DECOUPLING COMPONENTS OF AN ATTACK PREVENTION SYSTEM USING PUBLISH/SUBSCRIBE*

Joaquín García[1], Michael A. Jaeger[2], Gero Mühl[2], and Joan Borrell[1]

[1] *Autonomous University of Barcelona,*
Dept. of Information and Communications Engineering,
Edifici Q, 08193 Bellaterra, Spain
{jgarcia,jborrell}@ccd.uab.es

[2] *Technical University of Berlin,*
Institute for Telecommunication Systems,
Communication and Operating Systems Group,
EN6, Einsteinufer 17, D-10587 Berlin, Germany
{michael.jaeger,g_muehl}@acm.org

Abstract Distributed and coordinated attacks can disrupt electronic commerce applications and cause large revenue losses. The prevention of these attacks is not possible by just considering information from isolated sources of the network. A global view of the whole system is necessary to react against the different actions of such an attack. We are currently working on a decentralized attack prevention framework that is targeted at detecting as well as reacting to these attacks. The cooperation between the different entities of this system has been efficiently solved through the use of a publish/subscribe model. In this paper we first present the advantages and convenience in using this communication paradigm for a general decentralized attack prevention framework. Then, we present the design for our specific approach. Finally, we shortly discuss our implementation based on a freely available publish/subscribe message oriented middleware.

1. Introduction

When attackers gain access to a corporate network by compromising authorized users, computers, or applications, the network and its resources can be-

*This work has partially been funded by the Spanish Ministry of Science and Technology (MCYT) through the project TIC2003-02041 and the Catalan Ministry of Universities, Research and Information Society (DURSI) with its grant 2003FI-126.

come an active part of a globally distributed or coordinated attack. Such an attack might be a coordinated port scan or distributed denial of service attack against third party networks — or even against computers on the same network. Both, distributed and coordinated attacks, rely on the combination of actions performed by a malicious adversary to violate the security policy of a target computer system. To prevent these attacks, a global view of the system as a whole is necessary. Hence, different events and specifi c information must be gathered and combined from all the sources. This affects, for example, information about suspicious connections, initiation of processes, and addition of new fi les.

We are currently working on the design and development of an attack prevention framework that is targeted at detecting as well as reacting to distributed and coordinated attack scenarios [García et al., 2004]. Our approach is based on gathering and correlating information held by multiple sources. We use a decentralized scheme based on message passing to share alerts in a secure communication infrastructure. This way, we can detect and prevent these kind of attacks performing detection and reaction processes based on the knowledge gained through alert correlation.

In this paper we propose a decentralized infrastructure to share alerts between components. The information exchange between peers is intended to achieve a more complete view of the system in whole. Once achieved, one can detect and react on the different actions of a coordinated or distributed attack.

The rest of this paper is organized as follows: We start with an introduction to the publish/subscribe communication paradigm in Section 2 where we present the advantages and convenience in using this model for our problem domain and analyze related work. In Section 3, we discuss the communication mechanism used to exchange information among the components of our system using xmlBlaster, an open source publish/subscribe message oriented middleware [Ruff, 2000] and present the current state of our implementation. We close with conclusions and give an outlook on future work in Section 4.

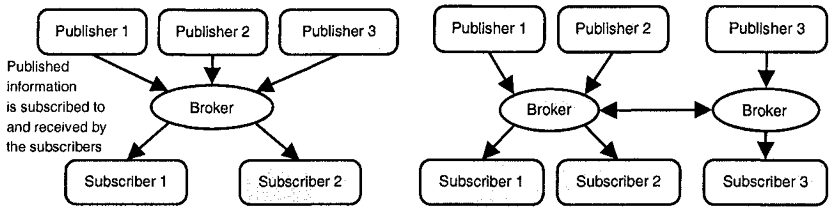
2. Publish/Subscribe Model

The publish/subscribe communication model is intended for group communication, i.e. for situations where a message (*notifications*) sent by a single entity is required by, and should be distributed to, multiple entities. It is often used for effi cient and comfortable information dissemination to group members which may have individual interests in arbitrary subsets of messages published. In contrast to multicast communication, clients have the possibility to describe the events they are interested in more precisely (e.g. based on the contents of the notifi cation). Clients can choose to either subscribe or unsubscribe to messages as time goes by, and all the subscribers are independent of each other.

Publish/Subscribe Systems

A publish/subscribe system consists of at least one broker forwarding notifi - cations published by clients to other clients that are interested in them. For scalability reasons, it is common to implement a distributed broker network that forms a so-called *notification service* through an overlay network consisting of brokers. This service provides a distributed infrastructure for notifi cation routing which includes the management of subscriptions and the dissemination of notifi cations in a possibly asynchronous way. Clients can publish notifi cations and subscribe to fi lters that are matched against the notifi cations passing through the broker network. If a broker receives a new notifi cation it checks if there is a local client that has subscribed to a fi lter that matches this notifi cation. If so, the message is delivered to this client. Additionally, the broker forwards the message to neighbor brokers according to the applied routing algorithm. We refer to [Mühl, 2002] for a good survey on the fi eld.

An example of a simple centralized publish/subscribe system is shown in Figure 1(a). Here, fi ve clients are connected to a single broker: three clients that are publishing notifi cations and two clients that are subscribed to a subset of the notifi cations published on the broker. Subscribers can choose to subscribe to the notifi cations available through the broker or cancel existing subscriptions as needed. The broker matches the notifi cations it received from the publishers to the subscriptions, ensuring this way that every publication is delivered to all interested subscribers.



(a) Simple publish/subscribe system. (b) Extended pub/sub system.

Figure 1. Examples for publish/subscribe environments.

This very basic publish/subscribe setup can be extended by connecting multiple brokers (cf. Figure 1(b)), enabling them to exchange messages. The extended design allows subscribers on one of the brokers to receive messages that have been published on another broker, further freeing the subscriber from the constraints of connecting to the same broker the publisher is connected to. Most available implementation make this transparent for the programmer by keeping the same interface operations as in the centralized design. This way, an application can easily be distributed. The subscribers are able to formulate their interests based e.g. on the contents of the notifi cations and a special attribute they carry. This is known as content-based and topic-based subscription, respectively.

Topic-based subscriptions are easier to handle than content-based subscriptions. Subscribers specify their interest in a topic and receive all messages published on this topic. Two different matching mechanisms are commonly used here. One matches subscriptions successfully to notifi cations if the topic of the subscription exactly matches the topic under which the notifi cation is published. Using this mechanism, topics become equivalent to "channels". The other mechanism arranges topics in a subject tree such that subscriptions not only match notifi cations if the topics are the same, but also if the topic of the subscription is an ancestor of the notifi cation topic in the subject tree (in this case, a topic becomes equivalent to a "theme").

Content-based subscriptions allow more sophisticated subscriptions on the cost of higher matching load and more complex routing decisions. Here, a subscription can be formulated extremely fi ne-grained based on the content of notifi cations using a query language that can be arbitrarily complex. Moreover, there does not have to be a system wide agreement on the set of topics as it is generally a good idea for topic based routing.

Related Work

Traditional client/server solutions for the prevention of distributed and coordinated attacks can quickly become a bottleneck due to saturation problems associated with the service offered by centralized or master domain analyzers. A master domain analyzer is the entity on top of a hierarchy of IDSs consisting of multiple analyzers and different domains to analyze. Centralized systems, such as DIDS [Snapp et al., 1991] and NADIR [Hochberg et al., 1993], use this approach to process their data in a central node although the collection of data is distributed. These schemes are straightforward as they simply push the data to a central node and perform the computation there. Hierarchical approaches, such as GrIDS [Staniford-Chen et al., 1996] and NetSTAT [Vigna and Kemmerer, 1999], have a layered structure where data is locally preprocessed and fi ltered. Although they mitigate some weaknesses present in centralized schemes, they still cannot avoid bottlenecks, scalability problems, and fault tolerance issues due to vulnerabilities at the root level.

In contrast to these traditional designs, alternative approaches try to eliminate the need for dedicated elements. The idea of distributing the detection process has some advantages regarding centralized and hierarchical approaches. Mainly, decentralized architectures have no single point of failure and bottlenecks can be avoided. Some message passing designs, such as CSM [White et al., 1999] and Quicksand [Kruegel, 2002], try to eliminate the need for dedicated elements by introducing a peer-to-peer architecture. Instead of having a central monitoring station to which all data has to be forwarded, there are independent uniform working entities at each host performing similar basic operations. To detect coordinated and distributed attacks, the different entities have to collaborate on the detection activities and cooperate to perform a decentralized correlation algorithm.

These designs seem to be a promising technology to implement decentralized architectures for the detection of attacks. However, the presented systems still exhibit very simplistic designs and suffer from several limitations. For instance, in some of them, every node has to have complete knowledge of the system: All nodes have to be connected to each other which can make the matrix of the connections, that are used for providing the alert exchanging service, grow explosively and become very costly to control and maintain. Another important disadvantage present in this design is that the different entities always need to know where a received notifi cation has to be forwarded (similar to a queue manager). This way, when the number of possible destinations grows, the network view can become extremely complex, which leads to a system that is not scalable. Other designs are based on flooding which makes the system easier to maintain on the cost of scalability, as the message complexity grows fast with the number of brokers.

Most of these limitations can be solved effi ciently by using a publish/subscribe based system. The advantage of this model for our problem domain over other communication paradigms is on the one hand that it keeps the producer of messages separated from the consumer and on the other hand that the communication is information-driven. This way, it can avoid problems regarding the scalability and the management inherent to other designs, by means of a network of publishers, brokers, and subscribers. A publisher in a publish/subscribe system does not need to have any knowledge about any of the entities that consume the published information. Likewise, the subscribers do not need to know anything about the publishers. New services can simply be added without any impact on or interruption of the service to other users.

3. Alert Communication Infrastructure

This section describes the alert communication infrastructure and implementational details of our approach. As our motivation is not targeted on developing a new publish/subscribe system, we try to reuse as much available code and tools as possible. For our experiments we used *xmlBlaster*, an open source publish/subscribe message oriented middleware [Ruff, 2000]. It connects a set of nodes that build up the infrastructure for exchanging alerts using the interface operations offered by the underlying middleware.

Alerts are formulated using XML as this is the standard format in xmlBlaster. Each message consists of a header fi ltering can be applied to, a body, and a system control section. Filters are XPath expressions that are evaluated over the header to decide if a message has to be delivered to a subscriber. We discuss the essential interface operations offered by xmlBlaster in the following section.

Interface Operations

Conceptually, the alert communication infrastructure offered through xmlBlaster can be viewed as a black box with an *interface.* It offers a number of *operations*, each of which may take a number of *parameters.* Clients can invoke *input operations* from the outside, and the system itself invokes *output operations* to deliver information to clients. We list the main operations that are of interest for our work in Figure 2.

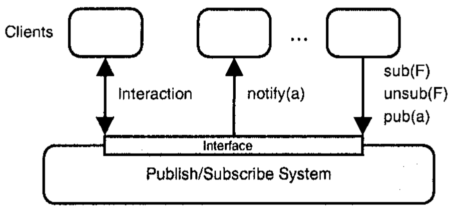


Figure 2. Black box view of a publish/subscribe system.

To publish alerts, clients invoke the *pub*(a) operation, giving the alert a as parameter. The published alert can potentially be delivered to all clients connected to the system via an output operation called *notify*(a). Clients register their interest in specifi c kinds of alerts by issuing subscriptions via the *sub*(F) operation, which takes a fi lter F as parameter. Each client can have multiple active subscriptions which must be revoked separately by using the *unsub*() operation.

All these operations are instantaneous and take parameters from the set of all clients $\mathcal{C}$, the set of all alerts $\mathcal{A}$, and the set of all fi lters $\mathcal{F}$. Formally, a fi lter $F \in \mathcal{F}$ is a mapping defi ned by

$$F : \quad a \longrightarrow \{\text{true}, \text{false}\} \qquad \forall a \in \mathcal{A}$$

We say that a *notification* n *matches filter* $F \in \mathcal{F}$ iff $F(a) = \text{true}$. We also assume that each alert can only be published once and that every fi lter is associated with a unique identifi er in order to enable the alert communication infrastructure to identify a specifi c subscription.

Components and Interactions

As shown in Figure 3, each node of the architecture is made up of a set of local analyzers (with their respective detection units or sensors), a set of alert managers (to perform alert processing and manipulation functions), and a set of local reaction units (or effectors). These components, the interactions between them, and the alert communication infrastructure, are described below.

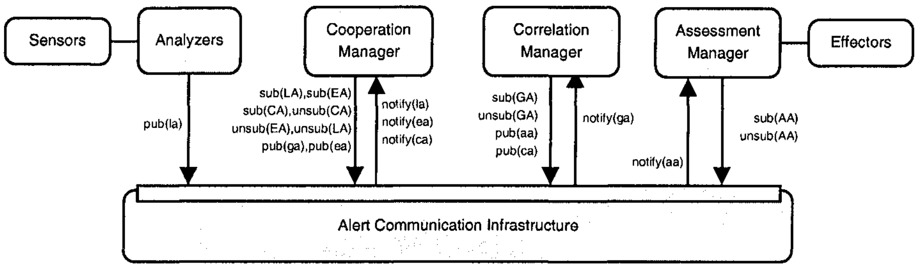


Figure 3. Overview of the Attack Prevention Framework.

Analyzers. Local elements which are responsible for processing local audit data are called *analyzers*. They process the information gathered by associated sensors to infer possible alerts. Their task is to identify occurrences which are relevant for the execution of the different steps of an attack and pass this information to the correlation manager via the publish/subscribe system. They are interested in local alerts. Each local alert is detected in a sensor's input stream and published through the publish/subscribe system by invoking the *pub*(la) operation, giving the *local alert* la as parameter.

Each notifi cation la has a unique classifi cation and a list of attributes with their respective types to identify the analyzer that originated the alert (*AnalyzerID*), the time the alert was created (*CreateTime*), the time the event(s) leading up to the alert was detected in the sensor's input stream (*DetectTime*), the current time on the analyzer (*AnalyzerTime*), and the source(s) and target(s) of the event(s) (*Source* and *Target*). All possible classifi cations and their respective attributes must be known by all system components (i.e. sensors, analyzers and managers) and all analyzers are capable of publishing instances of local alerts of arbitrary types.

Local alerts are exchanged using IDMEF messages [Debar et al., 2005]. The *Intrusion Detection Message Exchange Format* (IDMEF) is proposed as a standard data format for automated intrusion detection systems to raise alerts about events they report as suspicious. It allows analyzers and managers to assemble very complex alert descriptions.

Managers. Performing aggregation and correlation of local alerts and external events is the task of *managers*. While using multiple analyzers and sensors together with heterogeneous detection techniques increases the detection rate, it also increases the number of alerts to process. In order to reduce the number of false negatives and distribute the load that is imposed by the alerts our architecture provides a set of cooperation and correlation *managers*, which perform aggregation and correlation of both, local alerts (i.e., messages provided by the node's analyzers) and external messages (i.e., the information received from other collaborating nodes).

Cooperation Managers. The basic functionality of each cooperation manager is to cluster alerts that correspond to the same occurrence of an action. Each cooperation manager registers its interest in a subset $\mathcal{L}_A$ of local alerts published by analyzers on the same node by invoking the *sub*(LA) operation, which takes the fi lter LA as parameter, with

$$LA(a) = \begin{cases} \text{true} & , \ a \in \mathcal{L}_A \\ \text{false} & , \ \text{otherwise.} \end{cases}$$

Similarly, the cooperation manager also registers its interest in a set of related external alerts $\mathcal{E}_A$ by invoking the *sub*(EA) operation with fi lter EA as parameter, and

$$EA(a) = \begin{cases} \text{true} & , \ a \in \mathcal{E}_a \\ \text{false} & , \ \text{otherwise.} \end{cases}$$

Finally, it registers its interest in local correlated alerts $\mathcal{C}_A$ by invoking the *sub*(CA) operation with

$$CA(a) = \begin{cases} \text{true} & , \ a \in \mathcal{C}_A \\ \text{false} & , \ \text{otherwise.} \end{cases}$$

Once subscribed to these three fi lters, the alert infrastructure will notify the subscribed managers of all matching alerts via the output operations *notify*(la), *notify*(ea) and *notify*(ca) with $la \in \mathcal{L}_A$, $ea \in \mathcal{E}_A$ and $ca \in \mathcal{C}_A$. All notifi ed alerts are processed and, depending on the clustering and synchronization mechanism, the cooperation manager can publish global and external alerts by invoking *pub*(ga) and *pub*(ea). Finally, it can revoke active subscriptions separately by using the operations *unsub*(CA), *unsub*(EA) and *unsub*(LA).

Correlation Managers. The main task of this manager is the correlation of alerts described in [García et al., 2004]. It operates on the set of global alerts $\mathcal{G}_A$ published by the local cooperation manager. To register its interest in these alerts, it invokes *sub*(GA), which takes the fi lter GA as parameter with

$$GA(a) = \begin{cases} \text{true} & , \ a \in \mathcal{G}_A \\ \text{false} & , \ \text{otherwise.} \end{cases}$$

The alert infrastructure will then notify the correlation manager of all matched alerts with the output operation *notify*(ga), $ga \in \mathcal{G}_A$. Each time a new alert is received, the correlation mechanism finds a set of action models that can be correlated in order to form a scenario leading to an objective. Finally, it includes this information into the *CorrelationAlert* field of a new IDMEF message and publishes the correlated alert by invoking *pub*(ca), giving the notification $ca \in \mathcal{C}_A$ as parameter. To revoke the subscription, it uses *unsub*(GA).

The correlation manager is also responsible for reacting on detected security violations. The algorithm used is based on the anti-correlation of actions to select appropriate countermeasures in order to react and prevent the execution of the whole scenario [García et al., 2004]. As soon as a scenario is identified, the correlation mechanism looks for possible action models that can be anti-correlated with the individual actions of the supposed scenario, or even with the goal objective. The set of anti-correlated actions represents the set of countermeasures available for the observed scenario. The definition of each anti-correlated action contains a description of the countermeasures which should be invoked (e.g. hardening the security policy). Such countermeasures are included into the *Assessment* field of a new IDMEF message and published by invoking *pub*(aa), using the *assessment alert* aa as parameter.

Assessment Managers. Another manager called *assessment manager* will register and revoke its interest in these assessment alerts by invoking *sub*(AA) and *unsub*(AA). Once notified, the assessment manager performs post-processing of the received alerts before sending the corresponding reaction to the local response units.

Implementation

We deployed a set of three analyzers publishing ten thousand messages to evaluate our implementation of the alert communication infrastructure for the proposed architecture. Therefore, we used the *DARPA Intrusion Detection Evaluation Data Sets* [Lippmann et al., 2000] where more than 300 instances of 38 different automated attacks were launched against victim hosts in seven weeks of training data and two weeks of test data. These messages were published as local alerts through the communication infrastructure, and then processed and republished in turn to three subscribed managers. The evaluation on the alert communication infrastructure proved to be satisfactory, obtaining a throughput performance higher than 150 messages per second on an Intel-Pentium M 1.4 GHz processor with 512 MB RAM, analyzers and managers on the same machine running Linux 2.6.8, using Java HotSpot Client VM 1.4.2 for the Java based broker. Message delivery did not become a bottleneck as all messages were processed in time and we never reached the saturation point. This result gives us good hope that using a publish/subscribe system for the communication infrastructure indeed increases the scalability of the proposed architecture.

The implementation of both analyzers and managers was based on the *libidmef* C library [Migus, 2004] which was used to build and parse compliant IDMEF messages. The communication between analyzers and managers through xmlBlaster brokers was based on the xmlBlaster internal socket protocol and implemented using the xmlBlaster client C socket library [Ruff, 2000], which provides asynchronous callbacks to Java based brokers. The managers formulated their subscriptions using XPath expressions, fi ltering the messages they wished to receive from the broker.

4. Conclusions

We presented an infrastructure to share alerts between the components of a prevention framework. The framework itself is targeted at detecting as well as reacting to distributed and coordinated attack scenarios through the use of the publish/subscribe communication paradigm. In contrast to traditional client/server solutions, where centralized or hierarchical approaches quickly become a bottleneck due to saturation problems associated with the service offered by centralized or master domain analyzers, the information exchange between peers in our design achieves a more complete view of the system in whole. We believe that this is necessary to detect and react on the different actions of an attack. We also introduced an implementation based on an open source publish/subscribe message oriented middleware and conducted experiments showing that the architecture is performant enough for the application in real-world scenarios.

As future work we are considering to secure the communication partners by utilizing the SSL plugin for xmlBlaster. This way, each collaborating node will receive a private and a public key. The public key of each node will be signed by a certifi cation authority (CA), that is responsible for the protected network. Hence, the public key of the CA has to be distributed to every node as well. The secure SSL channel will allow the communicating peers to communicate privately and to authenticate each other, thus preventing malicious nodes from impersonating legal ones. The implications coming up with this new feature, such as compromised key management or certifi cate revocation, will be part of this work. We are also planning a more in-depth study about privacy mechanisms by exchanging alerts in a pseudonymous manner. By doing this, we hope that we can provide the destination and origin information of alerts (*Source* and *Target* fi eld of IDMEF messages) without violating the privacy of publishers and subscribers located on different domains. Our study will cover the design of a pseudonymous identifi cation scheme, trying to fi nd a balance between identifi cation and privacy.

Acknowledgments

The collaboration between J. García, F. Cuppens, and F. Autrel sharpened many of the arguments presented in this paper.

The work of J. García and J. Borrell is funded by the Spanish Government CICYT Commission and the Catalan Government DURSI Ministry. Michael A. Jaeger is funded by Deutsche Telekom Stiftung. Gero Mühl is funded by Deutsche Telekom.

References

[Debar et al., 2005] Debar, H., Curry, D., and Feinstein, B. (January 2005). Intrusion detection message exchange format data model and extensible markup language. Technical report.

[García et al., 2004] García, J., Autrel, F., Borrell, J., Castillo, S., Cuppens, F., and Navarro, G. (2004). Decentralized publish-subscribe system to prevent coordinated attacks via alert correlation. In *Sixth International Conference on Information and Communications Security*, volume 3269 of *LNCS*, pages 223–235, Málaga, Spain. Springer-Verlag.

[Hochberg et al., 1993] Hochberg, J., Jackson, K., Stallins, C., McClary, J. F., DuBois, D., and Ford, J. (May 1993). NADIR: An automated system for detecting network intrusion and misuse. In *Computer and Security*, volume 12(3), pages 235–248.

[Kruegel, 2002] Kruegel, C. (June 2002). *Network Alertness - Towards an adaptive, collaborating Intrusion Detection System.* PhD thesis, Technical University of Vienna.

[Lippmann et al., 2000] Lippmann, R., Haines, J., Fried, D., Korba, J., and Das, K. (2000). The 1999 darpa off-line intrusion detection evaluation. *Computer Networks*, (34):579–595.

[Migus, 2004] Migus, A. C. (March 2004). IDMEF XML library version 0.7.3. `http://sourceforge.net/projects/libidmef/`.

[Mühl, 2002] Mühl, G. (2002). *Large-Scale Content-Based Publish-Subscribe Systems.* PhD thesis, Technical University of Darmstadt.

[Ruff, 2000] Ruff, M. (2000). XmlBlaster: open source message oriented middleware. `http://xmlblaster.org/`.

[Snapp et al., 1991] Snapp, S. R., Brentano, J., Dias, G. V., Goan, T. L., Heberlein, L. T., Ho, C., K. N. Levitt, Mukherjee, B., Smaha, S. E., Grance, T., Teal, D. M., and Mansur, D. (October, 1991). DIDS (distributed intrusion detection system) - motivation, architecture and an early prototype. In *Proceedings 14th National Security Conference*, pages 167–176.

[Staniford-Chen et al., 1996] Staniford-Chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Levitt, J. Hoagland K., Wee, C., Yip, R., and Zerkle, D. (1996). GrIDS – a graph-based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference.*

[Vigna and Kemmerer, 1999] Vigna, G. and Kemmerer, R. A. (1999). NetSTAT: A network-based intrusion detection system. *Journal of Computer Security*, 7(1):37–71.

[White et al., 1999] White, G. B., Fisch, E. A., and Pooch, U. W. (February 1999). Cooperating security managers: A peer-based intrusion detection system. *IEEE Network*, 7:20–23.