EEP595: Eliminate Password Custodians with USB keys

Problem

The recommendations from OWASP [2] and experts for the industry is to stop using passwords and move to options such as passkeys. Among the available choices, of most interest are security keys (FIDO roaming authenticator) because of their phishing resistance. Between the products on the market, the Yubikey dominates the enterprise space and federal agencies. So it commands a higher price tag. This is challenging to a non-profit organization. So in order to increase adoption, we explore the open source project OpenSK to learn the process to make the key and show a DIY key can be achieved.

Previous work

To avoid passwords in authentication we actually began by trying OAuth2. It is widely supported across different deployment scenarios. The way it treats passwords is by forwarding the user to a identity provider. The ID provider authenticates the user then redirects back (fig. 20). In this way, the password/secret never gets exposed/revealed. Since the protocol involves the external service, it can be difficult to coordinate steps and lead to errors. Which is where integrators have built business products (Auth0/Okta, Cloudflare Access) to fill the niche; not without risk [20]. So choosing OAuth2 can mean doing the work to support each ID provider (Google, Apple, GitHub, Microsoft, etc.) or pay for Okta subscriptions.

Approach

To program the security key, we want to use OpenSK. The project started in 2020 as a collaboration between Google, ETH Zürich, and Nordic Semiconductor to encourage "researchers, security key manufacturers, and enthusiasts to help develop innovative features and accelerate security key adoption." [22] The open source license is Apache 2.0. The firmware utilizes the sandboxing architecture of the Tock embedded OS. Tock is written in Rust to restrict the kernel interfaces at compile time. The capsules that make up the core of the kernel and drivers adhere to rules such as static memory allocation, and cooperative scheduling. In contrast, applications are untrusted and isolated by hardware. Applications don't have to be written in Rust, are subject to preemptive scheduling, and can use dynamic loading.

Experiments

We start with the nrf52840 dongle [24]. The board is listed as supported hardware for OpenSK. Also the hope is that with its DFU mode, it can be programmed directly through USB without the need for extra adapters. In following the install document, it is noted that Python 3.10 is the latest that is compatible so we restrict our environment to 3.9 initially. The required dependencies (rustup, pip, libssl, uuidgen, pkg-config) install normally. It is not necessary to install tockloader tool directly, as I learned the `setup.sh` script will complain about newer versions. The other hiccup in running `setup.sh` appears in its bottom lines for `cargo +stable` which hint that the `+stable` override conflicts with rustup. Finally, the real issue arises when performing the stated [flash command](https://github.com/google/OpenSK/blob/2.1/docs/boards/nrf52840_dongle.md#flashing-using-dfu-preferred-method) as it results in the error about the nrfutil tool. This is when we learn that there are two versions of the nrfutil tool [23], pre 2022 and the newer version. More relevant is that the newer version is x86_64, whereas my development environment is a Raspberry Pi4 (aarch64). Now it seems obvious, but it wouldn't work. Then I tried adafruit-nrfutil which was a fork of the pre-2022, and that version worked up to a point; generates a merged HEX file but still fails flash step. It became necessary to resort to installing Mint Linux on an older spare x86 PC and running the newer nrfutil [25]. With the firmware flashed onto the nrf52840, it was simple to verify that it works by navigating to WebAuthn.io and exercise both register and authenticate flows. For me this was a great relief to see for myself a full working end-to-end.

Knowing it works for the $10 nrf52840 was a boost of encouragement which made me want to attempt to squeeze the cost savings as far as possible. The Raspberry Pi Pico typically retails at $5. Talk about making a convincing argument to the team for adoption of security keys at that price! Unfortunately, the OpenSK project does not target the Pico board. However, the Tock embedded OS includes instructions for flashing the kernel to the Pico. I needed to try. Following the "hello world" tutorial in the Tock book, it can successfully flash the kernel to the board [26]. However, the tockloader tool cannot detect the Pico in order to load the userspace program. Instead it offers a simulator option. Not knowing whether I had the pieces in place, I wanted to verify my Pico board and attempted the "blink" program from the official Raspberry Pi Pico SDK. Blink worked which meant that I did not understand Tock enough. It could be possible that Tock development also expects x86 for the environment. I did not investigate this.

I had one more experiment that I wanted to attempt. The Adafruit KB2040 also is based on the RP2040 microcontroller that is in the Pico. It is priced higher at $8, but should be equivalent. My idea was that it acts as additional proof if I was able to make it work because the KB2040 is not listed as supported hardware; hopefully I could modify the existing source to add support. For this reason and the sake of time, I decided to try the pico-fido project [27] with the KB2040. When I tried to build the source from the main branch it failed. When I tried the release v6.0 branch it failed. It did not look promising. My last resort was to download the pre-built UF2 file from the releases. With this UF2 file copied onto the KB2040 mounted as a mass storage volume, the board acted successfully as the security when visiting WebAuthn.io. So in one way

it shows it can work, however I consider this result incomplete because I want to build from source in order to be confident enough to do maintenance and upkeep.

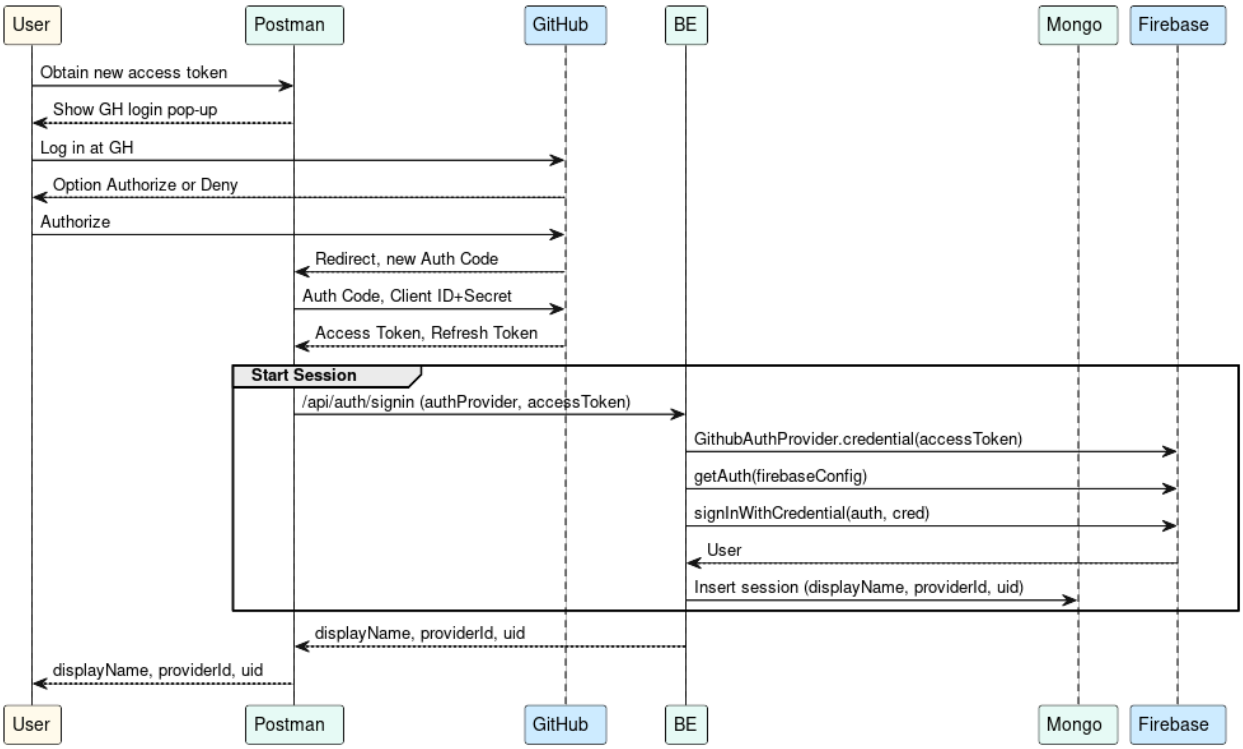

fig. 20, GitHub as ID provider

References

1. ASVS New Guide, Password Security; OWASP 2024; https://github.com/OWASP/ASVS/blob/master/5.0/en/0x11-V2-Authentication.md
2. ASVS Guide, Password Security; OWASP 2021; https://github.com/OWASP/ASVS/blob/master/4.0/en/0x11-V2-Authentication.md#v21-password-security-requirements

3. Most Common Passwords of 2024; NordPass Nov 2024; https://nordpass.com/most-common-passwords-list/
4. HIBP database; Troy Hunt; https://haveibeenpwned.com/Passwords
5. Meta fined $101M; Techcrunch Sep 2024 (https://techcrunch.com/2024/09/27/meta-fined-101-5m-for-2019-breach-that-exposed-hundreds-of-millions-of-facebook-passwords/)
6. Google released quantum resilient FIDO2 key; Bleeping Computer Aug 2023 (https://www.bleepingcomputer.com/news/security/google-released-first-quantum-resilient-fido2-key-implementation/)

7. Tock OS; (https://tockos.org/papers/)

20. Okta breach Nov 2023
(https://www.reuters.com/technology/cybersecurity/okta-says-hackers-stole-data-all-customer-support-users-cyber-breach-2023-11-29/)
21. Quantum resilient
(https://security.googleblog.com/2023/08/toward-quantum-resilient-security-keys.html)
22. Google releases OpenSK
(https://security.googleblog.com/2020/01/say-hello-to-opensk-fully-open-source.html)
23. OpenSK issue# 667 (https://github.com/google/OpenSK/issues/667)
24. NRF52840-Dongle, $10 DigiKey PN 1490-1073-ND
(https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/6470/NRF52840-DONGLE.pdf)
25. nrfutil
(https://devzone.nordicsemi.com/nordic/nordic-blog/b/blog/posts/nrf-util-unified-command-line-utility)
26. https://github.com/google/opensk/boards/raspberry_pi_pico/
27. https://github.com/polhenarejos/pico-fido.git