Proof-of-Concept-Implementierung einer Anwendung mit FIDO2 Support

Mara Schulke *(Matrikel-Nr. 20215853)*
Technische Hochschule Brandenburg
B.Sc. IT Sicherheit
Hardware Sicherheit

betreut durch Prof. Dr. Oliver Stecklina
Sommersemester 2022
Abgabetermin 4. Juni 2022

***Zusammenfassung*—Durch die Implementierung von FIDO2 können bestehende Systeme hinsichtlich ihrer Nutzbarkeit und Sicherheit stark verbessert werden und neue Systeme grundlegend neue Authentifizierungskonzepte implementieren. Die steigende Relevanz zeigt sich durch die Unterstützung großer Firmen. [5] Durch die Verwendung von ausgereiften Abstraktionen über die Protokolle *CTAP* und *WebAuthn* sind schnelle und zuverlässige Implementierungen des Standards leicht realisierbar.**

I. Einleitung

Authentifizierung ist eines der größten Probleme die durch verteilte Systeme entstehen. Es gibt zahlreiche Möglichkeiten die Identität einer Gegenseite sicherzustellen allerdings weisen viele von ihnen Schwachstellen hinsichtlich Man-In-The-Middle-Attacken und basieren auf der Annahme, dass das System des Nutzers nicht kompromitiert wurde. Als eine mögliche Lösung für viele dieser Probleme hat sich FIDO innerhalb der letzten Jahre zu einem beliebten Standard entwickelt der von vielen großen Anbietern wie Facebook, Amazon, Apple, Google, Mozilla, Microsoft etc. bereits untersützt beziehungsweise implementiert wird. [5]

II. Der FIDO2 Standard

FIDO2 steht für *F*ast *ID*entity *O*nline 2 und ist ein von der FIDO Alliance entwickleter, offener und lizenzfreier Standard für Hardware-Token gestützte Authentifizierung.

Ein Hardware-Token (kann auch in Form eines Trusted-Plattform-Moduls (TPM) oder als Teil des Betriebssystems implementiert sein) ist ein physischer Speicher für die FIDO Schlüsselpaare eines Nutzers. Kernmerkmale die FIDO2 von herkömmlicher asymmetrischer Kryptografie unterscheiden sind beispielsweise die Isolation der privaten Schlüssel auf dem Hardware-Token, die Notwendigkeit einer Nutzerinteraktion zum Verwenden eines privaten Schlüssels und die Generierung von einem Schlüsselpaar pro Online-Dienst. [3]

Durch all diese Eigenschaften werden Schlüsselverluste unwahrscheinlicher und weniger Sicherheitskritisch, da selbst bei einem hypothetischen Schlüsselverlustes der Schaden immer auf einen Online-Dienst begrenzt ist. Die größte Schwachstelle ist allerdings der physische Diebstahl des Hardware-Tokens, da dessen Besitz ausreicht für Impersonation-Attacken. Als Absicherung dagegen kann eine biometrische Authentifizierung erfolgen bevor ein privater Schlüssel verwendet werden kann - wie beispielsweise bei FaceID.

A. Welche Probleme löst FIDO2?

Die Notwendigkeit für einen solchen Standard hat sich in den letzten Jahren immer stärker gezeigt, da die klassische Passwort-Authentifizierung durch zunehmende Rechenleistung und effizientere Angriffe immer unsicherer und unhandlicher für Nutzer wird. Die minimale Passwortlängen steigen dementsprechend an und führen zur Wiederverwendung von gleichen Login-Daten für mehrere Online-Dienste. Bekannte Lösungen sind die Verwendung von sogenannten Passwort-Managern um lange und zufällige Passwörter für verschiedenste Online-Dienste zu verwenden ohne, dass sich Nutzer diese merken müssen. Solche Passwort-Manager sind zwar eine Lösung für die sichere Aufbewahrung von langen Passwörtern, können aber nicht die grundsätzlich durch Passwort-Authentifizierung eröffneten Angriffsvektoren wie z.B. Man-In-The-Middle-Attacken. So bald ein Angreifer in den Besitz des geheimen Wissens (in diesem Fall das Passwort) gelangt kann dieser uneingeschränkt und unbegegrenzt oft auf das Zielsystem zugreifen, bis der Nutzer seine Daten ändert (vorausgesetzt, der Angreifer hat dies noch nicht getan).

Durch den Wechsel von Passwort-Authentifizierung auf Zero-Knowledge-Proof basierte Authentifizierungsmethoden (in diesem Fall Public-Key-Authentifizierung) lassen sich ganze Angriffsvektoren ausschließen, da ein kompromitierter Server oder eine kompromitierte Verbindung niemals das geheime Wissen des Nutzers einem Angreifer zugänglich machen. Das heißt, dass sich ein Angreifer im Falle einer kompromitierten Verbindung maximal in die Sitzung des Nutzers einschleichen könnte, allerdings bei der nächsten Authentifizierung nicht erneut die Identität des Nutzers beweisen könnte und somit den Zugriff verlieren würde.

Im Falle von FIDO2 kennt nichtmal der Nutzer selber seine Schlüssel da diese auf einem externen Hardware-

Token oder TPM gespeichert werden und der Beweis der Identität durch die Signatur einer vom Authentifizierungs-Server ausgestellten Challenge erfolgt die das TPM oder der Hardware-Token intern durchführen und dem Nutzer nur die Signatur zurückgeben. So stellt selbst ein kompromitiertes Nutzersystem nur eine temporäre Schwachstelle dar.

III. Relevante Protokolle: CTAP & WebAuthn

Der FIDO2 Standard umfasst hauptsächlich die beiden Protokolle *CTAP* und *WebAuthn*. Diese unterteilen den gesammten Authentifizierungsvorgang in zwei Bereiche:

Client zu Authenticator also die Kommunikation zwischen dem Nutzersystem und dem Hardware-Token und *Client zu Server*, die Kommunikation zwischen dem Nutzersystem und dem Online-Dienst bei dem sich der Nutzer authentifizieren möchte. [3], [4]

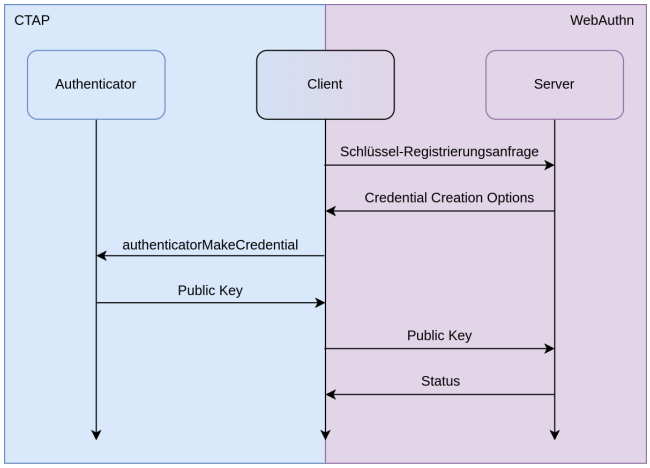

Abbildung 1. Unterteilung des Kommunikationsablaufs in CTAP und WebAuthn am beispiel einer Schlüssel-Registrierung

A. Gemeinsame Begriffsdefinitionen

1) Relying Party: Ein Online-Dienst der den FIDO2 Standard zur Nutzerauthentifizierung verwendet.

2) Authenticator: Ein externer Hardware-Token oder ein Teil des Betriebssystems (z.B. über ein TPM implementiert) der FIDO2 Schlüsselpaare verwaltet.

3) Credential: Ein FIDO2 Schlüssel

4) CBOR: Concise Binary Object Representation, ein Binär-Format zur Darstellung von JSON Objekten.

B. CTAP

Das Client-To-Authenticator-Protocol kurz CTAP ist Teil des FIDO2 Standards und beschreibt den Ablauf der Kommunikation zwischen einem Nutzersystem und dem Hardware-Token beziehungsweise dem Authenticator.

Neben einer Spezifikation für den Transportlayer / die Nachrichtenstruktur besteht das Protokoll primär aus der sogenannten “Authenticator API” - diese beschreibt Operationen die ein Nutzersystem auf einem Authenticator ausführen kann.

Spezifiziert sind die folgenden 6 Operationen:

1) *authenticatorMakeCredential* - Schlüsselpaar für eine “Relying Party” erstellen
2) *authenticatorGetAssertion* - Signatur einer Challenge
3) *authenticatorGetNextAssertion* - Nächste Signatur der Challenge erhalten bei mehreren Schlüsselpaaren
4) *authenticatorGetInfo* - Informationen über die Fähigkeiten des Authenticators
5) *authenticatorClientPIN* - Setzt den Authenticator-PIN
6) *authenticatorReset* - Zurücksetzen auf Werkseinstellungen

CTAP Spec references

Die im Standard beschriebenen Transportlayer umfassen USB, NFC oder Bluetooth Low Energy. Für jeden dieser Transportlayer gibt es eigene Mechanismen zum sicheren Verbindungsaufbau und zur Nachrichtenstruktur. CTAP Spec references

Die JSON Objekte werden innerhalb von *CTAP* mittels dem Datenformat *CBOR* dargestellt um eine effizientere Codierung zu erhalten und dennoch die kompatiblität mit dem weitverbreiteten Standard für Datenserialisierung zu garantieren. CBOR references

C. WebAuthn

Ein Server der das WebAuthn Protokoll implementiert enthält im wesentlichen zwei grundlegende Operationen die jeweils aus mehreren Schritten bestehen. Diese sind die Registrierung von Sicherheitsschlüsseln (Credentials) und deren zuordnung zu einem Nutzeraccount im System der Relying Party und die Austellung beziehungsweise Validierung von WebAuthn Challenges die bei erfolgreicher Validierung einem Nutzer zugriff auf das System der Relying Party gibt.

Ein wichtiger Punkt im Zusammenspiel der Spezifikationen und folgich der Implementierungen der beiden Protokolle *CTAP* und *WebAuthn* ist die kompatiblität der Datenttypen. So lässt sich die spezifizierte Ausgabe einer WebAuthn Registrierungs Challenge ohne veränderung mit der “authenticatorMakeCredential” Operation an den Authenticator weitergeben.

Dies hat zur Folge dass die Client-Implementierung selbst keine protokollrelevanten kryptografischen Berechnungen durchführen muss und effektiv der Server der Relying Party mit dem Authenticator des Nutzers kommuniziert. Die Client-Implementierung leitet lediglich jeweils die Ausgaben weiter.

weubauthn spec references

1) Registrierung: Um die Registrierung eines neuen Schlüssels durchzuführen muss die Relying Party auf dem Server `CredentialCreationOptions` für den Nutzer generieren und an den Client zurücksenden. Diese enthalten Informationen wie zum Beispiel erlaubte kryptografische Algorithmen, Informationen über den Nutzer und die Relying Party, ausgeschlossene Authenticator etc.

Diese Parameter nimmt der Client entgegen und gibt diese dann anschließend an die *CTAP*-Operation “authenticatorMakeCredential” weiter. Der Authenticator überprüft anschließend ob er ein Schlüsselpaar erstellen kann dass den Anforderungen des Servers entspricht und gibt entweder einen Fehler oder den öffentlichen Schlüssel an den Client zurück. Dieser wird nun vom Client an den Server weitergeben und die Registrierung wurde erfolgreich abgeschlossen.

weubauthn spec references

2) Authentifizierung: Für die Authentifizierung eines Nutzers mittels WebAuthn muss zu erst die Relying Party eine Challenge für den Client generieren. Diese muss zufällig und nur einmal gültig sein um Replay-Attacken zu verhindern. Nach dem der Client seine Challenge erhalten hat kann er diese über die CTAP-Operation “authenticatorGetAssertion” signieren lassen und die signierte Challenge wieder an den Server zurückgeben. Anschließend überprüft dieser ob einer der für den Nutzer hinterlegten Schlüssel verwendet wurde um die Challenge zu signieren. War dies der Fall gilt die Identität des Nutzers als verifiziert und der Server kann dem Nutzer Zugriff auf geschützte Ressourcen geben.

weubauthn spec references

Je nach Kommunikationsprotokoll zwischen dem Client und dem Server muss sich der Server den Status der Registrierung bzw. Authentifizierung über einzelne Nachrichten hinweg merken. So sind (auf Nachrichtenebene) verbindungslose Protokolle wie bspw. HTTPS davon betroffen. Eine bidirektionaler Transport würde dieses Problem umgehen, da sich der Server nur den Status für die Dauer der Verbindung merken müsste.

IV. Implementierung des Authetifizierungs-Servers

A. Zielsetzung

Ziel der Proof-of-Concept Implementierung ist es eine funktionale, flüchtige Nutzer-Verwaltung die optional FIDO2 Schlüssel als zweiten Login-Faktor unterstützt. Diese soll eine API anbieten auf der eine Client-Implementierung aufsetzen kann.

B. Authentifizierungsmechanismus

Nutzer ohne FIDO2 Schlüssel können sich bei dem Server mittels der Kombination aus E-Mail und ihrem Passwort authentifizieren und erhalten sofort einen Token zurück.

Die Nutzer die einen FIDO2 Schlüssel registriert haben erhalten nach initialer Angabe ihrer E-Mail und ihres Passworts eine Aufforderung ihren FIDO2 Schlüssel zu verwenden um letztendlich auch einen Token zu erhalten.

Der Token ist ein JSON-Web-Token der eine Nutzer-Id enthält und ein Ablaufdatum. Alle Tokens werden mit dem symmetrischen Algorithmus HMAC (Hash-based Message Authentication Code) signiert.

Alle geschützten beziehungsweise nutzerbezogenen Ressourcen die dieser Server vorhält dürfen nur durch die Mitgabe eines Tokens abgerufen werden. Dieser muss mit dem `Authorization`-Header gesetzt werden werden.

C. Speicherung der Nutzer-Passwörter

Da bekanntermaßen in Klartext gespeicherte Passwörter eine verherende Sicherheitslücke darstellen speichert der Server nur den Bcrypt-Hash der Passwörter die durch einen sogenannten Salt randomisiert werden.

Sowohl der Schlüssel zur signierung von Tokens als auch der Salt für Bcrypt müssen in einem realen Anwendungsszenario geheim gehalten werden.

D. Datenstrukturen zur flüchtigen Verwaltung von Nutzern

Ein Nutzer erhält bei der Registrierung eine eindeutige Id, einen Verifizierungscode (der in einem realen Anwendungsszenario ggf. per E-Mail versendet werden könnte) und eine leere Liste an FIDO2 Schlüsseln.

Die Nutzer werden in einer HashMap im RAM vorgehalten, eine Datenbankanbindung des Servers zur Nutzerverwaltung wären bei einem Produktivsystem unabdingbar.

Um die konsistzenz der Nutzerverwaltung innerhalb des RAM sicherzustellen befindet sich die Nutzer-HashMap in einem Mutex um gleichzeitige ggf. inkonsistente Schreibvorgänge zu unterbinden.

Bei jeder eingehenden Anfrage wird die Datenstrukturen zur Nutzerverwaltung der Anfrage zugewiesen, diese hat nun die Garantie dass sie die einzige Anfrage ist die in diesem Moment schreiben kann. Sollten mehrere Anfragen gleichzeitig eingehen muss eine warten bis die andere abgearbeitet wurde. Dies ist zwar absolut inperformant bei großen Anfragemengen aber bei einem Proof-of-Concept-Server mit sehr wenigen Nutzern vertretbar.

E. Implementierung von WebAuthn

Da die Erstellung von den durch WebAuthn definierten Datenstrukturen und die kryptografischen Algorithmen nicht trivial zu implementieren sind gibt es für gängige Sprachen die eine Anwendung im Web-Kontext finden Bibliotheken die das WebAuthn-Protokoll abstrahiert bereitstellen.

Für Rust ist eine umfangreicherere dieser Bibliotheken “webauthn-rs”, verfügbar auf GitHub unter der Mozilla-Public-License-2.0:

https://github.com/kanidm/webauthn-rs

Die Bibliothek lässt sich durch Informationen über den Server konfigurieren: Es wird eine Relying Party Id und eine registrierbare Domain erwartet um die Identität des Servers für die Schlüsselerstellung wiederverwenden zu können.

Nach erfolgreicher Initialisierung gibt es im wesentlichen 4 relevante Funktionen mit denen sich der WebAuthn-Flow komplett implementieren lässt:

1) start_securitykey_registration: Gibt die Credential-Creation-Options und einen Schlüssel-Registrierungsstatus für den momentanen Nutzer zurück. Dieser muss persistiert werden.

2) finish_securitykey_registration: Nimmt einen Schlüssel-Registrierungsstatus und den generierten Schlüssel und validiert die Gültigkeit des generierten Schlüssels.

3) start_securitykey_authentication: Gibt eine Challenge und einen Schlüssel-Authetifizierungsstatus für den momentanen Nutzer zurück. Dieser muss muss persistiert werden.

4) finish_securitykey_authentication: Nimmt eine Schlüssel-Authetifizierungsstatus und eine signierte Challenge und validiert diese gegeninander.

Somit sind Serverseitig alle Voraussetzungen geschaffen um eine API bereitzustellen die WebAuthn unterstützt. Detailierte Informationen zu den bereitgestellten Endpunkten finden sich unter Abschnitt VI.

V. Implementierung der Nutzer-Oberfläche

A. Zielsetzung

Ziel der Client-Implementierung ist es beispielhaftes Nutzer-Oberfläche zu schaffen das die unter Abschnitt VI beschriebene API verwendet um ein Nutzer-Login mit FIDO2 zu ermöglichen.

B. Konzept der Oberfläche

Die Oberfläche ist eine sehr kleine JavaScript Anwendung für die API. Folglich orientieren sich die Aktionen die der Nutzer in diesem ausführen kann direkt an dieser. In der Abbildung 2 sind der Ablauf aller möglichen Nutzeraktionen und die dazugehörigen API Endpunkte abgebildet.

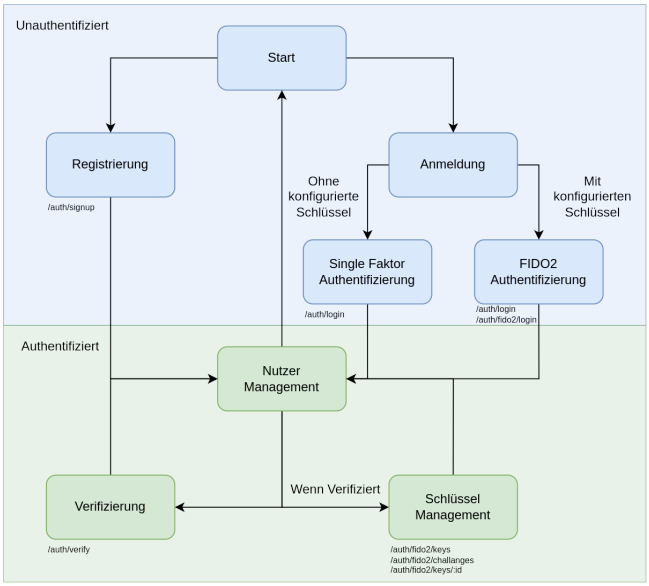

Abbildung 2. Abfolge möglicher Nutzer-Aktionen

C. Anwendungs Architektur

Da der Umfang der Nutzer-Aktionen sich stark in Grenzen hält gibt es keinen Grund eines der komplexeren Frameworks einzusetzen die typischerweise für Web-Oberflächen eingesetzt werden. Die einzige externe Bibliothek ist eine moderene und stabilere Base64 Implementierung als die Browser-Native Alternative.

Das Grundkonzept zum rendering der Oberfläche ist einen Anwendungszustand durch eine pure (also Seiten-Effekt freie) Funktion in HTML zu übersetzen. Nutzeraktionen können diesen Zustand durch Interaktion verändern und lösen somit einen neuen Render-Vorgang aus. Diese bzw. ähnliche Architekturen findet sich in Frameworks wie React oder Elm wieder.

refs

Der Zustand speichert den Token, Verifizierungsstatus, Ladezustand der Anwendung und die Liste aller registrierten Schlüssel.

D. Interaktionen

Die definierten möglichen Interaktionen die der Nutzer tätigen kann sind:

1) signup
2) verify
3) login
4) logout
5) addKey
6) removeKey

Diese werden durch die entsprechenden Bedienfelder der Oberfläche ausgelöst.

E. Implementierung von CTAP

Da die Nutzer-Oberfläche in JavaScript geschrieben ist und dazu gedacht ist in einem Browser-Kontext ausgeführt zu werden kann die Browser-Implementierung für CTAP verwendet werden was die komplexität der Anwendung drastisch veringert, da diese Implementierung sehr stark abstrahiert ist.

Im wesentlichen werden nur zwei der bereitgestellten Funktionen in der gesamten Nutzer-Oberfläche verwendet:

1) navigator.credential.create (führt die Operation “authenticatorMakeCredential” aus)
2) navigator.credential.get (führt die Operation “authenticatorGetAssertion” aus)

Da die Parameter dieser Funktionen von dem Server bereitgestellt werden ist die Implementierung der Kommunikation mit dem Authenticator also sehr trivial.

VI. Technische Dokumentation

Der Server öffnet den TCP Port 8080 und erwartet eine externe TLS Terminierung. Tokens können dem Server über den `Authorization` Header mitgegeben werden und der `Content-Type` aller Anfragen und Antworten ist ausschließlich `application/json`.

A. */auth/signup - Nutzer erstellen*

Methode: POST
Token: -
Eingabe: Credentials { email, password }
Ausgabe: UserDetails { token, verified, keys }
Beschreibung: Erstellt einen unverifizierten Nutzer ohne FIDO2 Keys. Gibt einen Verifizierungscode in den Server-Logs aus (könnte in einem echten Szenario per E-Mail verschickt werden).

B. */auth/verify - Nutzer verifizieren*

Methode: POST
Token: Notwendig
Eingabe: Verification { code }
Ausgabe: -
Beschreibung: Verifiziert einen Nutzer falls der Code mit dem bei der Registrierung generierten Code übereinstimmt.

C. */auth/login - Nutzer anmelden*

Methode: POST
Token: -
Eingabe: Credentials { email, password }
Ausgabe: UserDetails { token, verified, keys } | CredentialRequestOptions
Beschreibung: Gibt dem Nutzer entweder seine UserDetails zurück oder stellt eine WebAuthn Authetifizierungs-Challenge die der Nutzer signiert bei dem Endpunkt */auth/fido2/login* enreichen muss falls ein FIDO2 Schlüssel hinterlegt wurde.

D. */auth/fido2/login - Nutzer mit WebAuthn anmelden*

Methode: POST
Token: -
Eingabe: PublicKeyCredential
Ausgabe: UserDetails { token, verified, keys }
Beschreibung: Validiert die WebAuthn Challenge des Nutzers mit den hinterlegten FIDO2 Schlüsseln und gibt bei erfolgreicher Validierung dem Nutzer seine UserDetails zurück.

E. */auth/fido2/challenges - WebAuthn Registrierungs Challenge*

Methode: POST
Token: Notwendig
Eingabe: -
Ausgabe: CredentialCreationOptions
Beschreibung: Startet einen Registrierungsprozess für einen FIDO2 Schlüssel. Setzt Serverseitig den “Key-Registration-State” eines Nutzers und gibt die Registrierungs-Optionen inklusive einer Registrierungs-Challenge zurück.

F. */auth/fido2/keys - WebAuthn Registrierung abschließen*

Methode: POST
Token: Notwendig
Eingabe: RegisterPublicKeyCredential
Ausgabe: Key { id }
Beschreibung: Nimmt die CTAP Ausgabe der “authenticatorMakeCredential” Operation an und ordnet diesen Schlüssel dem Nutzer zu.

G. */auth/fido2/keys/:id - WebAuthn Schlüssel entfernen*

Methode: DELETE
Token: Notwendig
Eingabe: */:id*
Ausgabe: -
Beschreibung: Entfernt einen FIDO2 Schlüssel anhand seiner ID.

VII. Auswertung

A. Nutzbarkeit

Das System ist unter Einschränkungen voll funktional und bietet eine verhältnismäßig gute Nutzbarkeit. Durch die Verwendung eines flüchtigen Speichermediums (RAM) für die Nutzerverwaltung sind nach jedem Neustart alle Anwendungsdaten zurückgesetzt. Dies ist für Test-Zwecke absolut in Ordnung, und ließe sich für eine reale Anwendung leicht durch die Implementierung einer Datenbankanbindung beheben.

Die Nutzer-Oberfläche ist sehr schlicht, lässt aber den Nutzer übersichtlich alle definierten Interaktionen ausführen. Auch hier ließe sich die ausgestaltung der Oberfläche für eine reale Anwedung mit der entsprechenden Fachexpertise verbessern.

Die Nutzerauthentifizierung an sich verläuft zuverlässig, schnell und reibungslos. Neben der allgemeinen Dateneingabe (E-Mail und Passwort) besteht die Signierung der Challenge lediglich aus einem Tippen auf den Authenticator. Dies allein bietet ein deutlich verbessertes Erlebnis im vergleich zu SMS-Tokens oder One-Time-Passwords als zweiten Faktor (vorrausgesetzt der Authenticator befindet sich in der Nähe des Systems oder ist sogar bereits angeschlossen).

B. Risiken

Die Hauptrisiken des momentanen Systems ist die fehlende Passwortvalidierung und der fehlende Zwang einen FIDO Schlüssel bei der Registrierung zu hinterlegen. So ist es momentan möglich einen Nutzer mit einem schwachen Passwort anzulegen und keinen Schlüssel zu registrieren. Dies ließe sich gegebenenfalls auch durch Anpassung des Authentifizierungs bzw. Registrierungsprozesses beheben.

Unter der Annahme dass eine Schlüssel-Registrierung verpflichtend sei wäre ein Angriff auf einen Nutzeraccount nurnoch durch physikalische Entwendung des Hardware Tokens möglich und dementsprechend unwahrscheinlich. Somit kann unter dieser Annahme das System als sicher angesehen werden.

C. Mögliche Verbesserungen

Als abschließende Verbesserung wäre ein Umstellen auf Single-Faktor-Authentifizierung mit dem Hardware-Token hinsichtlich des Nutzererlebnises empfehlenswert (Sicherheitsimplikationen außer Acht gelassen).

Eine alternative Möglichkeit könnten weiterhin zwei Faktoren sein, allerdings unter der Bedingung, dass das Passwort lediglich zur einrichtung neuer Systeme abgefragt wird und sonst ein Single-Faktor-Login mit dem Hardware-Token möglich ist.

Desweiteren ist es bei Hardware Tokens - deren physikalischer Verlust nicht gänzlich auszuschließen ist - ratsam einen Backup Token zu besitzen und diesen bei jedem Online-Dienst als Backup Token zu hinterlegen um einen Schlüsselverlust abfangen zu können. Eine bekannte Alternative zu einem zweiten Hardware Token wären auch sogenannte Backup Codes, die bei Schlüsselverlust einmaligen Zugriff gewähren.

All diese ausgereiften Konzepte werden in der momentanen Proof-of-Concept-Implementierung noch außer Acht gelassen, ließen sich allerdings leicht mit geringem zusätzlichen Aufwand hinzufügen.

ABBILDUNGSVERZEICHNIS

LITERATUR

[1] “Fido2: Der neue standard für den sicheren web-log-in,” Jul 2020, [Abgerufen am 05. Juni 2022]. [Online]. Available: https://www.ionos.de/digitalguide/server/sicherheit/was-ist-fido2/
[2] “Ctap: Protokoll für mehr sicherheit & komfort im web,” Sep 2021, [Abgerufen am 05. Juni 2022]. [Online]. Available: https://www.ionos.de/digitalguide/server/sicherheit/client-to-authenticator-protocol-ctap/
[3] “Client to authenticator protocol (ctap),” Feb 2018, [Abgerufen am 05. Juni 2022]. [Online]. Available: https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-client-to-authenticator-protocol-v2.0-id-20180227.html#CTAP2ProtocolOverview
[4] “Web authentication: an api for accessing public key credentials,” Jun 2022, [Abgerufen am 05. Juni 2022]. [Online]. Available: https://w3c.github.io/webauthn/
[5] “Fido alliance member companies & organizations,” Apr 2022, [Abgerufen am 05. Juni 2022]. [Online]. Available: https://fidoalliance.org/members/

ANHANG

Quellcode der Implementierung

https://github.com/mara214/fido2-auth

Zur Entwicklung verwendeter Authenticator

https://store.google.com/de/product/titan_security_key