Columbia University Interim Report

SPONSOR:	**PROJECT TITLE:**	
MASTERCARD IMPACT FUND 615 WEST 131 STREET, 4TH FL NEW YORK, NY 10027	Universal Digital Assertion of Identity to Enhance Prosperity, Security, and Privacy	
	SPONSOR AWARD #:	MCIF CU21-2197
	CU PROJECT #:	PG010589
	PROJECT PERIOD:	07/01/21 - 06/30/24
PRINCIPAL INVESTIGATOR: Schulzrinne,Henning G.	**REPORT PERIOD:**	07/01/21 - 12/31/21
	REPORT TYPE:	INTERIM
AWARD AMOUNT: $350,000.00	**REPORT DATE:**	02/07/2022

FIDO2 Proof of Concept Report

George Litvinov

Abstract

One of the flaws that prevails in the FIDO2 protocol is it's need for "hand-off" points, or communication exchange points, where the many parts of the authentication process must interact with each other. These exchange points allow each component to approach the resulting authentication in a "Black Box" fashion – they simply read in data on one end, do their operations on it if there are any, and write it out the other – not concerning themselves with where the data came from. This verification process is left to the end points of the communication which are the Relying Party (RP) and the Roaming Authenticator. In this experiment, I have identified a vulnerable channel between the RP and the Roaming Authenticator which can be interrupted, hooked, and written to, returning spoofed credentials to the RP during the registration process without ever interacting with the Operating System and the Roaming Authenticator. This allows for the hijacking of the registration process of the FIDO2 protocol, and stuffing of fake credentials, resulting in the RP accepting the stuffed credentials as the user's.

Background

The FIDO2 protocol is a basis for passwordless authentication that enables users to leverage common devices to easily authenticate to online services. The FIDO2 specifications are made up of two parts: the World Wide Web Consortium's (W3C) Web Authentication (WebAuthn)[1] specification and FIDO Alliance's corresponding Client-to-Authenticator Protocol (CTAP)[2]. The reason for the split of the two protocols is due to the need of fulfilling the communication from the web service, to a hardware token and back. The WebAuthn protocol is responsible for communicating from the web service to the browser process, where the communication is handed over to the CTAP protocol, which connects the browser process to the hardware security token through the Operating System.

Usually the communication is made up of two parties:

1. The Relying Party – usually a website or web service requesting authentication
2. The Roaming Authenticator – a hardware security token that wants to authenticate

Due to the specificities of the FIDO2 protocol, there are four "hand-off" points, or points where one of FIDO2's parts must interact with another both during the registration and authentication phase. These points are made up of the interaction between the Relying Party and the browser's WebAuthn client, then between the WebAuthn Client and the Platform WebAuthn API, where the browser process hands off the communication to the Operating System via CTAP2,

[1] https://www.w3.org/TR/webauthn-1/
[2] https://fidoalliance.org/specifications/download/

which then communicates with the Roaming Authenticator also via CTAP2. All of these exchanges are displayed in Fig. 1 as arrows. The FIDO2 protocol was designed in such a way, where each entity within this exchange treats it's environment as a "Black Box": it receives input on one end of the pipeline, performs it's actions on the data, and outputs it through the other end of the pipeline. There is no verification as to where the data (credentials) came from, other than checking that the RP is what it says it is within the client-side javaScript.[3] This means that there is no verification as to whether the RP is interacting with the intended Roaming Authenticator.

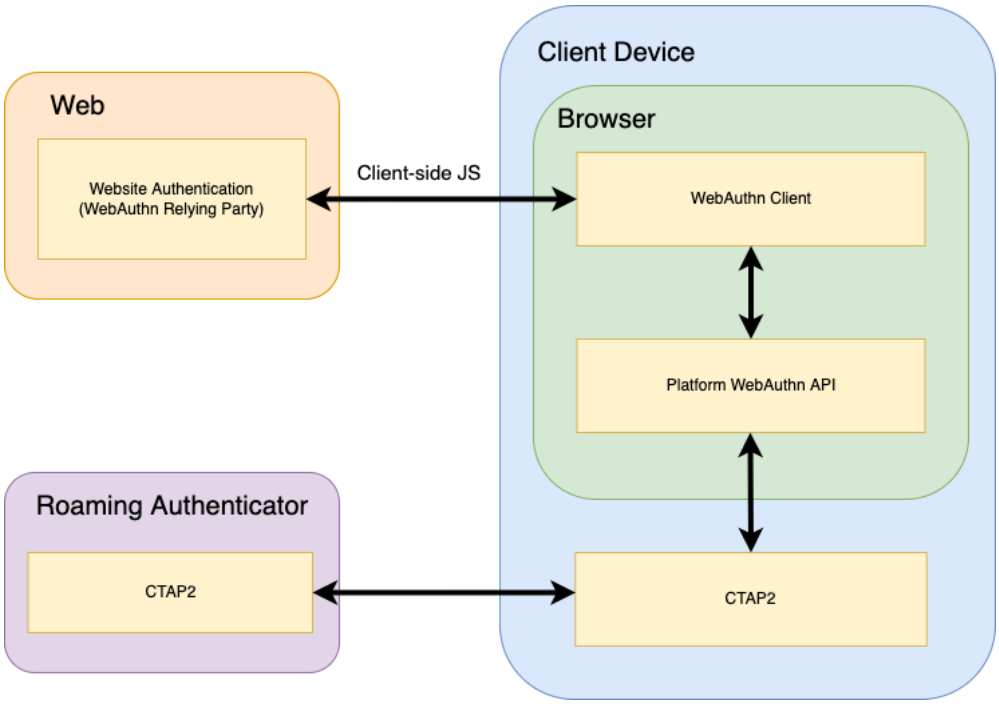

Fig. 1 FIDO2 Protocol in an Overview

The FIDO2 Security Reference lists Security Goads (SG), Security Measures (SM), Threats (T) and Security Assumptions (SA)[4] that have been identified during the development of the protocol. Reading through it, Security Assumption 4 (SA-4) stood out as an interesting detail and formed the basis of this project:

[SA-4]
The computing environment on the FIDO user device and the applications involved in a FIDO operation act as trustworthy agents of the user.

The security assumption assumes that any and all parties that are part of the FIDO2 authentication process are non-malicious, thus any web browser bug, such as the recent

---

[3] https://developers.yubico.com/WebAuthn/WebAuthn_Developer_Guide/WebAuthn_Client_Registration.html

[4] https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-security-ref-v2.0-id-20180227.html

Chrome browser Remote Code Execution bug (CVE-2020-16040[5]), puts the security out of scope for the protocol. In an extremely interconnected world, bugs are bound to happen in every piece of software that we as humans interact with, including the ones that interact during the FIDO2 authentication procedure.

In this experiment I aimed at exploring whether a potential Remote Code Execution bug within the Chrome browser could hijack the FIDO2 registration procedure, generate credentials, and send them back to the RP successfully. (Fig 2.)

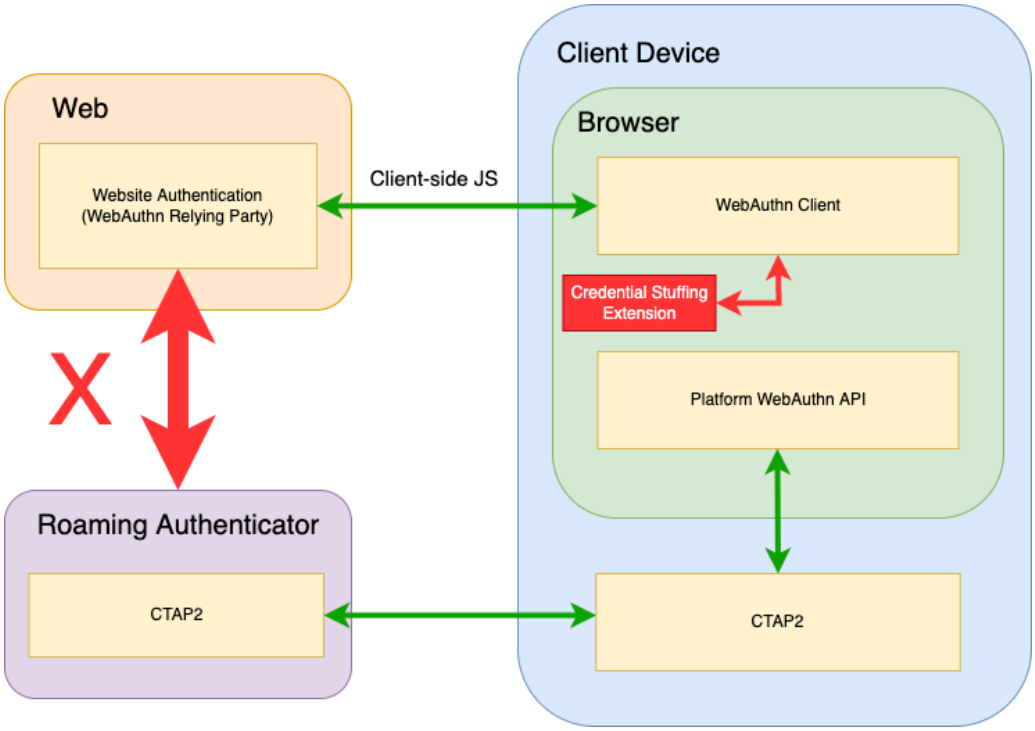

Fig 2. FIDO2 Registration Hijacking Overview

Methodology

I chose to implement a web browser extension that can be loaded into Chrome, since that allows loading custom javaScript to be run by the browser, imitating what an RCE bug may be able to do. There are multiple considerations that had to be done for the correct implementation of this attachment. Google Chrome leverages a multi-process architecture for security, usability and performance. This means that the main browser process, extensions and each tab have their own processes. This architecture also allows for the permissions aspect to be handed down to the OS level, thus it is the OS that allows for one process to communicate to another or not. The main browser process acts as an admin for the rest of the Chrome-related processes, being able to communicate with all of them and execute arbitrary commands. Therefore, an RCE that is able to overtake the browser process, is able to execute any arbitrary code in any other Chrome-related process. In order to simulate a browser process overtake, I have chosen

[5] https://nvd.nist.gov/vuln/detail/CVE-2020-16040

to use the Debug mode in Chrome, which allows for the Chrome Dev Tools to be attached to any tab, and furthermore execute javaScript code.

The Chrome browser has built-in functionality to intercept the calls from the RP, namely during the step where the tab passes the data off to the browser process. During the registration process, this data is utilized by the extension, which then generates a new key-pair and sends back a correctly formatted response with the newly generated key. The data never continues past the tab process, and thus never reaches the actual roaming authenticator.

Usually, a user is able to select the authenticator that they want to authenticate with by displaying a pop-up window. This step is done within the browser process. Since the action never reaches the browser process, the user is never prompted, and the registration of the stuffed credentials is autonomously accepted. Furthermore, a following log-in attempt results in a success, as the same kind of hijacking and stuffing occurs, but already with known credentials to the stuffer.

Results and Conclusions

The resulting browser extension displays the lack of security verification within the FIDO2 protocol and that a Registration Hijacking attack vector is possible. Below are links to two videos displaying a regular Registration and Login procedure, followed by the Hijacked Registration-Login Procedure. Please note that the printouts on the screen for the latter are only done for demonstration purposes.

Regular Registration Procedure: https://bit.ly/3L6dkyB
Hijacked Registration Procedure: https://bit.ly/34lczky

These results pose questions that need to be further explored:

1. How to distinguish between stuffer (bot) activity and actual human activity from relying party's perspective?
2. How to create a registration scheme in such a way, as to predict and identify the source of the registration information in a privacy preserving way?
3. How do we make the registration secure in an insecure environment?