PiratenID: Technische Dokumentation

Jan Schejbal

Dieses Dokument beschreibt das PiratenID-System sowie das darin verwendete Protokoll. Es ist relevant für Personen, die die PiratenID-Software nachvollziehen, analysieren oder modifizieren wollen, für Entwickler, welche einen eigenen Client programmieren möchten, sowie für technisch Interessierte, welche einen Blick hinter die Kulissen werfen möchten. Gewöhnliche Nutzer des ID-Systems benötigen dieses Dokument nicht und sollten stattdessen die Anleitung für Nutzer[1] lesen. Entwickler von Webanwendungen, die das PiratenID-System mittels fertiger Libraries nutzen möchten, müssen dieses Dokument nicht lesen, können dies aber tun, um einen Einblick in Sicherheitsmaßnahmen bei Webanwendungen und ein besseres Verständnis des Systems zu erhalten. Für Entwickler von Webanwendungen existiert eine separate Anleitung[2], die die wichtigsten Punkte zusammenfasst.

[1] TODO
[2] TODO

Inhaltsverzeichnis

1 Einführung

Das PiratenID-System soll es Mitgliedern der Piratenpartei ermöglichen, ihren Mitgliedschaftsstatus gegenüber von Dritten betriebenen Webseiten nachzuweisen, ohne dass Zugriff auf die Mitgliederdatenbank nötig ist und ohne mehr Daten als nötig offenlegen zu müssen.

An einem Authentifizierungsvorgang sind immer drei Parteien beteiligt: (1) Der Nutzer bzw. sein Browser, (2) der Server des ID-Systems und (3) die Webanwendung, beispielsweise ein von Dritten betriebenes Forum oder Umfragetool. Um Verwirrung zu vermeiden, wird der Begriff „Client“ hier nicht verwendet, da dieser sowohl den Browser als auch die Webanwendung (Client des ID-Systems) bezeichnen könnte. Die Parteien werden im Folgenden als ID-System, Browser und Webanwendung bezeichnet.

2 Authentifizierung

Statt eines ursprünglich geplaten Protokolls wird OpenID 2.0 [3] (mit gewissen Einschränkungen) mit Attribute Exchange 1.0 [4] verwendet. Die serverseitige Implementierung ist eine vollständige Eigenentwicklung, da die Libraries eine zu hohe Komplexität haben, um sie vollständig zu reviewen, und eine kurze Durchsicht einiger Libraries Zweifel an der Sicherheit weckte. [5] Dabei wurde nur ein Teil des Protokolls implementiert, wodurch viele potentielle Quellen für Fehler und Sicherheitslücken wegfallen. Die Kompatibilität mit gängigen OpenID-Libraries bleibt gewahrt, bei der Verwendung solcher Libraries ist aber auf mögliche Sicherheitsprobleme zu achten. Zur Übersicht über die Überlegungen zu anderen Protokollen wird hier auf das englischsprachige Dokument „Building an authentication system under strict real-world constraints“ [6] verwiesen.

Die Verwendung von OpenID statt einem eigenen Protokoll ermöglicht Kompatibilität mit bestehenden Anwendungen, ist im Fall einer Kompromittierung des Servers weniger problematisch (Sicherheit basiert auf SSL-Zertifikat, dieses kann separat z. B. auf einem Reverse-Proxy gelagert werden, Revocation des Zertifikats möglich falls Clients auf Revocation prüfen, Angreifer müsste zusätzlich MitM „in den Tiefen des Internets“ durchführen), macht es aber leichter, kaputte Implementierungen zu erstellen (z. B. unzureichende Zertifikatsprüfung) und kann je nach Implementierung (Liste der akzeptierten Zertifikate/CAs) gegen kompromittierte/bösartige Zertifizierungsstellen (CAs) anfällig sein.

Sämtliche OpenID-Transaktionen finden ausschließlich über HTTPS statt. Es ist darauf zu achten, dass Webanwendungen HTTPS korrekt einsetzen (z. B. Zertifikatsprüfung),

[3] http://openid.net/specs/openid-authentication-2_0.html
[4] http://openid.net/specs/openid-attribute-exchange-1_0.html
[5] Beispiel: Die PHP-OpenID-Library von Janrain greift bei fehlendem curl auf fopen zurück, was SSL-Zertifikate nicht prüft.
[6] http://www.janschejbal.de/arbeiten/authenticationsystem11.pdf

da über die HTTPS-Verbindung die Daten des Benutzers übertragen werden. Zur Erhöhung der Sicherheit sollen Webanwendungen zusätzlich den Fingerprint des Zertifikats prüfen, um gegen Angriffe auf Zertifizierungsstellen geschützt zu sein. Associations werden nicht unterstützt, ebensowenig ein Fallback auf OpenID 1.1.

Webanwendungen, die gewöhnliche OpenID-Libraries einsetzen, müssen sicherstellen, dass Logins nur mit dem ID-System und nicht mit beliebigen fremden OpenID-Servern möglich sind. Zusätzlich muss sichergestellt sein, dass die Library SSL korrekt nutzt und wenn irgendwie möglich nur das manuell eingepflegte Zertifikat des ID-Systems akzeptiert.

Bei der OpenID-Anfrage unterliegt das Realm einer recht strengen Prüfung, es muss sich um eine HTTPS-URL auf einen Ordner handeln, genauer: Die URL muss mit einem Schrägstrich enden und darf keine Query-Parameter oder Anker enthalten. Sowohl die anfragende Seite als auch die Return-URL müssen innerhalb dieses Realms liegen. Das Realm wird zur Berechnung des Pseudonyms verwendet und dem Nutzer als der Name der Seite, bei der er sich anmeldet, angezeigt.

2.1 Authentifizierungsmodi

Bei der Authentifizierung über PiratenID werden mehrere Arten unterschieden. Bei der OpenID-Anfrage kann das Feld claimed_id/identity entweder auf die Konstante `http://specs.openid.net/auth/2.0/identifier\_select` für identifier_select gestellt oder das Feld ganz weggelassen werden. (Andere Werte werden nicht akzeptiert.) In ersterem Fall wird eine pseudonyme Anmeldung durchgeführt, d.h. das ID-System berechnet das seitenspezifische Pseudonym und übermittelt eine URL der Form
`https://id.piratenpartei.de/openid/pseudonym.php?id=4e2b9b5163[...]faced7`
wobei der Teil nach „id=“ das Pseudonym ist.

In letzterem Fall wird eine anonyme Anmeldung durchgeführt: Die Anwendung erfährt bestimmte Informationen über den Nutzer (z. B. die Tatsache, dass er Pirat ist), aber nichts, womit sie den Nutzer wiedererkennen könnte. Dies könnte z. B. benutzt werden, um Zugang zu Informationen oder Diensten auf Piraten zu beschränken. Sollen Doppelaccounts verhindert oder Benutzer wiedererkannt werden, muss die pseudonyme Anmeldung benutzt werden.

Unabhängig davon, ob die Anmeldung anonym oder pseudonym stattfindet, wird die Piratenmitgliedschaft des Nutzers entweder implizit oder explizit geprüft: Wird das Attribut „mitgliedschaft-bund“ **nicht** über Attribute Exchange (siehe nächter Abschnitt) abgefragt findet eine implizite Prüfung statt - nur Piraten mit gültiger Mitgliedschaft (und korrekt eingetragenem Token) können sich anmelden, alle anderen bekommen nur eine Fehlermeldung. Wird es jedoch explizit abgefragt, kann sich jeder anmelden und die Anwendung muss/kann Piraten und Nichtpiraten selbst trennen, z. B. um ehemaligen Mitgliedern noch Zugriff auf ihre Accounts zu ermöglichen.

Wird mit einer gewöhnlichen OpenID-Library ohne besondere Optionen eine Authentifizierung durchgeführt, so findet eine **pseudonyme** Anmeldung mit **impliziter** Prüfung der Mitgliedschaft statt, was für die meisten Fälle die richtige Wahl ist. Anwendungen, die OpenID bereits unterstützen, können so in der Regel mit minimalen Veränderungen (Beschränkung auf den PiratenID-Server, Zertifikate) direkt verwendet werden.

2.2 Mitgliedschaftsattribute

Folgende Attribute können über Attribute Exchange abgefragt werden:

- mitgliedschaft-bund
- mitgliedschaft-land
- mitgliedschaft-bezirk
- mitgliedschaft-kreis
- mitgliedschaft-ort

Dem Attribut „mitgliedschaft-bund“ kommt hierbei eine besondere Bedeutung zu, da abhängig davon ob es abgefragt wird oder nicht eine implizite oder explizite Prüfung der Mitgliedschaft erfolgt (siehe oben).

Die Attribute müssen mit genau diesen Namen als Alias abgefragt werden. Als Attributtyp ist jeweils „https://id.piratenpartei.de/openid/schema/NAME“ anzugeben, wobei NAME durch den Namen des Attributs zu ersetzen ist.

Ist ein Attribut in der Antwort leer, kann dies daran liegen, dass kein gültiges Token eingetragen ist. Um absichtlich leere Attributfelder zu kennzeichnen (z. B. Pirat ist in keinem Ortsverband) werden drei Bindestriche (TODO WERT) verwendet.

Das Feld „mitgliedschaft-bund“ enthält bei Mitgliedern den Wert „ja“ (TODO WERT), die restlichen Attribute teilen mit, in welchen Untergliederungen der Nutzer Mitglied ist. Für mögliche Werte sei auf die Liste der Untergliederungen[7] verwiesen.

Die Verknüpfung des Useraccounts mit den zugehörigen Daten findet über das Token statt. Die Betreiber des ID-Servers können einen Account also nicht einer bestimmten Person zuordnen, da die Zuordnung zwischen Token und Person nur über die Mitgliederverwaltung möglich ist.

[7] TODO

2.3 Pseudonym

Das Pseudonym ermöglicht es einer Webanwendung, einen Benutzer wiederzuerkennen. Es handelt sich hierbei nicht um einen vom Nutzer gewählten Namen, sondern um einen berechneten Wert. Das Pseudonym wird nicht als Attribut, sondern als Teil des Felds claimed_identifier übermittelt.

Zu jedem Benutzer wird in der Datenbank ein Zufallswert gespeichert, das sogenannte Usersecret. Das Pseudonym ist ein SHA256-Hash über eine systemweite geheime Konstante (das Pseudonym-Secret), das Usersecret sowie das Realm aus dem OpenID-Request. Ändert sich einer dieser Werte, ändert sich auch das Pseudonym. Das Pseudonym ist somit für jede Kombination von Benutzer und Realm einmalig, d.h. ein Nutzer hat für jedes Realm ein anderes Pseudonym. Das konstante Pseudonym-Secret wäre hierbei eigentlich nicht nötig, erhöht aber die Sicherheit: Wenn ein Angreifer Lesezugriff auf die Datenbank erhält, kann er ohne das (außerhalb der Datenbank) gespeicherte Pseudonym-Secret nichts mit den Usersecrets anfangen. Versuche, die Hashes zu brechen und aus dem Pseudonym auf das Usersecret zu schließen sind bereits aufgrund der Entropie des Usersecrets aussichtslos. Ohne Kenntnis von User- und Pseudonymsecret ist es nicht möglich festzustellen, ob zwei Pseudonyme dem gleichen Nutzer gehören.

Es wird empfohlen, als zusätzliche Sicherheitsmaßnahme das Pseudonym bei der pseudonymen Anmeldung dem Nutzer nicht anzuzeigen und am Besten nur gehasht zu speichern. Hierbei ist es egal, ob die gesamte URL oder nur das Pseudonym an sich gehasht wird. Dadurch kann selbst ein Angreifer, der aus irgendeinem Grund Authentifizierungsvorgänge fälschen kann, nicht die Identität eines bestehenden Nutzers übernehmen, wenn er das Pseudonym nicht anderweitig erfahren hat.

Mit Zugriff auf die Secrets ist eine Zuordnung eines Pseudonyms zu einem Account möglich. Bei einer Accountlöschung wird das Usersecret vernichtet, womit die Zuordnung nicht mehr möglich ist.

2.4 Personenbezogene Daten

Auf Wunsch einiger Mitglieder wurde die Option vorgesehen, auch personenbezogene Daten über das ID-System bestätigen zu können. Hierzu wurden die Attribute `realname` und `mitgliedsnummer` sowie entsprechende Datenbankfelder reserviert.

Über ein zweites Token („B-Teil“) könnten diese Werte nach einem noch festzulegenden Verfahren nach ausdrücklicher und freiwilliger Zustimmung des Mitglieds aus der Mitgliederverwaltung nachgetragen werden. Der Zugriff auf diese Attribute ist derzeit vollständig deaktiviert. Nach der Aktivierung ist eine Whitelist vorgesehen, d.h. nur gesondert freigeschaltete Realms hätten die Möglichkeit, diese Daten anzufordern. Diese Freischaltung würde nur für Webanwendungen erfolgen, die die Daten zwingend brauchen und strenge Datenschutzvorschriften einhalten. Wie alle Attribute werden diese

Informationen nur nach ausdrücklicher Zustimmung des Nutzers an einen Dienst herausgegeben.

Die Möglichkeit, diese Daten zu speichern, wird nicht aktiviert, bevor ein entsprechender Bedarf vorliegt, d.h. eine Anwendung existiert die diese Daten zwingend benötigt. Für die meisten Webanwendungen sind diese Daten nicht erforderlich.

Der Großteil der IT möchte diese Daten **nicht** im System haben.

3 Sicherheitseigenschaften

In diesem Abschnitt werden Sicherheitseigenschaften des Systems aufgelistet, sowie die Maßnahmen genannt, die diese Eigenschaften sicherstellen.

3.1 Nicht geplante Sicherheitseigenschaften

Bevor auf die Sicherheitseigenschaften des Systems eingegangen wird, sollte erwähnt werden, welche möglichen Erwartungen an das System **nicht** erfüllt werden. Die Liste kann logischerweise nie vollständig sein, sollte aber die wichtigsten Punkte umfassen.

Absolute Sicherheit der Accounts

Wie bei Onlinediensten üblich kann unter Angabe der E-Mail-Adresse eine Mail mit einem Kennwortrücksetzlink angefordert werden. Ist ein Angreifer in der Lage, diese Mail mitzulesen, beispielsweise indem er den SMTP-Datenverkehr abhört oder Zugang zum E-Mail-Postfach des Nutzers hat, kann er sich Zugang zum betroffenen Benutzerkonto verschaffen.

Dieses Risiko wird akzeptiert, da die Einschränkung der Nutzbarkeit ohne eine solche Funktion zu hoch wäre und es sich um ein allgemein übliches Verfahren handelt. Angriffe, die auf dem Abhören der Mail bis zur Einlieferung ins Postfach des Empfängers beruhen, sind sehr selten. Nutzer werden besonders darauf aufmerksam gemacht, dass sie den Zugang zu ihrem Postfach schützen müssen.

Verwendete Mailadressen

Da pro E-Mail-Adresse nur ein Benutzerkonto erstellt werden kann, kann beim Versuch einen Account anzulegen anhand der Fehlermeldung festgestellt werden, ob mit der angegebenen Adresse bereits ein Benutzerkonto existiert. Es wäre zwar möglich, dies zu verhindern, indem bei bereits verwendeter Adresse auch eine Erfolgsmeldung angezeigt wird und der Nutzer per Mail benachrichtigt wird. Dies würde weiterhin erfordern, dass beim Zurücksetzen vergessener Kennwörter der Nutzer kein Feedback bekommt, wenn er

eine falsche Adresse eingegben hat. Eine solche Regelung wäre für den Nutzer verwirrend, und würde unter Umständen keine vollständige Sicherheit bieten (Seitenkanalangriffe).

Die Möglichkeit festzustellen, dass eine Adresse verwendet wird, wird daher akzeptiert, was im Übrigen auch gängige Praxis [8] ist.

Schutz vor Betreiber

Das System ist ein zentralisiertes Authentifizierungssystem. Die Sicherheit beruht auf der Annahme, dass der Betreiber des Systems vertrauenswürdig ist. Der Betreiber kann beliebige OpenID-Antworten liefern und hat somit die Möglichkeit, sich mit fremden oder falschen Attributen bei Diensten zu identifizieren. Der ID-Server sieht bei jedem Login-Vorgang, welcher Nutzer sich wann bei welchem Dienst angemeldet hat. Mit den Daten des ID-Servers können mit realistischem Aufwand Pseudonyme zu Nutzerkonten zugeordnet werden, solange das Usersecret nicht gelöscht wurde.

Fehler beim Nutzer

Das ID-System nutzt gewöhnliche Passwortauthentifizierung. Wenn ein Nutzer ein unsicheres Passwort wählt oder es nicht geheim hält, kann jemand der dieses Passwort errät oder erlangt den Account missbrauchen. Schadsoftware auf dem Rechner eines Nutzers kann das Kennwort ausspähen und den Account dieses Nutzers missbrauchen. Das ID-System kann dem Nutzer die Pflicht nicht abnehmen, sich um seine Sicherheit zu kümmen. Hinweise auf Sicherheitsmaßnahmen werden jedoch gegeben. Passwortrichtlinien (min. 8 Zeichen, min. 2 Arten von Zeichen) sind vorhanden.

Der Nutzer wird von fremden (d.h. potentiell bösartigen) Seiten auf die Seite des ID-Systems geleitet, wo er sich anmelden soll. Der Nutzer muss selbst prüfen, dass er sich auf der richtigen Seite befindet. Tut er dies nicht und gibt seine Nutzerdaten auf einer gefälschten Seite ein, können diese missbraucht werden. Entsprechende Hinweise werden auf der Seite eingeblendet.

Client-Zertifikate werden nicht genutzt, da die Mehrzahl der Nutzer mit der korrekten Nutzung (Backup, Transfer auf andere Geräte) überfordert sein dürfte. Außerdem würde es den Nutzern die Möglichkeit nehmen, sich von beliebigen Computern anzumelden, was von den Nutzern nicht akzeptiert würde.

[8] getestet: wordpress.com (Passwortreset), drupal.org (Accounterstellung), phpbb.com (Accounterstellung im Supportforum), news.piratenpartei.de (Accounterstellung)

Sicherheitslücken in Webanwendungen

Das ID-System schützt nicht vor Sicherheitslücken in den Webanwendungen. Ist eine Webanwendung unsicher, kann sie die Daten ihrer Nutzer offenlegen oder Missbrauch der Anwendung zulassen. Für die Entwickler von Webanwendungen steht ein Leitfaden zur Verfügung, der auch Sicherheitsthemen beinhaltet. Die Entwickler sind für die Sicherheit der Anwendungen jedoch selbst verantwortlich.

3.2 Vorgesehene Sicherheitseigenschaften

Sicherheit der Authentifizierung

Bei korreter Anwendung des Protokolls soll sich die Webanwendung darauf verlassen können, dass genau der Nutzer, zu dem die übermittelten Daten gehören, sich gegenüber dem ID-System eingeloggt hat, sowie dass diese Daten korrekt sind. Daraus folgt auch, dass es keinem Dritten möglich sein darf, sich mit einem fremden Pseudonym gegenüber einer Webanwendung auszuweisen.

Die Prüfung der Antwort beim ID-Server im Rahmen des OpenID-Protokolls stellt sicher, dass nur unverfälschte Antworten, die vom ID-System erstellt wurden, akzeptiert werden. Replay-Attacken werden verhindert, da die Korrektheit einer Antwort nur einmal bestätigt wird. Die Übermittlung per HTTPS verhindert das Mitlesen der Antworten.

Man-in-the-Middle-Attacken, bei denen eine Webanwendung die Login-Antwort eines Nutzers missbraucht, um sich als dieser Nutzer an einer anderen Anwendung anzumelden, werden ebenfalls verhindert, da die Return-URL im Rahmen des OpenID-Protokolls geprüft wird.

Vertraulichkeit der Daten

Die freigegebenen Daten werden nur an den angegebenen Dienst herausgegeben. Dies wird sichergestellt, indem die Seite (Realm), an die die Daten weitergegeben werden sollen, angezeigt wird, und die Return-URL (an die die Daten gesendet werden) zu diesem Realm gehören muss. Die Übertragung der Daten findet SSL-gesichert statt, sodass Dritte nicht mitlesen können.

Der Dienst, der die Daten erhält, kann technisch nicht daran gehindert werden, diese Daten zu missbrauchen oder weiterzugeben. Der Nutzer muss selbst entscheiden, ob er einem Dienst seine Daten anvertrauen möchte.

Datenweitergabe nur mit Einverständnis

Im ID-System gespeicherte Daten werden nur mit Einverständnis des Nutzers, bestätigt durch die Eingabe des Kennworts, weitergegeben.

Ein Pirat - ein Pseudonym (pro Realm)

Jeder Nutzer hat nur ein Pseudonym pro Realm. Eine Webanwendung, die Umfragen oder Abstimmungen anbietet, kann sich daher in der Regel darauf verlassen, dass jedes Mitglied nur einmal teilnehmen kann, wenn pro Pseudonym nur eine Teilnahme ermöglicht wird und die Mitgliedseigenschaft geprüft wird.

Aus diesem Grund ist es einem Teilnehmer auch nicht möglich, sein Pseudonym zu ändern, allerdings kann jeder mehrere Benutzerkonten haben. Die Bindung eines Kontos an eine Person geschieht erst durch die Eingabe eines Tokens - ohne Token ist das Konto nicht als das Konto eines Mitglieds bestätigt. Um sicherzustellen, dass ein Mitglied (in seiner Eigenschaft als Mitglied) nicht mehrere Konten (und somit mehrere Pseudonyme nutzt), wird bei der Eingabe eines Tokens dieses an das entsprechende Konto gebunden.

Ein einmal eingegebenes Token kann nicht mehr entfernt oder geändert werden, und es ist nicht möglich, das gleiche Token für ein anderes Konto zu verwenden. Wird ein Benutzerkonto gelöscht, bleibt das Token (bzw. ein Hash davon) gespeichert, um sicherzustellen, dass es nicht wiederverwendet werden kann (ein neues Konto hätte ein anderes Usersecret und somit andere Pseudonyme).

Da mit der Löschung eines Kontos bis auf das Token alle Daten (inkl. Usersecret) gelöscht werden, und ein Token nicht wiederverwendet werden kann, bedeutet dies, dass ein Mitglied, welches sein Konto mit eingetragenem Token löscht, das ID-System nicht mehr benutzen kann. Die Mitgliederverwaltung darf in diesem Fall **kein** neues Token ausstellen, da das Prinzip „Ein Pirat - ein Pseudonym“ damit verletzt würde!

Nicht verwendete Token können beliebig ausgetauscht werden. Vor der Neuausstellung eines Tokens sollte die Mitgliederverwaltung bestätigen lassen, dass das alte Token nicht verwendet wurde, und dieses sperren lassen. Sollte ein Austausch verwendeter Token gewünscht sein, muss entweder die Verletzung dieses Prinzips im Einzelfall in Kauf genommen werden, oder es muss ein weiteres Attribut „token-index“ eingeführt werden, durch welches Webanwendungen gegenüber angezeigt wird, ob ein Token neu ausgestellt wurde. Webanwendungen könnten somit selbst entscheiden, ob bzw. bis zu wie viele Neuausstellungen (d.h. Pseudonymänderungen) sie pro Mitglied zulassen.

Nichtverknüpfbarkeit der Pseudonyme

Es soll Dritten nicht möglich sein, Pseudonyme mit dem Benutzerkonto oder mit anderen Pseudonymen zu verknüpfen, d.h. es soll auch zusammenarbeitenden Diensteanbietern (mit verschiedenen Domains/Realms) nicht möglich sein festzustellen, dass ein Pseudonym zu einem bestimmten Benutzerkonto gehört oder dass Pseudonym 1 in der ersten Webanwendung und Pseudonym 2 in der zweiten Webanwendung dem gleichen Nutzer gehören. Dies wird durch das Berechnungsverfahren für Pseudonyme (siehe entsprechender Abschnitt) sichergestellt.

3.3 Vermutete Sicherheitseigenschaften

Die im folgenden benannten Eigenschaften sind während der Entwicklung „nebenbei“ entstanden und wurden nicht vollständig überprüft. Es handelt sich um zusätzliche Eigenschaften die keineswegs für einen sicheren Betrieb nötig sind, jedoch die Sicherheit erhöhen können (Defence in depth) oder technisch interessant sind.

Stärke gegen Angreifer mit Datenbankzugriff

Ein Angreifer, der Lesezugriff auf die (gegen solchen Zugriff selbstverständlich geschützte) Datenbank erlangt, kann zwar die Nutzerdatenbank auslesen, aber nicht am System missbrauchen. Insbesondere sollte es nicht möglich sein, ein benutzbares Token zu erhalten (da die Token gehasht gespeichert werden, aber im Klartext eingegeben werden müssen), ein Benutzerkonto zu nutzen/übernehmen (da die Passwörter gehasht gespeichert werden), ein Kennwort zurückzusetzen (Reset-Keys werden gehasht gespeichert) oder eine falsche E-Mail-Adresse zu bestätigen (Bestätigungskeys werden gehasht gespeichert). Auch bei einem unsicheren Kennwort sollte der Angreifer keine Möglichkeit haben, den Passworthash zu Bruteforcen, da ein zusätzliches, in den Systemkonstanten gespeichertes geheimes Salt verwendet wird. (Die Möglichkeit einer langsamen und somit nur gegen sehr schwache Kennwörter möglichen Online-Bruteforce-Attacke bleibt unverändert.)

Auch das Zuordnen von Pseudonymen sollte unmöglich sein, da das zusätzlich verwendete Pseudonym-Secret unbekannt ist.

Stärke gegen Angreifer mit Dateisystemzugriff

Ein Angreifer, der Lesezugriff auf das Dateisystem erlangt, könnte in der Lage sein, den privaten Schlüssel für das SSL-Zertifikat zu erlangen, sofern dieser auf dem System gespeichert ist. Damit könnte er Antworten fälschen, wenn es ihm gelingt, die Echtheitsanfrage der Webanwendung an den ID-Server auf sein System umzuleiten. Ohne vorherige Kenntnis des Pseudonyms kann er dieses nur berechnen (und darüber Pseudonyme zuordnen), wenn er auch Zugriff auf die Datenbank (oder die Dateien in der die

Datenbanktabellen liegen) hat. Ohne Kenntnis des Pseudonyms kann er gegenüber Webanwendungen nicht vortäuschen, ein bestimmter anderer Nutzer zu sein. Benutzerkonten mit sicheren Kennwörtern sollten weiterhin wie beim Angreifer mit Datenbankzugriff geschützt sein.

4 Schutzmaßnahmen gegen Angriffe

4.1 Server

Schwachstelle Mensch

Um Angriffe, die Fehler des Nutzers ausnutzen, zu erschweren, werden die Nutzer durch gut sichtbare Banner auf wichtige Sicherheitsmaßnahmen hingewiesen. Dazu zählt die sichere Verwendung von HTTPS, Sicherheit des verwendeten Computers, Passwortsicherheit, sowie die Absicherung des E-Mail-Accounts.

Cross-site request forgery, Session-basierte Angriffe

Der Server des ID-Systems arbeitet vollständig ohne Sessions. CSRF sowie andere Session-basierte Angriffe sind somit nicht möglich. Der Nutzer muss zur Bestätigung von Transaktionen jedes mal seine Logindaten eingeben. Der Zugriff auf Nutzerdaten wird wo möglich über einer Funktion gekapselt, die die Logindaten prüft, um Programmierfehler auszuschließen.

SQL-Injection/Truncation

Zum Schutz vor SQL-Injection werden Prepared Statements benutzt. Um Truncation-Angriffe zu verhindern, wird an mehreren Stellen die Länge von Parametern geprüft und die Datenbankverbindung so konfiguriert (SQL mode), dass auch leichte Fehler (z. B. überlange Felder) als fatale Fehler behandelt werden.

Passwort-Hashverfahren

Kennwörter werden nicht im Klartext, sondern mit SHA256 gehasht gespeichert. Dabei werden der Benutzername als Salt, ein zusätzliches systemspezifisches Salt und strengthening als zusätzlicher Schutz eingesetzt.

Das bedeutet, dass ein Angreifer, der die Datenbank erlangt, zwar prüfen kann, ob ein Benutzer ein bestimmtes Kennwort hat, die Kennwörter aber nicht entschlüsseln kann. Der gängige Angriff gegen derart gesicherte Passwörter ist es, jedes Wort aus einer Liste

(Wörterbuch, alle Buchstabenkombinationen, ...) gegen die Liste zu prüfen, um so Treffer zu finden und Kennwörter aufzudecken (Brute-Force/Dictionary attack).

Um solche Angriffe zu erschweren, werden die Hashes folgendermaßen gebildet:

1. Das Kennwort wird mit dem Benutzernamen (benutzerspezifisches Salt) und einer geheimen Systemkonstante (Systemsalt) verkettet und gehasht
2. Der aktuelle Hash wird verkettet mit einem Zähler und dem ersten Hash, und erneut gehasht (das Ergebnis wird der neue aktuelle Hash).
3. Dieser Schritt wird für viele Iterationen wiederholt (der Zähler wird jeweils hochgezählt)

Die Salts sorgen dafür, dass das gleiche Kennwort bei zwei unterschiedlichen Nutzern oder auf zwei verschiedenen Systemen unterschiedliche Hashes ergibt. Das geheime Systemsalt macht Brute-Force-Angriffe gegen die Hashes völlig unmöglich, solange der Angreifer es nicht erlangt. Die Verkettung und Verwendung des Zählers stellt sicher, dass jede Iteration anders ist und verhindert so, dass das Verfahren in einen Zyklus gerät.

Durch die Verwendung der Salts wird die Verwendung vorausberechneter Rainbow Tables zur Beschleunigung von Angriffen unmöglich. Um ein Kennwort auf Gültigkeit zu testen, müssen zahlreiche Hashes (Iterationen) berechnet werden, was einige Millisekunden dauert. Dadurch werden Brute-Force-Versuche so stark ausgebremst, dass sie unpraktisch werden.

Anzumerken ist, dass diese Schutzmaßnahmen nur für den unwahrscheinlichen Fall benötigt werden, dass es einem Angreifer gelingt, die Benutzerdatenbank zu kopieren. Da die Verlangsamung durch das strengthening auch auf den Server selbst zutrifft, wird bei Online-Brute-Force-Versuchen die Geschwindigkeit durch die CPU-Leistung des Servers selbst dann wirksam begrenzt, wenn andere (gegen DoS-Attacken gerichtete) Schutzmaßnahmen versagen.

Hashing

Auch andere Informationen werden wo möglich nur gehasht gespeichert, beispielsweise die Token und die zum Zurücksetzen von Passwörtern und Bestätigen von Mailadressen nötigen Keys.

Clickjacking-Schutzmaßnahmen

Bei webbasierten Diensten besteht die Gefahr des sogenannten Clickjackings. Hierbei wird die angegriffene Website (hier: das ID-System) unsichtbar eingebunden, und der Nutzer dazu gebracht, eine (sichtbare) Schaltfläche zu drücken. Der Klick auf die Schaltfläche trifft jedoch durch geschickte Anordnung der Elemente die angegriffene Webseite

und löst dort eine Aktion aus. Im Fall des ID-Systems könnte dies z. B. die Bestätigung der Datenfreigabe sein.

Auch der Zwang, zu jeder Aktion das Kennwort eingeben zu müssen, schützt nicht vor diesem Angriff, da viele Browser Kennwörter speichern und automatisch ausfüllen. Um diesen Angriff zu verhindern, werden drei Schutzmaßnahmen getroffen:

Ein entsprechender HTTP-Header (`X-Frame-Options: deny`) verhindert in modernen Browsern die Einbettung. Dies ist insbesondere deswegen wichtig, in modernen Browsern IFRAME-Attribute (z. B. `sandbox`) den Javascript-basierten Schutz aushebeln.

Nutzer ohne JavaScript müssen beim Login zufällige Ziffern abtippen, welche als ausformulierte Wörter mit zufälliger Position auf der Seite stehen. Da bei Clickjacking die einbettende Seite nicht auf den Inhalt der eingebetteten Seite zugreifen kann, verhindert dies den Angriff weitgehend. Die zufällige Positionierung verhindert, dass dieser Teil der Website sichtbar belassen und nur der Rest überdeckt wird. Das Ausschreiben als Wort verhindert, dass der Nutzer dazu gebracht wird, die Ziffern zu markieren und ins Zielfeld zu ziehen. Da sowohl vor als auch nach den Ziffern ein zufälliger Abstand eingefügt wird, kann ein Angriff auch aus der Länge der Webseite nicht die korrekte Position errechnen.

Bei aktivem JavaScript wird eine Einbettung erkannt und in einem solchen Fall eine Warnung angezeigt sowie die Seite des ID-Systems verborgen.

Sichere Zufallszahlen

Zufallszahlen, bei denen ein hohes Sicherheitsniveau erforderlich ist, werden über OpenSSL generiert, statt sich auf unsichere PHP-Methoden zu verlassen. Dadurch ist die hohe Qualität und Nichtvorhersagbarkeit der Zufallszahlen sichergestellt.

HTTPS

Sämtliche Kommunikation mit dem ID-System erfolgt über HTTPS. Um Nutzer, die versuchen, die Adresse mit „http“ statt „https“ einzugeben, zu schützen, wird der `Strict-Transport-Security`-Header genutzt. Dieser weist moderne Browser an, die Seite ausschließlich über HTTPS abzurufen und verhindert so, dass Nutzer es über HTTP versuchen und sich so der Gefahr von MitM-Attacken aussetzen.

Internet Explorer 6

Nutzer des Internet Explorer 6 und früher werden von der Nutzung des ID-Systems ausgeschlossen und darauf hingewiesen, dass sie ihr System aktualisieren sollen. Mit dem IE6 ist eine sichere Internetnutzung nicht möglich, zudem deutet sein Vorhandensein auch auf ein generell veraltetes (und somit unsicheres) System hin.

4.2 Client

Der speziell entwickelte PHP-Client für PiratenID enthält einige erwähnenswerte Schutzmaßnahmen:

Schutz vor Session fixation

Es wird versucht, den php.ini-Wert `session.use_only_cookies` zu setzen. Gelingt dies, werden session fixation-Versuche beim Start der Session ignoriert. Scheitert dies, wird nach jedem Start/Fortsetzen der Session eine neue Session-ID erstellt, um Angriffe zu verhindern. Dadurch kann die Stabilität der Sessions beeinträchtigt werden.

Da jedoch nicht ausgeschlossen werden kann, dass andere Skripte auf dem Server session fixation zulassen, wird die Sitzungs-ID auch neu generiert, wenn die Sitzung bereits gestartet war, oder wenn sie zum ersten Mal von PiratenID genutzt wird.

Dadurch wird sichergestellt, dass eine dem Angreifer dank session fixation bekannte Session-ID ungültig wird, bevor Daten bzw. ein Loginzustand von PiratenID in der Sitzung gespeichert werden.

Sicherheit des Session-Cookies

Die Gültigkeit des Session-Cookies wird beschränkt auf das eingestellte OpenID-Realm. Dadurch wird in Szenarien, wo mehrere Anwendungen sich eine Domain teilen, sichergestellt, dass Skripte außerhalb des Realms nicht an die Session-ID gelangen.

Das Session-Cookie wird weiterhin so eingestellt, dass es lediglich über HTTPS-gesicherte Verbindungen verschickt wird. Dies verhindert, dass die Session-ID z. B. durch unsicher eingebundene Bilddateien unverschlüsselt übertragen wird. Eine weitere Option, die aktiviert wird, verhindert das Auslesen das Cookies durch JavaScript (was z. B. XSS-Angriffe erschwert).

Parameterprüfungen, Selbstschutz

Die Library schützt sich gegen irrtümlich falsche Benutzung, z. B. indem die Einbindung auf Seiten, die nicht über HTTPS aufgerufen werden, verhindert wird. Gegen absichtlichen Missbrauch ist die Library nicht geschützt, da jeder, der die Library missbrauchen möchte, einfach den Code ändern kann.

Anleitung

Webanwendungsentwickler erhalten eine Anleitung, welche neben der korrekten Verwendung der PiratenID-Library auch allgemeine Sicherheit von Webanwendungen behandelt.

5 Dateien des Servers

5.1 Verzeichnis /includes/

Dieses Verzeichnis enthält Dateien, die von den anderen Skripten eingebunden werden.

/includes/header.inc.php

Diese Datei enthält den Anfang jeder ausgegebenen HTML-Seite und nimmt auch einige Einstellungen vor.

- Ein Error-Handler, der Fehler wie Warnungen zu fatalen Fehlern macht und wenig Informationen über den Fehler mitteilt, wird aktiviert
- Es wird sichergestellt, dass `register_globals` deaktiviert ist.
- `siteconstants.inc.php` und `functions-global.inc.php` werden eingebunden
- Die HTTP-Header `X-Frame-Options` und `Strict-Transport-Security`
- Das HTML-Grundgerüst der Seite mit Sicherheitshinweisen, Menü und IE6-Blocker werden ausgeliefert. Das maincontent-DIV wird geöffnet

/includes/footer.inc.php

Diese Datei enthält das Ende jeder ausgegebenen HTML-Seite:

- Der IE6-Blocker und das maincontent-DIV werden geschlossen
- Die Fußzeile wird ausgegeben
- Der JavaScript-basierte Clickjacking-Schutz wird ausgegeben

/includes/siteconstants.inc.php

Diese Datei enthält (z.T. geheime!) Konstanten, die bei der Installation des Servers gesetzt werden müssen:

- `$sitepath` - Die absolute Adresse des ID-Systems. Wird z. B. beim Versand von E-Mails verwendet.
- `$extendedAttributeRealms` - Die Liste der Realms, die erweiterte Attribute (Realname, Mitgliedsnummer) anfragen können.
- `getDatabasePDO()` - Hier werden die Datenbank-Zugangsdaten eingetragen
- `$pseudonymsecret` - Ein zusätzliches Salt zur Berechnung der Pseudonyme, siehe Abschnitt 2.3 Pseudonyme.
- `$passwordsaltsecret` - Ein zusätzliches Salt zur Berechnung der Passwort-Hashes, siehe Abschnitt 4.1 Passwort-Hashverfahren.
- `$openid_hmacsecret` - Das zur Berechnung der OpenID-Signaturen verwendete secret.

Maßnahmen, die zu treffen sind, wenn eine der geheimen Konstanten kompromittiert wird, sind in der Datei selbst dokumentiert.

/includes/functions-global.inc.php

Diese Datei enthält die globalen Funktionen:

- `safeout(&$text)` - Gibt eine Variable mit korrektem HTML-Escaping aus. Ist die Variable undefiniert, wird nichts ausgegeben, aber keine Warnung erzeugt
- `prefilter(...)` - Prüft einen Wert (z. B. GET/POST-Parameter) auf Existenz, Typ und Länge
- `generateNonce($entropy)` - Erzeugt die angegebene Anzahl sicherer Zufallsbytes (Hex-encoded)
- `hashPassword($username, $pw)` - Berechnet den Passwort-Hash, siehe Abschnitt 4.1 Passwort-Hashverfahren
- `class DB` - Die DB-Klasse kapselt Datenbankzugriffe über ein Singleton.
  - Die Felder connection, statement und error sind zwecks leichtem Zugriff public.
  - `query(...)` führt eine Datenbankabfrage (mit Längenprüfung der Parameter) durch. Die Felder statement und ggf. error werden gesetzt.
  - `get()` liefert die Singleton-Instanz

  - `cleanup()` löscht veraltete Daten und wird automatisch beim Erstellen einer neuen Instanz aufgerufen

- `getUser(...)` - Wenn in den POST-Variablen ein gültiges Login (inkl. Clickjacking-Schutz) liegt, wird der dazugehörige User mit allen Daten zurückgegeben. Zugriffe auf Benutzerdaten erfolgen über den Rückgabewert dieser Funktion, um sicherzustellen, dass immer ein Login erforderlich ist.
- `printLoginFields()` - Gibt Felder für ein Loginformular inkl. Clickjacking-Schutz aus (die mit getUser() kompatibel sind)
- `checkPassword(...)` - Prüft, ob ein Passwort inkl. Wiederholung in Ordnung sind, d.h. das Passwort setzt werden kann. Prüft **nicht** ob ein Login gültig ist, dafür ist getUser da!
- `checkMail(...)` - Prüft, ob eine E-Mail-Adresse gültig und noch nicht verwendet (d.h. für das Anlegen eines Accounts zulässig) ist

/includes/.htaccess

Diese Datei verhindert direkten Zugriff auf die Include-Files.

5.2 Verzeichnis /user/

Dieses Verzeichnis enthält Seiten, die mit der Verwaltung der Benutzerkonten zusammenhängen.

/user/create.php

Diese Datei regelt die Erstellung von Benutzerkonten. Sie zeigt das entsprechende Formular an, nimmt die Daten entgegen, prüft sie, trägt den Nutzer in der Datenbank ein und versendet die Aktivierungsmail.

/user/confirm.php

Diese Datei regelt die Bestätigung der Mailadresse/Aktivierung der Benutzerkonten. Sie wird ausschließlich über einen Link aus der Aktivierungsmail aufgerufen und erhält den Aktivierungsschlüssel. Daraufhin versucht sie die Datenbank entsprechend zu aktualisieren und teilt dem Nutzer das Ergebnis mit. Im Erfolgsfall wird (mit einem Link auf entertoken.php) darauf hingewiesen, dass der Nutzer jetzt sein Token eintragen soll.

/user/entertoken.php

Diese Datei ermöglicht es dem Nutzer, seinem Account ein Token hinzuzufügen. Sie zeigt das entsprechende Formular an, nimmt die Daten entgegen, prüft sie und trägt die Änderung in der Datenbank ein.

/user/changepw.php

Diese Datei ermöglicht es dem Nutzer, sein Kennwort zu ändern. Sie zeigt das entsprechende Formular an, nimmt die Daten entgegen, prüft sie und trägt die Änderung in der Datenbank ein.

/user/requestreset.php

Diese Datei ermöglicht es dem Nutzer, eine Passwortreset-Mail anzufordern. Sie zeigt das entsprechende Formular an, nimmt die Daten entgegen, prüft sie, erstellt ggf. ein Token und trägt es in die Datenbank ein und verschickt die Mail mit dem Link.

/user/reset.php

Diese Datei ermöglicht es dem Nutzer, sein Kennwort über einen via requestreset.php angeforderten Link zurückzusetzen. Sie gibt ein Formular zur Eingabe eines neuen Kennworts aus, nimmt die Daten entgegen, prüft das Token und die Daten und ändert ggf. das Kennwort.

/user/delete.php

Diese Datei ermöglicht es dem Nutzer, sein Benutzerkonto zu löschen. Sie zeigt das entsprechende Formular an, nimmt die Daten entgegen, prüft sie, und löscht ggf. das Konto. Der Nutzer muss die Löschung durch Eingabe des Wortes LOESCHEN bestätigen. Um den Account zu löschen, werden bis auf Token(hash), E-Mail-Verifizierungsstatus und Erstellungsdatum alle Daten überschrieben. Der Tokenhash bleibt gespeichert, damit das Token nicht wiederverwendet werden kann. Der Benutzername wird durch einen zufällig generierten Benutzernamen ersetzt.

5.3 Verzeichnis /openid/

Dieses Verzeichnis enthält die serverseitige Implementierung des OpenID-Teils

/openid/endpoint.php

Der OpenID-Endpoint, das Herzstück des Systems. Behandelt den Authentifizierungsvorgang:

- OpenID-Requests von Webanwendungen
- Login durch den Nutzer
- Antwort auf die OpenID-Anfrage
- Bestätigen der OpenID-Daten gegenüber der Webanwendung

Der Großteil der Datei besteht aus Funktionsdefinitionen. Am Ende wird entschieden, was für eine Anfrage vorliegt, und entsprechend die passende Funktion aufgerufen. Die einzelnen Funktionen sind innerhalb der Datei kommentiert, nur die wichtigsten werden hier aufgeführt, in der Reihenfolge, in der sie in einem normalen Authentifizierungsvorgang eine Rolle spielen.

- `getOpenIDFields` - liest die OpenID-Werte aus der GET- oder POST-Anfrage aus. Umgeht dabei ein PHP-„Feature“, welches Feldnamen umbenennt.
- `evaluateFields` - prüft die übergebenen OpenID-Werte von Anfragen (wird von handleCheckidSetup und handleUserConfirm genutzt/aufgerufen)
- `handleCheckidSetup` - liefert das Login-Formular, wenn eine OpenID-Authentifizerungsanfrage ankommt oder handleUserConfirm einen Fehler entdeckt.
- `handleUserConfirm` - verarbeitet die Antwort auf das Login-Formular (d.h. die Bestätigung der Anfrage durch den Nutzer)
- `addAXAttributes` - fügt einer von handleUserConfirm vorbereiteten OpenID-Antwort die zusätzlichen Attributfelder hinzu, falls nötig.
- `sendIndirectResponse` - liefert eine HTML-Seite mit einer indirekten OpenID-Antwort (wird von handleUserConfirm aufgerufen) (d.h. einem HTML-Formular, welches an die anfragende Webanwendung zurückgePOSTed wird und die OpenID-Antwort enthält. Enthält eine Warnung, dass der Nutzer nun zur ursprünglichen Seite zurückkehrt, um Phishing z. B. durch das Anfordern von HTTP Basic Auth (während der Nutzer noch die ID-Seite sieht) zu verhindern.
- `handleCheckAuth` - verarbeitet die Verifizierungsanfrage der Webanwendung (Prüfung der indirekten Antwort)
- `sendDirectResponse` - beantwortet eine direkte OpenID-Anfrage (für die Überprüfung der übermittelten Daten, wird von handleCheckAuth aufgerufen)

/openid/xrds.php

Liefert ein XRDS-Dokument mit einem Verweis auf den OpenID-Endpoint aus. Dies wird für die Verwendung mit normalen OpenID-Libraries benötigt.

/openid/pseudonym.php

Die Pseudonym-URLs verweisen auf diese Datei. Liefert ein XRDS-Dokument aus, welches bestätigt, dass die Pseudonyme vom PiratenID-Endpoint authentifiziert werden. Dies wird für die Verwendung mit normalen OpenID-Libraries benötigt.

5.4 Verzeichnis /static/

Dieses Verzeichnis enthält statische Dateien wie Grafiken, Stylesheets, die Sicherheits-Hinweisbanner etc.

6 Datenbank

Ein SQL-Dump der Datenbankstruktur sollte dieser Dokumentation beiliegen. Die Datenbank besteht aus drei Tabellen:

6.1 Tabelle „users“

Diese Tabelle enthält die Benutzerkonten. Folgende Felder sind vorhanden:

- `username` - der eindeutige, nicht änderbare Benutzername (Primärschlüssel)
- `usersecret` - das Benutzer-Secret zur Berechnung der Pseudonyme (siehe Abschnitt 2.3)
- `pwhash` - der Hash des Passworts (siehe Abschnitt 4.1)
- `email` - die E-Mail-Adresse (z. B. für Passwortresets)
- `email_activationkey` - der Hash des Aktivierungsschlüssels zum Bestätigen der Mailadresse
- `email_verified` - gibt an, ob die Adresse bestätigt ist
- `token` - das eingetragene Token (gehasht) - darf nicht mehr verändert oder gelöscht werden!
- `mitgliedsnr` - Feld zur Aufnahme der Mitgliedsnummer, sofern diese Funktion implementiert wird

- `realname` - Feld zur Aufnahme des Realnamens, sofern diese Funktion implementiert wird
- `resettoken` - Hash des Passwort-Reset-Tokens, falls verschickt (sonst NULL)
- `resettime` - Zeitpunkt, zu dem die Resetmail verschikt wurde
- `createtime` - Zeitpunkt, zu dem der Account erstellt wurde

6.2 Tabelle „tokens“

Diese Tabelle enthält die Zuordnungen zwischen Token(hashes) und Verbandsmitgliedschaften. Sie sollte regelmäßig geleert und neu aus der Mitgliederverwaltung importiert werden. Folgende Felder sind vorhanden:

- `token` - Das (gehashte) Token (Primärschlüssel)
- `mitgliedschaft-bund`
- `mitgliedschaft-land`
- `mitgliedschaft-bezirk`
- `mitgliedschaft-kreis`
- `mitgliedschaft-ort`

Bezüglich der Bedeutung und möglicher Werte für die Mitgliedschaftsfelder sei auf Abschnitt 2.2 verwiesen.

6.3 Tabelle „openid“

- drei tabellen - erklärung import - erklärung felder

7 Client (PHP)

Der bereitgestellte PiratenID-Client ist eine vollständige Eigenentwicklung. Er ist in PHP geschrieben und ermöglicht Webanwendungen in der gleichen Sprache eine bequeme und sichere Nutzung des PiratenID-Systems.

Die Parameter des PiratenID-Systems sind fest in den Client eingebaut. Der Client ist nicht für die Nutzung mit anderen OpenID-Servern gedacht oder geeignet. (Es fehlen wichtige Teile des OpenID-Protokolls, welche für die Nutzung mit einem fest vorgegebenen Server jedoch nicht nötig sind.)

Der Client besteht aus einer PHP-Klasse mit statischen Funktionen und einigen statischen, öffentlichen Variablen. Um ihn zu nutzen, werden (nach dem Einbinden der Datei)

zunächst über die Variablen die gewünschten Parameter eingestellt. Danach können die enthaltenen Funktionen benutzt werden.

Der Client kann das komplette Session-Handling übernehmen. Hierzu müssen die betroffenen Funktionen (run bzw. initSession) aufgerufen werden, bevor die Header gesendet worden sind, d.h. vor allen Ausgaben. Vorsicht: Ein Byte-order-mark, was manche Editoren am Anfang von UTF-8-Textdateien einfügen, kann in einer PHP-Datei dazu führen, dass die Header gesendet werden! Wenn der Client das Session-Handling übernimmt, werden automatisch einige zusätzliche Sicherheitsmaßnahmen angewendet. Falls es also möglich ist, sollte das Session-Handling immer dem Client überlassen werden.

Den einfachsten Einstieg in die Nutzung des Clients bietet das mitgelieferte Beispiel (`example.php`).

7.1 Voraussetzungen

Der Client benötigt PHP mit aktivierten Stream-Funktionen mitsamt HTTPS/SSL-Unterstützung. Der Client wurde auf PHP 5.3.5 getestet. Aus Sicherheitsgründen sollte stets eine aktuelle PHP-Version verwendet werden.

Die PHP-Einstellung `session.use_only_cookies` sollte zur Erhöhung der Sicherheit in der php.ini aktiviert werden, falls möglich. Kann diese Einstellung nicht aktiviert werden, wird der Client die Session-ID bei jedem Aufruf neu erzeugen, um session fixation-Angriffe zu verhindern. Dies kann die Stabilität der Sessions beeinträchtigen.

Session-Cookie

Falls das im Client eingebaute Session-Handling benutzt wird, gilt Folgendes:

Wenn ein Session-Cookie vorhanden ist, und dieses Session-Cookie als „Secure“ (nur über HTTPS abrufbar) konfiguiert ist, geht das PiratenID-System davon aus, dass sich jemand über Session-Sicherheit Gedanken gemacht hat und übernimmt die Konfiguration des Session-Cookies.

Ist das Secure-Flag nicht gesetzt, wird lediglich die Gültigkeitsdauer übernommen. Der Gültigkeitsbereich des Cookies wird auf das Realm festgelegt und die Secure- und HTTP-Only-Flags werden gesetzt.

7.2 Installation und Einrichtung

Die Buttongrafiken (`button-*.png`) müssen in einen geeigneten öffentlichen Ordner platziert werden. Der Pfad, unter dem die Grafiken erreicht werden können, ist im Parameter `PiratenID::$imagepath` zu setzen (siehe unten).

Die Clientdatei (`piratenid.php`) muss irgendwo platziert werden, wo sie eingebunden werden kann, und in den sie verwendenden PHP-Seiten eingebunden werden (beispielsweise mit `require('piratenid.php')`). Im gleichen Verzeichnis muss die Datei certificate.pem liegen. Diese enthält das SSL-Zertifikat, über das der PiratenID-Server authentifiziert wird. Dabei kann es sich entweder um das Zertifikat des Servers selbst, oder um das einer CA (Zertifizierungsstelle) im Zertifikatspfad des Serverzertifikats handeln.

Parameter

Nachdem die Datei eingebunden wurde, müssen vor der Nutzung einige Parameter gesetzt werden:

- `PiratenID::$realm` - Das OpenID-Realm, den Bereich, für welchen die Authentifizierung erfolgt. Muss gesetzt werden! Dieser Bereich muss sich unter alleiniger Kontrolle der Webanwendung befinden. Diese Angabe ist sicherheitskritisch und sollte fest eingetragen werden. Variablen aus `$_SERVER` können z. T. gefälscht werden und dürfen daher hier nicht genutzt werden! Das Realm einer recht strengen Prüfung, es muss sich um eine HTTPS-URL auf einen Ordner handeln, genauer: Die URL muss mit einem Schrägstrich enden und darf keine Query-Parameter oder Anker enthalten. Sowohl die anfragende Seite als auch die Return-URL müssen innerhalb dieses Realms liegen. Das Realm wird auch für den Gültigkeitsbereich des Cookies verwendet.
- `PiratenID::$returnurl` - Die URL, zu der der PiratenID-Server den Nutzer nach der Authentifizierung zurückschickt. Falls gesetzt, muss diese URL innerhalb des Realms liegen, d.h. sie muss mit der Realm-URL beginnen. Wird automatisch erkannt (Adresse der aktuellen Seite), falls nicht gesetzt.
- `PiratenID::$imagepath` - Der öffentliche Pfad zu den Buttongrafiken. Standardmäßig leer (d.h. aktuelles Verzeichnis). Dieser Wert wird vor die Button-URLs eingefügt und kann ein relativer oder absoluter Pfad wie '/', '/piratenid-buttons/' oder 'https://images.example.com/piratenid/' sein.
- `PiratenID::$attributes` - Die anzufragenden Attribute als durch Kommata getrennte Liste. Standardmäßig leer. Mögliche Attribute siehe Abschnitt **??**. Wenn hier `mitgliedschaft-bund` aufgeführt wird, können sich auch Nichtmitglieder einloggen, die Webanwendung prüft die Mitgliedschaft selbst (explizite Prüfung der Mitgliedschaft). Wird dieses Attribut nicht abgefragt, können sich nur Mitglieder einloggen.
- `PiratenID::$usePseudonym` - boolean-Wert (Standard: true), welcher angibt, ob ein Pseudonym angefragt werden soll. Ist dieser Wert false, erfolgt ein anonymes Login. Damit kann sichergestellt werden, dass sich gerade ein Pirat eingeloggt hat. Nutzer können jedoch nicht wiedererkannt werden, d.h. Doppelaccounts können nicht verhindert werden. Diese Option ist daher nur in Sonderfällen nützlich.

Mögliche Attribute siehe Abschnitt **??**. Wenn hier `mitgliedschaft-bund` aufgeführt wird, können sich auch Nichtmitglieder einloggen, die Webanwendung prüft die Mitgliedschaft selbst (explizite Prüfung der Mitgliedschaft). Wird dieses Attribut nicht abgefragt, können sich nur Mitglieder einloggen.

- `PiratenID::$logouturl` - Die URL, auf die der Logout-Button verweisen soll. Standard: Realm-URL. Siehe auch `PiratenID::$handleLogout`.
- `PiratenID::$handleLogout` - Boolean, gibt an, ob das integrierte Logout-Handling verwendet werden soll. Wenn true, wird beim Login eine Zufallszahl erzeugt und in der Sitzung gespeichert. Diese wird an die Logout-URL automatisch in einem Parameter (`piratenid_logout`) angehängt. `PiratenID::$run()` erkennt diesen Parameter und loggt den Nutzer aus, wenn die Zufallszahl übereinstimmt. (Die Zufallszahl verhindert böswillige Logouts über CSRF.) Der Parameter wird automatisch aus dem $_GET-Array entfernt. Bei Verwendung von `PiratenID::$run()` und des dadurch erzeugten Buttons wird das komplette Loginhandling übernommen!
  Vorsicht: Soll statt `PiratenID::$run()` direkt `PiratenID::$button(...)` aufgerufen werden, so muss dieser Parameter deaktiviert werden!

7.3 Nutzung

Für die Nutzung des Clients, nachdem die Parameter eingestellt worden sind, gibt es zwei Möglichkeiten:

Nutzung über `PiratenID::run()`

Das „Komplettpaket“ `PiratenID::run()` übernimmt sämtliche Aufgaben, die mit PiratenID verbunden sind (wie die Verarbeitung von OpenID-Antworten), und gibt den HTML-Code für einen Login/Logout-Button zurück, der einfach an einer geeigneten Stelle ausgegeben werden kann. Eine Session wird automatisch erzeugt, die Funktion ist daher vor allen Ausgaben aufzurufen. Die Session-Variable `$_SESSION['piratenid_user']` wird angelegt und mit einem assoziativen Array befüllt. Dieses Array enthält folgende Elemente:

- `authenticated`: Boolean, gibt an, ob ein Nutzer eingeloggt ist.
- `attributes`: Assoziatives Array, enthält bei eingeloggten Nutzern die Attribute, die der Server geliefert hat.
- `pseudonym`: String, enthält bei pseudonym eingeloggten Nutzern ein Pseudonym, welches direkt verwendet werden kann. (Hash der OpenID-Identity-URL. Wenn es Angreifern gelingt, OpenID-Antworten zu fälschen, sie aber nicht die Identity-URL eines Nutzers kennen, können sie sich so nicht als dieser Nutzer einloggen. Die Identity-URL sollte aus diesem Grund nirgendwo verwendet oder angezeigt werden und wird daher nicht bereitgestellt.)

Selbstverständlich muss der zurückgegebene Button nicht genutzt werden. Das Login-Verfahren kann in dem Fall wie im nächsten Abschnitt beschrieben eingeleitet werden. Ein Logout erfolgt über den Aufruf der Funktion `PiratenID::logout()`.

Nutzung über `PiratenID::handle()`

Wenn mehr Kontrolle über den Authentifizierungsprozess gewünscht ist, können die Funktionen `PiratenID::initSession()`, `PiratenID::handle()`, `PiratenID::makeOpenIDURL()` und `PiratenID::button(...)` separat genutzt werden.

Falls `PiratenID::button(...)` genutzt werden soll, muss der Parameter `PiratenID::$handleLogou` deaktiviert werden! Geschieht dies nicht, wird in eingeloggtem Zustand dauernd die folgende Fehlermeldung angezeigt: „WRONG USAGE OF PIRATENID LIBRARY. Tried to generate button with $handleLogout true without correctly initialized session.“

`PiratenID::initSession()` initialisiert die Session auf sichere Art und Weise. Wird ein eigenes Session Handling durchgeführt, muss diese Funktion nicht genutzt werden.

`PiratenID::makeOpenIDURL()` liefert die URL zum ID-System, die den Login-Prozess mit den derzeitigen Einstellungen einleitet. Ein eigener Login-Button sollte auf diese URL verweisen. Die einzelnen OpenID-Felder können auch per `PiratenID::getOpenIDRequest()` als Array abgerufen werden.

`PiratenID::handle()` verarbeitet die indirekte Antwort des PiratenID-Servers unter der Return-URL. Bei einem POST auf die Return-URL mit dem Feld `openid_mode` sollte diese Funktion aufgerufen werden. Der Rückgabewert ist ein assoziatives Array mit den Feldern

- `authenticated`: Boolean, gibt an, ob ein Loginvorgang erfolgreich war
- `attributes`: (nur bei erfolgreichem Login) assoziatives Array mit den Attributen, die der Server geliefert hat
- `pseudonym`: (nur bei erfolgreichem Login mit Pseudonym) Pseudonym-String (Hash der OpenID-Identity-URL, siehe oben)
- `rawIdentityURL`: (nur bei erfolgreichem Login mit Pseudonym) Die OpenID-Identity-URL - sollte nicht genutzt werden, siehe oben!
- `error`: Enthält den aufgeretenen Fehler (oder null, wenn kein Fehler aufgetreten ist)

`PiratenID::button($logout, $errortext = null)` erzeugt den HTML-Code für einen Login- bzw. Logout-Button. Der Boolean-Parameter $logout bestimmt, ob es sich um einen Logout-Button handelt. Wird ein Fehlertext ($errortext) übergehen, funktioniert der Button zwar weiter als Login/Logout-Button, zeigt jedoch den angegebenen Fehler

an. `PiratenID::$handleLogout` darf nicht aktiv sein, damit diese Funktion einwandfrei funktioniert.

Wenn für den Login-Status die von `PiratenID::run()` genutzte Session-Variable genutzt wird, kann `PiratenID::logout()` zum Ausloggen des Nutzers verwendet werden und `PiratenID::autoButton($errortext)` liefert automatisch den richtigen Buttontyp (Session wird initialisiert, falls noch nicht geschehen).

7.4 Dateien

Der Client besteht aus folgenden Dateien:

- `piratenid.php` - Die Client-Klasse selbst. Die Funktionen werden weiter unten beschrieben.
- `certificate.pem` - Das Zertifikat, über das der PiratenID-Server authentifiziert wird. Dabei kann es sich entweder um das Zertifikat des Servers selbst, oder um das einer CA (Zertifizierungsstelle) im Zertifikatspfad des Serverzertifikats handeln.
- `button-*.png` - Drei Grafiken für Login/Logout/Fehler-Buttons.
- `example.php` - Ein Beispiel für die Verwendung des OpenID-Clients. Nicht Teil des eigentlichen Clients, sollte nicht in die Produktivumgebung kopiert werden.

7.5 Funktionen

Die Funktionen sind in der PiratenID-Klasse kommentiert. Fett gedruckte Funktionen sind public.

- `run()` - Automatisiert alle relevanten Abläufe, siehe Abschnitt 7.3. Eine Session wird gestartet, OpenID-Antworten und Logout-Wünsche werden bearbeitet.
- `error(...)` - Setzt die $error-Variable und gibt den HTML-Code für einen Fehlerbutton zurück. Wird von `run()` genutzt.
- `autoButton(...)` - Liefert je nach Login-Status den passenden Button (Login oder Logout) zurück. Initialisiert die Session, falls nötig.
- `button(...)` - Liefert den HTML-Code für einen Button zurück, welcher auf eine passende URL (Login/Logout etc.) verweist.
- `logout()` - Loggt den Benutzer aus. Initialisiert die Session, falls nötig.
- `initSession()` - Initialisiert die Session und wendet dabei erweiterte Sicherheitsmaßnahmen an.
- `handle()` - Wertet eine OpenID-Antwort aus (inkl. Verifizierung via `checkSignature(...)`) und gibt das Ergebnis zurück. Siehe auch Abschnitt 7.3.

- `makeOpenIDURL()` - Liefert eine URL zum OpenID-Endpoint, welche eine Login-Anfrage mit den aktuellen Parametern enthält. Siehe auch Abschnitt 7.3.
- `getOpenIDRequest()` - Liefert die OpenID-Felder für eine Login-Anfrage mit den aktuellen Parametern. Siehe auch Abschnitt 7.3.
- `initParams()` - Prüft und initialisiert die Parameter, errechnet abgeleitete Defaultwerte. Wird von den meisten Funktionen, die Parameter nutzen, aufgerufen.
- `isSSL()` - Gibt eine Vermutung darüber ab, ob die Seite über eine SSL-Verbindung aufgerufen wird.
- `checkSignature(...)` - Prüft die Signatur einer OpenID-Antwort durch SSL-gesicherte Rückfrage beim Endpoint.
- `getOpenIDFields(...)` - Liest die OpenID-Werte aus einer GET- oder POST-Anfrage aus. Umgeht dabei ein PHP-„Feature“, welches Feldnamen umbenennt.
- `isValidKeyValue(...)` - Prüft, ob das übergebene Paar gültige Werte für einen OpenID-Feldnamen und -wert beinhaltet.

7.6 Schutzmaßnahmen

Besondere Schutzmaßnahmen im Client werden in Abschnitt **??** erläutert.