A Security and Usability Analysis of Local Attacks Against FIDO2

Tarun Kumar Yadav, Kent Seamons
Brigham Young University
tarun141@byu.edu, seamons@cs.byu.edu

***Abstract*—The FIDO2 protocol aims to strengthen or replace password authentication using public-key cryptography. FIDO2 has primarily focused on defending against attacks from afar by remote attackers that compromise a password or attempt to phish the user. In this paper, we explore threats from local attacks on FIDO2 that have received less attention—a malicious browser extension or cross-site scripting (XSS), and attackers gaining physical access to an HSK. Our systematic analysis of current implementations of FIDO2 reveals four underlying flaws, and we demonstrate the feasibility of seven attacks that exploit those flaws. The flaws include (1) Lack of confidentiality/integrity of FIDO2 messages accessible to browser extensions, (2) Broken clone detection algorithm, (3) Potential for user misunderstanding from social engineering and notification/error messages, and (4) Cookie life cycle. We build malicious browser extensions and demonstrate the attacks on ten popular web servers that use FIDO2. We also show that many browser extensions have sufficient permissions to conduct the attacks if they were compromised. A static and dynamic analysis of current browser extensions finds no evidence of the attacks in the wild. We conducted two user studies confirming that participants do not detect the attacks with current error messages, email notifications, and UX responses to the attacks. We provide an improved clone detection algorithm and recommendations for relying parties that detect or prevent some of the attacks.**

I. Introduction

Two-factor authentication (2FA) defends against account compromise due to stolen passwords and phishing attacks. The current state-of-the-art for 2FA on the Web is FIDO2, the FIDO (Fast Identity Online) Alliance, and the W3C's newest set of specifications supporting 2FA, multi-factor authentication (MFA), and passwordless authentication. FIDO2 provides a standard web services API (WebAuthn) on client machines to authenticate users using public-key cryptography. The API is available in all popular browsers and seeing adoption by major service providers, including Facebook, GitHub, and Gmail.

FIDO2 supports a variety of client-side authenticators, including hardware security keys (*HSK*s) and built-in platform authenticators such as biometrics. Although this paper refers to *HSK*s, the ideas apply to all FIDO2 authenticators.

The FIDO2 specification focuses on remote attackers, so the threat model assumes a trusted client (browser and browser extensions). However, malicious browser extensions and cross-site scripting (XSS) have a long-standing history of stealing user data, including passwords, financial information, and browsing history [2], [1], [3]. With the adoption of the FIDO2 protocol, it is crucial to recognize the potential risks posed by malicious extensions and XSS. Examples of password theft by malicious extensions such as "Web Security" and "Stylish" underscore the pressing need to study these attacks for new emerging protocols such as FIDO2 and develop countermeasures to mitigate the risks. As attackers constantly evolve their tactics, it is important to proactively research potential attack vectors to stay ahead of the curve and ensure the effectiveness of FIDO2's security measures. The threat posed by these extensions cannot be overlooked and warrants continued research, mitigation strategies, and vigilance against new and emerging threats.

Our research systematically analyzed local attacks against FIDO2 from malicious browser extensions and XSS. During our analysis, we also discovered a weakness in the clone detection algorithm that combats attackers that gain physical access to *HSK*s. As a result, we identified four fundamental flaws that attackers can exploit: (1) Lack of confidentiality/integrity of FIDO2 messages accessible to browser extensions, (2) Broken clone detection algorithm, (3) Potential for user misunderstanding from social engineering and notification/error messages, and (4) Cookie life cycle.

We describe seven attacks that exploit these flaws and implement the attacks to demonstrate their feasibility. Four of the attacks have not been described previously. Three of them have been described earlier as theoretical attacks, but we implement them and show they are feasible (see Table II).

We were surprised to discover two attacks (3 and 4) where passwordless authentication presents more risk than passwords alone in the presence of a compromised extension. After performing the attack, an attacker can log in to an account the victim never logs into from the vulnerable browser. These attacks are significant because users may log into only a low-value account from an untrusted computer, not anticipating that it puts their high-value accounts at risk if the computer is compromised. This risk did not exist in the world of passwords when a user had a different strong password on their high-value account. Surprisingly, passwordless authentication using public-key cryptography opened up a new risk.

We also determine that 47% of browser extensions on the Chrome Web Store have sufficient permissions to execute most of the attacks. Finally, we present an improved clone detection algorithm that an *HSK*'s firmware can implement and make recommendations for web servers to improve the security of

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.24327
www.ndss-symposium.org

FIDO2 implementations.

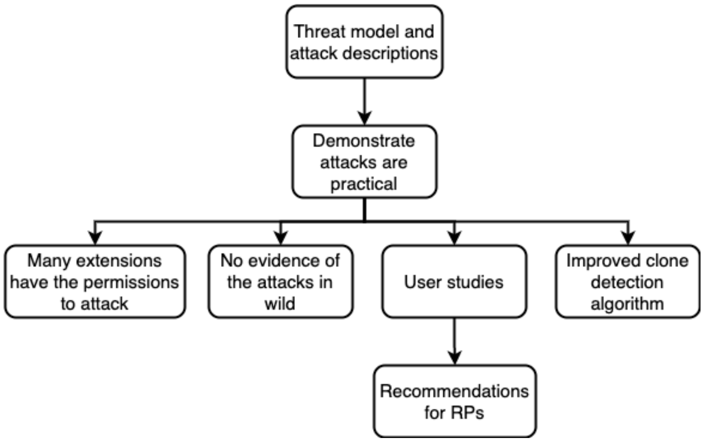

Threat model and attack descriptions
Demonstrate attacks are practical
Many extensions have the permissions to attack
No evidence of the attacks in wild
User studies
Improved clone detection algorithm
Recommendations for RPs

Fig. 1: Paper overview

An overview of the paper is depicted in Figure 1. The contributions of this paper are the following:

1) Systematization of attacks by two local adversaries on FIDO2 security: attacks from a malicious browser extension or XXS, and an attacker gaining physical access to an *HSK*. Our systematization reveals seven practical attacks or weaknesses.
2) Demonstrate the feasibility of the attacks:
   - We prototype a malicious browser extension for Chrome and Firefox, and demonstrate the attacks on ten popular web servers that use FIDO2.
   - We analyze current Google Chrome extensions to identify (1) how many have sufficient permissions to execute the attacks if they were compromised and (2) the scale of such attacks based on the number of users for the extensions. Our results show that 105,381 out of 211,026 extensions have sufficient permissions to compromise a WebAuthn client and execute the attacks. Furthermore, 404 of these extensions have more than one million users each.
   - We confirm that the current clone detection algorithm is vulnerable to a stealthy device cloning attack
   - We perform a static and dynamic analysis of current browser extensions and find no evidence of these attacks in the wild.
3) Through two user studies (n=80 and n=20), we confirm that current error messages, email notifications, or changes in the UX caused by these attacks were insufficient for participants in the user study to detect them.
4) Present an improved clone detection algorithm.
5) Provide recommendations for web servers and browsers that could help users detect some of the attacks.

II. FIDO2 Background

The FIDO2 protocol is a secure authentication method that utilizes public-key cryptography for user authentication to web services. In this protocol, users register their public key with the web service and prove ownership of the corresponding private key by signing challenges presented by the service.

The FIDO2 protocol involves three main entities:

The *Relying Party (RP)*: This entity represents the web application, such as "facebook.com," that supports FIDO2 authentication. The *RP* communicates with the authenticator through the WebAuthn client. It can choose to offer FIDO2 as a method for two-factor authentication (2FA) or passwordless authentication.

The *Authenticator*: This entity is a device that securely stores the user's private key. Users authorize an authenticator to generate a login credential to the *RP* by providing some form of input, such as a PIN or a button press. FIDO2 supports various types of authenticators, including built-in fingerprint readers and external (remote) authenticators like hardware security keys (*HSK*).

The *WebAuthn client*: Typically present in a web browser, the WebAuthn client acts as a mediator between the authenticator and the *RP*. It relays information and commands between the two entities. To prevent phishing attacks, the WebAuthn client reports the origin (URL) of the *RP* to the authenticator, ensuring the authentication process is bound to the correct website.

The FIDO2 protocol consists of two main components: the Web Authentication (WebAuthn) browser API and the client-to-authenticator protocol (CTAP). The WebAuthn API in the client provides an interface for the *RP* to interact with the authenticator, while the CTAP protocol enables secure communication between the WebAuthn client and external or roaming authenticators using Bluetooth, USB, or NFC.

A. Registration and authentication

The registration and authentication processes in FIDO2 involve the following steps:

Registration: Users initiate the registration process by clicking a "register/login" button. The *RP* sends a registration request to the WebAuthn client, including a challenge, user information, and *RP* information. The WebAuthn client forwards this information to the *HSK*, along with the *RP*'s origin and the request type. The *HSK* prompts the user for consent, generates a new asymmetric key pair, and sends the registration data (credential ID, public key, RP ID hash, counter, and attestation signature) as an attestation object to the WebAuthn client. The WebAuthn client then forwards this data to the *RP*, which verifies the signatures and critical parts of the response. Upon successful verification, the *HSK* is registered to the user's account.

Authentication: The authentication process is similar to registration, with a few differences. Authentication does not require user information, and instead of attestation, the *HSK* performs assertion by signing the response with the private key corresponding to the credential ID.

These mechanisms ensure secure and reliable authentication using the FIDO2 protocol, providing enhanced protection against various attacks and unauthorized access attempts.

B. Clone detection

By design, an *HSK* never releases the user's private key and other sensitive data. In theory, an attacker must steal an *HSK* to impersonate a user. NinjaLab recently demonstrated

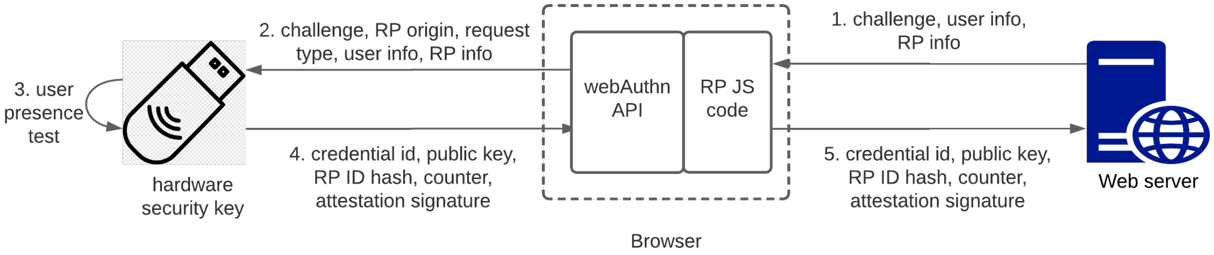


Fig. 2: FIDO2 Registration

a successful cloning of a Google Titan Security Key using a side-channel attack. The NinjaLab attack requires access to the device, 10 hours, $12,000 in equipment, and specialized expertise [15]. However, various HSK vendors have different firmware and hardware; attackers will find other ideas to clone HSKs faster at a cheaper cost. Roth et al. [23] demonstrate the cloning attack on Nordic nRF52832 by voltage glitching the nRF chip for firmware extraction with a low-cost setup (5€).

To detect cloning, the *HSK* and *RP* both maintain a counter. An *HSK* can have account-specific counters or a global counter. During registration, the *HSK* initializes the counter on the device and sends it to the *RP*, as shown in Figure 2. The *HSK* increments the counter and sends it to the *RP* each time it authenticates. The *RP* confirms the received counter is larger than the current counter and updates the counter. A cloning attack is detected if the *RP* ever receives a counter lower than the current counter.

If an attacker clones a device and impersonates the user, the *RP* increments the counter each time the attacker authenticates. When the victim eventually authenticates using the original *HSK* and counter, the *RP* detects the attack because the counter it receives is lower than its current counter and notifies the user about it.

C. Attestation

During registration, an optional attestation process allows an *RP* to verify the make and model of the *HSK*, allowing only specific devices (e.g., all employees must use a Google Titan Key) or blocking models with known security flaws.

Each *HSK* comes with a hard-coded private attestation key shared among a group of *HSK*s, such as $40K$ devices, to prevent user tracking. An *HSK* proves its make and model by signing part of the registration response with its attestation key. An *RP* needs access to the *HSK* public attestation keys to verify attestation signatures. The *RP* can access the keys on demand at the vendor or maintain a local copy.

III. Adversary Model and Attacks

This section provides an overview of the entities in FIDO2 and how they communicate. We then introduce our adversary model and describe seven attacks.

a) Entities and communication: In FIDO2, there are three main entities: (1) Relying Party (RP): a Web service, (2) WebAuthn Client: relays communication between an *RP* and an *HSK*, and (3) *HSK*: a user's hardware authenticator. To initiate the registration or authentication process with an *RP*, the user communicates with the webAuthn client. The webAuthn client communicates with the *RP* and the user's *HSK* to register or authenticate the user. The user may be asked to touch the *HSK* to confirm a request and prove that the user initiated the request. A more detailed explanation of the entities is provided in section II.

b) Adversary model: Our adversary model includes two independent adversaries:

- Adversary A1: a malicious/compromised browser extension or malicious web pages that leverage a vulnerable extension to compromise the FIDO2 webAuthn API. The adversary has access to plaintext FIDO2 communication and therefore can execute various attacks, such as MITM.
- Adversary A2: An adversary that gains temporary physical access to the victim's *HSK* and has cloned the device. The adversary cannot retrieve the private key or any other metadata from the *HSK*, they can only clone it.

c) Adversary goals: Adversary A1's goal is to impersonate a victim (Bob) who uses an *HSK* and gain unauthorized access to Bob's account. Ultimately, A1 wants to impersonate Bob from *A1's device* over an *extended period* without detection after executing a short-duration one-time attack from Bob's device or another device that Bob uses just once. Reliance on a single malicious code execution from Bob's device removes A1's continued dependence on an extension's malicious code to access Bob's accounts, decreasing the likelihood of detection.

A1 cannot achieve its goal by registering OAuth tokens, stealing session cookies, or monitoring all user communication. Many websites do not use OAuth access tokens (e.g., banking), and the tokens last only for several hours to a couple of weeks. Furthermore, cookie-based login sessions expire as soon as a user logs out. Therefore, if the attacker wants long-term access, they have to steal cookies frequently, which is infeasible if the victim only logs in only once from a vulnerable browser.

Adversary A2's goal is to bypass the FIDO2 clone detection algorithm and gain unauthorized access to the victim's account using the cloned device *without being detected*.

TABLE I: Relying Party Analysis

	Relying Party	Attestation	Authentication before adding additional *HSK*	Notification after adding an *HSK*	Least secure signature algorithm	Clone detection error
1	Facebook	○	password	○	ES256	(Page refresh)
2	GitHub	○	○	email	RS256	"Security key authentication failed"
3	Boxcryptor	required	password	○	ES256	N/A ‡
4	Dropbox	required	password	email	RS256	N/A ‡
5	Twitter	○	N/A †	○	RS256	(technical problem during testing)
6	Cloudflare	○	password	○	RS256	"Invalid security key used. Please use a security key registered to this account."
7	Basecamp	○	○	email	RS256	"We couldn't verify this security key. Make sure you have registered it."
8	Login.gov	○	○	○	RS256	(cloning not detected)
9	Shopify	○	password	email	RS256	"Couldn't connect to your security key. Try again."
10	1Password	○	○	○	ES256	"Unable to verify your security key."

○ none † Twitter supports only one *HSK* on an account ‡ Unable to test clone detection due to required attestation
$ES256 = ECDSA$ w/ SHA-256 $RS256(-256) = RSASSA$-$PKCS1$-$v1\_5$ using SHA-256

A. Attacks

This section describes the attacks Adversary A1 can execute on the webAuthn client API and Adversary A2 can execute with a cloned *HSK*. We take a holistic approach to explore a range of attacks in detail based on three types of flaws: protocol errors, HCI (Human-computer interaction) challenges that impact user understanding, and implementation weaknesses. Table II shows the flaws, the type of each flaw, and attacks that exploit each flaw. We are the first to demonstrate the practical feasibility of these attacks.

1) Mis-binding attack during registration – Attack 1: During registration of the victim's *HSK*, Adversary A1 replaces the victim's public key with the attacker's public key in the *HSK* response. It also replaces the digital signatures with signatures generated using the attacker's private key. This attack causes the *RP* to register the attacker's *HSK* instead of the victim's *HSK*. This attack was first identified by Hu et al., when investigating the security of the Universal Authentication Framework (UAF), a precursor to FIDO2[12].

An attacker can register either a software-based *HSK* or a hardware-based *HSK*. A software-based *HSK* makes it easier for the attacker to automate the MITM attack but is not an option if the *RP* requires *attestation*. From our analysis, popular RPs like Facebook and GitHub do not require *attestation* (see Table I). Attestation forces the attacker to use legitimate hardware *HSK*, which increases the attacker's effort because the attacker must forward the requests and responses to a remote machine where they can connect the hardware *HSK*. The attacker can use an *HSK* with the same make and model as the user to be more stealthy. The attacker can automate the key tap on their *HSK* by building additional hardware to perform the key tap without the user being present.[1] Other Robo projects, such as MattRobot,[2] aim to simulate touches by employing Robo fingers capable of handling various types of touch interactions.

Once the attacker has registered their *HSK*, they can mark the *Remember Me* option that causes the generation of a cookie that allows the victim to continue to log in while remaining oblivious that the attacker's *HSK* was registered. If the victim logs in through a different browser or device, they will receive an error indicating that their *HSK* was never registered. We answer the effectiveness of two email notifications in RQ1 in Section V.

TABLE II: FIDO2 flaws exploitable by local attacks

Flaws	**Type**	**Attacks**
Lack of confidentiality/integrity of FIDO2 messages accessible to browser extensions	Protocol	1[12], 2, 5
Broken clone detection algorithm	Protocol	7
User misunderstanding from social engineering and notification/error messages	HCI	1, 2, 3, 4 [10]‡, 7
Cookie life cycle	Implementation	6[20]

‡ = for FIDO UAF

2) Double-binding attack during registration & authenticated session – Attack 2:

a) During registration: During registration of the victim's *HSK*, Adversary A1 registers their malicious authenticator first to the victim's account and sends a second registration request in the background to register the victim's *HSK* to the same account. The victim and the attacker can respond to their respective registrations if the *RP* requires a touch for user presence on their *HSK*. A1 directs the attacker's user presence request to the attacker. Like the mis-binding attack, the attacker can automate the test for user presence. Unlike the mis-binding attack, the victim can log in using their *HSK* without even being aware that a second *HSK* is also bound to the account.

b) During authenticated session: Adversary A1 registers a malicious *HSK* to the victim's account during a logged-in session by sending a registration request and the response from their *HSK* in the background without any victim's involvement. This attack can be executed anytime during a session, providing ample opportunity to initiate an attack instead of being limited to only during registration. Adding a malicious *HSK* allows an attacker to access the account stealthily indefinitely unless the victim manually detects it through the registered *HSK* page.

A strong defense against this attack is to require authentication using an already-registered *HSK* before adding a new *HSK*. Requiring the previous 2FA prevents software attacks that register their malicious *HSK* in the background without user knowledge. Of the ten web services we analyzed, as shown in Table I, four do not require any authentication while adding a new *HSK* during a logged-in session. Five services require

[1] https://bert.org/2020/10/01/pressing-yubikeys/.

[2] https://www.mattrobot.ai

only passwords to add a new *HSK* even if there is already a registered *HSK*, which allows software-based attacks to add a malicious *HSK*. Twitter allows only one registered *HSK*, which prevents a double-binding attack. However, limiting registration to one *HSK* is not recommended as it prevents a user from registering an *HSK* as a secure form of backup.

Use of notifications to detect binding attacks: A victim may detect one of the binding attacks (Attacks 1, 2a, and 2b described earlier) in two ways unless A1 also compromises the notification channel.

- Some *RP*s send users a registration notification through a different channel, such as email. If users monitor these notifications, they can detect the attack when they see two notifications for Attack 2a or an unexpected notification for Attack 2b.
- An *RP* can display a list of registered *HSK*s on the account, including the make and model. A user can verify the list regularly to detect a malicious *HSK*. This passive detection depends on user awareness and effort.

We analyzed the notification process of ten popular web services that use FIDO2's WebAuthn for 2FA (see Table I). Six services do not send any registration notifications. Four services send an email notification. However, they all send an identical email when adding a new *HSK*, as shown in Figure 3). For Attack 2a, receipt of two duplicate emails that lack any information about the identity and number of *HSK* devices may not be sufficient to raise suspicion. We answer the effectiveness of two email notifications in RQ2a in Section V.

3) Synchronized login – Attack 3: As described in Figure 4, while a victim logs into a website with a registered *HSK*, Attacker A1 generates a login to another victim account that expects an *HSK* to authenticate. The second login is invisible to the user except that A1 coerces the user to perform the user presence test, believing it to be for the first site.

To accomplish this, A1 adds an invisible iframe to the `example.com` page that loads `facebook.com`, resulting in an authentication request to the *HSK* sent from the iframe. By default, the browser's webAuthn client does not allow cross-origin iframes to send an authentication request to an *HSK*. To allow cross-origin iframe authentication, A1 also adds (1) an allow attribute to the iframe (i.e., *allow="publickey-credentials-get *"*) and (2) a header in the response from facebook.com that loads the iframe (i.e., *Permissions-Policy: publickey-credentials-get=**).

There are two ways to handle the test for user presence. First, suppose `facebook.com` has a registered *HSK* but the user marked *Remember this device* during the login process. *Remember this device* uses a cookie for login once the user completes their first login from the device using their *HSK*. The user could misinterpret the prompt for using the *HSK* to login to `example.com` in the background as a request to re-login to `facebook.com` using the *HSK* and complete the user presence test. Second, even if the user authenticates to `facebook.com` using their *HSK* and the login to `example.com` also requires authenticating with their *HSK*, A1 can cause two prompts to tap the *HSK* for the user presence test and the user may comply believing the first tap failed to be recognized. Currently, browsers show the domain of the website that a user is authenticating to, but users do not usually verify it. Furthermore, tapping an *HSK* twice is common due to improperly touching it the first time. We explore this user behavior in RQ4 in Section V.

This attack is more insidious than some other attacks because the attacker succeeds against the victim using the victim's *HSK* without registering the attacker's *HSK*, reducing the amount of forensic evidence for detecting the attack.

This attack assumes that A1 has compromised the user's password to `facebook.com`. If passwordless authentication is enabled on the target site, then A1 does not need a stolen password. In this case, passwordless authentication increases the risk of this attack on this account since A1 does not need to first compromise the password. A1 can compromise an account the user has not accessed while A1 was present.

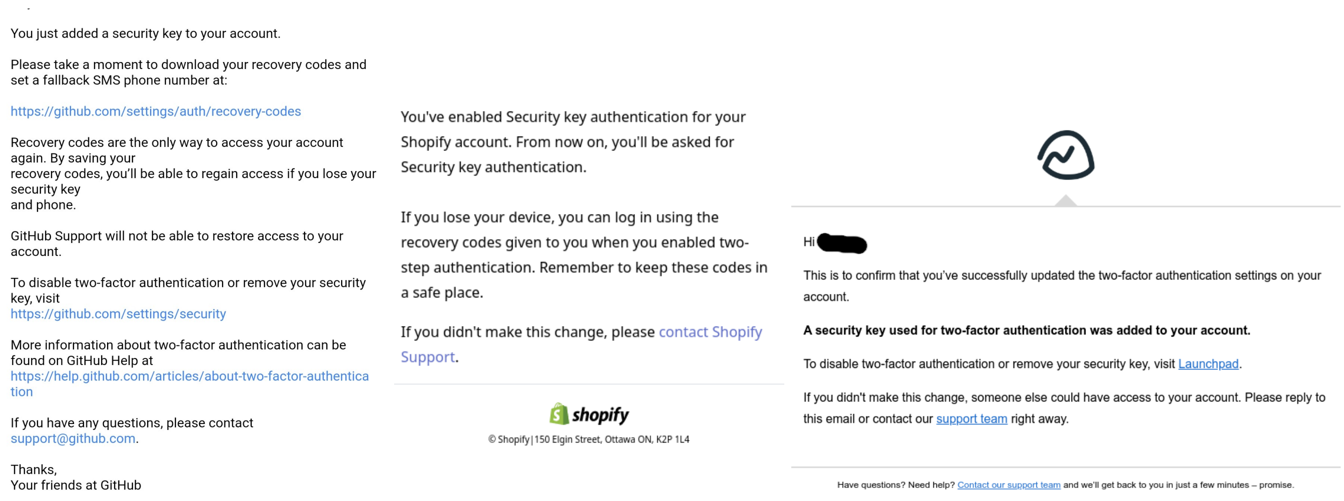
4) MITM – Attack 4: When a victim visits a website, such as `facebook.com`, an attacker can manipulate the authentication process to execute a MITM. If the victim is already logged in, the attacker can forcibly log them out by removing the session details from the network request during communication with the website.

To carry out the attack, the attacker generates a session from their own device and initiates an authentication request to the victim's `facebook.com` account. The attacker intercepts the authentication request received by the victim and replaces it with the request they received on their device, aiming to obtain the victim's signature using their hardware security key (*HSK*). The attacker then captures the response, transfers it to their device, and forwards it to `facebook.com`, enabling their device to create a login session on the victim's account. In case the user already had a session with `facebook.com`, the attacker adds the previous victim's session details to `facebook.com` to allow the victim to log in using their previous session.

In scenarios where the victim was not initially logged in to `facebook.com`, the attacker can either temporarily share their own session with the victim to avoid arousing suspicion or execute a variant of Attack 3. In this variant, the user is prompted to authenticate to `facebook.com` by tapping the hardware security key twice. One session is created for the victim, while the other is established for the attacker.

5) Signature algorithm downgrade – Attack 5: A signature algorithm downgrade attack occurs during registration when adversary A1 modifies the list of signature algorithms in the registration request sent from the *RP* to the WebAuthn client. In the registration request, the *RP* sends a prioritized list of signature algorithms that it supports. The attacker leaves only the least secure algorithm and removes all others.

All ten web services in our analysis use secure signing algorithms that are not straightforward to crack. However, this attack could be used with other web services in the wild that support at least one insecure signing algorithm. Also from the ten RPs we analyzed, as shown in Table I, seven support the inadvisable signing algorithm $RSASSA$-$PKCS1$-$v1\_5$ using SHA-256. An attacker can force the *HSK* to use an inadvisable algorithm and possibly exploit it. The $RSASSA$-$PKCS1$-$v1\_5$ using SHA-256 has been exploited using Bleichenbacher's attack, allowing one to perform arbitrary



You just added a security key to your account.

Please take a moment to download your recovery codes and set a fallback SMS phone number at:

https://github.com/settings/auth/recovery-codes

Recovery codes are the only way to access your account again. By saving your recovery codes, you'll be able to regain access if you lose your security key and phone.

GitHub Support will not be able to restore access to your account.

To disable two-factor authentication or remove your security key, visit https://github.com/settings/security

More information about two-factor authentication can be found on GitHub Help at https://help.github.com/articles/about-two-factor-authentication

If you have any questions, please contact support@github.com.

Thanks,
Your friends at GitHub

(a) GitHub

You've enabled Security key authentication for your Shopify account. From now on, you'll be asked for Security key authentication.

If you lose your device, you can log in using the recovery codes given to you when you enabled two-step authentication. Remember to keep these codes in a safe place.

If you didn't make this change, please contact Shopify Support.

shopify

© Shopify | 150 Elgin Street, Ottawa ON, K2P 1L4

(b) Shopify

Hi [redacted]

This is to confirm that you've successfully updated the two-factor authentication settings on your account.

A security key used for two-factor authentication was added to your account.

To disable two-factor authentication or remove your security key, visit Launchpad.

If you didn't make this change, someone else could have access to your account. Please reply to this email or contact our support team right away.

Have questions? Need help? Contact our support team and we'll get back to you in just a few minutes – promise.

(c) Basecamp

Fig. 3: Examples of email notifications when adding a new *HSK* to GitHub, Shopify, Basecamp

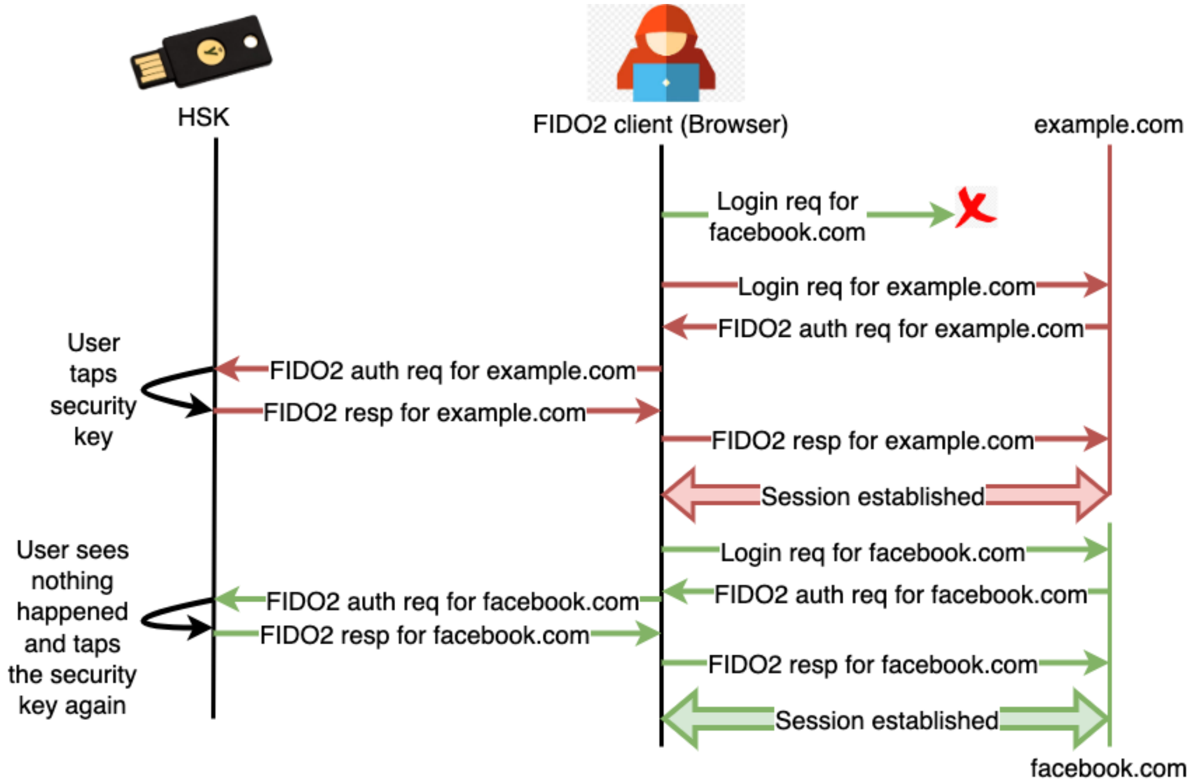

HSK
FIDO2 client (Browser)
example.com
Login req for facebook.com
Login req for example.com
FIDO2 auth req for example.com
User taps security key
FIDO2 auth req for example.com
FIDO2 resp for example.com
FIDO2 resp for example.com
Session established
Login req for facebook.com
User sees nothing happened and taps the security key again
FIDO2 auth req for facebook.com
FIDO2 auth req for facebook.com
FIDO2 resp for facebook.com
FIDO2 resp for facebook.com
Session established
facebook.com

Fig. 4: Synchronized login (Attack 3)

RSA private key operations. Given access to an oracle, and insecure exponents, the attacker can sign arbitrary messages with the *HSK* private key. Table I lists the minimum supported signing algorithms by individual RPs from our analysis.

The signature downgrade attack is more of a theoretical threat today. No RPs that we analyzed support weak signing algorithms. It is important to understand the threat exists in FIDO2 because experience shows that downgrade attacks are exploited after many years when algorithms become outdated.

6) Cookie Lifecycle – Attack 6: Although 2FA strengthens security, it decreases usability when users must always authenticate using an *HSK*. In response, some *RP*s use cookies to avoid 2FA for future logins on a device if users select "Remember this device."

For example, Facebook issues a ‘$datr$‘ cookie to denote the user has previously authenticated using an *HSK* on their device.

Future logins skip requiring the *HSK* until the cookie expires. If an attacker steals this cookie and adds it to the browser on their device, the attacker can authenticate as the user with only a stolen password and altogether bypass 2FA.

The "cookies" permission allows a Chrome extension to use the cookies API. With API access, A1 can steal long-term session cookies or cookies that help them bypass 2FA on any device for a domain. We analyzed 246,345 Chrome extensions and discovered 31,326 Chrome extensions with "cookies" permission. 18,024 of those extensions also have "://*" permission, allowing these extensions to steal the cookies of every *RP* a user accesses. 102 of these extensions have more than a million users each. Figure 6 contains the full distribution of users for these extensions.

To test the feasibility of a cookie-stealing attack, we logged into Facebook from a Chrome browser and enabled 2FA using an *HSK*. We added the current browser to the "Remember this device" list. Then, we logged out of Facebook and copied the '$datr$' cookie to a Chrome browser on another device. We confirmed that the cookie permitted us to successfully authenticate to Facebook from the second device with just a password and bypass 2FA.

Cookie-stealing attacks are well-known. From our experiment, we learned that Facebook sets cookie expiration at approximately two years. Although this is motivated by usability, attackers can exploit this to grant them long-term access to 2FA-enabled accounts without stealing the hardware authenticator.

7) Bypassing clone detection algorithm – Attack 7:

a) Lack of informative error message: Cloning an *HSK* allows an attacker to authenticate to the victim's account. However, the FIDO2 has a built-in clone detection mechanism (explained in Section II-B). We conducted experiments to trigger the clone detection error messages on ten RPs by simulating a cloned *HSK* using our malicious browser extension. First, we registered our virtual *HSK* on the target website and successfully authenticated. Second, we reduced the counter on the virtual *HSK* to simulate a cloned *HSK* and reattempted authentication on the target website to trigger an error message. Some websites did not report any errors, while most displayed generic error messages unrelated to device cloning (refer to Table I).

These generic messages may lead users to reattempt the login. Multiple reattempts could eventually lead to a successful login, depending on the number of times the attacker used the cloned device. For example, if the *HSK* and *RP* have a counter value of x, the cloned *HSK* would initially have the same value. When the attacker logs in, the counter's value updates to $x + 1$ at the *RP* and the cloned *HSK*. If users try to log in to their account, they may receive an error message, but the counter on the user's *HSK* will advance to $x + 1$. If the user reattempts the login, they will be granted access to their account and may not understand the failed attempt is due to a cloned *HSK*. We answer the effectiveness of two email notifications in RQ3 in Section V.

Figure 5 shows screenshots of error messages from *RP*s when the counter value submitted by the *HSK* is lower than the value stored at the *RP*. Even though this indicates a possible cloning attack, none of the error messages state that the user's account may be under attack. Several messages even encourage the user to switch to a less secure form of authentication.

b) Stealthy device cloning attack: FIDO2 supports a counter to detect device cloning, as explained in Section II. Assume A2 has cloned a victim's device to gain unauthorized, undetected, short-term access to their account. The following paragraph describes a stealthy attack by A2 that avoids detection by the clone detection algorithm in FIDO2.

Assume the user's *HSK* and the *RP* have an account-specific counter value of x. When A2 gains physical access to the user's *HSK*, they first clone the *HSK* and then increment the counter value on the user's *HSK* (by y) by sending it y dummy authentication requests for the RP they want to compromise. The dummy requests can be created using libraries such as Yubico's libfido2. The user's *HSK* counter is now $x+y$, the cloned *HSK* counter is x, and the *RP*'s counter is x. The attacker can now log in to the user's account up to y times before the user logs in without detection. When the user logs in, they won't trigger an error if $x + y > x + (numberOfTimesAttackerLoggedin)$, but the login will force an update to the RP counter to $x + y$. The attacker will not log in again without detection, but the attack provides a window of opportunity that the attacker can exploit. An attacker can perform a variation of this attack if the *HSK* maintains a global counter.

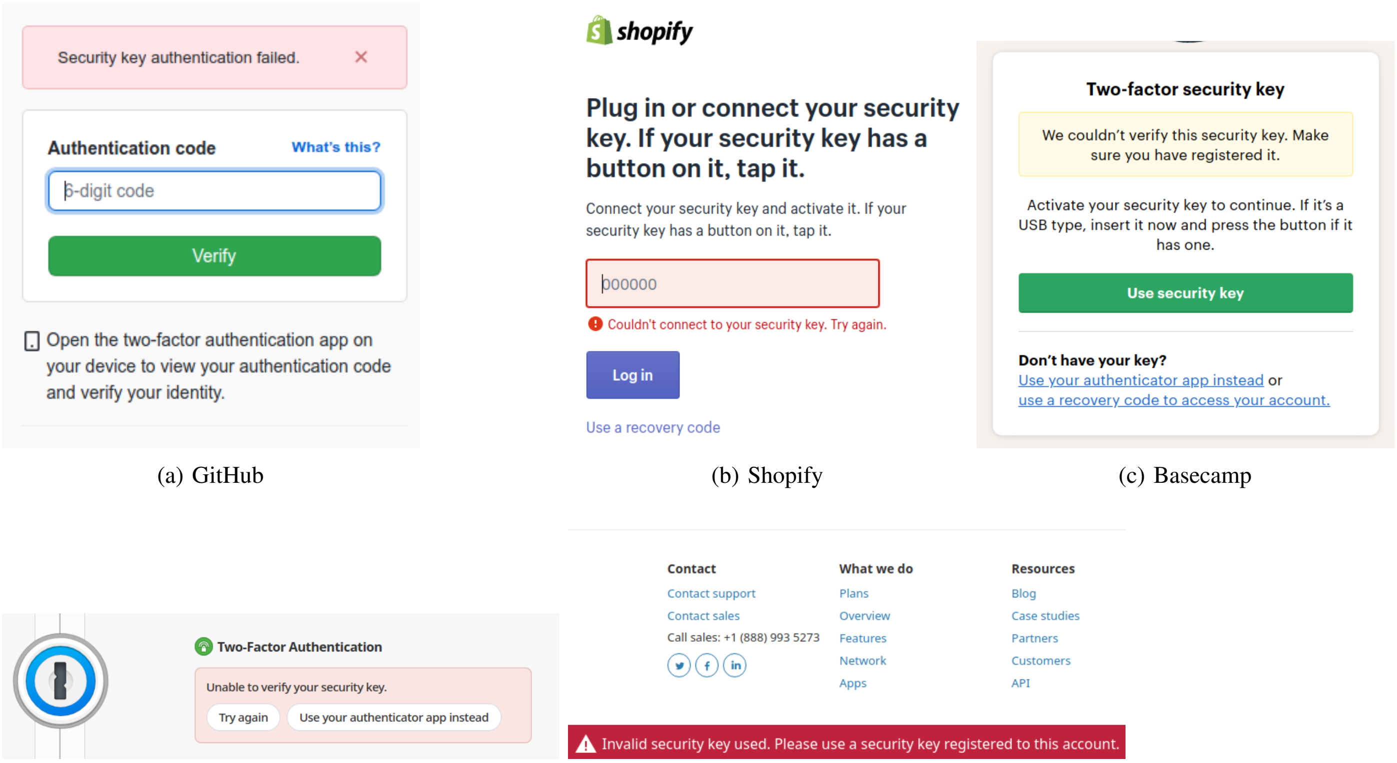
IV. Attack feasibility

To demonstrate the feasibility of the attacks, we built a prototype of a malicious Chrome extension that compromises a webAuthn client and executes the seven attacks. In our Chrome extension, content scripts allowed us to obtain details and make changes to the webpages a browser visits. We replace Chrome's web API function *navigator.credentials.create* with our custom handler on every webpage. Our custom handler modifies/replaces the original FIDO2 request or response with a malicious one.

Similar to Kaprevelos et at. [14], we analyzed the permissions requested for 246,345 Chrome extensions, extracted by CRXcavator [9] from the Chrome webstore on Jan 21st, 2021. We found that 115,881 Chrome extensions use *activeTab/tabs* permission, which is sufficient to execute a MITM attack by overriding webAuthn client APIs. There are 404 extensions with more than one million users each that can execute these attacks. A malicious actor only has to compromise one of these extensions to be in a position to launch an attack on over one million users. Figure 6 shows the distribution of users among these extensions.

Prior research demonstrates the feasibility of Attacker A2 obtaining a clone of an *HSK* once they have physical access. For example, Roche et al. [22] describe how to clone a Google Titan Security Key. It requires about 10 hours to complete the cloning process. Assuming that cloning has taken place successfully, we describe how an attacker can avoid the clone detection algorithm described in the FIDO2 specification.

The remainder of this section provides detailed descriptions of attacks that Adversary A1 can execute on the webAuthn client API and Adversary A2 can execute with a cloned *HSK*. Others have explored a few attacks against FIDO2 (see



Security key authentication failed. ×

Authentication code

What's this?

6-digit code

Verify

Open the two-factor authentication app on your device to view your authentication code and verify your identity.

(a) GitHub

shopify

Plug in or connect your security key. If your security key has a button on it, tap it.

Connect your security key and activate it. If your security key has a button on it, tap it.

000000

Couldn't connect to your security key. Try again.

Log in

Use a recovery code

(b) Shopify

Two-factor security key

We couldn't verify this security key. Make sure you have registered it.

Activate your security key to continue. If it's a USB type, insert it now and press the button if it has one.

Use security key

Don't have your key?

Use your authenticator app instead or use a recovery code to access your account.

(c) Basecamp

Two-Factor Authentication

Unable to verify your security key.

Try again

Use your authenticator app instead

(d) 1Password

Contact

Contact support

Contact sales

Call sales: +1 (888) 993 5273

What we do

Plans

Overview

Features

Network

Apps

Resources

Blog

Case studies

Partners

Customers

API

Invalid security key used. Please use a security key registered to this account.

(e) Cloudflare

Fig. 5: Examples of clone detection error messages

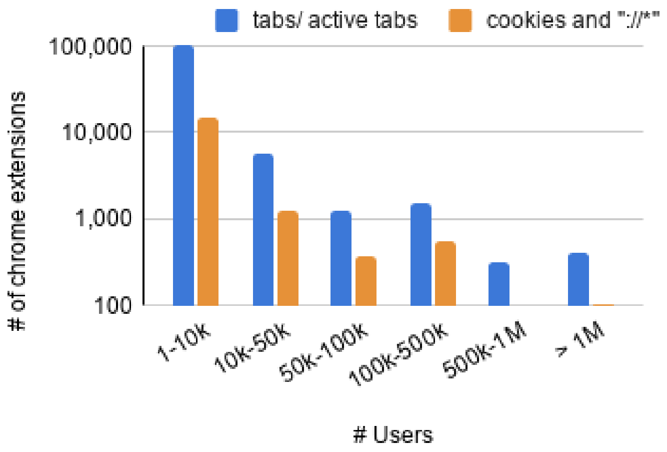


Fig. 6: User distribution for Chrome extensions with permissions that allow a MITM attack against FIDO2 *HSK*s

Section VIII). We take a holistic approach to explore a range of attacks in detail. We are the first to demonstrate the feasibility of these attacks.

a) Static and dynamic analysis of real Chrome extensions: We tried to find real-world attacks by analyzing extensions. We did not detect any attack happening in the wild as of now.

We analyzed extensions from the Chrome store to determine if any extensions in the wild were executing attacks on FIDO2. Specifically, we downloaded the source code of 152,526 Chrome extensions from the Chrome store over 20 days during September of 2022 to minimize additional load on the Google server. Our testing pipeline consisted of two stages: a static and dynamic analysis.

In the static analysis phase, we filtered extensions by two criteria: whether an extension had the 'modify all data' permission or if the phrases `navigator.get` or `navigator.create` were present in any of the source files. If either criterion was met, the extension was flagged and placed in a folder to be run through dynamic analysis.

In the dynamic analysis phase, we installed each flagged extension on a browser in a virtual machine in sequence. The virtual machine had a virtual FIDO2 authenticator that automatically approves registration and authentication requests. We ran an authentication server on the client outside the virtual machine. We performed complete FIDO2 registration and authentication flow between the webAuthn server and the virtual FIDO2 through the Chrome browser, where the browser extension under analysis was installed. For each request, we verified the Hash(ClientData) the server sent and the virtual authenticator received for registration and authentication. In response, we verified the public key and the signed object sent by the virtual key with what the server received.

We ran registration and authentication flows for each extension on the top 100 Alexa domains to account for extensions that might trigger malicious code based on URLs. We added a mapping of these URLs to our local server's IP address in the host file so that the browser connects to our local server for all the URLs. We tested our setup on our proof-of-concept malicious extensions and were able to detect all the attacks.

Our static analysis identified 155 extensions that matched our criteria. However, none of the extensions in the dynamic analysis were detected as performing attacks on FIDO2. The extensions flagged by static analysis, particularly those using the `credentials.create` and `credentials.get` APIs, fall into two categories: password managers providing FIDO2 support and poorly optimized extensions. The first category of false positives pertains to valid use cases such as password managers trying to implement 2FA for user's account, which were flagged for using the `credentials.create` and `credentials.get` APIs. The second category of false positives pertains to poorly optimized extensions. These extensions are created using webpack, which packs multiple dependencies into a single file. Sometimes parts of these dependencies are unused, and it seems they are included for an API call that never happens. One such dependency is 'simplewebauthn' which is depended on by other libraries but only makes calls to the `credentials.create` and `credentials.get` APIs not usable by extensions.

b) Applicability of the attacks across various browsers: The *tabs/activeTabs* permissions operate uniformly across browsers, enabling all our attacks except for synchronized login (Attack 3). This specific attack requires the utilization of the "*publickey-credentials-get*" API, currently supported by Chrome, Edge, and Samsung Internet. It is in an experimental phase in certain browsers, such as Firefox [17]. We successfully tested our attacks on Firefox except for Attack 3.

c) Cross-site scripting (XSS): : All the attacks we demonstrated using the browser extension can also be exploited through XSS, except for synchronized login (Attack 3). This attack necessitates adding a header to the incoming response from the target *RP* that the attacker aims to authenticate to. If an XSS-based attacker can employ other techniques to insert headers in the response received from the target website, they can successfully execute this attack.

V. USER STUDY

Our user studies aim to assess whether the existing error messages, email notifications, and UX behavior enable users to detect various attacks. The results will show the effectiveness of the attacks under the current system design.

We excluded the signing algorithm downgrade, cookie-stealing, and MITM attacks from our study because they do not cause any changes to the UX. One variant of the MITM attack modifies the UX by presenting two pop-ups with the same domain name. This alteration in UX resembles a synchronization attack and possesses a higher degree of stealthiness. We can establish an upper bound for detecting the MITM attack variant by measuring users' perception of UX changes during synchronization attacks. We designed our studies to address the following four research questions, corresponding to one of the four primary attack types.

RQ1 Misbinding– How do users interpret the error and their inability to log in after a misbinding attack?

RQ2 Double-binding–
- (a) How do users interpret when they receive two registration emails after a *HSK* registration?
- (b) How do users interpret the addition of a rogue *HSK* they encounter on the settings page?

RQ3 Clone detection– How do users interpret clone detection error messages they encounter during the login process?

RQ4 Synchronized login–
- (a) How do users interpret pressing the HSK button twice before logging in?
- (b) Do they detect the attack by observing the browser's popup displaying the website name?

We conducted two user studies to evaluate the detectability of attacks given the current error messages and user experience (UX). Based on our experience, we hypothesized users would likely not detect these attacks in practice. Therefore, we decided to prime the participants during the study by asking them to watch for potential attacks that would take place for some of them. The results represent the best-case scenario for detecting the attack due to the priming.

The objective of the first study, an online survey, was to assess users' comprehension of error messages, notifications, and unusual UX flow. For this investigation, we selected RQ2a, RQ3, and RQ4a, which were amenable to measurement through an online survey and necessitated minimal contextual information. RQ2a and RQ4a were also measured in the subsequent in-lab study to ascertain their validity.

The second user study was an in-lab investigation that addressed all the research questions (RQs) except RQ2a. We developed a malicious browser extension that executed the attacks to provide participants with the necessary contextual information. We then had participants register an *HSK* and log in to a test account. At the same time, the extension performed the attacks, allowing us to capture participants' perceptions of the events that transpired during an attack.

A. Survey study

We conducted an IRB-approved survey exploring participants' understanding and actions when encountering attacks 2a, 3, and 7a. We asked participants open-ended questions about what they understand and their next step in the following scenarios: (1) [RQ2a] Attack 2a- receive two *HSK* registration emails due to double authenticator registration, (2) [RQ3] Attack 7a- encounter clone detection error message, and (3) [RQ4a] Attack 3- requires two taps to complete authentication.

a) Demographics: We recruited n=80 participants from Prolific. We mentioned in our study page *"Please only take part in this survey if you have used a security key or hardware token such as Yubikey for authentication."* However, according to Prolific policies, we cannot screen for surveys using the description, and therefore we could not reject any participants. Out of 80 participants, 32 use an *HSK* for their accounts, 45 do not use an *HSK* for their accounts, and 3 were unsure. Table III presents the survey demographics. The median time to complete the survey by participants was 6 mins, and we paid them each $1.20, *i.e.* $12/hr.

Metric	Percent	Metric	Percent
Gender		**Ethnicity**	
Male	50	White	80
Female	50	Asian	10
		Black	6.2
Age		Mixed	3.8
18-29 years	38.8		
30-39 years	32.5	**Employment Status**	
40-49 years	10		
50-59 years	12.5	Full-time	56.3
60+ years	6.2	Part-time	20
Student status		Unemployed	12.5
Student	31.3	Unpaid work	5
Non Student	66.3		

TABLE III: Survey participant demographics. Percentages may not add to 100% because we do not include "Other" or "Prefer not to answer" percentages for brevity.

1) Methodology: We designed three questions corresponding to attacks 2a, 3, and 7a.

a) [RQ2a] Attack 2a: To measure users' reaction to Attack 2a, we described a scenario where they recently registered their *HSK* with GitHub. To simulate the attack, we provided them with credentials for a test Gmail account containing two *HSK* registration emails and other random emails such as GitHub account creation and 2FA enrollment. Figure 3a shows the email content. We tasked each participant with logging into the test Gmail account and identifying whether their GitHub account had any malicious activity. Then, we primed participants by telling them that half of the participants' GitHub accounts had some malicious activity. We introduced this priming to determine if even cautious users will ignore two consecutive *HSK* registration email notifications arriving within seconds of each other.

b) [RQ4a] Attack 3: To explore users' behavior during Attack 3, we described a scenario where they had used an *HSK* with their work account for a long time. We told them that one day while trying to log in, they tapped their *HSK* to satisfy the user presence test. It doesn't work the first time but works after tapping it the second time. We then asked them why this might occur and how they would respond.

c) [RQ3] Attack 7a: To explore user behavior during Attack 7a, we described a scenario where they have been using an *HSK* with their work account for a long time. One day, an error occurs while logging in. We showed participants GitHub's clone detection error message as shown in Figure 5a and asked what they thought was the reason for the error message and what would be their next step.

2) Results:

a) Attack 2a: None of the participants identified malicious behavior with their GitHub account, even with the priming. Five participants noticed the two *HSK* registration emails but did not consider it malicious activity. One of the reasons most participants did not notice two emails is that Gmail, by default, hides the content of the second email if the content is the same as the first email. Furthermore, even if somebody notices two emails, two identical emails within seconds can be considered an intentional notification mechanism or a configuration error. However, it is hard to imagine it as an attack.

> P11: *"It doesn't seem like there was any suspicious activity. There were only 3 emails and 2 of them talked about 2-step authentication processes."*

b) Attack 7a: We asked participants what they thought caused the error. No participant considered it an attack. Participants attributed the error to incorrect key connection, wrong hardware token, USB reading issues, dust on hardware or USB slot, corrupted security key or PIN, device synchronization problems, incorrect username or password, prompt interaction delay, security token expiration, computer not updated, additional layer of authentication, and server errors. Based on these reasons users said they would perform the following steps: unplugging and reconnecting the key, refreshing and retrying with the correct hardware token, using alternative 2FA methods, cleaning the USB port, resetting the security key PIN, re-entering username and password, restarting the computer, seeking help on Google for username/password issues, and contacting GitHub support for prompt interaction delays.

> P23: *"Could be a number of things. Could be the two-factor key got zapped or erased somehow. The USB port is not working. The authenticator key is dirty, and the contacts need to be cleaned with isopropyl. Something is corrupt in the computer. Needs rebooted."*

> P65: *"Sometimes websites have temporary bugs that are beyond the user's control."*

c) Attack 3: No participant considered that it was malicious behavior. Instead, participants mentioned that the *HSK* tap is unreliable and may not work the first time. Therefore, they would tap it again. If that did not work, they would follow the same steps mentioned for the clone detection error messages.

B. In-lab study

We conducted a second IRB-approved in-person study to determine whether participants would detect these attacks when provided with the context received during an actual attack.

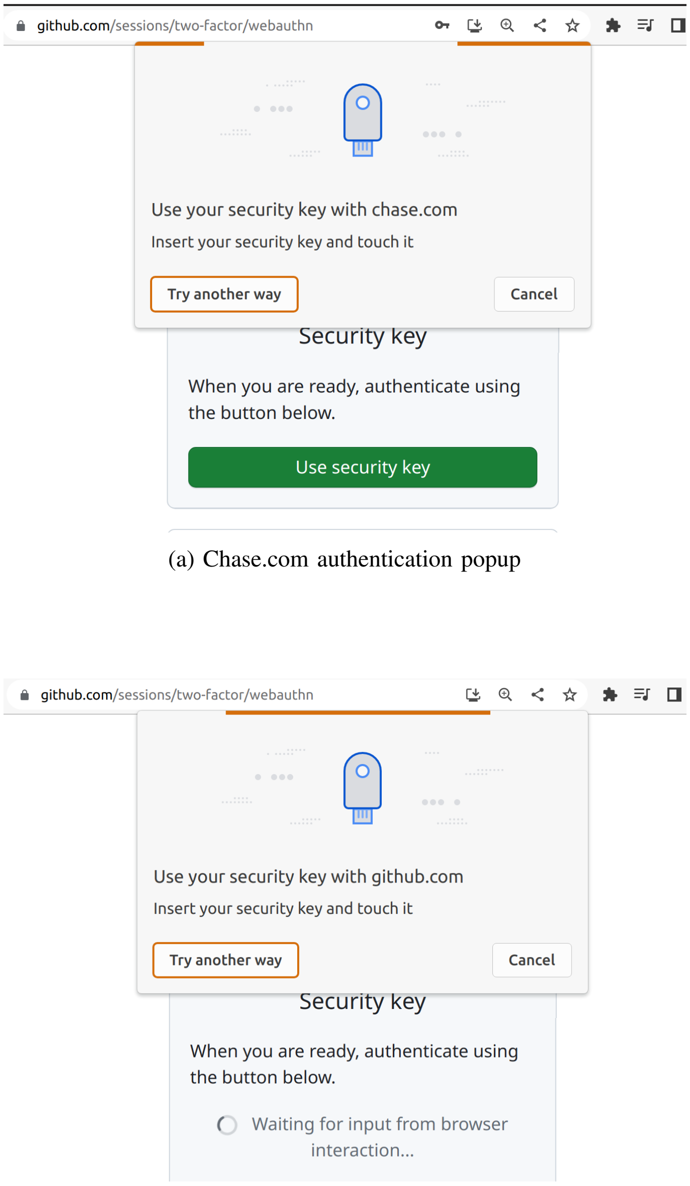
1) Methodology: To address the four research questions, we implemented the corresponding attacks in a browser extension. Participants were assigned the following four tasks, each corresponding to one of the research questions:

Task 1: Participants were presented with a scenario directing them to register an *HSK* for their work `github.com` account and verify it by logging in. Our attack implementation ensured that the user experience remained unchanged during registration. Participants received an error message during login.

Task 2: Participants were presented with a scenario directing them to set up an *HSK* with their `github.com` account. During the *HSK* registration process, we also registered a fake *HSK* with the nickname *admin*. Participants were asked to navigate to the settings page to verify if the *HSK* was registered correctly and suggest improvements for its security.

Task 3: Participants were presented with a scenario directing them to log in to their work `github.com` account, which they had been using for years. Upon the first login attempt, they encountered a clone detection error message. However, a subsequent login attempt succeeded due to incrementing the authentication counter after the failed attempt.

Task 4: Participants were presented with a scenario directing them to log in to their work `github.com` account, which they had been using for years. During login, our extension sent an authentication request on behalf of `chase.com` and another authentication request for `github.com`, as shown in Figure 7. Users were required to tap the *HSK* button twice, and before each popup, they observed the browser's popup displaying the domain they were authenticating to.


github.com/sessions/two-factor/webauthn

Use your security key with chase.com

Insert your security key and touch it

Security key

When you are ready, authenticate using the button below.

(a) Chase.com authentication popup

github.com/sessions/two-factor/webauthn

Use your security key with github.com

Insert your security key and touch it

Security key

When you are ready, authenticate using the button below.

Waiting for input from browser interaction...

(b) Github.com authentication popup

Fig. 7: Task 4: Synchronized login to `chase.com` while user logs into `github.com`

At the beginning of the study, participants were informed about the four tasks they would do and how Yubikeys worked. Participants were directed to a webpage containing detailed instructions for each task as they progressed through the study. The browser extension automatically detected the current task and executed the corresponding attack. We used four separate GitHub test accounts to maintain isolation between tasks. This automated setup allowed us to minimize the study coordinator's interaction with the participants and ensured privacy. We believed that providing autonomy to participants would result in behavior similar to their real-world actions. We instructed participants to try and resolve any errors they encounter to the best of their abilities and provide their reasoning for why the errors occurred and how to address them.

Recruitment We aimed to recruit technically proficient individuals to maximize the likelihood of detecting the attacks. We recruited 20 participants by making announcements in Computer Science department classes, utilizing Slack, and distributing flyers in the CS department building. To facilitate communication, we specifically selected participants who were fluent in English. Additionally, we restricted the study to individuals who had prior experience using GitHub. Initially, we offered a compensation of $10 for an expected 30-minute duration. However, we faced difficulties in obtaining sufficient participants. Consequently, we increased the compensation to $20 during the study and paid this amount to all 20 participants. On average, participants took 22 minutes to complete the study.

Demographics Out of 20 participants, 13 participants fell into the age range of 18-24, while 7 participants were in the age range of 25-34. In terms of gender distribution, 14 participants identified as male, while 6 participants identified as female. Additionally, 15 of the participants were undergraduate students, while 5 participants were graduate students.

Data collection and analysis During each task, we asked participants about their comprehension of encountered errors and their approach to resolving them. In addition, we recorded their screen activity to facilitate later analysis of their process. The research coordinator took notes of any comments related to our research questions. We analyzed the responses using inductive coding and content analysis techniques.

2) Results:

a) RQ1: For task 1, all participants successfully registered a Yubikey on a GitHub account. All participants saw the error message in Figure **??** during login, confirming the attack succeeded for each of them. As expected, none of the participants realized they were logging in with a different key than the one registered with their account. Participants provided various reasons for being unable to log in, including broken keys, improper keypresses, and lack of familiarity with the key. Three participants using a Yubikey for the first time mentioned specific reasons such as improper fingerprint scan and failure to unplug the Yubikey after registration. We believe such misunderstandings would not apply to experienced Yubikey users.

b) RQ2b: In task 2, participants were asked to register Yubikey on a GitHub account. In the background, we registered a second *HSK* with the nickname *admin*. Even when explicitly asked to check the settings page after login and verify the registered Yubikey and security of the account, only three out of twenty participants noticed two registered Yubikeys. Only one of the three mentioned they would try to identify and remove the extra Yubikey. The other two did not comment on whether they would remove it, but we assume they would take some action. Therefore, we observed three successful detections of the double-binding attack.

c) RQ3: In task 3, participants were asked to log in to GitHub using their Yubikey. The attack setup provided a cloning detection error during login on the first attempt, but upon a second attempt it successfully logged in. None of the participants associated the error with a potential cloning detection attack, as the error message did not give any indication of such an attack. The error message was identical to other error messages, such as using the wrong Yubikey. Nine out of twenty participants attempted a second login and were successful. All participants were unsure why the login failed the first time and attributed their inability to log in to an error in the authentication process. The rest of the participants mentioned they would log in using either SMS or another recovery mechanism and re-register the key as some issue could have happened with the registered *HSK*. Despite our mentioning that they have been using the same key successfully for a long time, some participants still attributed the issue to improper Yubikey configuration on their account. P4 even acknowledged the strangeness of the situation but could not connect it to any specific attack.

> P2: *"It worked the second time so it might have just been an error in when I inserted the key."*
>
> P4: *"Maybe the yubikey isn't configured correctly. I would log in with sms and check that it is configured correctly. But if I've hypothetically used it a bunch before it doesn't seem like that would be the issue. I would probably reconfigure the security key."*

d) RQ4: In task 4, participants were required to log in to their GitHub accounts. In the background, our extension authenticated first with a `chase.com` account and then with GitHub. Participants saw two popups from the browser for each of these authentications and had to tap the Yubikey twice to complete the login process. One participant encountered a technical error during the setup and was unable to log in, resulting in a sample size of 19 for this task. Among the 19 participants, all successfully logged in to their GitHub accounts by tapping the *HSK* twice. When explicitly asked if they noticed anything unusual during the authentication process, 6 participants reported no unusual observations. Thirteen participants mentioned that they had to tap the Yubikey twice, 9 attributing it to either an improper key tap or a glitch in the protocol or website. Only 1 out of 13 participants noticed the first popup prompt for "chase.com" but proceeded with authentication anyway. Two participants out of the 13 repeated the task upon seeing the question "Did you notice anything unusual?" and then noticed the `chase.com` prompt. They said they would not have noticed it without the prompt.

VI. Clone Detection Algorithm

We propose a cloning detection algorithm for account-specific counters that is not vulnerable to the stealthy cloning attack (as described in Attack 5a). The *HSK* (see Algorithm 1 below) saves a hash of the challenge it receives during registration as $hashC$. Upon receipt of an authentication request, the *HSK* includes $hashC$ in the response and updates $hashC$ with a hash of the authentication request challenge.

The *RP* (see Algorithm 2 below) maintains a linked list of hashed challenges it sends to the *HSK* in $hashList$. The *RP* initially creates the list and adds $hash(request.Challenge)$ to the list when registration completes (line 7). After sending the request, the *RP* adds the hashed challenge to the hashList. Upon receipt of an authentication response, the *RP* searches the list to determine whether it contains the hashed challenge in the response (line 10). A cloning attack is detected if it finds no matching challenge (line 11). If the hash is in the list, it removes items from the head of the list up to and including the matching challenge.

```
updateChallengeHashHSK (req)
    if $req.reg$
        $hashC$ <- $hash(req.Challenge)$
    else if $req.auth$
        $resp$ <- $genAuthnResp(hashC)$
        $hashC$ <- $hash(req.Challenge)$
```

Algorithm 1: Update Challenge hash on the HSK

```
 1  insertCurrentChallengeHashRP(req)
 2      $hashList.add(hash(req.challenge))$
 3
 4  updateChallengeHashRP(req, resp)
 5      if $resp.reg$
 6          $hashList$ <- $list()$
 7          $hashList.add(hash(req.Challenge))$
 8      else if $resp.auth$
 9          $hashC$ <- $resp.hashC$
10          if !$(hashList.contains(hashC))$
11              $cloning\ detected$
12          else
13              repeat
14                  $head$ <- $hashList.remove()$
15              until $head == hashC$
```

Algorithm 2: Update Challenge hash on the RP

After cloning occurs, whichever *HSK* logs in first (victim or attacker) will submit a matching challenge hash and proceed, while the other will fail its next login attempt. Regardless of who logs in last, the *RP* is alerted that an attack occurred and can take precautions. If the victim logs in last, they receive a clone detection alert message informing them an attack occurred so they can take appropriate action.

The hash-based clone detection algorithm is backward compatible with FIDO2. It avoids false positives when messages are lost or delayed. Suppose an authentication request (*RP->HSK*) is lost. Assume that *RP* and *HSK* each store hashes of $challenge_{i-1}$. When the *RP* sends $challenge_i$, it adds the hash to the tail of the list. If that challenge never arrives at the *HSK*, the next request will contain $challenge_{i+1}$, which is added to the list. The *HSK* will return a hash of $challenge_{i-1}$ in the response and store a hash of $challenge_{i+1}$. The *RP* will match $challenge_{i-1}$ in the list and remove it from the head of the list along with any earlier challenges still in the list. The hashes of $challenge_i$ and $challenge_{i+1}$ are still in the list.

Suppose an authentication response (*HSK->RP*) is lost. Assume that *RP* and *HSK* each store hashes of $challenge_{i-1}$. When the *RP* sends $challenge_i$, it adds the hash to the tail of the list. The *HSK* returns a hash of $challenge_{i-1}$ in its response and stores a hash of $challenge_i$. If the response never arrives, the *RP* sends $challenge_{i+1}$ in the next request and adds the hash to the tail of the list. The *HSK* includes $challenge_i$ in its response and stores a hash of $challenge_{i+1}$. When the *RP* receives a hash of $challenge_i$ in the response, it removes the hashes of $challenge_{i-1}$ and $challenge_i$ from the head of the list.

The likelihood of a false negative is negligible because the attacker authenticating with the cloned device would have to guess a correct challenge hash with a probability of 1 in 2^{256}. The following claim defines the security property of the new algorithm to prevent and detect attacks.

Claim 1. *Let a user u register HSK_u on a relying party RP for authentication. Assume an attacker $\mathcal{A}$ has physical access to HSK_u from time t to t', where $t <= t'$. Let u authenticate to RP at time t_i, where $t_i < t'$, and at time t_j, where $t_j > t'$. If $\mathcal{A}$ clones u's HSK (i.e. HSK_c) by time t', then one of the following occurs:*

1) *Both u and RP detect the clone attack during authentication at time t_j.*
2) *$\mathcal{A}$ cannot authenticate to RP as u.*

: *Proof Sketch:* Clone detection relies on a Hashed Challenge List that the *RP* maintains. The ordered list of hashes of challenges is a sliding window of challenges the *RP* has sent to the *HSK* for authentication. The *HSK* returns the challenge saved from the most recent authentication, which an *RP* will accept only once and then remove from the list.

Once the device is cloned after time t', $\mathcal{A}$ can authenticate either before or after u authenticates at time t_j. Case 1 describes what happens when $\mathcal{A}$ authenticates before u, and Case 2 describes what happens when $\mathcal{A}$ authenticates after u.

In Case 1, both HSK_u and HSK_c have a copy of the hashed challenge received at time t_i, so $\mathcal{A}$ successfully authenticates to *RP* as u and the hashed challenge at time t_i is removed from the list at the *RP*. When u authenticates at time t_j and returns the hashed challenge from time t_i, the *RP* detects the cloning attack and notifies the user.

The only way for $\mathcal{A}$ to bypass detection is to force u to authenticate at time t_j successfully. For this, $\mathcal{A}$ would need to predict the last challenge $\mathcal{A}$ receives before time t_j and send that challenge to HSK_u during the cloning process between time t and t', which is infeasible because the challenges are generated randomly for every authentication.

In Case 2, both HSK_u and HSK_c have a copy of the hashed challenge received at time t_i, so u successfully authenticates to *RP* at time t_j, and the hashed challenge at time t_i is removed from the list at the *RP*. If $\mathcal{A}$ attempts to authenticate after time t_j and returns the hashed challenge from time t_i, *RP* detects the cloning attack and $\mathcal{A}$ fails to impersonate u. To impersonate u, $\mathcal{A}$ must guess the challenge the *RP* returns to u at time t_j, which is infeasible given randomly generated challenges. Therefore, $\mathcal{A}$ cannot authenticate to the *RP*. ■

VII. Recommendations

Include and highlight nicknames, make & model in email notifications. *RP*s should send a notification to the owner of the account after every *HSK* registration, highlighting the nickname, make & model of the *HSK* and the total number of registered *HSK*s. This notification may help users detect the registration of an unrecognized *HSK* in Attack 2. The notification could also be effective against Attack 1 if the adversary uses an *HSK* with a different make and model than the victim.

Require *HSK* authentication before adding a second *HSK*. An authenticated user can usually register additional *HSK*s. But an adversary can register a malicious *HSK* in the background without user knowledge *i.e.* Attack 2b. Therefore, an *RP* should require authentication with a registered *HSK* before registering an additional *HSK*. However, this presents a problem when a user loses their *HSK* and wants to register a new one. In this case, the user must remove the lost *HSK* during an already logged-in session using less secure authentication methods like a password. The requirement to either (1) authenticate with a previous *HSK* or (2) remove a lost *HSK* ensures that either the attacker cannot register their *HSK* or the victim detects the removal of their *HSK* on the account.

Provide more specific context in error messages. To mitigate Attack 7a, device cloning error messages should explicitly state that (1) the device is cloned, (2) remove the device from the account, and (3) register another *HSK*. The failed authentication following device cloning could happen to either the victim or $\mathcal{A}$, depending on who authenticates last following a cloning attack. Future work could analyze an *RP* notifying a victim out-of-band to make them aware the attack occurred when $\mathcal{A}$ authenticates last.

When a user attempts to log in with a different key than the one registered, an informative error message explaining the problem would help users detect potential misbinding attacks or resolve the issue if they have multiple *HSK*s. To create these error messages, researchers should engage users by conducting interviews and evaluating the effectiveness of different alternatives.

VIII. Related Work

Prior work has investigated the security guarantees of the first FIDO protocols—Unified Authentication Framework (UAF) and Unified 2nd Factor (U2F). UAF is a passwordless authentication system, and U2F is a standardized 2FA system. Hu et al. performed the first informal analysis of UAF and outlined three attacks [12]. They identified the mis-binding attack where an attacker binds their authenticator to a *RP* instead of the user's authenticator.

Panos et al. also analyzed UAF [19]. They identified attack vectors that lead to system compromise. The threat model includes attackers with access to the authenticator and control of the UAF client.

Peirera et al. performed the first formal analysis of FIDO1 [21]. Their threat model includes a network attacker and an attacker that compromises the client or server. They verified the security of the protocol as long as the *RP* is verified. The analysis investigated only authentication, not registration.

Feng et al. [10] conducted a comprehensive analysis of the FIDO UAF protocol, confirming the mis-binding attack and identifying a parallel session attack. Building on their findings, we demonstrated the practical feasibility of the parallel session attack in FIDO2, which we call MITM - Attack 4.

FIDO2/WebAuthn Eventually, UAF and U2F merged into the W3C standardized protocol, FIDO2, which supports both passwordless authentication and 2FA. Guirat et al. [11] presented a formal proof of the protocol, with a threat model of a passive and active network attacker. They did not consider a compromised client. Finally, Jacomme et al. formally analyzed several multi-factor authentication schemes, including FIDO2 [13]. The threat model includes malware, fingerprint spoofing, and human error. FIDO2 was shown not to be provably secure against malware.

Other studies [6], [7], [16] investigated the usability challenges and perceived benefits of FIDO2 device usage. Alqubaisi et al. [4] evaluated the threats of password attacks and compared passwords to single-factor FIDO2. Their findings suggest that single-factor FIDO2 performs better against the password-based threat model, but they do not address targeted attacks on the FIDO2 protocol.

Chang et al. [5] exposed the weaknesses of U2F to side-channel and MITM attacks and proposed a modification to the U2F protocol to mitigate side-channel attacks. O'Flynn [18] also expanded the threat model by showing an attack on *HSK*s through Electromagnetic Fault Injection, which led to recovering secret data. Dauterman et al. considered the threat posed by hardware backdoors to *HSK*s and proposed True2F [8], a strengthened version of U2F. True2F protects against a malicious *HSK* and increases the protection against token fingerprinting. While this work does consider a compromised browser, it asserts that if the True2F token behaves faithfully, it is no less secure than traditional U2F. Our work is complementary and makes *HSK*s more secure from a compromised browser than traditional U2F.

Remember this Device (RTD) Patat et al. [20] identified RTD vulnerabilities for U2F devices by popular websites and proposed replacing cookies with a "soft U2F token." The token resists eavesdropping but not theft by a compromised client.

IX. Conclusion

FIDO2 has primarily focused on defending against attacks from afar by remote attackers that compromise a password or attempt to phish the user. In this paper, we explored threats from local attacks on FIDO2 that have received less attention—a browser extension compromise and attackers gaining physical access to an HSK.

We systematically analyzed local attacks on FIDO2. We demonstrated that some attacks are feasible against popular websites supporting FIDO2. Our systematization reveals four underlying flaws that lead to these attacks.

In addition, we showed that the threat to FIDO2 authentication from compromised browser extensions is real by providing data showing that many extensions have sufficient permissions to attack FIDO2. We then conducted a static and dynamic analysis of existing extensions and found no evidence that these attacks occur in the wild.

We conducted two user studies and found that participants did not detect the attacks from current error messages, email alerts, and other UX responses to the attacks. Future work can explore more effective ways to alert users of these attacks.

Disclosure We shared our results with Yubico and Firefox. Although the attacks are outside the current threat model, they appreciated receiving our results for future consideration.

Availability The code for our FIDO2 attack demonstrations is available to researchers upon request.

Acknowledgment

The authors thank the reviewers for their helpful feedback on the paper. We thank Alden Howard for assistance with the extensions analysis on the Chrome store and Hayden Taylor for assistance with the user study. We also thank Scott Ruoti and Garrett Smith for their feedback on an earlier draft of the paper, and Trevor McClellan and Parker Hanson for their feedback on the camera-ready draft of the paper.

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1816929.

References

[1] "Discovery of a massive, criminal surveillance campaign," "https://awakesecurity.com/blog/the-internets-new-arms-dealers\ \-malicious-domain-registrars/".

[2] "Google removes 500+ malicious chrome extensions," "https://www.ositcom.com/61".

[3] "Threat intelligence feeds and endpoint protection systems fail to detect 24 malicious chrome extensions," "https://www.catonetworks.com/blog/threat-intelligence-feeds-and-endpoint-\ \protection-systems-fail-to-detect-24\\-malicious-chrome-extensions/".

[4] F. Alqubaisi, A. S. Wazan, L. Ahmad, and D. W. Chadwick, "Should we rush to implement password-less single factor FIDO2 based authentication?" in *2020 12th Annual Undergraduate Research Conference on Applied Computing (URC)*. IEEE, 2020, pp. 1–6.

[5] D. Chang, S. Mishra, S. K. Sanadhya, and A. P. Singh, "On making U2F protocol leakage-resilient via re-keying." *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 721, 2017.

[6] S. Ciolino, S. Parkin, and P. 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.

[7] S. Das, A. Dingman, and L. J. 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*. Springer, 2018, pp. 160–179.

[8] E. Dauterman, H. Corrigan-Gibbs, D. Mazières, D. Boneh, and D. Rizzo, "True2f: Backdoor-resistant authentication tokens," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 398–416.

[9] Duo and Cisco, "Crxcavator chrome extension permissions," "https://crxcavator.io/".

[10] H. Feng, H. Li, X. Pan, Z. Zhao, and T. Cactilab, "A formal analysis of the FIDO UAF protocol." in *NDSS*, 2021.

[11] I. B. Guirat and H. Halpin, "Formal verification of the W3C web authentication protocol," in *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*, 2018, pp. 1–10.

[12] K. Hu and Z. Zhang, "Security analysis of an attractive online authentication standard: FIDO UAF protocol," *China Communications*, vol. 13, no. 12, pp. 189–198, 2016.

[13] C. Jacomme and S. Kremer, "An extensive formal analysis of multi-factor authentication protocols," in *IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 2018, pp. 1–15.

[14] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson, "Hulk: Eliciting malicious behavior in browser extensions," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 641–654.

[15] V. Lomne and T. Roche, "A side journey to titan, side-channel attack on the google titan security key (revealing and breaking NXP's P5x ECDSA implementation on the way)," NinjaLab, 161 rue Ada, 34095 Montpellier, France, Tech. Rep., January 2021. [Online]. Available: https://ninjalab.io/wp-content/uploads/2021/01/a_side_journey_to_titan.pdf

[16] S. G. Lyastani, M. Schilling, M. Neumayr, M. Backes, and S. Bugiel, "Is FIDO2 the kingslayer of user authentication? a comparative usability study of FIDO2 passwordless authentication," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 268–285.

[17] Mozilla, "Permissions-policy," "https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Permissions-Policy/publickey-credentials-get".

[18] C. O'Flynn, "MIN()imum failure:EMFI attacks against USB stacks," in *13th USENIX Workshop on Offensive Technologies (WOOT 19)*, 2019.

[19] C. Panos, S. Malliaros, C. Ntantogian, A. Panou, and C. Xenakis, "A security evaluation of FIDO's UAF protocol in mobile and embedded devices," in *International Tyrrhenian Workshop on Digital Communication*. Springer, 2017, pp. 127–142.

[20] G. Patat and M. Sabt, "Please remember me: Security analysis of U2F remember me implementations in the wild," in *Actes SSTIC 2020, 18th Information and Communications Technology Security Symposium (SSTIC 2020)*, 2020.

[21] O. Pereira, F. Rochet, and C. Wiedling, "Formal analysis of the FIDO 1. x protocol," in *International Symposium on Foundations and Practice of Security*. Springer, 2017, pp. 68–82.

[22] T. Roche, V. Lomné, C. Mutschler, and L. Imbert, "A side journey to titan," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 231–248.

[23] T. Roth, F. Freyer, M. Hollick, and J. Classen, "Airtag of the clones: Shenanigans with liberated item finders," in *2022 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2022, pp. 301–311.