Breaching Security Keys without Root: FIDO2 Deception Attacks via Overlays exploiting Limited Display Authenticators

Ahmed Tanvir Mahdad
Texas A&M University
College Station, TX, USA
mahdad@tamu.edu

Mohammed Jubur*
Jazan University
Jazan, Saudi Arabia
mjabour@jazanu.edu.sa

Nitesh Saxena
Texas A&M University
College Station, TX, USA
nsaxena@tamu.edu

Abstract

Two-factor authentication (2FA) systems aim to secure user accounts, provided that either the password or the second factor device remains uncompromised. However, in this research, we challenge this perception and analyze the security of FIDO2 hardware security keys, which are increasingly used in 2FA and passwordless systems. Specifically, we develop an attack framework, analyze the underlying protocols of FIDO2, and examine the associated OS-level security. Through practical demonstrations, we illustrate how adversaries can exploit this framework and OS-level security measures to execute our designed attack, known as FIDOLA (**FI**DO2 **D**eception Attack via **O**verlays exploiting **L**imited Display **A**uthenticators).

Our attack framework injects hidden login sessions, either into the same service the user intends to authenticate with or into a different service. It deceives users into approving the attacker's request using the limited display of authenticators. This cross-service attack raises concerns about compromising more sensitive accounts (e.g., financial) when users log into less sensitive ones. Our attack poses a practical and fundamental threat not addressed in the FIDO specification or prior research. Unlike prior research, our demonstration exposes FIDO2 authenticator vulnerabilities in real-world 2FA and passwordless setups, where OS-level security mitigates traditional concurrent attacks (simultaneous authentication attempts by the attacker). To assess our attack's effectiveness, we conducted a user study, revealing that users approved approximately 95.55% of cross-service attacks when presented with a screen overlay.

CCS Concepts

• **Security and privacy → Multi-factor authentication**.

Keywords

2FA; FIDO; WebAuthn; CTAP2; Security key; attack; overlay

*Work done as a PhD student in the SPIES Lab.

1 Introduction

Password-only authentication is a prevalent method for web authentication, yet it faces security and usability challenges. To address these concerns and provide an additional layer of security in the event of password compromise, Two-factor Authentication (2FA) and passwordless systems have been introduced. In 2FA systems, users must demonstrate possession of a pre-registered device alongside passwords. Similarly, in passwordless systems, possession factor devices are used along with other factors such as PINs or biometrics to strengthen security. However, OTP (One Time PIN), which is widely used as the most popular 2FA system, has limitations. Therefore, service providers have adopted advanced 2FA and passwordless systems such as FIDO2 security keys.

In this work, we focus on 2FA and passwordless systems that utilize FIDO2 security keys as possession factor devices (e.g., Yubikey [46]), which we refer to as *Challenge-Response 2FA* (*CR-2FA*) throughout this paper. These security keys implement the Fast Identity Online (FIDO) [20], an open authentication standard. We analyzed FIDO2 [17] and its underlying protocols (WebAuthn [13], CTAP2 [19]). To evaluate the security of FIDO2 keys, we propose an attack framework called *FIDO2 Deception Attack based on Overlays exploiting Limited Display Authenticators (FIDOLA)*, designed to perform attacks leveraging the limited display capabilities of FIDO2 authenticators (i.e., USB security keys). FIDOLA is capable of performing both same-service attacks (where the attacker logs in to the same service as the user) and cross-service attacks (where the attacker logs in to a different service than the user intended) by deceiving the user into accepting the attacker-generated session thinking it is user's own session.

In addition to the security measures imposed by WebAuthn and CTAP, operating systems such as Windows introduce OS-level security that effectively thwarts traditional concurrent attacks that attempt to send malicious requests simultaneously to the FIDO authenticator (e.g., security key) and exploit the user's tap to approve their request. However, OS-level security prevents these attacks by requiring browsers to call a specific API, which restricts multiple requests from being sent to the authenticator. Moreover, it displays the browser name and service name to the user for enhanced transparency and security (as shown in Figure 1). These concurrent attacks have already been reported in the literature [10, 26]. Nevertheless, with the implementation of OS-level security, traditional concurrent attacks are mitigated, providing an additional layer of protection to FIDO-based authentication systems.

Figure 1: Snapshot of Windows security message.

> It has been claimed in the literature [28] that only a powerful attacker model, such as root-level malware, can bypass OS-level security. However, FIDOLA, which operates in user space, demonstrates the capability to bypass OS-level security without requiring root-level access.

Through our analysis of WebAuthn and CTAP2, we demonstrate how FIDOLA can effectively bypass the security measures added by these protocols. Notably, the FIDO alliance threat analysis [18] did not address an attack model similar to FIDOLA. Furthermore, in contrast to previous works, FIDOLA is thoughtfully designed to block the legitimate user's request and send only a single request to the FIDO2 authenticator during the attack, thereby enabling it to bypass OS-level security measures, which are intended to prevent traditional concurrent attacks.

Most interestingly, in addition to performing same-service attacks, FIDOLA is capable of performing *cross-service attack*, where the adversary can send an authentication request to a different service (e.g., Twitter) than the service the user is trying to authenticate to (e.g., Google). FIDOLA exploits the *limited interface* (i.e., flashing LED button) used in FIDO2 authenticators and uses an overlay to deceive users to approve the attacker's request. When the operating system displays a security message containing the requested service and browser name (as shown in Figure 1), FIDOLA can overlay a message with altered information in real-time. This overlay message appears with the same size and format as the security message, aiming to convince users to accept the attacker's request. Here, the limited display of FIDO2 keys (blinking LED button) does not provide users with meaningful information (such as the service name) when attackers overlay an OS security message. This approach is particularly useful in FIDOLA's cross-service attack, where the attacker's intended service name differs from that of the user. Such a cross-service attack poses a high risk of compromising a high-value account (e.g., bank) when users are attempting to authenticate in a low-value account (e.g., email). In fact, we conducted a user study that revealed that participants could not identify 95.55% of the overlays in the cross-service attack, thereby establishing the effectiveness of the overlay-based cross-service attack we designed.

Previous literature (e.g., Bellare and Rogaway [7]) suggests that an adversary needs to impersonate the trusted users independently without relaying the user's input/session (e.g., session hijacking attack) to construct a valid attack on any cryptographic protocol. In that sense, all variants of session hijacking attacks (e.g., session hijacking by malware, active phishing, cookie stealing) are not valid attacks against a protocol. However, FIDOLA launches an independent session from the background, which also involves users in establishing user presence. FIDOLA constructs a valid "non-relay" attack that reveals vulnerabilities in FIDO2 and its underlying protocols.

> FIDOLA is fundamentally different from and significantly more devastating than a typical session hijacking attack.

In the session hijacking attack, the malicious entity compromises the user's ongoing session, which gives the attacker only a limited capability, such as only allowing for the same-service and same-session attack. On the other hand, FIDOLA is more effective, flexible, and stealthy, with the significant advantage of cross-service attack capability. We provide a comparison between session hijacking and FIDOLA in Table 1 and Figure 4.

> FIDOLA incorporates non-root malware components (e.g., keylogger, hidden browser session, browser extension) that do not require administrative privilege for installation or execution, operating within the user space of the operating system.

These components are commonly utilized by adversaries [8, 11, 24, 25] and previous works (e.g., Kucchal et al. [28]). Given the extensive adoption of common malware components and the capability of bypassing Windows Security, FIDOLA poses practical and real-world threats to 2FA and passwordless systems that rely on FIDO2. We refer the reader to our detailed exposition about the practicality aspects of FIDOLA pertaining to the underlying assumptions and threat model in our technical sections (Section 3.2, Section 4.2.1 ©1, and Section 4.2.1 ©2.)

Notably, the "User-space malware in the terminal" threat model has been included in the FIDO threat model, except in cases where it is utilized for session hijacking attacks, as noted by Lang et al. [29]. Furthermore, the user-space malware in the terminal threat model is prevalent in recent literature focused on evaluating the security of FIDO2 keys [10, 28]. As such, this threat model is an integral part of the overall understanding and evaluation of security vulnerabilities in *CR-2FA*.

Contributions: Our contributions in this research are three-fold:

(1) ***Designed an attack framework capable of performing attacks on FIDO2 Protocol bypassing OS-level security***: We design the FIDOLA attack framework, which exploits vulnerabilities in FIDO2 and its subprotocols (WebAuthn [13] and CTAP2 [19]) and bypasses OS-level security to demonstrate both same-service and cross-service attacks in real-world settings. It is a fundamental attack approach that has not been considered in threats in the FIDO specification [18] or previous literature.

(2) ***Developed a proof-of-concept attack with minimal resource consumption and dependencies:*** We develop a proof-of-concept attack to evaluate real-world *CR-2FA* deployments,

Table 1: Difference between our proposed attack and Session hijacking attack

Feature	Session Hijacking	FIDOLA
Existing vs. Fresh Session	Only allows to take over an active session that user-initiated	Allows to **inject a new, independent session** when the user initiates her own session. The new session can be for a more sensitive service (e.g., banking) than a user-initiated session (e.g., email).
Limited vs. Full-Control	Can only manipulate the user's ongoing activity and cannot do anything arbitrary.	Allows the attacker to have **full arbitrary control** over the new session.
User Dependency	If user's session terminates when transaction is underway, attack will be disrupted	**Attack can continue** even after the user session terminates.
Detectability	Attackers may have to linger on the terminal for a long time, as it will have to wait for the user to perform the desired activity (e.g., bank transaction). This makes the attack more easily detectable.	Attackers can quickly perform a fraudulent activity (as it is independent of the user activity) and leave the terminal, thereby remaining **more stealthy**.
Cross-Service Attack Possible?	No. The attacker can only perform attacks on the same service the user is trying to log in to	Yes. The attacker can perform attacks on the same or different service the user is trying to login to

Figure 2: Workflow of WebAuthn Registration

requiring minimal resources and installation without any administrative privileges. Our evaluation, using both anti-malware programs and user studies, demonstrates low detectability. Additionally, we successfully demonstrate both same-service and cross-service attacks on popular *CR-2FA* deployments (e.g., Google, Twitter) using well-known *CR-2FA* authenticators (e.g., Google Titan, Yubikey). This demonstration provides concrete evidence of the real-world applicability of FIDOLA.

(3) ***Evaluated cross-service attack's stealthiness through a user study:*** We conducted a user study that demonstrated the stealthiness and effectiveness of cross-service attacks. We observed that, in 95.55% of cases, participants were unable to detect carefully crafted malicious cross-service requests. This observation confirms the practicality and efficiency of our designed cross-service attack.

Attack Demonstrations: We provide our attack demonstration at: https://sites.google.com/view/cr-2fa-attack-demo/home.

2 Background and Systematization

Generally, authenticators (e.g., Yubikey) and devices with built-in security keys utilize the FIDO U2F and FIDO2 protocols, providing users with a high level of security (AAL3) through hardware-based phishing-resistant authentication mechanisms [36].

2.1 The Evolution of FIDO Protocol

FIDO released the passwordless protocol FIDO UAF (Universal Authentication Framework) and the second-factor protocol FIDO U2F in December 2014. FIDO2, in conjunction with the W3C WebAuthn protocol and CTAP2 (Client-to-Authenticator Protocol), was launched in January 2019. FIDO U2F acts as a second-factor protocol, whereas FIDO UAF primarily emphasizes passwordless authentication, especially for mobile devices.
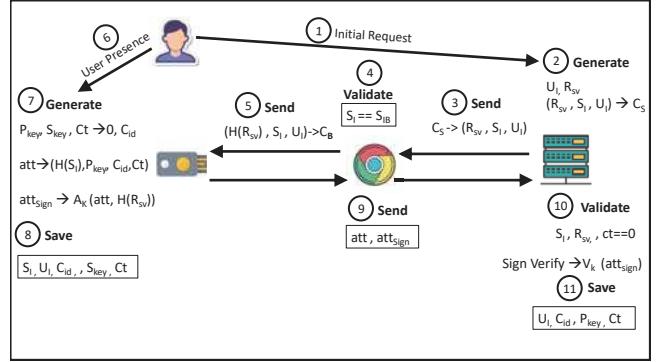
2.2 The W3C WebAuthn Protocol

The W3C (World Wide Web Consortium) WebAuthn is a specification written jointly by FIDO and W3C, and it is supported by all major browsers. The high-level workflow is discussed below.

Registration: A high-level workflow of the WebAuthn Registration procedure is illustrated in Figure 2. After receiving initial request, the server generates a user ID U_I and a random string R_{SV}. It then creates a challenge C_S by combining the server identity (i.e., domain name) S_I with the generated U_I and R_{SV}. In the next step, it sends the challenge C_S to the client (i.e., browser). The client extracts the server identity S_I and compares it to the web origin from the client S_{IB}. If the comparison fails, the protocol halts at that moment, and the registration procedure will fail. If it passes, then it creates a hash of the received random string $H(R_{SV})$ and passes the challenge to the authenticator (i.e., security key).

After getting the challenge, the authenticator generates a key pair (S_{key}, P_{key}). Then, it generates credential ID C_{id} and sets counter ct to zero. The authenticator then creates a hash of server identity $H(S_I)$, attestation signature of $H(S_I)$, public key P_{key}, and random hash $H(R_{SV})$ using attestation private key A_k. The authenticator saves S_I, U_I, C_{id}, secret key S_{key}, and counter ct for future authentication. It sends a response containing attestation signature, $H(S_I)$, ct, P_{key}, C_{id}, and $H(R_{SV})$ to the client which will be redirected to the service. After that, the authentication server validates S_I, R_{SV}, and checks if the counter ct is set to zero. It also validates the attestation signature with attestation public key V_k. If all checks pass, then the registration process will be successful, and the server saves U_I, C_{id}, P_{key}, and counter ct for future use.

Authentication: The workflow of the WebAuthn authentication procedure is shown in Figure 3. After receiving initial authentication request, the server creates a challenge containing server ID S_I and a random string R_{SV} and sends it to the client. The client checks the validity of S_I with web origin S_{IB}. If matched, it creates a hash of the received random string $H(R_{SV})$ and sends the challenge to the authenticator; otherwise, it halts the procedure.

Upon receiving the challenge, the authenticator retrieves user id U_I, credential id C_{id}, S_{key}, and counter ct. After getting the user response, it increases the counter and creates a hash of server id $H(S_I)$. Finally, it creates an attestation signature using the secret
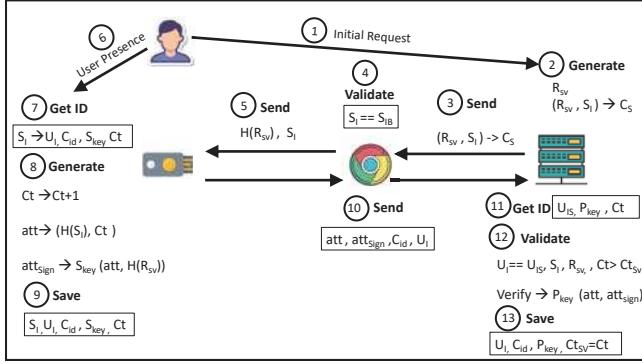
Figure 3: Workflow of WebAuthn Authentication

key S_{key}. The authenticator saves S_I, C_{id}, secret key S_{key}, and updated counter ct for future authentication. It then sends a response containing a signature, U_I, and C_{id}, to the client, which redirects it to the authentication server. Then, the server will retrieve information using credential id C_{id}, P_{key}, and saved user id U_{IS}. After that, server check validity of U_I, S_I, R_{SV}, check if ct is greater than stored counter ct_{sv}, verifies the attestation signature by P_{key}. If all checks are passed, then the server saves U_I, C_{id}, P_{key} and updated counter as ct_{sv} and sends authentication success decisions to the client.

2.3 Client-to-Authenticator Protocol (CTAP)

Client-to-Authenticator Protocol (CTAP) is a protocol developed as a complementary tool for the WebAuthn protocol. We illustrate the high-level workflow of the CTAP2 protocol in Figure 5.

High-level Workflow: The authenticator initializes the pinToken pt and binding state bs during setup. When the user sets and confirms their PIN on the client, it passes encrypted PIN $E(PIN)$ to the authenticator. Upon receiving, the authenticator checks the validity of PIN and saves it if it is valid. During the binding process, the client prompts for a PIN from the user. Upon receiving the PIN, the client encrypts the PIN and sends it to the authenticator for verification. A counter is set with the maximum retry number $retry_c$. If the provided PIN is correct, then it stores the PIN as Sk_{pin} in the authenticator. If it is wrong, then $retry_c$ will be reduced by one (1) until it reaches zero. If $retry_c$ =0, the authenticator will discard the binding process and enforce user interaction to start the reboot process. After successful binding, it sets pinToken pt as binding state bs_{sk} and sends pt to the client. The client then sets its binding state bs_c.

During the authorization, the client sends the hash of its binding state bs_c to the authenticator for validation. It also maintains the retry counter $retry_c$ to prevent PIN guessing attacks. The authenticator will match the hash of its binding state with the received client's binding state. If the validation passes, it sends a confirmation to the client. The client communicates with the authenticator to accomplish the registration or authentication process in WebAuthn.

2.4 OS Level Security

Operating systems may implement additional security measures to safeguard FIDO2 keys against various attacks. For instance, Windows security requires the browser to invoke a Windows OS-specific API before sending a request (see step 5 in Figure 3) in WebAuthn workflow. This process is illustrated in the modified Figure 6, where step 5 of the WebAuthn workflow ($H(R_{SV})$, S_I, B_N) is sent to the Windows security API. Subsequently, the API generates a message containing the service name (S_I) and the browser name (B_N), which are then sent to the authenticator, leading to the flashing of the authenticator button.

By displaying a message with the service and browser name, the Windows security API assists users in making informed decisions, thereby thwarting unwanted cross-service attacks. Additionally, the Windows security API enforces a prohibition on concurrent requests by allowing only single request at a time.

3 Threat Analysis

3.1 Risk of Limited Display in Authenticator

In 2FA, detailed information (e.g., service name) about authentication requests is crucial for making an informed decision. However, USB FIDO2 keys use a flashing LED button as communication medium, which is a limited display lacking essential identifying information.

We leverage this limited display in our work and design a concurrent attack from the background. While some desktop operating systems (e.g., Windows 10) display a security message with the service name, it cannot be considered entirely secure if the terminal is compromised. This vulnerability allows a malicious program to overlay any part of the screen, effectively hiding the service information shown in the OS security message, and launching an attack for a service other than the one the user intended to use. This malicious overlay prompt is synchronized to be shown when the Windows security message is shown to hide the message and it disappears after some time.

We refer to this attack as the "Cross-service" attack and will discuss it further later. .

3.2 Risk of Desktop-based Malware Infection

Our proof-of-concept attack program targets Windows as the operating system (with 38% market share) [41], and Google Chrome as the browser (with 65% market share [39]), exposing vulnerabilities in the most popular desktop OS/browser combinations. Malware with capabilities similar to FIDOLA is prevalent. For instance, Zeus is a widespread keylogging malware [14]. Additionally, a recent article reveals the identification of 500 malicious Chrome extensions that possess similar abilities, such as redirecting victims to malicious websites [43], resembling our attack. Another report indicates that 50% of malicious traffic originates from headless browsers, including those used in our attack (e.g., PhantomJS, Chrome Headless) [24]. Hence, we consider our proposed threat model for *CR-2FA* to be standard and practical in real-world scenarios.
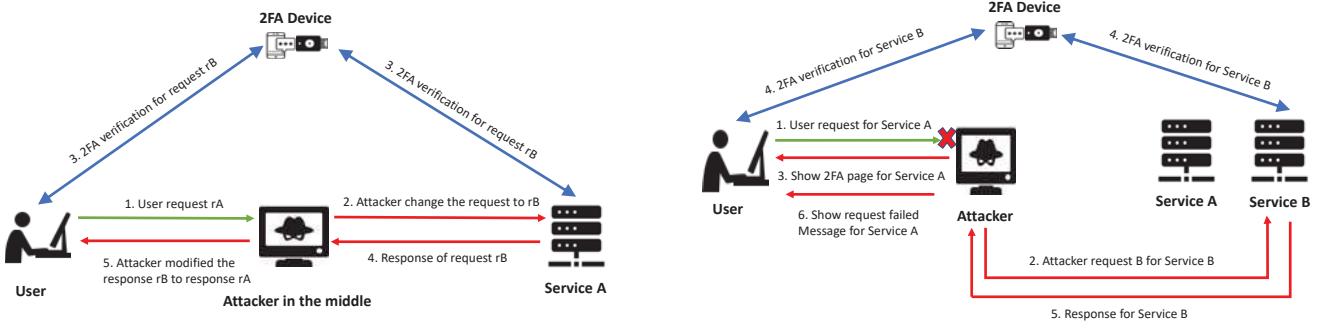
(a) Session Hijacking attack workflow where a man-in-the-middle attack is launched, which changes user requests rA to rB to service A. After getting response rB, attacker shows users altered response rA. This workflow shows the Session Hijacking attack can be performed only on the same service A.

(b) The workflow of FIDOLA attack where the attacker can block any request to service A and send different request B on different service B and get it approved by 2FA

Figure 4: Comparison between workflow of Session Hijacking Attack and FIDOLA Attack
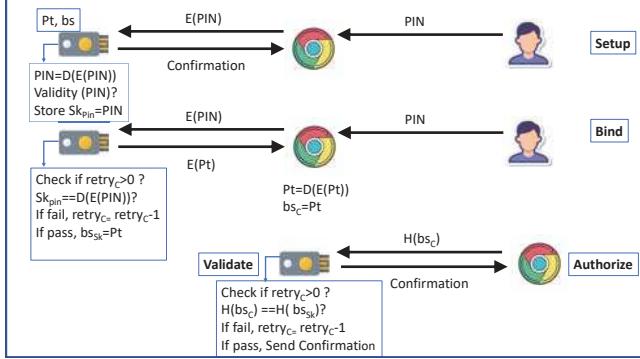
Figure 5: Workflow of CTAP2 Protocol
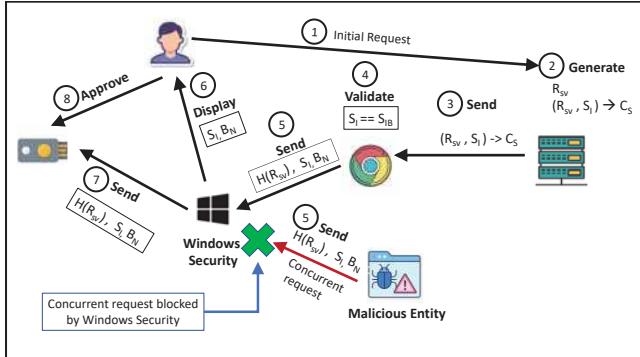
Figure 6: A workflow example of OS Level security in WebAuthn

3.3 Previous Works

Researchers have already reported many security [6, 26, 45] and usability [16, 30, 31] issues in literature.

Bui et al. introduced the man-in-the-machine attack (MitMa) in [10]. They utilized a browser process to capture the server's client data object and continuously transmitted it to the USB. In

Table 2: Comparison of FIDOLA with previous works.

	FIDOLA	MitMa [10]	Jacomme et al. [26]	Kucchal et al. [28]
Bypass Windows Security	✓	✗	✗	✗
Real World Demo	✓	✗	✗	✓
Overlay for User Deception	✓	✗	✗	✗
Evaluation via User Study	✓	✗	✗	✗

contrast, our approach involves launching a hidden browser session that sends a single client data to the USB, similar to the normal workflow. Unlike MitMa, which sends multiple requests and is susceptible to detection by anti-malware programs, FIDOLA sends only one request, significantly reducing the chances of detection. While Windows security can prevent MitMa by permitting only one request at a time and displaying a security message (explained in Section 2.4 and Figure 6), FIDOLA remains unaffected by this OS-specific feature.

Jacomme et al. have proposed another concurrent attack on FIDO [26], specifically targeting FIDO U2F. They generated two concurrent requests at the same time and showed that both requests could reach the authenticator simultaneously. They assumed that the attacker's request arrives first and is approved by the user's first tap. However, since the user's request is not approved, they may think that the tap is not registered, so they tap the button again. The later tap will authenticate the user's request. However, unlike FIDOLA, this concurrent attack cannot bypass Windows security because it only allows a single request to be sent to the authenticators at a time. Furthermore, their analysis only covers the FIDO U2F protocol, while we demonstrate attacks using the latest FIDO2 keys.

Kucchal et al. [28] conducted an evaluation of real-world FIDO2 deployments, demonstrating how malware in the user's terminal can bypass FIDO2 authentication attempts. However, their non-root

malware attack variant, which relies on malware running exclusively in user space, cannot bypass the OS-level security measures outlined in Section 2.4. These security features are particularly prevalent in Windows, the most widely adopted desktop operating system [40]. Consequently, their approach is ineffective for the majority of real-world deployment scenarios. In contrast, our attacker model considers malware running exclusively in user space, yet it has the capability to bypass OS-level security. Additionally, we conducted a user study that provides empirical support for our attack methodology, revealing a remarkable 95.55% success rate.

A detailed comparison of these works with our research is provided in Table 2.

Barbosa et al. [6] performed a detailed security analysis of FIDO2, WebAuthn, and CTAP2. They have pointed out some shortcomings of CTAP2 (i.e., they proved that CTAP2 is not UF (unforgeability)-secure) and proposed a protocol (WebAuthn+sPACA) that is UF-secure. Their proposed protocol, sPACA, differs from CTAP2 in bind execution. On the other hand, in our work, we identify (WebAuthn+CTAP2)'s weaknesses against our designed attacks, especially cross-service attacks, and evaluate their security against them.

Jubur et al. presented a variant of the concurrent attack [27], primarily for push notification authentication. The attacker generates multiple same-service concurrent notifications to deceive users into accepting attacker-controlled requests. In contrast, FIDOLA, which is designed for FIDO2 keys, can generate a single request capable of bypassing OS-level security and performing cross-service attacks.

4 Attack Design and Implementation

4.1 Definition of "Non-Relay-Attack"

A powerful adversary can compromise the communication medium, capture users' requests, and relay them to impersonate. A session hijacking attack steals session identifiers and impersonates the user's session. They are known as a "Relay Attack," which is not considered a valid attack against a cryptographic protocol. According to Bellare-Rogaway [7], "*an adversary in our setting can always make the parties accept by faithfully relaying messages among the communication partners. But this behavior does not constitute a damaging attack; indeed, the adversary has functioned just like a wire, and may as well not have been there*".

A valid attack on protocol should be one that the user does not initiate; nevertheless, it has the capability to impersonate the user and successfully authenticate as the user. According to Bellare-Rogaway [7], "*we formalize that a protocol is secure if the only way that an adversary can get a party to accept is by faithfully relaying messages in this manner. In other words, any adversary effectively behaves as a trusted wire, if not a broken one*". We formulate a valid "Non-Relay-Attack" that creates an independent session different from the users' created session, unlike session hijacking.

4.2 Threat Model

To assess the *CR-2FA* system, we have designed the FIDOLA attack framework, which encompasses specific attacker capabilities and assumptions. These align with the criteria of a "non-relay attack" and adhere to the FIDO2 attacker model. We have listed the capabilities and assumptions below.

4.2.1 Attacker Capabilities. $\mathbb{C}1$: *The adversary is capable of persuading users to click on links distributed through different channels (e.g., websites, emails), leading to the injection of malicious executables of FIDOLA into the users' computer.*

> **Justification of Practicality.** This attacker capability aligns with well-known real-world attacks against other 2FA schemes that compromise the terminal (e.g., Zeus [14]). The "user-space malware in the terminal" model has also been addressed in the FIDO threat model, with an exception when it is used to "ride" on the user's ongoing session (i.e., a session hijacking attack), as outlined by Lang et al. [29], which is an invalid attack according to Bellare-Rogaway [7] anyway. Furthermore, researchers have already considered a malware-in-terminal threat model (covering both user-space and system-level malware) to evaluate FIDO2 security [28], demonstrating the practicality of this attacker capability in assessing the risks associated with FIDO2 protocols.

$\mathbb{C}2$: *The attacker can convince users to install a seemingly harmless but malicious browser extension in their browsers.*

> **Justification of Practicality.** Installing useful browser extensions is a very common use case that may be exploited by attackers to inject malicious code into a seemingly benign and useful browser extension (e.g., e-commerce tool). This use case has already been reported as vulnerable and exploited by real-world attackers as a common entry point for attacks [8, 23, 43]. It is important to note that, as our objective is to design a non-relay and valid attack for the analysis and evaluation of *CR-2FA* systems, we do not consider capturing session cookies using a browser extension.

4.2.2 Assumptions. $\mathbb{A}1$: *The malicious component of FIDOLA only operates within the user space of the operating system. It does not require any system-level permissions or alter any core OS components or functionalities.*

> **Justification of Practicality.** This assumption distinguishes FIDOLA from strong attacker models that presume total compromise of the operating system in the user terminal. It also guarantees a more straightforward attack compared to system-level malwares. Furthermore, this characteristic grants FIDOLA a notable advantage, enabling it to function in restricted user terminals, such as organizational computers subject to various system-level restrictions, including restrictions on the installation of unknown programs.

$\mathbb{A}2$: *The attacker cannot execute a total browser compromise, which encompasses actions such as installing malicious browsers, convincing users to use them, altering core components of users' browsers, or modifying FIDO2 protocol components (e.g., WebAuthn) within the users' browser.*

Justification of Practicality. As one of our objectives is to adhere to the FIDO2 threat analysis [17] during our evaluation, our designed attack cannot involve total browser compromise, as this would imply compromising the FIDO2 client. Compromising the FIDO2 client and its components within the browser (e.g., WebAuthn) is beyond the scope of the FIDO2 threat analysis [17]. Moreover, implementing a malicious browser, integrating FIDO into it, and persuading users to install it from an illegitimate source require significantly more technical and social engineering effort from the attacker compared to our proposed attacker model. Importantly, the installation of a malicious browser or the total compromise of a user's browser typically requires administrative privileges on organizational computers, contradicting our previously stated Assumption 𝔸1.

𝔸**3**: *Users are assumed to utilize 2FA for all login attempts, and they will not use "Remember Me" features.*

Justification of Practicality. These assumptions are necessary to evaluate the performance of 2FA systems when both factors (password and FIDO2 keys) are needed for authentication. The "Remember me" feature or saving passwords in the browser would disable one or both factors, thereby undermining the full security of 2FA during security evaluations.

𝔸**4**: *The adversary cannot compromise FIDO2 Authenticators (i.e., hardware security keys).*

Justification of Practicality. FIDO2 authenticators, such as physical FIDO2 USB keys, contain secure hardware that runs a specific firmware. This firmware is inaccessible from outside, as it can only be accessed by some specific API. Furthermore, malicious FIDO authenticators are out of scope of FIDO2 security model.

𝔸**5**: *The adversary can not have any control over the user's already established session or can not steal any session cookies.*

Justification of Practicality. As FIDOLA is designed to be a valid attack against the FIDO2 protocol, we do not consider "relay attacks" such as session hijacking in its approach.

4.2.3   2FA against "User-Space Malware in Terminal". As per the capabilities and assumptions of the attacker, we are adopting a "User-space Malware in Terminal" threat model to assess FIDO2 key-based authentication systems. Here, to evaluate the effectiveness of the extra security added by possession factor (FIDO2 keys), we believe it is necessary to evaluate its security level when one factor (e.g., password) is already compromised [also stated in [28] threat model]. The threat model we consider, does not compromise the
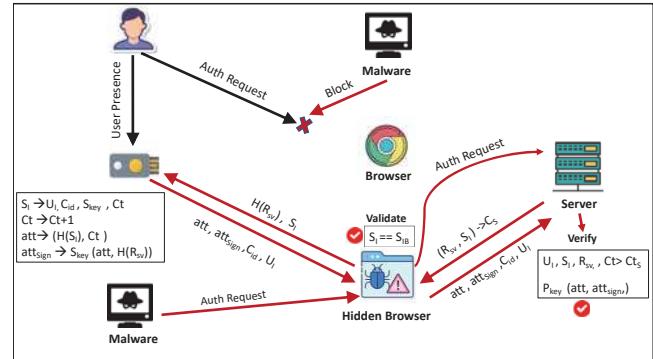

Figure 7: Workflow of Same-service Attack on WebAuthn Protocol

FIDO client (which is a part of the browser), the FIDO authenticator (i.e., security key), service or communication channel between these components. With this defined threat model, we aim to evaluate the common assumption found in the literature [9] that the attacker needs to compromise both factors to effectively undermine 2FA systems, which include FIDO2 keys.

4.3   Attack Overview and Crux

Our demonstration shows that it is sufficient to compromise the user terminal (e.g., desktop/laptop computer) to defeat *CR-2FA*. The attack initiates a concurrent authentication attempt from the terminal while the user tries to authenticate. It blocks the user's authentication attempt, and the service generates a challenge for the attacker's attempt, which will be forwarded to the *CR-2FA* authenticator, causing its LED light to flash. As users intend to authenticate, they unwittingly approve the attacker's session, believing their own authentication request triggers the LED flashing.

Attack Launching Condition. FIDOLA scans keypresses, detecting when the user types the URL of the targeted authentication site (e.g., mail.google.com) in the address bar. It captures the entered username and password, sending a parallel request through a hidden browser session using the collected authentication credentials. Concurrently, the browser extension blocks the users' authentication requests as they submit their credentials. Furthermore, it can redirect them to a page controlled by the attacker. As the attacker's session initiates and runs on a hidden browser session from the background, users cannot access or invalidate attacker's sessions.

Attack Vector in FIDO Specification. FIDOLA is different from all of the attacks considered in FIDO threat analysis [18]. Here, it loads different Relying Party App (RP App) (e.g., Google web app) other than the RP App user is intended to authenticate (e.g., Twitter web app), using a different FIDO client (i.e., hidden browser). FIDOLA does not convince users to use another FIDO client (i.e., a hidden browser); instead, it generates a concurrent attack using a hidden browser while users intend to authenticate using a legitimate FIDO client. **To the best of our knowledge, the FIDO specification has no mention of our attack in any of their threat analysis; thus, the attack is both novel and fundamental.**
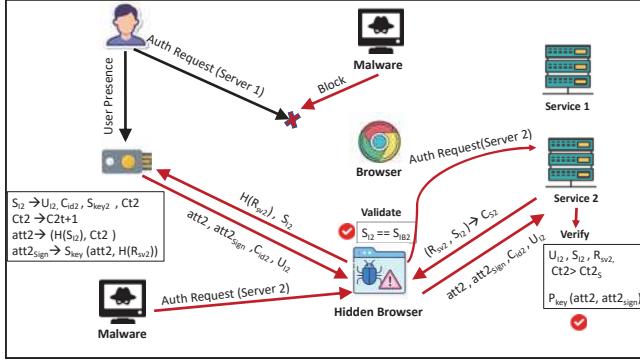
Figure 8: Workflow of Cross-service Attack on WebAuthn Protocol

Figure 9: Workflow of Cross-service attack in the presence of Windows security.

4.4 FIDOLA Attack on WebAuthn Protocol

FIDOLA can perform both same-service and cross-service attacks on WebAuthn.

Same-service Attack:We illustrate the Same-service attack workflow on WebAuthn in Figure 7. The malicious browser extension can block the user's request, initiate a hidden automated browser session and send an authentication request to the service using known authentication credentials. It can then redirect users to an attacker-controlled page, prompting them to tap the authenticator. Upon receiving the attacker's authentication request, the service sends the client a challenge containing the server ID S_I and a random string R_{SV}. The hidden browser session will pass the validity check of S_I with web origin S_{IB}. Subsequently, the client will send a hash of the received random string $H(R_{SV})$ and server identity S_I to the authenticator.

The authenticator would then retrieve user id U_I, credential id C_{id}, S_{key}, and counter ct after the user pressed the button. As users are actively authenticating and expect their authenticator LED to blink, they will eventually press it to approve. The authenticator would then increment the counter ct and send the attestation signature to the client containing the updated counter ct, server identity S_I, U_I, R_{SV}, and C_{id} to the client. The client redirects the received information to the service end where U_I, S_I, R_{SV} will be verified by the service and the attacker's session will be authenticated.

Cross-service Attack: Here, we assume the targeted user registers multiple services from the same authenticator (i.e., Security Key). We illustrate the Cross-service Attack workflow in Figure 8. Like the Same-service attack, the malicious browser extension would block users' authentication requests to the service (Service 1) and redirect them to another similar-looking page prompting them to tap the authenticator. At the same time, the automated browser would request another service (Service 2) for authentication. The service then would send a challenge to the hidden browser containing server ID S_{I2} and a random string R_{SV2}. The client (hidden browser) would validate the server ID S_{I2} with the web origin S_{IB2}, which would pass as the client here requested authentication from service 2.

As the user is trying to authenticate to Service 1 and waiting for the LED to blink on their authenticator; they are likely to approve the request (generated for Service 2) by pressing the button. After
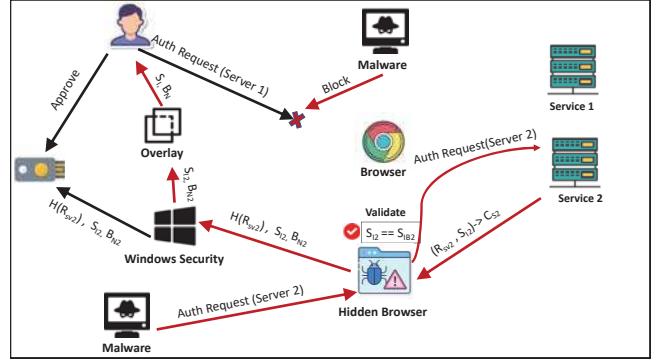
that, the authenticator would retrieve user id U_{I2}, credential id C_{id2}, S_{key2}, and counter $ct2$. After updating the counter $ct2$, the authenticator would create an attestation signature containing the updated counter $ct2$. It would send U_{I2}, and C_{id2} along with the attestation signature to the client, which would redirect it to the service (Service 2). As the attestation signature, U_{I2}, R_{SV2}, C_{id2}, and $ct2$ are valid, the service (Service 2) would approve the malicious authentication.

Attack in the presence of OS level security: For cross-service attack, in the presence of Windows security, the hidden browser would send R_{SV2}, S_{I2}, and browser name B_{N2} to Windows Security API. It would then display the service name with S_{I2} and the browser name B_{N2} in a message to the user. Here, FIDOLA will draw an overlay showing S_I and B_N, which are the user's service and browser names, to convince them to tap the key and inadvertently allow the attacker's session. The attack workflow is shown in Figure 9.

4.5 FIDOLA Attack on CTAP2 Protocol

FIDOLA impersonates the user from the same device (i.e., a computer) and simultaneously (i.e., during the user's authentication attempt). As such, when the hidden browser session sends a binding request, the authenticator will ask for the PIN. As the attack is executed during users' authentication attempts, the user would provide their PIN to complete the authentication process.

Passwordless authentication systems require a PIN for each attempt and may use phone lock for verification. In the same-service attack, blocking the user's request and sending a parallel hidden browser request is straightforward. The PIN prompt would appear only once (for the adversary's request), and the user would provide their PIN, inadvertently approving the adversary's session.

4.6 Attack Implementation

Our proof-of-concept program consists of two components:

Malware on Terminal (MoT): MoT is designed to run in the user space of the operating system. It uses a keylogger to capture the keystrokes and mouse actions. It's primary function is to detect keystrokes and capture usernames and passwords entered by the user. Its controller determines the attack start time and initiates hidden browser sessions. We use Python 3.7 to implement the MoT

controller and keylogger. We use selenium webdriver [37] and chromedriver [12] to implement the hidden browser session. The hidden browser session program executable is a JAR (Java Archive) file that contains the code and resources needed to launch and manage a hidden browser session. MoT also can draw an overlay on any part of the screen to be used in cross-service attacks. The Python executable and the jar file can be run without installation and portable Java.

Redirection Agent (RAgent): We implement a Chrome browser extension RAgent that can block any user request when a URL match is found and redirect them to an attacker-controlled web page during the authentication process. After blocking the authentication request, it immediately redirects the user to a similar-looking attacker-controlled webpage (which instructs them to touch the security key) to persuade them that they are still in the authentication process. RAgent waits for a specified duration (e.g., 15 seconds) to complete the authentication process. Following this, it redirects the user to the legitimate authentication page again, creating the illusion that the login was unsuccessful due to a technical glitch.

4.7 Attack Workflows

Same-service Attack on *CR-2FA*:

(1) **Step 1:** When MoT detects the user's intent to authenticate and the completion of credential input, it captures the password using a keylogger and prepares to launch a hidden browser session using the captured authentication credentials.
(2) **Step 2:** While the user is about to submit authentication credentials, RAgent blocks the user's request and redirects them to a similar-looking attacker-controlled page, which prompts the user to touch the security key to complete authentication.
(3) **Step 3:** In the meantime, the server receives the hidden browser session launched by MoT and sends a challenge to the hidden browser session, which will be passed as the web origin and server ID will be the same.
(4) **Step 4:** As the user is expecting the LED light to blink for their initiated authentication attempt, when the attacker's request arrives, they press the button to approve the attacker's request, thinking that it is generated for their request.

Cross-service Attack on *CR-2FA*:

(1) **Step 1**: Similar to Step 1 of a same-service attack, after detecting the user's intent to authenticate into service A, MoT captures the password and saves it for future use. In the meantime, it sends an authentication attempt to Service B, which is registered using the same hardware security key.
(2) **Step 2**: RAgent will block the authentication request for Service A when the user is about to submit their authentication credentials. The user is redirected to a similar-looking page for Service A that prompts the user to press their security key.
(3) **Step 3**: In the meantime, Service B receives the authentication request sent by the attacker and replies with a challenge to the hidden browser session. This challenge would be passed as the hidden browser session's web origin, which would be the same as Service B's ID.
(4) **Step 4**: When the browser redirects Service B's request to the security key, the Windows security will show the service name

(Service B) and browser name in a security message. However, at that stage, the attacker will draw an overlay with Service A information, effectively hiding the Windows security message showing Service B's information. As the user expects to authenticate and the original Windows security message is hidden by the overlay, the cross-service attack will be successful.

5 User Study

We conducted a user study to assess the detectability of cross-service attacks and whether users can detect it's overlay approach.

5.1 System Design and Implementation

Demographic Information: We recruited 20 participants for the user study, with a gender distribution of 60% male and 40% female. The majority (55%) were aged 31-40, while 40% were aged 21-30, and the remaining participants fell into the 41-50 age range. All participants were university students, with 55% holding a master's degree and 35% holding a bachelor's degree. The majority of participants (95%) were familiar with 2-factor authentication and used it regularly for email (80%) and banking (75%) purposes.

Authentication System: We have developed an authentication system that incorporates webAuthn in both the registration and authentication processes. To emulate a secure site, our system is deployed locally on a laptop using "host mapping" and a self-signed certificate. The web application was developed using PHP and MySQL. Participants are required to register a security key and set a password during the registration process.

Attack Module: We have developed an attack module that can randomly display an overlay containing the participant's intended service name in some of the attempts.

Devices Used: For the user study, we employed Yubikey [46] and Google Titan [22] USB security keys with a Windows 10 laptop.

Authentication Attempts Design: During the core study phase, participants completed 30 authentication attempts, with 30% (9 attempts) designed as malicious. In these malicious attempts, the actual security message (Figure 12b) was concealed by a carefully designed overlay.

Evaluation Metrics: Participants had two response options to the FIDO2 security request: approval or refraining from responding. We used the following notations: $Att_{B-approved}$ for approved benign attempts, $Att_{B-refrained}$ for benign refrained attempts, $Att_{M-approved}$ for approved malicious attempts, and $Att_{M-refrained}$ for malicious refrained attempts.

In this user study, we employed two evaluation metrics to measure both benign and malicious authentication attempts. The first metric, *BAR* (Benign Approval Rate), quantifies the percentage of approved benign attempts by all participants. It reflects participants' familiarity with FIDO2 keys and their task accuracy. The second metric, *AASR* (Adversarial Attempt Success Rate), quantifies the percentage of approved malicious attempts. This metric assesses the adversary's success in using an overlay approach to deceive participants during cross-service attacks. Equations 1 and 2 express *BAR* and *AASR*, respectively.

$$BAR = \frac{\sum_i Att_{B-approved}}{\sum_i Att_{B-approved} + \sum_j Att_{B-refrained}} \quad (1)$$

$$AASR = \frac{\sum_i Att_{M-approved}}{\sum_i Att_{M-approved} + \sum_j Att_{M-refrained}} \quad (2)$$

For an ideal authentication system, *BAR* should be 100% and *AASR* should be 0%.

5.2 Study Protocol

The user study was approved by our university's Institutional Review Board (IRB), and we adhered to standard IRB guidelines for study and data collection. Participants were not required to use their personal devices and instead used a laptop and a FIDO2 security key provided by facilitator. In the registration process, participants are instructed not to use their personal email addresses or passwords. Instead, they are advised to use dummy or non-existent email addresses to comply with IRB guidelines. The study comprises four phases.

(1) **Introduction**: In this phase, the facilitator explained FIDO2 keys functionality and their security features. Participants were informed about the experiment's purpose, including the purpose of security keys (e.g., preventing phishing) and OS-level security (e.g., preventing cross-service concurrent attacks), and instructed to check security messages (service/browser name) and not approve suspicious attempts. Before the data collection phase following the explanation, the facilitator administered a pre-test questionnaire to gather demographic information and assess the participant's familiarity with 2FA systems and FIDO2 keys. Subsequently, the facilitator registered a participant account, using a dummy email and password, and registered the FIDO2 Key.

(2) **Practice Phase**: During this phase, participants became familiar with the FIDO2 authentication system and interacted with the Windows security message. We designed our practice phase, requiring participants to complete the authentication system using provided credentials with FIDO2 keys 10 times. This practice phase is expected to establish sufficient familiarity among participants with both the FIDO2 authentication process and Windows OS-level security.

(3) **Study Phase**: In this phase, participants conducted 30 authentication attempts. Among them, 21 were benign, while 9 displayed a modified message through a malicious overlay. Benign and malicious attempts are presented to the user in a random order. The participants were instructed to authenticate using the provided credentials for each try with a gap of 5 seconds. Participant responses were recorded and stored.

(4) **Post-test Questionnaire**: After study completion, participants were debriefed about the attack and received a post-test questionnaire to provide feedback on their study experience. Here, participants provide answers regarding their observation of any suspicious behavior and the reasons for their unsuccessful attempts. They are also asked to offer feedback on their attentiveness to crucial details (e.g., URL) during the authentication process.

Table 3: User study outcome.

Attempts	Attempt Sent	Attempt Approved	BAR	AASR
Benign	420	311	97.85%	-
Malicious	180	172	-	95.55%

Table 4: Evaluation with off-the-shelf desktop-based antivirus software

Antivirus Name	Quick Scan	Full Scan	Runtime Warning
Windows Defender	✗	✗	✗
Avast	✗	✗	✗
MalwareBytes	✗	✗	N/A
Kaspersky Security Cloud	✗	✗	✗
Sophos Home	✗	✗	✗
Avira	✗	✗	✗
AVG	✗	✗	✓[1]
Mcafee Total Protection (Free Trial)	✗	✗	✗

✓- Detected, ✗- Not Detected

Table 5: Resource Consumption of the Concurrent Login Attack

Metric	Consumption (idle)	Consumption(peak)
CPU	0.01%	20.0%
Memory	27.5 MB	108.83 MB
Power	Very Low	Low
Network	0.0 MB	0.2 MB

6 Further Evaluation and Insights

6.1 User Study Result Analysis

Effectiveness of Cross-service Attack: During the study, we carried out 420 benign authentication attempts, of which 411 were correctly approved. We calculated the *Benign Approval Rate (BAR)* as 97.85% using Equation 1, indicating users' complete understanding of the study requirements. We calculated the *Adversarial Attempt Success Rate (AASR)* a 95.55%, indicating a very high rate of approval for authentication attempts with overlays. This indicates the effectiveness of adversarial cross-service attempts using overlays. Detailed results can be found in Table 3.

Detection of Suspicious Activities: Based on the post-test questionnaire, our observations reveal that 70% of participants reported no detection of suspicious activities, and 75% did not abstain from any authentication attempts. 15% of the participants suspected malicious activity on the user terminal and 20% reported that, they encountered messages at unexpected times. These findings highlight a notably low detection rate, persisting even after multiple malicious attempts. The results also suggest that the limited display cannot provide sufficient information to detect the attack, even when participants are threat-aware. Consequently, the limited display tends to increase trust in the attack when the Windows security message is hidden by an overlay.

Participant's Reasoning About the Attack. According to Q2, among the participants who were suspicious, 37.5% believed it was a malicious attack on the user terminal, while others were uncertain. After debriefing, participants were surprised to learn about the attack, as they assumed OS-level security would correctly

---

[1] After showing warning and performing initial scan, the program is allowed to run.

display service and browser names, not expecting that an overlay could compromise this trusted security measure.

Inattentiveness to Changed URL: 55% of participants either never or rarely (less than 40% of the time) checked the intermediate URL where the "Touch your security key" message appears. Given that FIDOLA blocks user attempts and redirects them to a similar-looking attacker-controlled page at this stage, paying attention to the intermediate URL is crucial for detecting such attacks. Adversaries can exploit this lack of user attentiveness to design and execute attacks similar to FIDOLA.

Detailed user responses in post-test questionnaire can be found in Appendix Figure 10.

6.2 Detectability

Detectability From User Terminal: The *MoT*, *RAgent* are designed to run from background to evade user and authentication service detection. *RAgent* intercepts and redirects the user's request to a visually similar attacker-controlled page without triggering browser warnings. While it is more common to check the URL in the authentication page, the users may not pay much attention to the intermediate URL changes. This has been reflected in our conducted user study, where 55% of the participants seldom checked the intermediate page URL (See Section 6.1), which suggests significant risks of being victim of such attack.

After redirection, *RAgent* automatically redirects the user to the legitimate login page after a brief delay, simulating a minor technical glitch which reduces the likelihood of user detection. Even if users grow suspicious at this stage, they would be unable to prevent potential harm to their account.

In cross-service attacks, FIDOLA displays an overlay with the user's intended service name to conceal the attacker's requested service. Our user study revealed the low detectability of this technique, demonstrating its effectiveness in cross-service attacks.

Anti-malware Programs: Anti-malware programs use two methods to detect malware: signature-based detection, which compares it with known patterns, and behavior-based detection, which monitors runtime activity for suspicious behavior like resource consumption and access attempts to sensitive areas. We applied code obfuscation techniques to hide the malicious payload of our designed attack program, and tested its detection rate against different anti-malware programs.

We test our proof-of-concept attack against eight desktop-based antivirus programs, Windows Defender [35], Avast [3], MalwareBytes [32], Kaspersky Security Cloud [2], Sophos Home [38], Avira [5], AVG [4] and Mcafee Total Protection (Free Trial) [33]. We show the detailed result in Table 4.

For a thorough assessment, we performed full scans as well as runtime scans, encompassing both signature-based and behavior-based detection. In our experiment, most antivirus programs, except AVG, failed to detect the attack executable during runtime. AVG did display a warning for the suspicious file but couldn't match it with existing signatures during the initial scan, allowing the file to run.

Resource Consumption: After completing the attack, the malicious program is designed to stop, avoiding prolonged resource consumption. During attack, we monitor CPU, memory, power, and

network usage and present average values in Table 5. In its "idle" state, the program has minimal resource demands. In an attack lasting less than a minute, the program briefly enters an "active" state, resulting in a temporary increase in resource usage before promptly returning to an "Idle" state after the attack ends. Resource consumption during the peak is comparable with other benign applications. Low CPU usage stems from keylogger being lightweight, and hidden browser session (Chromedriver) using only a single tab. Additionally, the extension (RAgent) is integrated with user's primary browser, which does not increase resource consumption.

Real-world *CR-2FA* Susceptibility: We assessed our attack method against the 2FA systems of four prominent services: Microsoft Outlook [1], Facebook [15], Google [21], and Twitter [44]. These services encompass email, social networking, and e-commerce applications. Our demonstration revealed that our attack can successfully bypass nearly all *CR-2FA* schemes employed by these services.

Deployability: Our attack program is highly deployable with minimal installation dependencies. We created the hidden browser session using phantomJS and Selenium Webdriver, both requiring minimal installations. The keyloggers and other components are packaged into an executable file for easy invocation via a malicious batch script. RAgent, another attack component, is developed as a standalone Chrome extension. While user consent is needed for extension installation, malicious code can be concealed within a benign extension, potentially evading detection.

7 Discussion & Future Work

7.1 Comparison with Other Attacks

Active Phishing Attack. In an active phishing attack, the attacker captures a user's credentials through an attacker-controlled web page using social engineering techniques. Following this, the attack script promptly initiates authentication requests and triggers user presence verification on the possession-factor device. However, in the case of *CR-2FA*, it utilizes the WebAuthn protocol, ensuring the authenticity of the requesting service by verifying the server identification with the web origin in the browser. In the event of an active phishing attack, where the server identification and web origin for the attacker-controlled web page do not match, the browser will halt the authentication process, and thus can effectively prevent the active phishing attack.

In contrast, FIDOLA collects user credentials using a keylogger, blocks users' requests, and sends authentication requests in the background using a hidden browser session to a legitimate authentication service. As a result, for FIDOLA the server identification and web origin would be the same, which helps to pass the web origin check of WebAuthn, and thus compromise the *CR-2FA* system. Additionally, FIDOLA is capable of performing cross-service attacks (discussed in Section 4.4), which is not possible in an active phishing attack.

Session Hijacking Attacks: The session hijacking attack aims to steal session cookies from an established user session and reuse them in the attacker's session. It operates as a relay attack, relaying captured session cookies, but it is not a valid attack on the 2FA system. Furthermore, as discussed in Section 4, a relay attack like

social hijacking can manipulate user requests to impersonate them, which is not a valid attack on a cryptographic protocol. In contrast, FIDOLA is a "non-relay attack" that operates independently of the user's session and does not relay their cookies or requests. Importantly, unlike session hijacking, FIDOLA enables cross-service attacks, giving it a clear advantage.

7.2 Potential Mitigation

Implementing Secure Display: The USB version of *CR-2FA* lacks a display, relying solely on LED buttons for communication. While this design is portable and cost-effective, it has limitations in conveying important authentication details. The observation from our research shows that malicious programs can easily spoof Windows' security pop-up displaying browser and service information. Therefore, we recommend *CR-2FA* device manufacturers to consider adding a small, low-cost display to enhance authentication information visibility and reduce the risk of cross-service attacks.

This proposed display should show the service name and the browser name which need to be verified by the user before authorizing the transaction. However, this potential mitigation strategy requires the users to pay close attention to the service name and to be aware of the possible risks of cross-service attacks. Here, as users need to physically insert the USB key into their device and press a button to initiate the login process, they are prompted to look at the USB FIDO2 keys and press the button to approve. In this scenario, users should be vigilant in identifying any unusual or unintended service names displayed during the process.

The proposed secure display should be integrated into secure hardware [34, 47] to ensure a trusted path between the FIDO client and the display. As outlined in the FIDO specification [18], our recommendation aligns more closely with [SM-10] (implemented by FIDO authenticators) than [SM-5] (implemented by FIDO clients). Deploying a secure display on user terminal devices (e.g., laptops, smartphones) carries potential security risks, as adversaries can exploit the permission levels of the device's operating system to introduce overlays displaying manipulated information. On the other hand, authenticator devices are equipped with secure hardware, providing a more secure avenue to thwart cross-service attacks.

Restriction of Overlays in Operating System: From our observation, it is evident that overlay messages can disrupt the visibility and functionality of Windows security messages, intended to alert users about critical security issues. However, overlays can also serve as a means for other programs to convey urgent or important information to users. Consequently, preventing the appearance of overlay messages completely may not be a practical solution, as it could reduce the user experience and functionality of other applications. Therefore, we recommend operating system designers to explore viable solutions aimed at addressing both the security and usability aspects of the problem, ensuring the visibility and functionality of Windows security messages without completely blocking overlays from trusted applications.

However, these mitigation techniques may only work to prevent cross-service attacks but will not be able to detect same service attacks. As the attack is fundamental, designing effective defense strategies is challenging. While we suggest some possible mitigation approaches, developing a comprehensive defense strategy is beyond the scope of this paper. We believe this topic warrants further exploration in future research.

Malicious site blockers, such as Google Safe Search, can be a potential mitigation technique. However, while they can block known malicious sites, they are ineffective against FIDOLA if attackers use new, unlisted domains.

7.3 Limitations and Future Work

Limitations. The primary limitation of the proof-of-concept attack design is that it only works on computers with Windows OS and Chrome browsers. Therefore, there is an opportunity for researchers to design more general attacks that can work on a wider range of platforms, including popular OS/browser combinations and smartphone platforms.

In the user study, participants were required to complete 30 authentication attempts during the data collection phase. Consequently, there is a possibility of habituation bias among the participants. To mitigate this, we instructed participants to carefully examine security messages in each attempt. Participants' attentiveness is evident from the post-test survey, where 20% reported noticing suspicious activities.

Although the sample size of our study is 20 users, all participants encountered a total of 600 authentication attempts (30% of which were malicious). We believe the number of participants and authentication attempts provide a sufficient and necessary foundational step to demonstrate the feasibility of our attack. Given that our participants were tech-savvy, security-aware students who still struggled to detect the attack, we believe the susceptibility of the general population might not be lower.

Additionally, the authentication system and device used in the study differ from their day-to-day systems and devices, which could impact their decision-making process.

Future Works. The disadvantage of limited display in the *CR-2FA* devices and the opportunity of the cross-service attack open an ample opportunity for the researchers to re-evaluate the security offered by *CR-2FA* devices. They can conduct research on effective, secure, and informed communication (e.g., secure display) during authentication with *CR-2FA* devices, as well as design more secure enhancements of FIDO, WebAuthn, and CTAP2 protocols.

FIDOLA is currently designed for Windows OS. Designing a similar attack in Linux-based OS, such as Ubuntu or Mac OS, can be challenging due to their strong permission models. However, researchers have an opportunity to explore potential loopholes in the permission model to obtain the necessary administrator privileges for implementing such attacks.

7.4 Other Discussion

Risk of FIDOLA Attack on Passkeys: Passkeys are a type of FIDO credentials stored on the device, such as a computer or smartphone, instead of FIDO2 keys. Introduced as an alternative to passwords, passkeys aim to provide a more secure and convenient authentication method for various service providers. However, our proposed attack has the potential to compromise passkeys, including cross-service attacks where malicious overlays can conceal the actual service information from the user.

Motivation to use Windows in proof-of-concept attack. We focus on Windows while developing our proof-of-concept attack, as it holds a significant share of the global desktop OS market (72.17% as of February 2024) [42]. This makes it a substantial user base for FIDO2. Nevertheless, FIDOLA is a fundamental vulnerability that can also affect other operating systems, based on the same general principles.

8 Conclusion

Effective 2FA should safeguard against web and terminal attacks, verifying the user's presence on a separate device during authentication. Previous research confirms 2FA's resilience, even against malicious terminals. However, our non-root malware based analysis of FIDO2 key deployments uncovers vulnerabilities to our concurrent login attacks from infected terminals, with low detectability by desktop anti-malware. Importantly, our attack differs from session hijacking, enabling more devastating cross-service attacks. We suggest security enhancements for service providers and users, mindful of usability trade-offs, to strengthen account security.

Acknowledgments

References

[1] 2021. Outlook- Free personal email and calender from Microsoft. https://outlook.live.com/owa/.
[2] AO Kaspersky Lab. 2021. Kaspersky Security Cloud - Free. https://www.kaspersky.com/free-cloud-antivirus.
[3] Avast Foundation. 2021. Avast Free Antivirus. https://www.avast.com/en-us.
[4] Avast Software s.r.o. 2021. AVG Free Antivirus. https://www.avg.com/en-us/.
[5] Avira Operations GmbH & Co. KG. 2021. Avira Antivirus. https://www.avira.com/.
[6] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, and Bogdan Warinschi. 2021. Provable security analysis of FIDO2. In *Annual International Cryptology Conference*. Springer, 125–156.
[7] Mihir Bellare and Phillip Rogaway. 1993. Entity authentication and key distribution. In *Annual international cryptology conference*. Springer, 232–249.
[8] BleepingComputer LLC. 2022. Malicious browser extensions targeted almost 7 million people. https://www.bleepingcomputer.com/news/security/malicious-browser-extensions-targeted-almost-7-million-people/.
[9] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. 2012. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy*. IEEE, 553–567.
[10] Thanh Bui, Siddharth Prakash Rao, Markku Antikainen, Viswanathan Manihatty Bojan, and Tuomas Aura. 2018. Man-in-the-machine: exploiting ill-secured communication inside the computer. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1511–1525.
[11] Checkpoint Software Technologies Limited. 2022. May 2022's Most Wanted Malware: Snake Keylogger Returns to the Top Ten after a long absense. https://www.checkpoint.com/press-releases/may-2022s-most-wanted-malware-snake-keylogger-returns-to-the-top-ten-after-a-long-absence/.
[12] Chromium Project. 2021. ChromeDriver- WebDriver For Chrome. https://chromedriver.chromium.org.
[13] Cisco. 2022. Guide to Web Authentication. https://webauthn.guide/.
[14] Comodo Group Inc. 2020. What is Zeus Trojan ? | How the Zeus Virus infects the computers? https://enterprise.comodo.com/blog/what-is-zeus-trojan/.
[15] Facebook. 2021. Facebook. https://www.facebook.com/.
[16] Florian M Farke, Lennart Lorenz, Theodor Schnitzler, Philipp Markert, and Markus Dürmuth. 2020. {"You} still use the password after {all"}–Exploring {FIDO2} Security Keys in a Small Company. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. 19–35.
[17] FIDO Alliance . 2022. FIDO2: WebAuthn & CTAP. https://fidoalliance.org/fido2/.
[18] FIDO Alliance. 2021. FIDO Security Reference (FIDO Alliance Review Draft 25 May 2021). https://fidoalliance.org/specs/common-specs/fido-security-ref-v2.1-rd-20210525.html.

[19] FIDO Alliance. 2022. Client-to-Authenticator protocol (CTAP2). https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html.
[20] FIDO Alliance. 2022. FIDO Alliance- Open Authentication Standards. https://fidoalliance.org/.
[21] Google. 2021. Google Accounts. https://accounts.google.com.
[22] Google Cloud. 2021. Titan Security Key. https://cloud.google.com/titan-security-key.
[23] Imparva. 2021. Headless Chrome: DevOps Love It, So Do Hackers, Here's Why. https://www.imperva.com/blog/headless-chrome-devops-love-it-so-do-hackers-heres-why/.
[24] Imperva. 2018. Headless Chrome: DevOps Love It, So Do Hackers, Here's Why. https://www.imperva.com/blog/headless-chrome-devops-love-it-so-do-hackers-heres-why/.
[25] InfoSecurity Magazine. 2018. Attackers keen on Automated Browsers. https://www.infosecurity-magazine.com/news/attackers-keen-on-automated/.
[26] Charlie Jacomme and Steve Kremer. 2021. An extensive formal analysis of multi-factor authentication protocols. *ACM Transactions on Privacy and Security (TOPS)* 24, 2 (2021), 1–34.
[27] Mohammed Jubur, Prakash Shrestha, Nitesh Saxena, and Jay Prakash. 2021. Bypassing push-based second factor and passwordless authentication with human-indistinguishable notifications. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 447–461.
[28] Dhruv Kuchhal, Muhammad Saad, Adam Oest, and Frank Li. 2023. Evaluating the Security Posture of Real-World FIDO2 Deployments. (2023).
[29] Juan Lang, Alexei Czeskis, Dirk Balfanz, Marius Schilder, and Sampath Srinivas. 2016. Security keys: Practical cryptographic second factors for the modern web. In *International Conference on Financial Cryptography and Data Security*. Springer, 422–440.
[30] Leona Lassak, Annika Hildebrandt, Maximilian Golla, and Blase Ur. 2021. " It's Stored, Hopefully, on an Encrypted Server": Mitigating Users' Misconceptions About {FIDO2} Biometric {WebAuthn}. In *30th USENIX Security Symposium (USENIX Security 21)*. 91–108.
[31] Sanam Ghorbani Lyastani, Michael Schilling, Michaela Neumayr, Michael Backes, and Sven Bugiel. 2020. Is FIDO2 the Kingslayer of User Authentication? A Comparative Usability Study of FIDO2 Passwordless Authentication.. In *IEEE Symposium on Security and Privacy*. 268–285.
[32] MalwareBytes. 2021. MalwareBytes Cybersecurity for Home and Business. https://www.malwarebytes.com/.
[33] McAfee. 2021. McAfee Total Protection. https://www.mcafee.com/en-us/antivirus/free.html.
[34] Jonathan M McCune. 2009. Safe passage for passwords and other sensitive data. In *Proceedings of the Network and Distributed System Security Symposium, 2009.*
[35] Microsoft. 2021. Windows 10 Security, Windows Defender Antivirus, Windows Defender Security Centre. https://www.microsoft.com/en-us/windows/comprehensive-security.
[36] NIST. 2022. Authenticator Assurance Level (AAL). https://csrc.nist.gov/glossary/term/authenticator_assurance_level.
[37] Software Freedom Conservancy- Selenium Project. 2021. Selenium WebDriver. https://www.selenium.dev/projects/.
[38] Sophos Ltd. 2021. Sophos Home - Cybersecurity made simple. https://home.sophos.com/en-us.aspx.
[39] StatCounter. 2021. Browser Market Share Worldwide. https://gs.statcounter.com/browser-market-share.
[40] StatCounter. 2023. Desktop Operating System Market Share Worldwide- October 2023. https://gs.statcounter.com/os-market-share/desktop/worldwide.
[41] StatCounter. 2023. Operating System Market Share Worldwide- September 2022. https://gs.statcounter.com/os-market-share.
[42] Statista. 2024. Global market share held by operating systems for desktop PCs, from January 2013 to February 2024. https://www.statista.com/statistics/218089/global-market-share-of-windows-7/.
[43] ThreatPost. 2021. 500 Malicious Chrome Extensions Impact Millions of Users. https://threatpost.com/500-malicious-chrome-extensions-millions/152918/.
[44] Twitter, Inc. 2021. Explore Twitter. https://twitter.com/explore.
[45] Peng Xu, Ruijie Sun, Wei Wang, Tianyang Chen, Yubo Zheng, and Hai Jin. 2021. SDD: A trusted display of FIDO2 transaction confirmation without trusted execution environment. *Future Generation Computer Systems* 125 (2021), 32–40.
[46] Yubico. 2021. Security Key by Yubico. https://www.yubico.com/product/security-key-by-yubico.
[47] Yongxian Zhang, Xinluo Wang, Ziming Zhao, and Hui Li. 2018. Secure display for FIDO transaction confirmation. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*. 155–157.

A   Appendix

## A.1   User Study Follow Up Questions and Other Snapshots
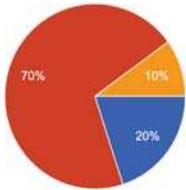
(a) Q1: Did you notice any suspicious behavior of web application during the study?
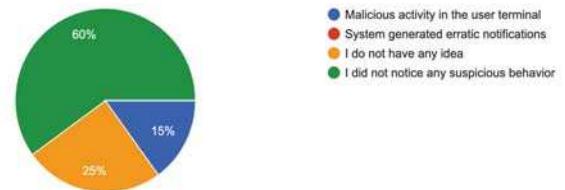
(b) Q2: What is the possible reason behind this suspicious behavior?
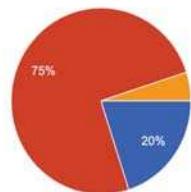
(c) Q3: Did you refrain from approving any notification from the security key?
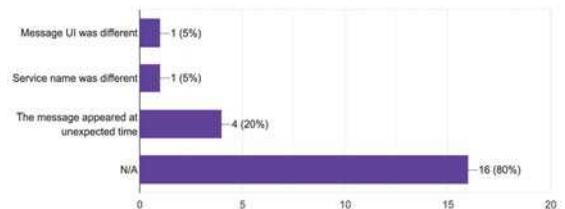
(d) Q4: If you refrain from approving any request, please list all the possible reasons. If you approved all the request please select "N/A"
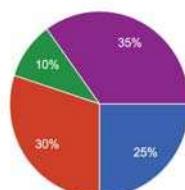
(e) Q5: How often you have checked the URL of the intermediate page (the page where "Touch your security key" message appeared)?

Figure 10: Follow-up survey findings.

(a) Google Titan *CR-2FA* USB and Bluetooth device

(b) Yubico *CR-2FA* device, USB variant

Figure 11: Snapshots of *CR-2FA* devices that we have used in our study
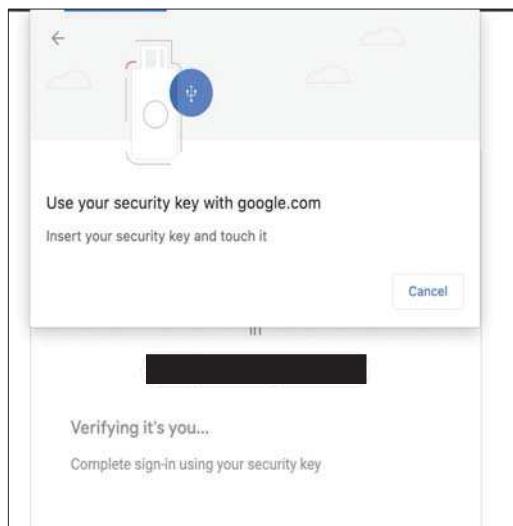
(a) MacOS Terminal UI
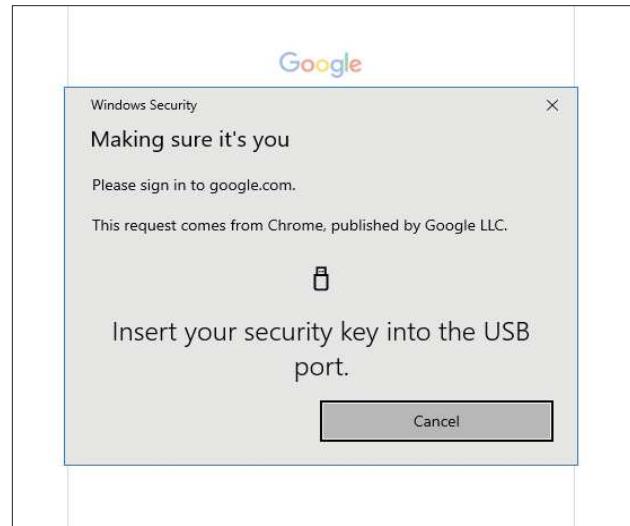
(b) Windows 10 Terminal UI (Windows Security pop-up has been shown which shows some useful information and adds extra level of security)

Figure 12: The interface of different OS terminal during USB *CR-2FA* log in
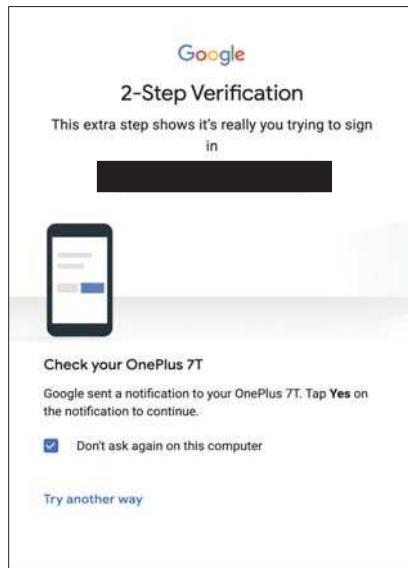
(a) Built-In *CR-2FA* Terminal UI


(b) Built-In *CR-2FA* device UI

Figure 13: Built-In *CR-2FA* UI offered by Android devices and Google service (some parts redacted for paper anonymity)