



Hochschule für Telekommunikation Leipzig
University of Applied Sciences

Hochschule für Telekommunikation Leipzig (FH)
Institut für Telekommunikationsinformatik

**Abschlussarbeit zur Erlangung des akademischen Grades
Bachelor of Science**

Thema: „Inwiefern bietet die Authentifikation ohne Passwort Vor- und Nachteile gegenüber Webanwendungen hinsichtlich Nutzerfreundlichkeit, Datenschutz und Sicherheit?“

Vorgelegt von: Ertugrul Sener

Geboren am: 17.10.1998

Geboren in: Berlin

Vorgelegt am: 4. Oktober 2020

Erstprüfer: Prof. Dr. Erik Buchmann

Hochschule für Telekommunikation Leipzig
Gustav-Freytag-Straße 43-45
04277 Leipzig

Zweitprüfer: Juri Lobov

T-Systems International GmbH
Holzhauser Straße 1-4
13509 Berlin

1 Vorwort

Vor Ihnen liegt die Bachelorarbeit zur Forschungsfrage „Inwiefern bietet die Authentifikation ohne Passwort Vor- und Nachteile gegenüber Webanwendungen hinsichtlich Nutzerfreundlichkeit, Datenschutz und Sicherheit“.

Diese habe ich in Zusammenarbeit mit der Telekom Security im Rahmen meines dualen Studiums für angewandte Informatik an der Hochschule für Telekommunikation Leipzig angefertigt. Ziel ist es typische Authentifizierungsstrategien kritisch zu bewerten und eine eigene prototypische Lösung zu entwickeln. Diese soll die Anforderungen an Nutzerfreundlichkeit, Datenschutz und Sicherheit bestmöglichst erfüllen und über die selektive Auswahl aus den möglichen Authentifizierungsstrategien gleichzeitig die damit einhergehenden Nachteile minimieren.

Die Fragestellung habe ich zusammen mit meinem Hauptprüfer Prof. Dr. Erik Buchmann von der HfTL und meinem betrieblichen Vorgesetzten Juri Lobov entwickelt. Ich bin davon überzeugt, dass mir die Kombination aus theoretischem Forschungshintergrund und langjähriger praktischer Erfahrung bei einer umfangreichen Beantwortung der Forschungsfrage behilflich sein wird.

Daher möchte ich meinen Begleitern für ihre Unterstützung bei der Erarbeitung meiner Arbeit und der Betreuung danken. Ebenfalls möchte ich an dieser Stelle meinen Dank an meine Kollegen bei der Telekom Security richten, die mir jederzeit Ihre Unterstützung zugesichert haben.

Ich wünsche Ihnen viel Freude beim Lesen.

Ertugrul Sener

4. Oktober 2020

Inhaltsverzeichnis

1 Vorwort	3
2 Einleitung	7
2.1 Motivation	7
2.2 Problemdefinition	7
2.3 Stand der Forschung	9
2.3.1 Bequemlichkeitsproblem	9
2.3.2 Unsichere Passwörter	9
2.3.3 Übertragungsproblem	13
2.4 Zielsetzung	16
3 Grundlagen	19
3.1 IT-Grundschutzkriterien	19
3.2 Standard nach FIDO2	22
3.3 Behandelte Verfahren	29
3.3.1 Username & Passwort	29
3.3.2 OTP's	30
3.3.3 WebAuthn	33
4 Konzeption	37
4.1 Auswahl der Authentifizierungsverfahren	37
4.2 Authentifikation in drei Schichten	39
4.3 Kriterien für eine erfolgreiche Authentifikation	40
5 Prototypischer Lösungsansatz	43
5.1 Implementationsdetails	43
5.2 Fehlerbetrachtung	50
6 Auswertung	57
7 Ausblick und Fazit	59
Selbstständigkeitserklärung	61
Abkürzungsverzeichnis	63

Abbildungsverzeichnis	65
Tabellenverzeichnis	67
Quellcodeverzeichnis	69

2 Einleitung

2.1 Motivation

Die Motivation für die Erarbeitung der Forschungsfrage ergab sich am 29. Juni 2020, als eine Rundmail von dem Data Breach Monitoring-Tool von Firefox die Bildschirme erreichte, welches einen neuen Incident bei Wattpad meldete. Wattpad ist eine Webseite, bei welcher Nutzer Geschichten frei für die Öffentlichkeit schreiben und publizieren können. Nun stellt sich die berechtigte Frage, inwiefern diese in Relation kleine Webseite verwertbare Daten für einen Angreifer oder ein böses Tool liefern kann. Dazu genügt es sich die Daten genauer anzusehen. Kompromittiert wurden neben den klassischen Daten wie Passwörtern, IP-Adressen, E-Mail-Adressen und Geburtsdaten auch sehr persönliche Informationen wie geografische Standorte, ein Kurzprofil des Nutzers, das Profilbild und über sogenannte Session-Tokens auch die Social-Media-Profile der Nutzer. Entnehmen kann man dies mehreren Artikeln, bei der die Angabe nach der Größe der Datenmengen zwischen 250 und 280 Millionen Einträgen variiert. Eine genaue Zahl liefert der Artikel der riskbasedsecurity mit 268.830.266 (nach der Entfernung von Duplikaten) kompromittierten E-Mail Adressen, die sich laut des Artikels in einer einzigen Datenbank befanden. [7] Diese Zahl ist dennoch nur als Richtwert zu verstehen, da wie bereits erwähnt das Ausmaß des Datenlecks bei Wattpad nicht in Gänze bekannt ist.

2.2 Problemdefinition

Die Identität dieser Nutzer hängt an ihrem Passwort, demnach ist die plötzliche Bedrohung für jeden Einzelnen nun allgegenwärtig. Das Passwort, welches nun im Netz ist, ist schwierig bis unmöglich aus allen (Sub-)Quellen eliminierbar. Die Identität der Personen kann nun von Angreifern genutzt werden, um z.B. finanziellen Schaden anzurichten und sich selbst zu bereichern.

Auch wenn es bei der Bekanntmachung und Berichterstattung zu solchen Datenlecks manchmal so scheint, ist auch dieses nicht das einzige Beispiel für kompromittierte Unternehmen der letzten Jahre. Angriffsszenarien auf technischer Seite (Code-Injektionen, 0-Day-Exploits, Brute-force-Angriffe) scheint die informationsaffine Menschheit der Neuzeit gut organisiert zu bekommen, doch dann scheitert es in riesigen Unternehmen oft an der Wahl eines Passwortes einer wichtigen Einzelperson, welches dann in Kettenreaktionen zu solchen Data Breaches führen. Passwörter wichtiger Personen innerhalb eines Unternehmens können nun ohne große Mühen über die Google Suchergebnisse gefunden werden. So liefert die Anfrage `'allintext:password filetype:log after:2019'` auf Google alle Logdateien nach 2019, die den Schlüssel `"password"` beeinhalteten. Alternativ kann man über Anfragen folgender Art: `'wattpad data breach download'` gezielt nach einem Downloadlink für spezielle Data Breaches suchen. Solche Listen werden häufig in unbekannten Foren geteilt. Meistens erscheinen Username und Passwort im Format `'Username:Passwort'`, der erste Doppelpunkt trennt demnach die beiden Zeichenketten. Der Zeilenumbruch trennt die unterschiedlichen Accounts.

Bei Diskussionen im Internet, wird sich häufig aufgrund der selben (wiederholenden) Gründe gegen eine Zwei-Faktor-Authentifizierung (2FA) und zusätzliche Sicherheiten für die Passwortwahl ausgesprochen. Zum Einen sei ein Verlust des Gerätes, mit dem die Webseite gekoppelt ist, gleichzusetzen mit dem Verlust der Identität. Ohne entsprechende Backup-Codes kann womöglich kein Zugang mehr gewährt werden, da das Zurücksetzen des Passwortes meist auch die Authentifizierung durch den zweiten Faktor benötigt. Außerdem spielt die Bequemlichkeit eine große Rolle. Ein User will so schnell wie möglich und ohne Zwischenschritte Zugang zu seinen Diensten erhalten. Ein zweiter Faktor bedeutet das nach Eingabe des Passwortes das Smartphone gesucht, womöglich noch entsperrt, die zugehörige App geöffnet und der Code (PIN, Fingerabdruck, Gesichtsmuster) an den Server übertragen und verarbeitet werden muss.

Das Speichern von allen Passwörtern, durch einen Passwort-Manager, an einem gesammelten Ort erscheint dem Durchschnittsnutzer als zu riskant. Der Verlust des Masterpasswortes würde den Verlust aller Identitäten des Nutzers bedeuten, sofern er die Daten nicht auswendig kennt oder sich anderweitig notiert hat. Das potenzielle Risiko welches durch verstreute Passwörter im Vergleich zu den Risiken neuerer Verfahren aufkommt, wird häufig ignoriert. Die Frage, die sich stellt ist es, ob man mit einer Kombination aus den vorhandenen vielfältigen Authentifizierungsmöglichkeiten eine bequeme aber gleichzeitig sichere Authentifizierungsvariante schaffen kann, bei denen es dem Anwender möglich sein soll, sich gegenüber einer Webseite zu authentifizieren.

2.3 Stand der Forschung

2.3.1 Bequemlichkeitsproblem

Bei einem Angriff ist es nicht zwingend notwendig, dass die Authentifizierungsquelle, also jene Quelle bei der die Daten persistiert sind und die die Authentifizierung bei Eingabe durchführt, diese Daten durch fehlerhafte Programmierung herausgibt. Durch sogenannte Metadaten, das sind Informationen und Merkmale zu Daten, ist es Angreifern häufig möglich das Passwort zu erraten bzw. zurückzusetzen. Wenn also der Benutzername lautet 'MichaelJacksonFanForever' ist die Antwort auf die Sicherheitsfrage zum Lieblingskünstler nicht weit entfernt.

Gleichzeitig neigen Menschen aufgrund von Bequemlichkeit zu leicht merkbaren Passwörtern, die sie mehrfach für verschiedene Dienste verwenden. Dies stützt eine kürzlich durchgeführte repräsentative Studie der Bitkom Reserach [1] im Auftrag des Digitalverbands Bitkom. So nutze etwa jeder dritte Onlinenutzer (36%) dasselbe Passwort für mehrere Dienste. Auch wenn gleichzeitig 63% der Befragten angaben, bei der Erstellung von Passwörtern auf "einen Mix aus Buchstaben, Zahlen und Sonderzeichen" zu achten, beweist diese Befragung an 1.000 Internetnutzern, dass die Frage nach der Sicherheit von Passwörtern auch im Jahre 2020 immernoch Relevanz hat. Eine ähnliche Studie hat die Bitkom zum Thema 'Nachlässigkeit bei Passwörtern' am 08.11.2016 [2] gemacht, bei der die prozentuale Verteilung an unsicheren Passwortnutzern die befragt wurden nur einen Prozent höher liegt. Das heißt konkret, dass sich innerhalb von 4 Jahren keine messbare Besserung ergeben hat. Das Bewusstsein über die Internetpräsenz und der Schutz dessen scheinen immernoch keine große Aufmerksamkeit vom modernen Nutzer zu erhalten. Das Problem mit unsicheren Passwörtern ist allerdings so alt wie das Internet.

2.3.2 Unsichere Passwörter

In der fünften Ausgabe der Zeitschrift "Wirtschaftsinformatik & Management" 2018 mit dem Titel "Schwache Passwörter - Nutzer spielen weiterhin Vogel Strauß" schrieb der Autor Geralt Beuchelt: "Der Umgang mit Passwörtern ist so ähnlich wie eine Diät: Eigentlich weiß man genau, was richtig ist - Macht aber oft genug das Gegenteil. Und nicht selten ist der Grund Bequemlichkeit. Warum selber kochen, wann nach einem langen Tag eine Pizza lockt? Und warum lange, umständliche Passwörter verwenden, wenn es einfach zu merkende, die man für alle Accounts verwendet, doch auch tun?" [3].

Damit greift der Autor sehr zutreffend die Hauptproblematik des Passwortes in der Neuzeit auf. Die symbolische Pizza steht für die Mehrfachverwendung von teils schwachen Passwörtern für alle genutzten Dienste inklusive des 'Verwaltungsdiens-tes' wie der Mail, welches als meist einziges Identifikationsmerkmal dient, über die weitere Dienste betroffen sein können. Der Begriff des Passwortes stammt aus dem militärischen Bereich des 16. Jahrhunderts, wobei tatsächlich das einzelne Wort adressiert war, welches einem Zutritt zu Gebäuden verschaffte. Damit verwandt ist das Kennwort, welches nicht das Passieren sondern die Kennung des gemeinsamen Geheimnisses betont. Dies bedeutet, dass der Passierer mit einem Kennwort auch automatisch ein Geheimnisträger ist. Als Computer immer leistungsfähiger wurden, wurde der Begriff der Passphrase etabliert, um die Notwendigkeit längerer Passwörter hervorzuheben. Dies geschah vor allem weil zu dieser Zeit keine Forschung rund um Passwörter betrieben wurde und die Komplexität noch keine Rolle fand. Weitere Schlüsselwörter für das heutzutage bekannte Passwort sind: Schlüsselwort, Kodewort (u.a: Codewort) oder die Parole, diese werden im Folgenden als Synonym für die heutige Passphrase verwendet. Die Länge gilt gemeinhin als die allumfassende Sicherheit von Passwörtern. Dem widerspricht die klare Trennung zwischen Länge und Komplexität von Passwörtern durch das Bundesamt für Sicherheit in der Informationstechnik, die sinngemäß in ihrer Empfehlung zum Thema "sichere Passwörter" schreiben, dass die Länge von Passwörtern nicht dessen Komplexität und Sicherheit gegen Angriffe widerspiegelt [4].

Um sichere Passwörter zu erzwingen, setzen Webseiten immer häufiger auf Passwort Polycys. Diese machen anhand von Regeln, die die Entwickler meist selbst aufstellen, das Wählen von Passwörtern komplizierter. Dies resultiert in Passwörtern mit Mindestlängen, einem Mindestzeichensatz und weiteren Regeln wie der Verbot von sich wiederholenden Zeichenketten wie "testtest", längeren Zahlenreihenwiederholungen wie "123123" oder dem Verbot der Nutzung des Usernamen im Passwort. Das gewählte Passwort soll für einen selbst leicht merkbar, für einen Computer oder menschlichen Angreifer schwer zu erraten sein. So empfiehlt das Bundesamt für Sicherheit in der Informationstechnik (BSI) Passwörter zu verwenden, die möglichst nicht aus Tastaturmustern bestehen wie 'asdfgh' oder '1234abcd'. Das BSI erwähnt in dem Ratgeber für sichere Passwörter häufig die Zeichenarten. Zu diesen gehören sowohl Buchstaben in Groß und Kleinschreibung, Zahlen, besondere Umlaute und Sonderzeichen. Allgemein wird ein 20 bis 25 Zeichen langes Passwort aus zwei Zeichenarten einem acht bis 12 Zeichen langem Passwort aus vier Zeichenarten in Punkto Komplexität gleichgesetzt. [4]

Das Wort 'Policy' ist in diesem Zusammenhang als 'die Regel' zu verstehen und in der Wortkombination sind Passwort Polycys die Regeln, die zu einem sicheren Passwort führen. Derart Regeln gibt das BSI vor. So seien der Kreativität bei Passwörtern

keine Grenzen gesetzt [4]. Zum Beispiel könne man einen leicht zu merkenden Satz nehmen, diesen mit Bindestrichen verbinden und von jedem Wort den ersten Buchstaben entfernen. Die Frage die sich dabei stellt ist es, ob dieser Satz dann die Tippgeschwindigkeit des Nutzers beeinträchtigt, weil relativ viele Denkprozesse während des Tippens stattfinden müssen. Zunächst ein Mal müsste sich hierbei der Satz in voller Länge gemerkt werden, dies ist noch recht unkompliziert für den allgemeinen Internetnutzer. Danach muss der Satz während des Tippens bereits mit Bindestrichen verbunden werden, auch dies ist noch kein großes Problem. Zum Problem wird es, sich die ersten Buchstaben beim Tippen automatisch wegzudenken, sodass einem kein Fehler unterläuft. Wenn einem doch ein Fehler unterläuft, ist es anders als bei gängigen Passwörtern, sehr schwer zur Fehlerquelle zu springen und den Fehler zu lokalisieren. Alternativ bleibt einem nur das Neutippen des Passwortes, welches eine große Zeitverzögerung für den Nutzer bedeutet und auf lange Sicht den Nutzer dazu bringen wird, ein einfacheres und leicht tippbares Passwort zu wählen.

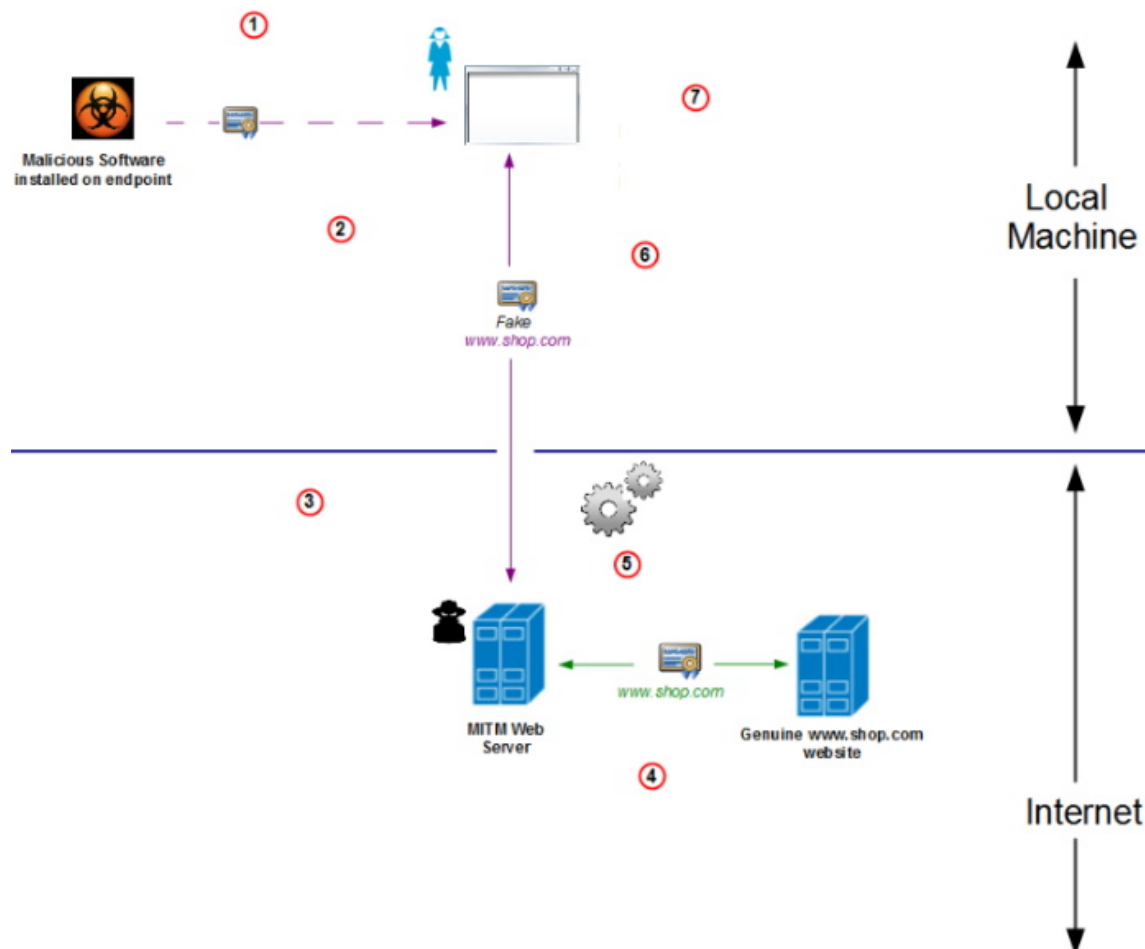
Dies bedeutet wie bereits oben angeschnitten nicht, dass das Passwort bestehend aus Buchstabenwiederholungen wie z.B. "aaaaaaaaaaaaaaaaaaaaa" ein mathematisch sicheres Passwort ist. Die Länge des Passwortes ist nur einer von vielen Faktoren, die am Ende zur Komplexität und der daraus resultierenden Sicherheit beitragen. Im Idealfall besteht das lange Passwort aus mehreren Zeichenarten. Eine weitere Empfehlung ist es, keine Sonderzeichen an den Anfang oder das Ende des Passwortes anzuhängen [4], um es für einen Angreifer schwerer erratbar zu machen. Dies lässt sich damit begründen, dass sobald ein Angreifer die restlichen Zeichen des Passwortes erraten konnte oder durch Metadaten anderer Dienste (wie oben in dem Michael-Jackson Beispiel) kennt, das Durchprobieren von allen verfügbaren Sonderzeichen für die erste und letzte Stelle der Zeichenkette keine große Leistung erfordert. Sie machen das Passwort mathematisch zwar sicherer (Mehr Zeichenarten bedeuten mehr Zeichen insgesamt und dadurch mehr Kombinationsmöglichkeiten für Passwörter), bei gegebenen Umständen sind diese einzelnen Sonderzeichen obsolet und können weggelassen werden. Passwort Policies können auch teilweise wertvolle Informationen für einen potenziellen Angreifer bieten. Denn was Angreifer durch sehr strikte Passwort Policies unter anderem erkennen können, ist die Mindest- und Maximalzeichenlänge. Dabei wird der Angreifer zum Beispiel aus der Regel 'Das Passwort muss mindestens 8 und maximal 16 Zeichen lang sein.' alle Kombinationen für weniger als 8 Zeichen und mehr als 16 Zeichen bei der Erratung eliminieren können. Weitere Regeln wie 'Das Passwort muss mindestens ein Sonderzeichen beinhalten' können zusätzliche Informationen bieten. Daher sind Passwort-Policies zwar ein sehr wichtiges Werkzeug, um Nutzer zu sicheren Passwörtern zu zwingen. Durch die beeinträchtigte Bequemlichkeit in der freien Passwortwahl des Nutzers können dennoch einfach zu erratende Passwörter gewählt werden, die den Kriterien

entsprechen. Verhindern lässt sich dies nicht ganz. Die Informationen die Nutzer beim Anmelden bekommen, nutzen Angreifer dann zum Knacken jener Passwörter. Aus Sicht des Angreifers lautet die Faustregel: Je mehr Metadaten, desto besser.

2.3.3 Übertragungsproblem

Das Problem mit der Unsicherheit von Passphrasen oder auch Passwörtern beginnt jedes Mal aufs Neue, sobald man ein Passwort eintippt. So ist die Bedrohung nicht mit der Wahl eines mathematisch sicheren Passwortes gebannt. Das Verfahren der Passwortauthentifikation bedient sich prinzipiell einer Zeichenkette, die man in ein Feld eintippt und ist per se dann unsicher sobald einer der Geheimnisträger (Menschen mit Kenntniss über die Parole) kompromittiert bzw. infiziert ist. So gibt es verschiedenste Angriffsvektoren um das Passwort eines Users für einen speziellen Dienst herauszufinden. Von personalisierten (oder auch allgemeinen) Phishing Mails, zu Shoulder Surfing bis hin zu Trojanern und Keyloggern auf dem System Desjenigen, der es zum Zeitpunkt des Angriffs in die Tastatur tippt. Die genannten Angriffsmöglichkeiten können nicht alle remote ausgeführt werden. So benötigt es für das Shoulder Surfing und USB-Sticks die Trojaner beinhalten physischen Zugang zum System oder zumindestens der Person, die das System regulär verwaltet. Ein weiteres großes Problem ist die Übertragung von Passwörtern über die klassische User-Browser-Schnittstelle. Dabei wird das Passwort im Browser des Clients gehasht und dann an den Server übertragen. Die sichere Kommunikation anhand des Hypertext Transfer Protocol Secure (HTTPS) findet erst bei der Übertragung zum Server statt, die Eingabe des Passwortes an den Browser ist ungeschützt. Diese Übertragung von Buchstaben kann mitgelesen werden.

Das Hashen der Zeichenkette kann in einem ungesicherten Netz sicherlich hilfreich sein, da nun ein Angreifer innerhalb des Netzes das Passwort im Klartext nicht lesen kann. Alles was jener Angreifer sieht, ist der Hash, der auf Clientseite erzeugt wurde und im Request steht. Die Nutzung von HTTPS macht es zudem noch schwieriger für Angreifer Kommunikationen zwischen User und Server mitzulesen und in Folge dessen zu manipulieren. Bei sicheren Verbindungen ist mitunter das Hashing der Nutzerdaten demnach theoretisch nicht mehr nötig. In der Praxis kommt es häufiger vor, dass Zertifikate auf dem Browser des Users installiert werden, die vom Angreifer selbst ausgestellt wurden. Der Durchschnittsnutzer achtet nicht darauf, wer das Zertifikat ausgestellt hat und bekommt womöglich nichts von einem Angriff mit. Solche Zertifikate können in Massen durch eine kompromittierte Root-CA (Certificate Authority) erstellt werden. Root-CA's stellen im wesentlichen Zertifikate aus und prüfen die Identität (und Angaben) des Anforderers eines Zertifikats. Dabei kann es Monate oder sogar Jahre dauern bis geschälste Zertifikate von einem Nutzer zufällig erkannt, gemeldet und im Anschluss zurückgezogen werden. Ein solcher Man-in-the-middle (MITM) - Angriff auf SSL-Verbindungen ist sehr typisch und wird im folgenden an einer verschlüsselten Verbindung zu einem Webshop sequenziell erläutert.



1. Im ersten Schritt muss der Angreifer ein eigenes Fake-Zertifikat (im folgenden MITM-Zertifikat genannt) im Zertifikatsspeicher des Nutzers installieren. Dies kann eine Malware oder sonstige Fremdsoftware machen. Hat man als Angreifer physischen Zugriff zum Rechner, kann man dies innerhalb von wenigen Sekunden selbst machen, ohne dass der Nutzer davon etwas mitbekommt. Der MITM - Angriff basiert auf diesem MITM-Zertifikat, das von einem potenziellen Angreifer im Namen einer Webseite fälschlicherweise ausgestellt wurde. Ohne diesen Schritt funktioniert der Angriff nicht.
2. Der User möchte eine Verbindung zum Webshop (im Schaubild `www.shop.com`) aufbauen. Hierzu ruft er die Seite mit dem Präfix `https://` auf im Glauben, eine sichere Verbindung aufbauen zu können, die von keinem Dritten mitgelesen werden kann. Der MITM fängt die Anfrage ab und leitet ihn zu seinem eigenen Webserver weiter.

3. Der Webserver erkennt einen HTTPS - Request und entschlüsselt mit dem eigenen privaten Schlüssel. Dies kann er durch die Installation der Fake-Root-CA in Schritt 1. Dort wurde die Nachricht mit dem öffentlichen Schlüssel des Angreifers verschlüsselt und abgesendet.
4. Simultan baut der MITM-Webserver eine Verbindung zum echten Webshop auf und leitet die Daten des Nutzers jeweils in beide Richtungen weiter. Dabei simuliert er eine gewöhnliche Nutzeranfrage, die der Webshop nicht als fälschlich erkennen kann. Der User hingegen merkt nichts von dem Angreifer, da er die gewohnte (dies ist Standard) Webseite angezeigt bekommt und keine Anomalien erkennt.
5. Der gesamte Traffic, der eigentlich verschlüsselt sein sollte, kann vom MITM mitgelesen, weitergeleitet oder manipuliert werden. Die Verbindung zwischen User und dem Webshop ist nicht sicher trotz HTTPS Anzeige im Browser.
6. Die Response vom Webshop wird mit dem erstellten MITM-Zertifikat für `www.shop.com` verschlüsselt und an den User übertragen.
7. Der Browser markiert die Verbindung als sicher und unbedenklich.

Der Nachteil für den Angreifer in diesem Szenario ist die zeitlich limitierte Möglichkeit eines Angriffs. Es muss also auf den initialen Request des Users gewartet werden, um dessen Daten abzufangen und den Prozess des Angriffs in Gang zu setzen. Wiederholungsangriffe (auch: Replay-Attacks) werden in diesem Szenario nicht verhindert. Der Angreifer muss womöglich demnach nicht einmal das Passwort im Klartext besitzen. Es genügt, den Hash und den Benutzernamen im Request abzufangen um diese dann in einem separaten Aufruf vom eigenen Rechner an die selbe Uniform Resource Locator (URL) zu senden. Es handelt sich bei dieser Art des Angriffs um das Imitieren von Benutzereingaben durch einen Angreifer, bei der der Angreifer die Passphrasen nicht im Klartext kennt.

Durch den Zugang zum Dienst ist es ihm somit (je nach Implementierung des Dienstes) möglich, sensible Daten des Nutzers einzusehen die nicht für den Angreifer bestimmt sind. Die Frage nach der 'Relevanz' von sensiblen Daten sollte obsolet werden, wenn man an die Möglichkeiten denkt, die der Angreifer mit ihnen nun in der Hand hält. Mit diesen könnte er den Nutzer zum Beispiel erpressen um an noch mehr Daten oder Geld des Nutzers zu kommen.

2.4 Zielsetzung



Passwörter können der allumfassenden sicheren Authentifikation nicht mehr gerecht werden. Es braucht weitere oder gänzlich andere Methoden, um die Identität eines Nutzers zu schützen. Die Menschheit benötigt einfache und unkomplizierte Methoden, die ihnen nicht das Gefühl geben allumfängliches technisches Know-How besitzen zu müssen, um die eigenen Daten im Internet zu schützen. Es sollte zum Konsens werden, dass es bequeme Mechanismen für jeden Nutzer gibt. Je nach Gewichtung der Relevanz der Daten in den Kriterien Sicherheit, Bequemlichkeit und Datenschutz gibt es Möglichkeiten dies zu bewerkstelligen.

Dies muss sowohl den Dienstbetreibern als auch dem Endnutzer bewusst werden. Nur die Erkenntnis über ein vorhandenes Problem kann zur Besserung führen und ist der erste Schritt. Nur so kann das Passwortproblem ein für alle Mal gelöst werden.

Ziel dieser Abschlussarbeit ist es eine Möglichkeit der Authentifikation zu spezifizieren, die im Jahre 2020 keine utopischen Szenarien beschreibt, die die Voraussetzungen für eine sichere, bequeme und datenarme Authentifikation erfüllt. Ziel ist es auch, den Leser in die Sicht des Angreifers auf Systeme einzuweisen, sodass im Idealfall automatische Schutzreaktionen wie das Wählen von sicheren Passwörtern hervorgerufen, wenn nicht sogar eine der beschriebenen FIDO2 Verfahren wie der erste oder sogar der zweite Faktor, verwendet werden. Der Prototyp soll die verschiedenen Authentifizierungsmöglichkeiten veranschaulichen und präsentieren, um dem Nutzer die Wahl auf eines der Verfahren zu erleichtern. Gleichzeitig ist eines der Hauptziele dieser Arbeit auch die Grenzen von 'modernen' Authentifizierungsverfahren aufzuzeigen und auf dessen Nachteile hinzuweisen. Inwiefern eine Kombination dieser vorhandenen Verfahren, Probleme löst und welche minderen Probleme dabei entstehen soll ausführlich erläutert werden. Ein weiteres Ziel soll es sein, in gewisser Weise kategorisch darzulegen, welches Verfahren für welchen Nutzertypen geeignet ist. Die Idee dahinter ist, dass es eine klare Trennung in der Nutzung von Accounts zwischen Entwicklern, Unternehmern und dem 'Casual Websurfer' gibt. Was alle drei Arten von Computernutzern gemeinsam haben ist: Sie besitzen persönliche Daten, an die kein Angreifer bzw. eben kein Dritter gelangen soll. Der Schutz dieser Daten sollte jedem Individuum selbst wichtig sein, um in den nächsten 5 bis 10 Jahren auf Besserung zu hoffen. Nicht nur technisch muss die Menschheit mit dem neuen digitalen Zeitalter umgehen und sich absichern können, sondern auch auf die menschliche Komponente muss von jedem selbst geachtet werden.

Es sollte dennoch erwähnt sein, dass diese Arbeit nicht darauf abzielt jede mögliche vorhandene Authentifikationsstrategie zu durchleuchten. Erläutert werden eher das klassische User-ID und Passwort als Authentifikationsmöglichkeit und dem Gegenüber stehn die meistgenutzten Verfahren, die meist eher auf private Schlüssel oder die Kategorien 2 und 3 der Möglichkeiten zur Authentifizierung setzen. So ist der Yubikey am Ende nur eine Möglichkeit zur Umsetzung von einer Challenge-Response-basierten Authentifikation anhand von privaten Schlüsseln welches bereits im FIDO Standard definiert ist. Auch ist es ein Nicht-Ziel dieser Arbeit das Resultat auf eine einzige perfekte Lösung zu dezimieren und diese zum neuen Standard zu erklären. Viel mehr soll es dem Leser durch eine tabellarische Auflistung von Vor- und Nachteilen jeder Methodik selbst überlassen sein, welche Methoden (und Tatempfehlungen dieser Arbeit) er für sich umsetzt.

3 Grundlagen

Laut des IT-Grundschutz-Kompendiums vom BSI könne ein Benutzer aus Bequemlichkeit oder pragmatischen Gründen bewusst auf komplizierte und unhandliche Kryptomodule verzichten und Informationen stattdessen im Klartext übertragen. [A1] Dies stellt ein hohes Sicherheitsrisiko für Unternehmen, aber auch Privatpersonen dar, da Benutzer nicht gewillt sind ihre Passwörter durch komplizierte Verfahren zu erzeugen und in regelmäßigen Abständen vollkommen randomisiert zu setzen. An neuartige Ansätze des Logins in nativen oder webbasierten Anwendungen stellen sich dadurch völlig neue Herausforderungen. So müssen neue Authentifizierungsmöglichkeiten nicht nur sicher sein, sondern auch komfortabel genutzt und bedient werden können, da sie sonst von den Endnutzern gemieden oder umgangen werden. Wichtig ist eben auch, dass die breite Masse Zugriff auf die Ressourcen hat, die es zur Nutzung dieser Verfahren braucht. Man denke nur an die ganzen betrieblichen Passwörter, bei denen zum Monatsende nur eine Zahl an der letzten Stelle des Passwortes geändert wird. Laut einer Statistik von 2019, der “Global Data Risk Report From the Varonis Data Lab”, gaben 38% aller Nutzer an ein Passwort im Unternehmen zu nutzen, das sie nicht (oder nur geringfügig) ändern. Außerdem wird laut dieser Statistik alle 364 Tage wahrscheinlich ein Data Breach aufgrund von unsicheren Passwörtern in einem mittelständigen Unternehmen stattfinden. Die monatliche Ablaufzeit von Passwörtern in Unternehmen scheint also nicht ganz den Effekt zu erzielen, der ursprünglich damit geplant war, da die Arbeiter die Bequemlichkeit über die Sicherheit stellen.

3.1 IT-Grundschutzkriterien

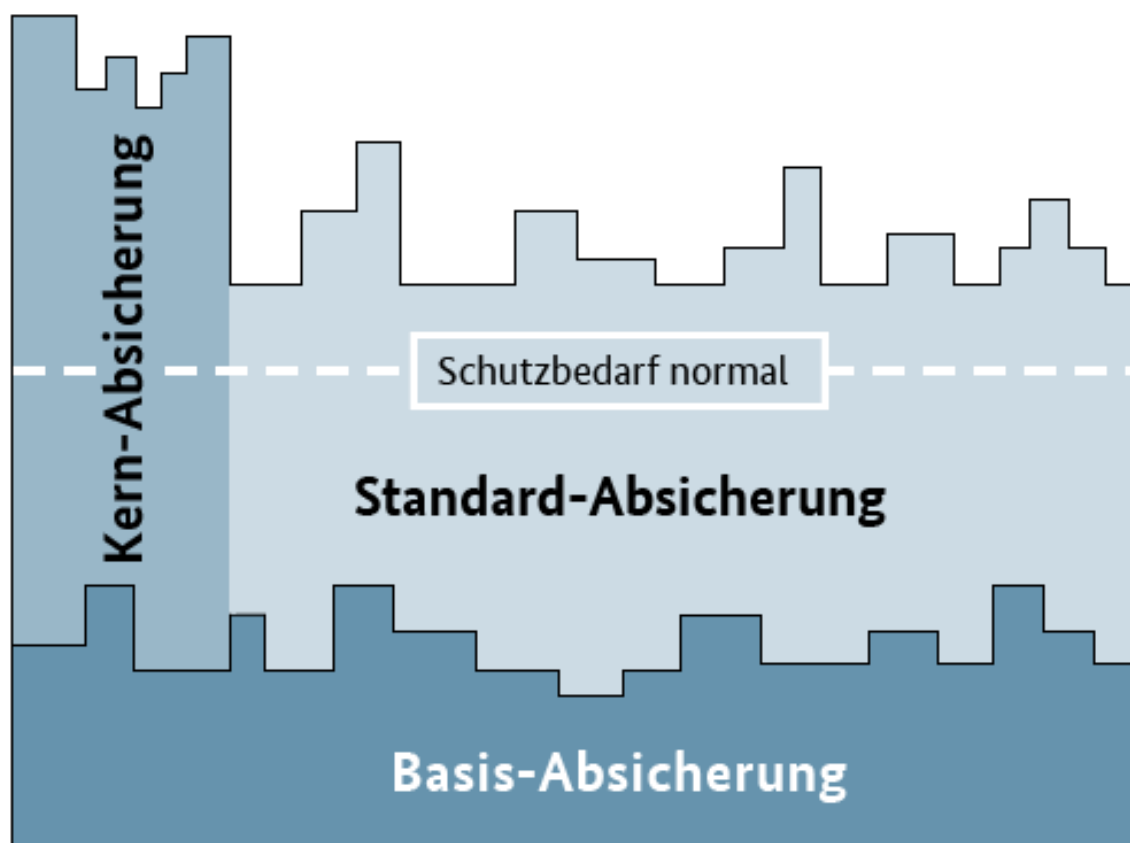
Der IT-Grundschutz definiert den Schutzbedarf eines bestimmten Assets je nachdem welches Risiko bei Verletzung der Grundwerte Vertraulichkeit, Integrität und Verfügbarkeit entstehen [10]. Allgemein existieren folgende Schutzbedarfskategorien:

- **normal** (Schadenauswirkungen begrenzt bis überschaubar)
- **hoch** (Schadenauswirkungen könnten hoch bzw. beträchtlich sein)

- **sehr hoch** (Schadensauswirkungen können ein existenziell bedrohliches Ausmaß annehmen)

Bei dem Authentifikationsprototypen wird die Schadensauswirkung für alle nicht personenbezogenen Daten wohl im Bereich normal liegen, da schon im Aufbau darauf geachtet wird, dass nur so viele Daten vom User verwendet werden, wie zwingend notwendig. (Nach dem Need-To-Know Prinzip) Sollte die Webseite oder Teile der Webseite publiziert bzw. im Business - Umfeld genutzt werden, muss eine Neubewertung der Daten nach Vertraulichkeit, Integrität und Verfügbarkeit stattfinden. Vor allem muss darauf geachtet werden, dass Daten über Fingerabdrücke verschlüsselt und Passwörter im gehashten Zustand in Datenbanken persistiert sind. Bei Kompromittierung des Hauptrechners, welches die größte Bedrohung in diesem Szenario darstellen würde, droht ein Data Breach mit dem Angreifer diese Daten weiterverwenden können. Durch Verschlüsselungen durch Schlüssel, die nicht auf dem Hauptsystem (oben u.a als Hauptrechner benannt) liegen. Gleichzeitig sollten die genutzten Verfahren insgesamt mathematisch sicher sein, auf veraltete Verschlüsselungsverfahren ist zu verzichten. Dies gilt auch für Hashes wie MD5, die mittlerweile relativ akkurat durch sehr große vorgerechnete Tabellen erraten werden können.

Im Falle einer Kompromittierung besäße der Schutzbedarf der Daten innerhalb der Datenbank, welche nicht gehasht oder anderweitig verschlüsselt sind, die Kategorie 'sehr hoch'. Eine Kompromittierung kann für das Unternehmen einen Imageschaden sowie weitreichende juristische Klagen zur Folge haben. Damit sind bereits 2 der 7 aufgeführten Schadensszenarien durch den BSI beschrieben. Je nach dem wie kompliziert die Ursprungsdaten sind und welcher Hashingalgorithmus in Kombination mit Salt und Pepper genutzt wurde, können vorallem Nutzerpassphrasen an die Öffentlichkeit gelangen und Angreifer können die Passwörter für andere Dienste nutzen. Die Wahrscheinlichkeit für diesen Schaden ist noch vergleichbar niedrig, weshalb die Kategorie nach dem BSI für Passwörter in dem Prototyp hoch statt sehr hoch ist.



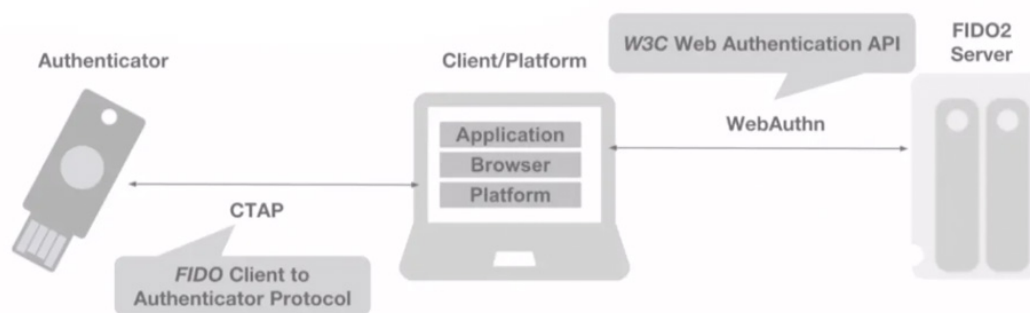
Außerdem unterscheidet der IT-Grundschutz drei Arten der Absicherung. Die Basis-Absicherung ist relevant für Institutionen die einen Einstieg in den IT-Grundschutz suchen und relativ schnell alle relevanten Geschäftsprozesse mit einfach umzusetzenden Basismaßnahmen sichern wollen. Die Kern-Absicherung konzentriert sich auf besonders wichtige Geschäftsprozesse und vertieft sich in die Sicherung dieser. Von einer Standard-Absicherung spricht man, wenn alle empfohlenen IT-Grundschutz-Vorgehensweisen durchgeführt werden. Sie beschreibt den allumfassenden Schutz der Prozesse und Bereiche der Institution, wie das Schaubild vom BSI verdeutlicht. [9] Bei dem Prototyp wird eine Basis-Absicherung nach BSI durchgeführt und darauf geachtet alle Datenschutzkriterien zu erfüllen. Dabei wird vor allem die Checkliste des IT-Grundschutzes zu Webservern und Webanwendungen betrachtet. Kriterien die nicht erfüllt werden konnten oder wurden, werden dokumentiert und im Fazit erläutert. Die Wahl der Basis-Absicherung begründet sich damit, dass lokale Testdaten im Prototyp verarbeitet werden, für die kein bis nur ein sehr geringer Schutzbedarf besteht. Eine Kern-Absicherung käme nur in Frage, falls ein ganz bestimmter Prozess oder Asset des Prototyps geschützt werden müsste wie zum Beispiel der Zugriff auf die Datenbank durch den Prototypen, welche nicht der Fall ist. Insgesamt ist dieser Teil der Abschlussarbeit auch als Einstieg in den IT-Grundschutz zu verstehen,

welcher laut BSI selbst die Basis-Absicherung als Empfehlung und zur Folge hat und sich am Besten für vorhandene Zwecke eignet.

3.2 Standard nach FIDO2

Die genannten Probleme sind den Menschen seit einigen Jahren bekannt und wurden vor allem mit Verfahren gelöst, die keine Passwörter (oder allgemein Verfahren der Wissenskategorie) zur Authentifikation benötigen und diese maximal als ersten Layer der Sicherheit implementieren. Das bekannteste Beispiel für einen Standard zur sicheren und bequemen Authentifikation im Internet findet man unter dem Schlüsselwort FIDO2. FIDO steht dabei für 'Fast Identity Online' (Schnelle Identität im Netz). Sie ist das Ergebnis einer Kooperation des World Wide Web Consortium (W3C) und der FIDO Alliance.

Was ist FIDO Authentication?



Grundsätzlich besteht der Standard aus folgenden Komponenten:

- **Authenticator**

Der Authenticator kann, wie im Schaubild von der offiziellen Seite der FIDO Alliance zur Frage 'Was ist die FIDO Authentifikation' bereits gezeigt, ein sogenannter "FIDO2-USB" sein. Also ein USB mit der Möglichkeit zur passwortlosen Authentifizierung des Nutzers im Netz. Eine weitere bekannte Möglichkeit für einen Authenticator ist Windows Hello (bestehend aus Hello-PIN, Hello-Fingerabdruck und Hello-Gesichtsmuster) oder Plat für Betriebssysteme vom Hersteller Apple.

So können also sowohl externe Geräte (mit eingebautem Schlüssel) sowie interne Geräte (sogenannte self-signed-devices, die selbst Schlüssel bei der Registration erzeugen) des Betriebssystems bzw. Rechners zur Authentifikation genutzt werden.

- **CTAP - Client to Authenticator Protocol**

Das Protokoll CTAP bzw. etwas genauer CTAP2 ist das Nachfolgerprotokoll zu U2F welches mit der Einführung von FIDO2 in CTAP1 umbenannt wurde. Mit FIDO2 wurde die Kommunikation zwischen dem Authenticator und der Plattform (dem Browser, der Applikation) standardisiert. Grundsätzlich definiert das Protokoll, dass der Server eine initiale Anfrage an den User sendet. Dieser erstellt mittels des Authenticators ein Key-Pair, also einen öffentlichen und einen privaten Schlüssel und übermittelt den öffentlichen Schlüssel an den Server. Der Server persistiert diesen um künftige Anfragen des Nutzers zu identifizieren und den Nutzer zu autorisieren. Viele Schlüssel erfordern vor diesem Prozess eine Userverifikation, die zum externen Authenticator (dem USB - Stick) noch eine PIN verlangt, die beim Initiieren gesetzt wurde.

- **Client/Plattform**

Zwischen dem FIDO2-Server und dem Authenticator befindet sich der Client, hier interagiert der Nutzer mit dem Authenticator und verifiziert sich ihm gegenüber. Über dessen Browser wird dann (über eine durch Webauthn erzwungene sichere Verbindung) ein Request an den FIDO2 - Server gestellt. Über den Client kommunizieren Server und Authenticator. Wichtig für alle diese Vorgänge ist, dass der private Schlüssel bei keiner Anfrage den Client verlässt. Er ist stets sicher auf dessen Rechner (im Trust Store unter Windows, je nach Betriebssystem variierend) lokalisiert.

- **Webauthn**

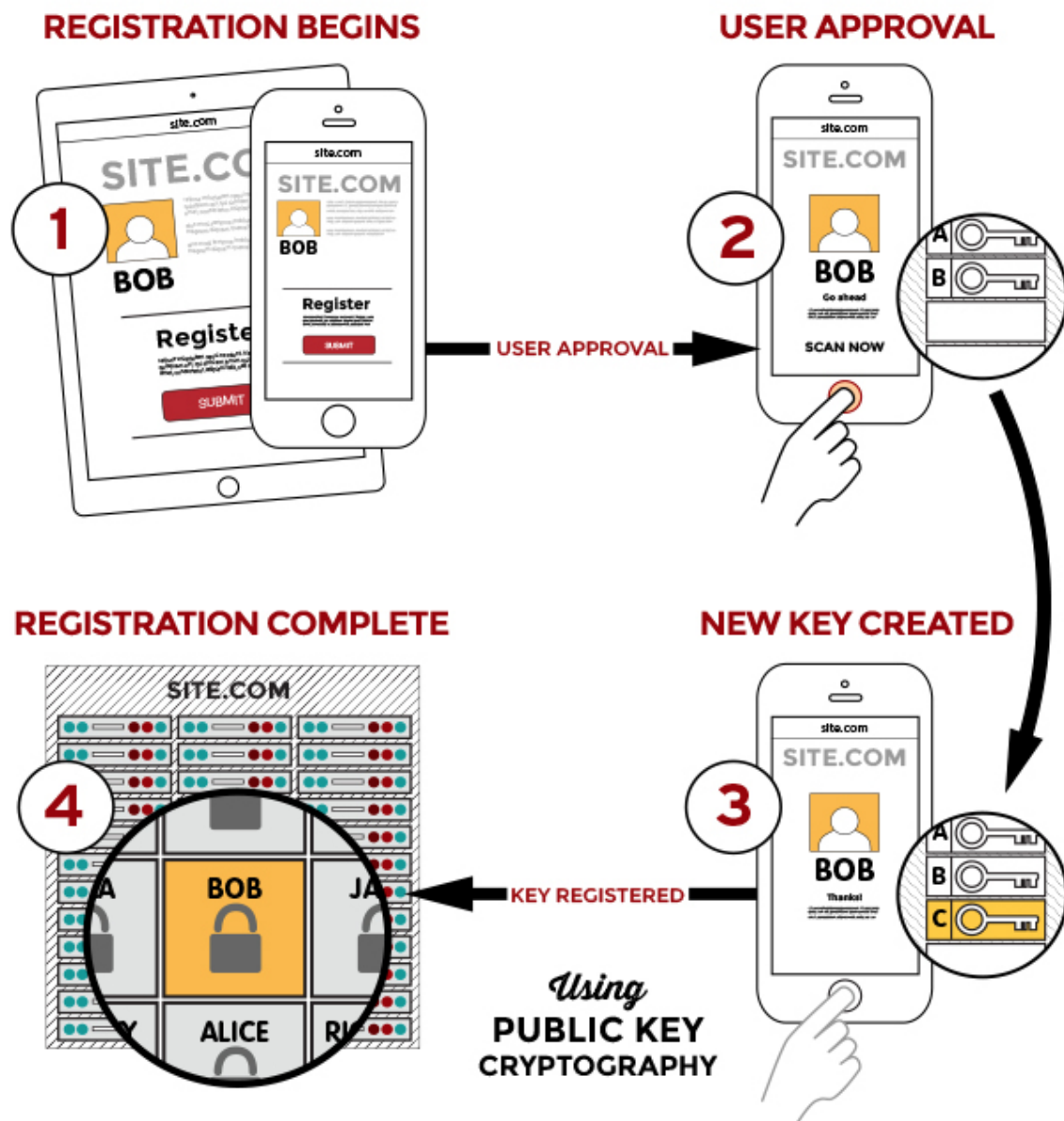
Zwischen dem Server und der Plattform befindet sich das WebAuthn - Protokoll, welches in folgenden Kapiteln näher erläutert wird. Das Protokoll basiert auf zwei Hauptbestandteilen: Einem Registrier und einem künftigen Loginvorgang. Das Protokoll vermittelt dem Authenticator unter anderem die Optionen, die der Server für die Authentifikation erlaubt. So kann der Server zum Beispiel externe USB-Sticks verbieten oder eine ausdrückliche Authentifikation mittels NFC erzwingen. Wichtig ist, dass Webauthn keinerlei Details über den User freigibt. Ferner weiß weder der Server noch das Protokoll Webauthn welche der eigentlichen Authentifikationsmethoden vom User genutzt wurden. Webauthn

gibt im ersten Schritt dem Nutzer die Optionen bei der Registration und kann im folgenden nur anhand des publicKey's und der credentialID erkennen, dass das Selbe gerät sich autorisieren möchte. Der Nutzer hinter dem Gerät ist dem Server nicht bekannt bzw. keinerlei privaten Daten des Nutzers. Die meisten FIDO2 - Authentifizierungsmechanismen kommen um einen Usernamen dennoch nicht herum um den User über die Browsersession hinaus zu erkennen.

- **FIDO2 - Server**

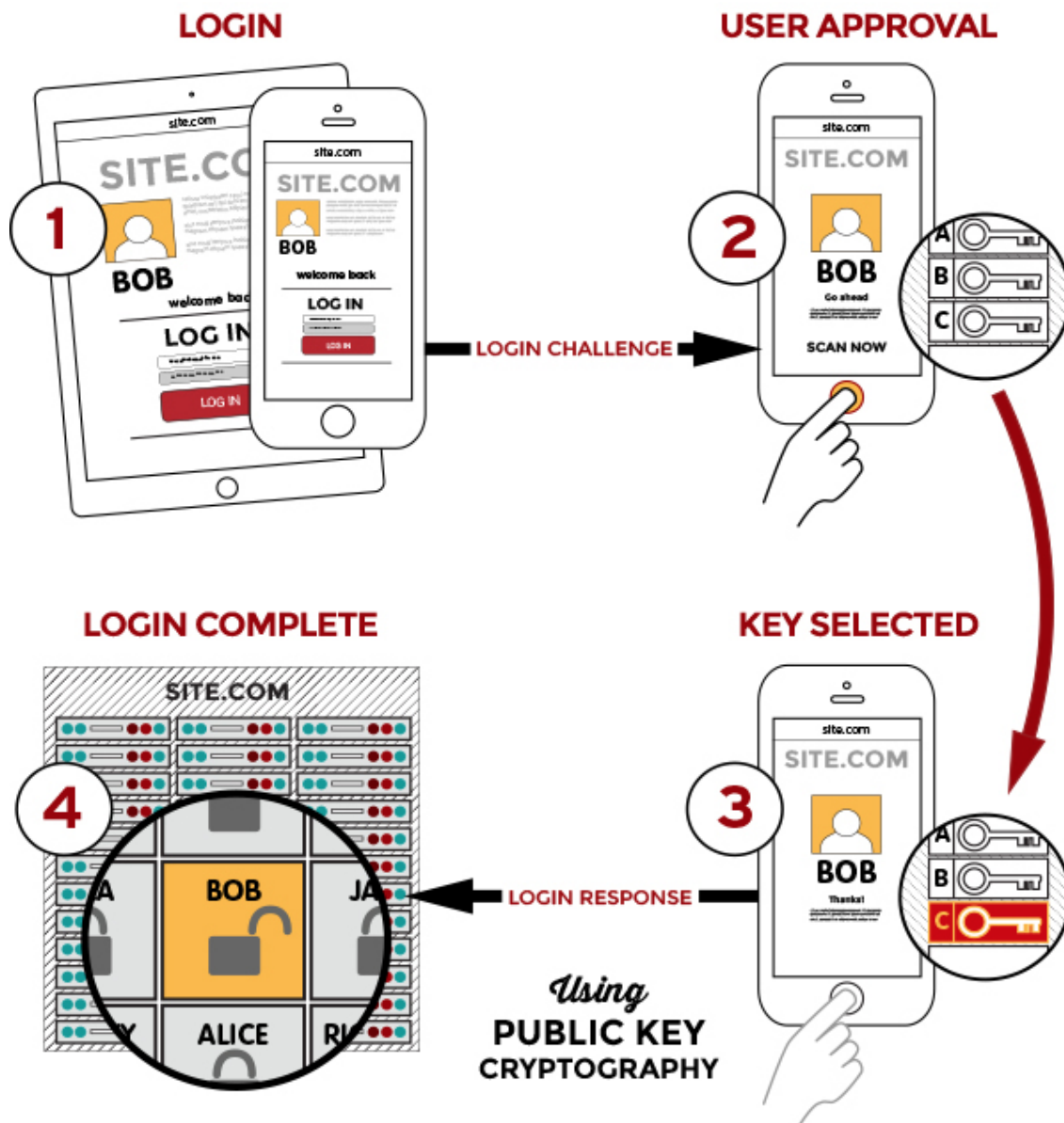
Ein FIDO2-Server ist ein durchschnittlicher Server auf dem typischerweise ein Javascript-Framework läuft, das als Backend für die Anfragen fungiert. Der Server verwaltet die öffentlichen Schlüssel des Nutzers und sorgt für die Verifikation der Anfragen für den Registrierungs und Loginprozess. Ferner wird bei sämtlichen Serverprozessen eine 'Relying Party Id' angegeben, welches die IP-Adresse des Servers ist. Teilweise findet man den Server in Abbildungen der FIDO - Alliance auch als 'Relying Party Server'. Erwähnt werden sollte allerdings, dass Client und Server sich die Authentifikation des Users bzw. die Logik hierzu teilen. Es findet im Vorhinein meist eine Userverifikation statt, bevor der Server über einen Loginversuch Kenntnis erlangt. Erst im Anschluss werden verschiedene Challenge-Response Anfragen zwischen Server und Client ausgetauscht.

Der FIDO Standard definiert zwei wichtige Prozesse: Die einmalige Registrierung und die folgend mehrmalige Authentifizierung ('den Login') des Nutzers.



Bei der Userregistrierung sieht der Standard vor, dass im ersten Schritt der User dazu aufgefordert wird einen vom Server akzeptierten Authenticator auszuwählen. Im Beispielfeld der FIDO Alliance entscheidet sich der Nutzer für den Fingerabdruck als Methode. Auch möglich gewesen wäre ein gesprochenes Wort ins Mikrofon des Nutzers oder ein PIN sowie vieles mehr. Je nach Gerät (und Betriebssystem / Browser) können die möglichen Authentifizierungsmechanismen variieren. Im zweiten Schritt entsperrt der Nutzer den FIDO Authenticator, dieser Schritt ist die Userverifikati-

on. In folge dessen wird im dritten Schritt ein Paar aus privatem und öffentlichen Schlüssel erstellt. Der private Schlüssel bleibt auf dem Gerät des Endnutzers, während der öffentliche Schlüssel an den Server gesendet wird. Dort wird dieser mit dem Useraccount verknüpft und persistiert um folgende Loginanfragen des Nutzers authentifizieren zu können. Der initiale Nutzer, der die Registrierung angestoßen hat, kann damit erkannt und effektiv authentisiert werden. Die Registrierung ist damit abgeschlossen. Erklärt wird dieser Vorgang in Stichpunkten auf der Webseite der FIDO - Alliance. [15]



Bei dem Loginvorgang muss der User den Login - Button drücken und wird in folge dessen dazu aufgefordert eine sogenannte Challenge zu lösen. Dies Auswahlmöglichkeiten begrenzen sich hierbei auf Diejenige, die bei der Registrierung angegeben wurden. Wenn der User also sowohl die PIN als auch die Gesichtserkennung eingerichtet hat, kann er sich mit beiden Varianten authentifizieren. Der zweite Schritt ist identisch zur Registration, der User entsperrt den FIDO - Authenticator. In folge dessen wir in Schritt 3 der öffentliche Schlüssel des Nutzers verwendet um den die Challenge des Servers zu signieren (mit dem privaten Schlüssel) um diesen dem Server zurückzusenden. Der Server entschlüsselt diese Challenge mit dem gespeicherten öffentlichen Schlüssel aus der Registration. Der Nutzer erhält zum Ende

der Authentifizierung eine zugehörige Response, die ihm den Login ermöglicht. Die Authentifikation des Nutzers ist damit abgeschlossen.

Zusammengefasst basiert FIDO2 auf vorhandenen Protokollen wie Web Authentication (WebAuthn) für die Browser-Server-Kommunikation und CTAP für die Browser-Authenticator-Kommunikation. Die Yubico, einem der Hauptentwickler des heutigen CTAP1 - Protokolls, fasst FIDO2 wie folgt zusammen: "an open authentication standard that enables internet users to securely access any number of online services with one single security key [...]. FIDO2 is the latest generation of the Universal second factor (U2F) protocol" [5] (übersetzt: Ein offener Authentifikationsstandard der Usern das Erreichen von Online-Services mittels eines einzigen sicheren Schlüssels erlaubt. [...] FIDO2 ist die letzte Generation des U2F - Protokolls). Während das Vorgänger - Protokoll U2F von Google und Yubico ins Leben gerufen wurde, ist FIDO2 ein offener dezentraler Kommunikationsstandart für die passwortlose Kommunikation welches die Authentifizierung für sowohl Privatanutzer als auch Unternehmen bequem und gleichzeitig sicher machen soll.

3.3 Behandelte Verfahren

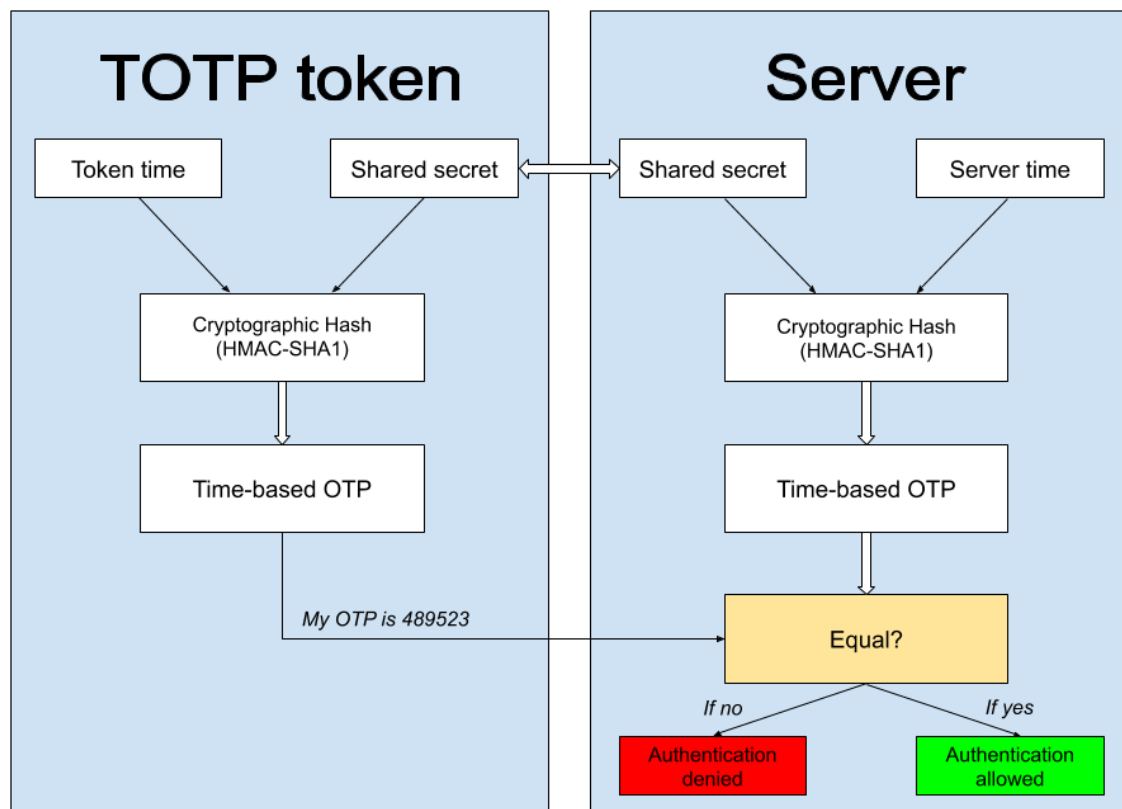
3.3.1 Username & Passwort

Trotz der bereits erwähnten Probleme des Passwortes, ist es laut eines wissenschaftlichen Artikels von Thomas Maus 2008 “nicht aus unserer Arbeitswelt [...] wegzudenken” [12]. Anzumerken ist, dass der Artikel bereits 12 Jahre in der Vergangenheit liegt und immer noch Relevanz hat. Herr Maus beschreibt alternative Authentifikationsmethoden wie die Einmalkennwörter und biologische Merkmale. Dies ist nur ein weiterer Beweis dafür, dass schon vor mehr als 10 Jahren die Passwortproblematik erkannt wurde. In dem Artikel geht Herr Maus der Hypothese nach, ob Passwörter wirklich per se unsicherer sind als die Authentifikationsmethoden der beiden anderen Kategorien Besitz und biologische Merkmale. So sei das Kernproblem des Passwortes, dass es direkt nach der Eingabe ein geteiltes Geheimnis ist, da alle beteiligten Systeme es mitschneiden konnten und automatisch Geheimnissträger sind. Das Passwort bietet sehr viele Angriffsvektoren, so “Shoulder Surfing, Phishing, Social-Engineering, Man-in-the-Middle Angriffe, schlechte Passwort Qualitäten, das Teilen des Passwortes mit Familie und Freunden” [12] und vielem mehr.

Bei dem Prototyp wird der Ist-Zustand einer Username und Passwort - Authentifikation demonstriert, wie man sie heutzutage auf höchstwahrscheinlich jedem Webdienst in der Form als ersten schwachen Faktor vorfinden wird. Teilweise wird sie sogar als einziger Authentifizierungsfaktor verwendet. Während man über alle Schwächen des Passwortes spricht, muss man dennoch anerkennen, dass das Passwort eine gewisse Flexibilität bietet. Es genügt das Wissen über eine bestimmte Zeichenfolge um sich zu authentifizieren, dieses Wissen muss nicht zwangsmäßig auf ein Blatt geschrieben oder in einem Passwort-Manager beherrbergt werden. Das beste Passwort ist jenes, welches nur in dem Gehirn des Nutzers persistiert ist und mit niemandem geteilt wird. Anders als bei neueren Verfahren, die als sicherer gelten, benötigt es keine Smart-Card, USB-Sticks oder anderweitige technische Hilfsmittel um sich zu authentifizieren. Lediglich das Gedächtnis des Nutzers ist gefragt, welches allerdings fortlaufend im digitalen Zeitalter von Social Media und Up-To-Date Informationen, nachlässt. So ist es Usern heutzutage immer schwerer möglich sich die verschiedenen Passphrasen zu merken, ohne sie als Behelf irgendwo zu notieren. Sobald das Passwort auf eine potenziell unsichere Plattform notiert wurde, gilt sie als geteiltes Geheimnis zwischen jedem Leser und dem Initialbenutzer dessen.

3.3.2 OTP's

Ein Kennwort ist eine "eine Zeichenfolge, die zur Authentifizierung verwendet wird. Damit soll die Identität einer Person [...] auf eine Ressource nachgewiesen werden." [A4]. Ein Einmalkennwort im Vergleich ist ein Kennwort, das nur ein einziges Mal für eine Authentifizierung genutzt werden kann. Man unterscheidet drei Arten von One time password (OTP)'s.



Bei der timerbasierten (Time based one time password (TOTP)) Methode wird die aktuelle Systemzeit (Token time) mit dem zu verschlüsselnden Text (Shared secret) anhand eines kryptografischen Verfahrens verschlüsselt. Es entsteht ein kryptografischer Hashwert (Cryptographic Hash), welcher meist den HMAC-SHA1 Hashingalgorithmus verwendet. Dieses Verfahren findet gleichermaßen auf dem Server statt. Der einzige Unterschied zum Server besteht darin, dass die Systemzeit des Servers genutzt wird. Findet innerhalb der festgelegten Zeit (ein Token entfällt laut RFC6238 standartmäßig nach 30 Sekunden, diese Zeit ist modifizierbar) eine erfolgreicher Match auf Serverseite statt, wird der User authentifiziert und ihm der Zugang gewährt. [13] Ein entscheidender Nachteil dieser Methode entsteht, falls der authentifizierende User innerhalb dieser 30 Sekunden (beispielsweise) durch

eine Störung des Netzes oder einen kompletten Netzausfall die Verbindung verliert. Der aktuelle TOTP Code wird ungültig und muss erneut angefragt bzw. erstellt werden. Apps wie der Google Authenticator lösen dieses Problem, in dem sie einen Timer setzen und alle 30 Sekunden einen automatisch generierten neuen Code mit der aktuellen ablaufenden Zeit des Timers anzeigen. Dabei muss natürlich drauf geachtet werden, dass Server und Client synchron sind, dieses Verfahren ist deshalb an eine bestimmte Zeitspanne und ein externes Gerät (sei es Smartphone, TOTP Smart Card oder ein externer Rechner) gebunden. Diese Zeitbegrenzung bzw. das Verlassen auf die Zeit des Gerätes und des Servers kann zum Nachteil werden wenn Geräte asynchron werden. So gibt es einige ältere TOTP - Generatoren, die einen internen Zähler haben und mit jedem Tick eine Sekunde auf den aktuellen Wert nach UTC draufrechnen. Findet bei Gerätestart keine erneute Synchronisation (im RFC6238, Resynchronisation) statt, so entstehen fehlerhafte TOTP - Codes auf Clientseite, die der Server ablehnen wird. Dieser Nachteil ist gleichzeitig allerdings ein großer Vorteil in punkto Sicherheit, da ein Angreifer potenziell einen sehr kleinen zeitlich begrenzten Angriffsvektor besitzt, in dem er den TOTP - Code lösen muss. Durch entsprechende Zugriffssicherheitsfeatures, wie das Sperren des Nutzers nach oftmalig wiederholter falscher Eingabe, wird ein Brute-force-Angriff verhindert.

Ereignisbasierte OTPs (folgend HMAC based one time password (HOTP)) besitzen einen Ereigniszähler, der bei jeder versuchten Authentifizierung einen Zähler auf Server und Clientseite synchronisiert inkrementiert. Sollte der Zähler asynchron werden bzw. der Server einen anderen Wert gespeichert haben als der Client bei der nächsten Authentifizierung sendet, wird der Authentifizierungsvorgang abgebrochen. Man findet diese Funktionalität wortwörtlich beschrieben in Googles Time and event based one time password - Patent [A6] in folgendem Wortlaut “[...] the characteristics of an event can be the value of a counter that is incremented each time the user pushes a button on the token” [A6]. Für diesen Prozess wird ein spezieller Algorithmus genutzt, der im RFC4226 näher beschrieben ist.

Challenge-response basierte OTP Verfahren bedienen sich an komplizierten mathematischen Verfahren. Das heißt, es erfolgt ein ACK (Acknowledge bzw. Initialanstoß zur Authentifizierung). Der Client, berechnet die Response mithilfe der mathematischen Formel und sendet das Ergebnis an den Server. Sollte es einen Match geben, erhält der Client eine Response vom Server, der seine Echtheit bestätigt. Synchronisationsprobleme kann es bei diesem Verfahren entgegen der ereignis- oder timerbasierten OTP-Verfahren nicht geben, da die Berechnung dieses 'Schlüssels' vollkommen auf der Clientseite funktioniert. Der Server überprüft diese Rechnung nur mit seinem eigenen Wert, stellt aber keine weiteren Rechnungen oder Umformungen mit diesem Wert an. Der Hauptvorteil dieses Verfahrens ist, dass unabhängig von der Zeit und einem speziellen Ereignis eine Anfrage gestellt werden kann. Der Server kann also seine

'Challenge' abschicken und muss keine 'Response' innerhalb einer festgegebenen Zeit erhalten, um authentifizieren zu können. Dieses Verfahren gilt als besonders sicher, da es auf Serverseite keinen Algorithmus gibt, der sich vorausberechnen lässt.

Die folgende Tabelle soll vergleichend die Gemeinsamkeiten, aber auch Vor und Nachteile von HOTP mit TOTP verdeutlichen. Das Wissen zu vor allem den Nachteilen der Verfahren ist wichtig um die gewählte Eigenstrategie im Prototypen zu verstehen.






Relies on Shared Key and Counter (moving factor), Counter values are generated based on the shared key	Relies on Share key and Counter but counter values changes with the function of time (every 30 or 60 sec)
Event-based OTP algorithm is used i.e. the moving factor is an event counter	Time-based OTP algorithm is used i.e. short-lived OTP values for enhanced security are used
The counter is subsequently incremented whenever a new OTP is generated	The counter values are incremented with a timeout of an OTP
Pass Token is valid for an unknown amount of time	Passcodes are valid only for short duration of time (can also be called as Extension of HOTP that involved time)
Requires more maintenance but no synchronization between user device and server	Requires less maintenance but high-level synchronization because of the factor of time
Published as IETF RFC 4226	Published as RFC 6238
An example can be Yubikey, Google Authenticator	Example Google Authenticator
Less Expensive as the counter is cheaper to implement for authentication purpose	More Expensive because accurate time devices are required for authentication






3.3.3 WebAuthn





Webauthn ist kurz für die 'Web Authentication'. Zur Verfügung gestellt wurde dieser Standard der Authentifikation 2018 von der FIDO Alliance und dem W3C und ist vollumfänglich im FIDO2 Standard definiert. [A13] Sie ermöglicht eine passwortlose Authentifikation durch die Nutzung des vorhandenen public-key-Verfahrens für Webseiten.

FIDO Platform/Browser Support

Updated 6/29/2020

U2F API		WebAuthn API		U2F API		WebAuthn API		U2F API		WebAuthn API		U2F API		WebAuthn API	
	Chrome/Windows		Edge/Windows		Firefox/Windows		Safari/iOS								
U2F		CTAP2		U2F		CTAP2		U2F		CTAP2		U2F		CTAP2	
USB	NFC	BLE	USB	NFC	BLE	BLE	Hello	USB	NFC	BLE	BLE	USB	NFC	BLE	Plat

U2F API		WebAuthn API		U2F API		WebAuthn API		U2F API		WebAuthn API		U2F API		WebAuthn API	
	Chrome/Android		Edge/Android		Firefox/Android		Safari/macOS								
U2F		CTAP2		U2F		CTAP2		U2F		CTAP2		U2F		CTAP2	
USB	NFC	BLE	USB	NFC	BLE	BLE	Plat	USB	NFC	BLE	BLE	USB	NFC	BLE	Plat

U2F API		WebAuthn API		U2F API		WebAuthn API		U2F API		WebAuthn API		U2F API		WebAuthn API	
	Chrome/macOS		Edge/macOS		Firefox/macOS										
U2F		CTAP2		U2F		CTAP2		U2F		CTAP2		U2F		CTAP2	
USB	NFC	BLE	USB	NFC	BLE	BLE	Plat	USB	NFC	BLE	BLE	USB	NFC	BLE	Plat

Implemented / Stable

In Development

Not Supported / No ETA

Die Abbildung von der FIDO - Alliance zeigt die Unterstützung von den FIDO - Protokollen, welche im vorangegangenen Kapitel näher erläutert wurden. So besitzt zwar jeder Browser unter jedem Gerät die Unterstützung für die Webauthn API, allerdings unterstützt zum Beispiel der Browser 'Firefox' unter macOS das CTAP2 Protokoll nicht, welches somit die Unterstützung der Webauthn API obsolet macht. Es sollte erwähnt werden, dass in der Abbildung ein gedanklicher Trennstrich zwischen den Beiden Paaren U2F-API und U2F sowie WebAuthn API und CTAP2 fehlt. Sie unterstützen sich nur als Paar gegenseitig allerdings nicht untereinander. Gleichzeitig bedeutet die Unterstützung der CTAP2 Schnittstelle nicht automatisch, dass auch jede Variante der Authentifizierung möglich ist. So wird unter Windows unter den drei Browsern Google Chrome, Microsoft Edge und Firefox jede der vier Varianten unterstützt. Unter Android fehlt für alle drei Browser die Unterstützung von CTAP2. Unter macOS wiederum wird CTAP2 im Safari-Browser unterstützt, doch NFC und BLE zum Stand des Bildes (26.09.2020) nicht. Die Abbildung verdeutlicht, dass Webauthn zwar auf Geräten mit dem Windows - Betriebssystem vollumfänglich unterstützt werden, allerdings noch nicht zum allgemein implementierten Standard für andere Betriebssysteme geworden sind. So wie es ECMAScript6 (bzw. heutiges

Javascript) oder Flash in jeden Browser geschafft hat. Dies ist ein Prozess und wird wohl noch einige Jahre andauern, bis alle Browser die CTAP2 Schnittstelle unterstützen und damit eine Webauthn - Authentifizierung ermöglichen.

Der Standard definiert verschiedene Arten von Authentifikationsmöglichkeiten. Neben sogenannten “plattform authenticators“ [13] also jenen Verfahren, die die geräteeigenen Sicherheitsfeatures wie den Fingerabdrucksensor oder der Gesichtserkennung durch eine Kamera nutzen, definiert der Standard auch die Authentifizierung durch externe Geräte wie USB-Sticks, auf denen sich der private Schlüssel des Nutzers (in gesicherter Form) befindet. Grundsätzlich definiert WebAuthn nur Dinge, die man im nativen Bereich bereits kennt, adaptiert diese allerdings für den Webbereich. Dabei basiert Webauthn auf vielen bereits vorhandenen Abhängigkeiten der Informatik wie die Standards von HTML5, ECMAScript, COSE (CBOR Object Signing and Encryption COSE, definiert im RFC8152) oder dem Nutzen von der Base64url encoding.

Laut dem W3C sei die grundsätzliche Idee hinter diesem Verfahren, dass die Daten des Users zu keinem Zeitpunkt den Rechner verlassen müssen. Die Daten werden vollumfänglich vom jeweiligen Authenticator gemanaged, welches dann mit einer sogenannten 'Relying Party' interagiert um den Nutzer zu authentifizieren. Eine Relying Party ist grob vereinfacht eine Entität, welche die Web Authentication API nutzt um den Nutzer zunächst einmal zu registrieren und in Zukunft zu authentifizieren. [13]

Auf einer von Codegram organisierten Rede von Suby Raman, einem der Mitentwickler des Web-Authentication-Protokolls und dem Hauptentwickler für Windows Hello, erläutert Raman die Probleme von Passwörtern, die im Rahmen dieser Arbeit im Kapitel der Einleitung bereits adressiert worden, und wie WebAuthn diese löst.

1. Passwörter sind geteilte Geheimnisse.

Der ausgetauschte öffentliche Schlüssel ist kein Geheimnis, der private Schlüssel bleibt stets auf dem Gerät des Nutzers und verlässt diesen zu keinem Zeitpunkt der Authentifikation.

2. Sichere Passwörter sind schwer zu finden und wenn, dann sind sie nicht leicht merkbar.

Der Authenticator auf den Geräten des Nutzers erstellt zufällige und sichere 'credentials' für die Authentifikation. Es liegt nicht mehr in der Verantwortung des Nutzers dies zu tun.

3. Passwörter können leicht entwendet werden.

Sichere interne (oder auch externe) Hardware macht es Angreifern sehr schwer, wenn nicht schon fast mathematisch unmöglich, durch physische Angriffe an die privaten Schlüssel zu kommen. Zum Verfassungszeitpunkt dieser Arbeit sind keine Angriffe auf diese Geräte bekannt, vereinzelt gibt es dennoch Angriffe auf die Software der Hersteller, die sich auf den Geräten vorinstalliert befindet.

4. Passwörter haben ein Bequemlichkeitsproblem und sorgen automatisch für die unsichere Wahl dessen.

Die 'credentials' des Nutzers sind gebunden an den aktuellen Scope des Nutzers. Das heißt, dass jede Anfrage einzigartig ist. Dies gilt sowohl für die Registrierung als auch den späteren Login mithilfe der Web Authentication API. Der erwähnte Wiederholungsangriff im Kapitel für Unsichere Passwörter wird damit verhindert. Ein mehrmaliges Nutzen der erstellten zufälligen Daten durch den Authenticator ist damit nicht möglich und nur für eine Anfrage valide.

5. Aus Sicht der Entwickler sind Passwörter schwer zu sichern.

Da der öffentliche Schlüssel kein Geheimnis darstellt, hat dieser auch keinen besonderen Wert und kann problemlos im Klartext in Datenbanken persistiert werden.

Zusammengefasst hat der FIDO2 Standard mit CTAP (dem Protokoll für externe Authentifikationen mit Mobilgeräten) und Webauthn (der Schnittstelle bzw. "API") vorhandene Funktionen definiert, mit der native Authentifizierungsmethoden wie das public-private Key Verfahren auf die Webseite übertragen werden können.

4 Konzeption

4.1 Auswahl der Authentifizierungsverfahren

Bei der Auswahl der Authentifizierungsverfahren musste vor allem berücksichtigt werden, inwiefern das Verfahren im Rahmen dieser Arbeit umsetzbar ist. Dies bezieht sich einerseits auf die Zeit, die es für die Implementation benötigt und andererseits auch auf die monetäre Machbarkeit. Es existieren für die Webauthentikation zum Beispiel ganz spezielle USB-Sticks von angesehenen Marken, die allerdings einige hunderte Euro kosten. Die Features die sie im Gegensatz zu einem ganz normalen USB-Stick (FIDO USB) liefern, beziehen sich dabei nur auf den Teil der Userverifikation. So besitzen einige USB-Sticks einen eingebauten Fingerabdrucksensor, ein Tastenfeld oder sogar zusätzliche Sicherheitssoftware auf den Massendatenträgern. Innerhalb dieser Arbeit sollte es allerdings um den Einsatz eines solchen Verfahrens für den Durchschnittsnutzer innerhalb von Unternehmen oder einen Casual Surfer gehen. Eine teure Anschaffung für die eigene Sicherheit erscheint dem Durchschnittsnutzer als nicht nötig.

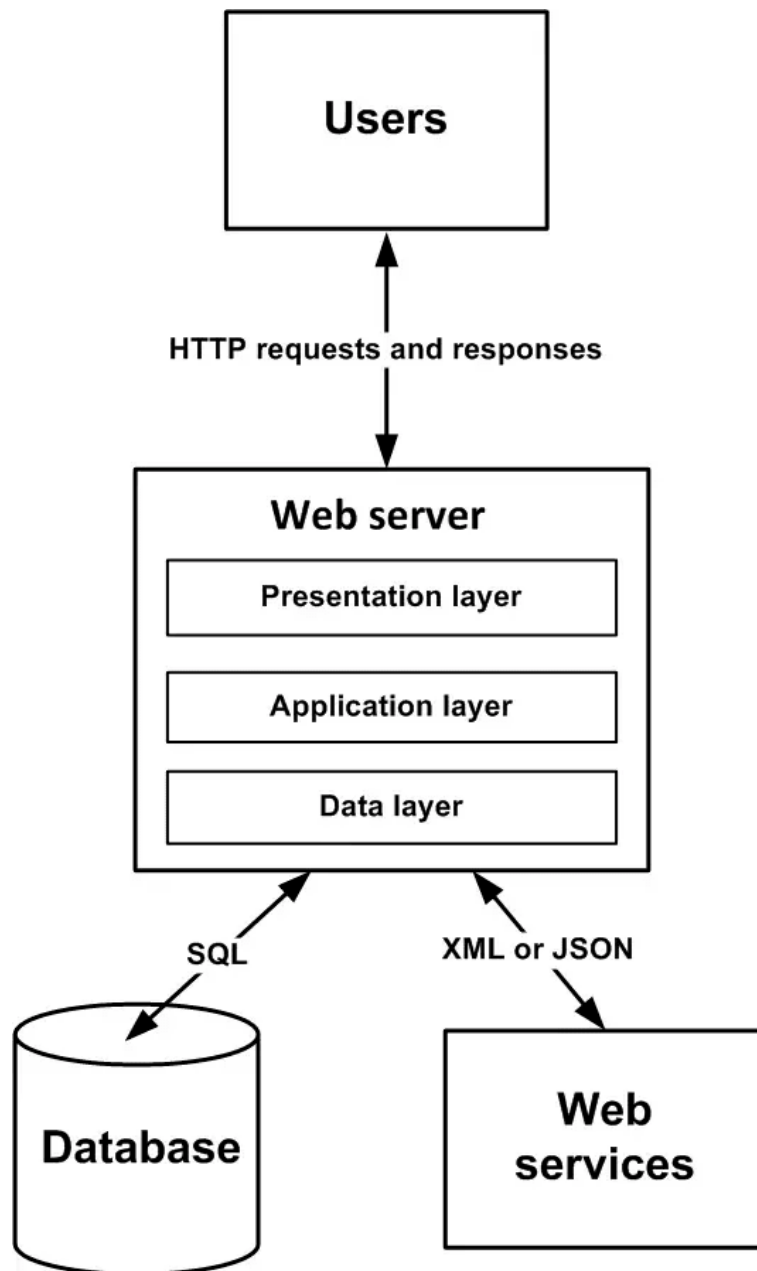
Das Verfahren des Usernamen und des Passwortes wurde gewählt, weil es zum Verfassungszeitpunkt dieser Arbeit die Nutzer von Webseiten meist alleinig sichert. Die Loginverfahren größerer Dienste wie Microsoft Outlook, Google Mail und Facebook besitzen zwar schon seit Jahren die Möglichkeit der Authentifikation mit einem zweiten Faktor, doch das sind eher Ausnahmen. Ein Großteil der Internetpräsenzen ist sich entweder nicht der Risiken von Passwörtern für ihre Nutzer bewusst oder kann sich den Mehraufwand von neuen Verfahren nicht leisten. Gleichzeitig war es im Rahmen dieser Arbeit wichtig eine typische Implementation mit Username & Passwort und ihren Scheinsicherheiten wie Hashes mit Salt und Pepper zu präsentieren. Es ist wichtig, den Ist-Zustand im heutigen Internet zu verstehen, um die Vorteile von neueren Methoden nachzuvollziehen.

Eine dieser neuen Methoden ist das zweite Verfahren. Nach dem Passwort ist das wohl der häufigste zweite Faktor den man findet. Apps wie der 'Google Authenticator' zählen mittlerweile als Synonym für das TOTP - Verfahren wie es im RFC beschrieben ist. Dies liegt unter anderem auch daran, dass die App tatsächlich die recht strikten Vorgaben aus dem RFC befolgt und nur minimale Unterschiede dazu aufweist. An

dieser Stelle stellte sich eher die Frage ob HOTP oder TOTP in den Prototypen implementiert werden sollte. Es wurde sich für das TOTP - Verfahren entschieden, da es zwar den Mehraufwand der Synchronisation zwischen Zeit des Gerätes und des Servers benötigt doch dafür die Sicherheit der begrenzten Zeit bietet wie bereits erläutert.

Das dritte Verfahren, die 'Web Authentication', ist wohl das Erforschungswürdigste der drei Verfahren. Sie ist von allen drei Verfahren am seltensten auf Internetseiten vertreten und da stellt sich nach einer Analyse des Mehrwertes dieses Verfahrens die Frage nach dem Grund dafür. Im Gegensatz zu den anderen beiden Authentifikationsmethoden wurde Dieses nicht für den Prototypen ausgewählt, weil es so weit verbreitet und bereits genutzt wird sondern genau im Gegenteil: Es wurde gewählt, um mögliche Gründe zu suchen weshalb das Verfahren der Webauthn heutzutage kein Standard darzustellen scheint. Es muss untersucht werden ob es starke Nachteile Gegenüber der anderen Verfahren bietet, denn im Ansatz scheint es alle Nachteile des Passwortes auszumerzen.

4.2 Authentifikation in drei Schichten



Das Ziel des Prototypen ist es, wie eingangs erwähnt, vorhandene Authentifizierungsverfahren abseits der klassischen UserID / Passwort Methode zu begutachten und dessen Schwächen aufzudecken. Daraus ergeben sich die typischen drei Komponenten von Drei-Tier-Client-Server Architekturen:

Ein Client, der Anfragen an die Applikation (das Backend bzw. den Server) sendet, welches die Daten dann in einer Datenbank mittels eines DBMS (Datenbankmanagementsystems) persistiert und verwaltet. Wie auch im Bild zur 3 Tier Architektur zu sehen, agiert der User auf dem Presentation Layer und stellt anfragen bzw. sendet JSON Objekte an den Server, der die entsprechenden Webservices liefert. Der Server speichert keine States, wodurch der Nutzer typischerweise mit jeder Anfrage alle benötigten Informationen liefern muss. Der Server liefert demnach lediglich die Representational State Transfer (REST) Schnittstellen. Implementationsdetails und weitere Informationen zum Prototyp gibt es in folgenden Kapiteln.

4.3 Kriterien für eine erfolgreiche Authentifikation

Auch wenn die Frage: “Ist eine gelungene Authentifikation auch eine erfolgreiche Authentifikation?” eine philosophische zu sein scheint, ist sie dennoch nicht minder interessant. Es wurde bereits gezeigt, dass Nutzer nicht gewillt sind komplizierte und aufwendige Dinge zu tun, um sich selbst und ihre Daten im Internet zu schützen. Sofern also die Methodik der Webseite zu kompliziert ist, sucht der Nutzer sich Ausflüchte um die vorhandenen Verfahren so angenehm wie möglich zu machen, was wiederum die Sicherheit ihrer Internetpräsenz gefährdet. Übertragen auf die altbekannten Passwörter bedeutet dies, dass nur weil ein Nutzer sich mit seinem Nutzernamen und seinem Passwort auf einer Webseite anmelden kann, dies nicht bedeutet das das eine wirklich erfolgreiche Authentifikation, aus Sicht des Nutzers, ist. Das Passwort besitzt zu viele Schwachstellen als das man dies als Erfolg verbuchen kann. Es scheint eher dem Zweck ‘irgendeiner Schutzbarriere zwischen fremder Person und Daten des Nutzers’ zu dienen als wirklich zur ‘Sicherheit’.

Überträgt man dieses Wissen auf die Kriterien dieser Arbeit ergeben sich folgende Anforderungen:

- **Nutzerfreundlichkeit**

Angefangen mit einem der wichtigsten Kriterien und der größten Herausforderung an die User Experience (UX) des 21. Jahrhunderts. Der Prototyp soll eine einfache Authentifikation ermöglichen. Keine allzu starken Weiterleitungen, keine Popups, keine langen Ladezeiten und vorallem zwecks Authentifizierung: Keine sinnfreie Aneinanderschaltung von verschiedenen Sicherheitsfaktoren. Ein Beispiel hierfür ist, dass nach der Bestätigung des Einloggenvorgangs durch

Fingerabdruck zusätzlich eine PIN und in folge dessen ein Sicherheitsschlüssel (mit zusätzlich benötigtem PIN) verlangt wird, obwohl nach dem ersten Schritt des Fingerabdrucks bereits die Authentizität des Users festgestellt wurde. So muss der Prototyp am Besten aus nur einem Faktor der Sicherheit bestehen und dem Nutzer die Auswahl zur Selbstentscheidung über die verschiedenen Möglichkeiten bieten. Bietet man dem Nutzer nur eines der Verfahren geht man Gefahr, dass dieses eine als zu umständlich befunden wird. Der User wird versuchen das Verfahren zu vereinfachen um seiner Bequemlichkeit dienlich zu sein. Im konkreten Fall des Prototyps soll es demnach auch reichen nur einen der registrierten Geräte für die Web Authentication zu nutzen, so kann der User bequem die PIN zur Authentifizierung nutzen. Falls der Rechner dies nicht unterstützt, kann der User zum Sicherheitsschlüssel greifen oder das Smartphone für das TOTP - Verfahren verwenden.

- **Sicherheit**

Ein konkretes wichtiges und messbares Kriterium, das der Sicherheit dient ist es, Metadaten zu vermeiden. Der Prototyp soll meist die selben Statuscodes und nur allgemeine Fehlermeldungen bei nicht erfolgreicher Authentifikation zurückgeben. So ist die Nachricht "Das Passwort ist inkorrekt." als absoluter Fehler zu betrachten, da ein Angreifer nun die Möglichkeit hat, durch einen Brute force - Angriff auf das Loginformular das richtige Passwort zu erraten. Diese Nachricht würde nämlich implizieren, dass ein Nutzer mit eingegebenem Nutzernamen in der Datenbank vorhanden ist, doch das Passwort nicht stimmt. Stattdessen werden allgemeine Responsemessages gegeben wie "Der Benutzername oder das Passwort sind falsch.", um dem Angreifer keinerlei Daten zu geben.

Gleichzeitig muss die Verbindung zum Backend, auch im Prototyp, durch ein eigenes SSL - Zertifikat gesichert sein. MITM - Angriffe sind, wie bereits erläutert, nicht immer verhinderbar. In unserem Falle sind wir selbst der Aussteller des Zertifikats und vertrauen uns selbst. Ein User kann (und wird) allerdings nicht die Vertrauenswürdigkeit seines Zertifikates erfragen.

Wenn man die drei Verfahren nach ihrer Sicherheit ordnet, käme an hinterster Stelle wohl das Passwort. Wieso wurde bereits erläutert. Darauf würde das TOTP - Verfahren kommen, das so lange sicher ist wie das Gerät welches die TOTP - Codes erfragt sicher ist. Hat man seinen QR-Code also im Google Authenticator eingescannt und lässt sein Smartphone ungesichert an einem öffentlichen Ort liegen, sind die aufzubringenden Ressourcen für den Angreifer recht gering. Er kann sich das Smartphone nehmen, die App öffnen und

den Code auf der Webseite eingeben. Je nach Betriebssystem können sich weitere Angriffsmöglichkeiten ergeben. So ist es auf Smartphones mit dem Betriebssystem IOS relativ schwierig an appeigene Daten zu gelangen, da jede App in ihrer eigenen Sandbox mit ganz speziellen Rechten geöffnet wird. Auf Android-Smartphones sieht dies bereits anders aus. Dort gibt es einen geteilten Zwischenspeicher, welches den Zugriff auf Daten anderer Apps begünstigt. So könnte man einen Screenshot von einer App machen, die im Hintergrund geöffnet ist und diese per Mail an den Angreifer weiterleiten. Andere bekannte Angriffe auf TOTP - Verfahren sind soweit nicht bekannt. Das sicherste Verfahren unter den Dreine ist die Webauthentikation. Sie bietet wenig Angriffsfläche, dadurch dass die privaten Schlüssel an einem sicheren Ort im Betriebssystem persistiert werden und man nicht so leicht an diese kommt. Der öffentliche Schlüssel besitzt keinen Wert und wird vom Authenticator generiert. Dadurch ist kein Zutun des Nutzers erforderlich wie bei einem Passwort. Dieses Verfahren ist auch relativ sicher gegen MITM - Angriffe, da ein Angreifer maximal die Challenge des Servers abfangen kann, um sein eigenes Gerät für einen Nutzer zu registrieren. Dies erfordert allerdings entweder eine komplette Kontrolle über den Rechner (Remote Access Control) oder physischen Zugriff auf den Rechner durch den Angreifer.

- **Datenschutz**

Die Forschungsfrage dieser Arbeit ist überhaupt erst entstanden, durch einen Datenschutzvorfall bei Firmen. Auch wenn der Datenschutz bis jetzt nicht viel Erwähnung fand, ist er neben der Sicherheit und der Nutzerfreundlichkeit kein minderes Kriterium für eine erfolgreiche Authentifikation. Zum Schutz der Daten gehört zum Beispiel, SQL - Injektionen durch das Benutzen von 'Prepared SQL Statements' zu verhindern. Dies bedeutet, dass Userdaten an keiner Stelle unformatiert in ein SQL Schnipsel eingebaut werden. Gleichzeitig muss sichergestellt werden, dass der Prototyp bei erfolgreichem Login keinerlei Metadaten über den Nutzer preisgibt. So wäre es ein Datenschutz-Problem wenn bei der Authentifikation über Webauthn zusätzliche Daten wie die E-Mail des Nutzers, sein Passwort und die Zeit der Erstellung des Accounts mitgesendet werden. Das Prinzip sollte stets lauten: So wenig Daten wie nötig. Das Recht auf informationelle Selbstbestimmung ist hier insofern gewährleistet, dass der User selbst entscheiden kann, ob er einen TOTP registriert oder welches Gerät er für die Webauthn nimmt. Ein großer Vorteil neuerer Verfahren ist, dass hinter einer TOTP oder Webauthn - Gerätereistration keinerlei persönliche Daten stehen. Selbst wenn Angreifer also den gesamten Registrationsprozess auf dem Gerät mitschneiden, können sie daraus keine verwertbaren Metadaten für zukünftige Angriffe erlangen.

5 Prototypischer Lösungsansatz

5.1 Implementationsdetails

1. Client / Webseite

Die Webseite besteht aus einer zur Single Page Application (SPA) ähnlichen Architektur, die ausschließlich Javascript und JQuery, also clientseitige Programmiersprachen nutzt. SPA's sind Webapplikationen, bei denen der Nutzer eine Seite betritt und diese nie wieder vollständig laden muss. Frameworks wie Angular, React oder VueJS basieren auf dieser Mechanik und nutzen Javascript und JQuery um die Ladezeiten einer Webseite durch Module zu reduzieren. Anstatt also die gesamte Webseite neuzuladen, laden diese Frameworks nur spezielle Bereiche der Webseite asynchron nach. In meinem Prototyp ist dies größtenteils der Fall, da neben der Hauptseite **index.html** eine weitere Seite existiert, auf die man bei erfolgreichem Login weitergeleitet wird: **secret_panel.html**. Auch werden keine Module nachgeladen, sondern bei entsprechender Aktion nur Elemente innerhalb des DOM - Baums per id identifiziert und versteckt.

Beim Betreten von beiden Seiten der Anwendung findet eine Prüfung nach dem Session-Cookie (**user_sid**) statt, die vom Server bei einer erfolgreichen Authentifikation im Header über 'Set-Cookie' als Response zurückkommt und vom Browser in folge dessen gesetzt wird. Der Aufruf der **index.html** Seite ist nicht möglich mit gesetztem Cookie und leitet den User auf **secret_panel.html** weiter und umgekehrt genauso. Die Sicherung dieses Cookies auf Serverseite durch Prüfungen ist nicht Bestandteil dieser Arbeit, hier wird sich lediglich auf den Loginprozess von Anwendungen konzentriert um dessen Schwächen und die Vorteile neuer Verfahren aufzuzeigen. Aktuell könnte ein Angreifer demnach selbst einen entsprechenden Cookie mit einem Wert setzen und das 'geheime Panel' erreichen. Bei weiter ausgereiften Webapplikationen wird bei jedem Request an den Server der Session Cookie mit einer temporären Map innerhalb des Backends abgeglichen. Befindet sich der Session Cookie nicht in dieser Map oder hat nicht genügend Rechte für diese Anfrage, erhält der Nutzer den Statuscode 401 (Unauthorized) zurück. So haben es etablierte Dienste wie 'Netflix' und 'Microsoft' bereits umgesetzt.

Aus der Forschungsfrage ergibt sich, dass die Webseite ausschließlich Javascript (oder auf Javascript basierende Programmiersprachen wie JQuery) zur Implementation der Programmlogik verwendet. Javascript selbst ist eine clientseitige Programmiersprache und gibt dem Nutzer den gesamten Code auf Webseiten preis. Demnach wurde bei der Anwendung darauf geachtet, alle clientseitigen Prüfungen, auch serverseitig zu implementieren. Zumindestens alle wichtigen Prüfungen, die sonst den Programmablauf verhindern würden oder sogar Serverabstürze zur Folge hätten. Nicht nur gibt Javascript den Code preis, über die Konsole (Unter Google Chrome und Firefox in den Entwickleroptionen des Browsers zu finden) lassen sich ganze Funktionen oder globale Variablen manipulieren. Variablen innerhalb von Funktionen können vom Angreifer nicht so leicht manipuliert werden.

Möglich ist es dennoch, den Inhalt dieser Funktion in eine andere Funktion zu schreiben, um die Variable dort auszutauschen. Um dem Nutzer nicht jede Funktion problemlos erreichbar zu machen, wurde bei der Implementation das Revealing-Module-Pattern angewendet, über die gewisse Variablen nicht im globalen Scope (window) sondern im Scope einer Funktion bleiben, die nur gewisse andere Funktionen in sich exportiert und als eine Art Schnittstelle fungiert. Auf dieses Pattern wird im Folgenden näher eingegangen.

Das Design der Webanwendung ist kein wesentlicher Punkt dieser Arbeit und deshalb schlicht gehalten. So besteht der Prototyp aus einer einfachen Webseite, die aus dem globalen Internet erreichbar sein wird und die zwei Eingabefelder und einen Loginbutton besitzt. Um sich die Designarbeit oder das Suchen von Icons und passenden Buttondesigns zu sparen wurde auf bekannte Frameworks wie Bootstrap4 und Funktionalitäten wie Flexbox gesetzt, die eine automatisches Rescaling und Positioning von Webseitenelementen beinhalten. So war es nicht nötig ein Loginformular zu designen sondern vorhandene Strukturen von Bootstrap für Buttons aller Art zu nutzen. Die unterschiedlichen Methoden der Authentifizierung wählt man über ein Dropdownmenü über dem Login - Button. Je nach Authentifizierungsverfahren werden kleine Popup-Boxen sichtbar, die die weiteren Schritte für die Authentifikation erläutern. Die Methode kann sowohl über das Dropdownmenü als auch über einen Klick auf die Pfeile links und rechts neben dem Dropdownmenü gewählt werden, die als Pagination zwischen den verschiedenen behandelten Verfahren fungiert. Ist man am Ende der drei Methoden, springt man wieder zur ersten Methode und andersherum.

Bei der Web Authentication gibt es eine Besonderheit. Die Webseite auf die Schnittstellen des Betriebssystems zugreifen, um den Nutzer zu verifizieren. Dieser Teil kann von meinem Prototyp nicht beeinflusst werden und wird vom

CTAP2 innerhalb des FIDO2 Standards definiert. Dadurch entstehen teilweise merkwürdige und aus UX (User Experience) - Sicht höchst fragwürdige Interaktionen. So fragt das Betriebssystem zunächst (bei entsprechender Möglichkeit) nach einem PIN um den Nutzer zu registrieren. Drückt man nun die Escape-Taste erscheint ein Dialog um einen Sicherheitsschlüssel (ein externes Gerät) einzurichten. Beim Login wiederum ist dies durch eine Dropdownliste schöner gelöst worden, wo der User alle möglichen Loginmethoden auf einem Blick sieht und diese Wählen kann. Während er bei der Registration keine Chance hat dies zu tun und immer erst ein PIN - Feld angezeigt wird. Auf die einzelnen behandelten Verfahren wird in späteren Kapiteln noch genauer eingegangen, da werden solche Schwierigkeiten aufgegriffen da dies nur eines von vielen 'Problemen' neuerer Verfahren ist: Die Abhängigkeit vom Betriebssystem.

2. Server

Der NodeJS - Server besteht aus einer REST Api, die keine Zustände speichert. Neben der Aufgabe des Cookie Managements und der Generierung von UUID's liefert er zudem die Schnittstelle zur Datenbank und verschiedene Funktionalitäten für die behandelten Verfahren Username & Passwort, TOTP und Webauthn:

- **/logout** - POST - Löscht den Session-Cookie (und damit die Session des Nutzers) und leitet ihn auf die Loginseite **index.html** weiter.
- **/get_public_key** - GET - Liest den öffentlichen Schlüssel des Servers ein und gibt diesen in einer JSON - Struktur zurück. Ist wichtig für die Username & Passwort - Authentifizierung.
- **/password/login** - POST - Nimmt einen mit dem öffentlichen Schlüssel des Servers verschlüsselten Usernamen und Passwort entgegen und entschlüsselt diese mit dem privaten Schlüssel. Im Anschluss darauf wird in der Datenbank über einen gepoolte Query nach dem Nutzer gesucht. Wurde dieser gefunden, erstellt der Server einen SHA512 - Hash aus dem Passwort (aus dem Request) und dem Salt des Users (aus der Datenbank) indem die **hashString** - Methode aufgerufen wird. Diese bezieht den Pepper des Servers mit ein. Sofern die Hashes aus der Datenbank und der soeben Generierte übereinstimmen, erhält der Nutzer die Response 200 und den Text "OK". Gleichzeitig wird wie bei jeder anderen Authentifizierungsmethode die **createSessionCookie** - Methode aufgerufen um den Session-Cookie (eine zufällige UUID) im Header zurück an den Nutzer zu senden, sodass dessen Browser ihn setzt..

- **/password/create_password_hash** - GET - Nimmt eine Zeichenkette des Nutzers entgegen und erstellt einen Password-Hash für den Nutzer, der manuell in die Datenbank persistiert werden kann. Ist als 'Quality of Service' Funktion zu verstehen, um es dem Administrator einfacher zu machen, Passwörter zu erstellen, die bei Vergleich einen gültigen Hash ergeben.
- **/totp/check_username** - POST - Diese Methode dient lediglich der Prüfung, ob der Nutzer die TOTP Authentifikation mittels zufälligem SECRET bereits gegenüber der Webseite validiert hat. Das Datenbankfeld 'totp_activated' wird hier geprüft, ist der Wert 1 (stehen für das boolsche 'wahr') liefert der Server den Text "Success" zurück, das der Client deutet um nur das Feld für den sechsstelligen OTP Code anzuzeigen. Hat das Feld allerdings den Wert 0, wird entweder (sofern noch nicht vorhanden im Feld 'totp_secret') ein neues Secret erstellt und in der Datenbank für diesen Nutzer persistiert. Gleichzeitig für eine URL beginnend mit otp:// erstellt, um diesen zusammen mit einem Base 64 enkodierten QR Code an den Client geschickt, der dies dem User präsentiert. Es wurde sich hier bewusst dazu entschieden, diese Schritte auf Serverseite vorzunehmen. Möglich wäre es auch auf Clientseite gewesen, wäre aber mit einem Mehraufwand verbunden, welches hier verhindert werden sollte.
- **/totp/check_token** - POST - Nimmt den Usernamen des Nutzers und einen sechsstelligen TOTP Token entgegen. Sucht im Anschluss darauf in der Datenbank nach dem Nutzer und prüft über die Library 'otplib' ob der eingegebene TOTP - Token zum secret in der Datenbank valide ist. Dieser Schritt findet sowohl bei der Registration als auch beim Login statt. Wenn das Feld 'totp_activated' also 0 ist, wird es bei der ersten Authentifikation auf 1 gesetzt und der Nutzer wird eingeloggt. (Session-Cookie wird gesetzt und nutzer weitergeleitet)

- **/webauthn/generate-attestation-options** - POST Liefert dem Nutzer die verschiedenen Registrieroptionen für die Web Authentication mittels WebauthnAPI. Dabei wurden verschiedene Optionen gesetzt, die im folgenden erklärt werden und die der User im Body der Response als JSON - Objekt erhält.

```

1      {
2          rpName: PROJECT_NAME,
3          rpId: WEBAPP_ORIGIN,
4          userID: queryRes.rows[0].id,
5          userName: username,
6          userDisplayName: username,
7          attestationType: 'none',
8          authenticatorSelection: {
9              requireResidentKey: false,
10             userVerification: "discouraged"
11         },
12         excludedCredentialIDs:
13             userAuthenticators.map(dev => dev
14                 .credentialID),
15     }

```

rpName: rp steht hier bei für Relying Party. Das ist der Name, der später auch im Registrierungsdialog als 'Webseiteninhaber' gelistet wird.

rpId: Die Relying Party Id bekommt eine URL, auf der sich der Nutzer registrieren möchte. In unserem Falle ist dies 'localhost' ohne Zusätze wie das Protokoll oder der Port. Diese Variable ist vor allem für Webseiten im Netz wichtig, sodass ein Angreifer nicht per Phishing den Nutzer zur Registrierung auf einer anderen Webseite bringt. Die Registrierung schlägt clientseitig fehl, wenn die rpId und die Webseitenadresse (Damit ist nicht die Domain sondern die wahrliche IP-Adresse gemeint) nicht übereinstimmen. Beim Wert 'localhost' sendet der Server diesen Teil der JSON - Abfrage nicht mit, da es localhost aus Testgründen nicht prüft. Gleichzeitig erzwingt Webauthn ausschließlich für diese rpId keine sichere Verbindung.

userID: Das ist die ID des Nutzers die verifiziert wird, meist eine fortlaufende Ganzzahl.

userDisplayName: Das ist der Name, der beim Registrieren als Username gelistet wird.

attestationType: Der Server definiert, wie viele Informationen über den Authenticator er im attestation statement haben möchte. Das attestation statement erhält er nach dem der User sich mit einem Authenticator registriert hat und ist ein wichtiger Bestandteil des Objekts welches den Server im Nachhinein über eine erfolgreiche Registration benachrichtigt. Ferner ist der öffentliche Schlüssel des Nutzers in diesem Objekt lokalisiert, den der Server dann in der Datenbank persistiert. 'none' bedeutet hierbei, dass keinerlei Informationen erwünscht sind. 'indirect' als Option würde dem Nutzer erlauben selbst zu entscheiden wie viele Informationen er preisgibt bzw. könnte er eine anonymisierte CA verwenden um sein Zertifikat auszustellen und 'direct' als Option würde die Daten direkt vom Authenticator ohne einen Eingriff des Nutzers erwarten.

authenticatorSelection: Das sind verschiedene Optionen um den verwendeten Authenticator zu bestimmen. Das Argument 'requireResidentKey' bestimmt hierbei darüber, ob ein Authenticator benutzt werden darf, der selbst nicht in der Lage wäre einen privaten Schlüssel auf dem Betriebssystem zu sichern und dem Server in folge dessen einen öffentlichen Schlüssel zu senden. Die 'userVerification' ist im Prototyp nicht erzwungen also 'required' sondern 'discouraged', es obliegt also dem Betriebssystem und dem zu authentifizierenden Gerät / und der initialen Konfiguration zu entscheiden, ob der Nutzer sich gegenüber seines Authenticators verifizieren muss. In dem Beispiel eines bereits sicheren FIDO2 - USB Sticks könnte dies zum Beispiel nicht vom Nutzer gewünscht sein.

excludedCredentialIDs: Dies ist eine Liste von credentialID's die in der Datenbank im Feld 'webauthn_authenticator_data' als JSON - Struktur persistiert sind. Sie dient dazu, bereits registrierte Methoden nicht erneut anzuzeigen, funktioniert dennoch nicht zuverlässig.

3. Datenbank

Das gewählte Datenbankmanagementsystem ist PostgreSQL 12, dies hat einen speziellen Grund. Das Datenbankmanagementsystem besitzt fortgeschrittene Funktionalitäten um JSON Strukturen zu persistieren und zu bearbeiten. Gleichzeitig wäre es möglich Constraints zu erstellen, die bei anderen Managementsystemen wie MySQL nur bei speziellen Tabellentypen wie InnoDB und das auch nicht vollständig verfügbar sind. Der Prototyp benötigt nur eine einzige Tabelle namens **users**. Er speichert typische Daten um den Nutzer zu authentifizieren. Jeder nutze besitzt eine eigene einzigartige ID, die eine Ganzzahl ist. Dann besitzt jeder User einen Nutzernamen im Feld 'username'. Die Felder 'password' und 'salt' besitzen logischerweise nur Nutzer, die die Passwort Authentifikation unterstützen. Die Felder 'totp_secret' und 'totp_activated' werden einerseits genutzt um das Geheimnis (welches der User beim TOTP Setup einscannet) zu erstellen und den Nutzer damit später zu authentifizieren und andererseits um dem Frontend zu signalisieren, ob der Nutzer das Setup vollzogen hat. Die Challenge-Felder für Webauthn 'webauthn_register_challenge' und 'webauthn_login_challenge' sind als eine Art Zwischenablage zu verstehen und hätten theoretisch auch im Cache des Backends persistiert werden können, da bei jeder neuen Registration und jedem neuen Login durch Webauthn eine neue Challenge vom Server erstellt und vom Nutzer nach dem Signieren zurückkommt. Interessant ist das Feld 'webauthn_authenticator_data', das in einer JSON-Liste alle registrierten Geräte für die Web Authentication sichert.

```

1  [
2      {
3          "counter": 0,
4          "publicKey": "pAEDAzkBACBZAQCgna71QXVhAofa -
5              ptd...",
6          "credentialID": "HEeJWgh9Q0WURee7dF..."
7      },
8      {
9          "counter": 2,
10         "publicKey": "pQECbt1via1qrycev0vyc...",
11         "credentialID": "_HbcWx_XWd6Pyjy56L0jqPh..."
12     },
13     {
14         "counter": 5,
15         "publicKey": "pQECAyYgASFYIGpEwjn -
16             koYPv8NjXshLZA5C..."
17     }
18 ]

```

```
15 |         "credentialID": "1  
    |             V145nEVRuXWky8CyvW79pq1qxe..."  
16 |     }  
17 | ]
```

Dabei besitzt jedes Element jeweils einen counter, um die Anzahl an Einloggversuchen zu zählen, den öffentlichen Schlüssel und eine einmalige credentialID, die bei jeder Registration als 'exludedCredentialID' an den Clienten übermittelt wird, um zu signalisieren das diese credentialID sich nicht erneut registrieren kann. (Um doppelte Registrationen von ein und dem selben Gerät zu verhindern für den selben Nutzer)

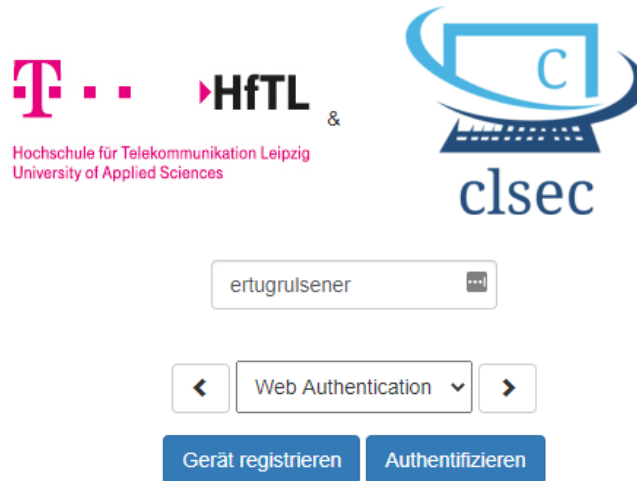
5.2 Fehlerbetrachtung

Die Fehlerbetrachtung ist essenziell um nun auf die Probleme einzugehen, die während der Implementation bereits ersichtlich wurden.

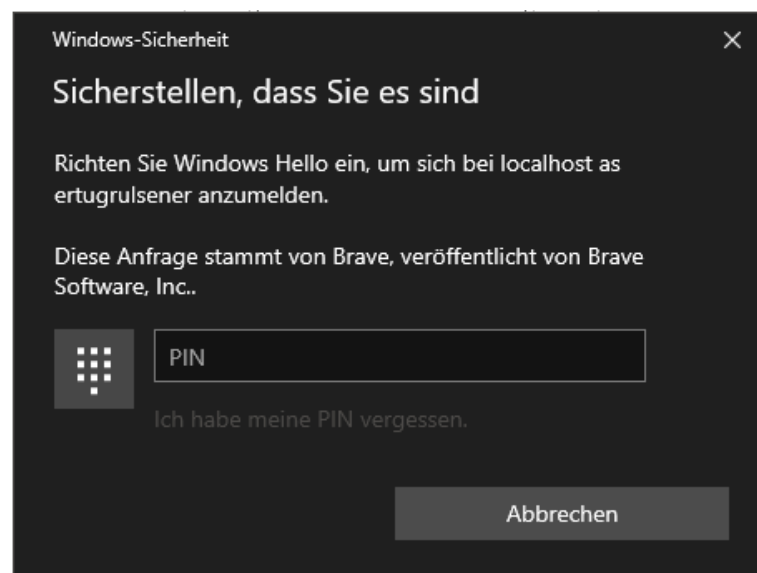
1. UX-Probleme bei Geräteregistrierung für Webauthn unter Windows

Bei der Implementation des Webauthn - Protokolls schien es zunächst ein Mal nicht ersichtlich ob die unerklärlichen Dialogboxen nun ein Implementationsproblem darstellten oder tatsächlich so gewollt sind. Erst die Prüfung durch das offizielle 'Playground-Tool' der FIDO Alliance, zu finden unter '<https://webauthn.io/>' wurde es ersichtlich, dass die verwirrenden UX Entscheidungen von Windows kein Fehler des Prototyps sind. Dennoch tragen sie zur allgemeinen Unverständnis bei. Bei diesem Problem geht es um folgende Abfolge an Dialogboxen, die bei der Registration eines neuen Geräts erscheinen:

- a) Der Nutzer klickt unter Angabe seines Nutzernamens auf den Button 'Gerät registrieren' und erwartet eine Auswahl an vorhandenen Geräten, die registriert werden können. In diesem Beispiel stehen ein physischer Sicherheitsschlüssel und Windows-Hello-PIN zur Verfügung.

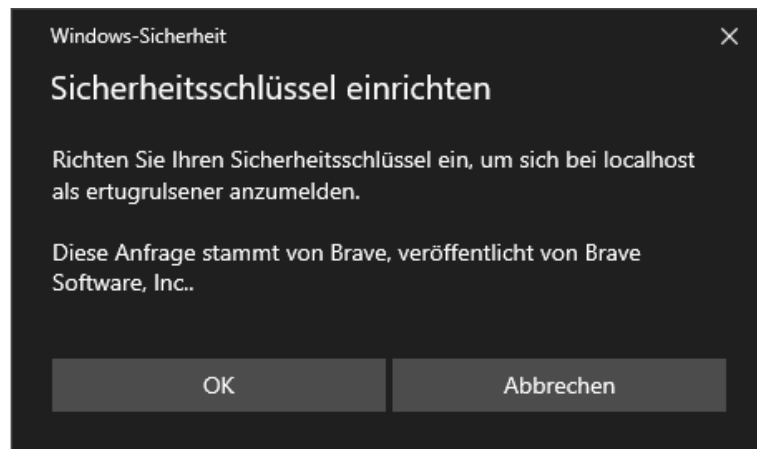


- b) Statt der erwarteten Auswahlbox erhält der Nutzer die Möglichkeit auf die Windows-Hello-PIN und geht nicht von weiteren Auswahlmöglichkeiten aus.

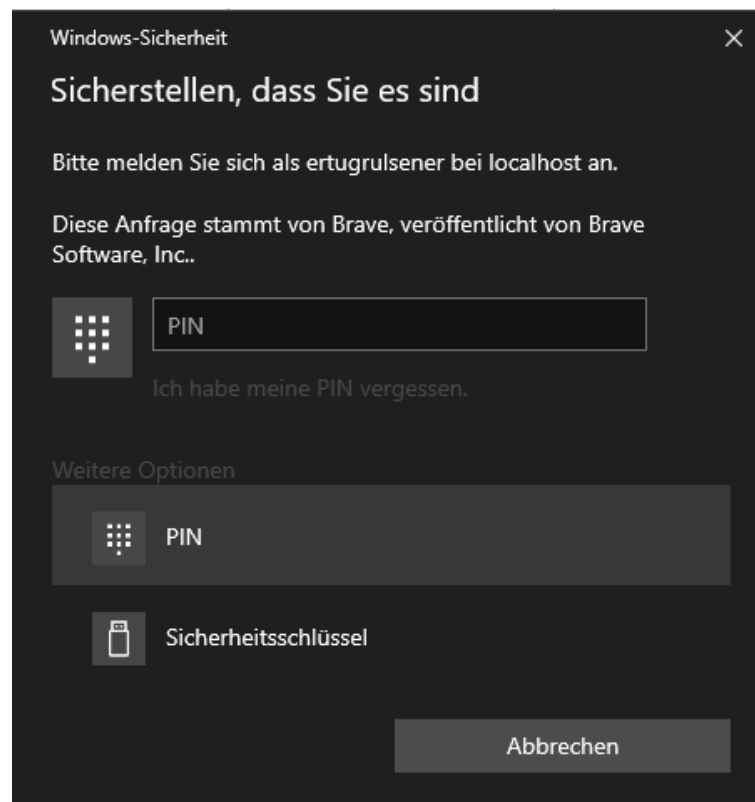


- c) Beim Betätigen der Escape-Taste erscheint plötzlich der Dialog, um einen Sicherheitsschlüssel zu registrieren. Dies ist bereits ein Problem, weil der User zum Zeitpunkt der Registration nicht wissen kann, dass dies

überhaupt möglich war. Bei der Implementation ist diese Option auch nur durch Zufall aufgefallen.

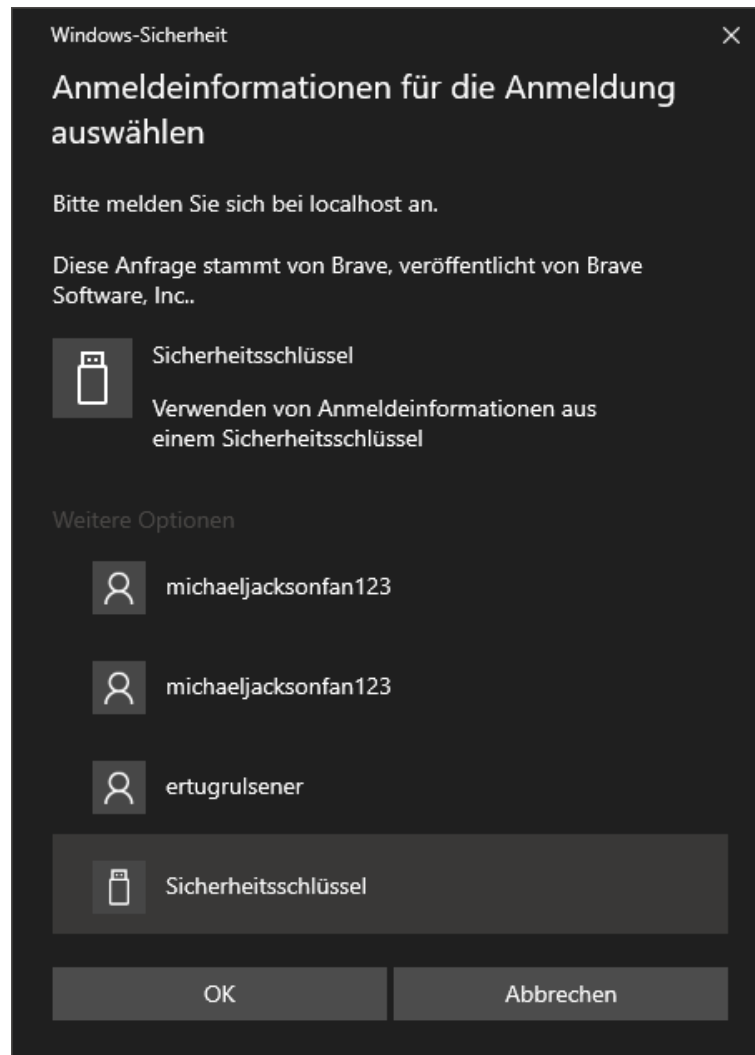


Im Vergleich dazu sieht der Dialog für den Login, wie vom Nutzer erwartet, durch ein Gerät folgend aus (sofern das Feld 'webauthn_authenticator_data' nicht leer ist):



2. Unerwartete Auswahlmöglichkeiten bei keinem registrierten Authenticator

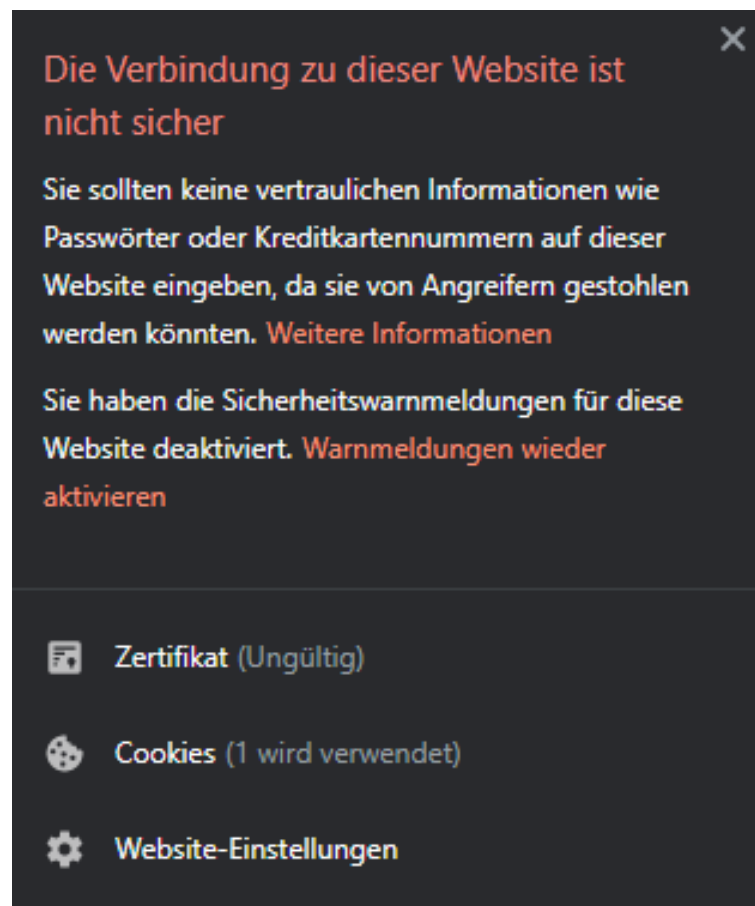
Sollte das Feld 'webauthn_authenticator_data' allerdings leer sein, erscheint folgender Dialog:



Dieser ist insofern ein Fehler, da kein Sicherheitsschlüssel registriert wurde und bei Auswahl dessen nichts passiert. Gleichzeitig werden hier von Windows alle erstellten 'Profile' während der Testphase angezeigt, dessen Name gleich verwirrend für den Nutzer ist. Wenn also das Profil mit dem Namen 'michaeljacksonfan123' versucht einzuloggen, erhält er einen Statuscode 401 (Nicht autorisiert) als Response. Der Fehler hierbei ist die Anzeige einer Einloption durch Windows-Sicherheit, die nicht valide ist. Selbst bei bestätigtem und korrektem PIN. Die Felder 'exludedCredentialIDs' für die Registration

und 'allowedCredentialIDs' für den Login scheinen keine Wirkung zu zeigen, dies könnte einerseits an der WebauthnAPI liegen, die ein Modul für NodeJS ist. Andererseits könnten nicht richtig konfigurierte Optionen für den Login ('/webauthn/generate-assertion-options' - Pfad) oder für die Registrierung ('/webauthn/generate-attestation-options' - Pfad) der Grund sein.

3. Selbsterstelltes Zertifikat wird vom Browser als unsicher eingestuft



4. Bruteforce-Angriffe können Datenbank überlasten

Der Prototyp besitzt keinerlei Zugriffskontrolle. Demnach ist es zum jetzigen Stand möglich, unendlich Anfragen an das Backend zu senden, welches z.B: SQL Anfragen an die Datenbank stellt. Dies könnte die Datenbank überlasten oder sogar den Server crashen. Sollte nämlich kein Pooling mehr möglich sein, wird der Server einen Fehler schmeißen und abstürzen. Dies sollte in einer im Business-Umfeld verwendeten Anwendung implementiert werden, um sowohl den Crash des Servers als auch die Integrität der Datne zu gewährleisten.

Ohne solche Abfragen ist das Erraten der Daten des Nutzers (vorallem des Passwortes) nur eine Frage der Zeit und nicht mehr der Möglichkeiten.

6 Auswertung

7 Ausblick und Fazit

Selbstständigkeitserklärung

Hiermit erkläre ich, Ertugrul Sener, dass die von mir an der *Hochschule für Telekommunikation Leipzig (FH)* eingereichte Abschlussarbeit zum Thema

„Inwiefern bietet die Authentifikation ohne Passwort Vor- und Nachteile gegenüber Webanwendungen hinsichtlich Nutzerfreundlichkeit, Datenschutz und Sicherheit?“

selbstständig verfasst wurde und von mir keine anderen als die angegebenen Quellen und Hilfsmittel verwendet wurden.

Leipzig, den 4. Oktober 2020

Ertugrul Sener

Abkürzungsverzeichnis

2FA	Zwei-Faktor-Authentifizierung	8
BSI	Bundesamt für Sicherheit in der Informationstechnik	10, 19, 20
HOTP	HMAC based one time password	31
HTTPS	Hypertext Transfer Protocol Secure	13, 15
MITM	Man-in-the-middle	13, 14, 15, 41, 42
OTP	One time password	30
REST	Representational State Transfer	40
SPA	Single Page Application	43
TOTP	Time based one time password	30
U2F	Universal second factor	28
URL	Uniform Resource Locator	15
W3C	World Wide Web Consortium	22
WebAuthn	Web Authentication	28

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis