Implementierung eines Atuhentifizierungs-Servers 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 28. Mai 2022

***Zusammenfassung*—Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.**

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. tbd

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. (quote)

Ein Hardware-Token (kann auch in Form eines Trusted-Plattform-Moduls 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.

Durch all diese Eigenschaften werden Schlsselverluste 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. Eine Absicherung dagegen kann eine biometrische Authentifizierung erfolgen bevor ein privater Schlüssel verwendet werden kann - wie beispielsweise bei FaceID. tbd

A. Welche Probleme löst FIDO2?

Die Notwendigkeit für einen solchen Standard hat sich in den letzten Jahren immer stärker gezeigt, da die klassische Knowledge-Based-Authentication (KBA) 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 sog. 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 durch KBA 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 KBA auf Zero-Knowledge-Proof basierte Authentifizierungsmethoden 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 Trusted-Plattform-Modul TPM oder einem externen Hardware-Token gespeichert wird und der Beweis der Identität durch die

Signatur einer vom Authentifizierungs-Server ausgestellten Challange 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.

B. Übersicht von FIDO2 kompatibler Hardware? Sinnvoll?

III. Relevante Protokolle: CTAP & WebAuthn

A. 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*
2) *authenticatorGetAssertion*
3) *authenticatorGetNextAssertion*
4) *authenticatorGetInfo*
5) *authenticatorGetClientPIN*
6) *authenticatorReset*

B. WebAuthn

IV. Implementierung des Authetifizierungs-Servers

V. 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 } | webauthn challange
Beschreibung: Gibt dem Nutzer entweder seine UserDetails zurück oder stellt eine WebAuthn Authetifizierungs-Challange 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: challange
Ausgabe: UserDetails { token, verified, keys }
Beschreibung: Validiert die WebAuthn Challange des Nutzers mit den hinterlegten FIDO2 Schlüsseln und gibt bei erfolgreicher Validierung dem Nutzer seine UserDetails zurück.

E. /auth/fido2/challenges - WebAuthn Registrierungs Challange

Methode: POST
Token: Notwendig
Eingabe: -
Ausgabe: ccr
Beschreibung: Startet einen Registrierungsprozess für einen FIDO2 Schlüssel. Setzt Serverseitig den “Key-Registration-State” eines Nutzers und gibt eine Registrierungs-Challange zurück.

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

Methode: POST
Token: Notwendig
Eingabe: PublicKeyCredential
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.

VI. Auswertung

VII. Abbildungsverzeichnis