A Case Study in Decentralized, Dynamic, Policy-Based, Authorization and Trust Management – Automated Software Distribution for Airplanes

Peter Hartmann[1] , Monika Maidl[2], David von Oheimb[2], Richard Robinson[3],

[1]Landshut University of Appl. Sciences, Am Lurzenhof 1, 84036 Landshut, Germany
peter.hartmann@fh-landshut.de
[2]Siemens Corporate Technology, Otto-Hahn Ring 6, 80200 München, Germany
{monika.maidl, david.von.oheimb}@siemens.com
[3]Boeing Research & Technology, P.O.Box 3707, MC 7L-70, Seattle, WA 98127-2207, USA
richard.v.robinson@boeing.com

Abstract. We apply SecPAL, a logic-based policy language for decentralized authorization and trust management, to our case study of automated software distribution for airplanes. In contrast to established policy frameworks for authorization like XACML, SecPAL offers constructs to express trust relationships and delegation explicitly and to form chains of trusts. We use these constructs in our case study to specify and reason about dynamic, ad-hoc trust relationships between airlines and contractors of suppliers of software that has to be loaded into airplanes.

Keywords: Authorization, trust management, security-tokens, logic, software-distribution

1 Why do we need dynamic, decentralized authorization and trust management?

Electronically collaboration increasingly takes place not only within security domains, but between enterprises and individuals with no pre-established trust relationships. The application areas comprise industrial applications, energy management and distribution, transportation systems, healthcare, and many others. The use case we focus on is the distribution of software ‘parts' to airplanes. The challenge is not only to transport such software parts from the airline to the airplane, but that a range of other parties – suppliers, the airplane manufacturer, and service providers – are involved. There is a strong security requirement that only unmodified parts that have been released by trusted producers are loaded into the airplane.

Triggered by the increasing demand for electronic communication and collaboration over the Internet, there is also a strong trend towards using standard protocols and frameworks as a uniform interface to existing computer systems and programming frameworks, based on standardized XML messages that are exchanged between various parties. In particular, grid computing and web services, based on XML/SOAP are used in scientific computing, automated business to business and business to customer scenarios (services) or cloud scenarios like Software-as-a Service. The use of web services has been standardized by the WS-* family of standards [1].

Many of these use cases have strong security requirements, and the classic security mechanisms based on enterprise perimeters protection (firewalls, DMZ etc.) are not suitable for dynamic, perimeter-crossing collaborations. In particular, mechanisms for the authentication of externals are required, as incorporating externals into the internal user management is considered to be inflexible and costly: accounts and access rights have to be managed, in case of changes in positions at a business partner, and accounts have to be removed when employees leave a company.

So new security models on top of http and web services communication have been developed, in particular SAML, SOAP message security (WS-* security standards), XACML and CardSpace [1,2]. The core of these approaches is the use of *security tokens*. Security tokens, e.g. SAML tokens or InfoCards, are short-lived, signed expressions that are used to transfer authentication status and attributes across domain boundaries. This means that the authentication decision is delegated to an external party, namely the one issuing the security token. We will discuss in Section 1.2 how security tokens form the basis of dynamic, decentralized authentication.

1.1 Authorization

Just as with authentication, established domain-centric security mechanisms do not fit the requirements of cross-domain collaboration. Authorization has to be decided by the local authority, i.e. the owner of the application or resource. Typically, this is done by ACL (access control lists) on operating system level, or by application-specific authorization models like RBAC (e.g. in SAP). When externals are involved, both approaches do not work, as externals are typically not covered by the provisioning processes of an enterprise.

In contrast to authorization within an enterprise, authorization decisions for externals critically depend on trust. There are many aspects of trust decisions: whether to trust the authentication mechanism, to what extent the individual or their organization is trusted, the criticality of the application or action that is requested, and so on. In addition, in many scenarios including our case study, chains of trusts are formed. Today, authorization and trust often is handled implicitly. An example is that users implicitly trust all the root certificates contained in the certificate store of their operating system or browser. Typically, such a certificate store contains up to several hundreds of certificates, and decisions which of these are still trusted are difficult and unreliable. [3] discusses this problem and describes possible attacks.

Hence in order to handle authorization in cross-domain collaboration scenarios, a flexible language is required to explicitly address authorization in the context of trust relationships. In the next section we explain the difference between long-term trust and dynamic, ad-hoc trust relationships in order to make clear what features such a language should have.

1.2 Dynamic Ad-hoc Trust Relationships

It is important to distinguish between *stable trust relationships*, which are often bilateral, and those relationships that are set up in a *dynamic, ad-hoc way*. Typically, users have a stable trust relationship within their domain. Typical features of long-term stable relationships are:

- Authentication by long-term credentials (password, PKI certificate, …).
- User accounts are managed (change of password, roles, etc.).
- User accounts are removed only when the user leaves the domain.

PKI (Public Key Infrastructure) certificates are also security tokens in the sense discussed above, in that they provide transferable proof of identity. However, PKI certificates are only issued within a stable trust relationship, as PKI certificates typically have a validity time of up to several years, and accordingly are issued under strict procedures and rules only. Inter-domain use of PKI certificates is also long-term, by formally setting up cross certification or bridge CAs, or implicitly by inserting root CA certificates in certificate stores of operating systems or browsers.

In contrast, the trust relationship between e.g. a user and a service provider, or two businesses which want to collaborate is often dynamic and ad-hoc. As already mentioned, short-lived security tokens can be used in such settings to transfer authentication status and other attributes between business partners or services without having stable trust relationships in place beforehand.

The issuer of a security token could be the organization of the individual whose identity or other attributes are stated, or a third party that is trusted by both the individual's organization and the recipient. For example, an owner of an airplane that needs software updates might not directly trust the supplier of the airplane manufacturer, but trusts the manufacturer. So the manufacturer can issue a signed security token to the supplier, stating the supplier's identity, and the supplier sends this security token to the airplane owner. That establishes a dynamic trust relationship between the supplier and the airline.

Dynamic trust might be established in chains, i.e. dynamic trust relationships might again be used to set up other dynamic relationships. As an example, a supplier might have contractors to develop software, and issue security tokens for the contractor. Using this token, the dynamic trust relationship between the supplier and the airline can be used to set up a dynamic trust relationship between the contractor and the airline.

The recipient of a short-term security token relies on the information conveyed in the token to verify the PKI signature of the token and trusts the signer to make these statements, so long-term PKI trust has to be established. The issuer can only issue tokens to an individual or an organization that it can authenticate reliably. So ultimately, dynamic trust relationships ultimately have to be built on stable trust relationships.

Security tokens can carry more information than just authentication information. Any sort of statement could be made, e.g. that a company is an official supplier of another company, or that an employee has certain responsibilities. Using such attributes plays an important part in trust decisions because such statements can be used as input to authorization decisions that take into account not only the identity of the subjects of the security token, but also asserted properties of them. In our case study, we use security tokens that state that a certain software part has been approved, or that a service provider is certified to handle airplanes of a certain type.

To summarize, we list typical features of dynamic, ad-hoc trust relationships:

- Authentication by short-term *security tokens*, i.e. delegation of authentication to trusted parties.
- Chains of trust.
- Attribute-based authentication.

Hence short-term *security tokens* are the basic building blocks of dynamic, ad-hoc trust relationships. For the establishment and management of dynamic relationships, a framework is required to explicitly reason about attributes, delegation and chains of trust. We will explain in Section 3.2 how SecPAL matches these requirements.

2 Case Study in Automated Software Distribution to Airplanes

In the past, distribution of embedded software packages to mobile platforms such as airplanes was accomplished manually. In a typical scenario, the owner or maintenance provider for a vehicle would receive software updates on physical media, delivered via courier. A mechanic would be given a work order that entails installing software updates directly from physical media onto the airplane systems by means of a CD drive or similar. In the near future, however, software updates will increasingly be delivered over networks in an automated way.

On the way from the software *supplier* to the target device, software items may be handled at intermediate entities: an airplane manufacturer, in our use case Boeing, typically receive software items from suppliers or their *contractors*, and send it to *airlines*, which bear responsibility for the safe operation of their *airplanes*, and have the authorization to send software there. An airline might commission local *service providers* at airports to install software on its airplanes. So the software distribution process consists of several hops, and the software distribution stretches over the IT systems related to the process at each of these entities.

The airline is the central element of the distribution system and has relations to suppliers and Boeing on the one hand and to service providers and airplanes on the other hand. The central role of the airline is justified by the fact that the airline is responsible for all software due to be installed on an airplane and has to approve these software parts.

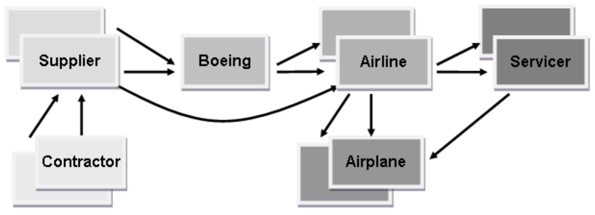


Fig. 1 Software distribution chain

In order to guarantee integrity and authenticity, software parts are signed, possibly in a nested way – e.g. a software package might be signed by a contractor, by a supplier and by the manufacturer. More details on the case study can be found in [4].

We assume that suppliers, the manufacturer, the airlines and the service providers have PKI certificates from some certificate authority, but we do not assume that each entity trusts all respective CA root certificates. In other words, we do not assume that stable trust relationships exist between all parties. This means for instance that the airline or the airplane cannot verify all signatures directly. Instead, we propose the use of ad-hoc dynamic trust relationships, managed by using SecPAL. First we discuss the stable relationships that we assume to exist in the case study.

2.1 Trust Relationships between Suppliers, Manufacturers and Airlines

We assume that there are contractual agreements and a technical set up so that suppliers are able to verify signatures of subcontractors, but neither Boeing nor the airline can. Similarly, Boeing has stable trust relations with suppliers, possibly by using a bridge CA like CertiPath Aerospace Bridge CA [5], but the airlines do not deal with suppliers or contractors directly. Instead, airlines trust Boeing to assert the identity and role of suppliers. We also assume that airlines own a trusted root certificate of Boeing's CA or use the CertiPath Trust Bridge CA, so that the airline can validate Boeing signatures and assertions.

2.2. Trust Relationships between Airline, Service Providers and Airplanes

We assume that airlines run a proprietary CA or buy commercial certificates, and that they have processes in place to equip airplanes with a trusted copy of the airline certificate, so that there is a direct stable trust relationship between the airline and its airplanes. The airplanes are maintained by local service providers who are contracted by the airline. An airline may have several service providers, responsible for maintenance of different airplane types or on different airports. The service providers use maintenance laptops to connect to the airplane or to the ground network, and will have to authenticate themselves to the airplane. As service providers contracts might be short-term, we do not assume stable trust between the airline and service providers. Instead, service providers obtain short-term security tokens from the airline that specify e.g. for which airplane types and during which time period a service provider is authorized to install software.

3 Policy-based Authorization and Trust Management

We use a logic-based language, SecPAL, to model our use case. There are other policy languages that defined in terms of syntactic language constructs, while their semantics, i.e. the evaluation algorithm, is described in an informal way. Examples are XACML and XrML. Such description can be difficult to understand and implement, and in particular for complicated languages that include delegation, like

XrML, subsequent analysis showed problems [14], which is not surprising because designers are likely to build something as complex as a logic framework from scratch. Unclear semantics also makes a language difficult to understand for users.

3.1 SecPAL - Logic-based, Decentralized Authorization

Logic programming is well-suited for expressing trust management, because logic provides a framework for defining attributes, and enables reasoning, in particular about chains of trust. After defining a set of statements or facts, an authorization request can be formulated and logic rules can be used to check whether the authorization request is a valid consequence of these facts.

SecPAL [6,7] is a logic-based policy language that addresses authorization and trust in distributed scenarios that involve frequent ad-hoc collaborations between entities with no pre-established trust relationships.

For example, an airline might decide to accept parts from suppliers and their contractors, where approval from contractors is sufficient for less critical parts, but critical parts require approval by suppliers. This can be expressed in SecPAL as follows:

```
Airline says p is accepted if
    p is type2-critical AND p is approved.
Airline says p is accepted if
    p is type1-critical AND p is supplier-approved.
```

Note that '`is accepted`', '`is type1-critical`' and '`is approved`' are examples of attributes. In SecPAL, a wide range of attributes or authorization primitives ("can read", "is entitled to discount", "has access from [time1] to [time2]") can be defined by the author of the policy.

In our example, the airline does not want to manage the information which parts are type1-critical, and which companies are direct suppliers or contractors. Instead, the airline delegates statements about suppliers and parts to the manufacturer Boeing like the following:

```
Airline says Boeing can say_0 x is type1-critical.
Airline says Boeing can say_0 x is a supplier.
```

Delegation is expressed in SecPAL by the '`can say`' construct.

These requirements are matched by assertions from other parties, for example:

```
Boeing says Part123 is type1-critical.
```

The suffix _0 for '`can say`' means that the statement '`Boeing says Part123 is type1-critical`' is only accepted if it is directly asserted by Boeing, and not derived by a chain of trust, as explained below.

The concept of parts being supplier-approved is defined by delegation:

```
Airline says x can say p is supplier-approved
      if x is a supplier.
```

If the following statements exist:

```
Boeing says Honeywell is a supplier.
Honeywell says Part123 is supplier-approved.
```

then the request "`Part123 is accepted`" can be derived.

Chains of trust are used to derive that a company is a contractor. The following statements are required to derive that 'FlightMedia is a contractor':

```
   Airline says x can say y is a contractor till t
     if x is a supplier AND currentTime < t.
  Airline says x can say that y is a contactor till t1
     if x is a contractor till t2 AND t1 < t2.
  Boeing says Honeywell is a supplier.
  Honeywell says EquipTech is a contractor till 2010.
  EquipTech says FlightMedia is a contractor till 2009.
```

As this example shows, the requirements of dynamic trust relationships we identified in Section 1.2, namely the use of attributes, delegation to trusted parties and chains of trust can be expressed in SecPAL.

3.1.1 SecPAL assertions

Formally, a SecPAL policy consists of several *SecPAL assertions*, which have the following form:

`A says` *fact* `if` $fact_1$ `AND … AND` $fact_n$ `AND constraint.`

where *fact* is a sentence that states properties of principals. All names of principals in an assertion, like *A* in the example, stand for public (signature verification) keys. The principal A is the issuer of the assertion, and signs the assertion.

There are three different types of facts. The first type of *fact* is based on authorization primitives like

`e has access from t1 to t2.`

where e is either a name (like A) or a variable, and where authorization primitives can be defined by the user. The second type of *fact* expresses delegation in the form

`e can say` *fact* (e.g. `B can say c has access from date1 to date2`*).*

Finally, facts can express principal aliasing in the form

`e can act as d.`

Note that delegation ('`can say`') may be nested, as the *fact* that is delegated can itself be a delegation expression.
Assertions are implicitly signed by the issuer, i.e. an assertion '`A says …`' is meant to be signed by A.

Assertion tokens and authorization rules

Informally, we can distinguish between SecPAL assertions that are used to prove identities, attributes or capabilities to another party, which we call *assertion tokens*, and assertions that express *authorization rules*. A service provider specifies authorization rules to state explicitly under which circumstances a user will be granted rights, and states precisely what sort of rights can be obtained. The authorization rules and the delegation rules issued by a service or communication partner together form its *local policy rules.*

In contrast, the assertion tokens are provided by the users who want to access a service in order to prove claims about themselves. Assertion tokens are typically issued by the home domain of a user, or by a trusted third party that can identify the user in a reliable way as part of a stable trust relationship. The policy engine of the service provider will use the user's assertion tokens together with the local policy rules to decide whether the access can be granted.

3.1.2 Solving SecPAL queries

A set of SecPAL assertions, consisting of local policy rules and assertion tokens, determines whether an authorization request to a service is accepted. Authorization requests are called queries in SecPAL. An atomic request is of form

```
e says fact
```

General queries are formed from atomic queries and constraints by the logical connectives AND, OR, NOT and EXISTS x. In the example above, a useful query would be

```
Part123 is accepted.
```

In order to solve queries, SecPAL assertions are translated into a formal logic model, more precisely Datalog [8], which is a restricted logic programming language, i.e., "Prolog without function symbols". The evaluation of a query against a set of SecPAL assertion corresponds to a run of the corresponding Datalog program. During such a run, nested delegation is resolved by iteration, and suitable variable assignments are generated.

The use of an established logic model as a basis offers a clear semantics, and because logic models like Datalog have been well studied in academia, there are results on decidability (i.e. termination) and on runtime complexity. Moreover, implementation methods for such logic models have been developed and iteratively improved. More concretely, for SecPAL these results state that query evaluation is decidable with polynomial time complexity. For practical purposes, such a statement is valuable, as it guarantees that query evaluation is tractable even for the most complex policies.

3.2 Other Security Token Frameworks: SAML, SOAP message security and XACML

As already mentioned in Section 1.2, the use of security tokens is well established by SOAP message security (WS-* security standards), the SAML and the XACML standards. These frameworks have different goals than SecPAL, but as they are widely used in practice and known to a broad audience, we briefly describe them in order to show the differences to SecPAL, thereby making the goals and usage of SecPAL clearer. The token-based security model that underlies the WS-* specifications is explained in [9] and shown in Fig. 2. Security tokens are carried in messages and are used to prove *claims* about the sender of the message, e.g. the name of the sender, or roles and attributes, i.e. attribute-based authentication is possible. Before sending a message, e.g. to request a service, a user has to obtain a suitable security token from a Security Token Services (STS), which is located in the user's domain and hence can authenticate the user as part of a stable trust relationship. Authentication might be via username and password, smart cards, biometrics or any other means. The STS might impose restrictions on authentication methods, e.g. require passwords to have a certain quality, or only allow access at office hours, or restrict the validity period of security tokens to one day. Such requirements form the *policy* of the STS.

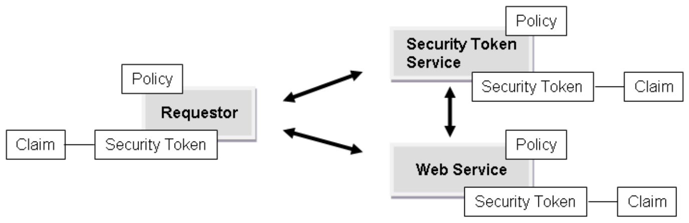


Fig. 2 The web services security model

After obtaining a security token, which is signed by the STS, the user can make a request to a service and include the security token into the request message. The service or application authenticates and authorizes the user by evaluating the security token. The service might impose its own restrictions, e.g. requirements on security tokens like the use of specific cryptographic algorithms, acceptable maximal validity periods and acceptable methods with which the user has been authenticated to the STS. These restrictions form the local policy of the web service.

A widely used type of security token is SAML assertions. The SAML standard also includes protocols for using SAML assertions for Single Sign-on, similar to Fig. 2 but for use in web browsers rather than web services (SOAP) messages.

Neither the WS-* security specifications nor SAML do cover authorization. The XACML standard defines an XML based policy language to describe general access control requirements, including extension points for defining new functions, data types, combining logic, etc. XACML also includes a request/response language to pose queries whether or not a given action should be allowed. XACML can be used in conjunction with SAML, i.e. a user provides a SAML security token to a service, and based on this information and a XACML policy, the services grants access or not. In such a setting, authentication decisions can be delegated and attribute-based authentication and authorization can be used. The main difference to the SecPAL framework is that delegation is implicit, i.e. it is not specified in the local policy of a service who is trusted to make assertions, i.e. there is not equivalent to the '`can say`' construct of SecPAL. Consequently, chains of trust cannot be specified. Another difference is that in XACML there is an implicit central authority that states the policy, whereas in SecPAL, the authorship is explicit, as in '`A says…`'.

3.3 Other Logic-based Authorization and Trust Policy Languages

The use of logic for authorization and delegation was pioneered by the Speaks-for Calculus [10], where the '`says`' construct was introduced. Many later logic-based authorization frameworks and trust management systems are based on variants and extensions of Datalog with constraints [8], e.g. Binder [11], Delegation Logic [12], and Cassandra [13]. SecPAL differs from these languages in that is offers constructs specifically targeted for distributed authorization policies, and that it is very expressive while requiring only a small number of constructs.

Other logic frameworks that have been used to provide semantics for policy languages include first-order logic, e.g. for XrML in [14] and SPKI/SDSI [15], and pushdown automata for SPKI/SDSI [16].

The development of DKAL (Distributed-Knowledge Authorization Language) [17], another logic-based language, was based on SecPAL, and had the goal to remove a potential information leak problem and to increase expressivity by allowing the free use of functions. Apart from the use of function symbols, the main difference to SecPAL is that communication is targeted, i.e. that the basic form of assertions is not '`A says …`' but '`A says … to B`'. In that way it can be avoided that confidential information is leaked by posing queries that are evaluated using confidential assertions. A revised version of DKAL, yet more expressive, is presented in [18] SecPAL can be naturally translated into DKAL, so we could have used DKAL instead of SecPAL. But as confidentiality is not relevant to our use case, and we also do not need to use free functions, we decided to use SecPAL. An important reason for our decision to use SecPAL was that an implementation is available for SecPAL, so we could build a demonstrator for our use case, described in Section 5.

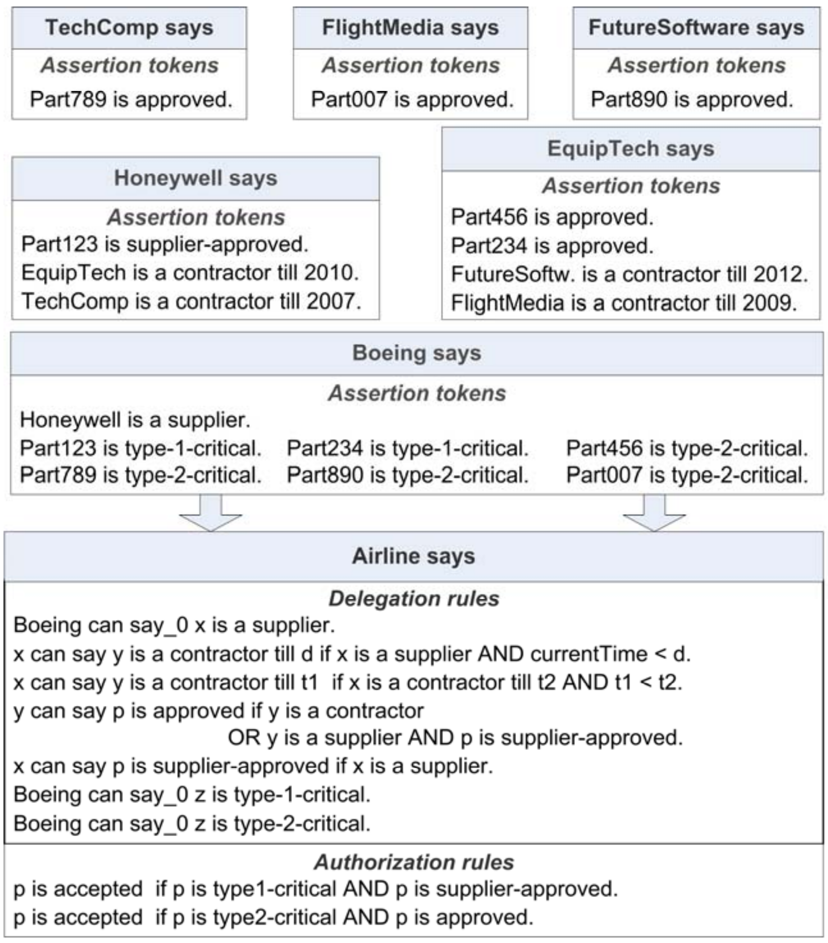
We should stress that SecPAL is not a formal language to reason explicitly about types of trust relationships and their quality or strength. Instead, SecPAL assertions are used to declare on which specific topics or statements somebody is trusted, thereby restricting the extent of trust: Parties are not assumed to trust each other fully, but only for the topics that are explicitly specified. As topics can be freely specified in assertions, SecPAL can be used to express fine-grained trust and access rights. SecPAL can however not be used to reason about the resulting trust relationships in a formal way at the metalevel as would be useful when designing a new system with incorporated trust management.

4 Applying SecPAL to the Case Study

4.1 Building and Managing Trust between Airlines and Suppliers

We represent the scenario of Section 2.1 in terms of SecPAL policies. In that scenario, the airline has to make decisions on whether to accept an incoming part that originates from a supplier or the contractor of a supplier. As explained, the airline usually does not have a stable trust relationship with suppliers or contractors, and hence cannot verify supplier signatures directly. However, airlines do have a stable trust relationship with the manufacturer Boeing, and in particular airlines can verify Boeing signatures. So SecPAL assertion tokens issued by Boeing can be used to establish trust between airlines and suppliers or contractors.

In Fig. 3 we present the SecPAL assertions of the different parties in that scenario, i.e. the airline, Boeing, the supplier Honeywell, and various subcontractors of Honeywell.



TechComp says
Assertion tokens
Part789 is approved.

FlightMedia says
Assertion tokens
Part007 is approved.

FutureSoftware says
Assertion tokens
Part890 is approved.

Honeywell says
Assertion tokens
Part123 is supplier-approved.
EquipTech is a contractor till 2010.
TechComp is a contractor till 2007.

EquipTech says
Assertion tokens
Part456 is approved.
Part234 is approved.
FutureSoftw. is a contractor till 2012.
FlightMedia is a contractor till 2009.

Boeing says
Assertion tokens
Honeywell is a supplier.
Part123 is type-1-critical. Part234 is type-1-critical. Part456 is type-2-critical.
Part789 is type-2-critical. Part890 is type-2-critical. Part007 is type-2-critical.

Airline says
Delegation rules
Boeing can say_0 x is a supplier.
x can say y is a contractor till d if x is a supplier AND currentTime < d.
x can say y is a contractor till t1 if x is a contractor till t2 AND t1 < t2.
y can say p is approved if y is a contractor
OR y is a supplier AND p is supplier-approved.
x can say p is supplier-approved if x is a supplier.
Boeing can say_0 z is type-1-critical.
Boeing can say_0 z is type-2-critical.
Authorization rules
p is accepted if p is type1-critical AND p is supplier-approved.
p is accepted if p is type2-critical AND p is approved.

Fig. 3 SecPAL policies for an airline accepting parts originating from suppliers

The airline specifies in its authorization rules the conditions to accept a part. We distinguish between critical software parts (type-1-critical) and less critical parts (type-2-critical). Critical parts have to be approved by a supplier, while type2-critical parts can also be approved by contractors of suppliers. In the first of its delegation rules, the airline states that Boeing is trusted to make claims about who is a supplier. The 'can say' construct is restricted by _0, meaning that the airline does not trust statements about supplier status that Boeing makes by using itself some form of delegation rules. The second rule, which states that suppliers are trusted to make statements about their contractors, does not have the _0 suffix. A contractor is allowed to have subcontractors, as long as the validity period of the contract is shorter then the validity of his own contract with the supplier. The next four delegation rules state that supplier and contractors are allowed to approve parts and that Boeing is the only party that is allowed to define the criticality type of a part. The authorization rules state that parts accepted only if they are approved by contractors or suppliers.

Only the airline specifies authorization rules, all other parties issue assertion tokens. Boeing issues several assertion tokens, thereby making claims about its suppliers, and classifying the criticality of parts. Honeywell, as a supplier, issues assertion tokens about the approval of parts, and about its contractors. The various contractors issue assertion tokens about the approval of parts, and about their contractors. Note the different validity times for the contractors. We list some example queries.

"SupplierPart": A software supplier delivers a part that is type1 critical for approval by the airline.

`Airline says Part123 is accepted.` (valid)

"ContractorPart": A contractor of a software supplier delivers a part that is type2 critical for approval by the airline.

`Airline says Part789 is accepted.` (valid)

"IllegalSubContractor": A contractor has a contract with another contractor, and the validity period of the subcontractor exceeds the validity period of the contractor's contract with the supplier. The subcontractor tries to deliver a part.

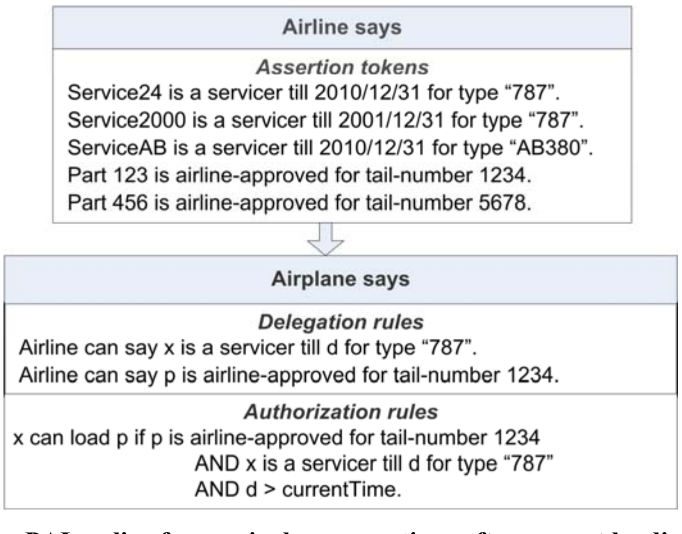
`Airline says Part890 is accepted.` (invalid)

"InvalidPartType": A contractor of a software supplier delivers a part which is type1 critical for approval by the airline.

`Airline says Part234 is accepted.` (invalid)

4.2 Authorizing Services to Perform Tasks in Airplanes

As a second example, we show how to implement the scenario from Section 2.2.



Airline says

Assertion tokens

Service24 is a servicer till 2010/12/31 for type "787".
Service2000 is a servicer till 2001/12/31 for type "787".
ServiceAB is a servicer till 2010/12/31 for type "AB380".
Part 123 is airline-approved for tail-number 1234.
Part 456 is airline-approved for tail-number 5678.

Airplane says

Delegation rules

Airline can say x is a servicer till d for type "787".
Airline can say p is airline-approved for tail-number 1234.

Authorization rules

```
x can load p if p is airline-approved for tail-number 1234
                AND x is a servicer till d for type "787"
                AND d > currentTime.
```

Fig. 4 SecPAL policy for an airplane accepting software part loading

Service providers install (load) parts from in an airplane. As service providers are contracted by airlines at various airports for a limited time, we use SecPAL assertions to authenticate service providers to airplanes.

As shown in Fig. 4, the authorization rules of the airplane state under which conditions a service providers is allowed to load a part into the airplane: the part has to be approved by the airline for the specific airplane and the person installing the part in the airplane has to be a service providers with a valid service provider contract with the airline. The airline is not mentioned in the authorization rules directly, but comes into play by the delegation rules or the airplane, where the airplane states that its airline is trusted to issue SecPAL assertion tokens about who is a service provider, and which parts are approved.

The airline issues a range of assertion tokens about current service providers and about approved parts. A service provider has to authenticate towards an airline in order to get an assertion token about his identity. How this is done is not addressed by the SecPAL policy – it could by done by personal exchange at some airline authority at an airport.

Note that most principal names in SecPAL assertions denote public keys and that all SecPAL assertions of the form 'A says …' are signed with the public key of A. So in the assertion token 'Airline says Service24 is a servicer till 2010/12/31 for type "787". Service24 is a public key of a current service provider, and when the airplane receives the token, it validates the signature to verify that its airline has issued the token. As the airplane and the airline have a stable long-term relationship, the airplane knows the public key of its airline and can perform the signature verification. After accepting the token, the airplane with tail number 1234 accepts the public key of Service24, and can use this key for authentication of the service provider, e.g. by using SSL. Again, we list example queries.

"ServicerValid": A service provider who is authorized by the airline for an airplane type wants to load a part which was approved by the airline and released for upload to the actual airplane.

Airline `says` *Service24* `can install` *Part123.* (valid)

"ServicerOutdated": A service provider whose contract with the airline has expired, wants to load a part which was approved by the airline and released for upload to the actual airplane.

Airline `says` *Service2000* `can install` *Part123.* (invalid)

"ServicerIncompetent": A service provider who has a valid contract with the airline, but for a different airplane type, wants to load a part which was approved by the airline and released for upload to the actual airplane.

Airline `says` *ServiceAB* `can install` *Part123.* (invalid)

5 Demonstrator

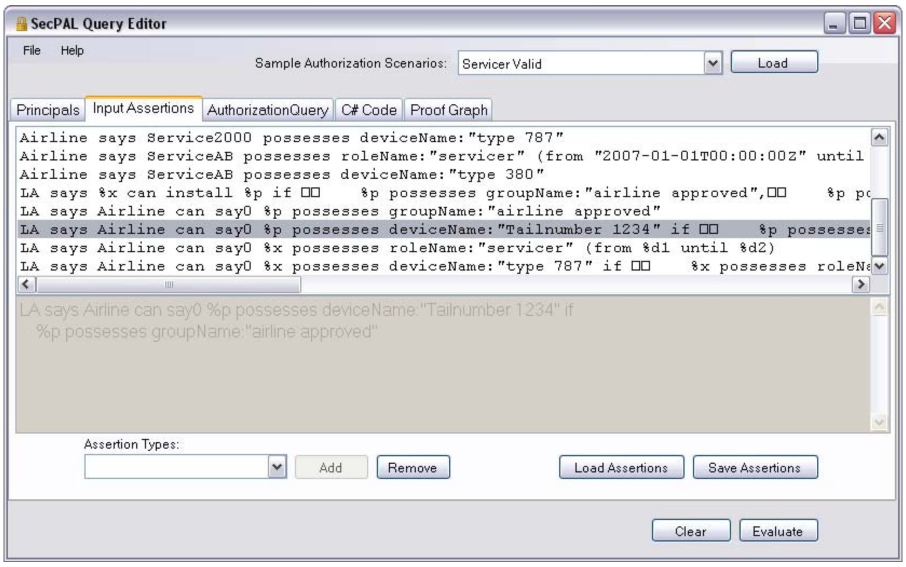
As part of the project, a demonstrator application has been created to show how the scenarios described in the previous section can be evaluated with SecPAL. An implementation is of course very helpful in developing a case study because it allows to check whether the statements have the intended meaning, and to experiment with different formulations.

Our demonstrator is based on the SecPAL Authorization Engine provided by Microsoft and available at [7]. A Windows forms application called "QueryEditor" is also offered by Microsoft. The QueryEditor offers the user basic English language constructs to automatically generate policies, assertions and queries. But as the automatic generation proved to be too restricted for our case study scenarios, we implemented the scenarios manually in C#, using the provided SecPAL class libraries.

The QueryEditor serves as user interface to start query evaluation. If a query is accepted, the corresponding proof tree can be examined. The Query editor can save assertions as XML files. The XML representation of policies and assertions will be the basis for real world implementations of SecPAL applications that can be integrated with distributed services.

The figures below are screenshots of the QueryEditor application. The “Principals” tab contains all principals that are part of the sample. Using the next two tabs, the policies and assertions of all principals and the queries can be displayed. Fig. 5 shows the assertions for the "ServicerValid" sample query. The underlying C# code, whether generated or written manually, can be seen at the “C# Code” tab .

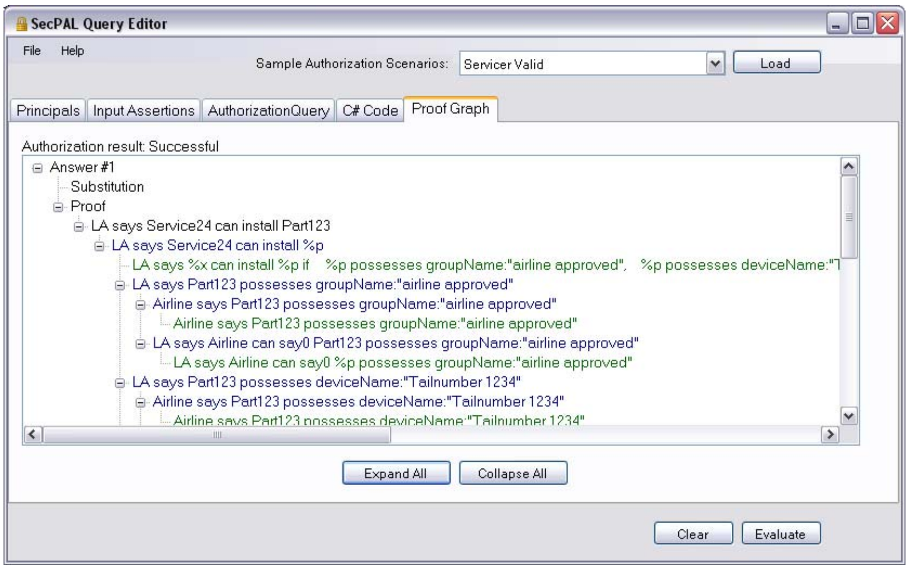
The button "Evaluate" starts the evaluation of the query. Using the "Proof Graph" tab, the proof of the evaluation can be displayed when successful (Fig. 6).



```
Airline says Service2000 possesses deviceName:"type 787"
Airline says ServiceAB possesses roleName:"servicer" (from "2007-01-01T00:00:00Z" until
Airline says ServiceAB possesses deviceName:"type 380"
LA says %x can install %p if ☐☐    %p possesses groupName:"airline approved",☐☐    %p pc
LA says Airline can say0 %p possesses groupName:"airline approved"
LA says Airline can say0 %p possesses deviceName:"Tailnumber 1234" if ☐☐    %p possesses
LA says Airline can say0 %x possesses roleName:"servicer" (from %d1 until %d2)
LA says Airline can say0 %x possesses deviceName:"type 787" if ☐☐    %x possesses roleNa
```

```
LA says Airline can say0 %p possesses deviceName:"Tailnumber 1234" if
   %p possesses groupName:"airline approved"
```

Fig. 5 QueryEditor: Assertions



Authorization result: Successful

```
⊟ Answer #1
    Substitution
  ⊟ Proof
    ⊟ LA says Service24 can install Part123
      ⊟ LA says Service24 can install %p
          LA says %x can install %p if    %p possesses groupName:"airline approved",    %p possesses deviceName:"T
        ⊟ LA says Part123 possesses groupName:"airline approved"
          ⊟ Airline says Part123 possesses groupName:"airline approved"
              Airline says Part123 possesses groupName:"airline approved"
          ⊟ LA says Airline can say0 Part123 possesses groupName:"airline approved"
              LA says Airline can say0 %p possesses groupName:"airline approved"
        ⊟ LA says Part123 possesses deviceName:"Tailnumber 1234"
          ⊟ Airline says Part123 possesses deviceName:"Tailnumber 1234"
              Airline says Part123 possesses deviceName:"Tailnumber 1234"
```

Fig. 6 QueryEditor: Proof Tree

Conclusion

Currently trust and authorization across domain boundaries is often established by setting up contractual agreements and inserting appropriate PKI certificates into local certificate stores. Doing so will become unmanageable when data exchanges between partners without long-term trust relationships increase. By handling a real-life use case, we have convinced ourselves that logic-based policy languages for authorization and trust, in particular SecPAL, are well suited to address this challenge. We found that we could handle our case study in a straightforward way, and that the resulting policies are easy to grasp for non-experts. Microsoft's implementation of SecPAL served as a practical proof of concept. However, for a wider applicability of SecPAL or its successor languages, it would be important to have a standardization of the language in the form of a syntax coding, say, in XML, a binding to transport protocols, and a processing model. We will continue to apply SecPAL or its successor languages whenever appropriate in our future work. For instance, we are investigating the use of DKAL for the application scenarios of AVANTSSAR [19].

References

[1] The WS-*, SAML and XACML standards are available at http://www.oasis-open.org/specs/
[2] Microsoft Corporation: Introducing Windows CardSpace. Available at http://msdn.microsoft.com/en-us/library/aa480189.aspx

[3] Soghoian, C., Stamm, S.: Certified Lies: Detecting and Defeating Government Interception Attacks Against SSL. Under submission.

[4] Maidl, M., von Oheimb, D., Hartmann, P., Robinson, R.: Formal Security Analysis of Electronic Software Distribution Systems. In: SAFECOMP 2008. LNCS, vol. 5219, pp. 415—428. Springer, Heidelberg (2008)

[5] Nigriny, J., Phaltankar, K.: Identity Assurance in Commercial Aviation Facilitated Through a Trusted Third Party Hub. White paper of CertiPath, available online at http://www.certipath.com/white-papers.htm

[6] Becker, M., Fournet, C., Gordon, A.: SecPAL: Design and Semantics of a Decentralized Authorization Language. In: 20th IEEE Computer Security Foundations Symposium, pp. 3—15. IEEE Press, New York (2007)

[7] SecPAL homepage. http://research.microsoft.com/en-us/projects/SecPAL/

[8] Li, N., Mitchell, J.C.: Datalog with Constraints: A Foundation for Trust Management Languages. In: PADL 2003, LNCS, vol.2562, pp. 58—73. Springer, Heidelberg (2003)

[9] IBM Corporation and Microsoft Corporation: Security in a Web Services World: A Proposed Architecture and Roadmap. Available online at http://www.ibm.com/developerworks/library/specification/ws-secmap/ (2002)

[10] Abadi, M., Burrows, M., Lampson, B., Plotkin, G.: A Calculus for Access Control in Distributed Systems. ACM Transactions on Programming Languages and Systems, 15:4, pp. 706—734 (1993)

[11] DeTreville, J.: Binder, a logic-based security language. In: IEEE Symposium on Security and Privacy, pp.105—113. IEEE Press, New York (2002)

[12] Li, N., Grosof, B., Feigenbaum, J.: Delegation Logic. In: ACM Trans. on Information and System Security (TISSEC) 6:1, pp. 128—171 (2003)

[13] Becker, M., Sewell, P.: Cassandra: Flexible trust management, Applied to Electronic Health Records. In: 17th IEEE Computer Security Foundations Workshop (CSFW), IEEE Press, New York (2004)

[14] Halpern, J.Y., Weissmann, V. A formal foundation of XrML. In: 17th IEEE Computer Security Foundations Workshop (CSFW), IEEE Press, New York (2004)

[15] Li, N, Mitchell, J. C.: Understanding SPKI/SDSI using first-order logic. In: 16th IEEE Computer Security Foundations Workshop (CSFW), pp. 89—103. IEEE Press, New York (2003)

[16] Jha, S., Schwoon, S., Wang, H., Reps, T.: Weighted Pushdown Systems and Trust-Management Systems. In: TACAS 2006. LNCS, vol. 3920, pp. 1—26. Springer, Heidelberg (2006).

[17] Gurevich, Y., Neeman, I.: DKAL: Distributed-Knowledge Authorization Language. In: 21th IEEE Computer Security Foundations Workshop (CSFW), pp. 149—162. IEEE Press, New York (2008)

[18] Gurevich, Y., Neeman, I.: A Simplified and Improved Authorization Language. Microsoft Research Tech Report, February 2009, available online at http://research.microsoft.com/en-us/um/people/gurevich/dkal.htm

[19] EU-funded project AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures. http://avantssar.eu/