RUHR-UNIVERSITÄT BOCHUM

Praktische Sicherheitsanalyse des Mozilla Single Sign-on Protokolls BrowserID

Henning Thiel

Bachelorarbeit - 1. September 2014.
Lehrstuhl für Netz- und Datensicherheit – Prof. Dr. Jörg Schwenk
Betreuer: Christian Mainka, Vladislav Mladenov

Abstract

In dieser Bachelorarbeit wird die Funktionsweise und Sicherheit des Single Sign-on Protokolls BrowserID untersucht.
Ziel dieser Arbeit ist es, das Protokoll detailliert zu analysieren und zu dokumentieren. Außerdem soll die Sicherheit des Protokolls überprüft werden, um mögliche Schwachstellen aufzuzeigen.
Hierfür wird eine praktische Analyse des Protokolls durchgeführt. Das heißt, es werden die einzelnen Protokollnachrichten untersucht und dahingehend überprüft, ob ein Angreifer hier effektiv eingreifen könnte.

Im weiteren Verlauf der Arbeit wird ein eigener Identity Provider implementiert. Mit diesem wird untersucht, inwieweit ein Angreifer Einfluss nehmen kann, wenn er nicht nur in der Rolle als Benutzer agieren kann. Dazu wird davon ausgegangen, dass ein Angreifer neben seiner Rolle als Benutzer einen Identity Provider betreibt. Dabei wird überprüft, ob er andere Benutzer impersonifizieren oder das Protokoll auf andere Weise manipulieren kann.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Auch erkläre ich mich damit einverstanden, dass meine Bachelorarbeit am Lehrstuhl NDS dauerhaft in elektronischer und gedruckter Form aufbewahrt wird, und dass die Ergebnisse aus dieser Arbeit unter Einhaltung guter wissenschaftlicher Praxis in der Forschung weiter verwendet werden dürfen.

Henning Thiel

Inhaltsverzeichnis

1 Einführung

Dieses Kapitel wird sich der Motivation und den Zielen dieser Arbeit widmen. Der letzte Teil 1.3 dieses Kapitels wird eine Übersicht über die gesamte Arbeit gegeben.

1.1 Motivation

Die heutigen Single Sign-On Systeme bieten dem User eine Möglichkeit, seine Konten der verschiedenen Dienste zu bündeln. Da aber durch diese Bündelung die gesamte Online-Identität von einem System abhängt, ist die Sicherheit dieser Single Sign-on Systeme essentiell.
Gleichzeitig ist zu beachten, dies gilt im besonderen für den Identity Provider, dass zentrale Instanzen existieren an denen die Daten der Benutzer hinterlegt sind oder gesammelt werden können. Dies wiederum kann ein Privatsphärenproblem darstellen, da die Nutzungsgewohnheiten des Benutzers für den Identity Provider verfolgbar sind.
Mozilla möchte mit dem hier untersuchten Protokoll diesem Problem entgegentreten; denn dies ist einer der Gründe, weshalb BrowserID dezentral aufgebaut ist.

Aus dieser Dezentralität folgt, dass ein Angreifer einen eigenen Identity Provider betreiben könnte. Dies eröffnet dem Angreifer weitreichende Möglichkeiten in das Protokoll einzugreifen oder auf andere Teilnehmer wie Benutzer oder Service Provider einzuwirken. Um diese Problematik zu untersuchen, wird im späteren Verlauf der Arbeit ein eigener Identity Provider implementiert.

1.2 Ziele der Arbeit

Ein großer Teil der Arbeit beschäftigt sich damit, dass BrowserID Protokoll detailliert zu untersuchen und für die späteren Fragestellungen ausführlich zu dokumentieren; denn nur mit einem umfassenden Verständnis für alle Abläufe, Funktionen und Mechanismen eines Protokolls ist es möglich, dieses effektiv auf Schwachstellen zu untersuchen.
Im weiteren Verlauf dieser Arbeit werden primäre Identity Provider vorgestellt. Dabei handelt es sich um Identity Provider, die im ungünstigsten Fall von nicht vertrauenswürdigen Akteuren betrieben werden können. Da diese Identity Provider aber eine elementare Rolle im BrowserID Protokoll einnehmen, wird diese Problematik im späteren Verlauf der Arbeit genau untersucht.
Der letzte Eckpunkt dieser Arbeit ist eine Sicherheitsanalyse des Protokolls. Diese Analyse wird auf den beiden vorangegangenen Punkten aufbauen, um eine Einschätzung zur Sicherheit des Protokolls zu geben.

1.3 Gliederung der Arbeit

Nach der Einführung in Kapitel 1 wird im Kapitel 2 das Hintergrundwissen vermittelt, welches für das Verständnis von Single Sign-on Protokollen und BrowserID unerlässlich ist.
Kapitel 3 beinhaltet die Protokollanalyse. In dieser Analyse wird das Protokoll detailliert untersucht und dokumentiert. Darauf folgt das Kapitel 4, welches die Realisierung eines eigenen primären IdP beschreibt. Dieses IdP wird für die spätere Sicherheitsanalyse und das praktische Verständnis dieses Protokollteils benötigt.
Das Kapitel 5 - Sicherheitsanalyse - befasst sich aufbauend auf den beiden vorangegangen Kapiteln mit der Sicherheit des gesamten Protokolls. Dazu werden die einzelnen Sicherheitsmechanismen des Protokolls untersucht. Hierzu zählen unter anderem die verwendeten Signaturen oder die Verschlüsselung.
Im letzten Kapitel 6 erfolgt eine Zusammenfassung und es werden offene Probleme für nachfolgende Arbeiten aufgezeigt.

2 Hintergrund

In diesem Kapitel werden die Grundlagen erklärt, die für das Verständnis des Protokollablaufes von BrowserID benötigt werden.
Dazu werden zuerst die verschiedenen Protokollteilnehmer vorgestellt und erläutert. Im Anschluss wird auf die Besonderheiten von Persona eingegangen und abschließend die Vor- und Nachteile aufgezeigt, die sich durch ein Single Sign-On Protokoll ergeben.

2.1 Protokollteilnehmer

Dieses Kapitel beschäftigt sich mit den Teilnehmern, die während des Protokollablaufs auftauchen. Dazu wird sowohl auf ihre Aufgabe und Funktion, als auch auf ihre Beziehung zueinander eingegangen.

2.1.1 Benutzer

Bei dem Benutzer handelt es sich um den Teilnehmer, der verschiedene Dienste von Service Providern nutzen möchte. Aus diesem Grund muss er sich bei einem Identity Provider registrieren und seine Identität bestätigen.
Da es sich bei BrowserID um ein Protokoll handelt, welches im Internet Anwendung findet, kommen die Benutzer typischerweise über ihren Browser mit diesem in Kontakt. Aus diesem Grund werden die Benutzer im weiteren Verlauf dieser Arbeit oft durch ihren Browser repräsentiert.
Wir werden später noch sehen, dass BrowserID HTML5 Funktionen voraussetzt. Aus diesem Grund benötigt der Benutzer einen aktuellen Browser, der diese Eigenschaften mitbringt. Der Browser Kompatibilitätsliste[1] von Mozilla lässt sich aber entnehmen, dass praktisch alle aktuellen Browser für die Benutzung geeignet sind.

2.1.2 Service Provider

Der Service Provider (SP) oder auch Relying Party (RP) ist der Teilnehmer, der seine Dienste den Benutzern zugänglich machen möchte.
Dies möchte er typischerweise nur mit vertrauenswürdigen Benutzern tun. Im Falle von BrowserID sind das alle Benutzer, die sich bei einem Identity Provider angemeldet haben. Der Vorteil für SP besteht darin, dass sie die sensiblen Teile der Benutzerverwaltung an eine dritte Instanz abgeben können, das heißt, dass Überprüfen und Speichern der Mailadressen sowie Passwörter.
Damit sich ein Benutzer bei einem SP mittels des BrowserID Protokolls authentifizieren

[1] `https://developer.mozilla.org/en/Persona/Browser_compatibility`

kann, muss dieser allerdings ein paar technische Voraussetzungen[2] erfüllen. Dazu zählt zum Beispiel das Bereitstellen der Schnittstelle, an der sich die Benutzer über das BrowserID Protokoll anmelden können. Außerdem muss ein SP diese Anmeldung überprüfen, was bedeutet, dass er die vom Benutzer erhaltenen Daten validieren muss.

2.1.3 Identity Provider

Der Identity Provider (IdP) hat die Aufgabe, die Identität seiner Benutzer sicherzustellen und für die SP zu beglaubigen. Diese Aufgabe erfüllt er, indem er die Mailadresse seiner Nutzer verifiziert und jedem dieser Nutzer ein Zertifikat ausstellt.
Da ein IdP typischerweise sehr viele Nutzer verwaltet, sollte er auch sehr auf die Sicherheit dieser Daten bedacht sein; denn für Angreifer ist er durch diese Bündelung an Nutzerdaten ein lohnenswertes Ziel. Auch sollte nicht vernachlässigt werden, dass ein IdP oft die zentrale Stelle ist, durch die er ein Problem für die Privatsphäre seiner Nutzer werden kann.
Dieses Problem versucht BrowserID zu lösen, indem es als dezentrales Protokoll entwickelt wurde. Das heißt, es ermöglicht den Einsatz von mehreren unterschiedlichen Identity Providern.

Die Idee hinter BrowserID ist, dass praktisch jeder Benutzer bereits eine Mailadresse besitzt. Da Mailadressen sehr oft als Identifikationsmerkmal bei SP eingesetzt werden, versucht BrowserID dies für das Protokoll zu nutzen. Das legt auch den Schluss nahe, dass Mailprovider gleichzeitig als IdP agieren können, da die Benutzer zwangsläufig ihrem Mailprovider vertrauen; denn eine Mailprovider kann theoretisch alle Kommunikationspartner und Mails seiner Benutzer einsehen.

2.2 Persona

Persona ist der Name unter dem die Firma Mozilla ihr Produkt und das BrowserID Protokoll betreibt. Bei dem Protokoll BrowserID handelt es sich um eine Weiterentwicklung des Verified Email Protocols [1], welches sich aber nur in wenigen Punkten unterscheidet. Mozilla stellte erstmals Mitte 2011 BrowserID vor. Im Jahre 2012, als die Betaphase begann, wurde es in Persona umbenannt. Heute findet sich Persona als Authentifikationsmöglichkeit auf praktisch jeder von Mozilla betriebenen sowie auf einigen weiteren Webseiten. Dazu zählen zum Beispiel `www.mahara.org` und `www.voo.st`.
Eine Besonderheit von Persona ist, dass man das Protokoll mit einem bestehendem Yahoo- oder GMail-Konto nutzten kann; denn Mozilla stellt unter dem Projektnamen BigTent eine Schnittstelle bereit, die es ermöglicht, mit dem dort zum Einsatz kommenden OpenID Protokoll zu kommunizieren. Dies ermöglicht dem Benutzer, die bei den beiden Anbietern bestehenden Konten für den Login mit dem BrowserID Protokoll zu nutzen.
Die Dezentralität des Protokolls ist eine weitere Besonderheit, durch die sich Persona von anderen Single Sign-On Lösungen unterscheidet. Für denn Fall, dass ein Mailprovider

[2] `https://developer.mozilla.org/de/docs/Persona/Schnellstart`

keinen eigenen IdP betreibt, stellt Persona den sogenannten Fallback IdP oder auch sekundären IdP bereit. Dieser ist unter `https://login.persona.org` erreichbar. Alle sonstigen IdP werden als primäre Identity Provider bezeichnet.
Die technischen Details sowie Unterschiede des primären und sekundären Identity Providern werden im Kapitel 3 erläutert.

2.3 Single Sign-on

Protokolle wie BrowserID fasst man unter dem Begriff Single Sign-On (SSO) Protokolle zusammen. Die Funktion dieser Protokolle ist, dass sich der Benutzer nur einmal gegenüber einem IdP authentifizieren muss. Der IdP stellt dem Benutzer, bei erfolgreicher Authentifikation ein Zertifikat oder Assertion aus, welche bestätigt, dass der Benutzer dem IdP bekannt ist.
Für alle weiteren Authentifizierungen gegenüber einem oder verschiedenen SP benötigt der Benutzer keine weitere Auhentifikation, sondern überträgt seine Assertion. Anhand dieser Assertion kann der SP überprüfen, ob es sich bei dieser Nachricht um eine legitime Anfrage handelt. Der technische Hintergrund hierzu wird ebenfalls im Verlauf des kommenden Kapitels 3 erläutert.

2.3.1 Vorteile und Nachteile

Die Vorteile eines Single Sign-On Systems liegen klar bei dem Benutzer. Der Nutzer muss sich - im Optimalfall - nur eine Benutzername-Passwort-Kombination merken. Ohne SSO müsste er sich zu jedem SP eine neue Kombination überlegen oder er wählt bei jedem SP dieselbe Kombination, was die Sicherheit negativ beeinträchtigen kann.
Ein weiterer Sicherheitsaspekt ist, dass das Passwort nur einmal übertragen werden muss. Dies schränkt unter anderem Phishing Angriffe ein, da der Benutzer sich nur einmal an einer bekannten Stelle authentifizieren muss.
Im Falle einer Änderung oder Löschung eines Benutzerkontos kann der Nutzer dieses Zentral in seinem Single Sign-On Konto tun. Auch spart der Nutzer Zeit, weil er sich nur einmal gegenüber dem IdP authentifizieren muss. Alle weiteren Authentifikationen laufen mit minimalen Aufwand für den Benutzer ab.

Ein großer Nachteil hingegen ist, das der IdP der sogenannten Single Point of Failure ist. Bei einem fehlerhaften IdP besteht das Risiko, dass alle SP von einem erfolgreichen Angriff betroffen sind und unerlaubter Zugriff auf Ressourcen erfolgt.
Ein weiterer Kritikpunkt ist, dass IdP die Privatsphäre ihrer Benutzer gefährden könnten, weil diese die Nutzergewohnheiten verfolgen können.
Auch darf nicht vernachlässigt werden, dass SSO Systeme nur funktionieren, solange der IdP erreichbar ist.

3 Protokollanalyse

Dieses Kapitel befasst sich mit dem eigentlichen Protokoll das hinter Persona steht. Es beschreibt detailliert die Protokollnachrichten und wie die einzelnen Teilnehmer miteinander kommunizieren.
Dabei werden die beiden IdP Arten, also primärer und sekundärer IdP getrennt betrachtet. Da während des Ablaufes einige JavaScript Dateien nachgeladen und verschiedene Daten im Local Storage gespeichert werden, wird auf diese Punkte in getrennten Kapiteln eingegangen.
Abschließend werden wichtige Funktionen wie die Assertion und ihre Bedeutung für das Protokoll erklärt.

3.1 Testumgebung

Für die praktische Analyse des Protokolls wurden verschiedene Programme genutzt. Für das Aufzeichnen und die Manipulation der HTTP Nachrichten zwischen Browser und IdP oder Browser und SP wurde die Free Edition der Burp Suite[1] verwendet.
Als Browser wurden hauptsächlich der Google Chrome Browser und der Internet Explorer von Microsoft verwendet, weil diese die besten Debugging Tools mitbrachten.
Hier ist zu beachten, dass diese vor jeder Benutzung komplett zurückgesetzt, bzw. im Incognito Modus betrieben wurde. Dies sollte unverfälschte und voneinander unabhängige Tests sicherstellen.

Für den hier beschriebenen Ablauf des primären IdP dient der Persona-TOTP als Grundlage. Es handlet sich dabei um einen in Python geschriebener IdP, der die nötigsten Funktionen mitbringt. Ausführlichere Informationen zu diesem IdP finden sich im folgenden Kapitel 4.

3.2 Sekundärer Identity Provider

Dieses Kapitel beschreibt die Kommunikation aus der Sicht eines Benutzers während des SSO Vorgangs mit dem sekundären IdP von Persona. Dabei wird davon ausgegangen, dass der Benutzer ein verifiziertes Konto bei Persona besitzt, es sich aber um einen SP-initiierten Login handelt. Das heißt, der Benutzer ruft zuerst die Seite des SP auf, ist aber noch nicht gegenüber seines IdP authentifiziert. Außerdem wird vorausgesetzt, dass der User hier einen Browser benutzt, in dem noch keine Cookies oder andere Daten der Protokollteilnehmer existieren.

[1] `http://portswigger.net/burp`

Zur Notation sei gesagt, beginnt die Nachricht mit einem Schrägstrich handelt es sich um GET Parameter, bei kursivem Text werden Parameter per POST übertragen und gestrichelte waagerechte Linien stellen optionale Nachrichten dar.
Zu beachten ist auch, dass alle Nachrichten, die zwischen IdP und Benutzer ausgetauscht werden, per Hypertext Transfer Protocol Secure (HTTPS) geschützt sind. Wohingegen die Nachrichten zwischen Benutzer und SP via Hypertext Transfer Protocol (HTTP) also unverschlüsselt übertragen werden können.
Aus Gründen der Übersicht wird der gesamte Ablauf in drei Phasen unterteilt. Eine vollständige Abbildung (Abb. A.1) findet sich im Anhang dieser Arbeit.

Phase 1 (Abb. 3.1) zeigt das Aufrufen der SP Webseite. In Nachricht 1 ruft der Benutzer die Webseite eines beliebigen SP auf. Dieser antwortet mit der eigentlichen Webseite und veranlasst das automatische Nachladen der *auth.js* sowie der *include.js* von dem Persona Server.

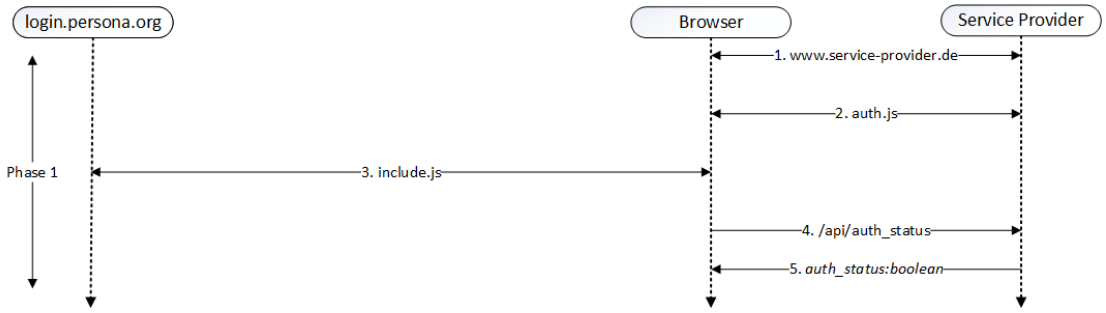

Abbildung 3.1: Protokollablauf Persona IdP Phase 1

Auf die *include.js* sowie auf alle weiteren JavaScript (JS) Dateien, die von dem Persona Server nachgeladen werden, wird in einem eigenen Kapitel 3.4 eingegangen.
Die *auth.js* übernimmt ab dieser Stelle die Kommunikation mit dem SP. Sie wird von jedem SP eigens implementiert und übernimmt die Authentifikation und Sessionverwaltung.
Mit Sessionverwaltung ist hier gemeint, dass überprüft wird, ob der Benutzer schon authentifiziert ist. Dies wird typischer Weise anhand eines Session Cookies erkannt.
Genau dieser Funktion kommt die *auth.js* in Nachricht 4 nach. Sie fragt den aktuellen Sessionstatus des Benutzers ab. In diesem Fall antwortet der SP in Nachricht 5 mit einem *auth_status:false*, da der User noch nicht authentifiziert ist.

Phase 2 (Abb. 3.2) zeigt die Authentifikation des Benutzers gegenüber dem Persona IdP.
Die Phase beginnt damit, dass der Benutzer auf der Webseite des SP den Loginbutton betätigt. Diese Aktion löst eine Reihe an Nachrichten zwischen dem Browser und Persona aus. Zum einen öffnet der Browser den Communication IFrame, der wiederum die dazugehörige *communication_iframe.js* Datei lädt, zum anderen öffnet sich das Sign In Popup, welches die *dialog.js* lädt. Beide Dateien werden ebenfalls in einem eigenen

Kapitel erläutert.
In Nachricht 10 erfragt der Browser den aktuellen Session Kontext bei Persona, ob er bereits authentifiziert ist. Die Antwortnachricht 11 enthält zudem eine ganze Reihe an Parametern[2], die für den späteren Verlauf von Bedeutung sind.

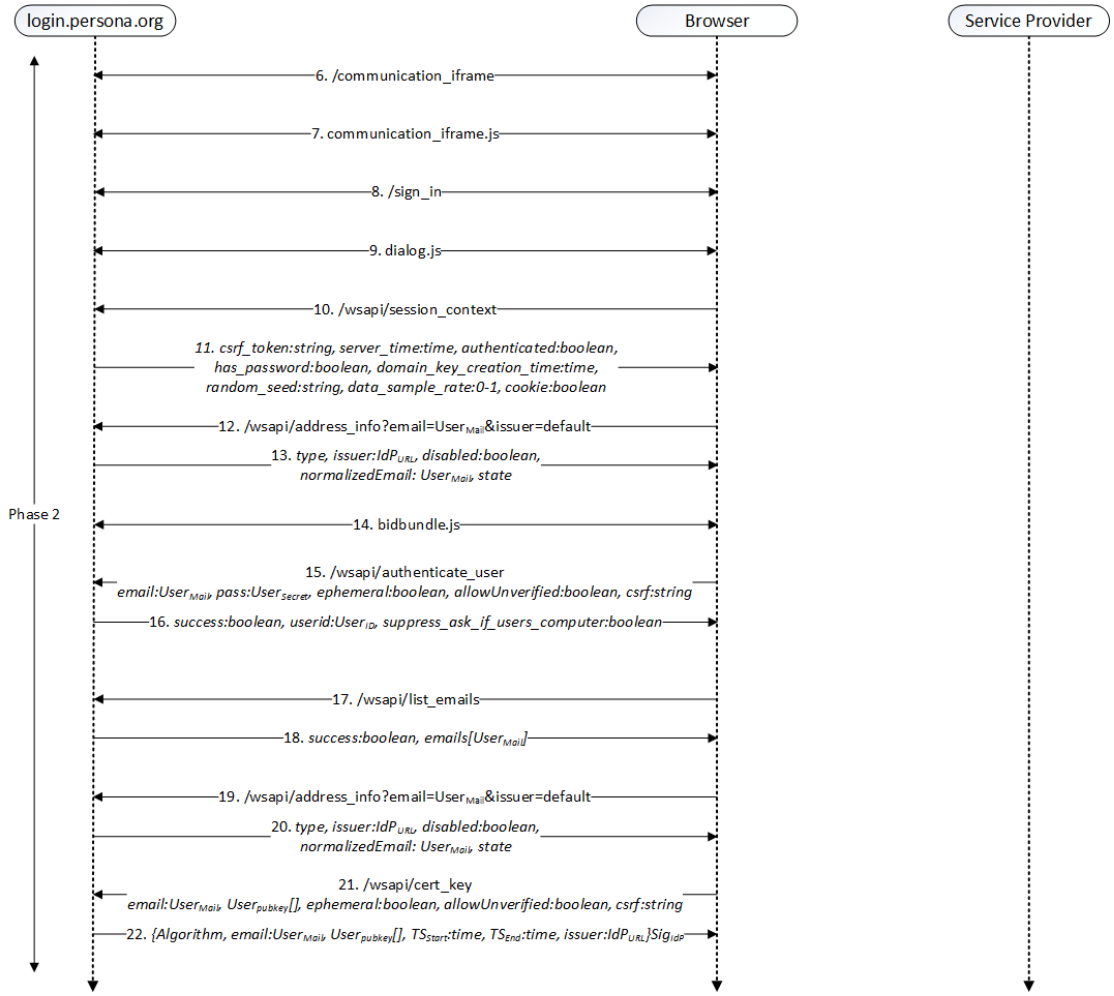

Abbildung 3.2: Protokollablauf Persona IdP Phase 2

Sie enthält ein einmalig generiertes Cross-Sit Request Forgery (CSRF) Token, mit dem im späteren Verlauf die Kommunikation gegen CSRF Angriffe gesichert werden soll. Da im späteren Protokollverlauf verschiedene Nachrichten einen Zeitstempel benötigen, sendet der Server seine Uhrzeit in der *server_time* Variablen mit. Hier ist zu beachten, dass alle Zeitangaben in Millisekunden gemacht werden, sie also vom gebräuchlichen Unix Format abweichen, da sie drei weitere Stellen besitzen.

[2] `https://github.com/mozilla/persona/wiki/Front-End-Development#what-is-thiswsapisession_context-stuff`

Die beiden Variablen *authenticated* und *has_password* sind beide *false*, weil der Benutzer hier noch nicht authentifiziert ist. Die *domain_key_creation* Variable enthält den Zeitpunkt, an dem der Master Key des IdP geändert wurde. Sollte ein Benutzer eine Assertion besitzen, welche vor diesem Zeitpunkt ausgestellt wurde, muss diese erneuert werden.
Der *random_seed* enthält einen serverseitig generierten Zufallswert, der für die spätere Schlüsselerzeugung des Browsers von Bedeutung ist.
Die *data_sample_rate* kann einen Wert zwischen 0 und 1 annehmen und dient Performancemessungen von Persona. Die Variable *cookie* dient lediglich dazu, beim Browser eine Fehlermeldung auszulösen, sollte dieser keine Cookies unterstützen. Persona stellt dies fest, indem es in Nachricht 10 schon das *can_set_cookies* Cookie erwartet.

Vor der Nachricht 12 muss der Benutzer wieder aktiv werden, indem er in das geöffnete Popup Fenster, seine Mailadresse eingibt. Diese wird in Nachricht 12 per GET Request zusammen mit dem *Issuer* Parameter an Persona gesendet. In dem *Issuer* Parameter lässt sich ein abweichender Issuer spezifizieren. Dazu könnte hier die abweichende Domain mitgesendet werden, im Normalfall wird aber ein *Issuer=default* übertragen.
Persona kann nun anhand der Mailadresse überprüfen, ob zu dieser ein IdP existiert. Ist dies, wie hier, nicht der Fall, so wird im Typefeld *secondary* eingetragen, da wir hier den sekundären IdP benutzen werden. Das *issuer* Feld enthält die URL zu dem IdP, also zu *login.persona.org*. Der *disabled* Parameter zeigt im Falle eines primären IdP an, ob dieser seinen Dienst eingestellt hat. *NormalizedEmail* enthält die Mailadresse, die der Benutzer angegeben hat. Die darin enthaltenen Mailadresse ist auf eine Normalform gebracht, um Probleme mit exotischen Kodierungen vorzubeugen.
Der *state* Parameter gibt den aktuellen Status des Kontos bei Persona wieder. Er kann *unknown* enthalten was bedeutet, das die Mailadresse dem System noch unbekannt ist. *Unverified* steht für eine bekannte, aber noch nicht verifizierte Mailadresse. In diesem Fall enthält die Variable aber den Wert *known*, da wir bereits ein freigeschaltetes Konto bei Persona besitzen.
In Nachricht 14 wird die *bidbundle.js* Datei nachgeladen. Diese enthält die später benötigten kryptografischen Funktionen, näheres im Kapitel 3.4.

Im nächsten Schritt wird der Benutzer aufgefordert, sein Passwort einzugeben, das in Nachricht 15 übertragen wird. Diese Nachricht enthält, neben der Mailadresse und dem Passwort, noch drei weitere Werte, unter anderem den *ephemeral* Wert, der Einfluss auf die Gültigkeit des Userzertifikats hat, dazu näheres im Kapitel 3.5. Einige SP lassen auch unverifizierte Konten zu, was dem IdP in der *allowUnverified* Variable mitgeteilt wird. Der *CSRF* Parameter enthält den Wert aus Nachricht 11.
Persona setzt in Nachricht 16 nun das Session Cookie und antwortet mit dem *success* Parameter. Der *userid-* und *suppress_ask_if_computer*-Parameter werden ebenfalls im Kapitel 3.5 näher erläutert.
Mit Nachricht 17 fragt der Browser die optional zusätzlich hinterlegten Mailadressen ab, denn Persona erlaubt es, verschiedene Mailadressen in einem Konto zu bündeln. In der Antwortnachricht 19 sendet Persona den *success* Parameter und ein *emails* Array,

bestehend aus einer oder mehrere Mailadressen, zurück.
Nachricht 19 und 20 sind Wiederholungen von Nachricht 13 und 14. Sie stellen eine Finished Nachricht dar. Dies dient unter anderem zur Bestätigung, dass es sich wirklich um ein bekanntes und verifiziertes Konto handelt.
Da nun die Authentifikation gegenüber Persona abgeschlossen ist, erzeugt der Browser selbständig das Schlüsselpaar, bestehend aus einem privaten und einem öffentlichen Schlüssel. Diesen öffentlichen Schlüssel sendet der Browser in Nachricht 21 zusammen mit der Mailadresse, dem *ephemeral-*, *allowUnverified-* und *CSRF*-Parameter an Persona. Dort wird der Schlüssel zusammen mit anderen Werten signiert und als sogenanntes Userzertifikat zurückgesendet.

Phase 3 (Abb. 3.3) zeigt das Ende des Protokollablaufes. Da der Browser das Userzertifikat des IdP zu der zuvor gesendeten Mailadresse besitzt, konstruiert er die Identity Assertion, die er mit seinem privaten Schlüssel signiert.

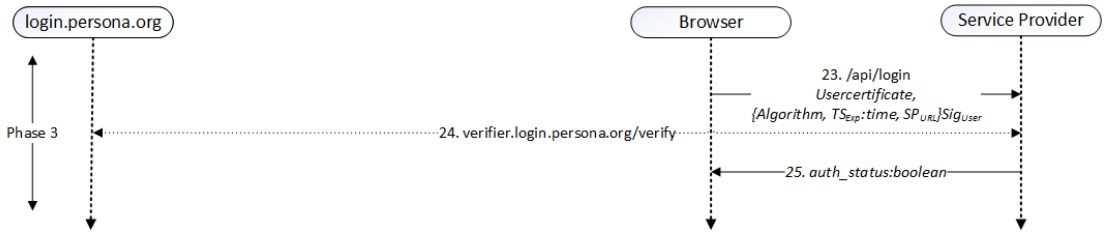

Abbildung 3.3: Protokollablauf Persona IdP Phase 3

Die Bedeutung von Parameter und Signaturen des Userzertifikat und der Identity Assertion werden im Kapitel 3.6 genau erläutert.
In der Nachricht 23 überträgt er nun die Assertion, bestehend aus Userzertifikat und der Identity Assertion, an den SP.
Der SP hat nun zwei Möglichkeiten diese Assertion zu validieren. Er kann es lokal auf seinem Server durchführen, indem er die enthaltenen Werte und Signaturen prüft. Alternativ nutzt er eine von Persona zur Verfügung gestellte Verifikationsfunktion[3]. Dazu sendet er die Assertion in Nachricht 24 an Persona.
Handelt es sich um eine valide Assertion setzt der SP in Nachricht 25 dem Browser ein gültiges Session Cookie und teilt ihm den neuen *auth_status* mit.
Ab dieser Stelle ist der Benutzer bei dem SP mit der in Nachricht 12 eingegebenen Mailadresse authentifiziert.

3.3 Primärer Identity Provider

Dieses Kapitel erklärt den Protokollablauf eines primären IdP. Dabei handelt es sich um denselben IdP der auch in Kapitel 4 näher erläutert wird. Dort werden auch weitere

[3] `https://developer.mozilla.org/en/Persona/Remote_Verification_API`

Besonderheiten erklärt, die während des Protokollablaufes auffallen. Dazu zählt das One Time Token, das anstatt eines Passwortes zum Einsatz kommt und die Sessionverwaltung mittels einer Number used once (Nonce) statt eines Sessioncookies handhabt.

Trotz allem gelten bei dieser Beschreibung wieder dieselben Voraussetzungen und Rahmenbedingungen wie im vorangegangen Kapitel. Es handelt sich wieder um einen SP-initiierten Login, bei dem der Benutzer bereits ein aktives Konto beim IdP besitzt. Es wird weiterhin davon ausgegangen, dass er einen frischen Browser, also einen Browser ohne Cookies oder Daten im Local Storage benutzt.
Der Ablauf wird in drei Phasen unterteilt. Hier ist allerdings zu beachten, das Phase 2 aus Gründen der Übersicht in weitere Teile aufgeteilt ist. Der Ablauf in voller Länge befindet sich im Anhang (Abb. A.2).

Phase 1 (Abb. 3.4) ist identisch zu der ersten Phase des Persona IdP. Der Browser baut eine Verbindung zum SP auf, lädt die *auth.js* sowie die *include.js* nach und erfragt seinen aktuellen Authentifikationsstatus.

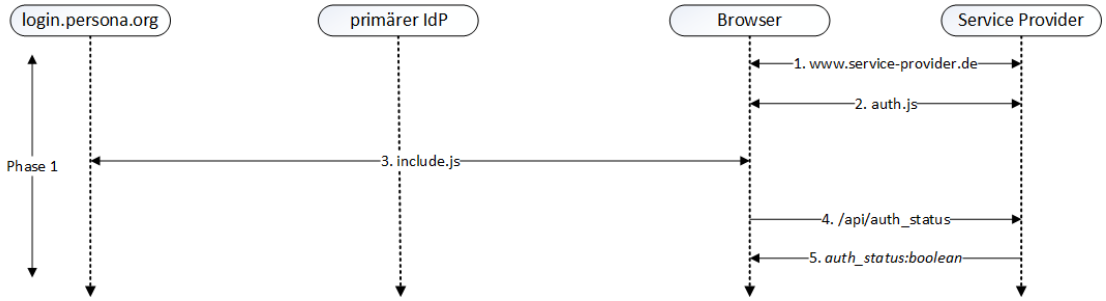

Abbildung 3.4: Protokollablauf primärer IdP Phase 1

Phase 2.1 Betätigt der Benutzer nun auf den Login Button, beginnt die nächste Phase (Abb. 3.5). Sie beschreibt die Authentifikation des Benutzers gegenüber dem primären IdP.
Dazu baut der Browser zuerst wieder eine Verbindung zum Persona Server auf, öffnet den Communication IFrame und den SignIn Popup und lädt die beiden JS Dateien *communication_iframe.js* und *dialog.js* vom Server.
In Nachricht 10 erfragt der Browser den Session Kontext zwischen sich und Persona, worauf er in der Antwort 11 die im Kapitel zuvor beschrieben Parameter zurück erhält. Gibt der Benutzer nun die Mailadresse ein, wobei es sich hierbei um eine Adresse von einem primären IdP handelt, so weicht das Protokoll von dem bekanntem Ablauf ab. Der Browser überträgt in Nachricht 12 die Mailadresse zusammen mit dem Issuer Parameter. Persona antwortet nun in Nachricht 13 mit drei Domains und den beiden bekannten Parametern *normalizedEmail* sowie *state*. Der *auth* Parameter enthält die URL zu der

Authentifikationsseite des primären IdP. Der *prov* Parameter zeigt auf die Provisioningseite des IdP, von welcher der Benutzer später sein Userzertifikat erhält. Beide Parameter erfährt Persona aus der *.well-known* URI des primären IdP. Die genaue Bedeutung und Funktion dieser URI findet sich im Kapitelteil 3.6.2. Das *issuer* Feld enthält die Domain des primären IdP.

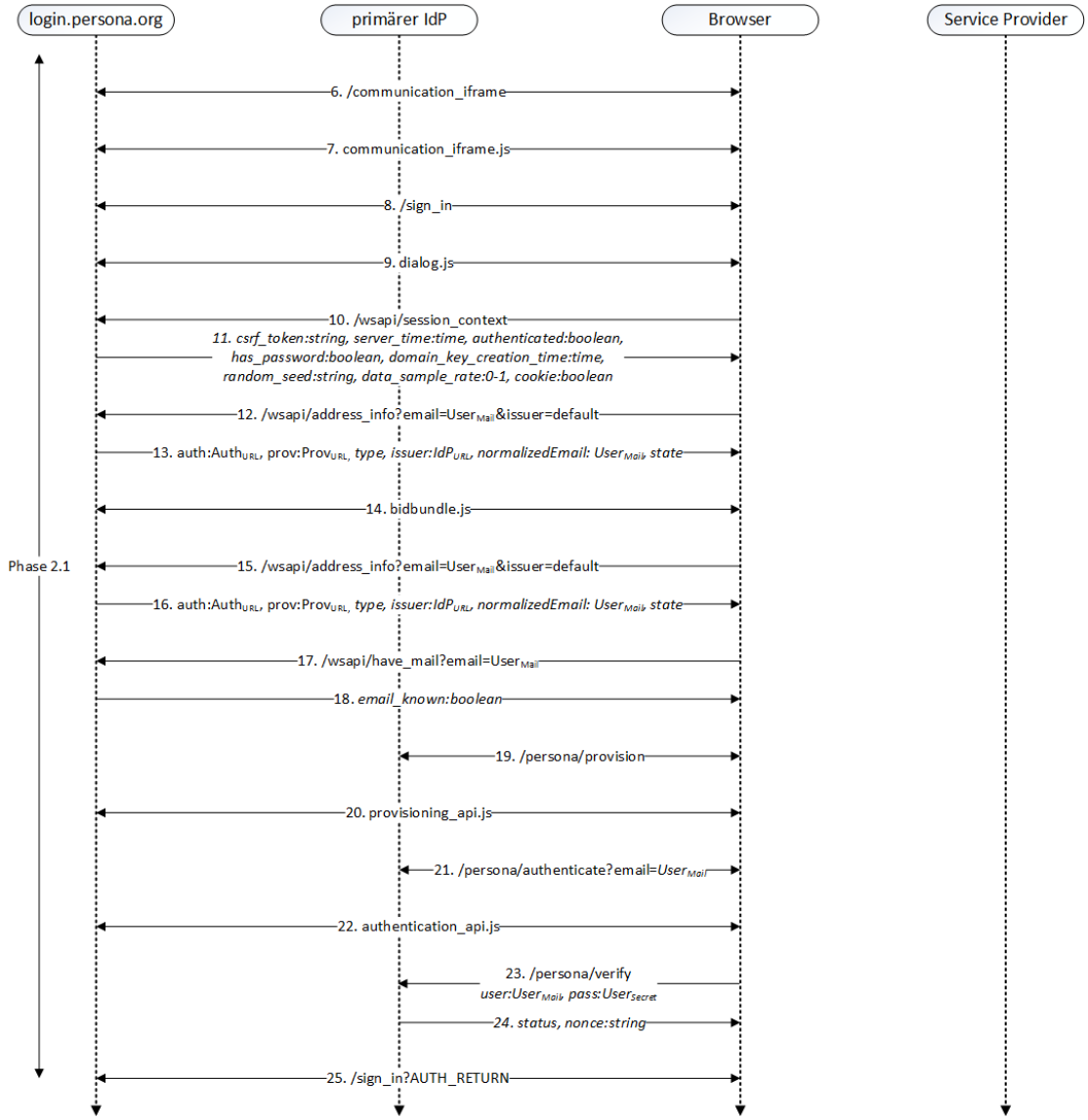

Abbildung 3.5: Protokollablauf primärer IdP Phase 2.1

In Nachricht 14 lädt der Browser selbständig die Krypto Bibliothek *bidbundle.js* nach. Nachricht 15 und 16 sind vom Browser ausgelöste Wiederholungen der Nachrichten 12 und 13.

In Nachricht 17 erfragt der Browser, ob die Mailadresse schon bekannt ist, worauf Persona mit dem boolean Wert *email_known:true* antwortet.
Ab dieser Stelle baut der Browser das erste Mal eine Verbindung zum primären IdP auf. Dazu öffnet er zuerst eine Verbindung zur Provisioningseite und lädt die *provisioning_api.js* nach. Danach teilt er in Nachricht 21 dem primären IdP die Mailadresse mit, unter der er sich authentifizieren möchte. Außerdem lädt er die *authentication.js* vom Server. Die beiden Domains zum primären IdP hat der Browser in der Nachricht 13 vom Persona erhalten.
Nun wird der Benutzer aufgefordert, sich gegenüber dem IdP zu authentifizieren. Dies tut er in Nachricht 23 mit seiner Mailadresse und einem One Time Token. Der IdP antwortet mit dem *status:okay* und einer *nonce*.

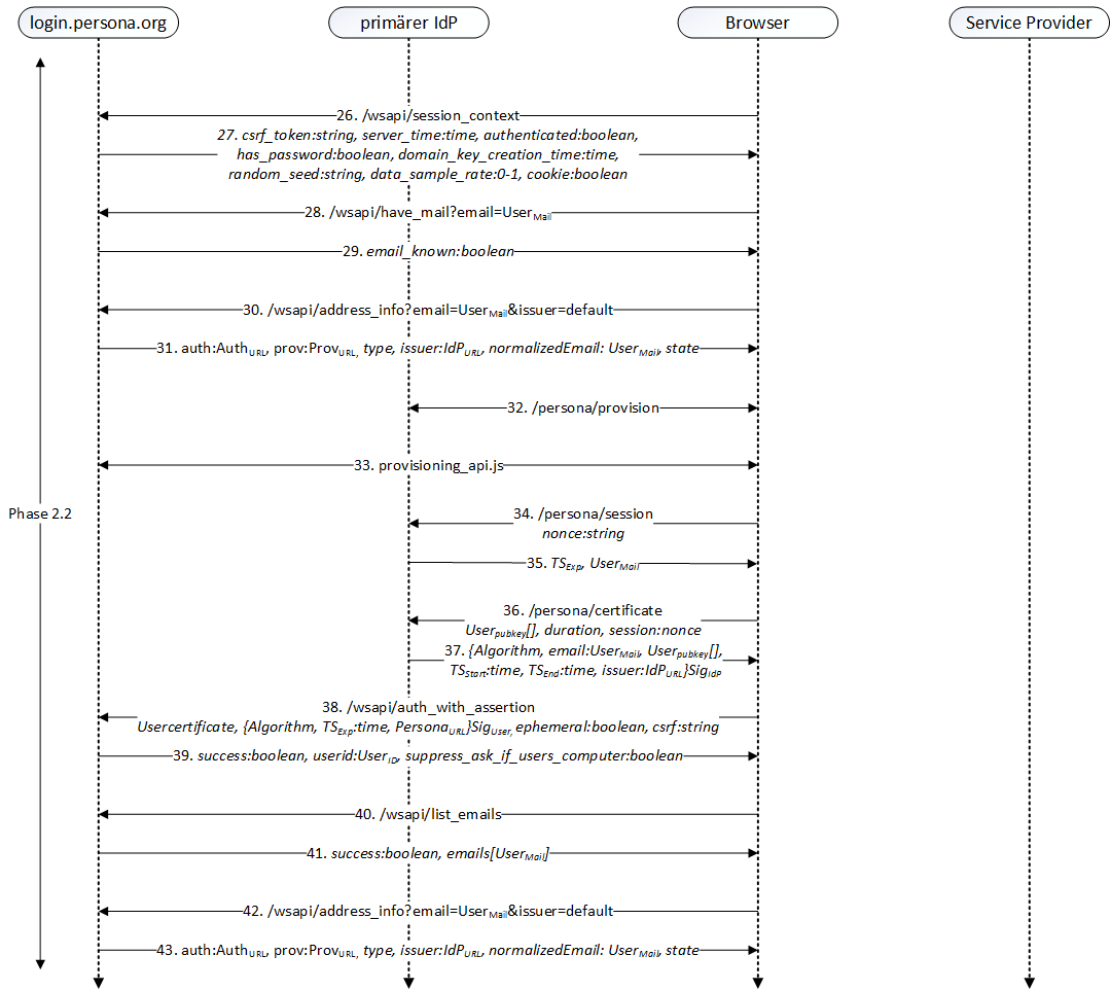

Abbildung 3.6: Protokollablauf primärer IdP Phase 2.2

Sollte die Authentifikation erfolgreich verlaufen sein, so teilt der Browser dieses Persona in Nachricht 25 mit. Sollte ein Problem aufgetreten sein, sendet der Browser stattdessen

/sign_in?AUTH_RETURN_CANCEL und der Benutzer wird erneut zur Eingabe seiner Mailadresse aufgefordert.

Phase 2.2 Da der Benutzer nun gegenüber seinem primären IdP authentifiziert ist, beginnt Phase 2.2. In dieser Phase (Abb. 3.6) stellt der primäre IdP dem Benutzer das Userzertifikat aus.
Dazu wiederholt der Browser selbständig Teile aus Phase 2.1. Darunter befindet die Session Kontext Nachricht, die Abfrage ob Persona die Mailadresse bekannt ist und die Address_Info Abfrage.
Der Sinn dieser Wiederholungen besteht darin, dass Persona die Authentifikationphase sowie die Provisioningphase getrennt behandelt. Zum weiteren Verständnis findet sich im Anhang ein alternatives Ablaufdiagramm A.3 zum EyeDee.Me IdP[4] . Der wichtige Punkt bei diesem primäre Test-IdP ist, dass er ein IdP-initiierten Login unterstützt. Dies hat zur Folge, dass bei diesem IdP die Phase 2.1 entfällt da der Benutzer sich im Vorfeld authentifiziert. Außerdem handhabt er die Session mittels Cookie.

Nachdem der Browser in Nachricht 32 noch einmal die Provisioningseite und die *provisioning_api.js* geladen hat, bestätigt er in Nachricht 34 seine aktive Session. Dazu sendet er die *nonce*, die er nach der Authentifikation erhalten hat. Der primäre IdP antwortet zur Bestätigung mit dem darin enthalten Timestamp und der Mailadresse.

In Nachricht 36 fordert der Browser das Userzertifikat an. Zu diesem Zweck sendet er seinen öffentlichen Schlüssel, sowie die Dauer (*duration*) für die das Zertifikat gültig sein soll. Als Nachweis über die aktive Session dient der *session* Parameter, der erneut die *nonce* enthält. Der IdP antwortet mit dem Userzertifikat, welches in Kapitel 3.6 näher erläutert wird.

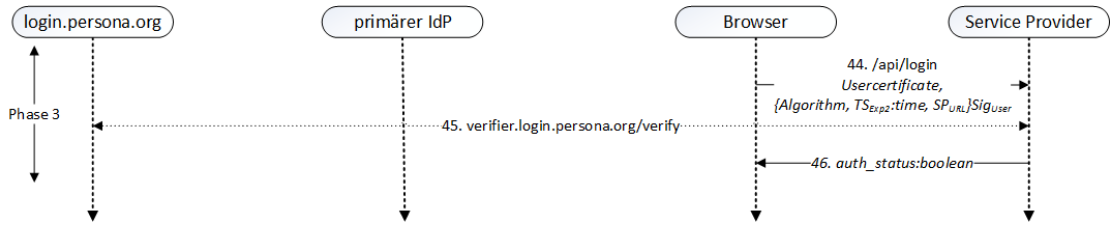

Abbildung 3.7: Protokollablauf primärer IdP Phase 3

In Nachricht 38 authentifiziert sich der Browser mit dem Userzertifikat und einer Identity Assertion gegenüber Persona. Dies hat den Zweck, dass der User so auf die zentrale Nutzerverwaltung zugreifen kann, in welcher er zum Beispiel weitere Mailadressen hinterlegt hat. Diese Mailadressen werden dann auch in Nachricht 40 und 41 abgefragt und es folgen die aus dem letzten Kapitel bekannten Finished Nachrichten 42 und 43.
An dieser Stelle ist der Benutzer gegenüber seinem IdP und Persona authentifiziert und

[4] `https://eyedee.me`

besitzt sein Userzertifikat vom primären IdP, womit er sich bei einem beliebigen SP anmelden kann.

Phase 3 Genau diesen Fall beschreibt die dritte Phase. Diese Phase (Abb. 3.7) ist wieder identisch zur Phase 3 des Persona IdP.
Der User sendet in Nachricht 44 sein Userzertifikat und eine Identity Assertion an den SP. Dieser Verifiziert beide wahlweise lokal, oder sendet sie in Nachricht 45 an Persona. Sollte alles korrekt sein, setzt der SP in Nachricht 46 das Sessioncookie und teilt dem Browser den neuen *auth_status* mit.

Eine Erkenntnis aus der Protokollanalyse ist, dass, sollte der Benutzer selber einen primären IdP betreiben, der Ablauf stark vereinfacht und gekürzt werden könnte.
Hierfür wäre ein Programm oder eine Browsererweiterung denkbar, welche die Aufgaben des primären IdP und des Benutzers vereint. Sie erzeugt ein Userzertifikat und die Identity Assertion mit unterschiedlichen Schlüsselpaaren und sendet die Assertion an den SP.
Die Kommunikation mit Persona kann entfallen, da sie hauptsächlich der Beschaffung der JS Dateien dient. Die Authentifikation gegenüber Persona dient zur Abfrage der möglicherweise hinterlegten Mailadressen und könnte optional erfolgen.
Allerdings ist zu beachten, dass der Benutzer trotzdem unter der im Userzertifikat eingetragenen Maildomain eine Webseite mit der *.well-known* URI betreiben müsste. Näheres findet sich im Kapitel 3.6.2.

3.4 Nachgeladene Dateien

Da Mozilla sein SSO System gerne einer breiten Masse an Benutzern zugänglichen machen möchte, und das Protokoll im Internet Anwendung findet, liegt es nahe den Browser des Benutzers als Bedienelement zu verwenden.
Da aber bereits gezeigt wurde, dass der Browser eine große Menge an Daten, Parametern und kryptografischen Operationen verarbeiten muss, hat er diese Funktionalität durch die nachgeladenen JS Dateien und Bibliotheken nachzurüsten.
Der Vorteil liegt darin, dass so praktisch jeder Browser genutzt werden kann und keine zusätzliche Funktionen durch Browsererweiterungen oder andere Programme bereitgestellt werden müssen. Dies erleichtert die Benutzung des Protokolls, da die technischen Voraussetzungen niedrig gehalten werden.
Da diese Bibliotheken teilweise sehr viele Funktionen enthalten, diese aber nicht alle essentiell für das Protokollverständnis sind, werden in diesem Kapitel nur die wichtigsten Funktionen erläutert. Eine vollständige Liste aller Funktionen findet sich nach den JS Bibliotheken getrennt im Anhang A.

include.js Möchte ein Service Provider Persona als Single Sign-on Lösung nutzen, so muss er eine Sessionverwaltung implementieren, die auf die Persona API[5] [6] zugreift. Die

[5] `https://developer.mozilla.org/de/docs/Persona/Schnellstart`
[6] `https://developer.mozilla.org/en-US/docs/Web/API/navigator.id`

include.js stellt neben dieser API verschiedene Funktionen bereit, die gewährleisten sollen, dass BrowserID mit den verschiedensten Browsern genutzt werden kann.
Dazu prüft die *include.js*, ob der Browser die nötigen Funktionen mitbringt. Dazu gehört zum Beispiel der Local Storage und ob er JavaScript Object Notation (JSON)- sowie POST-Nachrichten verarbeiten kann.
Daneben beinhaltet sie noch die Winchan Bibliothek[7]. Diese ist für die Kommunikation zwischen dem SP und den verschiedenen JS Bibliotheken zuständig; denn hier müssen Parameter über unterschiedliche Domains hinweg ausgetauscht werden.
Außerdem wird die *communication_iframe.js* sowie die *dialog.js* geladen. Dies sind die beiden Bibliotheken, die beim Öffnen des Login Popups geladen werden.

communication_iframe.js Der Communication IFrame überwacht den Loginvorgang. Er prüft, ob der Benutzer schon eingeloggt ist, synchronisiert die verschiedenen Mailadressen mit dem Persona-Backend oder händelt die abschließende Userzertifikatgenerierung. Der Kommunikation IFrame nutzt nicht die Winchan Funktionen, um mit dem SP zu kommunizieren, sondern bringt dazu die JSChannel Bibliothek[8] mit.

dialog.js Die *dialog.js* übernimmt die Aufgabe, den Benutzer durch den Login- oder auch Registrierungsvorgang zu lenken. Sollten Probleme auftreten, bringt sie Funktionen mit, um die Werte in den Local Storage des Browser schreiben zu können oder löst entsprechende Fehlermeldungen aus. Daher ist sie auch die Bibliothek, die sämtliche *wsapi*-Nachrichten, also sämtliche Kommunikation zwischen Browser und dem Persona IdP, sendet und verarbeitet.
Nach dem Senden der Mailadresse ist sie es auch, welche das Laden der CryptoLoader Funktion die *bidbundle.js* Bibliothek startet.

bidbundle.js Diese *bidbundle.js* bringt die nötigen kryptographischen Funktionen, die während des Protokollablaufs benötigt werden. Da Mozilla sich hier für das JSON Format entschieden hat, werden diese Kryptooperationen mit der eigenen *jwcrypto*-Bibliothek [2] verarbeitet. Näheres findet sich in den kommenden Kapitel 3.6 und der Sicherheitsanalyse 5.
Wird auf einen primären IdP zurückgegriffen, sind noch weitere JS Bibliotheken von Bedeutung. Dazu zählt die *authentication_api.js* sowie die *provisioning_api.js*. Auch wenn diese beiden JS Dateien bei der Kommunikation mit einem primären IdP zum Einsatz kommen, werden sie trotzdem vom Persona Server geladen. Dies soll sicherstellen, dass bei kurzfristigen Protokolländerungen die primären IdPs trotzdem funktionstüchtig bleiben.

authentication_api.js Die *authentication_api.js* stößt die Authentifikation an und teilt dem Persona-Server mit, wie die Authentifikation bei primären IdP verlaufen ist. Dazu

[7] `https://github.com/mozilla/winchan`
[8] `https://mozilla.github.io/jschannel/docs`

sendet sie die *AUTH_RETURN* oder im Fehlerfall die *AUTH_RETURN_CANCEL* Nachricht. Näheres findet sich in Nachricht 25 in Abbildung 3.5.

provisioning_api.js Da die Authentifikation von der Userzertifikatgenerierung im Protokollablauf getrennt ist, benötigt der Browser noch die *provisioning_api.js*. In der Provisioningphase fordert der Browser das Userzertifikat an.
Die *provisioning_api.js* bringt neben der zuvor genannten JSChannel Bibliothek die Funktionen zur Schlüsselerzeugung und Zertifikatsanfrage mit.

browserid.js Eine letzte JS Datei ist noch zu nennen, obwohl sie in der bisherigen Protokollanalyse nicht aufgetaucht ist. Bei dieser handelt es sich um die *browserid.js*, die nur geladen wird, falls die `login.persona.org` Webseite direkt aufgerufen wird. Sie ist fast identisch zur *dialog.js*, allerdings fehlen ihr einige Funktionen, die nur bei der Kommunikation mit Service Providern benötigt werden. Im Gegenzug besitzt sie aber mehre Funktionen, die nur auf der Persona Webseite zum Einsatz kommen. Dazu zählt unter anderem der Account Manager, oder die Funktion um Passwörter zurückzusetzten. Die genauen Unterschiede zur *dialog.js* finden sich ebenfalls im Anhang A.

3.5 Local Storage

BrowserID nutzt den sogenannten Web Storage oder auch DOM Storage des Browsers. Dies ist ein Speicherbereich im Browser, welche Webanwendungen nutzen können um persistent Daten im Browser abzulegen. Dieser Web Storage war Teil der HTML5 Spezifikation [3] und wird von praktisch allen aktuellen Browsern unterstützt.
Der Web Storage ist in zwei Speicherbereiche aufgeteilt. Im Session Storage werden die Daten beim Schließen des Browser gelöscht. Der Local Storage hingegen speichert die Daten dauerhaft, so dass sie auch bei den nächsten Aufrufen des Browsers noch abrufbar sind. BrowserID nutzt genau diesen Local Storage, um Daten[9] im Browser zu speichern. Dabei handelt es um folgende Werte:

- **emailToUserID** Hier wird festgehalten, welche UserID welchen Mailadressen zuzuordnen ist.
- **emails** Hält fest, wann sich der Benutzer mit welcher Mailadresse angemeldet hat und zu welchem Zeitpunkt die letzte Änderung auftrat. Daneben wird hier der öffentliche und private Schlüssel sowie das codierte Userzertifikat hinterlegt.
- **interaction_data** Enthält Daten die Persona für Performancemessungen nutzt. Näheres findet sich bei dem *data_sample_rate* Parameter.
- **returnTo** Hier wird die Domain hinterlegt, zu welcher der Benutzer zurückkehren soll, falls seine Mailadresse noch nicht verifiziert ist.

[9] `https://github.com/mozilla/persona/wiki/Front-End-Development#persisting-data-on-the-client`

- **storageCheck** Enthält einen Boolean Wert, welcher prüft ob der Communication IFrame lesend auf den Local Storage zugreifen kann.
- **managePage** Hier wird vermerkt, wenn der Benutzer die Benutzerverwaltung auf login.persona.org aufgerufen hat.
- **siteInfo** Enthält zu jedem SP getrennt drei Informationen: die Mailadresse, den Issuer und die Adresse, mit welcher er sich eingeloggt hat.
- **userComputer** Hier wird nach der UserID getrennt gespeichert, zu welchem Zeitpunkt der Benutzer das letzte Mal aktiv war sowie der dazugehörige *state* Paremeter.

Der *state* Parameter aus der *userComputer* Variable muss näher erläutert werden, da dieser direkten Einfluss auf den Protokollablauf hat.
Nach einer gewissen Anzahl an Loginvorgängen wird der Benutzer gefragt (Abb. 3.8), wie lange er die aktuelle Session behalten möchte. Er hat dann die Wahl zwischen einer flüchtigen Session oder einer dauerhaften für einen Monat.

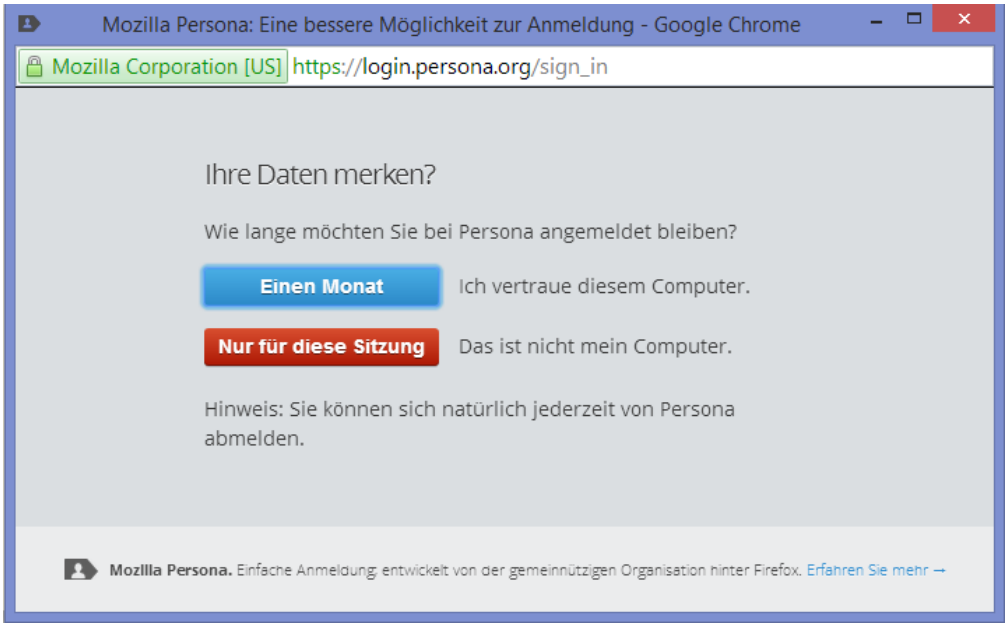

Abbildung 3.8: Abfrage zu Sessiondauer

Wählt er hier die längerfristige Session, so wird der *state* Parameter von *seen* auf *confirmed* geändert. Dies hat zur Folge, dass der Browser bei seiner nächsten Anfrage für ein neues Userzertifikat den *ephemeral* Wert auf *false* setzt.
Dieselbe Änderung des *state* Parameters kann auch der IdP herbeiführen. Dazu sendet er in der *suppress_ask_if_user_computer* Variablen ein *true*.
Die Änderung des *ephemeral* Parameters hat wiederum Auswirkungen auf die Gültigkeitsdauer des Userzertifikats; denn sendet der Browser ein *ephemeral:false* an den IdP

bedeutet dies, dass der Benutzer einer längeren Sessiondauer zugestimmt hat, weshalb er die Gültigkeit des Userzertifikats von einer Stunde auf 24 Stunden erhöht.

3.6 Wichtige Konstrukte

In diesem Kapitel werden Eigenschaften des Protokolls erläutert, welche für die Funktionalität und Sicherheit von BrowserID unerlässlich sind.

3.6.1 Assertion

Gegen Ende eines jeden Protokolldurchlaufs sendet der Browser eine Assertion an den SP. Anhand dieser Assertion kann der SP überprüfen, ob es sich bei der Anfrage um einen legitimen Benutzer handelt, dem er Zugriff auf seine Dienste gewähren möchte. Dazu muss er die in der Assertion enthaltenen Werte überprüfen. Die Integrität der Werte wird durch zwei digitale Signaturen sichergestellt, die er ebenfalls validieren sollte.
Wichtig ist, dass die Assertion aus zwei Teilen besteht. Zum einen aus dem Userzertifikat, das vom IdP generiert und signiert wird und zum anderen aus der Identity Assertion. Die Identity Assertion wird vom Browser sowohl erzeugt als auch signiert.
BrowserID verwendet für die Formatierung das JSON Format. Dabei greift das Protokoll auf die Javascript Object Signing and Encryption (JOSE)[10]- und JSON Web Key (JWK)-Spezifikationen [4] zurück.
Dadurch sind beide Teile, also das Userzertifikat und die Identity Assertion, wie folgt aufgebaut: Beide beginnen mit dem Signaturalgorithmus, gefolgt von den Daten. Der dritte Teil enthält den eigentlichen Signaturwert. Alle Teile sind durch einen Punkt voneinander getrennt. In der Assertion werden beide durch das Tilde (~) Satzzeichen concatiniert und an den SP gesendet.
Bei der Übertragung ist noch zu beachten, dass die Daten Base64URL-encodet werden, nicht zu verwechseln mit dem Base64 Encoding.

Userzertifikat

Wie zuvor erwähnt, wird das Userzertifikat vom IdP erzeugt. Es dient dazu, die Identität des Benutzer gegenüber eines SPs zu beglaubigen.
Das Userzertifikat enthält folgende Parameter:

- **alg** In dieser Variablen wird das Signaturverfahren hinterlegt. Es wird später vom SP benötigt, um die Signatur zu validieren.
- **public-key** Hier wird der öffentliche Schlüssel des Browser gespeichert. Er wird später vom SP genutzt, um die Signatur der Identity Assertion zu überprüfen. Typischerweise enthält dieses Feld mehrere Werte, darunter den Algorithmus sowie verschieden Schlüsselparameter, die der Signaturalgorithmus benötigt.
- **principal** Der Principal zeigt auf das nachfolgende *email* Feld.

[10] `http://datatracker.ietf.org/wg/jose/charter/`

- **email** Diese Variable enthält die Mailadresse des Benutzers auf den das Userzertifikat ausgestellt wird.
- **iat** Enthält einen Unix Timestamp des Zeitpunkts der Erstellung.
- **exp** Hält ebenfalls im Unix Format den Zeitpunkt fest, an dem das Userzertifikat ungültig wird.
- **iss** Hier wird die Domain des IdP hinterlegt. Diese Domain wird vom SP genutzt, um den öffentlichen Schlüssel abzurufen, der für die Signaturvalidierung benötigt wird.
- **Signatur** Enthält den Signaturwert des Userzertifikats.

Das Userzertifikat speichert der Browser im Local Storage, da es, solange der Timestamp im *exp* Parameter nicht überschritten wird, für alle SP Gültigkeit besitzt.

Identity Assertion

Die Identity Assertion wird vom Browser erzeugt und signiert, um die Integrität sicherzustellen.
Während das Userzertifikat eine Gültigkeitsdauer von 24 Stunden besitzen kann, ist diese bei einer Identity Assertion auf fünf Minuten beschränkt. Dies soll Replay Angriffe erschweren.
Die eigentliche Funktion einer Identity Assertion ist, eine Verbindung zwischen Userzertifikat und dem Browser bzw. Benutzer herzustellen. Diese Beziehung wird durch die Signatur der Identity Assertion gewährleistet, da der öffentliche Schlüssel zur Überprüfung im Userzertifikat hinterlegt ist.

- **alg** In dieser Variable wird das Signaturverfahren hinterlegt, welches später vom SP benötigt wird, um die Signatur zu validieren.
- **exp** Legt den Zeitpunkt im Unix Format fest, wann die Identity Assertion abläuft.
- **aud** Hier trägt der Browser das Protokoll, die Domain und den Port des SP ein, an den die Assertion geschickt wird.
- **Signatur** Enthält den Signaturwert der Identity Assertion.

3.6.2 Well-Known URL

Im vorangegangen Kapitel wurde gezeigt, dass die Assertion zwei Signaturen besitzt. Die Identity Assertion wird vom Browser signiert. Zur Überprüfung muss der SP den öffentlichen Schlüssel des Browsers aus dem Userzertifikat extrahieren. Möchte er allerdings die Integrität des Userzertifikats überprüfen, benötigt er den öffentlichen Schlüssel des IdP. Da dieser aber nicht in der Assertion enthalten ist, muss er sich diesen über einen anderen Weg beschaffen.
Dazu bedient sich Persona der Well-Known Uniform Resource Identifiers (URIs)-Spezifikation[5].

Diese erlaubt es - unter einer zuvor spezifizierte URL - Daten abzulegen. BrowserID spezifiziert[11] dazu den *.well-known/browserid* Pfad den jeder IdP anbieten muss. Im Falle des Persona-IdP wäre das zum Beispiel `https://login.persona.org/.well-known/browserid`.
Unter dieser URL finden sich im Standardfall folgende JSON-formatierte Werte:

- **public-key** Dieses Feld enthält den öffentlichen Schlüssel des IdP.
- **authentication** Enthält den Pfad, unter den sich der Benutzer gegenüber dem IdP authentifizieren kann.
- **provisioning** Enthält den Pfad, unter dem der Browser das Userzertifikat erhält.

Davon abweichend kann ein IdP auch seine Tätigkeit an eine abweichende Domain delegieren. Dazu bietet er nicht die oben genannten Parameter unter diesem Pfad an, sondern ersetzt diese durch den *authority* Parameter, der die geänderte URL enthält.

Möchte ein IdP seinen Dienst komplett einstellen kann er dies tun, indem er den *disabled* Parameter in der well-known Datei hinterlegt. Dies wird auch dem Browser mitgeteilt, indem der *disabled:true* Parameter in der */wsapi/address_info* Nachricht gesendet wird.

[11] `https://developer.mozilla.org/en-US/Persona/.well-known-browserid`

4 Realisierung eines primären IdP

In diesem Kapitel geht es um die Realisierung und Implementierung eines primären IdPs. Da das Konzept des primären IdP ein zentralen Eckpunkt im BrowserID Protokoll ist und für die spätere Sicherheitsanalyse neue Möglichkeiten eröffnet, war die Umsetzung eines eigenen IdP ein wichtiger Teil dieser Arbeit.
Ziel dieses IdPs ist es, zum Protokollverständnis beizutragen und das Testen neuer Angriffsvektoren auf das Single Sign-On Protokoll zu ermöglichen.

4.1 Verwendete Bibliothek

Um nicht einen IdP von Grund auf selbst implementieren zu müssen, diente eine minimal Implementierung als Grundgerüst. Dabei handelt es sich um den Persona-TOTP[1] IdP. Dieser, in Python geschriebene, IdP unterstützt SP-initiierten Logins für eine zuvor festgelegte Mailadresse.
Das genaue Verhalten sowie die einzelnen Nachrichten, die ein primärer IdP generiert, wurden schon im vorangegangenem Kapitel 3.3 erläutert. Trotzdem wird an dieser Stelle auf einige Besonderheiten dieses IdP eingegangen.
Untypisch bei dieser Implementierung ist, dass sie zur Authentifikation kein Passwort, sondern ein One Time Token verlangt. Dabei handelt es sich um eine Spezifikation [6], die zeitbasierte Einmalpasswörter erzeugt. Diese Spezifikation wird auch von einigen Webseiten zur 2Faktor-Authentifikation eingesetzt. Allerdings wird an dieser Stelle nicht weiter darauf eingegangen, da erwähnte Funktion für diese Arbeit durch eine normale passwortbasierte Authentifikation ersetzt wurde. Der Grund dafür liegt darin, dass für die Token weitere Software benötigt wird und jedes Token nur 30 Sekunden gültig ist. Dies hätte die Arbeit mit dem IdP unnötig verkompliziert.
Der zweite Punkt, der bei diesem IdP auffällt ist, dass er die Session nicht mittels eines Cookies, sondern einer signierten Nonce im Browser speichert.
Diese Nonce enthält, genau wie Cookies, eine Gültigkeitsdauer. Sie beträgt bei diesem IdP 30 Tage. Daneben wird noch die Mailadresse des Benutzers festgehalten, an welcher der IdP die Nonce zuordnen kann. Beide Werte werden vom IdP signiert und vom Browser im Local Storage gespeichert. Verlangt der Browser nun das Userzertifikat, muss er erst durch das Senden der Nonce bestätigen, dass er bereits authentifiziert ist. Im nächsten Schritt validiert der IdP die Nonce und stellt das Userzertifikat aus.

[1] `https://github.com/djc/persona-totp`

4.2 Eigene Erweiterungen

Der verwendete Persona-TOTP IdP bringt zwar alle Funktionen mit, um Anhand dieser den Protokollablauf eines primären IdP zu untersuchen, allerdings reicht diese reguläre Funktionalität nicht aus, um verschiedene Angriffe durchzuführen. Aus diesem Grund wurde er um einige Funktionen erweitert.
Die erste Änderung war, wie bereits zuvor erwähnt, dass die Token-basierte Authentifikation durch eine einfache passwortbasierte Authentifikation ersetzt wurde.
Des Weiteren sollten die verschiedenen Parameter manipulieren werden können, welcher der IdP im Userzertifikat an den Browser sendet. Dabei sollen aber der Aufbau und die Signatur des Userzertifikats erhalten bleiben.
Um die späteren Testszenarien einfacher konfigurieren zu können, sollten die Parameter über ein Graphical User Interface (GUI) während des Loginvorganges änderbar sein. Als Resultat (Abb. 4.1) lassen sich nun die IdP Domain, die Mailadresse des Benutzers sowie der Gültigkeitszeitraum des Zertifikats beliebig manipulieren. Zusätzlich lässt sich der öffentliche Schlüssel des Benutzers durch einen ungültigen Schlüssel ersetzen oder das gesamte Userzertifikat mit einem ungültigen Schlüssel signiere.

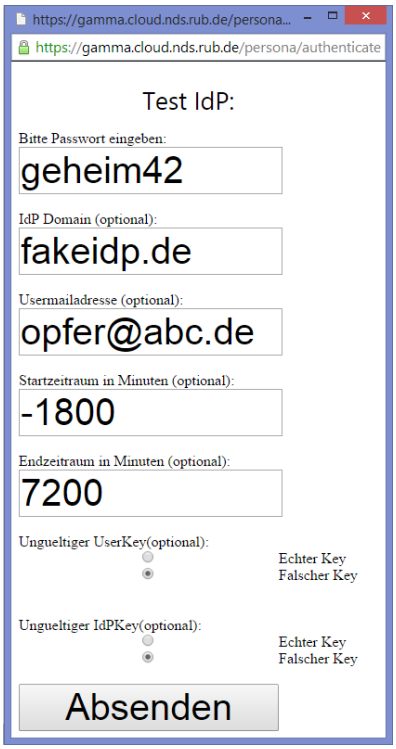

Abbildung 4.1: Erweiterungen des Test IdP

Die verschiedenen Tests, die damit durchgeführt werden können, werden im folgenden Kapitel 5 vorgestellt.

4.3 Anwendungsbeispiele

Dieses Kapitel zeigt den praktischen Nutzen des vorgestellten IdPs. Dazu wird unter anderem ein möglicher Angriff simuliert.

Doch zunächst muss erläutert werden, wie der erweiterter IdP in den regulären Protokollablauf aus Kapitel 3.3 eingreift.
Dazu sollte man sich zuerst noch einmal das Userinterface (Abb. 4.1) des IdP ins Gedächtnis rufen. Dort ist zu erkennen, dass die Parameter parallel zum Passwort, also während der Authentifikation eingegeben werden. Diese zusätzlichen Parameter werden, neben dem Passwort, in Nachricht 23 (Abb. 3.5) an den primären IdP gesendet. Dieser speichert diese vorerst temporär zwischen, da sie erst in der folgenden Phase wieder von Bedeutung sind.
In der Nachricht 36 (Abb. 3.6) fordert der Browser das Userzertifikat vom IdP an. Darauf hin generiert der IdP das Userzertifkat und sendet es in Nachricht 37 an den Browser.
Hier ist zu beachten, das alle Werte, bis auf den *Algorithm* Parameter, manipuliert werden können. Dazu müssen aber, während der Passwortabfrage, entsprechende Änderungen erfolgt sein. Wurden ein oder mehrere Felder nicht ausgefüllt oder geändert, trägt der IdP die regulären Werte ein.

Die Abbildungen (Abb. 4.2) zeigen zwei mögliche Konfigurationen des Test IdP.

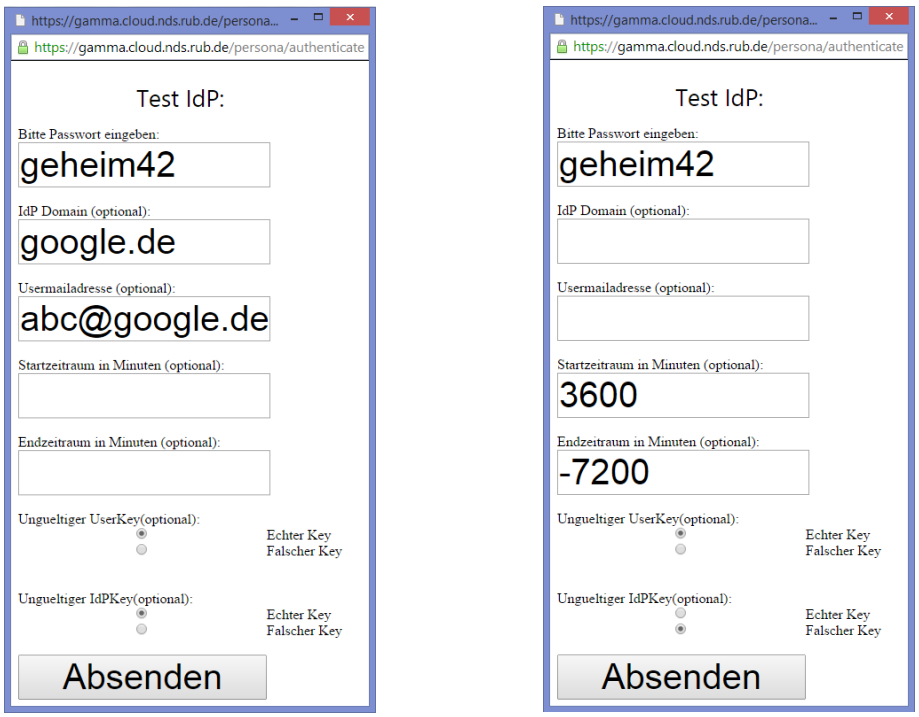

Abbildung 4.2: Mögliche IdP Konfigurationen

Die linke Konfiguration zeigt einen möglichen Angriff. Details finden sich im Kapitel 5.2. In dieser Konfiguration erstellt der IdP ein Userzertifikat für den google.de IdP und einem dazu gehörigem User. Diesen möglichen Angriff sollte der SP daran erkennen, das er die Signatur nicht mit dem öffentlichen Schlüssel von google.de validieren kann; denn der passende Schlüssel liegt unter der Domain des IdP.

Die zweite Konfiguration zeigt weniger einen konkreten Angriff. Hier geht es eher darum, wie sich ein SP oder dessen Validierungsfunktion bei einem fehlerhaften Userzertifikat verhält. Dazu wurde zum einem der Ausstellungszeitpunkt des Zertifikats um eine Stunde in die Zukunft und zum anderen der Ablaufzeitpunkt um zwei Stunden in die Vergangenheit verlegt. Zusätzlich wurde das Userzertifikat mit einem falschen Schlüssel signiert. Es ist dem SP also nicht möglich, den passenden öffentlichen Schlüssel von der *.well-known* URL des IdP zu erhalten.

5 Sicherheitsanalyse

In diesem Kapitel werden sicherheitsrelevante Aspekte des BrowserID Protokolls untersucht und aufgezeigt. Außerdem werden neue Angriffe untersucht, die mit dem zuvor implementierten IdP getestet werden.
Schließen wird dieses Kapitel mit einigen Anwendungsbeispielen die auf dem eigenem IdP basieren.

5.1 Protokollanalyse

Das eigentliche Protokoll lässt sich in beiden Varianten, also mit primärem oder sekundärem IdP, auf folgende, relevante Nachrichten reduzieren:

1. Browser	$\longrightarrow$	IdP:	$Mail_{User}, PubKey_{User}$
2. IdP	$\longrightarrow$	Browser:	$\{Mail_{User}, URL_{IdP}, T_{exp}, T_{iat}, PubKey_{User}\}_{Sig_{IdP}}$
3. Browser	$\longrightarrow$	SP:	$\{URL_{SP}, T_{exp2}\}_{Sig_{User}}$, $\{Mail_{User}, URL_{IdP}, T_{exp}, T_{iat}, PubKey_{User}\}_{Sig_{IdP}}$
4. IdP	$\longrightarrow$	SP:	$PubKey_{IdP}$

Diese Nachrichten zeigen den eigentlichen Protokollkern, der das BrowserID Protokoll ausmacht. Dabei ist zu beachten, dass nach der Spezifikation von Persona die Nachrichten 1,2 und 4 verschlüsselt übertragen werden sollen.
In der Nachricht 1 sendet der Browser, nach der Authentifikation gegenüber dem IdP, die Mailadresse und den öffentlichen Schlüssel des Benutzers an den IdP. Der IdP antwortet in Nachricht 2 mit einem durch seine Signatur geschütztem Userzertifikat. Dies enthält die Mailadresse des Benutzers, die eigene URL und den öffentlichen Schlüssel des Benutzers. Daneben enthält das Userzertifikat noch zwei Timestamps. Dabei handelt es sich um den Austellungs- sowie Ablaufzeitpunkt des Zertifikats.
Nachricht 3 wird vom Browser an den SP gesendet und enthält das zuvor empfangene Userzertifikat sowie die Identity Assertion. Diese Assertion wird vom Browser erzeugt und signiert. Darin enthalten ist die Domain des SP und der Ablaufzeitpunkt der Identity Assertion.
In Nachricht 4 lädt der SP den öffentlichen Schlüssel des IdP von dessen .well-known URI.

Möchte ein Angreifer nun das Protokoll angreifen, müsste er die Nachrichten manipulieren. Das würde aber voraussetzen, dass er in der Lage ist, einen Man-in-the-Middle (MitM)-Angriff durchzuführen und die Transportverschlüsselung zu brechen.

Eine Arbeit von Forschern der Universität Rice [7] hat bereits gezeigt, dass unter dieser Annahme zwei Angriffe möglich sind.
Im ersten Angriff wäre der Angreifer in der Lage, die Identität seines Opfers zu übernehmen. Dazu würde er in der ersten Nachricht den öffentlichen Schlüssel des Opfers durch seinen eigenen ersetzten. Das hätte zur Folge, dass der IdP ein Userzertifikat erzeugt, in welchem die Mailadresse des Opfers mit dem öffentlichen Schlüssel des Angreifers kryptografisch gebunden ist. Jetzt erzeugt der Angreifer, mit seinem privaten Schlüssel, beliebige Identity Assertion zu einem gewünschten SP. Das versetzt den Angreifer in die Lage, zusammen mit dem manipulierten Userzertifikat, sich als sein Opfer auszugeben.

Beim zweiten Angriff handelt es sich um einen Replay Angriff. Dazu fängt der Angreifer die dritte Nachricht ab und sendet sie erneut. Bleibt er dabei im dem Zeitraum, der von den beiden Timestamps T_{exp} und T_{exp2} vorgegeben ist, so ist er wieder im Namen seines Opfer gegenüber dem SP authentifiziert.
Eine mögliche Lösung für diesen Replay Angriff wäre ein Challenge-Response Parameter zwischen dem SP und dem Benutzer. Würden die beiden zum Beispiel eine Nonce aushandeln, könnte der Benutzer diese in der Identity Assertion mitsenden. Der SP könnte den Angriff daran erkennen, dass, sobald er eine zweite Assertion mit dieser Nonce erhält, ein Problem vorliegen muss.

Eine Arbeit [8] von Forschern der Stanford Universität zeigt einen weiteren Angriffsvektor. Sie gehen von einem direkten Angriff auf den IdP Server aus. Denn sollte es einem Angreifer gelingen, die .well-known/browserid Datei zu manipulieren, kann er dort auf seinen eigenen IdP verweisen. Dazu nutzt er die Delegation-Funktion die BrowserID mitbringt.

Ein weiteren wichtigen Punkt, den es zu beachten gibt ist, dass es sich bei BrowserID um ein dezentrales Protokoll handelt. Das bedeutet, dass auch der Angreifer in der Lage wäre einen IdP zu betreiben. Daraus folgt, dass er tiefer in den Protokollablauf eingreifen kann als es bei zentralen SSO Protokollen möglich wäre. Genau für dieses Szenario wurde der in Kapitel 4 vorgestellte IdP entsprechend erweitert. Mit ihm ist es möglich, die verschiedenen Protokollparameter zu manipulieren, um zu untersuchen, wie sich das Protokoll oder ein SP in diesem Fall verhält. Die konkreten Tests finden sich im folgendem Kapitel 5.2.

5.2 Untersuchungspakete

Ein möglicher Angriff, den man mit einem primären IdP durchführen kann, wäre ein Userzertifikat für eine Dritt-Domain auszustellen. Als Beispiel dient ein Angreifer, der unter attacker.com einen primären IdP betreibt. Er stellt ein Zertifikat für opfer@google.de aus. Um diesen Angriff zu erkennen, muss der SP zwingend prüfen, ob die Mailadresse opfer@google.de zum Issuer Feld attacker.com passt. Da attacker.com nicht befähigt ist, Userzertifikate für google.de auszustellen, würde er das Zertifikat ablehnen.

Im nächsten Schritt könnte der Angreifer dazu übergehen, das Issuer Feld ebenfalls zu manipulieren. Dazu stellt er ein Userzertifikat für opfer@google.de aus und trägt im Issuer Feld ebenfalls google.de ein.
In diesem Fall reicht es nicht, wenn der SP die beiden Felder vergleicht. Um den Angriff zu erkennen, muss er die Signatur des Userzertifikats überprüfen, wofür er den öffentlichen Schlüssel des IdP benötigt. Da er dazu die Domain im Issuer Feld aufruft, lädt er den falschen Schlüssel; denn nur der Schlüssel von attacker.com könnte die Signatur validieren.
Damit der Angriff trotzdem erfolgreich verläuft, müsste der Angreifer wieder einen MitM-Angriff gegen den SP durchführen. Dazu müsste er die verschlüsselten Verbindung zwischen dem SP und google.de unterbrechen und durch seinen eigenen öffentlichen Schlüssel ersetzten.
Diese theoretischen Angriffe zeigen, wie im Kapitel zuvor, das die Sicherheit von BrowserID elementar vom HTTPS und dem dahinter liegenden Secure Sockets Layer (SSL)- beziehungsweise Transport Layer Security (TLS)-Protokoll abhängt.

5.3 Weitere Sicherheitsaspekte

Dieses Kapitel geht weniger auf konkrete Angriffe ein. Stattdessen werden unterschiedliche Mechanismen untersucht, die für die Sicherheit des Systems von Bedeutung sind.
Hier sind, wie wir bereits gesehen haben, die Signaturen, die bei dem Userzertifikat und der Identity Assertion zum Einsatz kommen, von elementarer Bedeutung. Dazu gehören auch die Schlüsselpaare, die der Browser und der IdP erzeugen.
Zur Erzeugung dieser Signaturen und Schlüssel setzt Mozilla die eigene *jwcrypto* Bibliothek [2] ein. Diese wiederum basiert auf den Standarts der JSON Web Key Spezifikation [4] in Verbindung mit den JSON Web Algorithms (JWA) [9]. Aus dieser Reihe an Algorithmen bedient sich BrowserID dem RSA-Algorithmus. Daneben wird noch der Digital Signature Algorithm (DSA) genutzt. Bei beiden handelt es sich um erprobte kryptografische Verfahren. Allerdings ist kritisch anzumerken, dass auch RSA-Schlüssel von der Länge 1024 Bit zugelassen[1] sind. Diese entsprechen nicht mehr den aktuellen Empfehlungen für diesen Algorithmus.

Ein weiterer möglicher Schwachpunkt von BrowserID kann sein, dass praktisch das ganze Protokoll in JavaScript abläuft. In Kapitel 3.4 wurde bereits gezeigt, dass eine Reihe von JavaScript Dateien geladen werden. Darunter befindet sich auch die *bidbundle.js* Bibliothek, die das Schlüsselpaar des Benutzers erzeugt. Dieses Vorgehen soll das Protokoll möglichst unabhängig von den verschiedenen Browsern machen, vergrößert aber die Angriffsfläche; denn Angreifer könnten hier zum Beispiel wieder einen MitM-Angriff durchführen und manipulierte Bibliotheken an den Benutzer ausliefern. Diese manipulierten Bibliotheken könnte fehlerhafte kryptografische Funktionen beinhalten was dazu führt, dass der Browser schwache Schlüssel erzeugt oder einen Fehler bei dem Signaturvorgang macht. Hier wäre es wünschenswert, wenn BrowserID auf die jeweils

[1] `https://github.com/mozilla/jwcrypto/blob/master/lib/algs/rs.js`

browsereigenen kryptografie Funktionen zugreifen würde. Als Beispiel ist hier die Network Security Services Bibliothek[2] von Mozilla zu nennen. Diese wäre schon im Firefox Browser enthalten und übernimmt dort alle kryptografischen Operationen.
Einen weiteren Punkt den es zu beachten gilt ist, dass das Schlüsselpaar des Benutzers im Local Storage des Browsers abgelegt wird. Da es dort unter der Domain des IdP hinterlegt ist, können andere Webseiten nicht darauf zugreifen; denn die Daten sind durch die Same-Origin-Policy (SOP) geschützt. Sobald sich aber auf der Seite des IdP eine Cross-Site-Scripting (XSS) Lücke befindet, wären diese sensiblen Daten zugänglich.
Genau aus diesen Gründen gibt Mozilla entsprechende Sicherheitsempfehlungen[3] heraus. Darunter das strikte Einhalten der HTTPS Verbindungen, umsetzen von Gegenmaßnahmen zu CSRF-Angriffen und die Erschwerung von XSS Angriffe durch Einführung einer Content Security Policy (CSP).

5.4 Sicherheit der Service Provider

Der dritte Teilnehmer im BrowserID Protokoll ist der SP. Es wurde zuvor gezeigt, dass die Sicherheit des Protokolls von der sicheren Übertragung der Nachrichten sowie der korrekten Überprüfung der Assertion abhängt. Da dem SP die Aufgabe zukommt, die Assertion zu validieren, gibt ihm Persona hier zwei Möglichkeiten.
Die erste Möglichkeit, dass der SP die Assertion selbst validiert. Hier sollte er zwingend darauf achten, dass er dies lokal auf seinem Server tut und die zuvor beschriebenen Überprüfungen vornimmt. Da zu diesen Prüfungen auch die Signaturvalidierung zählt, muss er darauf achten, dass er den öffentlichen Schlüssel des IdP über eine verschlüsselte Verbindung abruft, da dieser sonst manipuliert werden könnte.
Die zweite Möglichkeit der Assertion Überprüfung ist, diese an den Verifikationsservice von Persona zu senden. Persona bietet unter `https://verifier.login.persona.org/verify` genau diesen Service an. Dazu sendet der SP die Assertion, bestehend aus Userzertifikat und Identity Assertion, an diese Domain. Persona übernimmt dann die Verifikation und teilt dem SP das Resultat mit. Auch hier ist zwingend darauf zu achten, dass diese Verbindung verschlüsselt ist, da sonst ein Angreifer die Antwort von Persona zu seinen Gunsten manipulieren könnte.

Da zum Zeitpunkt dieser Arbeit leider alle getesteten Service Provider Personas Verifikationservice nutzten, wurden keine praktischen Angriffe gegen dritte Verifikationsbibliotheken durchgeführt. Allerdings wurde im Rahmen dieser Arbeit, neben der IdP Implementierung, eine Wordpress Installation mit BrowserID Unterstützung realisiert, welche für spätere Tests erweitert werden kann.

[2] `NetworkSecurityServices`
[3] `https://developer.mozilla.org/en-US/Persona/Security_Considerations`

6 Zusammenfassung

Dieses Kapitel gibt eine Zusammenfassung über die gesamte Arbeit und zeigt Fragestellungen und Projekte für nachfolgende Arbeiten auf.

6.1 Fazit

Die Zielsetzung dieser Arbeit war es, vorbereitend auf die abschließende Sicherheitsanalyse, das Protokoll zu untersuchen. Das Resultat dieses Untersuchung ist eine ausführliche Dokumentation des Protokollablaufes.
Eine Besonderheit des BrowserID Protokoll ist, dass es zwei Arten von Identity Providern gibt. Um diese Eigenschaft, also die Existenz von primären sowie sekundären Identity Providern, auch in der Untersuchung gerecht zu werden, war die Implementierung eines primären IdPs ein wichtiger Teil dieser Arbeit; denn auch wenn der eigene IdP für die Sicherheitsanalyse erweitert wurde, so trug er trotzdem zum eigentlichen Protokollverständnis und der daraus resultierenden Dokumentation bei.
Aufbauend auf dieser Dokumentation ist es nun möglich, bekannte und auch neue Angriffsvektoren zu erkennen oder zu ermitteln. Sollen diese Angriffsvektoren im nächsten Schritt praktisch getestet werden, so kommt wieder eigene IdP oder auch der Wordpressblog mit BrowserID Unterstützung zum tragen. An diesen beiden lassen sich mögliche Angriffe testen und gegeben falls nachvollziehen.
Der letzte Teil dieser Arbeit besteht daraus, eine Sicherheitsanalyse für das gesamte Protokoll zu erstellen. In diese Analyse flossen alle zuvor gewonnen Erkenntnisse ein. Außerdem kamen hier die selbst implementierten Erweiterungen des IdP zum Einsatz. Diese erlauben es, die verschiedenen Parameter eines Userzertifikat zu manipulieren, um zu untersuchen, wie sich das Protokoll oder die SP darauf hin verhalten.
Als Fazit der Sicherheitsanalyse lässt sich festhalten, dass das Protokoll in der aktuellen Fassung als sicher angesehen werden kann. Allerdings muss auch gesagt werden, dass diese Sicherheit von sehr vielen kleinen Faktoren abhängt. Das hat zur Folge, sollte sich ein Protokollteilnehmer nicht an die aktuellen Empfehlungen und Vorgaben von Mozilla halten, die Sicherheit in keinster Weise mehr gewährleistet ist. An dieser Stelle sollte und könnte Mozilla noch einiges tun, um das BrowserID Protokoll widerstandsfähiger gegen mögliche Angriffe zu gestalten.

6.2 Zukünftige Arbeiten

Zu Beginn der Arbeit wurde die BigTent Erweiterung des BrowserID Protokolls erwähnt. Dabei handelt es sich um eine Schnittstelle, die es erlaubt existierende GMail- und Yahoo-

Konten innerhalb des Protokolls zu benutzen. Da eine Forschergruppe der Universität Trier bereits gezeigt hat, dass in diesem Protokollteil Probleme [10] auftraten und er in dieser Arbeit keine Berücksichtigung gefunden hat, wäre dies ein Thema, dass es noch zu untersuchen gilt.
Die Protokollanalyse hat auch gezeigt, dass BrowserID zwar sehr komplex erscheint und bei der praktischen Nutzung viele Nachrichten versendet werden. Allerdings hat sich am Ende ergeben, dass der Protokollkern auf wenige Nachrichten reduziert werden könnte. Darauf aufbauend könnte man ein Programm oder eine Browsererweiterung entwickeln, welche den Großteil der Kommunikation einspart und Benutzer mit IdP vereint. Dies würde ein automatisierteres Testen, aus der Perspektive eines Angreifers der Benutzer und IdP vereinen möchte, ermöglichen.
Da zum Zeitpunkt dieser Arbeit leider keine der getesteten Anwendungen eine eigene Verifikationsfunktionen nutzte, konnten diese Verifikationsfunktionen nicht näher untersucht werden. Sollten andere Verifikationsbibliotheken, neben der von Persona, an Relevanz gewinnen, wäre dies ein guter Ansatzpunkt für weitere Tests. Denn nur durch eine fehlerfreie Verifikation der Assertion ist die Sicherheit des Protokolls gewährleistet.

A Anhang

Übersicht zu Persona IdP

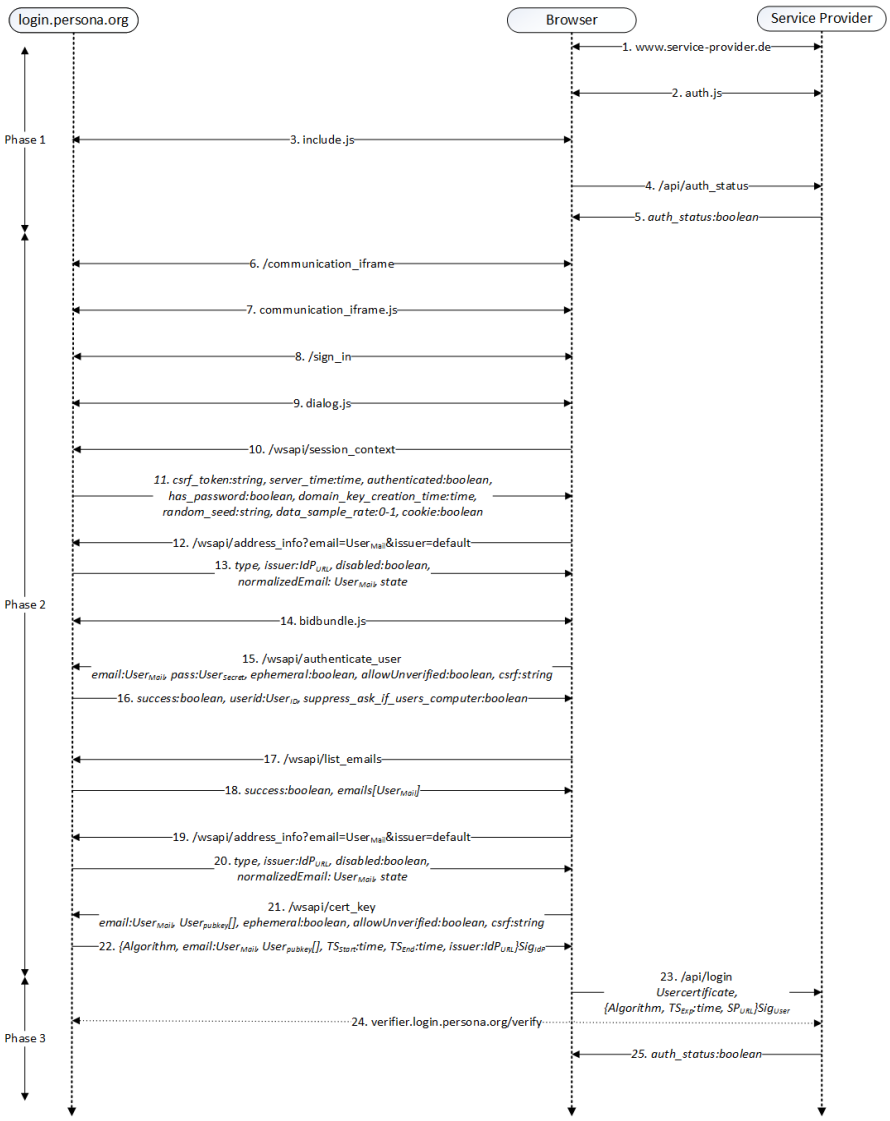

Abbildung A.1: Protokollablauf sekundären IdP von Persona

Übersicht zu primärem IdP

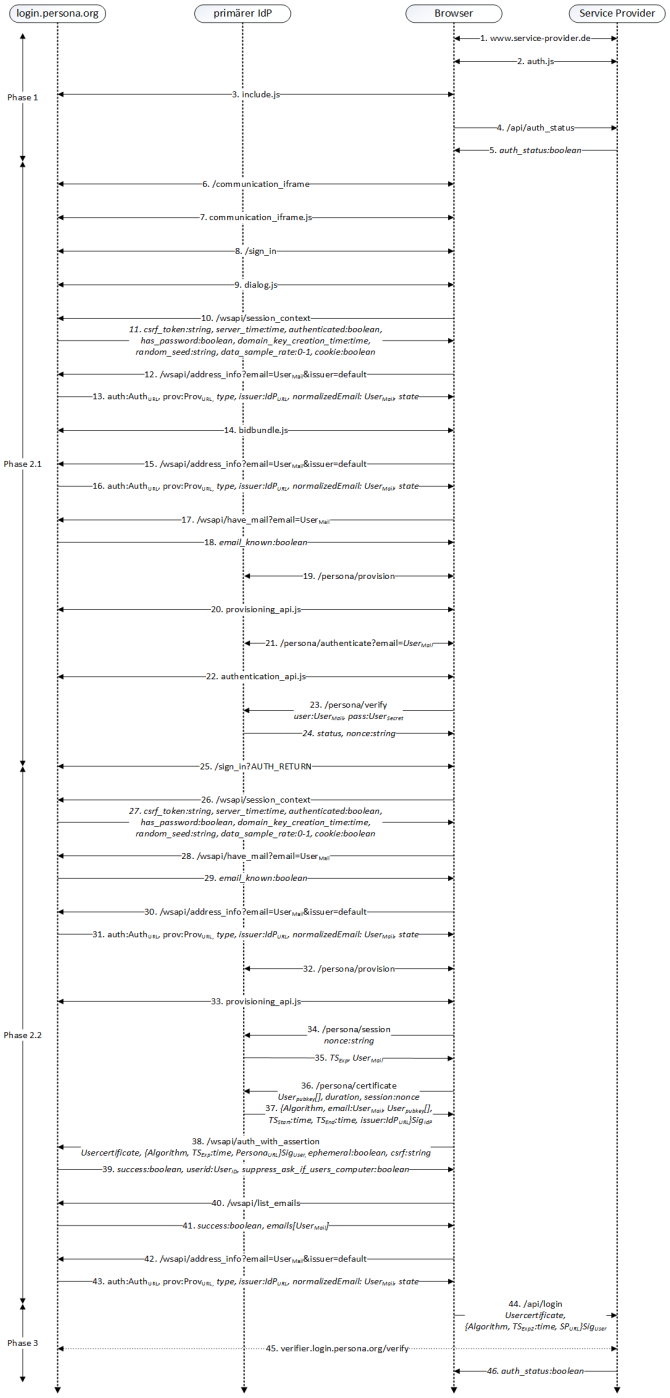

Abbildung A.2: Protokollablauf primärer IdP

Übersicht zu EyeDee.Me IdP

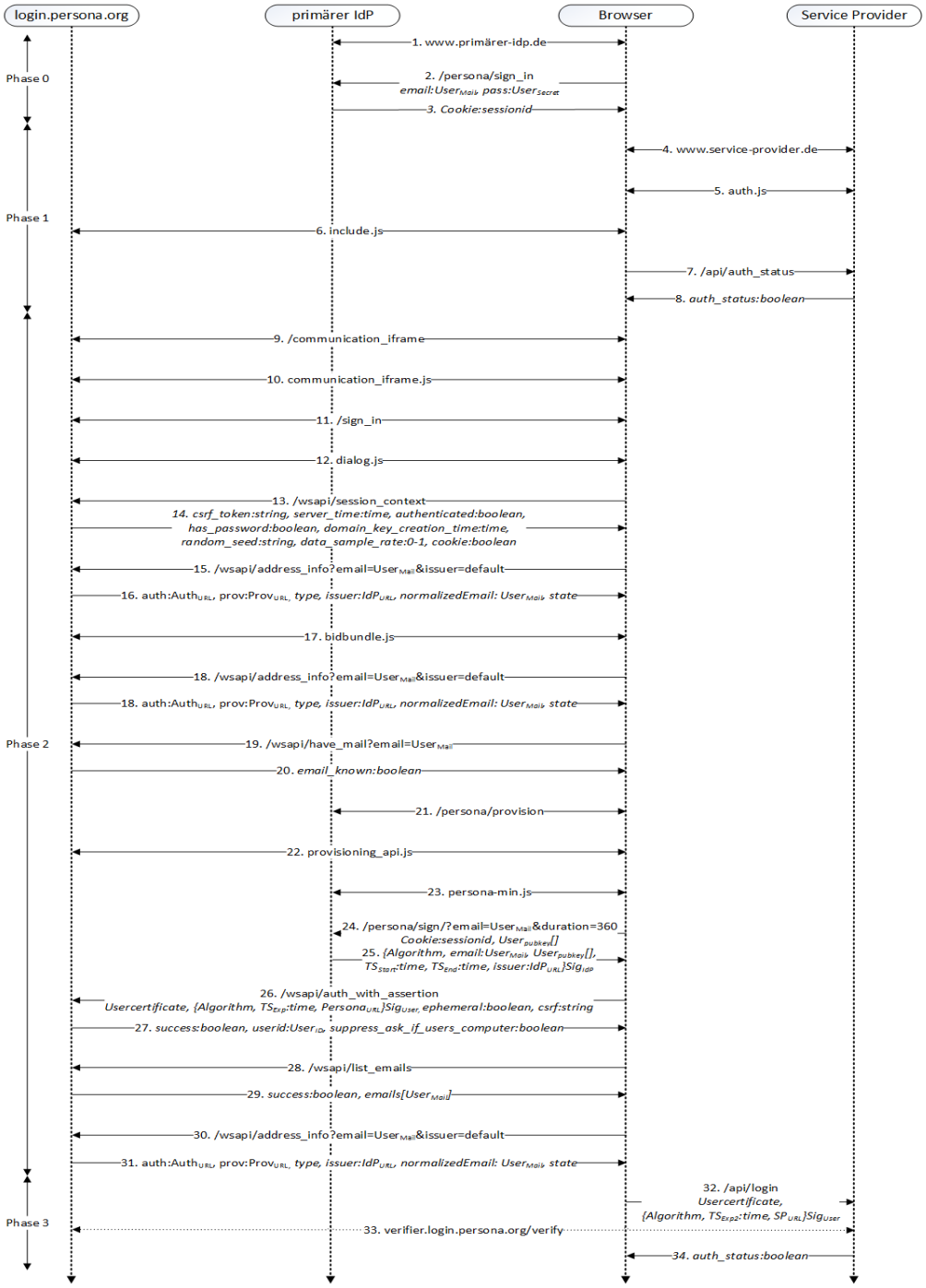

Abbildung A.3: Protokollablauf EyeDee.Me IdP

communication_iframe.js

```
var Channel = function()
BrowserID.CODE_VERSION
Hub
window.Micrajax
BrowserID.Mediator
BrowserID.getStorage
BrowserID.Storage
BrowserID.Class
BrowserID.XHRTransport = window.Micrajax
BrowserID.Modules = BrowserID.Modules
BrowserID.Modules.Module
BrowserID.Modules.XHR
BrowserID.Models
BrowserID.Models.UserContext
BrowserID.Models.NetworkContext
BrowserID.Models.RpInfo
BrowserID.Models.InteractionData
BrowserID.Network
BrowserID.CryptoLoader
BrowserID.Provisioning
BrowserID.User
```

dialog.js

```
WinChan
windows.Micrajax
BrowserID.CODE_VERSION
var json_load_data
Hub = function()
BrowserID.DOM
BrowserID.module
var Channel
BrowserID.Templates
BrowserID.Templates.add_email
BrowserID.Templates.cannot_verify_required_email
BrowserID.Templates.complete_sign_in
BrowserID.Templates.confirm_email
BrowserID.Templates.development
BrowserID.Templates.error
BrowserID.Templates.generic
BrowserID.Templates.inline_tospp
```

```
BrowserID.Templates.invalid_required_email
BrowserID.Templates.is_this_your_computer
BrowserID.Templates.load
BrowserID.Templates.pick_email
BrowserID.Templates.primary_offline
BrowserID.Templates.primary_user_not_verified
BrowserID.Templates.rp_info
BrowserID.Templates.rp_info_mobile
BrowserID.Templates.set_password
BrowserID.Templates.test_template_cachify
BrowserID.Templates.test_template_no_input
BrowserID.Templates.test_template_with_input
BrowserID.Templates.test_template_with_partial
BrowserID.Templates.verify_primary_user
BrowserID.Templates.wait
BrowserID.Renderer
BrowserID.Class
BrowserID.Mediator
BrowserID.Tooltip
BrowserID.Validation
BrowserID.DOMHelpers
BrowserID.Screens
BrowserID.BrowserSupport
BrowserID.EnableCookiesURL
BrowserID.Wait
BrowserID.Errors
BrowserID.getStorage
BrowserID.Storage
BrowserID.XHRTransport
BrowserID.Modules.Module
BrowserID.Modules.XHR
BrowserID.Models
BrowserID.Models.UserContext
BrowserID.Models.NetworkContext
BrowserID.Models.RpInfo
BrowserID.Network
BrowserID.CryptoLoader
BrowserID.Provisioning
BrowserID.User
BrowserID.Modules.DOMModule
BrowserID.Modules.PageModule
BrowserID.Modules.XHRDelay
BrowserID.Modules.XHRDisableForm
BrowserID.Modules.CookieCheck
```

```
BrowserID.Modules.Development
BrowserID.Modules.ExtendedInfo
```

Bis zu dieser Stelle enthält die *dialog.js* die selben Funktionen wie die *browserid.js*. Der untere Teil weicht davon ab und ist nur in der *dialog.js* enthalten.

```
BrowserID.Command
BrowserID.History
BrowserID.StateMaschine
BrowserID.Models.InteractionData
BrowserID.Modules.InteractionData
BrowserID.State
BrowserID.Modules.Action
BrowserID.Modules.Dialog
BrowserID.Modules.Authenticate
BrowserID.Modules.CheckRegistration
BrowserID.Modules.PickEmail
BrowserID.Modules.AddEmail
BrowserID.Modules.VerifyPrimaryUser
BrowserID.Modules.ProvisionPrimaryUser
BrowserID.Modules.PrimaryUserProvisioned
BrowserID.Modules.PrimaryUserNotProvisioned
BrowserID.Modules.PrimaryOffline
BrowserID.Modules.GenerateAssertion
BrowserID.Modules.IsThisYourComputer
BrowserID.Modules.SetPassword
BrowserID.Modules.RPInfo
BrowserID.Modules.InlineTosPp
BrowserID.Modules.CompleteSignIn
BrowserID.Modules.ValidateRpParams
```

browserid.js

Dies sind die Funktionen, die neben dem gemeinsamen Teil mit der *dialog.js* nur in der *browserid.js* enthalten sind.

```
BrowserID.PageHelper
BrowserID.verifySecondaryAddress
BrowserID.resetPassword
BrowserID.manageAccount
BrowserID.about
```

Abbildungsverzeichnis

Acronyms

CSP Content Security Policy. 29

CSRF Cross-Sit Request Forgery. 8

DSA Digital Signature Algorithm. 28

GUI Graphical User Interface. 23

HTTP Hypertext Transfer Protocol. 7

HTTPS Hypertext Transfer Protocol Secure. 7

IdP Identity Provider. 4

JOSE Javascript Object Signing and Encryption. 19

JS JavaScript. 7

JSON JavaScript Object Notation. 16

JWA JSON Web Algorithms. 28

JWK JSON Web Key. 19

MitM Man-in-the-Middle. 26

Nonce Number used once. 11

RP Relying Party. 3

SOP Same-Origin-Policy. 29

SP Service Provider. 3

SSL Secure Sockets Layer. 28

SSO Single Sign-On. 5

TLS Transport Layer Security. 28

URIs Well-Known Uniform Resource Identifiers. 20

XSS Cross-Site-Scripting. 29

Literaturverzeichnis

[1] Mike Hanson. A verified email protocol for the browser @ONLINE. http://www.open-mike.org/entry/verified-email-protocol, June 2011.

[2] Mozilla. Javascript implementation of json web signatures, json web tokens, and json web certificates @ONLINE. https://github.com/mozilla/jwcrypto.

[3] Ian Hickson. Web storage @ONLINE. http://www.w3.org/TR/webstorage/, July 2013.

[4] M. Jones. Json web key (jwk) @ONLINE. http://tools.ietf.org/html/draft-ietf-jose-json-web-key-00, January 2012.

[5] E. Hammer-Lahav M. Nottingham. Defining well-known uniform resource identifiers (uris) @ONLINE. https://tools.ietf.org/html/rfc5785, April 2010.

[6] M. Pei und J. Rydell D. M'Raihi, S. Machani. Totp: Time-based one-time password algorithm @ONLINE. http://tools.ietf.org/html/rfc6238, May 2011.

[7] Michael Dietz und Dan S. Wallach. Hardening persona – improving federated web login @ONLINE. http://www.internetsociety.org/sites/default/files/08_2_0.pdf.

[8] Connor Gilbert und Laza Upatising. Formal analysis of browserid/mozilla persona @ONLINE. http://web.stanford.edu/~lazau/BrowserIDPersona.pdf.

[9] M. Jones. Json web algorithms (jwa) @ONLINE. http://tools.ietf.org/html/draft-ietf-jose-json-web-algorithms-00, January 2012.

[10] Ralf Küsters und Guido Schmitz Daniel Fett. An expressive model for the web infrastructure definition and application to the browserid sso system @ONLINE. http://infsec.uni-trier.de/publications/paper/FettKuestersSchmitz-arXiv.1403.1866-2014.pdf, April 2014.