Security Aspects in Publish/Subscribe Systems

L. Fiege A. Zeidler A. Buchmann
TU Darmstadt
{fiege,az,buchmann}@dvs1.informatik.tu-darmstadt.de

R. Kilian-Kehr
SAP Corporate Research
roger.kilian-kehr@sap.com

G. Mühl
TU Berlin
gmuehl@acm.org

Abstract

Publish/subscribe is emerging as a very flexible communication paradigm that is applicable to environments demanding scalable and evolvable architectures. Although considered for workflow, electronic commerce, mobile systems, and others, security issues have long been neglected in publish/subscribe systems. Recent advances address this issue, but only on a low, technical level. In this paper, we analyze the trust relationships between producers, consumers, and the notification infrastructure. We devise groups of trust to model and implement security constraints both on the application and the system level. The concept of scopes helps to localize and implement security policies as an aspect of structured publish/subscribe systems.

1 Introduction

The publish/subscribe paradigm is an interest-oriented communication model [3]. Event notifications are published by producers, and consumers receive those that match one of the subscriptions they have specified. The paradigm is successfully applied in many areas of distributed computing, and the loose coupling of producers and consumers leverages reconfigurability and evolution. Recent research mainly focused on functional aspects of the intermediary pub/sub service that conveys the notifications. It is considered a black box optimized for notification routing and scalability in distributed settings. Today, an increasingly important emerging aspect of publish/subscribe systems is *security* and *trust*. This includes *access control* to the pub/sub infrastructure (and the data it transports), as well as the need to establish mutual *trust* between producers and consumers of data, i.e., granting the *authenticity* and *validity* of data in the system.

This imposes the question of *how* the mutual trust between publisher and consumer can be established despite the decoupling facilitated by the pub/sub paradigm. The obvious approach is to delegate some of the aspects of trustworthy interaction to the pub/sub service for enforcement. For instance, access control and secured delivery can be added to the pub/sub infrastructure [1]. Unfortunately, this often implies that the infrastructure as a whole is trusted, a frequently found assumption.

At Internet scale, however, the pub/sub infrastructure itself has to be considered as a security issue. A distributed network of event brokers likely spans a larger number of service providers and many administrative domains. Consequently, the security considerations of producer-consumer interaction must include the infrastructure, and a black box view on it is no longer applicable.

Initial work is available on security issues in publish/subscribe. A general description of requirements is given by Wang et al. [12]. An apparent problem is access control to the pub/sub service and certain classes of notifications [8, 1]. Miklós [8] uses the Siena covering relations to constrain allowed subscriptions and advertisements; a trusted broker network is assumed. Perhaps the most advanced result is Belokosztolszki et al. [1], who combine role-based access control with a distributed notification service. The privilege to publish or subscribe to a specific *type* of event is granted by a designated owner of this type. A relaxation of the trusted network assumption is sketched that finds connected broker subgraphs that use encrypted communication links. However, globally valid type hierarchies are problematic to establish and limited in their modeling capabilities [7].

In this paper, we want to weaken the assumption of a trusted network to a large degree. The settings we consider are systems where the notification service (a) can be part of a larger network consisting of different transport networks of unknown trustworthiness; and (b) the notification delivery itself may span several separate notification services or administrative domains. Obviously, mechanisms must be in place to bridge potentially malicious networks or brokers, as well as to establish mutual trust between different administrative domains on behalf of a client. We discuss these issues in greater detail in Section 2. Then, in the remainder of this paper, we show our approach of applying *scopes* to the problems aforementioned. Scopes originally were designed to model *visibility* of event dissemination within the distributed pub/sub notification service REBECA (cf. Section 3). In Section 4 we exploit scopes to enforce and maintain security aspects in Internet-scale pub/sub systems. Section 5 sketches an implementation using aspect-oriented programming techniques, before Section 6 concludes this paper.

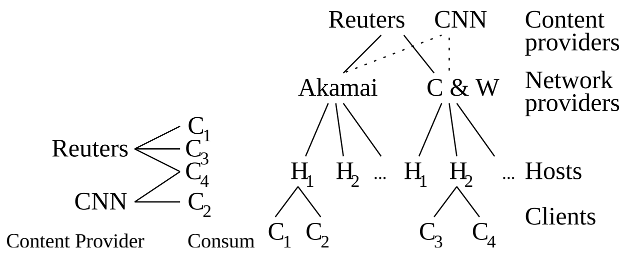
2 Trust in Pub/Sub Systems

2.1 System Model

A minimal pub/sub system consists of producers, consumers, and the intermediary pub/sub service to convey the published notifications. The pub/sub service offers three simple-to-use primitives: `subscribe`, `advertise`, and `publish` to register consumer interests, to announce potential future notifications, and to publish a notification. The notification service itself acts as a black box and is conceptually centralized, which we refine later. We assume a distributed implementation in a network of event brokers; the brokers to which clients are connected are called border brokers. Each broker maintains a routing table that keeps track of the network links and the subscriptions that were received on them. Notifications are forwarded on those links for which a matching subscription is stored.

2.2 Trust

Trust in the sense we use it in this paper has two different aspects: a "real-world" aspect of *trusting someone or something* on the basis of some contract (Fig. 1(a)) and, second, the aspect of implementing trust through some security measures in a more technical sense (Fig. 1(b)). In traditional systems, security is mostly based on knowing the identity of involved parties, which is not possible in publish/subscribe. Indeed, at first sight one might argue that security contradicts its open and decoupled nature.



(a) Real world trust (b) Implementation

Figure 1. Trust

Figure 1 depicts a common example from the domain of e-commerce applications: a customer subscribes to a "premium stock market ticker" provided by *Reuters*, for instance. As the service comes with a monthly fee, contracts are concluded between customers and the service provider (Fig. 1(a)) describing the terms of their trust relationships.

Obviously, the provider of such a service has an interest in *access control*. Only *authenticated* and *authorized* customers should be able to receive the stock market quotes. The most basic requirement for a security implementation is to allow access to the service for the group of valid customers and to deny access to anybody else. On the other hand, a customer of such a premium service wants to be sure that information received from the service is *authentic*, i.e., originates from the premium service and is not manipulated. Therefore the customer has to trust the *authenticity* and *validity* of the received information. Taken together, provider and customers share a common group of trust in which they interact.

Obviously, the presence of the pub/sub infrastructure as an intermediary introduces an additional level of trust concerns. Not only that the infrastructure must be trustworthy itself, it also must be leveraged to implement the trust relationship between the producer and consumer (cf. Fig. 1(b)). From the point of view of a *group of trust*, as described above, the infrastructure must be part of the application-specific group of trust that customers and provider share.

Consequently, an implementation of real-world trust must secure groups of application components *and* the underlying groups of event brokers necessary to connect the components. On both levels, measures must be taken to separate communication within the group from outsiders and to base group admission on credentials sufficient to establish mutual trust.

2.3 Current Deficiencies

Contemporary design of pub/sub services focuses on functional aspects of the pub/sub paradigm, i.e., efficiency of message routing, scalability, expressiveness of filter languages, or event composition, to name only a few.

However, trust and security is not part of the pub/sub paradigm, and the trust relationship is not directly enforcable in producers and consumers. Security is a separate aspect of publish/subscribe, *outside* of the pure *ability* to convey messages. Trust is injected into a system based on external contracts on the level of applications. The goal must be to map trust agreements to the underlying notification service for implementation and enforcement. The current model of pub/sub assumes that the black box model of a conceptually centralized pub/sub service is applicable at all times. But implementing a group of trust requires additional control on *how* messages are delivered in the infrastructure.

What is needed to implement trust and security on top of the pub/sub paradigm, is fine-grained control over every part of the infrastructure used to transport messages from a producer to a consumer. Each part on this way has to have the same trustworthiness *as if* producer and consumer would communicate directly. To inject this extensive level of control we exploit the concept of scoping introduced in the next section.

3 Scoping

Scopes in publish/subscribe systems [5, 6] delimit groups of producers and consumers on the application level and control the dissemination of notifications within the infrastructure. Hence, they offer a technical basis to realize groups of trust. This section describes their basic functionality and security extensions are shown in Section 4.

3.1 Model

The fundamental idea of the scoping concept is to control the visibility of notifications outside of application components and orthogonal to their subscriptions. A scope *bundles* a set of simple application components, i.e., producers and consumers, which are typically not aware of this bundling. Additionally, it may contain other scopes as well. The resulting structure of the system is given by a directed acyclic graph of simple and complex components .

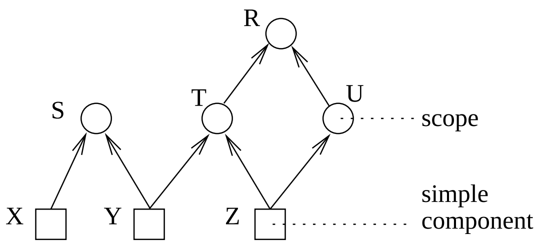


Figure 2. An exemplary scope graph

The visibility of notifications is initially limited to the scope they are published in. The transition of notifications between scopes is governed by *scope interfaces*, i.e., a scope issues subscriptions and advertisements in order to act as regular producer and consumer in its superscope(s). The scope's interface selects the eligible notifications that are forwarded to their superscopes and the external notifications that are relayed towards the scope's (sub-)components. In Figure 2, a notification published by Z is delivered to Y and to any other consumers in T and U if their subscription matches. Also, it is visible in R if it matches the output interface of T or U, but it is not visible in S.

3.2 Using Scopes

Four new functions are needed for maintaining a scope graph: *creating* new scopes (`cscope`), *destroying* scopes (`dscope`), *joining* an existing scope (`jscope`), and *leaving* a scope (`lscope`). Two approaches to scope administration exist. First, the functions may be directly accessed by the clients of the pub/sub service, i.e., the producers and consumers, i.e., the components of applications. In this case the functions are provided as extensions of the publish/subscribe API.

However, in accordance with the loose coupling of the event-based paradigm, scope management should be done outside of the application components. We identified the role of an administrator who is responsible for orchestrating existing components into new scopes, which in turn are available for higher level composition. At deployment time, *descriptors* assign newly created components to certain scopes. At runtime, we leverage management interfaces to remotely administrate scope membership of existing components. For the implementation of *trust* we can exploit the same mechanisms for assigning components to certain application-dependent scopes, representing a group of trusted components.

3.3 Scope Architectures

We sketch a distributed implementation of scopes as an extension of the REBECA distributed notification service [10]. This approach opens the black box and determines groups of event brokers that implement a specific scope, thus correlating groups on the application and the system level.

Integrated routing reconciles distributed notification routing with the visibility constrains defined by the scope graph. The original routing table is broken into multiple tables, one for each locally available scope. Thus, for each scope a connected subset of event brokers constitute an overlay within the broker network that conveys scope-internal traffic. Another routing table, the scope routing table, records scope-link pairs signifying in which directions brokers of the respective scope can be found.

Upon scope creation, an initially empty routing table is created at some broker, together with any management information regarding this scope, such as interface definitions. The creation is announced with a notification that is distributed in the network to update the scope routing tables. The overlay can either be extended manually by administrative commands to preset a certain extent of the overlay, or it is extended dynamically when other components are to join the scope. Both ways, a scope *join request* is always issued at a broker currently not part of the overlay. A request is traveling in the direction stored in the scope's routing table, leaving a temporary trail of references to the request source. The first broker encountered that is part of the requested scope, processes the join request and sends a reply back along the trail. If affirmative, the reply contains management information needed to set up routing tables in the involved brokers; they become part of the scope's overlay.

The transition of notifications between two scopes requires the two scope overlays to share at least one broker. Consider scopes T and R of Figure 2. T is a component of R and has joined R. For each subscription of T, a respective entry is added to the routing table of R that points to the table of T. For each advertisement an entry is added in

T's table that points to R. Mechanisms are in place to prevent multiple transitions at different brokers, but they are omitted here.

With this implementation, scopes not only group clients of the pub/sub service on the application level. They are also an important tool to group event brokers, extending their structuring capabilities to the infrastructure. They determine which subset of brokers belong to the same grouping and even allow for different routing algorithms in separate overlays as long as the transition between the scopes adhere to the constraints of the scope graph.

4 Security in Scopes

The preceding discussion introduced scopes as a means to group application and infrastructure components. They are therefore an apparent place to implement groups of trust. A scope isolates intra-scope traffic from the rest of the system, if the infrastructure is trusted. In Section 4.1 we address access control of clients, i.e., at the application level. Section 4.2 enhances scope overlay management to extend application-depend trust groups to the infrastructure.

4.1 Client Access Control

In many scenarios, like e-Commerce applications or mobile applications, access to the pub/sub infrastructure must be controlled on the level of subscriptions, advertisements, and publications, i.e., client access control. It must be ensured that only authorized clients have access to the network of brokers to publish and subscribe to notifications they are privileged to. In general, access control is implemented at the border brokers of a system, assuming a trusted infrastructure (cf. Section 4.2).

The presented solution uses rather simple policies because the main focus lies on how security is integrated—more sophisticated policies would be available if role-based access control schemes are bound to scopes, cf. [1]. *Attribute certificates* (AC) as specified in RFC 3281 [4] are utilized to encode privileges. An AC is a credential with a digitally signed (or certified) identity and a set of attributes. It carries here a reference to a public key certificate to authenticate the client and authorized filter expressions the client is allowed to advertise or subscribe for. ACs are issued by the provider of the broker network itself or by some other trusted *attribute authority* (AA). A legitimate content provider has got an AC from the network provider that authorizes its advertising. On the other hand, access to premium content may require an AC of the content provider, which is checked by the network.

Consider a service requesting the pub/sub system to propagate an advertisement A. To do so, it calls `advertise` of the pub/sub interface together with an AC showing its privilege to do so. The border broker verifies the AC by checking the contained signature of the net-

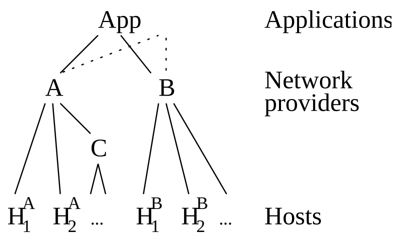


Figure 3. Trust relationships

work provider and, depending on the result of the check, grants access or, e.g., simply discards the advertisement. Included in a valid advertisement is another certificate carrying the public key of the content provider for this advertisement. Later, this certificate is used to authorize subscribers to the content published after the advertisement (e.g., the group of "premium content subscribers"). Advertisements are flooded through the overlay network of the scope they are published in. Thereby, access control information for subscriptions matching the advertisement is made available at all border brokers—overlay extensions are handled transparently as the network is trusted, so far.

When a client subscribes to some information at a border broker, it also gives its credentials in form of an AC. The border broker checks the signature of the certificate based on the network provider key or the keys contained in its list of received advertisements. The AC the client provides must match the attributes specified by the signing AA contained in the advertisement. If and only if they match, the subscription is processed further like in the standard pub/sub case without additional security.

4.2 Infrastructure Security

So far, we considered access control on the level of applications, i.e., in the graph of scopes. As discussed in Section 2, the trust relationship manifested in the application must also be secured within the infrastructure. Routing and the decision (how) to use specific parts of the broker network is subjected to application-specific security considerations. Scoping is exploited to correlate groups of application components with groups of trusted event brokers, making the scope overlay accessible to system engineers.

The previous assumption of having a homogeneous trusted network is relaxed and we first investigate when to extend a scope overlay. One can suppose large broker networks to be hosted by different service providers, like it is the case for the Internet. City carriers are likely to provide event brokers on wireless access points while global players link cities, countries, and continents. We assume a trust relationship as sketched in Fig. 3, e.g., negotiated in business contracts. Certificates authenticate hosts, their relation to providers, and the provider-application relation.

Connected Overlays. Assume that a scope overlay in a trusted network of brokers H_1^A, H_2^A, and H_3^A exists (Figure 4) and that a scope join request is received from a neighbor broker H_1^B, which is not yet part of the overlay. The decision whether the requesting, directly connected broker is trusted is application- and scope-specific. If positively ascertained, the implementation described in Section 3.3 is used for extending the scope overlay. A likely pre-installed policy of the network provider is to trust all the brokers within its own administrative domain.

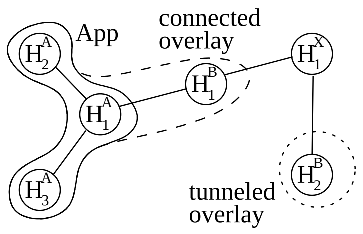


Figure 4. Extending the scope overlay

If a broker from a different administrative domain, say H_1^B, is asked to join the scope, it forwards a scope join request according to its scope routing table towards H_1^A. It appends to the request a list of chained attribute certificates of the path in the trust graph of Figure 3 from its node to the node of the respective scope. Upon receiving such a request, H_1^A tests the included certificates. If a shared ancestor in the trust relationships is found, extending the overlay may proceed as described. At this place, various security policies could be applied that are assigned to the scope *App* to govern its extension in the broker network. For instance, a scope might mandate link layer encryption with Transport Layer Security (TLS).

Tunneling. If a trustworthy node is about to join that is only reachable via an untrusted broker, the previous approach is not applicable. Consider a join request from a host H_2^B that is routed through an untrusted broker H_1^X. The latter is assumed to have a routing entry for the scope *App* in its scope routing table. H_2^B digitally signs the request and includes its own public key. If H_1^A accepts the request, the scope overlay would include an untrusted intermediary if the above implementation is used. The solution applied here is to *tunnel* the traffic through H_1^X. The clear text part of the reply contains an indication of whether to tunnel the scope and, if so, triggers an update of the scope routing table to include an entry pointing to H_2^B—provided that H_1^X cooperates. Notifications are encrypted and tagged with the scope's name so that they can be forwarded by H_1^X, although they are not part of its content-based routing. Eavesdropping and modifications are prevented, while malevolent omissions are detectable by application level heartbeats.

The tunnel can span more than one broker and it may even be used to connect clients via untrusted border brokers. The problem is, however, that multiple join requests lead to multiple tunnels. A second broker requesting to join the scope via H_1^X, or multiple clients connected via point-to-point tunnels at untrusted border brokers, will result in duplicated messages individually encrypted for the various destinations. At least in the former case of multiple trusted brokers behind an untrusted one, scope-level broadcast with a shared session key can attenuate bandwidth consumption. The same session key is forwarded to any newly attached broker so that the overlay connected via H_1^A-H_1^X is reached with only one message. Of course, this trades computing resources with bandwidth, for the new brokers have to filter out notifications consumed at other brokers.

The described tunneling is similar to secure (application-level) multicast, giving raise to the known problems of multicast key management [9]. Shared session keys must be changed if some brokers leave the overlay. However, if session keys are only used between brokers, it is plausible to assume that fluctuation is rather low and the frequency of key changes is limited.

5 Implementation

Clients access the REBECA notification service via *local event brokers*, which offer the plain pub/sub API as a library collocated to the client code. Local brokers maintain connections to the event broker network. There, event brokers are implemented as separate processes, which maintain TCP connections to other brokers and clients and at least one routing table for unscoped traffic. Brokers are customizable *software containers* and thus the implementation of the routing engine, connection pooling, transmission protocols and message handlers is specified at deployment of the broker. REBECA messages transmitted between brokers may contain (a) control messages, like subscriptions and scope admin messages, or (b) notifications, which consist of a management header and notification data. Appropriate message handlers process these messages according to their type (a) or (b).

Scope configuration is accessed through a remote management interface to the event broker functionality, using the Java Management Extensions (JMX) (cf. Figure 5). On creation of a scope, the desired scope parameters can either be specified directly, or via a *scope type*, which refers to a predefined configuration.

Although flexible, the current REBECA architecture does not allow for an easy inclusion of security policies. A number of core classes would have to be re-implemented for the integration of each specific kind of security handling. Furthermore, the proposed implementation of security is only partially tied to the sketched scope implementation – integrated routing in this case – and is designed to be applicable to other forms, as well.

```
interface ScopeEBIf :
    public RemoteEBInterface {
  createScope (ScopeName, IOInterface)
  createScope (ScopeName, IOInterface,
               SuperScopeName)
  createScope (ScopeName, Type)

  joinScope    (ComponentName, ScopeName)

  subscribe (ScopeName, Filter)
  ...
}
```

Figure 5. Remote event broker interface

To achieve greater flexibility, we employ aspect-oriented programming (AOP) techniques and AspectJ [2] to implement security aspects of scopes. We briefly sketch three main security extensions. First, access control on the API level is required for the authorization of the invocation of management functions. Certificates are stored with the callee and are transparently sent with each call to the remote management interface. These are verified before access to management functionality is granted by the broker. Second, some features, like admission tests of join requests, are also applicable to implementations other than integrated routing. Thus, a new function was introduced that calls a list of *interceptors* before starting to update the routing tables. Third, specific to integrated routing, extending the scope overlay must be governed by an authorization test of the original requesting broker and the next-hop broker. This test checks chains of certificates according to Figure 3 and is evaluated prior to calling the handler that processes the extension. Depending on the result a new *session key* may be generated. It must be added to all affected routing table entries by the extension handler and is used for secure message exchange between brokers over untrusted parts of the broker network. The encrypted fan-out to consumers uses point-to-point connections; in case of performance problems, caching schemes like [11] may be employed to reduce the number of encryptions needed.

6 Conclusion

Trust in publish/subscribe systems cannot be associated with specific producers and consumers without impairing their loose coupling. Instead, we have associated trust with the interaction within a group of components. This facilitates the design of loosely coupled applications and their security policies. Security is considered on the level of both applications and its implementation in the infrastructure, allowing for enforcement of security measures even across different administrative domains.

We introduced scopes as a suitable means to model and implement the above issues. Originally designed as general concept to control visibility, they make component interaction explicit. By opening up the black box of the pub/sub service, they provide for the appropriate locations to *weave* security aspects into a distributed pub/sub notification service. The separation of intra-scope traffic makes it possible to implement different security implementations that are bound to different parts of an application's structure, depending on the actual needs for security and trust. To avoid re-implementing larger parts of the pub/sub service every time the system evolves, we employed AOP technology to add the implementation to the REBECA notification service.

References

[1] 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)*, June 2003. ACM Press.

[2] T. Elrad, R. E. Filman, and A. Bader. Aspect-oriented programming: Introduction. *Communications of the ACM*, 44(10):29–32, 2001. Special Issue on Aspect-Oriented Programming.

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

[4] S. Farrell and R. Housley. An internet attribute certificate profile for authorization. Request For Comment 3281 (RFC 3281), April 2002.

[5] L. Fiege, M. Mezini, G. Mühl, and A. P. Buchmann. Engineering event-based systems with scopes. In *Proc. of the ECOOP 2002, LNCS* 2374, Malaga, Spain, June 2002. Springer-Verlag.

[6] L. Fiege, G. Mühl, and F. C. Gärtner. Modular event-based systems. *The Knowledge Engineering Review*, 17(4):359–388, 2003.

[7] W. Harrison and H. Ossher. Subject-oriented programming (A critique of pure objects). In *Proc. of OOPSLA 1993*, 411–428, 1993.

[8] 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, 2002. IEEE Press.

[9] R. Molva and A. Pannetrat. Network security in the multicast framework. In *NETWORKING 2002 Tutorials, LNCS* 2497, pages 59–82, Pisa, Italy, 2002. Springer-Verlag.

[10] G. Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Darmstadt University of Technology, 2002. http://elib.tu-darmstadt.de/diss/000274/.

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

[12] C. Wang, A. Carzaniga, D. Evans, and A. L. Wolf. Security issues and requirements for Internet-scale publish-subscribe systems. In *Proc. of the HICSS-35*, Big Island, Hawaii, Jan. 2002.