A Capability-Based Access Control Architecture for Multi-Domain Publish/Subscribe Systems

Lauri I.W. Pesonen, David M. Eyers, and Jean Bacon
University of Cambridge, Computer Laboratory
JJ Thomson Avenue, Cambridge, CB3 0FD, UK
{firstname.lastname}@cl.cam.ac.uk

Abstract

Publish/subscribe has emerged as an attractive communication paradigm for building Internet-wide distributed systems by decoupling message senders from receivers. So far most of the research on publish/subscribe has focused on efficient event routing, event filtering, and composite event detection. Very little research has been published regarding securing publish/subscribe systems. In this paper we present a capability-based access control architecture that enables multiple domains to co-operate in order to build a shared, wide-scale publish/subscribe system. Our architecture employs SPKI authorisation certificates for delegating access control responsibilities to access control services within independent domains in order to balance security and scalability. The architecture supports controlling access both for new event brokers joining the broker network as well as for clients accessing the publish/subscribe API.

1. Introduction

Publish/subscribe has emerged as a popular communication paradigm for Internet-wide, large-scale distributed systems. This is because it decouples message senders and receivers from each other, routing messages based on their topic or content instead. Modern large-scale publish/subscribe systems are based on a peer-to-peer network of event brokers that routes events from publishers to subscribers in an efficient manner. A peer-to-peer network re-balances itself dynamically in the case of node and network link failures, and when nodes join and leave the peer-to-peer network. Broker networks based on peer-to-peer overlays are thus highly fault-tolerant and scalable.

Most of the research in publish/subscribe systems has concentrated on efficient event routing, event filtering, and composite event detection. Very little work has been done on the security aspects of publish/subscribe systems, yet scalable access control will be a prerequisite for Internet-wide publish/subscribe due to ever-present malicious users (and possibly also for legal reasons within organisations). When the scale of a system increases, so do its access control requirements. Small-scale systems can be deployed in company intranets without any access control, but that is not feasible in a public or large-scale setting.

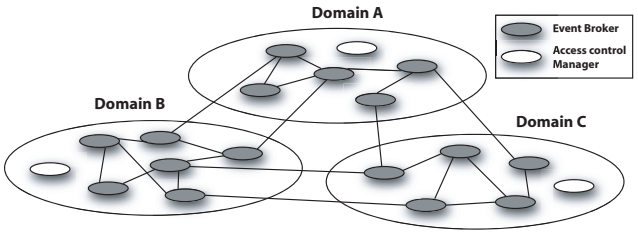

Figure 1. Three domains co-operate in order to form a shared broker network.

We envision a large-scale, multi domain publish/subscribe system where multiple domains co-operate to form a shared broker network, as seen in Fig. 1. Brokers may be inherently mutually trusting, as in the sub-domains of a national police force (See Sect. 4), or mutually suspicious as when police domains interwork with different public service domains.

We propose an access control architecture that scales well with the number of domains, clients, and event types, while at the same time provides the benefits of decentralised trust management, e.g. delegation of responsibilities. We present a number of uses for SPKI authorisation certificates in implementing a full-featured access control architecture for a multi-domain publish/subscribe system.

The rest of the paper is organised as follows. Sect. 2 provides background knowledge required to understand our architecture, which itself is introduced in Sect. 3. Sect. 4 discusses an example application. We present related work in

Sect. 5, and finally, in Sect. 6 make our conclusions and indicate areas of future research.

2. Background

This section provides a brief introduction to publish/subscribe systems, capability-based access control, and decentralised trust management.

2.1. Publish/Subscribe Message Delivery

Publish/subscribe communication facilitates many-to-many communication between parties based on the event content or topic, rather than the explicit source and destination addresses.

The simplest form of publish/subscribe system, called topic-based publish/subscribe [9], classifies events based on their topic. Subscribers subscribe to specific topics and receive all events published to that topic. Content-based publish/subscribe [9] allows filtering based on the content of an event, i.e. a subscriber defines a filter and only matching events are delivered. Topic-based and content-based approaches can be combined so that a subscription is topic-specific, but it includes a filter expression for filtering events of that topic.

2.1.1. Typed, Content-based Publish/Subscribe We have implemented our access control architecture on a Hermes-like [13, 14] publish/subscribe middleware called Maia. Maia implements Hermes, but also includes security features that were not part of the original design.

Hermes is a content-based publish/subscribe middleware with strong event typing. It is built on a peer-to-peer routing substrate to provide scalable event dissemination and fault tolerance in case of node or network failures.

A Hermes system consists of *event brokers* and *event clients*, the latter being *publishers* and/or *subscribers*. Event brokers form an application-level overlay network that performs event propagation by means of a content-based routing algorithm. Event clients publish and/or subscribe to events in the system. An event client connects to a *local broker*, which then becomes *publisher-hosting*, *subscriber-hosting*, or both. An event broker without connected clients is called an *intermediate broker*.

Hermes supports *event typing*: every published event (or *publication*) in Hermes is an instance of an *event type*. An event type defines a *type name* and a set of *attributes* that consist of an *attribute name* and an *attribute type*. Supported attribute types typically depend on the types supported by the language used to express subscription filters. For example, XPath 2.0 [18], which can be used to specify filters for XML documents, defines primitive types like string, boolean, decimal, float, double, duration, and date etc.

Another Hermes-specific feature is support for *event type hierarchies* – an event type inherits all of the attributes defined and inherited by its ancestor. In addition to making defining new types easier, subscriptions to super-types in an event hierarchy match notifications of events of both the subscribed type and all its subtypes.

2.1.2. Secure Names This work relies on the fact that access rights can be bound to a unique and unforgeable name. We introduced *secure event types* in [12] that provide secure names both for event types as well as attributes within those types. The type definition is digitally signed with the type owner's private-key. In secure event types the type owner's public-key is included in the name of the event type. This creates a unique name that is bound to the type definition, because the signature in the type definition is verifiable with the public-key from the type's name. The digital signature guarantees that type and attribute names cannot be forged nor tampered with.

The same approach, adding the owner's public-key in the name, can be utilised also with topic-based publish/subscribe, in which case the topic name would include the topic owner's public-key. Because the capability granting the client access to the topic must be linked to the topic owner's public-key, as will be explained in Sect. 2.3, the topic name can be seen as secure.

2.2. Capabilities

Access rights in a system can be described with an *access control matrix* where the rows represent *subjects* (i.e. users), the columns represent *objects* (i.e. resources), and the cells define the *access rights* that a specific subject has over a specific object.

Access control systems typically implement either a column centric view or a row centric view of the matrix. In a column centric view each column of the matrix is translated to an *access control list* (ACL) that is stored with the object that the column represents. The ACL contains entries for each subject defining the access rights for that subject regarding the specific object.

In a row centric view each row of the matrix is translated to a set of capabilities that are stored with the subject. Each capability defines what access rights the subject, or holder of the capability, has over a given object.

A common use for certificates is to map principals to trusted identities. Rather than using certificates as an authentication token, however, it is also possible to use certificates for authorisation of actions. Certificates used in this manner are often referred to as signed capabilities.

Maia will use the access control policy engine of the *Open Architecture for Secure Interworking Services* (OASIS) [2, 3], which is an established, distributed, Role-Based Access Control (RBAC) [16] system. In OASIS, principals

authenticate into roles and activate capabilities according to role activation and authorisation policies, respectively. Role membership and capabilities are both represented by digital certificates that are used in access requests.

2.3. Decentralised Trust Management

Decentralised trust management was first introduced by Blaze et al. in [6]. *PolicyMaker* was later followed by *KeyNote* [5], and the *Simple Distributed Security Infrastructure* (SDSI) [15], which was later integrated with the *Simple Public-Key Infrastructure* (SPKI) [8] to create SDSI 2.0 [7].

The central idea in decentralised trust management is to decentralise access control decision making, and policy and credential management. This is achieved by requiring the *owner* of an object, or their delegatee, create the capabilities which can be used to access it. Distributing management responsibilities over all of the principals results in an extremely scalable access control system.

In SPKI the decentralisation is based on *certificate loops*. A typical certificate loop is depicted in Fig. 2 where the owner, P_a of an object O, grants P_b an SPKI authorisation certificate, C_{ab}, with access rights, A_{ab}, for the object O. P_b then further delegates access rights A_{bc}, where $A_{bc} \subseteq A_{ab}$, to P_c by granting P_c another delegation certificate C_{bc}. Now, when P_c wants to access O, she shows P_a both certificates C_{ab} and C_{bc}. P_a is now able to form a certificate chain from P_c to P_b via C_{bc} and from P_b to itself via C_{ab}. Finally P_c authenticates herself to P_a by proving ownership of the key-pair P_c. P_c does this by executing a public-key challenge-response protocol with P_a. This completes the certificate chain which now forms a *certificate loop* flowing from P_a to P_b to P_c and back to P_a again. P_a has now verified that P_c is authorised to access O within the privileges granted by $A_{ab} \cap A_{bc}$[1]. Typically the verification is performed by an access control service rather than P_a. Note that in SPKI, principals and key-pairs are synonymous.

An SPKI authorisation certificate is a 5-tuple containing the following items: *Issuer*, *Subject*, *Delegation*, *Authorisation*, and *Validity*. *Issuer* is the public-key (or a hash of the public-key) of the issuer of the certificate. *Subject* is the public-key of the entity the certificate is issued to. *Delegation* is a boolean value specifying whether the *Subject* is permitted to further propagate the *Authorisation* granted by this certificate. In a certificate chain all certificates bar the last one must have *Delegation* set to $true$, otherwise the certificate chain is not valid. *Authorisation* is an application specific representation of access privileges granted to the *Subject* by the *Issuer*. And finally, *Validity* defines the date range when the certificate is valid and optional on-line validity tests, e.g. *certificate revocation lists* (CRLs).

1 The verification process will also consider optional validity conditions (e.g. expiration dates) for each certificate in the chain. Any invalid certificate will render the whole chain invalid.

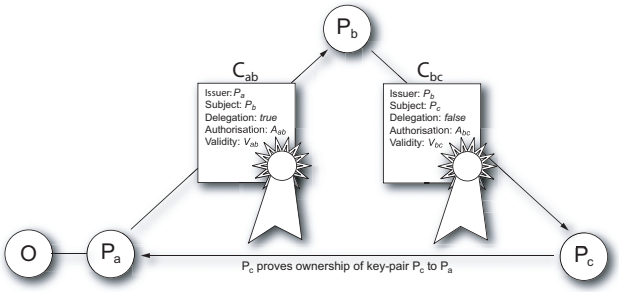

Figure 2. An SPKI authorisation certificate loop with three principals and two level delegation.

Two certificates are reduced to a single 5-tuple as follows:

$$< I_1, S_1, D_1, A_1, V_1 > + < I_2, S_2, D_2, A_2, V_2 > \Rightarrow < I_1, S_2, D_2, A_1 \cap A_2, V_1 \cap V_2 >$$
$$\text{iff } A_1 \cap A_2 \neq \emptyset, V_1 \cap V_2 \neq \emptyset, S_1 = I_2 \text{ and } D_1 = true$$

This 5-tuple reduction rule is applied recursively to the certificate loop in order to collapse the loop to a single 5-tuple that will then be evaluated.

Our work relies on SPKI authorisation certificates to propagate authorisation from resource owners to domains in a decentralised and scalable fashion. We refer to the authorisation certificates as delegation certificates in order to emphasise the fact that they are used to delegate access control duties by one party to another.

3. Access Control in Publish/Subscribe

We envision a multi-domain publish/subscribe system, as explained in Sect. 1 and Fig. 1, where each domain contains a number of event clients and brokers, and an *access control manager*. The access control manager is responsible for granting privileges to brokers and clients in that domain according to the domain's access control policy.

One of the domains in the system is the *coordinating domain* which coordinates the forming of the shared publish/subscribe system. The coordinating domain invites other domains to join the shared publish/subscribe system. In a sense the coordinating domain owns the shared publish/subscribe system and is responsible for managing it.

The incentive for domains to join the network is twofold: on the one hand domains are interested in implementing shared applications with other domains, e.g. when one domain produces events while others consume them. On the

other hand, even with domain-internal applications, the increased number of brokers increases the reliability and coverage of the broker network, i.e. a broker network with more nodes is increasingly fault-tolerant and the increased coverage enables the domain to reach a larger geographic area with lower infrastructure investments. Both incentives are attractive to domains, but only if the system provides proper access control to prevent unauthorised access.

The publish/subscribe system needs to control access to a number of resources. All of the following actions have to be authorised: (i) nodes, i.e. event brokers and event clients, connecting to the broker network; (ii) type and topic owners (in content-based and topic-based publish/subscribe, respectively) introducing new types and topics to the system; (iii) type and topic owners extending an existing type/topic; (iv) event clients accessing the publish/subscribe API, i.e. to publish or subscribe to events.

We propose a common approach to access control for publish/subscribe systems where access control decisions in all four cases are ultimately rooted at the appropriate resource owner. In (i) and (ii) the resource owner is the coordinating domain. In (iii) and (iv) the resource owner is the type or topic owner. Employing delegation certificates in the architecture and distributing the access control policy management, decision making, and credential management over all resource owners enables the access control architecture to scale well in a multi-domain environment.

3.1. Delegating Authority

As explained in Sect. 2.3, a resource owner delegates authority to another subject by issuing a delegation certificate for that subject. In our model the resource owner grants a delegation certificate to a domain's access control manager. The delegation certificate authorises the access control manager to further delegate the granted authority to clients (e.g. event brokers and event clients) in that domain. The access control manager is then responsible for further delegating the authority to its clients by issuing capabilities to them according to the domain-internal access control policy.

The delegation certificates and capabilities form a certificate loop that links the client to the resource owner through the access control manager, as seen in Fig. 3.

When a client requests an action, it will show the capability it received from the access control manager and the delegation certificate(s) linking the access control manager to the resource owner. The capability and the delegation certificate chain are linked to each other by the access control manager's public-key which is the subject in the last delegation certificate in the certificate chain and the issuer in the client's capability. The verifier then verifies the capability and the delegation certificates, verifies the certificate chain that they form, and executes a public-key challenge-response protocol with the requesting client, as depicted in Fig. 2. If everything validates correctly, the client's request is processed.

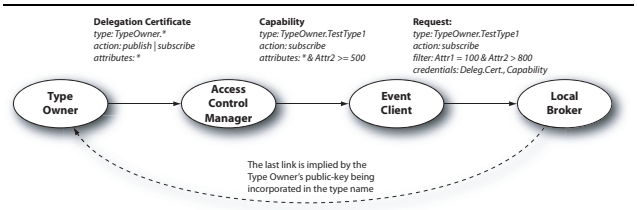

Figure 3. The type owner delegates authority to the access control service to issue capabilities for accessing the type owner's event types.

Note that the certificate chain must end at a resource owner that the verifier trusts. That is, the prover and the verifier must have a common trust root. In the coordinating domain's case, all nodes participating in the shared publish/subscribe system trust the coordinating domain (the public-key of the coordinating domain is installed in all nodes as a trusted public-key or alternatively the access control manager gives the public-key to the client with the capability). In the case of the type and topic owners, the owner's public-key is included in the name of the type/topic. This links together the root of the certificate chain and the type/topic name, and enables the verifier to trust the certificate chain.

We assume that delegation certificates will be delivered to the domain access control managers out-of-band. Out-of-band delivery enables the resource owners to change access control policy after deployment by issuing new and revoking existing delegation certificates. For example, the coordinating domain is able to accept a new domain to the common broker network by issuing a delegation certificate to the joining domain's access control manager. Similarly, a type owner is able to grant a new domain access to an existing event type by issuing the new domain a delegation certificate granting access to that event type.

The authorities described below can be combined into a single certificate assuming that the issuer is authorised to grant that authority. For example, a coordinating domain could issue a single certificate granting both `connect` and `install` rights to an access control manager (e.g. `action: connect | install`). The field values of the authority support also wildcards, e.g. `action: *` in order to enable all actions. It is important to notice that the resource owner can only grant authority for her own resources. That is, a type owner can grant the authority `type: *`, but that will only grant access to that type owner's types. Similarly with coordinating domains the au-

thority `network: *` will grant access only to networks coordinated by that domain.

3.1.1. Broker Network Access. The lowest level access control decision in the system is granting nodes access to the broker network. The authority is rooted at the coordinating domain which is seen as the resource owner in the case of the shared publish/subscribe system.

The authority granted is very coarse compared to the publish/subscribe API related authorities: it can only grant or not grant access to a specific network. The authority specifies the name of the network and the authorised action:

```
network: TestNetwork1
action: connect
```

When connecting to a broker, both parties should verify each others credentials. This is to prevent a client from connecting to a malicious broker that pretends to be part of the system, but in reality is not.

3.1.2. Introducing New Types/Topics. The authority to install new types or topics is also controlled by the coordinating domain since this is seen as a service related to the publish/subscribe system. For example:

```
network: TestNetwork1
action: install
```

3.1.3. Extending Types/Topics. Extending an existing type or topic is related to the type/topic being inherited. Thus the owner of that type/topic should be the one delegating the authority to extend that type/topic.

By using wildcards the type/topic owner is able to include multiple types/topics into one certificate. In the example below, authority is granted to extend all types with names starting with `Test` owned by the this type owner:

```
type: Test*
action: extend
```

In type-checked publish/subscribe the delegation certificate granting authority to extend a type must be included in the inheriting type definition. This way the type definition is a self-contained package and its validity can be verified without external assistance [12].

3.1.4. Accessing the Publish/Subscribe API. Accessing the publish/subscribe API is always related to a specific type/topic. Therefore the type/topic owner is seen as the resource owner and she is responsible for issuing delegation certificates to access control managers.

The authority for accessing the publish/subscribe API can be very fine grained. On the one hand the authority can be extremely generous granting the client access to all types and all attributes in those types both for publishing and subscribing. On the other hand the authority can be very specific and specify a single type that the authority applies to, whether it grants only publishing or subscription rights, and which attributes are accessible to the client:

```
type: TestType1
action: subscribe | publish
attributes: Attr1 & Attr2 >= 500
```

Notice that the authority can place restrictions on event content. That is, in the above example the value of `Attr2` is restricted to be greater or equal than 500. When publishing this means that the local broker drops the publication if the publisher tries to publish a value less than `500`. When subscribing the local broker of the subscriber adds a filter for `Attr2 >= 500` to the subscription.

3.2. Access Control Policy

The access control policy for a resource is distributed between the resource owner and the authorised access control managers. The resource owner authorises access control managers to delegate authority to domain members. The access control service of a domain implements an independent, domain-internal access control policy which defines the access rights granted to domain members. This means that the resource owner has no control over delegation certificates issued to domain members by the domain's access control manager. We assume that the resource owner is willing to trust the access control manager of a domain within the extent of the delegation certificate. This seems reasonable assuming that the resource owner is able to easily revoke access from misbehaving access control managers.

3.3. Credential Propagation

A publish/subscribe client presents its credentials to the local broker when making a publish/subscribe API call. The local broker verifies that the provided credentials authorise the client to make the request and routes the request through the broker network [13]. For the intermediate brokers to be able to verify the request during transit, the message must include the credentials authorising the client to make the request. In addition to the credentials, the request must also include a timestamp and it must be signed by the client. The timestamp allows brokers to reject old, replayed requests. The signature binds the request to the client's public-key which is again bound to the provided credentials. Together the credentials, the timestamp, and the signature allow any broker in the network to verify that an authorised client has made the given request.

In principle we assume that brokers trust each other, but propagating the credentials with requests enable intermediate brokers to verify the legitimacy of a request if they so desire. The brokers are free to implement various verification strategies. A paranoid broker might verify every request. A

more trusting broker might implement a probabilistic approach where each request is verified with a configurable probability.

The cost of verifying a request in each node it passes through in a network can be estimated. A DHT-based peer-to-peer network routes messages within $\log(n)$ hops to a known destination, where n is the number of nodes in the network. A subscription request is first routed to the rendezvous node and from there towards all known publishers following the reverse path of advertisement messages. Thus, each publisher publishing the same type/topic adds another $\log(n)$ hops to the aggregate hop count of the subscription. Each broker on the route from the subscriber to each of the publishers verifies the digital signature of the publish/subscribe request and one digital signature for each certificate in the certificate chain leading from the subscriber to the type/topic owner. Thus, verifying the validity of a subscription in all intermediate brokers results in up to $(c+1)(p+1)\log(n)$ extra certificate computations compared to not verifying the validity of the request (where p is the number of publishers and c is the number of certificates in the certificate chain). We ignore the other computations required to verify a certificate chain, e.g. set operations over the authority field, because they are most likely very fast compared to digital signature verification.

In case of advertisements the incurred cost would be $(c+1)\log(n)$, because the advertisement is routed from a publisher only to the rendezvous node.

For publications the incurred cost would be $(c+1)(s+1)\log(n)$, where s is the number of subscribers. The required effort is reduced if subscriptions include filters that filter out some of publications in the intermediate brokers.

3.4. Access Rights Revocation

Rapid, reliable, distributed revocation of certificates is a non-trivial problem, yet one for which we need a mechanism if rights issued in an access control system such as ours might ever need to be revised.

Traditional approaches to certificate revocation include expiry dates, certificate revocation lists (CRLs), and various on-line tests. When a given revocation occurs through any of these mechanisms, the OASIS policy rules for which that credential was a prerequisite for can be scanned to effect *active security* – we can send events to the other parties that need to be notified of this revocation. That is, if a subject loses her role membership or capability, all registered parties will be notified. This allows, for example, event brokers to be notified if a client loses a capability that authorises her to subscribe to an event type.

The proper approach to certificate revocation is very much application dependent. In some cases very short expiry dates will suffice. Other applications will require more complex approaches.

4. Example Application

The architecture we have presented is motivated by problems facing organisations with which we do collaborative research, such as the Police Information Technology Organisation (PITO) in the UK. In this particular case, we consider the British Police Force (BPF) – a federation of more than fifty largely autonomous regional forces. Many of PITO's projects aim to increase the efficiency of communications between these independent police forces. This is a challenging task, given the diversity of software deployed, and the different ontologies used within the separate forces.

For the sake of administration and infrastructure costs, added fault-tolerance, and more efficient message delivery, a shared publish/subscribe broker network across all the police forces is very appealing.

In addition to decreasing costs, the shared infrastructure enables the separate forces to implement shared applications that make police work more efficient. For example a detective working on a robbery in the jurisdiction of police force P_1 can subscribe nationwide to events related to a car seen at the crime scene. Then a report, after the initial robbery investigation began, of a stolen vehicle in another police force, P_2, results in an event being delivered to the detective. Instead of repeatedly checking for changes in the vehicle registration databases of all the separate police forces, the detective is notified of a change asynchronously.

Although the regional forces are all part of the BPF and thus trusted, there still needs to be access control in place to provide confidentiality and to guarantee message integrity. For example, investigations include witness statements where the identity of the witness must remain confidential in order to protect the witness. Thus an infrastructure shared among multiple seemingly mutually trusting domains must implement an access control system such as ours. In addition to protecting data confidentiality and integrity, access control is also necessary in order to keep applications domain-internal. In the case of the BPF, the regional forces have their own proprietary applications that should not be accessible to all members of the Force.

5. Related Work

Related work on securing publish/subscribe systems includes [10], where Miklós presents an access control mechanism for large-scale publish/subscribe systems that supports access control decisions based on publication and subscription filter content. Similarly Belokosztolszki et al. present a role-based access control mechanism for publish/subscribe systems in [4]. In both cases the lack of trusted

names renders the scheme infeasible in a multi-domain setting, whereas our approach in binding access rights to secure names avoids this problem. This paper, alongside [12], provides the security mechanisms underpinning [1].

Wang et al. present in [17] a number of security issues in Internet-scale publish/subscribe systems that need to be addressed in the future. The paper covers problems related to authentication, data integrity and confidentiality, accountability, and service availability. We feel that access control provides a foundation for solving these problems. Our approach provides access control on all tiers of the publish/subscribe system which goes a long way in answering many of the questions they have posed.

Opyrchal and Prakash concentrate on the separate problem of providing confidentiality for events during the last hop from the local broker to the subscribers with as few encryptions as possible [11]. Where we see event encryption as an effective way to enforce access control in an untrusted broker network, we assume that local brokers have enough resources to maintain secured network connections to clients in which case the efficient encryption of single events for the last hop is not relevant.

6. Conclusions and Future Work

We have presented an SPKI-based access control architecture for multi-domain publish/subscribe systems. By applying decentralised trust management, we are able to conveniently administer and enforce access control within publish/subscribe systems that span multiple domains.

There are aspects of future work resulting from this paper. So far we have focused on allowing clients to verify that they are communicating using consistent event types/topics through checking the certificate chains that authorise their use of them. We are keen to move towards encrypting events, or attributes of them, so that we can enforce access control in an untrusted broker network, and thus evolve away from a boundary-oriented access control approach.

7. Acknowledgements

Lauri Pesonen is supported by EPSRC (GR/T28164/01).

References

[1] J. Bacon, D. Eyers, K. Moody, and L. Pesonen. Securing publish/subscribe for multi-domain systems. In *Middleware 2005*, November 2005. Forthcoming.

[2] J. Bacon, K. Moody, and W. Yao. Access control and trust in the use of widely distributed services. In *Middleware 2001*, volume LNCS 2218, pages 300–315. Springer-Verlag, Nov. 2001.

[3] J. Bacon, K. Moody, and W. Yao. A Model of OASIS Role-Based Access Control and its Support for Active Security. *ACM Transactions on Information and System Security (TISSEC)*, 5(4):492–540, Nov. 2002.

[4] A. Belokosztolszki, D. M. Eyers, P. R. Pietzuch, J. Bacon, and K. Moody. Role-based access control for publish/subscribe middleware architectures. In *Proc. of the 2nd International Workshop on Distributed Event-Based Systems (DEBS'03)*, ACM SIGMOD, San Diego, CA, USA, June 2003. ACM.

[5] M. Blaze, J. Feigenbaum, and A. D. Keromytis. KeyNote: Trust management for public-key infrastructures (position paper). In *Proc. of the Cambridge 1998 Security Protocols International Workshop*, volume 1550, pages 59–63, 1998.

[6] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proc. of the IEEE Conference on Security and Privacy*, Oakland, CA, USA, May 1996. IEEE.

[7] CIS. SDSI (a simple distributed security infrastructure). `http://theory.lcs.mit.edu/~cis/sdsi.html`, Sept. 2001.

[8] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylönen. SPKI certificate theory. RFC 2693, Internet Engineering Task Force, Sept. 1999.

[9] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, 2003.

[10] Z. Miklós. Towards an access control mechanism for wide-area publish/subscribe systems. In *Proc. of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02)*, Vienna, Austria, July 2002. IEEE.

[11] L. Opyrchal and A. Prakash. Secure distribution of events in content-based publish subscribe systems. In *Proc. of the 10th USENIX Security Symposium*. USENIX, Aug. 2001.

[12] L. I. Pesonen and J. Bacon. Secure event types in content-based, multi-domain publish/subscribe systems, Sept. 2005. Forthcoming.

[13] P. R. Pietzuch and J. Bacon. Peer-to-peer overlay broker networks in an event-based middleware. In H. A. Jacobsen, editor, *Proc. of the 2nd International Workshop on Distributed Event-Based Systems (DEBS'03)*, ACM SIGMOD, San Diego, CA, USA, June 2003. ACM.

[14] P. R. Pietzuch and J. M. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *Proc. of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02)*, pages 611–618, Vienna, Austria, July 2002. IEEE.

[15] R. L. Rivest and B. Lampson. SDSI – A simple distributed security infrastructure. Presented at CRYPTO'96 Rumpsession, Oct. 1996.

[16] R. Sandhu, E. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[17] C. Wang, A. Carzaniga, D. Evans, and A. L. Wolf. Security issues and requirements in internet-scale publish-subscribe systems. In *Proc. of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, Big Island, HI, USA, 2002. IEEE.

[18] World Wide Web Consortium. *XML Path Language (XPath) Version 2.0*, Apr. 2005.