Deklarative Sicherheit zur Spezifikation und Implementierung der elektronischen Fallakte

Raik Kuhlisch, Jörg Caumanns

Fraunhofer-Institut für Software- und Systemtechnik ISST
Steinplatz 2
10623 Berlin
raik.kuhlisch@isst.fraunhofer.de
joerg.caumanns@isst.fraunhofer.de

Abstract: Anwendungen zur sektorübergreifenden Kommunikation im Gesundheitswesen stellen für deren Betreiber (z. B. Kliniken) große Investitionen mit oftmals unsicherem Marktpotenzial dar. Es ist daher wesentlich, dass die einmal aufgebauten Dienste für eine Vielzahl von Anwendungen mit- und nachnutzbar sind. In diesem Papier wird am Beispiel der elektronischen Fallakte (eFA) dargestellt, wie Konzepte einer deklarativen Sicherheit dazu beitragen, bestehende Dienste flexibel an sich verändernde Sicherheitsanforderungen anzupassen bzw. parallel in unterschiedlichen Sicherheitskontexten zu nutzen.

1 Motivation

Die elektronische Fallakte (eFA) erlaubt einen fallbezogenen Austausch von medizinischen Daten zwischen Leistungserbringerorganisationen. Hierzu werden bestehende Systeme in Kliniken und Praxen über ein föderiertes Peer-to-Peer Netzwerk von Registern und Speichern virtuell integriert [CBN07]. Der Patient willigt explizit in das Anlegen und Nutzen einer Fallakte ein und berechtigt Ärzte oder Einrichtungen für den Zugriff auf seine Daten. Die durch den besonders schützenswerten Status der ausgetauschten Daten bedingten Anforderungen werden durch ein Datenschutzkonzept und ein übergreifendes Sicherheitskonzept adressiert[1].

Die elektronische Fallakte basiert auf einer serviceorientierten Architektur, die strikt in Sicherheits- und Anwendungsdienste getrennt ist. Verschiedene Schutzmechanismen in der eFA adressieren die hohen Datenschutzanforderungen an die Verarbeitung medizinischer Daten mit Patientenbezug. Adäquate Schutzmechanismen beantworten Fragen wie etwa: Wer darf welchen Dienst nutzen bzw. wer darf welche Daten zuordnen, zusammenführen, lesen oder ändern? Dedizierte Sicherheitsdienste übernehmen die Aufgaben der Identifizierung/Authentifizierung, Pseudonymisierung, sowie Autorisierung von Leistungserbringern [BSI07].

[1] Siehe http://www.fallakte.de für die vollständigen Spezifikationen zur elektronischen Fallakte.

1.1 Deklarative Sicherheit

Sicherheitsfunktionen zur Erfüllung von Anforderungen an die Vertraulichkeit und Authentizität von geschützten Daten folgen oftmals dem gleichen Muster: Eine die geschützte Ressource kapselnde Anwendung erhält einen Dienstaufruf. Der Aufrufende muss authentisiert und anschließend autorisiert werden. Dies führt zum Zugriff auf entsprechende Ressourcen oder eben nicht. Hier liegt es nahe, solche wiederkehrende Aufgaben in *Security Frameworks* auszulagern und zu beschreiben (deklarieren), welche Sicherheitsfunktionen zu aktivieren sind, um einer übergeordneten Sicherheitsstrategie gerecht zu werden. Die Ausübung dieser Sicherheitsfunktionen ist dann Sache des Frameworks. Dieser Schutzziele und Sicherheitsdienste betonende deklarative Sicherheitsansatz steht im Gegensatz zur programmierten Sicherheit, bei der konkrete Sicherheitsmechanismen und –objekte fest an die Implementierung der Anwendungslogik gebunden werden.

Deklarativen Elemente zur Vereinfachung und Flexibilisierung der Umsetzung von Sicherheitsvorgaben im Bereich der Authentisierung und Autorisierung sind mittlerweile Bestandteil verschiedener *Business-Level Frameworks* (z. B. OASIS ebXML Business Processes [ebXMLBP]) und werden auch von J2EE unterstützt [SUN08]. Ziel ist es hierbei immer, die für eine Komponente oder Kommunikationsbeziehung geltenden Sicherheitsziele zu beschreiben und die Auswahl des geeigneten Mechanismus dem Framework zu überlassen.

Im Umfeld von Web Services stellen *WS-Policy* [WSPOL1.5] und der darauf aufbauende *WS-SecurityPolicy* Standard [WSSPOL1.2] Sprachen bereit, um Anforderungen an einen Dienstaufruf zu beschreiben. In sogenannten *Policy Assertions* wird so das Verhalten des Web Service deklarativ gesteuert. Die Implementierung des Web Service kann sich somit auf die Fachlogik konzentrieren und die Durchsetzung der Sicherheitsanforderungen als vorausgesetzt ansehen.

Diese Nutzung deklarativer Sicherheit bietet eine Reihe von Vorteilen:

- Sicherheitsdienste und –mechanismen können ohne Änderungen am Code der Anwendungsdienste weiterentwickelt und an neue Anforderungen angepasst werden

- In einem Framework enthaltene Stubs der Sicherheitsdienste können sehr einfach an bestehenden Anwendungen angebunden werden. Hierdurch wird eine Entkopplung von Sicherheitsdiensten - z. B. für Authentifizierung und Autorisierung – unterstützt [EDOC07].

- Die Kongruenz der Umsetzung von Sicherheit zu ihrer Spezifikation und einer übergeordneten Sicherheitsrichtlinie ist explizit gegeben, d. h. deklarative Sicherheit kann unmittelbar aus dem Sicherheitskonzept abgeleitet werden

- Die umgesetzten Sicherheitsmechanismen und die genutzten Objekte werden an der Schnittselle der Dienste sichtbar und damit überprüfbar.

Ein Nachteil der deklarativen Sicherheit sind die durch eine Interpretation der Deklarationen bedingten Performanzeinbußen. Performanzanalysen der elektronischen Fallakte am Fraunhofer ISST und bei Siemens haben jedoch gezeigt, dass schon durch einfache Mechanismen wie z. B. das Caching von Deklarationen auch bei einer intensiven Nutzung deklarativer Sicherheit eine sehr gute Performanz erzielt werden kann.

1.2 Deklarative Sicherheit am Beispiel der elektronischen Fallakte

Die Sicherheitsdienste der elektronischen Fallakte sind so ausgelegt, dass sie auch für eine Vielzahl weiterer Anwendungen zur Kooperation von Leistungserbringern genutzt werden können. Um dieses Ziel zu erreichen, erfolgt die Kopplung zwischen Sicherheits- und Anwendungsdiensten soweit als möglich deklarativ. D. h. für jede Schnittstelle wird beschrieben, welche Sicherheitsobjekte beim Aufruf bereitzustellen sind und welche der eFA-Sicherheitsdienste diese Objekte bereitstellen. Das daraus abgeleitete Konzept des *Assertion-Chaining* ist in Kapitel 2 dieses Papiers beschrieben.

Die Hauptaufgabe der eFA-Sicherheitsarchitektur ist es, einen Satz authentischer Berechtigungsregeln (*Access Policy)* eines authentisierten Nutzers als Teil des *SOAP Security Headers* bei jedem Aufruf eines Anwendungsdienstes mitzuliefern. Vor jeder geschützten Ressource ist ein *Policy Enforcement Point* (PEP) positioniert, der sicherstellt, dass nur die *Access Policy* erfüllende Zugriffe zugelassen werden. Da *Policy* und Nutzerattribute an der Schnittstelle des Anwendungsdienstes bereitgestellt werden, muss dieser für eine *Policy*-Entscheidung lediglich noch Attribute der angeforderten Ressource beifügen. In Kapitel 3 dieses Papiers wird beschrieben, wie die hierzu genutzten XACML-Policies aufgebaut sind, um ein rollenbasiertes Berechtigungsmanagement durchzusetzen.

Ein wesentliches Element von service-orientierten Sicherheitsarchitekturen sind sog. *Security Token*, die eine Entkopplung von Sicherheitsdiensten erlauben [EDOC07]. Standards wie *WS SecurityPolicy* erlauben eine deklarative Steuerung des Austauschs von *Security Token* (siehe Kapitel 2). Leider gibt es momentan jedoch keinen dedizierten Standard, um den einer Sicherheitsrichtlinie entsprechenden Aufbau eines *Security Token* zu beschreiben, so dass die Ausstellung und Verifizierung der *Token* zumindest in Teilen deklarativ erfolgen könnte. In der Referenzimplementierung der elektronischen Fallakte wird der XACML-Standard genutzt, um die Verifizierung eines *Security Token* am konsumierenden Dienst zu flexibilisieren. Wie hiermit eine deklarative Verifizierung von Elementen einer Sicherheitsrichtlinie umsetzbar ist, wird in Kapitel 4 dieses Papiers beschrieben.

2 Deklarative Sicherheit bei der Schnittstellenspezifikation

Die Sicherheitsarchitektur der elektronischen Fallakte definiert für jeden Anwendungsdienst mittels *WS-Policy* und *WS-SecurityPolicy* welche Sicherheitsnachweise (*Security Assertions,* z. B. kodiert als *Security Token)* notwendig sind, um die Operationen aufrufen zu können. SAML *Assertions* [SAML2.0] kodieren die notwendigen

Authentisierungs- und Autorisierungsinformationen, welche von speziellen *Security Token Services* ausgestellt werden. Hierbei kann ein *Security Token Service* zur Ausstellung eines geforderten Sicherheitsnachweises (*Security Assertion*) die Vorlage eines *Security Token* verlangen, dass in die Zuständigkeit eines anderen *Security Token Service* fällt. Auf diese Weise können Abhängigkeiten in Sicherheitsdiensten (z. B. Autorisierung erfordert Authentifizierung) auf sequentielle Ketten von Sicherheitsnachweisen abgebildet werden (siehe Abbildung 1).

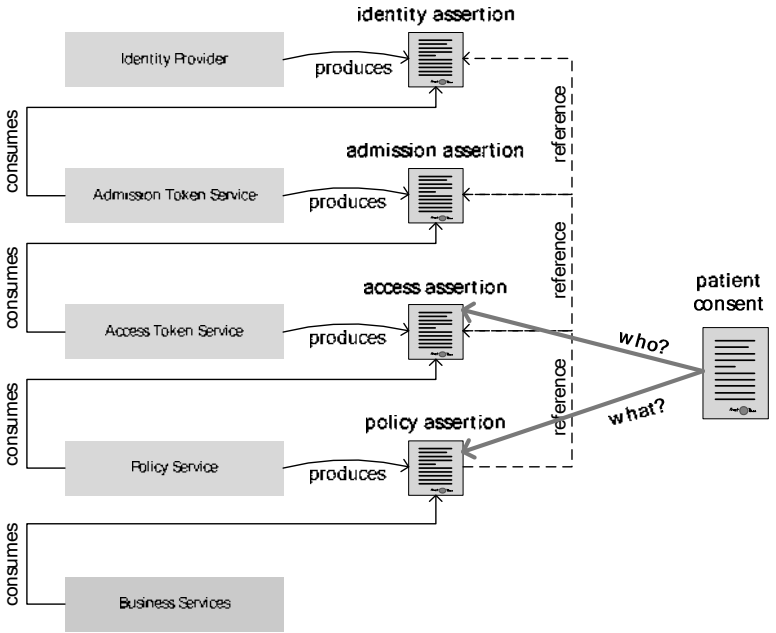

Abbildung 1: Authentisierungs- und Autorisierungsinformationen als Assertion-Kette

In der Deklaration der zur Umsetzung der Schutzziele beizubringenden Sicherheitsobjekte können „klassische" X.509 Token und *Security Context Token* mit SAML Token kombiniert werden. Die eFA-Sicherheitsarchitektur erlaubt es auch, mehrere SAML Token an einen Web Service zu versenden, d. h. iterativ ganze Ketten von aufeinander verweisenden Sicherheitsnachweisen aufzubauen. Hiermit können die Nachweise der Nutzung einzelner Sicherheitsdienste (Authentisierung, Pseudonymisierung, Autorisierung, etc.) als vom unterliegenden Security Framework zu bearbeitende Bestandteile des Aufrufs eines Anwendungsdienstes kodiert werden[2].

Die Einbettung mehrerer SAML *Assertions* in das SOAP *Security Header* stellt auf der Implementierungsseite vielerlei Ansprüche: Zum einen muss die Semantik der

[2] Dieses Muster wird u. a. auch im europäischen epSOS-Projekt (www.epsos.eu) genutzt, um einem Anwendungsdienst Nachweise zu der in einem anderen Land erfolgten Nutzerauthentisierung und –autorisierung zu übermitteln.

Assertions selbst (strukturierte Elemente in Attributen) und zum anderen die korrekte Kombination einer *Assertion*-Kette überprüft werden können. Einem Aufrufenden muss zudem ein Besitznachweis für diese SAML *Assertions* abverlangt werden. Um unterschiedliche Sicherheitsanforderungen für die verschiedenen Vertrauensbeziehungen zwischen Dienstnutzern und -anbietern deklarieren zu können, wird für jede Vertrauensbeziehung in der Schnittstellenspezifikation ein eigener Port definiert. So können z. B. sehr einfach unterschiedliche *Policies* für Zugriffe aus verschiedenen Sicherheitszonen heraus definiert werden (z. B. Zugriffe innerhalb eines *Circle-of-Trust* und in einen *Circle-of-Trust* hinein).

Die folgende Abbildung verdeutlicht die Komplexität verschiedener Schnittstellen mit den Policies, in denen deklariert ist, welche Sicherheitsnachweise bei Aufruf eines Dienstes aus verschiedenen Sicherheitskontexten heraus jeweils beizubringen sind (z. B. wird bei einem Aufruf von „außen“ ein expliziter Authentisierungsnachweis verlangt, während beim Aufruf durch Dienste innerhalb eines definierten *Circle-of-Trust* aufgrund von vorab aufgebauten *WS-SecureConversation* Kanälen solche Nachweise nicht erforderlich sind).

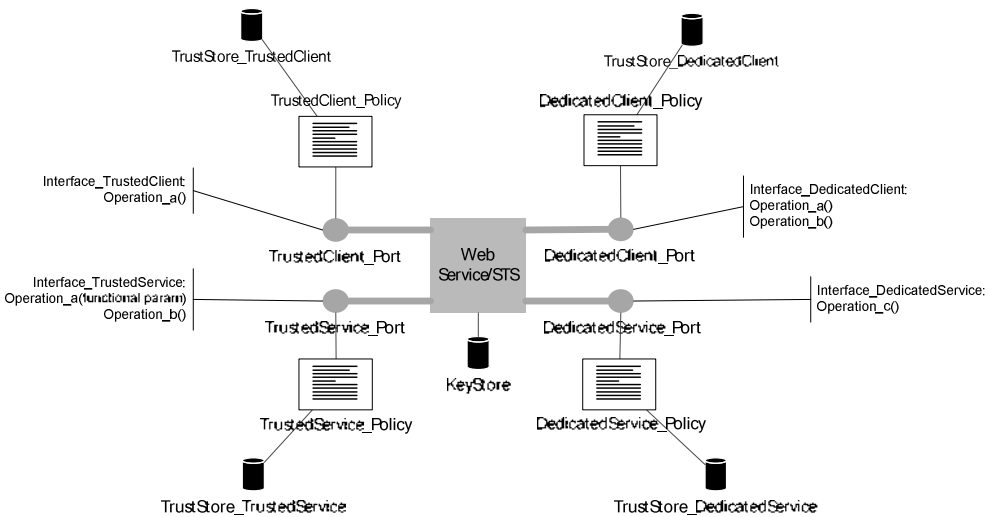

Abbildung 2: Web Service Schnittstellen mit differenzierten Security Policies

3 Deklarative Autorisierung

Das eFA Berechtigungsmanagement setzt Autorisierungen über den Standard XACML v2.0 [XACML2.0] um. Die Zugriffsrechte einzelner Rollen auf Fallakten werden in Zugriffsregeln (Access *Policies)* festgelegt. Durch Bindung von Rollen an Personen kann ein dedizierter Dienst (*eFA Access Token Service*) die für den jeweiligen Nutzer

geltenden Zugriffsregeln ermitteln und einen entsprechenden Autorisierungsnachweis ausstellen, der an die *Assertion*-Kette des authentisierten Nutzers angefügt wird.

Die XACML-Spezifikation definiert sowohl die *Policy*-Sprache mit entsprechendem Datenmodell als auch ein *Request/Response*-Protokoll für die Anfrage von Autorisierungsentscheidungen. Eine *Access Policy* besteht aus einer oder mehreren Regeln. Um konkurrierende Regeln zu synchronisieren, werden Kombinationsalgorithmen definiert (z. B. deny-overrides). Jede Regel gewährt oder verweigert den Zugriff auf eine Kombination aus Subjekt (Aufrufender Nutzer), Ressource, Benutzeraktion und Ausführungsumgebung. An eine Regel können noch weitere optionale Bedingungen gestellt werden. Die Zugriffskontrolle gestaltet sich vereinfacht wie folgt: Ein *Policy Enforcement Point* (PEP) unterbricht den Kontrollfluss vor der geschützten Ressource und stellt eine Autorisierungsanfrage (XACML *Request*) an einen *Policy Decision Point*[3] (PDP). Der PDP lädt im System vorhandene Zugriffsregeln über einen *Policy Administration Point*, welcher für die Erstellung und Verwaltung der *Policies* zuständig ist. Die Auswertung der Zugriffsregeln gegen einen konkreten Zugriffsversuch durch den PDP resultiert in einer Autorisierungsentscheidung, welche dem PEP in einem XACML *Response* zurückgesandt wird. Nur bei einer positiven Autorisierungsentscheidung gibt der PEP die weitere Ausführung des Kontrollflusses – und damit den Zugriff auf die geschützte Ressource - frei.

Um ein flexibles *Policy*-Design zu ermöglichen, sieht die Sicherheitsarchitektur der elektronischen Fallakte einen Indirektionsschritt bei der Ausstellung einer Zugriffsberechtigung vor. Der Autorisierungsnachweis bzw. die *Access Assertion* enthält ein XACML *PolicySet* mit Subjekt und Fallakten-ID, welcher ein oder mehrere Zugriffs-Policies zugewiesen werden. Die *Policy* selber ist in einer sog. *Policy Assertion* enthalten, die anhand einer *Access Assertion* von einem dedizierten Dienst (*Policy Token Service*) abgerufen werden kann. Diese Trennung von Policy-ID und eigentlicher Policy erlaubt es, mit den gleichen Sicherheitsdiensten sowohl Sicherheitsarchitekturen aufzubauen, die (wie die eFA) deklarativ kodierte *Policies* nutzen als auch Anwendungsdiensten die Umsetzung impliziter Regelwerke zu überlassen (z. B. IHE BPPC, wo Consent-Modelle über IDs ausgerückt werden und die Abbildung dieser Modelle auf den Daten zugeordnete *HL7 Confidentiality Codes* durch das Dokumenten-Register erfolgt).

```
<xacml:PolicySet
  PolicyCombiningAlgId="policy-combining-algorithm:permit-overrides">
  <xacml:Target>
    <xacml:Subjects>
      <xacml:Subject>
        <xacml:SubjectMatch MatchId="xacml:1.0:function:base64Binary-
         equal">
          <xacml:AttributeValue>
            VYDKTmgZ6JarY2xJ8RFUO5e35bYgewAIhLVFtfEN0Cg=
          </xacml:AttributeValue>
```

[3] Fehlen Informationen für die Evaluierung einer Policy oder ist ein PEP nicht in der Lage, einen XACML Request zu erzeugen, kann ein Context Handler genutzt werden, welcher die Daten z. B. durch Nutzung eines Policy Information Points aufbereitet und dann an den PDP sendet.

```
          <xacml:SubjectAttributeDesignator
            AttributeId="urn:ecr:attributes:user-information-hash"/>
        </xacml:SubjectMatch>
      </xacml:Subject>
    </xacml:Subjects>
    <xacml:Resources>
      <xacml:Resource>
        <xacml:ResourceMatch MatchId="xacml:1.0:function:string-equal">
          <xacml:AttributeValue>
            1.3.6.1.4.1.778.51.623.3.1.20.163f20c8
          </xacml:AttributeValue>
          <xacml:ResourceAttributeDesignator
            AttributeId="urn:ecr:attributes:record-id"/>
        </xacml:ResourceMatch>
      </xacml:Resource>
    </xacml:Resources>
  </xacml:Target>
  <xacml:PolicyIdReference>
    urn:ecr:names:xacml:2.0:default:policyid:read-only
  </xacml:PolicyIdReference>
  <xacml:PolicyIdReference>
    urn:ecr:names:xacml:2.0:default:policyid:deny-all
  </xacml:PolicyIdReference>
</xacml:PolicySet>
```

Abbildung 3: Deklarative Rechtezuweisung mit Referenzen

In diesem Beispiel wird einem Subjekt (in diesem Falle ein Hashcode über einen Benutzer mit seinen Rollen)[4] für eine Fallakte das Leserecht gewährt. Die Referenzen (read-only, deny-all) können genauso gut implizite Zugriffs-Policies und keine weiteren XACML Policies bedeuten, welche dann jedoch bei der Zugriffsprüfung im Programmcode ausgewertet werden können. Dies ist umsetzungsspezifisch. Die eFA-Sicherheitsarchitektur lässt jedoch noch komplexere und damit feingranularere Zugriffsbeschreibungen mit diesem Mechanismus zu.

Dieser Aspekt der deklarativen Sicherheit geschieht in der Datenbasis des *Access Token Service*, bei der einer Kombination aus Subjekt und Fallakte semantisch die Zugriffsrechte zugewiesen werden. Wie die referenzierten Zugriffs-Policies tatsächlich ausgestaltet sind, ist bei der Vergabe der Berechtigungen an dieser Stelle nicht von Interesse.

Das *Policy Management* definiert, was sich hinter diesen sogenannten Referenzen verbirgt. Dazu werden sogenannte Building Blocks, also wiederverwendbare Policy-Bausteine, definiert. Die Policy-IDs aus der *Access Assertion* zeigen jeweils genau auf solch einen Block. Das folgende Beispiel zeigt einen Ausschnitt einer Zugriffs-Policy für Leseoperationen – in diesem Fall die Berechtigung, eine Ordnerliste einer Fallakte abzurufen.

```
<xacml:Policy
  PolicyId="urn:ecr:names:xacml:2.0:default:policyid:read-only">
```

[4] Mehr Flexibilität kann dadurch erreicht werden, wenn den zugewiesenen funktionalen Rollen Zugriffsrechte zugewiesen werden.

```
  <xacml:Target />
  <xacml:Rule Effect="Permit"
    RuleId="urn:ecr:names:xacml:2.0:default:rules:
    ecrfolderregistry_ecrfolderregistrydedicatedclient_getfolderlist">
    <xacml:Target>
      <xacml:Resources>
        <xacml:Resource>
          <xacml:ResourceMatch MatchId=" xacml:1.0:function:string-equal">
          <xacml:AttributeValue>
            {http://isst.fhg.de/ecr/application/ecrfolderregistry}
            ECRFolderRegistry
          </xacml:AttributeValue>
          <xacml:ResourceAttributeDesignator
            AttributeId="xacml:2.0:profile:webservices:v1.1:service"/>
        </xacml:ResourceMatch>
        <xacml:ResourceMatch MatchId="xacml:1.0:function:string-equal">
          <xacml:AttributeValue>
            {http://isst.fhg.de/ecr/application/ecrfolderregistry}
            ECRFolderRegistryDedicatedClientSoap12HttpPort
          </xacml:AttributeValue>
          <xacml:ResourceAttributeDesignator
            AttributeId="xacml:2.0:profile:webservices:v1.1:port"/>
        </xacml:ResourceMatch>
        <xacml:ResourceMatch MatchId="xacml:1.0:function:string-equal">
          <xacml:AttributeValue>
            {http://isst.fhg.de/ecr/application/ecrfolderregistry}
            getFolderList
          </xacml:AttributeValue>
          <xacml:ResourceAttributeDesignator
            AttributeId="xacml:2.0:profile:webservices:v1.1:operation"/>
          </xacml:ResourceMatch>
        </xacml:Resource>
      </xacml:Resources>
    </xacml:Target>
  </xacml:Rule>
  <!—
    Add more permit rules with read operations.
  -->
</xacml:Policy>
```

Abbildung 4: XACML Policy als Building Block

Um die Policy-Evaluierung in einem *Security Provider* eines Anwendungsdienstes zu vereinfachen, brauchen die Zugriffs-Policy-Referenzen aus dem XACML *PolicySet* der *Access Assertion* lediglich mit den Building Blocks ersetzt zu werden. Das Ergebnis ist eine von einen XACML Framework direkt verarbeitbare Zugriffs-Policy. Eine entsprechende Autorisierungsanfrage (XACML *Request*) an einen *Policy Decision Point* übergibt die Fallakten-ID (Ressource), das aufrufende Subjekt und die aufzurufende Aktion (Web Service Operation) auf der Ressource.

4 Deklarative Sicherheit bei der Prüfung von Security Token

Wie oben geschildert, bieten *WS-Policy* und *WS-SecurityPolicy* die Beschreibungsmöglichkeit, um nichtfunktionale Anforderungen (z. B. Sicherheit, Datenschutz oder

garantierte Zustellung) in XML für eine Kommunikation zu formulieren.[5] Ein Operationsaufruf eines Web Service wird entsprechend einer Anforderungsbeschreibung geprüft und ggf. zugelassen oder verweigert (*WS-SecurityPolicy Enforcement* und *Decision*). Der Einsatz von Web Service Frameworks bietet hier schon eine umfangreiche Unterstützung. Dazu zählen u. a. die Prüfung von Zeitstempeln und Signaturen oder Möglichkeit der Anforderung von *Security Token*.

Spezielle *Security Token*[6] wie das SAML Token können von einem Web Service Framework lediglich auf Wohlgeformtheit, allgemeine Attribute (Gültigkeit, etc.) oder Signatur geprüft werden. Hinzu kommt noch die Möglichkeit, die Authentizität des Einreichers eines Token über *Holder-of-Key SAML Assertions* nachprüfen zu können. Alles was darüber hinaus geht, wird von Frameworks nicht mehr adäquat adressiert. Die Semantik der SAML Token kann mit bestehenden WS-* Spezifikationen derzeit beispielsweise nicht beschrieben werden. Auch deren Überprüfung muss folglich manuell d. h. im Programmcode erfolgen.

XACML Policies für die Anforderungsbeschreibung von Security Token

Das Autorisierungsframework XACML lässt sich nicht nur dazu einsetzen, Web Service Operationen zu speziellen Ports zu schützen (*Web Services Profile of XACML* [WS-XACML]), sondern auch um die Korrektheit von SAML Token festzustellen. Existierende Open-Source XACML Frameworks wie SunXACML oder Enterprise Java XACML erlauben die schnelle Evaluierung von Autorisierungsentscheidungen bei einem gegebenen Pool von Zugriffs-Policies. Komfortable Funktionen wie das *Decision* und *Policy Caching* ermöglichen gute Antwortzeiten auch bei umfangreichen Sicherheitsprüfungen.

SAML Token bzw. SAML *Assertions* entsprechen immer einem definierten Profil, so dass die verarbeitenden Systeme auf die Existenz bestimmter Attribute in einer *Assertion* vertrauen. Beispielsweise akzeptiert ein Service Provider eine SAML *Assertion* als Authentisierungsnachweis nur, wenn diese von einem vertrauten *Identity Provider* ausgestellt und der Nutzer sich mit einem X.509 Zertifikat authentisiert hat (In diesem Fall existiert dann ein Attribut *AuthenticationMethod* in einem *AuthenticationStatement* mit dem Wert „urn:oasis:names:tc:SAML:1.0:am:X509-PKI“).

Entsprechend dem Profil der SAML *Assertion* kann eine XACML *Policy* erstellt werden, wonach die einem Dienst vorgelegte SAML *Assertion* mit den existierenden Mechanismen des Sicherheitsframeworks validiert wird. Als Beispiel dient im Folgenden die *Identity Assertion* aus der elektronischen Fallakte, welche eine Ausprägung eines SAML Token ist und einen Authentisierungsnachweis mit Attributen für einen Nutzer (Leistungserbringer) von Fallakten darstellt [EFASEC1.2]. Eine wichtige Eigenschaft von XACML ist die Möglichkeit, Werte aus einer

[5] Dies kann mit entsprechender Tool-Unterstützung (z. B. NetBeans) Assistenten-basiert oder manuell in der Policy (XML-Datei) erfolgen.
[6] Dazu zählen z. B. Username Token, Binary Security Token (X509Token), XML Security Token (SAML Token).

Autorisierungsanfrage mittels XPath-Ausdrücken zu ermitteln. Dies macht es möglich, ein Anforderungsprofil eines beliebigen XML-Dokuments (in unserem Fall die *Identity Assertion*) maschinenlesbar zu definieren und zu überprüfen.

Das folgende Beispiel zeigt eine XACML *Policy* für eine *Identity Assertion*. Mittels XPath-Ausdrücken wird z. B. geprüft, ob dem Subjekt eine entsprechende eMail-Adresse zugeordnet wurde. Das in der eFA definierte Profil für die *Identity Assertion* ist wesentlich komplexer, so dass die korrespondierende XACML *Policy* die Attributwerte der SAML *Attribute Statements* ebenfalls beschreiben muss. Aus Platzgründen wurde hier auf die vollständige Darstellung verzichtet.

```
<Policy PolicyId="identity-assertion-policy"
        RuleCombiningAlgId="deny-overrides">

  <PolicyDefaults>
    <!-- Define the URI for the XPath 1.0 specification. -->
    <XPathVersion>
       http://www.w3.org/TR/1999/Rec-xpath-19991116
    </XPathVersion>
  </PolicyDefaults>

  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="xacml:1.0:function:string-equal">
          <AttributeValue>
            urn:oasis:names:tc:SAML:1.0:assertion
          </AttributeValue>
          <ResourceAttributeDesignator
            AttributeId="..target-namespace" />
        </ResourceMatch>
      </Resource>
    </Resources>
  </Target>

  <Rule RuleId="identity-assertion-rule" Effect="Permit">
    <Description>The assertion is valid, if ...</Description>
    <Target/>
    <Condition>
      <Apply FunctionId="xacml:1.0:function:and">
        <Apply FunctionId="xacml:1.0:function:integer-equal">
          <Apply FunctionId="xacml:1.0:function:xpath-node-count">
            <AttributeSelector
                  RequestContextPath="//xacml-context:Request/node()"/>
            </Apply>
              <AttributeValue>1</AttributeValue>
            </Apply>

            <!-- Node Values -->
            <Apply FunctionId="xacml:1.0:function:string-equal">
              <Apply
                FunctionId="xacml:1.0:function:string-one-and-only">
                <AttributeSelector RequestContextPath=
                      "//*[local-name()='ConfirmationMethod']/text()"/>
              </Apply>
              <AttributeValue>
                urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
```

```
              </AttributeValue>
            </Apply>

            <!-
            Define value for attribute 'format' in
            subject name identifier
            -->
            <Apply FunctionId="xacml:1.0:function:string-equal">
              <Apply
                FunctionId="xacml:1.0:function:string-one-and-only">
                <AttributeSelector
                  RequestContextPath="/Request/Resource/ResourceContent/
                  Assertion/AttributeStatement/Subject/NameIdentifier/
                  Format" />
              </Apply>
              <AttributeValue>
                urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress
              </AttributeValue>
            </Apply>

            <!-
            Check other values according to the assertion's profile
            -->
          </Apply>
        </Apply>
      </Apply>
    </Condition>
  </Rule>
</Policy>
```

Abbildung 5: XACML Policy für eine SAML Assertion

Token-Prüfung mit XACML

Einem XACML Framework muss nun das *Security Token* bzw. die SAML *Assertion* übergeben werden. Hierzu wird von einem XACML *Policy Enforcement Point* (PEP) bzw. einem *Context Handler* des genutzten Frameworks ein XACML *Request* erstellt und dann einem XACML *Policy Decision Point* (PDP) für eine Autorisierungsprüfung (in unserem Fall eine Token-Prüfung) zugesandt. Dieser XACML *Request* enthält als *Resource Content* die SAML *Assertion* selbst. Ein weiteres Attribut definiert den Namensraum der SAML *Assertion*, um das entsprechende *ResourceMatch* der XACML *Policy* zu erwirken.

```
<xacml:Request>
  <xacml:Subject/>
  <xacml:Resource>
    <xacml:ResourceContent>
      <saml:Assertion> ... </saml:Assertion>
    </xacml:ResourceContent>
    <xacml:Attribute
      AttributeId="urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
      DataType="http://www.w3.org/2001/XMLSchema#anyURI">
      <xacml:AttributeValue>
        urn:oasis:names:tc:SAML:1.0:assertion
      </xacml:AttributeValue>
```

```
    </xacml:Attribute>
  </xacml:Resource>
  <xacml:Action/>
  <xacml:Environment/>
</xacml:Request>
```

Abbildung 6: XACML Request für eine Token-Prüfung

Die Vorteile der Nutzung deklarativer Sicherheit zur Prüfung von SAML *Assertions* liegen insbesondere in der dadurch erreichbaren Flexibilität in Bezug auf die definierten SAML Profile. So können z. B. bestehende Profile weiterentwickelt oder sogar verschiedenen Profile parallel genutzt werden, ohne dass eine Anpassung des Programmcodes der verarbeitenden Dienste erforderlich wäre. Soll beispielsweise in der Zukunft der Zugriff auf eine Fallakte nur nach Authentifizierung mit einem Heilberufsausweis oder aus einem über eine SMC geschützten Kontext erfolgen können, muss lediglich die XACML *Policy* zur Verifizierung des entsprechenden Attributs der *Identity Assertion* geändert werden. Hiermit ist im Endeffekt die Anbindung bestehender eFA-Implementierungen an die Sicherheitsobjekte der Telematikinfrastruktur möglich, ohne auch nur eine Zeile Programmcode ändern zu müssen. Insbesondere können Anpassungen dieser Art auch vom Sicherheitsadministrator des eFA-Providers ohne Erfordernis der Einbeziehung des KIS-Herstellers vorgenommen werden, wodurch zusätzliche Flexibilität gewonnen wird.

Die folgende Abbildung fasst den Einsatz von XACML noch einmal zusammen:

(I) Ein SOAP *Request* wird einem Service Provider zugesandt und das Web Service Framework prüft diesen entsprechend der *WS-Policy/WS-SecurityPolicy* des Web Service;

(II) Ein Security Provider oder *Interceptor* triggern einen PEP, welcher die im SOAP-Aufruf vorhandene SAML *Assertion* in einen XACML *Request* überführt und an den PDP sendet.

(III) Ist die Autorisierungs- bzw. Token-Prüfung positiv, wird die eigentliche Autorisierung der Web Service Operation überprüft.

(IV) Die Web Service Operation wird ausgeführt und der Zugriff auf die Ressource gewährt.

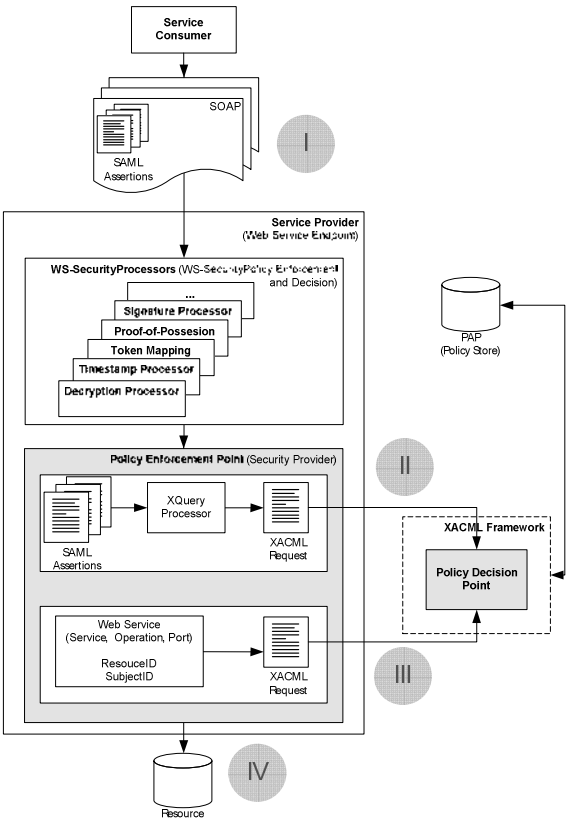

Abbildung 7: Token-Prüfung und Zugriffskontrolle mit XACML

4. Zusammenfassung

In diesem Papier wurde am Beispiel der Sicherheitsarchitektur der elektronischen Fallakte beschrieben, wie durch Nutzung deklarativer Sicherheit eine hoher Grad an Flexibilität und Evolutionsfähigkeit erreicht werden kann. Hierbei wird deklarative Sicherheit nicht nur zur Externalisierung von Berechtigungsregeln genutzt, sondern auch um die Umsetzung von Sicherheitsanforderungen an *Security Token* zu verifizieren. Dadurch dass Zugriffe auf einen Web Service, die aus verschiedenen Sicherheitszonen kommen, auf verschiedene Ports abgebildet werden, ist dieser Ansatz insbesondere auch geeignet, um Anwendungen einer sektorübergreifenen Kommunikation zu vereinfachen. Gerade hier erfolgen Zugriffe auf geschützte Ressourcen sowohl von internen als auch von externen Systemen, die jeweils unterschiedlichen Sicherheitsannahmen unterliegen und potenziell auch unterschiedliche Sicherheitsobjekte nutzen.

Literaturverzeichnis

[BC07] Boehm, O.; Caumanns, J.: Föderatives Identitätsmanagement am Beispiel der elektronischen Fallakte. Informatik-Spektrum, Nr.4/07, 2007; S. 240-250.

[BSI07] Caumanns, J.; Boehm, O.; Neuhaus, J.: Elektronische Fallakten zur sicheren einrichtungsübergreifenden Kommunikation. In (Bundesamt für Sicherheit in der Informationstechnik): Innovationsmotor IT-Sicherheit. SecuMedia Verlag, 2007.

[CBN07] Caumanns, J.; Boehm, O.; Neuhaus, J.: Elektronische Fallakten zur einrichtungsübergreifenden Kooperation. E-HEALTH-COM, Nr.1/07, 2007; S. 68-70.

[ebXMLBP] Dubray, J.; Amand, S.; Martin, M. (Eds.): ebXML Business Process Specification Schema, Technical Specification v2.0.4. OASIS Standard, Dezember 2006. http://docs.oasis-open.org/ebxml-bp/2.0.4/OS/spec/ebxmlbp-v2.0.4-Spec-os-en.pdf

[Edoc08] Boehm, O.; Caumanns, J.; Franke, M.; Pfaff, O.: Federated Authentication and Authorization: A Case Study. In: Proc. 12th IEEE International EDOC Conference. 16.8.08. München, 2008;.S. 356-362.

[EFASEC1.2] Boehm, O.; Kuhlisch, R.: Sicherheitsarchitektur der elektronischen Fallakte. Version 1.2. Februar 2008.

[SAML2.0] Cantor, S.; Kemp, J.; Philpott, R.; Maler, E. (Eds.): Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS Standard, März 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[SUN08] Singh, I.; Stearns, B.; Johnson, M.: Designing Enterprise Applications with the J2EE Platform, Second Edition. http://java.sun.com/blueprints/guidelines/ designing_enterprise_applications_2e/ security/security4.html

[WS-XACML] Anderson, A. (Ed.): Web Services Profile of XACML Version 1.0. Working Draft 8, Dezember 2006. http://www.oasis-open.org/committees/download.php/21490/ xacml-3.0-profile-webservices-spec-v1.0-wd-8-en.pdf

[WSPOL1.5] Vedamuthu, A. et al (Eds.): Web Services Policy 1.5 – Framework. W3C Recommendation. September 2007. http://www.w3.org/TR/ws-policy/

[WSSPOL1.2] Nadalin A. et al (Eds.): WS SecurityPolicy 1.2. OASIS Standard, Juli 2007. http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.pdf

[XACML2.0] Moses, T. (Ed.): eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard, Februar 2005. http://docs.oasis-open.org/xacml/ 2.0/ access_control-xacml-2.0-core-spec-os.pdf