVE-2024-35311). We disclosed it to Yubico, who fixed it.

We present CTRAPS, a new toolkit to experiment with CTAP and conduct the CTRAPS attacks. The toolkit has three modules: CTAP testbed, malicious CTAP clients, and Wireshark dissectors. The testbed allows virtual clients and relying parties to test a FIDO2 authenticator locally and safely over CTAP. The malicious clients include CI and AC proximity and remote attack implementations that can be tested on real-world devices. For example, it ships an Android app and Proxmark3 scripts to test the CI attacks over NFC. The dissectors module includes an extended FIDO2 dissector for Wireshark, adding new and valuable packet information such as status codes and support for credential management.

We demonstrate the practicality of the attacks by successfully evaluating them on popular FIDO2 authenticators, clients, and relying parties and conducting them over different CTAP transports (USB, NFC). We attack *six authenticators* from Yubico (including a FIPS-compliant one), Feitian, SoloKeys, and Google. We conducted the attacks over USB and NFC. We also exploit *ten relying parties* offering passkeys and second-factor authentication, including Microsoft, Apple, GitHub, and Facebook.

To fix the CTRAPS attacks and their flaws, we design eight countermeasures that are backward-compliant with the CTAP standard. The fixes include CTAP client authentication, stricter authorization requirements for destructive APIs, introduce a dedicated PIN for destructive operations (e.g., credential deletion), and rotate user identifiers and credentials to mitigate user tracking. They are practical as they rely on mechanisms already available on the authenticator (e.g., PIN and LED) and do not require adding extra hardware (e.g., an extra display).

We summarize our contributions as follows:

- We perform the first assessment of the CTAP Authenticator API. We unveil two classes of CTAP protocol-level attacks: CI and AC. The attacks compromise the security, privacy, and availability of the FIDO2 ecosystem. For instance, they (remotely) delete FIDO2 credentials,

track users via FIDO2 credentials, and DoS authenticators. They are enabled by eight CTAP protocol level vulnerabilities, six of which are new.

- We provide a toolkit to evaluate the CTAP Authenticator API surface and test our attacks locally in a virtual environment or on actual devices. We successfully evaluate our attacks against popular FIDO2 devices. We exploit six authenticators, two transports, and ten relying parties. The affected vendors include key FIDO2 players like Google, Apple, Microsoft, and Yubico.

- We fix the attacks and their root causes by proposing eight practical and backward-compliant countermeasures. We responsibly disclosed our findings to the FIDO2 Alliance and affected vendors.

Responsible Disclosure We responsibly disclosed our findings to the FIDO Alliance in November 2023 [7]. They acknowledged our report, provided feedback in May 2024, and shared it with their members. In December 2023, we reported our findings to the affected authenticator manufacturers (i.e., Yubico, Feitian, SoloKeys, and Google). Google assigned priority P2 and severity S2 to our report. Yubico acknowledged the implementation bug we found, pushed a fix in production, published a security advisory [70], and created CVE-2024-35311 [69]. The other manufacturers acknowledged the report without commenting on it. We also contacted Apple and Microsoft regarding their weak credential protection policy that facilitates user tracking and profiling. They responded that our report has no security implications for their products.

Ethics and Availability We conducted our experiments ethically. We evaluated our authenticators and accounts. We did not collect personal data or involve third parties. We will open source our contributions, including the `CTRAPS` toolkit, after responsible disclosure with the FIDO Alliance, the manufacturers, and the relying parties.

2 Background and System Model

We introduce the FIDO2 standard, its underlying Client To Authenticator Protocol (CTAP), and our system model.

2.1 FIDO2

FIDO2 [4] is an open standard for user authentication based on *asymmetric* cryptography and curated by the FIDO Alliance. Four entities compose the FIDO2 ecosystem: an authenticator, a client, a user, and a relying party. In a typical scenario, a user connects his authenticator to the client to access an online service hosted by a relying party.

The FIDO2 specification includes the WebAuthn and CTAP application-layer protocols. WebAuthn provides a secure and private communication channel between a relying party and a client, and its latest version is WebAuthnL2 [62]. CTAP, the

focus of this work, enables a secure and private connection between an authenticator and a client via the CTAP Authenticator API. For example, calling `MakeCred` registers a new credential, and `GetAssertion` authenticates an existing one.

A FIDO2 *credential* is a key pair used to sign and verify challenges by applying standard cryptographic techniques, such as the Elliptic Curve Digital Signature Algorithm (ECDSA). Access to the private key of a FIDO2 credential is safeguarded by encryption using a credential master key unique to each authenticator and securely stored within the authenticator's Secure Element. FIDO2 credentials can be discoverable or non-discoverable. Discoverable credentials, also known as passkeys, are *stored on the authenticator* and used for passwordless authentication. Non-discoverable credentials are stored on the web by the relying party and used for multi-factor authentication.

A FIDO2 credential is bound to three identifiers: the credential identifier (CredId), the relying party identifier (RpId), and the user identifier (UserId). CredId is derived from the credential master key and uniquely identifies a credential. Before deleting a credential, the client needs to specify a CredId. The RpId identifies a relying party, usually coincides with its origin (e.g., `login.microsoft.com`), and should be considered public. A relying party randomly generates a UserId when a user creates his first credential and associates the UserId to all credentials generated by that user. At registration time, a relying party can attach additional data to a credential, including sensitive or personally identifying information, using the optional *CredBlob* FIDO2 extension.

2.2 CTAP

As part of the FIDO2 standard, CTAP has considerably evolved over time. CTAP1, also known as FIDO U2F (Universal 2nd Factor), provides phishing-resistant 2FA. CTAP2.0 maintains backward compatibility with CTAP1 while introducing passwordless authentication. CTAP2.1 [1] adds the credential protection policy, discoverable credential management (i.e., the `CredMgmt` API), and biometric authentication. The draft for CTAP2.2 [3] is the latest available version, offering new features such as support for hybrid authenticators equipped with cameras to scan QR codes.

CTAP relies on two core user authorization mechanisms to secure API calls: (i) *User Verification (UV)*, which requires the user to enter a PIN or biometric data, and (ii) *User Presence (UP)*, which requires the user to press a button on the authenticator or to bring it into the client's NFC range. Table 1 shows the most common *CTAP Authenticator APIs* and their *UV* and *UP* requirements. We describe each API:

- **MC:** `MakeCred` registers a new credential bound to an online account with a relying party.

- **GA:** `GetAssertion` authenticates to a relying party by proving possession of a credential.

Table 1: CTAP Authenticator API entries, short names (SN), *UV* and *UP* authorization requirements and support for sub-commands. Yes[1]: depends on the client and relying party configuration, Yes[2]: depends on API subcommand. In the CTAP standard, `MakeCred` is called `MakeCredential` and `CredMgmt` is called `CredentialManagement`.

CTAP API	SN	UV	UP	Subcmd
MakeCred	MC	Yes	Yes	No
GetAssertion	GA	Yes[1]	Yes[1]	Yes
CredMgmt	CM	Yes	No	Yes
ClientPin	CP	Yes[2]	No	Yes
Reset	Re	No	Yes	No
Selection	Se	No	Yes	No
GetInfo	GI	No	No	No

CM: `CredentialMgmt` manages the authenticator's discoverable credentials (e.g., enumerate, modify, and delete).

CP: `ClientPin` handles *UV* based on a user PIN to be submitted via the client's UI.

Re: `Reset` factory resets the authenticator (i.e., wipes all discoverable and non-discoverable credentials by re-generating the credential master key).

Se: `Selection` selects an authenticator to operate among the available ones.

GI: `GetInfo` returns the authenticator's details (e.g., manu-facturer, transports, extensions, and settings).

CTAP offers other optional security and privacy mechanisms. The authorization requirements for `GetAssertion` depend on the client and relying party configuration. A client can specify the option *up=false* to skip *UP*. At registration time, a relying party can enforce access control by specifying a credential protection policy via the optional *CredProtect* extension. However, the default policy skips *UV*, weakening privacy protection. Authenticators may also feature additional security mechanisms unrelated to CTAP, such as the FIDO authenticator certification level [5] and the FIPS [51] certification.

The `GetAssertion`, `CredMgmt`, and `ClientPin` APIs offer multiple functionalities through API subcommands. For example, `CredMgmt(GetCredsData)` returns the amount of stored discoverable credentials and `CredMgmt(DelCreds)` deletes all discoverable credentials. Compared to their original API, some API subcommands have more relaxed requirements. For example, `ClientPin(KeyAgreement)` requests the authenticator's public key without requiring *UV*.
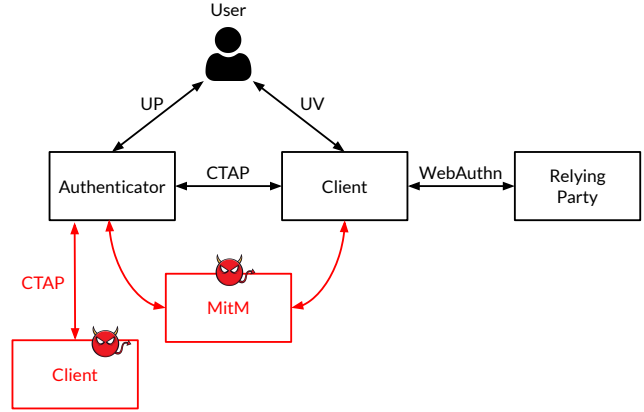

Figure 1: **CTRAPS threat model.** The user authenticates to the relying party using a client (e.g., browser) and an authenticator (hardware dongle). The user, when needed, grants UP by pressing a button on the authenticator and UV by submitting a PIN to the client. We study two attacker models: (i) a client impersonation attacker targeting the authenticator over CTAP (left), (ii) a MitM attacker in the CTAP channel between the authenticator and the client.

2.3 System Model

Figure 1 shows the standard FIDO2 system model [4], composed by an authenticator, a client, and a relying party. The user connects the authenticator to the client to authenticate on a service hosted by the relying party. The entities support up to CTAP2.2 and WebAuthnL2 (i.e., the latest and supposedly most secure FIDO2 protocols). We describe each entity in detail.

Authenticator The authenticator is a FIDO2 *roaming* authenticator: a physical device carried around by the user that can be connected to the client (e.g., a USB/NFC dongle). The authenticator runs a CTAP server that exposes the CTAP Authenticator API. The API is accessible over USB, NFC, and BLE, which are the standard CTAP transports. The authenticator supports FIDO2's *UP* (e.g., via a button press) and *UV* (e.g., via a user PIN) user authorization mechanisms. It stores discoverable credentials and the credential master key.

Client The client is a FIDO2 client handling the communication between the authenticator and the relying party. It exposes a CTAP client to the authenticator and a WebAuthn client to the relying party. The client could be a web browser, a mobile app for Android [26] or iOS [27], or a command line tool like the Yubico CLI [68].

Relying party The relying party is an online service that relies on FIDO2 passwordless or multi-factor authentication. It runs a WebAuthn server that responds to FIDO2 registration and authentication requests over TLS from the client. The relying party stores non-discoverable credentials, and user and credential identifiers. Offline operations on the authenticator, like deleting discoverable credentials, indirectly affect the

relying party by making the user unable to log into their online service.

User The user owns an authenticator and a device running the client, e.g., a YubiKey and a laptop. He utilizes his authenticator to register FIDO2 credentials and authenticate to the associated relying party. To do so, he connects his authenticator to the client and provides *UV* and *UP*, if necessary. The user locally manages the authenticator via the client without connecting to a relying party. For example, he can check his discoverable credentials and change the authenticator's PIN.

3 CTRAPS Client Impersonation Attacks

3.1 Introduction and Motivation

We unveil the four CTRAPS **Client Impersonation (CI) attacks** exploiting the CTAP Authenticator API. The attacks factory reset the authenticator via the `Reset` API, track the user via `GetAssertion`, lock the authenticator via `ClientPin`, and profile the authenticator via `GetInfo`. They exploit five protocol-level vulnerabilities described later in Section 5.1 and the unrealistic FIDO reference threat model we discuss in Section 8.2.

The attacks advance the state of the art. Prior CI attacks required: (i) to trick the user to obtain authorization [38], (ii) a compromised CTAP client (e.g., malicious browser extension) [11], (iii) or physical access to the authenticator [65]. Our attacks instead are: (i) *zero-click*, as we bypass *UV* and *UP* authorizations which require user interaction; (ii) require *no client compromise* as they are conducted from an attacker-controlled device (e.g., an NFC reader); and (iii) require *no physical access* but rely on a proximity-based attacker conducting the attacks over NFC. Next, we introduce the CI attacker model and describe the attacks.

3.2 CI Attacker Model

The CI attacker model assumes an adversary impersonating a CTAP client to an authenticator, as shown in Figure 1. The attacker is in proximity (e.g., an NFC reader) or can connect to the authenticator remotely (e.g., a mobile app with Internet access). The attacker can send legitimate CTAP commands to the authenticator. She cannot modify the authenticator's firmware or compromise a legitimate FIDO2 client and relying party. She has no physical access to the authenticator.

The attacker goal is to compromise the authenticator *security*, *privacy*, and *availability* by exploiting the CTAP Authenticator API (introduced in Section 2). For example, she wants to tamper with discoverable credentials stored on the authenticator (security), track a user via the authenticator (privacy), or DoS the authenticator (availability).
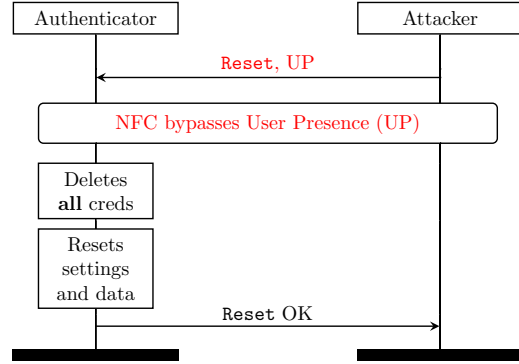

Figure 2: **CI$_1$ attack.** Factory reset authenticator via `Reset`. While in NFC range, the attacker calls the `Reset` API. Over NFC, the authenticator skips *UP* and instantly factory resets, deleting its discoverable and non-discoverable credentials.

3.3 CI Attacks Description

We describe the four CI attacks. We label them CI$_1$, CI$_2$, CI$_3$, and CI$_4$.

CI$_1$: Factory reset authenticator In CI$_1$, the attacker abuses the `Reset` API to factory reset an authenticator, as shown in Figure 2, deleting discoverable and non-discoverable credentials, PIN, user preferences, and stored data. The attacker connects to the authenticator and, despite not authenticating, issues a factory reset command (requiring *UP*). She bypasses the *UP* check as, according to the CTAP standard, connecting over NFC implies user presence. This novel trick results in a *zero-click* factory reset over NFC. Instead, CI$_1$ is a one-click factory reset when deployed over USB, as the *UP* bypass is only available to the NFC transport. A factory reset wipes out all credentials, as it erases the credential master key necessary for decryption. It also deletes the authenticator's settings, including the PIN, user preferences, and stored data. Then, the authenticator confirms the successful reset.

CI$_2$: Track user from credentials In CI$_2$, instead of using `GetAssertion` API for authentication, the attacker exploits it to leak identifying information and track the user, as shown in Figure 3. The attacker does not require *UV* as she only targets relying parties that register credentials using the weak and default *CredProtect=UVOptional* default policy, such as Microsoft and Apple. She does not require *UP* either, as her `GetAssertion` command contains the *up=false* option. As a result, she achieves a *zero-click* leak of all credential and user identifiers registered with specific relying parties. With the identifiers, she fingerprints the users and tracks them over multiple connections by performing CI$_2$ each time and looking for matching fingerprints. CI$_2$ is effective even on credentials protected by stronger credential protection policies (i.e., *CredProtect=UVRequired* and *CredProtect=UVOptionalWithCredIDList*), but requires *UV* or the knowledge of credential identifiers.
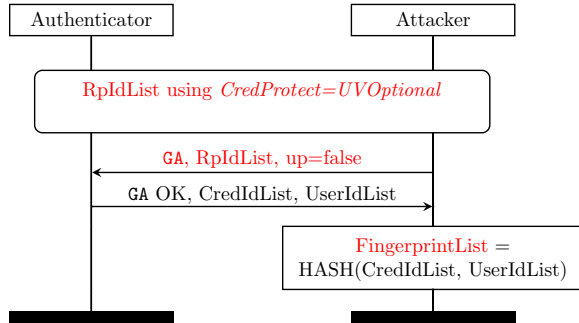
Figure 3: CI$_2$ attack. Track user from credentials via GetAssertion. The attacker connects to the authenticator and calls the GetAssertion API (GA in the figure). She skips *UV* by targeting relying parties using the weak and default *CredProtect* default policy and skips *UP* by passing *up=false*. The authenticator returns a list of credential and user identifiers used by the attacker to fingerprint the authenticator and track the user.

CI$_3$: Force authenticator lockout In CI$_3$, the attacker abuses the ClientPin, which protects the authenticator from PIN brute-forcing, to lock the authenticator or even force a factory reset. Through the GetPinToken subcommand, she submits several wrong PIN guesses in a row to the authenticator. After three wrong guesses, the device enters a soft lock mode preventing further actions until a reboot (i.e., leaving and re-entering a client's NFC range or detaching and re-attaching to a USB port). After a maximum number of failed PIN attempts (CTAP mandates eight), the authenticator enters a hard lock mode that is only restorable through a factory reset, which wipes out all credentials and can lead to account loss.

CI$_4$: Profile authenticator In CI$_4$, the attacker calls GetInfo to leak the authenticator details as a stepping stone to more advanced attacks, profile the user, and track him in future connections, and assess whether the authenticator is vulnerable to an implementation-specific attack like [69], The leaked details include the manufacturer, model, and FIDO2 version, and the supported algorithms, transports, options, and extensions. The authenticator also discloses user settings, such as FIDO2 being disabled over a specific transport.

4 CTRAPS API Confusion Attacks

4.1 Introduction and Motivation

We present the seven CTRAPS **API Confusion (AC) attacks** taking advantage of a novel attack technique we define as *API confusion*.

API confusion tricks a client, an authenticator, and their user into calling an unwanted CTAP Authenticator API. The unwanted API call must have the same or lower *UV* and *UP*

requirements of the confounded API. This technique is very effective as it does not require social engineering [61] or other deception techniques [50] to trick the user into calling a bad API. The user cannot detect ongoing API confusion because, unlike prior attacks, he only performs expected actions (e.g., does not grant *UV* if the API he calls does not require it). The attacks exploit protocol-level vulnerabilities we outline in Section 5.1 and the FIDO reference threat model in Section 8.2.

The AC attacks represent a new class of attacks that has not been explored. As shown in Table 7, prior work on CTAP eavesdropped on unencrypted CTAP traffic and exploited the unauthenticated Diffie-Hellman in a MitM attack but did not inspect the Authenticator API. Physical access and side channel attacks targeted the authenticator to leak its credential master key, while the remaining works put CTAP on the sidelines, targeting WebAuthn instead with authenticator rebinding and session hijacking. Next, we introduce our attacker model and describe the AC attacks.

4.2 AC Attacker Model

The AC attacker model assumes a man-in-the-middle (MitM) attacker between the authenticator and the client, as shown in Figure 1. The attacker is either in proximity to the authenticator (e.g., a NFC skimmer) or can contact the authenticator from remote (e.g., using a remotely controllable USB hub). She maintains stealthiness by not triggering unexpected behaviors. For example, she calls APIs when the user is operating the authenticator and does not require extra *UV* and *UP* authorizations. She cannot modify the authenticator's firmware or compromise a legitimate FIDO2 client and relying party. She has no physical access to the authenticator.

The attacker goal is to violate the authenticator *security*, *privacy*, and *availability* by exploiting the CTAP Authenticator API (introduced in Section 2). For example, she wants to leak and delete the discoverable credentials stored in the authenticator, including passkeys (security), track a user via the authenticator (privacy), or DoS the authenticator (availability).

4.3 AC Technique and Combinations

The seven AC attacks rely on the API confusion attack technique. The attacker intercepts a call to API A and changes (i.e., confounds) it to API B. The adversary only requires that API B has the same or lower *UV* and *UP* authorization requirements than API A. The AC technique has six steps:

1. The user calls API A through the client. The API might require *UV* and/or *UP*.

2. If required by API A, the attacker obtains *UV* by executing the CTAP PIN/UV authentication protocol v1 (via ClientPin). The user inputs the PIN on the client,

Table 2: There are 49 ways to perform AC against 7 CTAP Authenticator APIs. The user intends to call `API A`, instead is tricked into calling `API B`. $\checkmark^1$: proximity-based attacker, $\checkmark^2$: default *CredProtect=UVOptional* if credential protection is enabled, n/a: not applicable.

	CM	Re	GA	MC	CP	Se	GI
MC	✓	✓	✓	n/a	✓	✓	✓
GA	✓	✓	n/a	✓	✓	✓	✓
CM	n/a	$\checkmark^1$	✓	$\checkmark^1$	✓	✓	✓
CP	✓	$\checkmark^1$	✓	$\checkmark^1$	n/a	✓	✓
Re	n/a	n/a	$\checkmark^2$	n/a	✓	✓	✓
Se	n/a	✓	$\checkmark^2$	n/a	✓	n/a	✓
GI	n/a	$\checkmark^1$	$\checkmark^2$	✓	✓	✓	n/a
Total	3	6	6	4	6	6	6

which encrypts it and submits it to the authenticator. The authenticator responds with an encrypted User Verification Token (UVT) that will be attached to any API call requiring *UV*.

3. The attacker calls `API B` rather than `API A` based on the AC combinations in Table 2.

4. If required by `API A`, the attacker obtains *UP* from the user, who is unable to realize he is under attack. The attacker can only obtain *UP* once, as multiple requests would alarm the user. This step is skipped over NFC as proximity implies *UP*.

5. The authenticator executes `API B` and returns a success message.

6. The attacker informs the victim via the CTAP client that `API A` was successfully executed with compatible authorizations,

The AC strategy is effective on *7* CTAP Authenticator APIS and provides *49* ways to confound the victim as shown in Table 2. Multiple (`API A`, `API B`) pairs achieve the same goal. The amount of available pairs depends on their *UV* and *UP* requirements and, in the case of AC$_3$, also on the *CredProtect* policy. The first column lists seven APIs the user intends to call (`API A`), and the remaining columns represent the API called by the attacker (`API B`). For instance, AC$_1$ is available whenever the user calls `MakeCred`, `GetAssertion`, or `ClientPin`, confounding the call to `CredMgmt`. Some combinations are only feasible by a proximity-based attacker or under the default *CredProtect* policy. An API cannot be confounded with itself or APIs with incompatible authorization requirements.
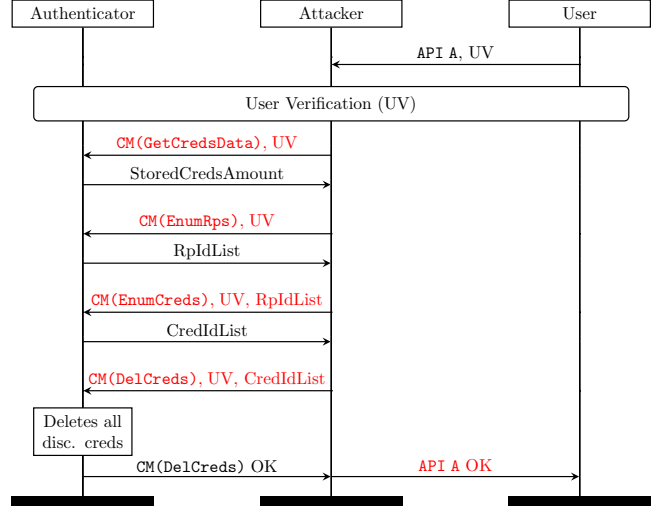

Figure 4: **AC$_1$ attack.** Delete discoverable credentials attack with proximity. The user intends to call `API A`, requiring *UV* but not necessarily *UP*. For example, `GetAssertion`, `ClientPin`, or `MakeCred`. The attacker obtains *UV* from the unsuspecting user. Instead of `API A`, she calls `CredMgmt` (`CM` in the figure). She executes four `CredMgmt` subcommands, which list and then delete all discoverable credentials on the authenticator.

4.4 AC Attacks Description

We describe the seven AC attacks. We label them AC$_1$, AC$_2$, AC$_3$, AC$_4$, AC$_5$, AC$_6$, and AC$_7$. The attacks are related to the second to last column of Table 2. AC$_1$ exploits all possible ways to call `CM`, AC$_2$ to call `Re`, and so on.

AC$_1$: Delete discoverable credentials In AC$_1$, the attacker abuses the `CredMgmt` API to delete the discoverable credentials on the authenticator, as shown in Figure 4. The user intends to call `API A`, which requires *UV* but not necessarily *UP*, such as `GetAssertion`, `ClientPin`, or `MakeCred`. Instead, the attacker executes four separate `CredMgmt` subcommands, none of which require *UP*. First, she checks the existence of discoverable credentials to erase (StoredCredsAmount) via `CredMgmt(GetCredsMetadata)`. Second, she retrieves the list of relying parties stored on the authenticator (RpIdList) via `CredMgmt(EnumRps)`. Third, she uses RpIdList to retrieve the list of stored credential identifiers (CredIdList) via `CredMgmt(EnumCreds)`. Fourth, she uses CredIdList to delete all discoverable credentials via `CredMgmt(DelCreds)`. Finally, she falsely returns `API A` OK to the user.

AC$_2$: Factory reset authenticator In AC$_2$, the attacker exploits the `Reset` API to factory reset the authenticator, similar to CI$_1$. Since `Reset` over USB requires *UP*, but not *UV*, an attacker can confound `MakeCred`, `GetAssertion`, and `Selection` into a `Reset` call. An attacker over NFC, able to bypass *UP*, can also confound `CredMgmt`, `ClientPin`, and `GetInfo`.

AC$_3$: Track user from credentials In AC$_3$, the attacker misuses the `GetAssertion` API to leak unique identifiers as fingerprints and track the user, similar to CI$_2$. She can confound `MakeCred`, *CredMgmt*, and *ClientPin* into a `GetAssertion` call, if she wants to access credentials protected by the *CredProtect=UVRequired* or *CredProtect=UVOptional WithCredIDList* policies. Additionally, the attacker can also confound `Reset`, `Selection`, and `GetInfo` if she only wants to access credentials protected by the weak *CredProtect=UVOptional* default policy.

AC$_4$: Fill authenticator credential storage In AC$_4$, the attacker repeatedly calls `MakeCred` to register new discoverable credentials, until the authenticator's credential storage is full. She exploits the *rk=true* option to enforce the generation of discoverable credentials over non-discoverable ones. A full storage compromises the authenticator's availability as the user cannot register new discoverable credentials.

AC$_5$: Force authenticator lockout In AC$_5$, the attacker abuses the `ClientPin` API to lock the authenticator and force a mandatory factory reset, similar to CI$_3$. Although `ClientPin` requires *UV*, the attacker wants to fail multiple PIN attempts (i.e., she does not need *UV*). Consequently, she can confound any API call into a `ClientPin` call, as she does not need authorization.

AC$_6$: Authenticator DoS In AC$_6$, the attacker calls the `Selection` API to trigger an unwanted *UP* check to keep the authenticator busy and to deny availability. Since the attacker can detect when the busy state ends (e.g., the user pressed the authenticator's button or 30 seconds have passed), she can prolong the attack indefinitely.

AC$_7$: Profile authenticator In AC$_7$, the attacker invokes the `GetInfo` API to retrieve the authenticator's details. Then, similar to CI$_4$, she uses this information as a stepping stone to other attacks, to track the user, or to check whether the authenticator is vulnerable to implementation-specific attacks [69]. Not requiring *UV* or *UP*, the attacker can confound any API call into a `GetInfo` call.

5 CTRAPS Vulnerabilities and Impact

5.1 Vulnerabilities

The CTRAPS attacks are enabled by *eight vulnerabilities* we discovered in the CTAP specification. Six of them are novel, while V2 was discussed in the misbinding and misauthentication attacks of [64]. Still, this is the first work exploiting V2 via AC. We describe them and how they map to the CI and AC attacks presented in Sections 3 and 4.

V1: Unauthenticated CTAP client The CTAP client does not authenticate to the user, the authenticator, or the relying party. FIDO2 clients (and, by extension, CTAP clients) have no identity, meaning that the authenticator cannot distinguish an official client developed by its manufacturer from a third-party client. The authenticator has no choice but to trust any

connecting client, including compromised ones.

V2: No authenticator feedback about API calls Despite having an LED, the authenticator does not provide the user with visual feedback when invoking APIs or granting *UV* and *UP*. The user cannot confirm whether the intended API has been called (or confounded) and which API utilized the granted *UV* and *UP* authorizations.

V3: NFC range provides *UP* Authenticators inside the NFC range of a FIDO2 client automatically obtain *UP* without the user pressing a button on the authenticator. Bypassing *UP* decreases the security protections of `MakeCredential`, `GetAssertion`, and `Reset` to only *UV* or no authorization requirement at all.

V4: Weak destructive APIs authorization Destructive API calls, such as credential deletion (`CredMgmt`) or authenticator factory reset (`Reset`), and non-destructive ones, like authentication (`GetAssertion`) are authorized by the same *UV* PIN. For example, the user intends to authenticate (non-destructive) and provides *UV* and *UP*, instead the attacker factory resets the authenticator (destructive).

V5: User trackable via CredId and UserId Discoverable credentials contain *static* and *unique* CredId and UserId, exploitable to reliably track users. These values can be obtained without *UV* or *UP* via the `GetAssertion` API. We note that the more credentials are stored in the authenticator, the more this vulnerability is effective as each credential contributes to the user fingerprint.

V6: `Reset` does not require *UV* The `Reset` should enforce stricter authorization requirements. Despite being destructive, the `Reset` API does *only* require *UP*. Anyone close to the authenticator can obtain *UP* by pressing its button or being in NFC range.

V7: `CredMgmt` does not require *UP* The `CredMgmt` API does *not* require *UP*, but only *UV*, to delete discoverable credentials. The user submits the PIN only once but can delete any number of credentials. In contrast, creating N credentials also requires N *UP* checks.

V8: `Selection` is usable for DoS The `Selection` API can be used to continuously request *UP* checks to the Authenticator and put it in an unresponsive state as the API is not rate limited.

Mapping to attacks Table 3 maps the eight vulnerabilities (columns) to the eleven CTRAPS attacks. V1 is the root cause of CI and AC attacks, as it allows an untrusted client or a MitM attacker to connect to the authenticator without authenticating it. V2 provides stealthiness to AC attacks since, without visual feedback, the user cannot confirm whether the API he called has been confounded. Due to V3, CI$_1$ and CI$_2$ over NCF require zero clicks instead of one (i.e., *UP* check). V3 also unlocks several new API confusion combination, such as `GetInfo` (no authorization requirement) into `Reset` (V3 bypasses *UP*.

V4 allows to perform the destructive CI$_2$, CI$_3$, AC$_1$, AC$_2$, and AC$_5$ even when the user calls a non-destructive API, such

Table 3: Mapping the eight vulnerabilities (columns) to the four CI and seven AC attacks.

	V1	V2	V3	V4	V5	V6	V7	V8
CI_1	✓	✓	✓	✗	✗	✓	✗	✗
CI_2	✓	✓	✓	✗	✓	✗	✗	✗
CI_3	✓	✓	✗	✗	✗	✗	✗	✗
CI_4	✓	✓	✗	✗	✗	✗	✗	✗
AC_1	✓	✓	✗	✓	✗	✗	✓	✗
AC_2	✓	✓	✓	✓	✗	✓	✗	✗
AC_3	✓	✓	✓	✗	✓	✗	✗	✗
AC_4	✓	✓	✓	✗	✗	✗	✗	✗
AC_5	✓	✓	✗	✓	✗	✗	✗	✗
AC_6	✓	✓	✗	✗	✗	✗	✗	✓
AC_7	✓	✓	✗	✗	✗	✗	✗	✗

as `Selection`. V5 enables the usage of identifiers as persistent fingerprints, resulting in two user tracking attacks (i.e., CI_2 and AC_3). V6 allows for a zero-click factory reset attack (i.e., CI_1) over NFC. V7 allows for a one-click credential deletion attack (i.e., AC_1). V8 enables a persistent and reliable DoS attack on the authenticator (i.e., AC_6).

5.2 Impact

The eleven CTRAPS attacks break the security, privacy, and availability of the FIDO2 ecosystem, with widespread and severe implications. We support our claims with the experimental results in Section 7.

Our attacks exploit *protocol-level* CTAP vulnerabilities, working regardless of the transport and the implementation details of the authenticator, the client, and the relying party. They threaten millions of authenticators in the wild, and their respective users, with destructive and scalable attacks. Being at the protocol-level, the root causes are complex to fix, and most authenticators (e.g., Yubikeys) do not support firmware updates anyways, leaving them vulnerable forever.

Our attacks affect users, authenticators, relying parties, and clients alike. For example, by erasing credentials, we remove the ability to perform web authentication from users and relying parties, by locking the authenticator, we prevent its usage, and by performing API confusion, we trick the client into believing that a confounded API call was executed legitimately instead.

Anyone can deploy our *practical* and *low-cost* attacks, as they require minimal equipment, such as an NFC reader or a smartphone. However, their realistic outcome causes concrete and immediate damage with limited or no user interaction and notice. For instance, we lost access to our test Google and Apple ID accounts because we could not pass 2FA after deleting our credentials with AC_1. No prior attack achieved similar goals, such as credential tampering and user tracking.

The CI attacks over NFC do not require compromising the user device or any interaction. They work out-of-the-box on any NFC-enabled device. The AC attacks are stealthy and persistent. They do not trigger abnormal behavior, only asking for *UV* and *UP* when the user expects it. As a direct consequence, the user will likely trigger the attacks on multiple occasions, not realizing the lingering threat. CI and AC attacks can even be *combined*. For example, the attacker can obtain a fingerprint with AC_7 and track the user with CI_2.

6 Implementation

We present `CTRAPS`, a novel toolkit implementing the CTRAPS attacks. It has *three* modules: a CTAP testbed (Section 6.1), the malicious CTAP clients (Section 6.2), and the Wireshark dissectors (Section 6.3). We describe how we implemented each module and their novelties.

6.1 CTAP Testbed

Our CTAP testbed is a Python3 module that includes a virtual relying party with a customizable WebAuthn server and a virtual client talking to a real authenticator over CTAP and a virtual relying party over WebAuthn. The testbed has two benefits: (i) It allows to perform experiments locally, safely, and without an Internet connection, without interacting and tampering with real relying parties. (ii) It allows to simulate different attack scenarios by realistically replicating client and server configurations, including the credential protection policy (*CredProtect*). Our virtual relying party and client are extensions of the `python-fido2` [67], Yubico's open-source Python library for FIDO2.

The virtual relying party is implemented as a customizable WebAuthn server running on our testbed, and not on a network like an ordinary relying party. We extended existing code by adding standard relying party templates and fast customization of the server's parameters. For example, we implemented a template imitating Microsoft relying party, including the same identifier (i.e., *login.microsoft.com*). Options such as the credential protection policy and the attestation verification. The virtual relying party was instrumental in setting up the authenticator in the correct state for our CI and AC attacks (e.g., registering credentials with a weak protection policy) quickly and without involving real relying parties. A separate malicious client deploys the actual attacks.

The virtual CTAP client offers a low-level API to interact with the authenticator and the virtual relying party. It can send CTAP commands in any order with custom or even malformed payload values. We extended the existing code by adding a set of common and abnormal use cases useful for security testing and vulnerability assessment. For example, authentication to a relying party or a mass credential registration that fills the entire memory of the authenticator. Each use case includes appropriate and configurable settings and capabilities, such as
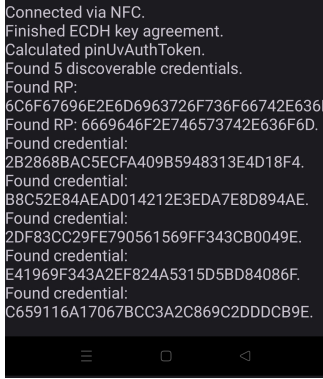
Figure 5: Our malicious Android CTAP client performing zero-click CI_2 over NFC, leaking relying party and credential identifiers.

the CTAP authorization requirements, challenge, origin, and CTAP *rk* and *up* options.

The testbed requires the user to authorize direct access to the authenticator. Linux requires adding extra `udev` rules, MacOS asks to accept a notification on the screen, and Windows needs admin privileges. However, this behavior is expected by the user and does not hinder the effectiveness of our attacks (i.e., the user approves the notification on MacOS as he is the one plugging in the authenticator and wanting to use it).

6.2 Malicious CTAP Clients

In our toolkit, we develop three malicious CTAP clients, deploying the CI and AC attacks. We also release five video demonstrations of our attacks on real authenticators.

We implemented the CI attacks on a malicious CTAP client running on an Android app. Currently, only attacks over NFC are fully functional, but we plan to add the USB transport in the future, as it only requires engineering effort.

The app supports both a proximity and a remote CI attack mode. In the proximity mode, the attacker controls the app and constantly scans for authenticators to connect to and exploit. For example, the app can perform CI_2 to track the user with leaked identifiers, as shown in Figure 5. In the remote mode, the app spoofs a legitimate NFC app, enticing the user to connect their authenticator (e.g., by asking for FIDO2 authentication). We include in the app a CBOR parser for CTAP that we wrote.

The app does not need root privileges and asks at runtime for the dangerous `android.permission.NFC`, required to access the `android.nfc` [24] API. However, this is not an issue, as we are not trying to hide the app's NFC capabilities. The app also needs the standard install-time `android.permission.INTERNET` to exfiltrate the data collected through CI_2 and CI_4.

We also deployed the CI attacks using a Proxmark3 [54], an open-source and programmable development kit for NFC

Table 4: Details about the six authenticators we attack. All authenticators support USB and NFC, except OpenSK, which only supports USB. FVer: firmware version, OSF: open-source firmware, DCr: discoverable credentials.

Authenticator	Manuf	Year	FVer	OSF	DCr
YubiKey 5	Yubico	2018	5.2.7	No	25
YubiKey 5 FIPS	Yubico	2021	5.4.3	No	25
Feitian K9	Feitian	2016	3.3.01	No	50
Solo V1	SoloKeys	2018	4.1.5	Yes	50
Solo V2 Hacker	SoloKeys	2021	2.964	Yes	50
OpenSK	Google	2023	2.1	Yes	150

(RFID). By equipping the Proxmark3 with a long-range high-frequency antenna, we were able to extend the reach of our attacks. The long-range antenna has an indicative range of 100 to 120 millimeters, as opposed to the 40 to 85 millimeters of the built-in antenna. We developed the CI attacks in a custom Lua script using the Proxmark3 ISO14443 Type A module (i.e., `read14a`). The Proxmark communicates with the authenticator utilizing the same APDU commands we use for the Android app.

We implemented the AC attacks in an Electron app simulating a MitM attacker. The app runs a malicious CTAP client developed as a Javascript library. Our code imports the *node-hid* module to access the USB HID traffic. It scans for local HID devices, identifies the authenticator from their properties (e.g., the product and manufacturer fields), and connects to it. The client sends binary data over USB to the authenticator, achieving the same results as MitM attackers.

6.3 FIDO Wireshark Dissectors

We extended the official Wireshark FIDO2 dissectors [71] with new and valuable features. We add support for the `Cred Mgmt` API. We include parsers for `WAITING` and `PROCESSING` keepalive status codes that identify when authenticators are unavailable waiting for *UP*. We parse the authenticator's capabilities in the `CTAPHID_INIT` message, which helps test AC_7. We provide an improved way to display CTAP data when dissecting CTAPHID (USB) and ISO7816/ISO14443 (NFC). Finally, we add missing vendor and product identifiers to the dissector tables. We release the dissectors as a Lua script (i.e., `fido2-dissectors.lua`) that can be found in our toolkit.

7 Evaluation

We successfully evaluated our eleven attacks against *six* popular authenticators from Yubico, Feitian, SoloKeys, and Google supporting CTAP over USB and NFC, and *ten* well-known relying parties, including Microsoft, Apple, GitHub, and Facebook. Our evaluation includes relying parties because their

Table 5: CI and AC attacks on six authenticators. The first column lists the authenticators' names. The remaining columns report our four CI and seven AC attacks on CTAP. ✓: attack is effective on the authenticator, **n/a**: not applicable as the authenticator does not implement the `Selection` API.

Authenticator	CI_1	CI_2	CI_3	CI_4	AC_1	AC_2	AC_3	AC_4	AC_5	AC_6	AC_7
YubiKey 5	✓	✓	✓	✓	✓	✓	✓	✓	✓	n/a	✓
YubiKey 5 FIPS	✓	✓	✓	✓	✓	✓	✓	✓	✓	n/a	✓
Feitian K9	✓	✓	✓	✓	✓	✓	✓	✓	✓	n/a	✓
Solo V1	✓	✓	✓	✓	✓	✓	✓	✓	✓	n/a	✓
Solo V2 Hacker	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
OpenSK	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

CI_1: Factory reset authenticator, CI_2: Track user from credentials, CI_3: Force authenticator lockout, CI_4: Profile authenticator, AC_1: Delete discoverable credentials, AC_2: Factory reset authenticator, AC_3: Track user from credentials, AC_4: Fill authenticator credential storage, AC_5: Force authenticator lockout, AC_6: Authenticator DoS, AC_7: Profile authenticator.

credential protection policy affects our attacks (i.e., by requiring *UV*). At the same time, operations on the authenticator (i.e., deleting credentials) indirectly affect relying parties due to the user losing access to his online accounts.

We tested the CI attacks using the malicious Android CTAP client and the Proxmark3, and the AC attacks using the Electron app that simulates a MitM position in the USB traffic (for more details about our toolkit, refer to Section 6).

We present our evaluation setup and results.

7.1 Setup

Authenticators We evaluate *six* popular FIDO2 authenticators. Table 4 shows their technical details. The YubiKey 5 NFC, YubiKey 5 NFC FIPS, and Feitian NFC K9 are closed-source and do not support firmware updates, The Solo V1, Solo V2 Hacker, and Open Security Key (OpenSK) have an open-source firmware (OSF) that we updated to their latest version. The authenticators support USB and NFC, except for OpenSK, which has an NFC module but supports only USB. The Solo V1 requires a button press to activate NFC. We did not find any FIDO2 authenticator supporting BLE.

The authenticators store a maximum of 25 (Yubico), 50 (Feitian and SoloKeys), or 150 (OpenSK) discoverable credentials. The YubiKey 5 FIPS is FIPS140-2 compliant. Hence, it should provide strong security guarantees. We run OpenSK on an NRF52840 dongle, but the attacks could be tested on any board supporting OpenSK.

Relying parties Our list of relying parties covers pervasive and heterogeneous online services, including software as a service, social, gaming, cryptographic signing, authentication, and cloud storage. We registered our authenticators with *ten* FIDO2 relying parties: Adobe, Apple, DocuSign, Facebook, GitHub, Hancock, Microsoft, NVidia, Synology, and Vault Vision. Some offer Single Sign-On (SSO), enabling access to multiple services. For example, a single set of FIDO2 credentials can log into Microsoft, OneDrive, Outlook, and Minecraft. Consequently, erasing a single credential has a

widespread effect on multiple online services.

CTRAPS toolkit We ran the CTAP testbed and the Electron app on a Dell Inspiron 15 3502 laptop (OSes: Ubuntu 22.04.3 LTS and Windows 11 Home) and on a MacBook Pro M1 (OS: MacOs Ventura 13.4). We connected the authenticator by plugging it into a USB port.

We installed the malicious CTAP client Android app on a rooted Google Pixel 2 (OS: Android 11), a non-rooted RealMe 11 Pro (OS: Android 14) and Xiaomi Redmi Plus 5 (OS: Android 8.1). Root access was not required and did not affect the attack in any way. We equipped a Proxmark3 RDV4 with a long-range high-frequency antenna to extend the range of our NFC attacks. We connected the authenticator to the Android app and the Proxmark3 by placing it within NFC range.

7.2 Authenticators Results

Table 5 shows the evaluation results for the CI and AC attacks. We tested the CI and AC attacks in proximity (NFC) and remotely (malicious app). We successfully ran the attacks on all tested authenticators, including a recent and FIPS-compliant YubiKey. Four AC_6 attacks are not applicable as the related authenticators do not support the `Selection` API. As expected, since we attack CTAP at the protocol level, the attacks are effective regardless of the CTAP transport (i.e., USB or NFC) or the authenticator's software and hardware.

We also found a `CredMgmt` implementation vulnerability on the YubiKey 5 and YubiKey 5 FIPS, now tracked with CVE-2024-35311 [69]. An authenticator should not execute the subcommand `CredMgmt(EnumRpsGetNextRp)`, unless `CredMgmt(EnumRpsBegin)` was called first. This is relevant because the latter requires *UV*, the former does not. However, Yubico failed to implement this requirement due to an incorrect handling of the authenticator's state. We exploit this flaw to perform a zero-click *relying party leakage*. We repeatedly call `CredMgmt(EnumRpsGetNextRp)`, which does not require *UV* and is not affected by *CredProtect*, to reveal all relying parties, except one, linked to the discoverable cre-

Table 6: CTRAPS attacks on ten relying parties. The first and second columns list the relying parties' names and identifiers. The third column highlights whether they register discoverable (Disc, DiscWeak) or non-discoverable (NonDisc) credentials. We indicate with DiscWeak a relying party using the default and weak *CredProtect=UVOptional* policy. Columns four, five, and six specify the effect of each attack. n/a: the attack is not applicable because the relying party currently does not support discoverable credentials.

Rp	RpId	Cred	Delete Creds	Track User	DoS Authenticator
Adobe	`adobe.com`	Disc	AC_1, CI_1, AC_2	CI_2, AC_3	AC_4, CI_3, AC_5, AC_6
Apple	`apple.com`	DiscWeak	AC_1, CI_1, AC_2	CI_2, AC_3	AC_4, CI_3, AC_5, AC_6
DocuSign	`account.docusign.com`	NonDisc	CI_1, AC_2	n/a	CI_3, AC_5, AC_6
Facebook	`facebook.com`	NonDisc	CI_1, AC_2	n/a	CI_3, AC_5, AC_6
GitHub	`github.com`	Disc	AC_1, CI_1, AC_2	CI_2, AC_3	AC_4, CI_3, AC_5, AC_6
Hancock	`hancock.ink`	Disc	AC_1, CI_1, AC_2	CI_2, AC_3	AC_4, CI_3, AC_5, AC_6
Microsoft	`login.microsoft.com`	DiscWeak	AC_1, CI_1, AC_2	CI_2, AC_3	AC_4, CI_3, AC_5, AC_6
NVidia	`login.nvgs.nvidia.com`	Disc	AC_1, CI_1, AC_2	CI_2, AC_3	AC_4, CI_3, AC_5, AC_6
Synology	`account.synology.com`	Disc	AC_1, CI_1, AC_2	CI_2, AC_3	AC_4, CI_3, AC_5, AC_6
Vault Vision	`auth.vaultvision.com`	Disc	AC_1, CI_1, AC_2	CI_2, AC_3	AC_4, CI_3, AC_5, AC_6

dentials stored on the authenticator. Given a sufficiently large amount of relying parties, this information can also be used to fingerprint and track the user.

Depending on their complexity, the proximity CI and AC attacks required 50 to 500 milliseconds within the NFC range. For example, AC_1 takes the longest because it involves significantly more CTAP messages than any other attack. Typically, a smartphone's NFC range is four centimeters or less [8], but, on authenticators, we only achieved up to two centimeters of range. The Proxmark3 built-in antenna also reached up to two centimeters, which we extended up to six and a half centimeters with a long-range antenna. However, prior work demonstrated that, with specialized equipment, the NFC range can be extended up to 50 centimeters [41].

We confirmed on all six authenticators that, as prescribed by CTAP, a factory reset over USB can only be executed if the device has been plugged into the USB port within the last ten seconds. Therefore, we CI_1 and AC_2 over USB features this additional constraint.

7.3 Relying Parties Results

Table 6 shows that the CTRAPS attacks directly or indirectly affect all evaluated relying parties, even without sending WebAuthn messages. We exploit eight relying parties supporting discoverable credentials, including two using a weak *CredProtect* policy, and two employing non-discoverable ones.

AC_1, CI_1, and AC_2 prevent access to a relying party via credential deletion. CI_2 and AC_3 utilize the user identifiers a relying party provides to track users. AC_4, CI_3, AC_5, and AC_6 indirectly prevent relying parties from authenticating registered authenticators. Since AC_1 and AC_4 target discoverable credentials, they do not apply to relying parties registering non-discoverable ones, like DocuSign and Facebook.

Among the relying parties supporting discoverable credentials, we found that Microsoft and Apple rely on the weak and default *CredProtect* policy. As a consequence, performing CI_2 and AC_3 on them does not require *UV*.

8 Discussion

8.1 Comparison with prior FIDO attacks

Table 7 compares our work with relevant attacks on FIDO2. The CI attacks are the first client impersonation attacks on FIDO2 (CTAP2+). While the AC attacks are the first API confusion attack on FIDO2. The CI attacks are low-cost as they do not require a compromised client or user device or prior knowledge of user secrets (e.g., credential identifiers). The AC attacks have a moderate cost, as they require a man-in-the-middle position between the authenticator and the client. The CTRAPS attacks have a higher impact than most other previous attacks, as, for example, they can permanently destroy all credentials and track users. We now directly compare our work with a selection of the attacks from Table 7.

In [50], the authors presented WebAuthn deception attacks through a terminal-based malware, redirecting the user to an attacker-controlled authentication page and stealing credentials via a keylogger. Our attacker model is *weaker*, relying solely on a compromised (CTAP) client instead of a malware, keylogger, and malicious browser session.

In [10] and [11], the researchers formally verified FIDO2 and then demonstrated the feasibility of client impersonation by exploiting the unauthenticated CTAP ECDH also discussed in [34]. They required a compromised client (i.e., a browser) running on the user's device to decrypt the authenticator's PIN necessary for *UV*. In contrast, we focus on bypassing or stealing *UV* and *UP*, granted by the user to other CTAP

APIs, without needing to decrypt the PIN or resorting to social engineering, UI deception, or other manipulative tactics [42].

The authors of [65] examined attacks involving local (i.e., browser extension) and physical (i.e., temporary access to the authenticator) adversaries, including misbinding, MitM, and session hijacking. They identify two protocol-level flaws in FIDO2: a lack of confidentiality and integrity and broken clone detection. In comparison, we evaluate a *proximity-based* attacker (e.g., an NFC reader) never considered before in FIDO2. Moreover, we find six new FIDO2 protocol-level issues in addition to theirs.

8.2 FIDO2 Reference Threat Model Issues

FIDO2 has a non-normative *reference threat model* [2] that includes security assumptions, goals, and threats against clients, authenticators, and relying parties. We found *three issues* (IS1, IS2, and IS3) with their threat model:

IS1: Unclear security boundaries The threat model presents six broad security assumptions but then violates them when discussing threats. For example, SA-4 states that the FIDO user device and applications involved in a FIDO operation act as trustworthy agents of the user. This implies that the FIDO client (e.g., the user's browser or mobile app) must be inherently trusted. However, the threat model includes threats breaking SA-4 like *T-1.2.1: FIDO client corruption* that identifies an attacker with code execution on a FIDO client. Instead, security assumptions should hold to enable a security analyst to draw security boundaries (i.e., differentiate what we trust from what can be attacked).

IS2: Proximity threats are missing Despite FIDO supporting proximity transports like NFC and BLE, their reference model classifies proximity-based threats as physical access, even though these two categories have significant differences. For example, compared to physical access, the range of a proximity attack can be extended. Hence, our proximity attacks, which do not require physical access, cannot be accurately described within this reference threat model.

IS3: Security goals are narrow The threat model has narrow security goals based on [18] (2006) and [14] (2012). The security goals focus on web authentication but overlook FIDO clients and roaming authenticators. For example, there are no security goals for the Authenticator API (i.e., addressing all AC attacks) or discoverable credentials (i.e., addressing AC_1, CI_1, and AC_2).

9 Countermeasures

We present the design and evaluation of *eight* practical and backward-compliant countermeasures fixing the eleven CTRAPS attacks and their related eight vulnerabilities. Each countermeasure addresses a vulnerability (e.g., C1 fixes V1) and reduces the CTAP attack surface. The countermeasures

are implementable as amendments to the FIDO2 standard or as FIDO2 extensions. We describe each countermeasure.

C1: Trusted CTAP clients We address V1 by recommending that FIDO provide a list of trusted CTAP clients. FIDO offers several certifications, including the FIDO Functional Certification [6] which only attests the *interoperability* of clients, servers, and authenticators. We suggest extending this certification also to cover the *trustworthiness* of CTAP clients. For instance, FIDO could implement a Software Bill Of Materials (SBOM) solution to monitor trusted CTAP clients and their vulnerabilities [63].

C2: Authenticator visual feedback We address V2 by requiring the authenticator to provide visual feedback of the called APIs. For instance, the authenticator's LED could blink *once* for non-destructive API calls and *twice* for destructive ones. The CTAP *wink* command, which blinks the LED, must be disabled during this visual feedback.

C3: User interaction for *UP* over NFC We address V3 by requiring user interaction during *UP* checks over NFC. For example, the user could press a button on the authenticator to grant *UP* over NFC, similar to *UP* checks over USB.

C4: Dedicated PIN for destructive APIs We address V6 by introducing a dedicated PIN to authorize destructive API calls (e.g., `CredMgmt` and `Reset`) and by repurposing the current PIN to authorize non-destructive API calls (e.g., `Selection` and `GetInfo`). The new PIN should have the same or stricter requirements as the non-destructive PIN (i.e., four to sixty-three Unicode characters [1]).

C5: Dynamic and *UV*-protected CredId and UserId We address V5 by implementing dynamic CredId and UserId and mandating *CredProtect=UVRequired*. CredId and UserId should rotate after a set amount of logins (e.g., every ten logins) or a time interval (e.g., once per month). Hence, we raise the bar for user profiling and tracking attacks on authenticators. Currently, the user can indirectly change a CredId by calling `MakeCred` to generate a new credential for his account, replacing the old one. However, the user cannot change the UserId, which the relying party determines and, based on our experience, remains fixed to the user account.

C6: `Reset` must require *UV* We address V6 by requiring *UV* to call `Reset`. Hence, the user must validate a factory reset by entering a valid PIN.

C7: `CredMgmt` must require *UP* We address V7 by requiring *UP* to call `CredMgmt`. Hence, the user must authorize credential deletion one by one, making it impossible to delete multiple credentials with a single API call.

C8: `CredMgmt` must require *UP* We address V8 by enforcing temporal rate limiting on `Selection` calls to a maximum of three calls within two minutes. We are not expecting issues with our rate limiting, akin to the limiting already existing for `ClientPin(GetPinToken)`, as a client typically calls `Selection` once per session.

Usability We believe that the stronger security granted by our countermeasures is worth the inevitable usability trade-

Table 7: Recent attacks on FIDO. We assign each attack a cost and impact. For example, the cost for a MitM attacker is Mid, and for a proximity-based attacker is Low. Similarly, hijacking a session has a Mid impact, and permanently destroying credentials has a High impact.

Attack	Class	Protocol	Transp	Surface	Impl	Reqs	Cost	Impact
CTAP MitM [34]	DH MitM	CTAP2.0	All	ClientPin	✗	MitM	Mid	Mid
Privacy leak [34]	Eavesdropping	CTAP2.0	All	MakeCred	✗	n/a	Low	Low
Auth rebind [34]	Auth rebind	WebAuthn	All	Creds.create	✗	n/a	High	High
Parallel session [34]	Session hijack	WebAuthn	All	Creds.get	✗	n/a	Mid	Mid
Evil maid [47]	Phys access	n/a	n/a	Auth TEE	✗	Phys access	High	High
Titan phone imp [47]	Impersonation	U2F	BLE	Android	✓	Proximity	Low	Mid
Titan key imp [47]	Impersonation	U2F	BLE	Google Acc	✓	Proximity	Low	Mid
Auth imp [11]	DH MitM	CTAP2.0/2.1	USB	ClientPin	✓	Mal browser	Mid	Mid
Web MitM [11]	Session hijack	WebAuthn	USB	Creds.get	✓	Mal browser	Mid	Mid
Rogue key [11]	Auth rebind	WebAuthn	USB	Creds.create	✓	Mal browser	Mid	High
FIDOLA [50]	Session hijack	WebAuthn	USB	Creds.get	✓	Malware	High	Mid
CTRAPS CI	Impersonation	CTAP2.0/2.1/2.2	All	Auth API	✓	Proximity	Low	High
CTRAPS AC	API confusion	CTAP2.0/2.1/2.2	All	Auth API	✓	MitM	Mid	High

offs. C1, C2, and C8 do not affect usability. C5 only introduces one additional *UV* and *UP* check every time the credential and user identifier need to rotate out (e.g., once per month), barely affecting usability. On the other hand, C4 requires the user to remember a second PIN, and C6 and C7 add more authorization requirements to `Reset` and `CredMgmt`. C3 also adds user interaction when connecting to the authenticator over NFC.

Adding a display We do not consider adding a display to a roaming authenticator an optimal solution as it is *not backward-compliant*. Millions of deployed authenticators would remain vulnerable. Moreover, it would require significant hardware and software modifications, such as adding a secure display, a display controller firmware, and a battery, that would introduce usability, performance, and cost issues.

10 Related Work

Attacks on FIDO(2) Researchers demonstrated practical attacks on older FIDO versions, such as authenticator rebinding, parallel sessions, and multi-user attacks [37, 46], USB HID man-in-the-middle attacks [16], BLE pairing [15], relying party public key substitution [58], bypassing push-based 2FA [38], real-time phishing [61], and side channel attacks [39, 55]. FIDO2 was also found vulnerable to side-channel attacks [47] and rogue key or impersonation threats [11]. Moreover, attacks on lower layers trusted by FIDO2 were presented including IV reuse on Samsung Keystore [59]. No prior attack investigated *API confusion* on CTAP, including its *latest* version.

Formal Analysis The formal analysis and verification community extensively researched FIDO. The community formally verified FIDO's Universal Authentication Framework (UAF) [28, 53], FIDO2 (including its privacy, revocation, attestation, and post-quantum crypto) [10, 12, 13, 35]. Yubico proposed a key recovery mechanism based on a backup authenticator that was proven secure using the asynchronous remote key generation (ARKG) primitive [31]. The formal analyses are *not* covering our CI and AC attacks.

Extensions FIDO supports extensions to add optional features in a backward-compliant way. For instance, FeIDO [57] proposes an extension to recover a FIDO2 credential using an electronic identifier. Extensions are not secure by default, and researchers proposed a fix to protect them against MitM attacks [17]. We suggest to *update* the CTAP specification rather than implementing our countermeasures as FIDO extensions that would be optional and insecure by design.

Enhancements Researchers proposed (cryptographic) enhancements to FIDO protocols. In [32], the authors present a hybrid post-quantum signature scheme for FIDO2 and tested it using OpenSK [33] (which we exploit in this work). In [36], the authors propose a global key revocation procedure for WebAuthn that revokes credentials without communicating to each individual relying party WebAuthn server. True2F [23] presented a backdoor-resistant FIDO U2F design, protecting the authenticator from a malicious browser by requiring the authenticator interaction during every authentication, and from fingerprinting by rate limiting credential registration. Proposed enhancements are *not* addressing our attacks, which are effective *regardless of* the FIDO2 cryptographic primitives.

Usability Researchers performed extensive usability studies on FIDO U2F [19, 20, 22, 45], FIDO2 roaming authenticators [25, 52], passkeys [40], and cross-site 2FA [48]. Our paper is *orthogonal* to usability studies.

Surveys There are several FIDO survey papers. In [9] the authors describe the evolution of FIDO protocols, security

requirements, and adoption factors. In [49], the authors surveyed the adoption of passwordless authentication among a large user base, considering users' perceptions, acceptance, and concern with single-factor authentication without passwords. Our paper is *orthogonal* to surveys.

11 Conclusion

No prior work assessed the CTAP Authenticator API, a critical API exposed by a client to an authenticator to manage, create, and delete credentials. We address this gap by presenting the first security and privacy evaluation of the CTAP Authenticator API. We uncover two classes of protocol-level attacks capable of abusing the API. The CI attacks spoof a CTAP client to a victim authenticator to factory reset, track, and DoS the authenticator. The AC attacks utilize a MitM position to change user CTAP API calls to an API desired by the attacker, potentially destructive while stealing their authorization. We deploy the first CTAP client impersonation in FIDO2, enabling an attacker to call CTAP APIs without authorization or user interaction. We also introduce a novel attack strategy called API confusion, which changes, without user consent, the API called by the user to an API chosen by the attacker.

We uncover eleven new proximity-based and remote attacks that can severely impact millions of FIDO2 users. For example, our attacks delete FIDO2 credentials and master keys (security breach) and track users through their credentials (privacy breach). The attacks are effective on the entire FIDO2 ecosystem as they target eight vulnerabilities we discovered in the CTAP specification. These flaws include the lack of CTAP client authentication and improper API authorizations. CTRAPS attacks are low-cost, as they do not require specialized or expensive equipment, and stealthy, as they do not trigger unexpected user interactions.

We develop the `CTRAPS` toolkit to test our attacks with a low-cost setup and on a large scale. It includes a CTAP testbed with a virtual relying party and a virtual client, a CTAP client NFC impersonator (i.e., malicious Proxmark scripts and Android NFC app), and enhanced Wireshark dissectors for CTAP. We successfully exploit six authenticators and ten relying parties from leading FIDO2 players such as Yubico, Feitian, Google, Microsoft, and Apple. We develop eight effective and legacy-compliant countermeasures to fix our attacks and their root causes.

We share *three lessons* we learned about FIDO2 *credential storage* and *passwordless-ness*, which are valuable for the current transition from single-factor authentication to 2FA and passkeys [21, 56]: (i) Being stored on the authenticator, FIDO2 discoverable credentials are protected from third-party data breaches. However, this introduces new attacks that work exclusively on discoverable credentials (i.e., AC_1, CI_2, AC_3, and AC_4); (ii) FIDO2 users cannot prevent attacks targeting discoverable credentials, as they cannot choose the type of credentials they register and their protection policies, decided by the relying party and the client instead. (iii) The FIDO2 core message is to steer away from passwords because they are vulnerable to phishing. However, digging deeper, we realized that FIDO2 still relies on phishable mechanisms, even for passwordless authentication. For instance, a passwordless credential is protected by an alphanumeric PIN (i.e., a phishable sequence the user must remember).

Acknowledgments

References

[1] FIDO Alliance. CTAP 2.1 Proposed Standard with Errata. https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-errata-20220621.html, 2022.

[2] FIDO Alliance. FIDO Security Reference, Review Draft 23 May 2022. https://fidoalliance.org/specs/common-specs/fido-security-ref-v2.1-ps-20220523.pdf, 2022.

[3] FIDO Alliance. CTAP 2.2 Review Draft 01. https://fidoalliance.org/specs/fido-v2.2-rd-20230321/fido-client-to-authenticator-protocol-v2.2-rd-20230321.html, 2023.

[4] FIDO Alliance. FIDO Alliance Specifications Overview. https://fidoalliance.org/specifications, 2024.

[5] FIDO Alliance. FIDO Certified Products. https://fidoalliance.org/certification/fido-certified-products/, 2024.

[6] FIDO Alliance. FIDO Functional Certification. https://fidoalliance.org/certification/functional-certification/, 2024.

[7] FIDO Alliance. FIDO Security Secretariat. https://fidoalliance.org/certification/secretariat/, 2024.

[8] Android. Near Field Communication (NFC) Overview. https://developer.android.com/develop/connectivity/nfc, 2024.

[9] Anna Angelogianni, Ilias Politis, and Christos Xenakis. How many FIDO protocols are needed? Surveying the design, security and market perspectives. *arXiv preprint arXiv:2107.00577*, 2021.

[10] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, and Bogdan Warinschi. Provable security analysis of FIDO2. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part III 41*, pages 125–156, 2021.

[11] Manuel Barbosa, André Cirne, and Luís Esquível. Rogue key and impersonation attacks on FIDO2: From theory to practice. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*. Association for Computing Machinery, 2023.

[12] Nina Bindel, Cas Cremers, and Mang Zhao. FIDO2, CTAP 2.1, and WebAuthn 2: Provable security and post-quantum instantiation. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1471–1490. IEEE, 2023.

[13] Nina Bindel, Nicolas Gama, Sandra Guasch, and Eyal Ronen. To attest or not to attest, this is the question–Provable attestation in FIDO2. *Cryptology ePrint Archive*, 2023.

[14] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE symposium on security and privacy*, pages 553–567. IEEE, 2012.

[15] Christiaan Brand. Advisory: Security Issue with Bluetooth Low Energy (BLE) Titan Security Keys. https://security.googleblog.com/2019/05/titan-keys-update.html, 2019.

[16] Thanh Bui, Siddharth Prakash Rao, Markku Antikainen, Viswanathan Manihatty Bojan, and Tuomas Aura. Man-in-the-Machine: Exploiting Ill-Secured Communication Inside the Computer. In *27th USENIX security symposium (USENIX Security 18)*, pages 1511–1525, 2018.

[17] Andre Büttner and Nils Gruschka. Protecting FIDO Extensions Against Man-in-the-Middle Attacks. In *International Workshop on Emerging Technologies for Authorization and Authentication*, pages 70–87. Springer, 2022.

[18] Tsai Chwei-Shyong, Lee Cheng-Chi, and Min-Shiang Hwang. Password Authentication Schemes: Current Status and Key Issues. *International Journal of Network Security*, 2006.

[19] Stéphane Ciolino, Simon Parkin, and Paul Dunphy. Of Two Minds about Two-Factor: Understanding Everyday FIDO/U2F Usability through Device Comparison and Experience Sampling. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 339–356, 2019.

[20] Jessica Colnago, Summer Devlin, Maggie Oates, Chelse Swoopes, Lujo Bauer, Lorrie Cranor, and Nicolas Christin. "It's not actually that horrible" Exploring Adoption of Two-Factor Authentication at a University. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–11, 2018.

[21] Mike Hanley (GitHub CSO). Securing millions of developers through 2FA. https://github.blog/2024-04-24-securing-millions-of-developers-through-2fa/, 2024.

[22] Sanchari Das, Andrew Dingman, and L Jean Camp. Why Johnny doesn't use two factor a two-phase usability study of the FIDO U2F security key. In *International Conference on Financial Cryptography and Data Security*, pages 160–179. Springer, 2018.

[23] 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)*, pages 398–416, 2019.

[24] Android developers. Android NFC basics. https://developer.android.com/develop/connectivity/nfc/nfc, 2023.

[25] Florian M Farke, Lennart Lorenz, Theodor Schnitzler, Philipp Markert, and Markus Dürmuth. You still use the password after all–Exploring FIDO2 Security Keys in a Small Company. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, pages 19–35, 2020.

[26] Feitian. Feitian Android App. https://play.google.com/store/apps/details?id=com.ft.entersafe.iepassmanager, 2022.

[27] Feitian. Feitian iOS App. https://apps.apple.com/us/app/iepassmanager/id1504200260, 2022.

[28] Haonan Feng, Hui Li, Xuesong Pan, Ziming Zhao, and T Cactilab. A Formal Analysis of the FIDO UAF Protocol. In *Network & Distributed System Security Symposium (NDSS'21)*, 2021.

[29] FIDO Alliance. U.S. General Services Administration's Rollout of FIDO2 on login.gov. https://fidoalliance.org/u-s-general-services-administrations-rollout-of-fido2-on-login-gov/, 2023.

[30] FIDO Alliance. FIDO Alliance Publishes New Specifications to Promote User Choice and Enhanced UX for Passkeys. https://fidoalliance.org/fido-alliance-publishes-new-specifications-to-promote-user-choice-and-enhanced-ux-for-passkeys, 2024.

[31] Nick Frymann, Daniel Gardham, Franziskus Kiefer, Emil Lundberg, Mark Manulis, and Dain Nilsson. Asynchronous Remote Key Generation: An Analysis of Yubico's Proposal for W3C WebAuthn. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 939–954, 2020.

[32] Diana Ghinea, Fabian Kaczmarczyck, Jennifer Pullman, Julien Cretin, Rafael Misoczki, Stefan Kölbl, Luca Invernizzi, Elie Bursztein, and Jean-Michel Picod. Hybrid post-quantum signatures in hardware security keys. In *4th ACNS Workshop on Secure Crytographic Implmentation*, 2023.

[33] Google. OpenSK: a Rust Implementation of a FIDO2 Authenticator. https://github.com/google/OpenSK, 2024.

[34] Jingjing Guan, Hui Li, Haisong Ye, and Ziming Zhao. A Formal Analysis of the FIDO2 Protocols. In *European Symposium on Research in Computer Security (ESORICS)*, pages 3–21, 2022.

[35] Lucjan Hanzlik, Julian Loss, and Benedikt Wagner. Token meets wallet: Formalizing privacy and revocation for FIDO2. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1491–1508. IEEE, 2023.

[36] Lucjan Hanzlik, Julian Loss, and Benedikt Wagner. Token meets Wallet: Formalizing Privacy and Revocation for FIDO2. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1491–1508, 2023.

[37] Kexin Hu and Zhenfeng Zhang. Security analysis of an attractive online authentication standard: FIDO UAF protocol. *China Communications*, 13(12):189–198, 2016.

[38] Mohammed Jubur, Prakash Shrestha, Nitesh Saxena, and Jay Prakash. Bypassing push-based second factor and passwordless authentication with human-indistinguishable notifications. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 447–461, 2021.

[39] Michal Kepkowski, Lucjan Hanzlik, Ian Wood, and Mohamed Ali Kaafar. How Not to Handle Keys: Timing Attacks on FIDO Authenticator Privacy. In *Proceedings on Privacy Enhancing Technologies*, volume 4, pages 705–726, 2022.

[40] Michal Kepkowski, Maciej Machulak, Ian Wood, and Dali Kaafar. Challenges with Passwordless FIDO2 in an Enterprise Setting: A Usability Study. *arXiv preprint arXiv:2308.08096*, 2023.

[41] Ziv Kfir and Avishai Wool. Picking Virtual Pockets using Relay Attacks on Contactless Smartcard. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*, pages 47–58, 2005.

[42] Dhruv Kuchhal, Muhammad Saad, Adam Oest, and Frank Li. Evaluating the Security Posture of Real-World FIDO2 Deployments. In *Proceedings of the ACM conference on computer and communications security (CCS)*, 2023.

[43] Ninja Lab. A Side Journey to Titan. https://ninjalab.io/a-side-journey-to-titan, 2024.

[44] Juan Lang, Alexei Czeskis, Dirk Balfanz, Marius Schilder, and Sampath Srinivas. Security keys: practical cryptographic second factors for the modern web. In *International Conference on Financial Cryptography and Data Security*, pages 422–440. Springer, 2016.

[45] Leona Lassak, Annika Hildebrandt, Maximilian Golla, and Blase Ur. It's Stored, Hopefully, on an Encrypted Server: Mitigating Users' Misconceptions About FIDO2 Biometric WebAuthn. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 91–108, 2021.

[46] Hui Li, Xuesong Pan, Xinluo Wang, Haonan Feng, and Chengjie Shi. Authenticator rebinding attack of the UAF protocol on mobile devices. *Wireless Communications and Mobile Computing*, 2020:1–14, 2020.

[47] Victor Lomne. An Overview Of The Security Of Some Hardware FIDO(2) Tokens. https://www.youtube.com/watch?v=hpOp9X4sMaE, 2022.

[48] Sanam Ghorbani Lyastani, Michael Backes, and Sven Bugiel. A systematic study of the consistency of two-factor authentication user journeys on top-ranked websites. In *30th Annual Network & Distributed System Security Symposium (NDSS'23)*, 2023.

[49] Sanam Ghorbani Lyastani, Michael Schilling, Michaela Neumayr, Michael Backes, and Sven Bugiel. Is FIDO2 the Kingslayer of User Authentication? A Comparative Usability Study of FIDO2 Passwordless Authentication. In *IEEE Symposium on Security and Privacy*, pages 268–285, 2020.

[50] Ahmed Tanvir Mahdad, Mohammed Jubur, and Nitesh Saxena. Breaching Security Keys without Root: FIDO2

Deception Attacks via Overlays exploiting Limited Display Authenticators. In *Proceedings of the ACM conference on computer and communications security (CCS)*, 2024.

[51] National Institute of Standards and U.S. Department of Commerce Technology. NIST Special Publication 800-63B, Digital Identity Guidelines, Authentication and Lifecycle Management. https://pages.nist.gov/800-63-3/sp800-63b.html#-5112-memorized-secret-verifiers, 2017.

[52] Kentrell Owens, Olabode Anise, Amanda Krauss, and Blase Ur. User Perceptions of the Usability and Security of Smartphones as FIDO2 Roaming Authenticators. In *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, pages 57–76, 2021.

[53] Olivier Pereira, Florentin Rochet, and Cyrille Wiedling. Formal analysis of the FIDO 1.x protocol. In *Foundations and Practice of Security: 10th International Symposium, FPS 2017, Nancy, France, October 23-25, 2017, Revised Selected Papers 10*, pages 68–82. Springer, 2018.

[54] Proxmark. Proxmark RFID Tool. https://proxmark.com, 2024.

[55] Thomas Roche, Victor Lomné, Camille Mutschler, and Laurent Imbert. A Side Journey to Titan. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 231–248, 2021.

[56] Google Safety and Security. The beginning of the end of the password. https://blog.google/technology/safety-security/the-beginning-of-the-end-of-the-password/, 2023.

[57] Fabian Schwarz, Khue Do, Gunnar Heide, Lucjan Hanzlik, and Christian Rossow. FeIDo: Recoverable FIDO2 Tokens Using Electronic IDs. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2581–2594, 2022.

[58] Michael Scott. FIDO–That Dog Won't Hunt. In *Security and Privacy in New Computing Environments: EAI Conference, SPNCE 2020*, pages 255–264. Springer, 2021.

[59] Alon Shakevsky, Eyal Ronen, and Avishai Wool. Trust Dies in Darkness: Shedding Light on Samsung's TrustZone Keymaster Design. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 251–268, 2022.

[60] Transparency Market Research. FIDO Authentication Market Forecast. https://www.transparencymarketresearch.com/fido-authentication-market.html, 2023.

[61] Enis Ulqinaku, Hala Assal, AbdelRahman Abdou, Sonia Chiasson, and Srdjan Capkun. Is Real-time Phishing Eliminated with FIDO? Social Engineering Downgrade Attacks against FIDO Protocols. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3811–3828, 2021.

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

[63] Wei Wu, Pu Wang, Lei Zhao, and Wei Jiang. An Intelligent Security Detection and Response Scheme Based on SBOM for Securing IoT Terminal devices. In *2023 IEEE 11th International Conference on Information, Communication and Networks (ICICN)*, pages 391–398, 2023.

[64] Peng Xu, Ruijie Sun, Wei Wang, Tianyang Chen, Yubo Zheng, and Hai Jin. Sdd: A trusted display of fido2 transaction confirmation without trusted execution environment. *Future Generation Computer Systems*, 125:32–40, 2021.

[65] Tarun Kumar Yadav and Kent Seamons. A Security and Usability Analysis of Local Attacks Against FIDO2. In *31th Annual Network & Distributed System Security Symposium (NDSS'24)*, 2024.

[66] Yubico. Q3 Interim Report. https://investors.yubico.com/en/wp-content/uploads/sites/2/2023/03/Q3-investor-morning-presentation-231110.pdf, 2023.

[67] Yubico. Yubico FIDO2 Python Library. https://github.com/Yubico/python-fido2, 2023.

[68] Yubico. Yubikey Manager CLI. https://github.com/Yubico/yubikey-manager, 2023.

[69] Yubico. CVE-2024-35311. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-35311, 2024.

[70] Yubico. Security Advisory YSA-2024-02 FIDO Relying Party Enumeration. https://www.yubico.com/support/security-advisories/ysa-2024-02, 2024.

[71] z4yx. FIDO Wireshark protocol dissectors over USB HID. https://gist.github.com/z4yx/218116240e2759759b239d16fed787ca, 2019.