CTRAPS: CTAP Client Impersonation and API Confusion on FIDO2

Marco Casagrande
Department of Digital Security
EURECOM
Sophia Antipolis, France
marco.casagrande@eurecom.fr

Daniele Antonioli
Department of Digital Security
EURECOM
Sophia Antipolis, France
daniele.antonioli@eurecom.fr

***Abstract*—FIDO2 is a popular 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 the communication between FIDO2 clients and authenticators. No prior work explored the CTAP Authenticator API which is a critical protocol-level attack surface as it 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 CTAP protocol-level attacks 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 users into calling unwanted CTAP APIs while thinking they are calling legitimate ones. For example, a victim thinks is authenticating to a website, when they are deleting their credentials. The CTRAPS 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. Seven of them 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' feasibility 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 discuss eight backward-compliant countermeasures to fix the attacks and their root causes. We responsibly disclosed our findings to the FIDO alliance and the affected vendors.

***Index Terms*—FIDO, CTAP, API Confusion**

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 [**?**] designed FIDO to offer a practical and scalable solution for authentication. FIDO has been widely adopted by industries and organizations including Apple, Microsoft, and the US government [**?**]. Market forecasts predict the FIDO market to rapidly grow from USD 230.6 million in 2022 to USD 598.6 million in 2031 [**?**]. Yubico, a FIDO authenticator market leader, sold more than 22 million YubiKey authenticators [**?**]. This growth will continue because of the recent industry-wide push towards single-factor passkey-based authentication [**?**], [**?**], [**?**].

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). It exposes the client to 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.

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 [**?**] performed a provable security analysis on CTAP, highlighting unauthenticated DH key exchange. In a follow-up work [**?**], they proposed an impersonation attack exploiting CTAP to register an authenticator with an arbitrary relying party. The authors in [**?**] present a Machine-in-the-Middle (MitM) attack on CTAP resulting in a privacy leak. Other works target the authenticator with fault injection and side channel attacks [**?**], [**?**].

No prior work investigated the FIDO2 *CTAP Authenticator API*. This API is a critical protocol-level attack surface as it enables the creation, management, and deletion of credentials and the administration of authenticators. FIDO2 credentials are security and privacy critical as they authorize access to popular online services, including, social media, banking, data sharing, and e-commerce. A protocol-level attack on the CTAP Authenticator API would enable access to and manipulation of any credential stored on any 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 if they hold in practice.

We fill this gap by presenting the first security and privacy assessment of the CTAP Authenticator APIs. We uncover two attack classes and eleven related attacks on CTAP that we call **CTRAPS**. The *client impersonation (CI)* attacks exploit the lack of client authentication to tamper with an authenticator. Among others, they allow factory resetting an authenticator without user interaction. The *API confusion (AC)* attacks abuse the lack of protocol API enforcements and confound a FIDO2 authenticator, a client, and a user into calling unwanted CTAP Authenticator APIs while believing they are calling legitimate ones. For instance, a user thinks to be authenticating to a website but they are instead deleting their authenticator credentials.

We consider two attacker models: a CI attacker impersonating a CTAP client and an AC attacker with a MitM position between the client and the authenticator. The adversaries perform the attacks in *proximity* or *remotely*. They do not require physical access to the authenticator, e.g., no side channel or fault injection. Moreover, they do not need to compromise the client or the authenticator, e.g., no client or authenticator malware.

The CTRAPS attacks have a *critical* and *widespread* impact on the FIDO2 ecosystem. They are critical as they violate the security, privacy, and availability of FIDO2 devices. For example, a CI or an AC attacker can factory reset an authenticator deleting all FIDO2 credentials and locking out the victim from the related service. Despite targeting CTAP, the attacks also impact FIDO2 relying parties. For example, they invalidate the non-discoverable credentials stored by the relying party. 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 whether CTAP is transported over USB, NFC, or BLE.

We isolate *eight vulnerabilities* in the CTAP specification enabling the CTRAPS attacks. Seven of them are novel within FIDO2. They include unauthenticated CTAP clients, trackable credentials, and weak authorization of (destructive) API calls. The vulnerabilities are *severe* as they affect authenticators and clients implementing CTAP v2.0, v2.1, and v2.2. We also find and disclose an implementation flaw on Yubico's authenticator firmware, allowing an attacker to leak sensitive data and track users. Yubico addressed the problem and assigned it CVE-2024-35311 [**?**].

We present `CTRAPS`, a new toolkit to experiment with CTAP and conduct the CTRAPS attacks. The toolkit has three modules: CTAP testbed, CTAP clients, and Wireshark dissectors. The testbed provides virtual clients and relying parties, enabling local testing of the attacks without the involvement of actual devices. The CTAP clients module performs the CI and AC attacks. We implemented them to work from proximity and remotely. Our CTAP clients allow the testing of the attacks on real-world authenticators and clients. For example, we release an Android app and a Proxmark3 script to test the CI attacks over NFC. The dissectors module includes an enhanced FIDO2 dissector for Wireshark praising new and useful packet information such as status codes and support for credential management.

We evaluate popular FIDO2 authenticators, clients, and relying parties. We deploy them from proximity and remotely, testing different CTAP transports (USB and NFC). We attack *six authenticators* from Yubico, Feitian, SoloKeys, and Google. One authenticator from Yubico is FIPS-compliant, meaning that it utilizes cryptographic algorithms guaranteeing strict security standards. We also exploit *ten relying parties* offering passkeys and second-factor authentication, including Microsoft, Apple, GitHub, and Facebook.

We discuss eight backward-compliant countermeasures that fix the CTRAPS attacks and their root causes. 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. The countermeasures are backward-compliant as they rely on mechanisms already available in the authenticator (e.g., PIN and LED) and do not require extra hardware (e.g., adding a 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, seven of which are new.
- We provide a toolkit to evaluate the CTAP Authenticator API and test our attacks in a virtual environment and on actual devices. We successfully conduct our attacks against six authenticators, two transports, and ten relying parties.
- We design eight backward-compliant countermeasures to fix our attacks and their root causes. We also 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 [**?**]. They acknowledged our report and shared it with their members. In May 2024, they provided feedback highlighting that the CI and AC attacks deployed by a proximity-based attacker are less scalable than remote ones. They argued on the effectiveness of CTRAPS attacks against authenticators running on a TEE. They also discussed the possible addition of our attacks to FIDO's threat model.

In December 2023, we reported our findings to the affected authenticator manufacturers (i.e., Yubico, Feitian, SoloKeys, and Google). Google confirmed our findings, assigning them priority P2 and severity S2. They responded that our attacks required a compromised FIDO client and closed the issue without resolution. We argue that Google's assessment is incorrect as our attacks do not require a compromised FIDO client. Yubico confirmed the implementation bug we found, pushed a fix in production, published a security advisory [**?**], and assigned it CVE-2024-35311 [**?**]. 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 and involved third parties. To advance open science, we open source our contributions, including the `CTRAPS` toolkit, found at https://github.com/Skiti/CTrAPs.

2. Background and System Model

We introduce FIDO2, CTAP, and our system model.

2.1. FIDO2

FIDO2 [**?**] is an open and pervasive standard for single-factor and multi-factor authentication. It is managed by the FIDO Alliance. FIDO2 has four entities: an authenticator, a client, a user, and a relying party. In a typical scenario, a user connects their 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 communication channel to a relying party and a client. Its latest version is WebAuthnL2 [**?**]. CTAP, the focus of this work, enables a secure connection between a FIDO2 authenticator and a client via the CTAP Authenticator API. For example, the `MakeCred` API registers a new credential while the `GetAssertion` API authenticates a credential.

A FIDO2 *credential* is a key pair used to sign and verify authentication challenges to authenticate a user. The digital signature is computed using standard techniques, like Elliptic Curve Digital Signature Algorithm (ECDSA). Access to the credential private key is guarded by encryption using a credential master key, which is unique to each authenticator and stored in the authenticator.

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 by the relying party and used for multi-factor authentication.

FIDO2 credentials are associated with a credential identifier (CredId), a relying party identifier (RpId), and a user identifier (UserId). The CredId uniquely identifies a FIDO2 credential and is derived from the credential master key stored in the authenticator. When FIDO2 clients authenticate a credential, they must know its associated CredId. The RpId indicates the relying party with which the credential was registered. It is public as it corresponds to the base domain of the relying party (e.g., `login.microsoft.com`).

The UserId represents the user's online account within the relying party's service. The relying party assigns a random UserId to the user during account registration, and it is shared across all FIDO credentials associated with that user. The optional FIDO2 *CredBlob* extension allows a relying party to store additional metadata inside a credential.

2.2. CTAP

The Client-to-Authenticator Protocol (CTAP) is a core part of the FIDO2 standard, alongside WebAuthn. It is an application-layer protocol that defines the communication between a FIDO client and an authenticator. CTAP has considerably evolved since its inception. CTAP1, also known as FIDO U2F (Universal 2nd Factor), introduced a second-factor authentication mechanism to combat phishing. CTAP2.0 maintains backward compatibility with CTAP1 while introducing passwordless (single-factor) authentication. CTAP2.1 [**?**] adds the credential protection policy, discoverable credential management (i.e., the `CredMgmt` API), and biometric authentication. CTAP2.2 [**?**], the latest CTAP version still considered a draft, supports hybrid authenticators and QR codes.

TABLE 1. CTAP AUTHENTICATOR APIS WITH THEIR *UV* AND *UP* AUTHORIZATION REQUIREMENTS, AND SUPPORT FOR SUBCOMMANDS. YES[1]: DEPENDS ON THE CLIENT AND RELYING PARTY CONFIGURATION, YES[2]: DEPENDS ON API SUBCOMMAND.

CTAP API	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

CTAP relies on two user authorization mechanisms to secure API calls from the client: (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 **??** shows the seven *CTAP Authenticator APIs* studied in this paper and their *UV* and *UP* requirements:

- `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.
- `CM`: `CredMgmt` enumerates, modifies, and deletes the authenticator's discoverable credentials.
- `CP`: `ClientPin` handles *UV* based on a user PIN to be submitted via the client's UI.
- `Re`: `Reset` wipes all discoverable and non-discoverable credentials and generates a new master key.
- `Se`: `Selection` selects an authenticator to operate among the available ones.
- `GI`: `GetInfo` returns the authenticator's details, like manufacturer, transports, extensions, and settings.

The `GetAssertion`, `CredMgmt`, and `ClientPin` APIs have API subcommands. For example, `CredMgmt(GetCredsData)` returns the number of stored discoverable credentials and `CredMgmt(DelCreds)` deletes all discoverable credentials. Some API subcommands, compared to their original API, have more relaxed requirements. For instance, `ClientPin(KeyAgreement)` does not require *UV*.

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*.

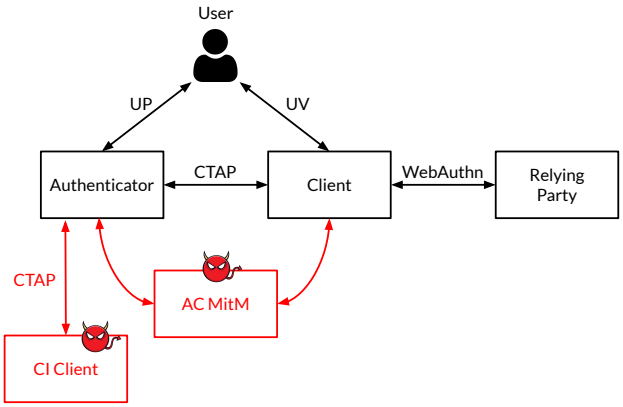

Figure 1. **CTRAPS threat model.** The user authenticates to the relying party using a client (e.g., browser) and an authenticator (hardware dongle). The user when needed grants UP by pressing a button on to the authenticator and UV by submitting a PIN to the client. We study two attacker models: (i)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

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*, resulting in weak privacy protection.

2.3. System Model

We adopt the official FIDO2 system model [**?**]. Figure **??** shows the system model's four entities: authenticator, client, relying party, and user. 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 protocol versions). Next, we describe each entity. The attacker models are presented in Sections **??** and **??** as they are specific to the CI and AC attacks.

Authenticator. The authenticator is a FIDO2 authenticator: a user device 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. The authenticator supports FIDO2's *UP* and *UV* 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 [**?**] or iOS [**?**], or a command line tool like the Yubico CLI [**?**].

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. The relying party stores non-discoverable credentials, and user and credential identifiers. The relying party communicates with the client using TLS. 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 that runs the FIDO2 client, e.g., a YubiKey dongle and a laptop. They utilize their authenticator to register FIDO2 credentials and authenticate to the associated relying party. To do so, they connect their authenticator to the client and provide *UV* and *UP*, if necessary. The user manages the authenticator via the client, without connecting to a relying party. For example, they can check their discoverable credentials and change the authenticator's PIN.

3. CTRAPS Client Impersonation Attacks

The *CTRAPS CI attacks* target an authenticator while spoofing a client to perform CTAP API calls without user authorization. CI 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 CTAP vulnerabilities we found (described in Section **??**). For instance, the absence of CTAP client authentication facilitates impersonation, the use of NFC transport allows to bypass *UP*, and the lack of *UV* when calling `Reset` enables unauthorized factory resets.

The attacks advance the state of the art in FIDO2's security and privacy by introducing client impersonation. This is a new class of attacks previously unseen in FIDO2, as shown in Table **??**. The CI attacks require limited or no user interaction, depending on the CTAP transport. For example, by using NFC, they bypass *UP*, leading to zero-click attacks. The CI attacks also involve *no client compromise*, being deployed from a client owned by the attacker. Next, we introduce the CI attacker model and describe the attacks.

3.1. CI Attacker Model

The CI attacker model assumes an attacker impersonating a CTAP client to the victim's authenticator, referenced as CI Client in Figure **??**. The attacker is in proximity of the victim's authenticator, or can remotely connect to it. They have no physical access to and do not tamper with the victim's client and authenticator. They do *not* install malware on the victim's device running the FIDO2 client.

The CI attacker model maps to several relevant attack scenarios. For example, they can approach a target authenticator over NFC while impersonating a client (e.g., via a smartphone or a Proxmark), place a malicious NFC device in a place where a user might touch it with an authenticator (e.g., under a table), They can also communicate with the victim's authenticator using a compromised hardware device, such as a USB hub that connects the user's machine and the authenticator, or virtual USB peripheral, through a setup similar to [**?**].

3.2. CI Attacks Description

We describe the four CI attacks, which we label 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 **??**. The attacker connects to the

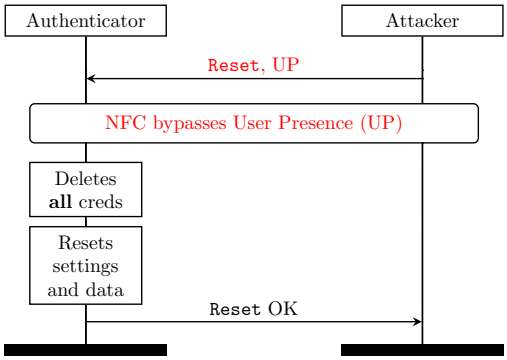

Figure 2. **CI_1 attack.** Factory reset authenticator via `Reset`. While in NFC range, the attacker calls the `Reset` API. Over NFC, the authenticator skips *UP* and instantly factory resets, deleting all of its discoverable and non-discoverable credentials.

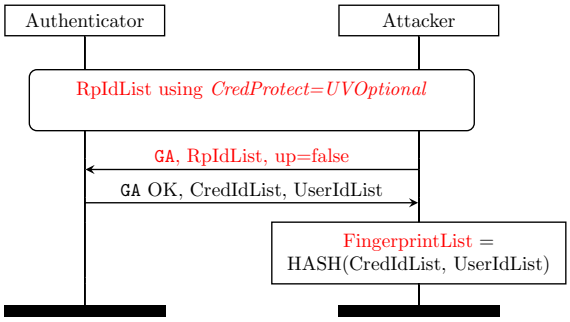

Figure 3. **CI_2 attack.** Track user from credentials via `GetAssertion`. The attacker connects to the authenticator and calls the `GetAssertion` API (`GA` in the figure). They skip *UV* by targeting relying parties using the weak and default *CredProtect* default policy and skip *UP* by passing *up=false*. The authenticator returns a list of credential and user identifiers, used by the attacker to fingerprint the authenticator and track the user.

authenticator and, without authenticating, issues a factory reset command (which requires *UP*). Over USB, the attack requires one click (*UP*) and the authenticator having been plugged into the USB port within the last ten seconds. Over NFC, the attacker achieves zero-click reset by exploiting a CTAP quirk intended to enhance usability. That is, NFC communication inherently implies user presence, allowing *UP* to be bypassed. The execution of the factory reset wipes out all credentials, even the non-discoverable ones stored by the relying party, 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` for authentication, the attacker exploits it to leak identifying data and track the user, as shown in Figure **??**. CI_2 requires a pre-determined list of RpId for which the attacker aims to leak credentials. This is straightforward, as this information is publicly accessible. Although the `GetAssertion` API requires both *UV* and *UP*, the attacker can circumvent both authorizations, resulting in a *zero-click* data leak and enabling user tracking. They bypass *UP* by issuing a `GetAssertion` command containing the *up=false* option. They bypass *UV* by only targeting relying parties that register credentials using the weak and default *CredProtect=UVOptional* policy, such as Microsoft and Apple. Executing `GetAssertion` returns a list of credential and user identifiers. These identifiers can be used to fingerprint the user and to track them over multiple connections by performing CI_2 each time and looking for matching fingerprints. CI_2 also works on credentials protected by stronger policies (i.e., *CredProtect=UVRequired* and *CredProtect=UVOptionalWithCredIDList*), but requires *UV* or knowledge of the credential identifiers.

CI_3: Force authenticator lockout. In CI_3, the attacker abuses the `ClientPin` API, protecting the authenticator from PIN brute-forcing, to lock the authenticator or even force a factory reset. They submit to the authenticator several wrong PIN guesses in a row via the `ClientPin(GetPinToken)` subcommand. After three wrong guesses, the authenticator enters a soft lock mode preventing 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 of failed PIN attempts (CTAP mandates eight), the authenticator enters a hard lock mode 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's technical details. This attack can be used as a stepping stone to more advanced attacks, to profile the user and track them in future connections, and to assess whether the authenticator is vulnerable to an implementation-specific attack like [**?**]. 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

The *CTRAPS AC attacks* take advantage of a novel attack technique for FIDO, which we refer to as *API confusion*. API confusion tricks a client, an authenticator, and their user into calling a CTAP Authenticator API while they think they are calling a different one. The called API has the same or lower *UV* and *UP* requirements of the intended API. For example, AC attacks can erase FIDO2 credentials, including passkeys, lock the user out of their authenticator, and track them.

AC is effective as it does not require social engineering [**?**] or other deception techniques [**?**] to trick the user into calling an unwanted API. The user cannot detect an API confusion because it requires expected *UV* or *UP* actions. The AC attacks exploit the eight protocol-level vulnerabilities we outline in Section **??**. For instance, the absence of authenticator feedback during API calls grants stealthiness and the use of static credential and user identifiers enables user tracking.

No prior work considered the AC attack vector for FIDO2, as shown in Table **??**. Existing attacks on FIDO include MitM on the Diffie-Hellman key exchange, CTAP traffic eavesdropping, U2F impersonation, physical access, and side channel attacks on the authenticator. Moreover, the AC attacks target the entire CTAP Authenticator API surface, whereas previous research only focused on `Clie`

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. ✓[1]: PROXIMITY-BASED ATTACKER, ✓[2]: DEFAULT *CredProtect=UVOptional* IF CREDENTIAL PROTECTION IS ENABLED, N/A: NOT APPLICABLE.

	CM	Re	GA	MC	CP	Se	GI
CM	n/a	✓[1]	✓	✓[1]	✓	✓	✓
Re	n/a	n/a	✓[2]	n/a	✓	✓	✓
GA	✓	✓	n/a	✓	✓	✓	✓
MC	✓	✓	✓	n/a	✓	✓	✓
CP	✓	✓[1]	✓	✓[1]	n/a	✓	✓
Se	n/a	✓	✓[2]	n/a	✓	n/a	✓
GI	n/a	✓[1]	✓[2]	✓	✓	✓	n/a
Total	3	6	6	4	6	6	6

ntPin and MakeCred. Next, we will introduce the AC attacker model and attacks.

4.1. AC Attacker Model

The AC attacker model assumes a MitM attacker between the authenticator and the client, referenced as AC MitM in Figure **??**. The attacker is either in proximity to the authenticator and the client (e.g., an NFC skimmer) or can contact them from remote (e.g., a remotely controllable USB hub). They are unable to modify the authenticator's firmware or compromise a legitimate FIDO2 client and relying party. They have no physical access to the authenticator.

An AC attacker model has several associated real-world attack scenarios. For example, they can get a MitM position over NFC interposing an NFC skimmer between the client and the authenticator. They can achieve a MitM position over USB with setups such as those discussed in [**?**] and [**?**]. For instance, the attacker can remotely compromise a USB device connected to the user's device running the FIDO2 client, such as a USB hub that routes traffic between other USB peripherals.

Alternatively, they can gain privileged access via techniques like UACMe [**?**] and then leverage USBPcap to USB MitM a victim's Windows machine running the FIDO2 client. The attacker can also install on the user's machine a malicious app exploiting libraries that provide access to USB HID traffic. We implement this attack scenario in Section **??** by developing an Electron app that mimics a MitM attacker using the *node-hid* module.

4.2. 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. This action 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, 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 **??**.
4) If required by API A, the attacker obtains *UP* from the user, unable to realize they are under attack. The attacker can only obtain *UP* once, as multiple requests would alarm the user. This step is bypassed whenever NFC 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.

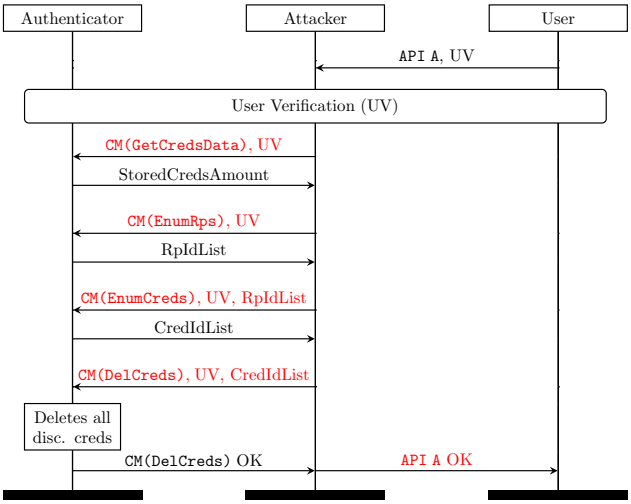

Figure 4. **AC_1 attack.** Delete discoverable credentials attack with proximity. The user intends to call API A, requiring *UV* but not necessarily *UP*. For example, GetAssertion, ClientPin, or MakeCred. The attacker obtains *UV* from the unsuspecting user. Instead of API A, they call CredMgmt (CM in the figure). They execute four CredMgmt subcommands which list and then delete all discoverable credentials on the authenticator.

The AC strategy is effective on *7* CTAP Authenticator APIS and provides *49* ways to confound the victim as shown in Table **??**. 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.

4.3. AC Attacks Description

We describe seven AC attacks labeled AC_1, AC_2, AC_3, AC_4, AC_5, AC_6, and AC_7. AC_1 exploits all possible ways to call CM, AC_2 does this with Re, and so on.

AC_1: Delete discoverable credentials. In AC_1, the attacker abuses the `CredMgmt` API to delete all discoverable credentials stored on the authenticator, as shown in Figure **??**. 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, they check the existence of discoverable credentials to erase (StoredCredsAmount) via `CredMgmt(GetCredsMetadata)`. Second, they retrieve the list of relying parties stored on the authenticator (RpIdList) via `CredMgmt(EnumRps)`. Third, they use RpIdList to retrieve the list of stored credential identifiers (CredIdList) via `CredMgmt(EnumCreds)`. Fourth, they use CredIdList to delete all discoverable credentials via `CredMgmt(DelCreds)`. Finally, they falsely return `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. They can confound `MakeCred`, *CredMgmt*, and *ClientPin* into a `GetAssertion` call, if they want to access credentials protected by the *CredProtect=UVRequired* or *CredProtect=UVOptionalWithCredIDList* policies. Additionally, the attacker can also confound `Reset`, `Selection`, and `GetInfo` if they only wants to access credentials protected by the weak *CredProtect=UVOptional* default policy.

AC_4: Fill authenticator's credential storage. In AC_4, the attacker repeatedly calls `MakeCred` to register new discoverable credentials, until the authenticator's credential storage is full. They exploit the *rk=true* option to enforce the generation of discoverable credentials over non-discoverable ones. A filled 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., they do not need *UV*). Consequently, they can confound any API call into a `ClientPin` call, as they do not need authorization.

AC_6: Authenticator DoS. In AC_6, the attacker calls `Selection` to trigger an unwanted *UP* check, keeping the authenticator busy and denying 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), they can prolong the attack.

AC_7: Profile authenticator. In AC_7, the attacker invokes `GetInfo` to retrieve the authenticator's details. Then, similar to CI_4, they use this information as a stepping stone to other attacks, tracks the user, or checks whether the authenticator is vulnerable to implementation-specific attacks [**?**]. Not requiring *UV* or *UP*, the attacker can confound any API call into a `GetInfo` call.

5. CTRAPS Vulnerabilities and Impact

We present the root causes of the CTRAPS attacks and discuss our attacks' impact on the FIDO2 ecosystem.

5.1. Vulnerabilities

The CI and AC attacks are enabled by *eight vulnerabilities* we discovered in the CTAP specification. Seven of them are novel, whereas V2 was discussed in [**?**]. Regardless, this is the first work exploiting V2 via AC. Now, we will describe the vulnerabilities and map them to the CI and AC attacks.

V1: Unauthenticated CTAP client. The CTAP client does not authenticate to the authenticator. FIDO2 clients (and, by extension, CTAP clients) lack an identity, preventing the authenticator from distinguishing an official client developed by its manufacturer and a third-party client. As a result, the authenticator trusts any connecting client, including spoofed ones.

V2: No authenticator feedback about API calls. The authenticator does not provide visual feedback to the user when invoking the CTAP Authenticator API. As a result, the user is unable to verify whether the intended API has been correctly called (or has been confounded instead), or which API utilized the most recent *UV* and *UP* authorizations granted.

V3: NFC range provides *UP*. Authenticators within the NFC range of a FIDO2 client automatically obtain *UP* without the user pressing a button on the authenticator. Bypassing *UP* means that `MakeCred`, `GetAssertion`, and `Reset` are solely protected by *UV*, or they now require no authorization at all.

V4: Weak access control to destructive APIs. 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. Whenever the user grants *UV* for a non-destructive operation, they unknowingly over-privilege the client, enabling destructive operations as well. For example, the user grants *UV* to authenticate (non-destructive), but an AC attacker exploits the over-privileged access to instead factory reset 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 improves to the user's fingerprint.

V6: `Reset` does not verify the user. Despite being destructive, the `Reset` API *only* requires *UP*, which does not verify that the person operating the device is the owner. Anyone nearby the authenticator can obtain *UP* by pressing its button or by being within NFC range.

V7: `CredMgmt` allows to delete multiple credentials at once. The `CredMgmt` API *only* requires *UV*, meaning that no user interaction is needed to delete discoverable credentials. This allows multiple discoverable credentials to be deleted without first alerting the user or asking for a confirmation.

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	✓	✓	✗	✗	✗	✗	✗	✗

V8: `Selection` is usable for DoS. The `Selection` API can be used to DoS an authenticator by continuously prompting it for *UP* checks.

Table **??** maps the eight vulnerabilities (columns) to the eleven CTRAPS attacks (rows). V1 is needed to perform all CI and AC attacks, as it allows an untrusted client or a MitM attacker to connect to the authenticator without authenticating to it. V2 provides stealthiness to AC attacks because, without visual feedback, the user cannot confirm whether the API they are calling is being confounded or not. Due to V3, CI_1 and CI_2 over NCF require zero clicks instead of one (*UP*). V3 also unlocks several new API confusion combinations, such as `GetInfo` into `Reset`.

Moreover, V4 allows to perform the (destructive) CI_2, CI_3, AC_1, AC_2, and AC_5 attacks even when the user calls a non-destructive API, such as `Selection`. V5 enables the usage of identifiers as persistent fingerprints, resulting in two user tracking attacks (CI_2 and AC_3). V6 allows for a zero-click factory reset attack (CI_1) over NFC. V7 allows for a one-click credential deletion attack (AC_1). V8 enables a persistent and reliable DoS attack on the authenticator (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 presented in Section **??**.

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. Hence, they threaten millions of authenticators in the wild, their users, and the relying parties. Being at the protocol-level, the root causes are challenging to fix, as most authenticators, including YubiKeys, do not support firmware updates.

The CTRAPS attacks are *practical* and *low-cost*, requiring minimal equipment, such as a smartphone. Their outcome is realistic and involves limited or no user interaction, as shown in the video demonstrations available in the CTRAPS GitHub repository. 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.

5.3. Comparison with prior FIDO attacks

Table **??** compares our attacks with previous attacks on FIDO. The CI attacks are the first client impersonation attacks targeting FIDO2 (CTAP2+), while the AC attacks utilize API confusion, a novel attack strategy for FIDO2. Prior research on CTAP evaluated only the `ClientPin` and `MakeCred` APIs. Instead, our attacks target the entire Authenticator API, regardless of the CTAP transport, covering a broader surface.

The CI attacks have low complexity, as they do not require a compromised client or prior knowledge of user secrets (e.g., credential identifiers). The AC attacks have a moderate complexity, as they require a MitM position. Both CI and AC attacks have a high impact as they can, for example, destroy credentials and track users.

The existing FIDO attacks with a high impact also require a strong attacker model, such as physical access or malware installed on the user's browser or device. In contrast, the CTRAPS attacks employ a *weaker* attacker model, i.e., client impersonation and MitM attacker, while still achieving high impact with low-to-mid complexity.

6. Implementation

In this section, we present `CTRAPS`, a novel toolkit that implements the CTRAPS attacks and enables experimentation with CTAP. The toolkit has *three* modules: a CTAP testbed (Section **??**), four customizable CTAP clients (Section **??**), and an enhanced FIDO2 Wireshark dissector (Section **??**).

The CTAP testbed and the Electron app CTAP client need the user's authorization to connect and communicate with the authenticator. Linux requires adding extra `udev` rules, macOS asks to accept a notification on the screen, and Windows needs admin privileges. This limitation is expected, as it is also present in FIDO2 apps released by authenticator manufacturers, such as the Yubico Authenticator App and the Feitian Authenticator Tool. Now, we will describe the implementation of each module and highlight their novelties.

6.1. CTAP Testbed

Our CTAP testbed includes a virtual WebAuthn relying party and a virtual WebAuthn/CTAP client. The testbed can test real authenticators without having to tamper with actual credentials and also launch the CTRAPS attacks. Our relying party and client extend the Yubico open-source Python library for FIDO2 called `python-fido2` [**?**].

Virtual relying party. The virtual relying party is implemented as a customizable WebAuthn server. It includes standard relying party templates and fast customization of the server's parameters. For example, we implemented a template imitating a Microsoft relying party, including its FIDO2 identifier (i.e., *login.microsoft.com*). The virtual relying party is useful to quickly test real authenticators

against CI and AC attacks. For example, we can automatically register credentials with different protection policies on the authenticator.

Virtual client. The virtual client offers a convenient CTAP API, offering low-level access to any CTAP message. It can send CTAP commands in any order, or issue custom and malformed payloads. It can be configured with different CTAP authorization requirements, authentication challenges, and origins.

6.2. CTAP Clients

We developed four custom CTAP clients: an Android app performing proximity CI over NFC, an Android app performing remote CI over NFC, a Proxmark3 script that executes proximity CI over NFC, and an Electron app simulating a MitM attacker to test remote AC over USB. We released in the CTRAPS GitHub repository five video demonstrations, showing how to deploy the CTRAPS attacks on real authenticators using our clients.

Android app for proximity CI over NFC. We implemented the proximity CI attacks using an Android app which impersonates a FIDO2 client over NFC. The app runs on a device owned by the attacker and targets any authenticator that comes within the NFC range. For example, it can perform CI_2 to leak identifiers and track the user.

Android app for remote CI over NFC. We utilized an Android app to implement the remote CI attacks over NFC. The app is installed on a device owned by the victim. It spoofs a legitimate NFC app, enticing the user to scan their authenticator (e.g., by asking for FIDO2 authentication). The attacker can connect to the app and manage the CTAP connection with the authenticator. The app does not need root privileges and asks at runtime for the dangerous `android.permission.NFC`, required to gain access to the `android.nfc` [**?**] API. However, this is not a concern, as the app is not trying to conceal its NFC capabilities. The app also needs the standard install-time `android.permission.INTERNET` to exfiltrate the data collected through CI_2 and CI_4.

Proxmark3 for proximity CI over NFC. We implemented the proximity CI attacks using the Proxmark3 [**?**], an open-source and programmable development kit for NFC (RFID). We wrote a Lua script using the Proxmark3 ISO14443 Type A module (i.e., `read14a`) to communicate to the authenticator via CTAP-compliant APDUs. By equipping the Proxmark3 with a long-range high-frequency antenna, we were able to extend its reach. 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.

Electron app to simulate AC over USB. We developed an Electron app that simulates a MitM attacker. The app uses the *node-hid* module to access the USB HID traffic, gaining a MitM position between a FIDO client and an authenticator communicating over USB. The app scans for local HID devices and identifies the authenticators from their properties (e.g., the product and manufacturer fields). Then, it connects and sends binary data over USB to the authenticator. The Electron app is compatible with Windows, macOS, and Linux.

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

6.3. FIDO2 Wireshark Dissector

We extended an unofficial Wireshark FIDO2 dissector found in [**?**]. We add valuable features, such as support for the `CredMgmt` API. We include parsers for `WAITING` and `PROCESSING` keepalive status codes that identify when authenticators are unavailable. We parse the authenticator's capabilities in the `CTAPHID_INIT` message, which are useful for testing 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. The FIDO2 dissector is included in our toolkit as a Lua script (i.e., `fido2-dissectors.lua`).

7. Evaluation

We evaluated our eleven attacks against *six* popular and recent authenticators from Yubico, Feitian, SoloKeys, and Google. We also tested *ten* widely used relying parties, including Microsoft, Apple, GitHub, and Facebook. Next, we will present our evaluation setup and results.

7.1. Setup

Authenticators. We evaluate *six* popular FIDO2 authenticators. Table **??** 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. Unfortunately, we could 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, and, as such, it should provide high security guarantees. We ran OpenSK on an NRF52840 dongle, but any board supporting OpenSK would have worked.

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 of them offer

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's credential storage, **AC**$_5$: Force authenticator lockout, **AC**$_6$**:** Authenticator DoS, **AC**$_7$: Profile authenticator.

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. As a consequence, erasing a single credential has a widespread effect on multiple online services.

CTRAPS toolkit. We evaluated the CI and AC attacks using the tools included in the CTRAPS toolkit. We installed our two Android apps, found in the CTAP clients module, on a Google Pixel 2 (OS: Android 11), a RealMe 11 Pro (OS: Android 14), and a Xiaomi Redmi Plus 5 (OS: Android 8.1). We used a Proxmark3 RDV4 with a long-range high-frequency antenna to deploy the proximity CI attack with an extended NFC range. We tested our 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).

7.2. Authenticators Results

Table **??** shows the evaluation results for the CI and AC attacks on six FIDO2 authenticators. All six of them were vulnerable to the CTRAPS attacks, even the FIPS-compliant YubiKey. 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. However, AC_6 does not apply to the four authenticators which do not support the `Selection` API.

We also found a `CredMgmt` implementation vulnerability on the YubiKey 5 and YubiKey 5 FIPS, which improperly handles the authenticator's state for `CredMgmt`. They allow the client to call `CredMgmt(EnumRpsGetNextRp)` without invoking `CredMgmt(EnumRpsBegin)` first, which is an illegal state. We exploit this flaw to achieve a zero-click *leak of relying party names*. Our attack calls `CredMgmt(EnumRpsGetNextRp)` to reveal the names of all the relying parties, stored on the authenticator, except one. This attack bypasses *UV* and works regardless of the *CredProtect* policy. We reported it to Yubico, which assigned it CVE-2024-35311 and addressed it in their latest firmware. However, since YubiKeys do not support firmware updates, this fix is only available to newer authenticators, leaving older ones vulnerable.

The CI and AC attacks over NFC have a maximum range of two centimeters on a smartphone. The Proxmark3 built-in antenna also achieved the same range, which we could extend to six and a half centimeters by attaching a long-range antenna. Prior work demonstrated that, with specialized equipment, the NFC range can be extended up to 50 centimeters [**?**].

We also tested *combinations* of CI and AC attacks, to develop more advanced variants. For example, we found multiple ways to enhance our user tracking attacks (CI_2 and AC_3). The attacker can refine the user's fingerprint using AC_7 or register new credentials, with metadata of their choice, on the authenticator using AC_4.

7.3. Relying Parties Results

As shown in Table **??**, we tested eight relying parties supporting discoverable credentials and two employing non-discoverable credentials. Our evaluation includes relying parties because our attacks affect them, even though we do not utilize WebAuthn. For example, AC_1 deletes discoverable credentials, causing the user to lose access to their online account. Although relying parties using non-discoverable credentials are not vulnerable to AC_1 and AC_4, they remain open to our factory reset, user tracking, and DoS attacks.

CI_1, AC_1, and AC_2 block web authentication to the relying party by deleting the user's FIDO2 credentials. CI_2 and AC_3 utilize the user identifiers generated by the relying party to track users. CI_3, AC_4, AC_5, and AC_6 prevent relying parties from communicating with the authenticator. Among the relying parties supporting discoverable credentials, we found that only Microsoft and Apple employ the weak *CredProtect=UVOptional* policy. This policy allows to bypass *UV* when accessing credentials. As a result, an attacker can deploy a zero-click variant of CI_2 and AC_3 to track users through their Microsoft and Apple credentials.

8. Discussion

We discuss the countermeasures to fix the CTRAPS attacks and the issues we found in the FIDO reference threat model.

8.1. Countermeasures

We discuss *eight* backward-compliant countermeasures fixing the eleven CTRAPS attacks and their associated eight vulnerabilities. Each countermeasure addresses

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 DOES NOT SUPPORT DISCOVERABLE CREDENTIALS.

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

a specific vulnerability (e.g., C1 fixes V1) and helps reduce the CTAP attack surface. Although we have not implemented these countermeasures, we designed them to be implementable as amendments to the FIDO2 standard or as FIDO2 extensions. Next, we will describe each countermeasure.

C1: Trusted CTAP clients. We address V1 by recommending that the FIDO Alliance provides a list of trusted CTAP clients. The FIDO ecosystem offers several certifications, including the FIDO Functional Certification [**?**] which attests to the *interoperability* of clients, servers, and authenticators. We suggest extending this certification to also 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 [**?**].

C2: Authenticator visual feedback. We address V2 by requiring the authenticator to provide the user with visual feedback regarding the API that was called. 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 step.

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 V4 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 [**?**]).

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 their account, replacing the old one. However, the user cannot change the UserId, which is determined by the relying party 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 authorize 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, to avoid deleting multiple credentials with a single API call.

C8: Rate limiting `Selection` calls. We address V8 by enforcing temporal rate limiting on `Selection` 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 of the countermeasures. The deployment of C1 and C8 does not affect usability. C2 requires the user to notice the authenticator's visual feedback. Implementing C3 introduces an additional *UP* check each time the client connects to the authenticator over NFC, which is costly. C4 forces the user to remember a second PIN. C5 introduces one *UV* and *UP* check each time the credential and user identifiers are being rotated out, e.g., once per month. The implementation of C6 would require PIN verification for every call to `Reset`. C7 would add a button press each time the `CredMgmt` API is invoked.

Authenticator with a display. We do not consider adding a display to a roaming authenticator in the list of countermeasures as is not backward-compliant and would require to recall all vulnerable authenticators without a display. Moreover, for newer authenticators, it entails significant hardware and software modifications, such as adding a secure display, a secure display controller firmware, and a battery, that would introduce usability, performance, and cost issues.

8.2. FIDO Reference Threat Model Issues

The FIDO Alliance released a reference threat model [**?**] outlining security assumptions, goals, and threats against clients, authenticators, and relying parties. Although non-normative, it is the only official source

TABLE 7. COMPARING PRIOR ATTACKS ON FIDO WITH THE CTRAPS ATTACKS. WE ASSIGN EACH ATTACK A COMPLEXITY AND AN IMPACT. FOR EXAMPLE, THE COMPLEXITY FOR A MITM IS MID, WHEREAS WE CONSIDER SPOOFING A CLIENT AS LOW COMPLEXITY. SIMILARLY, HIJACKING A SESSION HAS A MID IMPACT, WHILE PERMANENTLY DESTROYING CREDENTIALS CARRIES A HIGH IMPACT.

Attack	Class	Protocol	Transp	Surface	Impl	Reqs	Complex	Impact
CTAP MitM [?]	DH MitM	CTAP2.0	All	ClientPin	✗	MitM	Mid	Mid
Privacy leak [?]	Eavesdropping	CTAP2.0	All	MakeCred	✗	n/a	Low	Low
Auth rebind [?]	Auth rebind	WebAuthn	All	Creds.create	✗	n/a	High	High
Parallel session [?]	Session hijack	WebAuthn	All	Creds.get	✗	n/a	Mid	Mid
ECDSA extract [?]	Side channel	n/a	n/a	NXP A7005	✗	Phy access	High	High
Titan sign in [?]	Relay	U2F	BLE	Google acc	✓	Proximity	Mid	Mid
Evil maid [?]	Phy access	n/a	n/a	Auth TEE	✗	Phy access	High	High
Auth MitM [?]	DH MitM	CTAP2.0/2.1	USB	ClientPin	✓	Mal browser	Mid	Mid
Web MitM [?]	Session hijack	WebAuthn	USB	Creds.get	✓	Mal browser	Mid	Mid
Rogue key [?]	Auth rebind	WebAuthn	USB	Creds.create	✓	Mal browser	Mid	High
FIDOLA [?]	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

detailing the FIDO threat model. After studying it and working on the CTRAPS attacks, we identified *three issues* (IS1, IS2, and IS3) with the FIDO reference threat model:

IS1: Unclear security boundaries. The threat model presents six broad security assumptions, but breaks them when discussing threats. For example, SA-4 states that the user device and applications involved in a FIDO2 operation act as trustworthy agents of the user. This implies that the client (e.g., browser or mobile app) must be inherently trusted. However, at the same time, the threat model includes threats that violate SA-4, such as *T-1.2.1: FIDO client corruption*. This leads to unclear security boundaries, making it difficult to differentiate trusted components from ones that could be compromised.

IS2: Missing proximity threats. Although FIDO supports proximity transports like NFC and BLE, its threat model groups proximity-based threats together with physical access ones. However, these threats differ in key aspects. For example, proximity threats have a range. Consequently, our proximity CI and AC attacks do not fit within this threat model.

IS3: Security goals are narrow. The security goals of the threat model are based on [?] (2006) and [?] (2012). These two research papers outlined the security goals of an ideal authentication scheme, focusing on password-based schemes and web authentication. As a result, the security goals are too narrow to capture the complexities of the FIDO ecosystem. For example, there are no security goals for the Authenticator API, that could address the AC attacks, or discoverable credentials, that are relevant to AC_1, CI_1, and AC_2.

9. Related Work

We present related work on FIDO, covering existing attacks, formal analysis, FIDO extensions and enhancements, usability studies, and surveys.

Attacks on FIDO(2). Researchers found attacks on older FIDO versions (UAF, U2F), such as authenticator rebinding, parallel sessions, and multi-user attacks [?], [?], USB HID man-in-the-middle attacks [?], BLE pairing [?], relying party public key substitution [?], bypassing push-based 2FA [?], real-time phishing [?], and side channel attacks [?], [?]. FIDO2 was also found vulnerable to deception [?], misbinding [?], physical [?], [?], and rogue key or impersonation attacks [?], [?]. Moreover, researchers found issues on lower layers trusted by FIDO2, including an IV reuse on the Samsung Keystore [?]. No prior attack investigated *client impersonation* or *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) [?], [?], FIDO2 (including its privacy, revocation, attestation, and post-quantum crypto) [?], [?], [?], [?]. Yubico proposed a key recovery mechanism based on a backup authenticator that was proven secure using the asynchronous remote key generation (ARKG) primitive [?]. Existing research on formal analysis is *not* covering our CI and AC attacks.

Extensions. FIDO supports extensions to add optional features in a backward-compliant way. For instance, FeIDO [?] 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 [?]. 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 [?], the authors present a hybrid post-quantum signature scheme for FIDO2 and tested it using OpenSK [?] (which we exploit in this work). In [?], the authors propose a global key revocation procedure for WebAuthn that revokes credentials without communicating to each individual relying party WebAuthn server. True2F [?] 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 [**?**], [**?**], [**?**], [**?**], FIDO2 roaming authenticators [**?**], [**?**], passkeys [**?**], and cross-site 2FA [**?**]. Our paper is *orthogonal* to usability studies.

Surveys. There are several FIDO survey papers. In [**?**] the authors describe the evolution of FIDO protocols, security requirements, and adoption factors. In [**?**], 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.

10. Conclusion

No prior work assessed the CTAP Authenticator API, a critical surface 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 that abuse it. The CI attacks spoof a CTAP client to a target authenticator. The AC attacks leverage a MitM position to change CTAP API calls made by the user to an API desired by the attacker while stealing their authorizations. They utilize API confusion, a novel attack strategy within FIDO2.

We uncover eleven CI and AC attacks, impacting millions of FIDO2 users. They can be deployed by a proximity-based or a remote attacker. For example, they delete FIDO2 credentials and master keys (security breach) and track users through their credentials (privacy breach). Our 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. The CTRAPS attacks are low-cost, as they do not require specialized equipment, and stealthy, as they do not trigger unexpected user interactions.

We develop the `CTRAPS` toolkit to test our attacks with a cheap setup. It includes a CTAP testbed with a virtual relying party and a virtual client, four CTAP clients that deploy our attacks (e.g., Android apps and Proxmark3 scripts), and an enhanced Wireshark dissector for CTAP. We successfully exploit six authenticators and ten relying parties from leading FIDO2 players such as Yubico, Feitian, Google, Microsoft, and Apple. We design eight 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 [**?**], [**?**]: (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., CI_2, AC_1, 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).

Acknowledgment

Work funded by the European Union under grant agreement no. 101070008 (ORSHIN project). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them. Moreover, it has been partially supported by the French National Research Agency under the France 2030 label (NF-HiSec ANR-22-PEFT-0009) and the Apricot/ENCOPIA ANR MESRI-BMBF project (ANR-20-CYAL-0001).

Appendix A. Data Availability

All experiments in this study were conducted ethically and solely on authenticators and accounts under our control, with no involvement of third-party personal data. To support reproducibility and advance open science, we are releasing our artifacts, including the CTRAPS toolkit. Our toolkit is securely hosted in a repository at https://github.com/Skiti/CTrAPs. We already responsibly disclosed our findings to all affected parties, including the FIDO Alliance, and we respected their timeline.