Kapitel 6
Sicherheit in Verteilten Systemen

bahn.de oder ebay.com sind ohne Frage hochgradig verteilte Systeme, wenn sie sich ihren Kunden im Internet auch als einziges „monolithisches“ System präsentieren. Verfolgt man beispielsweise einige Requests an bahn.de, sieht man, dass eine Reihe externer Systeme zum Tracking von Benutzern angesprochen werden. Auch ein Blick auf das Online-Ticket verriet ursprünglich das Vorhandensein verschiedenster Backend-Systeme.

Noch ausgeprägter ist die Verteilung auf ebay.de. Dort kommen verschiedenste Kunden in Kontakt, Inhalte und teilweise auch Code werden zwischen unterschiedlichen Maschinen ausgetauscht – wenn auch vermittelt durch die Zentrale des ebay-Portals.

Bei diesen verteilten Systemen stellt sich die Frage, wie die Subsysteme untereinander abgesichert sind. Wie werden beispielsweise Anfragen von Kunden von System zu System weitergeleitet, ohne dass sie – zum Beispiel von Innentätern – missbraucht werden können? Wie kann überhaupt so etwas wie eine „Distributed Trusted Computing Base“ realisiert werden?

Hinter jeder Kommunikation in verteilten Systemen steht das Problem des Vertrauens (Trust): Wer traut wem aus welchen Gründen? Vertrauen kann auf Autorität beruhen – aber welche Autoritäten gibt es in verteilten Systemen und wie funktionieren sie?

Dieses Kapitel hat als Aufgabe, einige zentrale Eigenschaften verteilter Systeme im Zusammenhang mit der eingesetzten Sicherheitstechnologie zu beleuchten. Damit wollen wir die Basis für das Verständnis komplexer Infrastrukturen in den folgenden Kapiteln legen. Speziell suchen wir Antworten zu den folgenden Problemstellungen:

- Welche Authentisierungsweisen eignen sich besonders für verteilte Systeme?
- Welche sicherheitsrelevanten Informationen (Identität, Authentizität) werden zwischen Subsystemen ausgetauscht?
- Wie fließen sicherheitsrelevante Informationen zwischen Subsystemen?
- Wie lässt sich ein Aufruf über verschiedene Systeme hinweg verfolgen?
- Wie kann eine einmal erfolgte Authentisierung weitergegeben werden? Was bedeutet das für die Vertrauensbeziehungen zwischen Systemen?

- Was bedeuten die Begriffe Delegation und Impersonation in verteilten Systemen?
- Wie hängen Identitäten und Rechte zusammen? Welche Identitäten sind an einem verteilten Request beteiligt?
- Was konstituiert Domänen und wie arbeiten sie zusammen?
- Wie funktioniert Föderation von Systemen?
- Wie werden die Identitäten und Credentials verschiedener Systeme aufeinander abgebildet?
- Wie werden Credentials in verteilten Systemen gesichert?

Noch stellen die meisten Firmenportale relative Inseln dar – ausgedrückt durch eine eigene Autorität zur Identifikation von Nutzern. Noch sind die Infrastrukturen der Firmen weit davon entfernt, eine interne Ende-zu-Ende Sicherheit von den Eingangsrechnern über mittlere Systeme mit Geschäftslogik bis hin zu den Backendsystemen auf Mainframes zu bieten. Zu unterschiedlich sind die vorhandenen Datenstrukturen und Schnittstellen im Bereich Sicherheit.

Dennoch zeichnet sich bereits ein neuer Trend ab: Webservices und Grid-Architekturen ermöglichen die Nutzung firmenübergreifender Dienste. Das Internet-Business nutzt Services der verschiedensten Partner. Und nicht zuletzt erscheinen im Peer-To-Peer Bereich total verteilte Applikationen, die sich nicht nur auf effizientes Filesharing beschränken.

Eine zentrale Autorität fehlt allen diesen Architekturen, und so ist das Problem der effizienten Föderation von Nutzern und Services ein sehr bedeutsames für die Zukunft.

6.1 Lokale versus verteilte Sicherheit

Was unterscheidet überhaupt die Sicherheitsanforderungen in einem verteilten System von denen in einem lokalen System, bei dem alle beteiligten Anwendungen sich auf demselben Rechner befinden? Zunächst einmal fällt im verteilten System die Notwendigkeit einer Kommunikation zwischen den Instanzen ins Auge. Dies impliziert als Sicherheitsanforderung die Absicherung der zur Kommunikation benutzten Kanäle (*kanalbasierte* Sicherheit) bzw. der ausgetauschten Nachrichten (*nachrichtenbasierte* Sicherheit). Hinzu kommt die Notwendigkeit der Authentifizierung der miteinander kommunizierenden Instanzen.

In beiden Fällen müssen sich Nutzer des Systems authentifizieren, jedoch gibt es in einem lokalen System logischerweise nur eine einzige zentrale Instanz, die authentisiert und auch authorisiert. Insbesondere besteht im lokalen Szenario keinerlei Notwendigkeit, das Ergebnis der Authentifikation und Authorisierung an andere Instanzen weiterzugeben. In verteilten Systemen stellt gerade diese Notwendigkeit eines der zentralen Sicherheitsproblem dar: Wie können Authentisierungen und damit verbundene Rechte auf sichere Weise von einer Instanz an die nächste weiter gegeben werden? Oder soll sich ein Nutzer für jede Instanz, auf die

er zugreifen möchte, separat authentifizieren? Eng damit verbunden ist die Frage, welche Instanzen in einem verteilten System überhaupt vertrauenswürdig sind: Wie kann Vertrauen in einem verteilten System aufgebaut werden? Und wie soll eine empfangende Instanz mit dem Ergebnis einer Authentifizierung umgehen? Führt sie daraufhin ihre eigene Authorisierung durch oder vertraut sie den Rechten, die ihr durch die sendende Instanz mitgeteilt werden? Schließlich stellt sich noch die Frage, welche Identität bei einer eigenen Authorisierung zu Grunde gelegt werden soll: Die des Nutzers, der die ursprüngliche Anfrage gestellt hat, oder die der weitergebenden Instanz?

Diese kurze Diskussion zeigt, dass die Sicherheit in einem verteilten System sehr viel mehr Aspekte beinhaltet als die in einem lokalen System. Als erstes werden wir die Frage der Authentisierung in einem verteilten System genauer untersuchen.

6.2 Authentisierung in verteilten Systemen

6.2.1 Authentisierung versus Identifizierung

Ganz selbstverständlich taucht die Authentisierung eines Clients häufig als erstes Softwareproblem bei der Entwicklung einer verteilten Applikation auf. Meist wird sie als unabdingbare Vorstufe zur Authorisierung gesehen, das heißt, dem Zugestehen von Rechten an authentisierte Nutzer.

Bevor wir uns dem „wie, wann und wo“ dieses Problems zuwenden, möchten wir diesen Automatismus kurz hinterfragen. Dazu ist es nötig zu untersuchen, was bei der Authentisierung eines Requests eigentlich passiert. Betrachten wir zunächst den klassischen Fall, wie er auch im Portal von bahn.de auftritt:

- Die Firma klassifiziert ihre Ressourcen (Informationen, Applikationen) nach Sicherheitskriterien (öffentlicher Zugriff, Zugriff für authentisierte Kunden, Zugriff nur für Mitarbeiter etc.). Im Anschluss daran wird die Infrastruktur so konfiguriert, dass die geschützten Ressourcen nur über eine durchgeführte Authentisierung erreichbar sind. Softwaretechnisch taucht schnell das Problem auf, dass ein Kontrollfluss von öffentlichen Funktionen in geschützte verhindert werden muss.
- Die Identität, die einen Request an eine Firmeninfrastruktur stellt, wird geprüft. Diese Identität kann eine „echte“ sein, das heißt, eine reale Person (Mensch, Firma) repräsentieren. Die Identität kann aber auch eine Maschine darstellen. Diese Form der Authentisierung macht Sinn, wenn eine in der Realität gültige Identität für die Durchführung des Requests nötig ist, sei es zur Abrechnung (Bahnticket) oder auch zur Bestrafung bei Missbrauch.
- Schnell stellt man fest, dass hinter der Frage der Identität des Initiators des Requests eine Reihe großer Fragezeichen stecken: Erhalten beispielsweise registrierte Clients eine neue Identität, sobald sie Kunden werden? Welche Iden-

tität nutzen Mitarbeiter, die gleichzeitig Kunden sind? Akzeptiert die Firma Identitäten, die von anderen Institutionen vergeben wurden? In der Praxis wird das Problem der Identität häufig einfach durch Anlegen eines Verzeichnisses gelöst. Geschieht das für jede Applikation gesondert, so entsteht in kürzester Zeit ein völliges Chaos in Bezug auf Identitäten und Authentisierungen. Heute gibt es Infrastrukturen, die unter dem Schlagwort *Identity Management* dieses Chaos eindämmen wollen. Speziell Portale benötigen solche Infrastrukturen, die wir weiter unten vorstellen werden.

- Im Zuge der Prüfung einer vorgegebenen Identität legt die Infrastruktur der Firma ein softwaretechnisches Stellvertreterobjekt für den Client an. Dieser wird genannt und stellt einen authentisierten Client dar. Je nach Güte der Infrastruktur gilt dieser Principal nur auf der Maschine, die die Prüfung durchgeführt hat, oder kann im Verlaufe der Bearbeitung an andere Maschinen und Applikationen weitergegeben werden. Das Stellvertreterobjekt dient der Festlegung, mit welchen Rechten ein Request ausgeführt werden soll, und außerdem der Aufzeichnung aller Aktionen, die für diesen Request durchgeführt wurden (Auditing). Gleichzeitig wird meist eine so genannte *Session* erstellt, damit der Authentisierungsprozeß nicht beim nächsten Request des Clients sofort wieder durchgeführt werden muss. Das Ergebnis der Authentisierung muss also in irgendeiner Form festgehalten werden. (Je nach Industriestandard oder Framework wird statt Identität (Name) und Principal (erfolgreich authentisierte Identität) auch das Paar Principal (Name) und Subject (erfolgreich authentisierter Principal) verwendet. Wichtig ist nur die Unterscheidung zwischen Namen und erfolgreicher Authentisierung und im Anschluss daran die Frage in welcher Form die erfolgreiche Authentisierung festgehalten wird bzw. wie lange.

Etwas anders gelagert ist der Fall, wenn die Feststellung einer Identität zur Bearbeitung eines Requests streng genommen gar nicht nötig ist: Stellen wir uns die Prüfung eines elektronischen Tickets im Zug vor. Hier geht es lediglich darum festzustellen, ob das Ticket gültig ist. Derzeit stellt die Schaffnerin jedoch auch die Identität des Fahrgasts anhand seiner Bahncard als Teil der Ticketprüfung fest. Dies wäre im Prinzip jedoch erst nötig, wenn sich das Ticket als ungültig herausstellen würde, da dann aus Gründen der Bestrafung eine Authentisierung durchgeführt werden müsste. Allerdings bindet das Geschäftsmodell der Bahn an dieser Stelle das Ticket an den Besitzer der Bahncard um zu verhindern dass günstigere Tickets für Nicht-Bahncard Besitzer ausgestellt werden können. Natürlich bevorzugen Firmen grundsätzlich die Feststellung der Identität zur Authentisierung von Kunden, da sich über das Konzept der Identität hervorragend Daten über das Konsumverhalten der Kunden sammeln lassen und so Profile der Kunden entstehen. Es sei jedoch angemerkt, dass das Interesse der Firmen nicht immer unbedingt im Einklang mit dem Interesse der Kunden steht und aus technischer Sicht nicht immer eine Authentisierung nötig ist.

Bei näherem Hinsehen stellt sich schnell heraus, dass auch aus wirtschaftlichen Gründen die Authentisierung von Berechtigungen statt von Personen ein sehr attraktives Modell sein kann. Man denke beispielsweise an die Zulieferfirmen für

große Konzerne, die bei identitätsbasierter Authentifikation gezwungen sind, die Mitarbeiter der Konzerne in ihren Datenbanken zu führen und jede Änderung im Personalbestand nachzuvollziehen. In beiden Fällen jedoch, der Authentisierung zur Feststellung einer Identität oder der Authentisierung von Berechtigung durch eine Gültigkeitsprüfung, muss auf Empfängerseite ein Ergebnis dieser Prüfung festgehalten werden. Die Art und Weise, wie dies geschieht ist durch den jeweiligen Authentisierungsmechanismus festgelegt und wird weiter unten sowie im Kapitel zu föderativer Sicherheit diskutiert.

6.2.2 Authentisierung mit Passwörtern

Nichts scheint einfacher, als Passwörter zum Zwecke der Authentisierung von Sendern einer Nachricht einzusetzen. Kunden, aber auch die Rechner einer Webapplikation sowie „künstliche“ User (so genannte *Functional User*) besitzen Passwörter. Mitarbeiter melden sich mit Passwörtern am firmeninternen Intranet an. Häufig verschaffen sich sogar Administratoren von Webapplikationen per Passwort Zugang zu ihrer Applikation, um diese aus dem Internet heraus zu bearbeiten.

Aus sicherheitstechnischer Sicht und von ihrem Verwaltungsaufwand her müssten Passwörter jedoch längst der Vergangenheit angehören, wie wir noch sehen werden. Entwickler dürften eigentlich gar nicht mehr an ihre Verwendung denken. Die Realität sieht jedoch anders aus: Bei Entwicklern sind Passwörter geschätzt, weil sie das am leichtesten verständliche Mittel der Authentisierung sind. Schnell ist eine Passwortabfrage implementiert. Notfalls lässt sich auch ein eigenes Verzeichnis (eine so genannte *Registry*) erstellen, um einen kleinen Stamm von Benutzern zu verwalten.

Damit entsteht aber ungewollt schnell die Situation, dass Administrationsrechte von Fremden auf einfachste Weise erlangt werden können, beispielsweise aufgrund der Tatsache, dass nicht mehr existierende oder nicht mehr benötigte User in diversen Applikationen und deren Registries eingetragen bleiben.

Es lohnt sich also, die Gründe für die Probleme mit Passwörtern in verteilten Systemen einmal systematisch anzugehen.

6.2.2.1 Generelle Eigenschaften von Passwörtern

Wir haben die Authentifikation mittels Passwörter bereits im Kapitel „Sicherheitsdienste“ kurz angesprochen. Hier noch einmal zusammengefasst die wichtigsten Eigenschaften von Passwörtern:

- Passwörter sind Geheimnisse zwischen zwei Parteien, zwischen denen sie auf einem sicheren Weg etabliert werden müssen.
- Passwörter müssen durch Menschen merkbar sein.

- Passwörter müssen auf sicherem Weg übertragen werden
- Passwörter dürfen nicht durch Dritte erratbar sein.
- Werden Passwörter gespeichert, muss dies auf sichere Weise geschehen.

Der Punkt, dass Passwörter auf sicherem Weg zwischen zwei Partner etabliert werden müssen, spielt eine wesentliche Rolle im e-Business-Bereich, wie schon am Beispiel ebay diskutiert: Eine Firma bietet Services am Internet an. Ein Kunde möchte diese nutzen und registriert sich zu diesem Zweck über ein Web-basiertes Formular. Die Firma muss den Kunden jetzt über einen externen Mechanismus, zum Beispiel den Postweg authentisieren. Das bedeutet einen erheblichen Mehraufwand für die Firma und außerdem einen gewissen Zeitraum, in dem die Identität des Kunden und des Senders nicht geklärt ist.

Dieses Problem versuchen Webservices zu lösen, indem sie ein Sicherheitsmodell verwenden, das auf vertrauenswürdigen Dritten (*Trusted Third Parties* oder auch *Security Token Issuer*) aufbaut. Diese fungieren als zentrale Authentisierungsautorität für mehrere Services zugleich. Somit wird vermieden, dass sich ein Kunde selbst beim gewünschten Zielservice authentisieren muss und der Aufwand für den Serviceanbieter wird geringer (siehe Kapitel Web Services Security).

Das Kriterium der Merkbarkeit von Passwörtern stellt uns ebenfalls vor große Probleme: Generell lässt sich sagen, dass ein merkbares Passwort entweder relativ kurz ist oder aber eine Bedeutung trägt und somit mit hoher Wahrscheinlichkeit in einem Wörterbuch zu finden ist. In beiden Fällen kann das Passwort durch einen Angreifer erraten werden – im ersten Fall durch eine Brute-Force-Attacke, die alle möglichen Passwörter unterhalb einer bestimmten Länge durchprobiert, im zweiten Fall durch eine Dictionary-Attacke, bei der die in einem Wörterbuch enthaltenen Wörter als Passwörter durchprobiert werden.

Beide Angriffe funktionieren dann am besten, wenn der Angreifer das gesuchte Passwort in einer gehashten oder auch leicht verschlüsselten Form kennt und dadurch die Vergleiche mit den geratenen Passwörtern offline in großer Geschwindigkeit durchführen kann.

Eine Weiterentwicklung von Passwörtern ist die so genannte Passphrase, wie sie zum Beispiel im populären Verschlüsselungsprogramm PGP zum Einsatz kommt: Ein längerer Ausdruck, der aus Wörtern und Zahlen gebildet wird, der zwar nicht mehr so leicht angreifbar ist, aber auf jeden Fall auch merkbar sein muss. Dies bedeutet im Normalfall einen Ausdruck, der entweder semantisch sinnvoll ist oder aber aufgeschrieben werden muss.

Darüber hinaus neigen wir dazu, Passwörter wieder zu verwenden, obwohl ja zur Definition eines Passwortes gehört, dass es ein Geheimnis zwischen genau zwei Parteien darstellen sollte. Sobald wir ein Passwort mehrfach verwenden, gefährden wir nicht nur unsere eigene Sicherheit, sondern auch die unserer Partner: Ein weiterer Service, bei dem ich mich mit dem gleichen Passwort anmelde, könnte sich beim ersten Service unter meinem Namen (und mit meinen Rechten) einloggen.

6.2.2.2 Sicherheitstechnische Bewertung

Wenn Sie einen Internet-basierten Dienst anbieten und dabei die Authentisierung via Passwort erlauben wollen, müssen Sie folgende Punkte klären:

- Sollen UserID und Passwort vom Service vorgegeben werden?
- Soll ein Pseudonym des Users nach außen hin erscheinen?
- Soll das Passwort gewechselt werden können?
- Wie verhalten Sie sich im Falle eines vergessenen Passworts?
- Auf welche Services erlauben Sie den Zugriff über Passwörter?
- Wie werden die Passwörter sicher übertragen?
- Wie werden die Passwörter sicher gespeichert, und zwar auf Server- und Clientseite?
- Wer erhält Zugriff auf die Passwörter?
- Wie sichern Sie sich vor mehrfach verwendeten Passwörtern?
- Wie sichern Sie sich gegen Brute-Force und Dictionary Attacken?
- Geben sie Usern Tipps zum Umgang mit Passwörtern?
- Sind Passwörter der einzige Zugang zu Ihren Services?

Das sind eine Menge Fragen für einen scheinbar so einfachen Mechanismus. Schon die erste Frage zeigt einen grundsätzlichen Gegensatz zwischen Benutzbarkeit (Usability) und Sicherheit auf: Kunden hassen es, wenn man ihnen eine neue Identität sowie ein halbwegs sicheres (das heißt schlecht zu merkendes) Passwort aufzwingt. Der Kunde ist gezwungen, beides aufzuschreiben bzw. auf seinem Desktop zu speichern.

Was sind die Konsequenzen? Für seltene Anmeldungen, die von immer dem gleichen Platz aus durchgeführt werden, mag das Verfahren tragbar sein. Sobald sich der Kunde jedoch häufig anmelden muss bzw. dies von wechselnden Orten aus tun muss, besteht die Gefahr, dass die aufgeschriebenen UserID/Passwort-Kombination mitgeführt wird, mit der entsprechenden Gefahr bei Verlust.

Überlässt man hingegen dem Kunden die Wahl von UserID und Passwort, besteht natürlich die Gefahr, dass einfachste Passwörter gewählt werden, die von Angreifern leicht erraten werden können. Zur Verdeutlichung der Problematik noch ein kleines Gedankenexperiment:

Angenommen, Sie bieten einen Service auf dem Web an, wo Sie gegen eine kostenlose Registrierung irgendeine Leistung versprechen. Mit den UserIDs und Passwörtern, die Ihre User bei Ihnen verwenden, versuchen Sie, Zugang zu bekannten Webdiensten zu bekommen. Das Ergebnis wird vermutlich erschreckend sein. Im Endergebnis bedeutet dies, dass der Versuch der Vermeidung von Brute-Force und Dictionary Attacken bei Passwörtern genau entgegengesetzt zur Benutzerfreundlichkeit verläuft.

Die Frage nach der Sichtbarkeit des Pseudonyms ist ebenfalls nicht einfach zu beantworten. Nehmen Sie einen Service wie ebay, bei dem die Nutzer untereinander mit der UserID bekannt sind. Diese UserID ist zwar ein Pseudonym, das frei gewählt werden kann, gleichzeitig stellt sie aber auch die LoginID dar, mit der die Benutzer ihren ebay-Account verwalten. Manche Systeme trennen diese beiden

Identitäten, um zu verhindern, dass durch Erraten von Passwörtern ein Angreifer die Identität eines legitimen Nutzers übernehmen kann (und zum Beispiel falsche Angebote in dessen Namen abgibt). Zum Schutz gegen das Erraten von Passwörtern ist ein automatischer Abschaltmechanismus denkbar, der zum Beispiel nach drei fehlgeschlagenen Login-Versuchen zu einer Sperrung des Accounts führt. Werden jedoch LoginID und sichtbares Pseudonym nicht voneinander getrennt, so stellt ein solcher Mechanismus gleichzeitig auch eine Einladung zu Denial-of-Service-Angriffen dar.

Wenn Ihr Service an öffentlichen Plätzen, zum Beispiel einem Internet-Café oder auch im Intranet genutzt werden kann, besteht die Gefahr, dass Passwörter abgeschaut werden. Sie können versuchen, dem Missbrauch durch einen regelmäßigen erzwungenen Wechsel der Passwörter zu begegnen. Sehr häufig stellt jedoch der erzwungene Wechsel des Passworts ein großes Problem sowohl für die User wie auch die Infrastruktur dar (Was passiert in einer komplexen Application Server Infrastruktur wenn während eines Passwort-Wechsels ein Absturz von Server, Verzeichnisdienst oder Datenbanken geschieht?). Oft erfolgt nach einem Zwangswechsel des Passworts nach kurzer Zeit der Fall des vergessenen Passworts, denn meist sind die letzten vom User verwendeten Passwörter gesperrt. Das bedeutet, dass die Nutzer sich – während sie eine Aufgabe in der Firma erledigen sollen – parallel ein komplett neues Passwort von möglichst hoher Qualität ausdenken und merken müssen, mit der Konsequenz, dass es schnell vergessen wird.

In diesem Fall müssen Sie sich für einen zweiten Mechanismus (Fallback-Mechanismus) entscheiden, der benutzt wird, wenn das eigentliche Passwort nicht mehr verfügbar ist. Soll der Mechanismus auf vorher abgefragten „Geheimnissen“ des Users basieren wie zum Beispiel der berühmt-berüchtigte „Mädchenname der Mutter“? Wie sicher kann ein solches Verfahren überhaupt sein? Oder soll per Post ein neues Geheimnis zugestellt werden, das dann vom User sofort durch ein eigenes neues Passwort ersetzt werden muss? Schließlich muss auch die Frage geklärt werden, wie der Ersatzmechanismus in Kraft treten soll: Reicht dafür ein einfacher Anruf beim Helpdesk aus oder müssen weiter gehende Bedingungen durch den User erfüllt werden?

Die Frage, wie Passwörter sicher übertragen werden können, scheint zu Zeiten von SSL/TLS eindeutig beantwortet zu sein. Aber erinnern wir uns daran, dass SSL nur einen sicheren Tunnel zwischen zwei Beteiligten zur Verfügung stellt. Es bleiben die Grundprobleme einer jeden kanalbasierten Verbindung:

- Zwischen wem besteht die Verbindung?
- Wie abhörsicher/vertraulich ist die Verbindung?
- Was passiert mit den Daten am Ende der Verbindung?

Serverseitig lässt sich das Problem, wer der Endpunkt des Tunnels ist, zwar durch ein Zertifikat lösen, jedoch muss die Vorstellung des Kunden vom Servicenamen nicht mit dem übereinstimmen, was im Server-Zertifikat steht. Auf Client-Seite ist völlig offen, wer Endpunkt des Kanals ist, da Ihre Kunden im Normalfall nicht über ein Zertifikat verfügen werden. Auch die Abhörsicherheit der Verbindung lässt sich zwar durch geeignete Wahl der Ciphersuites sicher stellen, dies

geschieht aber nicht automatisch, sondern muss durch eine geeignete Server-Konfiguration erzwungen werden. Zudem haben Sie keinerlei Kontrolle darüber, wie ihr Kunde mit dem Passwort verfährt, wenn es erst einmal auf seinem Rechner angekommen ist. Aber auch der Kunde muss sich darauf verlassen, dass der Server sorgfältig mit seinen Credentials umgeht – eine Gewähr hierfür gibt es nicht.

Dies bringt uns zur noch schwierigeren Frage nach der Speicherung von Passwörtern: Wo und wie sollen sie gespeichert werden und wo am besten gar nicht? Nicht gespeichert werden sollten sie beispielsweise beim Benutzer, da der heimische PC ein sehr verwundbares Speichermedium darstellt.

Leider bietet eine ganze Reihe von Werkzeugen wie Browser eine Speicherfunktion von Credentials, bei der diese wiederum mit einem einzigen Passwort geschützt sind. Dies ist zwar einerseits erfreulich, da es dem User erlaubt, eine ganze Reihe von Passwörtern zu verwalten (also eine Mehrfachnutzung zu vermeiden). Auch können dadurch wesentlich schwieriger merkbare Passwörter verwendet werden (keine Simplifikation). Der Nachteil liegt jedoch in der Qualität der Speicherung. Zur Speicherung durch den Browser wird nämlich ein so genanntes SSM – ein *Software Security Module* – verwendet. Es liegt ganz in der Hand der Softwareentwickler wie sicher ein solches SSM ist – da das SSM mit keinem externen System interagieren muss, braucht es auch keinen Standards zu genügen.

Aber auch in der eigenen Serviceinfrastruktur sollten keine Passwörter beispielsweise in den Logfiles diverser Server auftauchen, wo sie bereit liegen zum Mitlesen durch die Mitarbeiter in Betrieb oder Entwicklung. Damit nicht genug: Werden Passwörter über verschiedene SSL/TLS Verbindungen weitergereicht, so entstehen an jedem Tunnelendpunkt ungeschützte Momente, wo die Passwörter im Klartext vorliegen. Hier stellt sich die Frage, ob Passwörter im RAM von Servern längere Zeit verbleiben oder gleich nach der Verifizierung gelöscht werden.

Letztlich spielt auch der Ort, an dem serverseitig die Passwörter gespeichert werden, eine wichtige Rolle. Wie werden die Passwörter abgelegt? Wer erhält Zugriff darauf? Wie werden unberechtigte Zugriffe von Seiten von Mitarbeitern verhindert? Im engen Zusammenhang mit der Frage der Speicherung steht die Frage, wie und wo ein eingegebenes Passwort gegen ein gespeichertes verifiziert wird. Geschieht die Verifikation beim Application Server oder innerhalb eines Repositories? Werden hierbei Authentisierungsmethoden verwendet, die die Kenntnis des unverschlüsselten Passwortes auf einem vorgezogenen Server voraussetzen? Dies ist zum Beispiel bei Digest Verfahren nach Art der http Digest Authentication der Fall. Während diese Verfahren auf dem Transportweg vom Client zum Server Vorteile bieten, bedeutet die Notwendigkeit, dass vorgezogene Server die Passwörter im Klartext kennen müssen, gegenüber der Variante, bei der das gespeicherte Passwort die Registry nicht verlässt (und dort auch nur in verschlüsselter Form gehalten wird) auf Serverseite einen deutlichen Sicherheitsnachteil.

Schließlich müssen Sie die Frage klären, welche Services Sie über Passwörter absichern wollen. Dahinter steckt letztlich das Problem der Bewertung der Qualität einer Authentisierung durch Passwörter.

Nach dieser Diskussion, die mehr Fragen aufgeworfen als Antworten gegeben hat, lässt sich zusammenfassend das Folgende über Passwörter sagen:

- Passwörter sind eine „schwache“ Authentisierungsmethode.
- Sie sind nicht geeignet für jede Art von Administrationsservices.
- Sie sind nicht geeignet für Services, bei denen größere Werte im Spiel sind.
- Sie sind nicht geeignet für Services, auf denen der Dienstleister eine Form von Nicht-Abstreitbarkeit benötigt, wie es der Fall ist, wenn zum Beispiel teure Spezialanfertigungen für Kunden erstellt werden.

6.2.2.3 Verwaltung von Passwörtern

In diesem Abschnitt wollen wir etwas genauer auf die Schwierigkeiten eingehen, die vergessene Passwörter und ein automatischer Wechsel von Passwörtern (so genanntes *Password Aging*) für die Server-Infrastruktur mit sich bringen können.

Vergessene Passwörter stellen gerade bei qualitativ höheren Passwörtern ein großes Problem dar. Die meisten Firmen sind gezwungen, teure Helpdesks einzurichten, bei denen ihre Nutzer ein neues Passwort beantragen können. Für Firmen, die ausschließlich im Internet tätig sind, bleibt wie oben angesprochen oft nur ein automatischer Mechanismus, basierend auf vorher mit dem User abgesprochenen Daten. Als letzte Option kann der User eine neue Identität beantragen. In beiden Fällen handelt es sich um sehr teure Organisationsformen, deren Notwendigkeit noch größer wird, wenn die Security Policy der Firma einen regelmäßigen Wechsel der Passwörter mit Verbot der Wiederbenutzung alter Passwörter vorschreibt.

Aber nicht nur für die User bedeutet der erzwungene Passwortwechsel eine unangenehme Situation. Wesentlich weiter reichende Auswirkungen besitzt der Wechsel des Passworts für hochkomplexe Infrastrukturen wie zum Beispiel verteilte Webapplikationen mit ihren Datenbanken, Registries, Application Servern, Web Servern, Proxies, Caches etc.

In diesem Umfeld werden sehr viele verschiedene Identitäten benutzt, darunter auch so genannte *Functional User*, also Identitäten, unter denen eine Applikation eine andere aufruft. Die dazu gehörigen UserID und Passwörter finden sich häufig in den Konfgurationsfiles von Web Applikationen, Servern etc. Laufen auch diese Passwörter ab – zum Beispiel weil sie in einer Registry mit automatischem Password Aging gespeichert sind – dann kann eine Reihe von Problemen auftreten. So kann eine falsche Reihenfolge bei der Passwortänderung zum Stillstand des ganzen Systems führen. Manche Komponenten bemerken eine Änderung der Credentials „hinter ihrem Rücken“ mit der Folge, dass sie nicht mehr booten. Mit einem „Warnsystem“, das den Wechsel bzw. Ablauf des Passworts ankündigt, lassen sich einige dieser Probleme lösen, allerdings können die Probleme auch noch komplexer werden, wenn mehrere Registries hintereinander geschaltet sind, beispielsweise zuerst ein LDAP (Lightweight Directory Access Protocol) und anschließend ein RACF (Resource Access Control Facility) auf dem Mainframe.

Kann nun das User Interface zum LDAP auf eventuelle Hinweise zum Password Aging reagieren, die von RACF kommen?

Eine passwortbasierte Authentisierung ist also teurer und aufwändiger als man denkt – jedenfalls sobald man versucht, gewisse Sicherheitsregeln zu beachten.

6.2.3 Software-Architektur der Authentisierung mit Passwörtern

Zwei Fragen sind für das Verständnis der SW-Architektur der Authentisierung in verteilten Systemen wesentlich:

- Wer nimmt die Authentisierungsdaten (also die Credentials) entgegen?
- Wo werden die Credentials gehalten – in einer lokalen Registry oder einer zentralen, gemeinsam genutzten?

Als Antwort auf die erste Frage lassen sich Betriebssysteme und Applikationen unterscheiden. Betriebssysteme besitzen häufig bereits fortgeschrittene Verfahren der Authentisierung, während sich Applikationen oft noch wie Inseln in der Infrastruktur verhalten, so dass wir häufig auf das Szenario von Abb. 6.1 treffen.

Hier führt jede Applikation ihre Authentisierung selbst durch und speichert die Credentials darüber hinaus in einer eigenen Registry ab.

Aus Sicht der Softwareentwickler ist dies ein harmloses Szenario, da es für eine maximale Entkopplung der Entwicklungsteams in den Unternehmen sorgt. Jede Gruppe kann ihre eigene Userverwaltung aufbauen. Selbstverständlich finden auch die Zugriffskontrolle (Access Control) sowie die Verwaltung der Rechte (Authorisierung) der User innerhalb der jeweiligen Applikation statt. Noch immer findet man auch Coventional-off-the-shelf-(COTS-)Applikationen, die diese Architektur vorweisen.

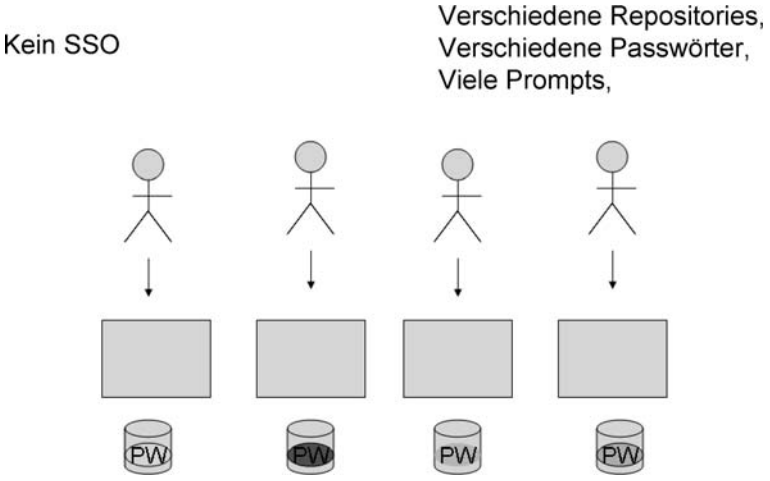

Abb. 6.1 Separate Authentisierung durch jede Einzelanwendung

Erst in der Integration in der Firma kommen die Probleme zu Tage: Jetzt müssen die speziellen Registries abgeglichen werden, sei es durch automatische Datenbankreplikationen oder durch selbst geschriebene Extract- und Importprogramme. Im Falle des Ein- oder Austritts von Mitarbeitern müssen alle Registries auf den neuesten Stand gebracht werden. Dies ist kaum einmal in der Realität der Fall, so dass oft noch nach Jahren ausgeschiedene User bei bestimmten Applikationen registriert sind.

Die User leiden in diesem Szenario an einer Vielzahl von Identitäten und Passwörtern, die sich kaum vereinheitlichen lassen – können doch zum Beispiel die Applikationen verschiedene eingebaute Regeln zum Passwort Aging halten. Die folgende Topologie erlaubt deshalb die Vereinfachung der Passwortbehandlung durch den User, indem die verschiedenen Beziehungen zwischen User und Applikation hinter einem Frontend versteckt werden (Abb. 6.2).

Diese Architektur wird als ein Beispiel für die Realisierung des so genannten *Single-Sign-On* (SSO) praktiziert und wird uns später bei der Behandlung von Single-Sign-On für Portale wieder begegnen. Beim Single-Sign-On hat der Benutzer nur noch ein Passwort für verschiedene Applikationen zu verwalten (und wird auch nur einmal danach gefragt). Die Bequemlichkeit für den User wird jedoch mit einem permanenten Aufwand für das Abgleichen der Registries erkauft. Allerdings gibt es für diese Aufgabe verschiedene Werkzeuge, die entweder durch Replikation der Datenbank oder durch regelbasierten Update die Daten abgleichen.

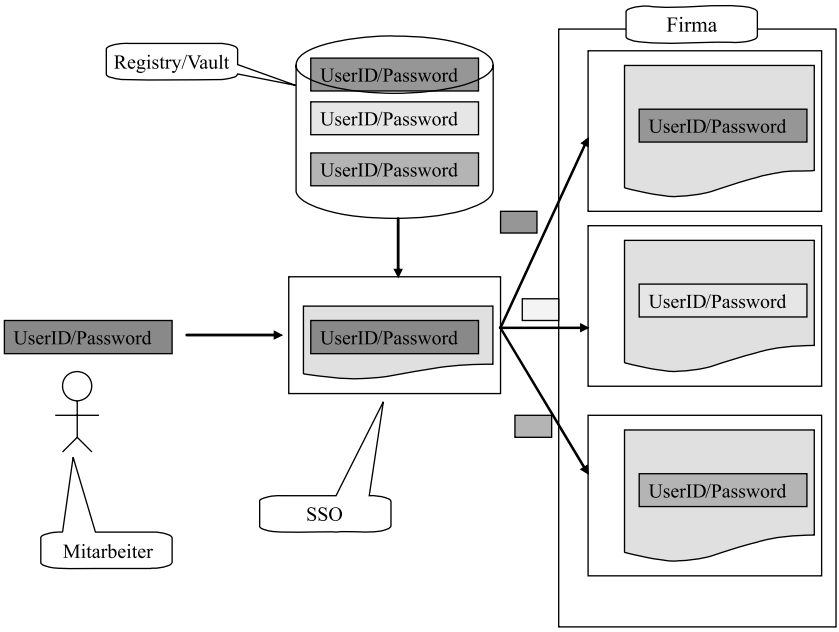

Abb. 6.2 SSO Frontend mit Legacy-Anschluss

Hinzu kommt noch eine andere Überlegung: Wir haben oben die passwortbasierte Authentisierung als „schwache Authentisierung“ bezeichnet. Mit dieser Architektur öffnen wir nun eine ganze Reihe von Applikationen durch eine einmalige, schwache Authentisierung.

Um diese Probleme zu umgehen, kann eine dezentrale Alternative eingesetzt werden: Eine Smartcard als „Digital Wallet“ zur Aufbewahrung aller nötigen Passwörter. Man spricht auch von einem *Hardware Security Module* (HSM), das zur Speicherung von Schlüsseln, aber auch zur Generierung von Signaturen und anderen Krypto-Operationen verwendet werden kann. In unserem Fall wird es lediglich zur Speicherung der Passwörter eingesetzt (Abb. 6.3).

Hier authentisiert sich der User mit einer PIN gegenüber der Smartcard. Je nach Applikation wird dann die korrekte UserID/Passwort-Kombination ausgelesen und zur Anmeldung verwendet. Somit sind in diesem Fall keine Änderungen an den Applikationen oder deren Registries nötig. Nun zur zweiten Frage: Wird von den Applikationen jeweils ein lokales oder eine gemeinsame Registry benutzt? Bei der Nutzung einer gemeinsamen Registry ist jetzt darauf zu achten, dass sie in der Lage sind, die Authentisierung eines Users gegenüber einem konfigurierbaren LDAP vorzunehmen (Abb. 6.4).

In einem zweiten Schritt können dann die UserID und Passwort Kombinationen vereinheitlicht werden (Abb. 6.5).

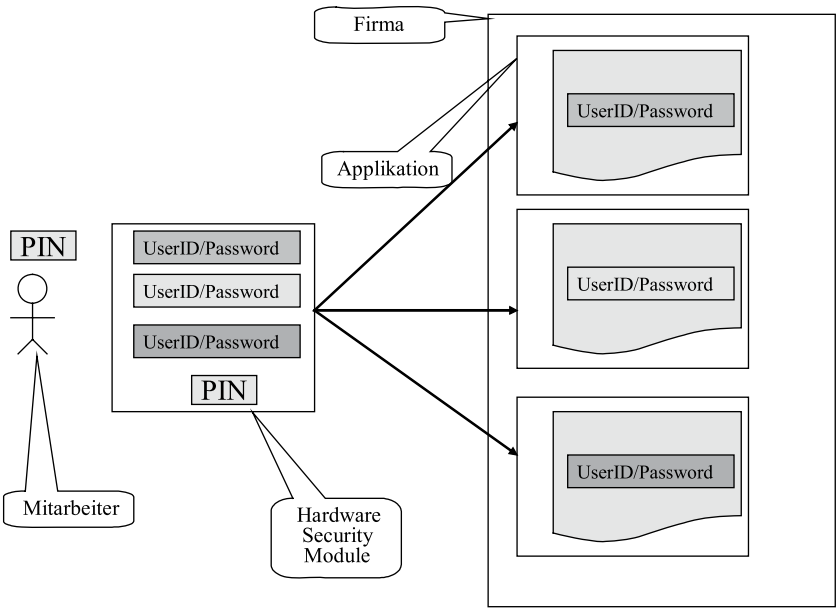

Abb. 6.3 Smartcard-basierter SSO mit Legacy-Anschluss

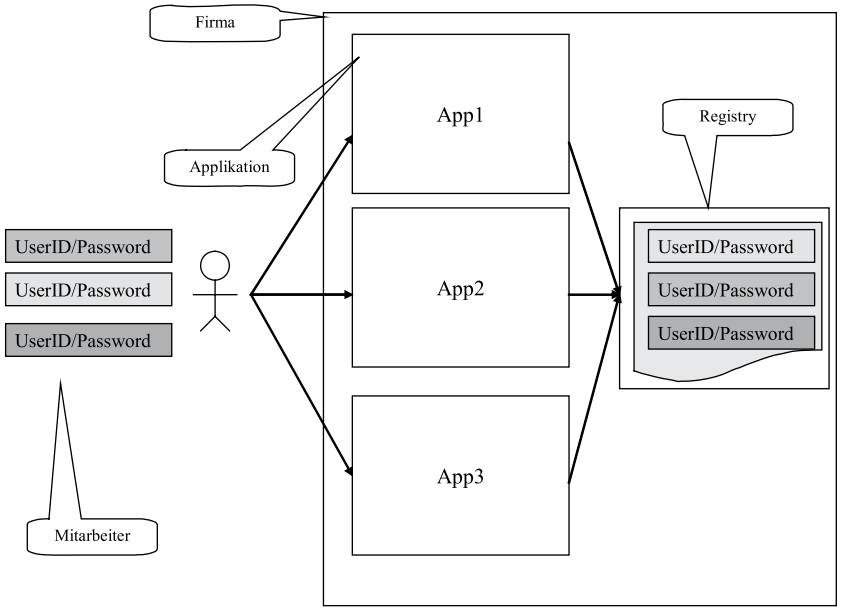

Abb. 6.4 Anpassung der Applikationen auf eine gemeinsame Registry

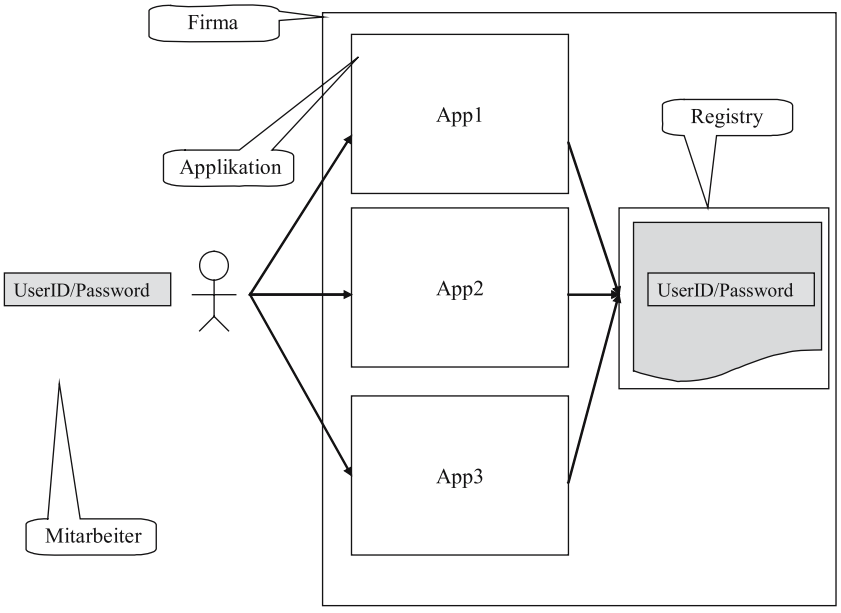

Abb. 6.5 Synchronisierung der Passwörter

Auch hier stellt sich das gleiche Problem wie oben: Jetzt schützt nur noch eine schwache Authentisierung eine ganze Reihe von Applikationen. Das hat ganz praktische Konsequenzen: Konnte vorher ein User aufstehen und in die Mittagspause gehen, ohne sich bei einer Applikation abzumelden, so bestand Gefahr nur für diese spezielle Applikation bzw. deren Daten. Verlässt der User jetzt bei Nutzung einer gemeinsamen Registry seine Arbeitsstation im eingeloggten Zustand, so stehen potentiell alle Applikationen, auf denen er Rechte besitzt, für Fremde offen.

Des Weiteren gelangen die User Credentials weiterhin an jede beteiligte Applikation. Nehmen wir nun an, eine Applikation besitzt einen Fehler oder wurde manipuliert, so kann jetzt diese Applikation andere Applikationen im Namen des Users aufrufen. Sie kann ihn auf beliebigen Applikationen impersonifizieren, ohne dass der User dies will – im schlimmsten Fall sogar, ohne dass überhaupt ein aktueller User-Request vorlag, indem die manipulierte Applikation die Credentials vom letzten Request speichert und wieder verwendet (Abb. 6.6).

Schauen wir uns obiges Szenario etwas genauer an, so stellen wir fest, dass es keine Möglichkeit gibt, zu prüfen, ob die Weiterleitung des Requests von App2 an App3 im Sinne des Mitarbeiters erfolgt (also legal ist) oder ob App2 manipuliert wurde. Der Mechanismus, mit dem App3 den Sender authentisiert, ist in beiden Fällen der gleiche, nämlich mittels UserID und Passwort. Das bedeutet, dass App3 auch gar keine Vorstellung von anderen Formen der Authentisierung besitzt, beispielsweise einer prüfbaren Delegation von Aufträgen an Dritte.

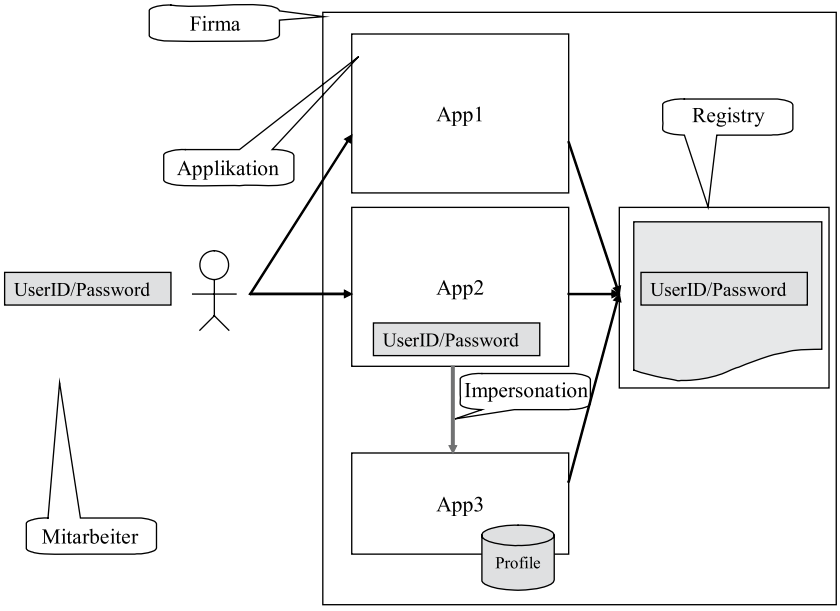

Abb. 6.6 Impersonation bei Nutzung gemeinsamer Passwörter

Letztlich bedeutet dies, dass nicht nur die Speicherung von Credentials in jeder Applikation ein Problem darstellt, sondern dass die Authentisierung durch jede Applikation selbst das zentrale Problem ist.

Solange wir also im Rahmen der Authentisierung mittels UserID und Passwort zulassen, dass alle Applikationen die Credentials erhalten, können wir nicht von einem sicheren Single-Sign-On sprechen und verfügen auch nicht über einen sicheren Delegationsmechanismus. In den folgenden Kapiteln werden wir Authentisierungsmethoden kennen lernen, die auf symmetrischen wie auch asymmetrischen Verfahren beruhen, und die sowohl sicheres Single-Sign-On wie auch eine sichere Delegation erlauben. Zuvor fassen wir noch einmal die Ergebnisse unserer Diskussion der passwort-basierten Authentifikation zusammen:

- Mit Passwörtern allein gibt es keine sichere Delegation.
- Passwörter skalieren nicht mit der Anzahl der Systeme und Applikationen, wenn nicht eine gefährliche Synchronisation betrieben wird.
- Auf jeden Fall sollte die Speicherung der Passwörter nicht in der Applikation selber stattfinden, sondern ausgelagert werden an eine spezielle gemeinsame Registry.
- Ein Single-Sign-On (SSO) ist zwar im Prinzip realisierbar, hat jedoch bedeutende Nachteile und verringert die Sicherheit des Gesamtsystems beträchtlich.
- Hardware Security Module können die User von der Verwaltung vieler Passwörtern entlasten, bieten jedoch ebenfalls kein wirkliches SSO. Dies wird für den User sichtbar, wenn er etwa eine Aufforderung einer einzelnen Applikation oder eines einzelnen Services bekommt, ein bestimmtes Passwort sei zu erneuern.

6.3 Delegation und Impersonation

Die Akteure (das heißt Personen oder Maschinen) in einem verteilten System interagieren miteinander, direkt oder über dritte und weitere Beteiligte. Um diese Interaktionen abbilden zu können, sind die Entwickler von Applikationen mit Begriffen wie „Delegation“, „Impersonation“ oder „Propagation“ konfrontiert, deren Bedeutung je nach Kontext und Autor zu wechseln scheint, und die zudem untereinander Abhängigkeiten besitzen. Die Übersetzung der englischen Begriffe ins deutsche sorgt zudem für Unklarheiten. In der obigen Diskussion der Authentisierung in verteilten Systemen haben wir manche dieser Begriffe bereits angerissen. Im Folgenden wollen wir diese Begriffe genauer klären und versuchen, exakte Definitionen zu geben.

6.3.1 Begriffsklärung

Betrachten wir zur Einstimmung und Verdeutlichung der Begriffe zunächst folgende Szenarien aus dem täglichen Leben:

1. Jemand kommt in eine Polizeikontrolle und muss den Personalausweis vorzeigen. Die Polizisten vergleichen Gesicht mit dem Bild im Ausweis und können die Person so *identifizieren*.
2. Eine gehbehinderte ältere Frau bittet eine Nachbarin, für sie einen bestimmten Geldbetrag vom Automaten zu holen und gibt ihr dazu ihre EC-Karte und ihre PIN. Damit setzt sie Vertrauen (*Trust*) in die Nachbarin. Sie ist zwar Initiator (*Originator*) der Abhebung, *delegiert* sie aber an die Nachbarin.
   Die Nachbarin geht mit der Karte und PIN zum Automaten und hebt Geld ab. Sie *impersonifiziert* die ältere Frau, denn der Geldautomat hat keine Möglichkeit, zu entscheiden, wer die Abhebung tatsächlich vorgenommen hat. Deshalb kann die Nachbarin auch das in sie gesetzte Vertrauen missbrauchen und zum Beispiel einen größeren Betrag als gewünscht abheben oder den Kontostand der älteren Frau einsehen. Falls sie selbst verhindert sein sollte, hat sie die Möglichkeit, die EC-Karte mit PIN an ihre Tochter weiterzugeben, also die Abhebung weiter zu delegieren.
3. Alice bittet Bob, ein Auto auf ihren Namen zuzulassen. Dazu stellt sie Bob (dem *Stellvertreter* oder *Proxy*) eine Vollmacht aus. Die Aufgabe, das Fahrzeug zuzulassen, wird also delegiert. Die Zulassungsstelle überprüft nun

   a) die Identität des Stellvertreters Bob
   b) die Gültigkeit der Vollmacht
   c) die Tatsache, dass sich die Vollmacht auf Bob bezieht.

   Wenn alle drei Prüfungen erfolgreich sind, so behandelt die Zulassungsstelle Bob wie den Fahrzeughalter Alice – allerdings ohne dabei zu vergessen, dass es sich nur um einen Stellvertreter handelt, der „im Auftrag agiert“. Versucht Bob nun mit der gleichen Vollmacht, Geld von Alices Konto bei der Bank nebenan abzuheben (also Alice zu impersonifizieren bzw. sich als Alice zu *maskieren*), so wird er scheitern, da er sich nicht als Alice authentifizieren kann und sich seine Vollmacht nur auf die Zulassung eines Autos beschränkt.
4. Einem Agenten A wird von seiner Dienststelle ein Passwort zugesandt. Damit soll er am Ziel mit einem anderen, ihm persönlich nicht bekannten Agenten B Kontakt aufnehmen und sich durch das Passwort ausweisen. Das Passwort wird jedoch vom Geheimdienst abgefangen und dazu genutzt, anstelle von A einen eigenen Agenten X zu B zu schicken. Mit Hilfe des Passworts kann sich X gegenüber B als A ausgeben (so genanntes *masquerading*).
5. Die Kfz-Zulassungsstelle benötigt zur Durchführung einer Zulassung gewisse vertrauliche Fahrzeugdaten vom TÜV. Es gibt nun mehrere Möglichkeiten an diese Daten zu kommen:

   a) Die Zulassungsstelle schickt die Antragstellerin Carol direkt zum TÜV. Dort wird ihre Identität anhand des Ausweises geprüft.
   b) Ein Mitarbeiter der Zulassungsstelle ruft beim TÜV an, identifiziert sich, da er schon häufiger angerufen hat, über seine Stimme und fordert die Daten an.
   c) Die Zulassungsstelle ruft beim TÜV an und identifiziert sich wie unter b). Dann erklärt sie, dass sie die Daten im Auftrag von Carol benötigt.

d) Die Zulassungsstelle handelt wie unter d), faxt jedoch gleichzeitig eine von Carol ausgestellte Vollmacht zur Herausgabe der Daten an den TÜV.
e) Gleicher Ablauf wie unter d), jedoch ruft der TÜV zusätzlich bei Carol an und prüft dadurch die Richtigkeit der Delegation über die Zulassungsstelle.

In diesen Szenarien aus dem täglichen Leben unterscheiden wir sehr genau zwischen einem Auftraggeber (*Originator*) und einem Mittelsmann oder Stellvertreter (*Proxy*). Hingegen reden wir im Umfeld verteilter Systeme oft wesentlich ungenauer nur vom „Requestor", „Caller", „Sender" oder „Client", ohne dabei anzudeuten, ob es sich wirklich um den Originator oder eine dazwischen geschaltete Instanz handelt, die im Auftrag agiert.

Im Folgenden wird häufig der Begriff *Originator* verwendet, und zwar im Sinne eines Akteurs, der sowohl die Inhalte einer Kommunikation (Nachrichten) erzeugt als auch der Absender einer Nachricht ist. Splitten sich diese beiden Eigenschaften auf, so unterscheiden wir zwischen *Sender* (Absender einer Nachricht) und *Autor* (Erzeuger einer Nachricht). Der Sender einer Nachricht kann dies im Auftrag des Autors tun oder nicht. Im ersten Fall spricht man auch von einem Stellvertreter, im zweiten Fall handelt es sich um einen Angreifer. Es muss also geprüft werden, ob der Sender tatsächlich im Auftrag des Autors handelt. Dabei ist wichtig, wo diese Prüfung stattfindet, welche Eigenschaften sie besitzt und auf welche Weise das Ergebnis der Prüfung weiter gegeben wird.

Auch beim Auftrag (*Request*) selbst unterscheiden wir in unserem sozialen Umfeld genau zwischen dem, was ein Originator ursprünglich gewollt hat und dem, was im Rahmen der Weitergabe des Auftrags von anderen getan wurde. Unsere Beispiele zeigen, wie wichtig es ist, dem Stellvertreter keine globalen Vollmachten zu erteilen, sondern diese eng auf den delegierten Auftrag zu begrenzen.

Aus Business-Sicht steht hinter der Frage nach dem Originator einer Nachricht natürlich oft schlicht die Frage, wer etwas bestellt, also letztlich wer etwas zu verantworten hat. Technisch ausgedrückt ist dies die Frage nach der Nicht-Abstreitbarkeit (Non-Repudiation) von Aufträgen oder Handlungen.

Damit kommt der ersten Prüfung der Identität des Originators im verteilten System eine besondere Bedeutung zu. Die Stelle, die eine Authentisierung durchführt, kann einen unmittelbaren Beweis der Identität eines Originators erhalten, indem der Originator nachweist, dass er einen behaupteten Namen besitzt. Dies kann nur derjenige sein, der tatsächlich den Request stellt, wie unser Beispiel 3 zeigt: Nur der persönlich anwesende Stellvertreter Bob kann unmittelbar bei der Zulassungsstelle authentisiert werden, und zwar durch eine Nachprüfung seiner Behauptung („Mein Name ist Bob") durch einen Proof-of-Possession (Gesicht und Ausweis). Der Personalausweis garantiert dabei den Zusammenhang von Name und Gesicht – ausgestellt von einer allgemein akzeptierten Autorität (Trusted Third Party – in diesem Fall die Bundesdruckerei).

Schwieriger wird die Prüfung der Gültigkeit der Vollmacht. Natürlich muss darin stehen, dass Bob berechtigt ist, das Auto in Alices Namen zuzulassen. Um aber sicher zu stellen, dass Bob sich die Vollmacht nicht selbst ausgestellt hat, muss sie eine Unterschrift des Auftraggebers Alice beinhalten. Um diese Unter-

schrift beurteilen zu können, benötigt die Vollmacht noch eine Kopie des Personalausweises des Auftraggebers, Damit kann der Zusammenhang der Identität „Alice“ mit ihrer Unterschrift geprüft werden und anschließend die Übereinstimmung der Unterschrift im Ausweis mit der Unterschrift unter der Vollmacht. Wiederum ist an dieser Stelle Vertrauen nötig in den Herausgeber des Ausweises.

6.3.2 Delegation von Aufträgen

Wir sprechen von *Delegation*, wenn ein Akteur (so genannter *Intermediate*), der nicht der Originator ist, einen Auftrag (Request) zur weiteren Bearbeitung an einen anderen Akteur schickt. Dieser Akteur kann Endpunkt (*Target*) des Requests oder ein weiterer Intermediate sein. In den Beispielen oben ist die hilfsbereite Nachbarin ein Intermediate der gehbehinderten älteren Dame, Bob ein Intermediate von Alice und die Zulassungsstelle ein Intermediate der Autobesitzerin Carol. Das Beispiel der älteren Dame, die ihre geheime Authentisierungsinformation (in diesem Fall die PIN) weitergibt, zeigt, wie schädlich diese Form der Delegation ist, da sie dem Intermediate eine vollständige Impersonation des Originators erlaubt. Eine Einschränkung der Vollmachten ist nicht möglich, sowenig wie die Möglichkeit, auf Seiten der Bank nachzuvollziehen, wer genau die Transaktionen durchgeführt hat. Schließlich hat die ältere Dame die Möglichkeit, die mit Hilfe ihrer ec-Karte und PIN durchgeführten Aktionen abzustreiten – ob sie damit vor Gericht Erfolg hätte, ist eine andere Frage. Wenden wir uns nun dem fünften Beispiel aus dem letzten Abschnitt zu und schauen uns an, wie das Target TÜV mit den an es gerichteten Anfragen umgeht. Dazu untersuchen wir die Optionen des TÜV zur Authentisierung von Anfragen:

a) Persönliches Erscheinen des Antragstellers mit Personalausweis
b) Anfragen von Mitarbeitern der Polizei – oder von der Polizei selbst
c) Anfragen von Mitarbeitern der Zulassungsstelle – oder der Zulassungsstelle selbst, sofern sie im Zusammenhang mit einem konkreten Auftrag zur Zulassung stehen
d) Anfragen von authorisierten TÜV Mitarbeitern
e) Telefonische Anfragen von Privatleuten mit Namensnennung

Im Fall a) haben wir das schon bekannte Authentisierungsverfahren, mit dem auch Bob als Stellvertreter von Alice bei der Zulassungsstelle authentisiert wurde. Es benützt ein von einer Trusted-Third-Party ausgestelltes Zertifikat bei gleichzeitiger Prüfung gegen ein Biomerkmal (Gesicht).

Im Fall b) erhält eine ganze Institution Zugriff auf die Daten und zwar ohne Einschränkung. Das bedeutet, es muss entweder eine eigene Identität für die Institution „Polizei“ geschaffen werden und alle Mitarbeiter der Polizei können diese Identität beweisbar annehmen (die Institution impersonifizieren bzw. sich als diese maskieren) oder alle Mitarbeiter der Polizei erhalten persönliche Identitäten im System des TÜV. Eine dritte Variante besteht darin, dass Mitarbeiter sich unter

ihrer eigenen, von der Polizei vergebenen Identität anmelden. Dies setzt voraus, dass eine gemeinsame Domain (oder auch Realm) für die Rechner der beiden Ämter eingerichtet wurde. Der Rechner des TÜV vertraut dann dem Polizeirechner und übernimmt die dort ermittelte Identität des Mitarbeiters. Dasselbe gilt im Fall c), jedoch mit der zusätzlichen Einschränkung, dass die Anfrage nur dann erfüllt wird, wenn sie im Zusammenhang mit einem existierenden Auftrag steht. Das bedeutet, der TÜV möchte wissen, für wen der Auftrag auszuführen ist. Die Zulassungsstelle kann dazu entweder nur den Namen des Auftraggebers oder aber dessen Vollmacht plus Ausweiskopie an den TÜV weitergeben. In der ersten Variante muss das Target viel Vertrauen in den Intermediate setzen. Werden weitere Daten wie eine Vollmacht zur Auftragsvergabe weitergereicht, kann der TÜV die Gültigkeit in ähnlicher Weise prüfen, wie es die Zulassungsstelle selbst mit der von Alice für Bob ausgefüllten Vollmacht getan hat.

Die Variante d) bietet einen indirekten Zugang für externe Stellen zu den Daten: Die authorisierten Mitarbeiter könnten theoretisch ohne jeden Auftrag beliebige Daten ermitteln und weitergeben.

Variante e) ermöglicht letztlich jede Form von Impersonation oder Maskierung, da die im Telefongespräch behauptete Identität lediglich übernommen, jedoch nicht geprüft wird. Dies stellt natürlich einen besonders gefährlichen Umgang mit Authentisierungsdaten in verteilten Systemen dar.

Zusammenfassend halten wir folgendes fest:

- Die unmittelbare Authentisierung eines Originators ist oft nur an einem Punkt (der Stelle die den unmittelbaren Kontakt mit dem Originator besitzt) eines verteilten Systems möglich. Nachgeordnete Akteure (Intermediates) besitzen die Möglichkeit der Authentisierung mangels Authentisierungsdaten meist nicht.
- Die Weitergabe von geheimen Daten zur Authentisierung (zum Beispiel eines Passworts) ist eine Form von schädlicher Impersonation.
- Im Falle der Verwendung von PKI basierter Authentisierung auf Seiten des Originators ist eine Weitergabe der Authentisierungscredentials (des privaten Schlüssels) ohnehin nicht zulässig und die einfache Impersonation des Originators durch Intermediates entfällt.

Daraus ergeben sich als sicherheitstechnische Anforderungen an den Umgang mit delegierten Requests in verteilten Systemen:

- Der Originator authentisiert sich an der Eintrittsstelle in eine Infrastruktur gegenüber einer Stelle. Intermediates, die im Verlauf eines Requests aufgerufen werden, erhalten keine Credentials des Originators, die zu einer unmittelbaren Authentisierung verwendet werden könnten (z. B. Geheimnisse). Die Intermediates vertrauen auf die Authentisierung der Eintrittstelle, erhalten aber Kenntnis der Identität des Originators und können in seinem Auftrag handeln bzw. die dabei verwendeten Rechte auf diejenigen des Originators einschränken.
- Die unmittelbare und gegenseitige Authentisierung von Intermediates untereinander oder gegen Targets ist unbedingt nötig, garantiert dem Target aber nicht, dass der Intermediate wirklich im Auftrag des Originators handelt.

- Es sollten nur solche Aktionen ausgeführt werden, die mit dem Willen, aber auch den Rechten des Originators in Einklang stehen. Das bedeutet, die beteiligten Systeme sollten sich im Sinne der Business-Logik so verhalten, als wäre der Originator persönlich auf jedem System angemeldet und hätte dort eine initiale Authentisierung vollzogen.
- Idealerweise sollte eine Kette von Daten existieren, die vom Originator über die beteiligten Intermediates bis hin zum Target genau belegt, wer was mit welcher Identität und mit welcher Berechtigung getan hat.

Wir werden in den folgenden Kapiteln technische Möglichkeiten kennen lernen, diese Anforderungen zu erfüllen. Anzumerken wäre noch, dass der Originator bei seiner Anmeldung an der Eintrittsstelle in einigen Standards auch als „Holder of Key“ bezeichnet wird, da er ja die unmittelbaren Authentisierungs-Credentials besitzt. Idealerweise benötigt der „Holder of Key“ seine Schlüssel nur gegenüber einer Stelle in einem verteilten System, die sich dadurch authentisiert. Dadurch wird eine Verbreitung der Schlüssel verhindert, dennoch müssen nachgelagerte Beteiligte ein hohes Maß an Vertrauen in vorgelagerte Intermediates aufbringen. Denn aus der bloßen Tatsache, dass sich ein Originator erfolgreich authentisiert hat, können nachgelagerte Targets nicht auf den Willen des Originators schliessen. Hier zeigt sich die Grenze kanalbasierter Sicherheit. Bessere Möglichkeiten ergeben sich dann durch objektbasierte Sicherheit wie im Falle der Vollmacht in unserem Beispiel. Mit Vollmachten – also digital signierten Willenserklärungen – lässt sich auch für nachgeordnete Targets der Willen des Originators nachprüfen. Dadurch verringert sich auch die Bedeutung der Infrastruktur für die Sicherheit in einem verteilten System. Vollmachten erlauben darüber hinaus auch eine wesentlich bessere Umsetzung des POLA-Prinzips, da sie die Autorität der Beteiligten auf den jeweils vorliegenden Request des Originators begrenzen können.