An Alert Communication Infrastructure for a Decentralized Attack Prevention Framework

Joaquin Garcia-Alfaro, Joan Borrell
dEIC, Autonomous University of Barcelona,
08193 Bellaterra, Catalonia, Spain
Email: Name.Surname@uab.es

Michael A. Jaeger, Gero Mühl
KBS, Berlin University of Technology,
EN6, Einsteinufer 17, D-10587 Berlin, Germany
Email: {michael.jaeger,g_muehl}@acm.org

Abstract

The cooperation between the different entities of a decentralized prevention system can be solved efficiently through the use of a publish/subscribe system. Clients share and correlate alert information about the systems they monitor. The brokers themselves form the infrastructure used for the routing of alerts. In this paper we present the advantages and convenience in using this communication model for a general decentralized prevention framework. Then, we present the design for a specific architecture, and evaluate our design through an available publish/subscribe message oriented middleware.

1 Introduction

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 [4]. 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. 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.

The communication between the different sources in our approach has been realized efficiently through the use of the publish/subscribe communication paradigm. A publish/subscribe system consists of brokers and clients that are connected to brokers. The brokers themselves form the infrastructure (notification service) used for the routing of notifications. Clients can publish notifications and subscribe to filters that are matched against the notifications passing through the broker network. If a broker receives a new notification it checks if there is a local client that has subscribed to a filter that matches this notification. If so, the message is delivered to this client. Next, the broker forwards the message to other brokers according to the applied routing algorithm.

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 whole system. Once achieved, one can detect and react on the different actions of a coordinated or distributed attack. As we discuss in Section 2, the use of a publish/subscribe paradigm efficiently fits for the communication between the different sources in our approach.

The rest of this paper is organized as follows. We first present in Section 2 the advantages and convenience in using the publish/subscribe communication paradigm for our problem domain, analyzing some related work. Then, we discuss and evaluate in Section 3 the communication mechanism used to exchange information among the components of our system through the use of xmlBlaster, an open source publish/subscribe message oriented middleware.

2 Motivations

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.

Centralized systems, such as DIDS [11] and NADIR [5], process their data in a central node although the collection of data is distributed. These schemes are straightforward as they simply place the data at a central node and perform the computation there.

Hierarchical approaches, such as GrIDS [12] and NetSTAT [13], have a layered structure where data is locally preprocessed and filtered. This way, they mitigate some weaknesses present in centralized schemes, but they still cannot avoid bottlenecks, scalability problems and fault tolerance vulnerabilities at the root level.

Current 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 or and bottlenecks can be avoided.

Some message passing designs, such as CSM [14] and Quicksand [6], 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 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 inherent to these designs is that the different entities always need to know where a received notification 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 scalability problems. Other designs are based on flooding which makes the system easy to maintain but still lacks scalability, as the message complexity grows fast with the number of brokers. Most of these limitations can be efficiently solved by means of a publish/subscribe based system.

The advantage of the publish/subscribe model over other communication paradigms for our problem domain is, that it keeps the producer of messages separated from the consumer and that the communication itself is information-driven. This model can avoid problems regarding the scalability and the mangagement 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. Basically, it consists of a set of nodes which are connected by a network and exchange alerts using a publish/subscribe model. As our motivation is not targeted on developing a publish/subscribe system, we try to reuse as much available code and tools as possible. For our experiments we use *xmlBlaster*, an open source publish/subscribe message oriented middleware [10]. The alerts are formulated in XML as xmlBlaster uses this format for its message syntax [1]. Each message consists of a header filtering can be applied to, a body, and a system control section. Filters are XPath expressions [2] that are evaluated over the header to decide if a message will be delivered to a subscriber.

3.1 Interface Operations

Conceptually, the alert communication infrastructure offered through xmlBlaster can be viewed as a black box with an interface. This *interface* 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 the outside. As pointed out in [9], these input and output operations can be formally described using linear temporal logic [7]. To publish alerts, clients invoke the *pub*(a) operation, giving the alert a as parameter. The published alert can potentially be delivered to all connected clients via an output operation called *notify*(a). Clients register their interest in specific kinds of alerts by issuing subscriptions via the *sub*(F) operation, which takes a filter F as parameter. Each client can have multiple active subscriptions which must be revoked separately by using the *unsub*(F) 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 filters $\mathcal{F}$. Formally, a filter $F \in \mathcal{F}$ is a mapping from $\mathcal{A}$ to the boolean values *true* and *false*. Hence, a *notication* n *matches filter* $F \in \mathcal{F}$ iff $F(a)$ evaluates to *true*. Additionally, we also assume that each alert can only be published once, and that every filter is associated with a unique identifier in order to enable the alert communication infrastructure to identify a specific subscription.

Each node in the architecture is made up of a set of local analyzers (with their respective detection units or sensors), and a set of alert managers (to perform alert process and manipulation functions). These components, and the interactions between them, are described below.

3.2 Analyzers

Analyzers are local elements which are responsible for processing audit data. 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. The interesting occurrences are 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 notification la (local alert) as parameter.

Each local alert notification (la) has a unique classification, 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 classifications and their respective attributes must be known by all system components (i.e. sensors, analyzers and managers) and all analyzers are capable to publish various instances of local alerts of one or more types.

The local alerts are exchanged as IDMEF messages [3], and formulated using XML syntax [1]. The *Intrusion Detection Message Exchange Format* (IDMEF) is intended to be a standard data format that automated intrusion detection systems can use to raise alerts about events that they report as suspicious. It allows analyzers and managers to assemble very complex alert descriptions.

3.3 Managers

The use of multiple analyzers and sensors using heterogeneous detection techniques increases the detection rate, but it also increases the number of alerts to process. In order to facilitate this process, and to reduce the number of false negatives, 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).

The basic functionality of each cooperation manager is the clustering of alerts that correspond to the same occurrence of an action. Each cooperation manager registers its interest in local alerts ($\mathcal{L}_A$) by invoking the *sub*(LA) operation, which takes the filter LA as parameter. The filter $LA \in \mathcal{L}_A$ is a mapping from $\mathcal{L}_A \subseteq \mathcal{A}$ (i.e., the subset of alerts published by analyzers of the same node). Similarly, the cooperation manager also registers its interest in related external alerts ($\mathcal{E}_A$) by invoking the $sub(EA)$ operation, with the filter $EA \in \mathcal{E}_A \subseteq \mathcal{A}$ as parameter, and its interest in local correlated alerts ($\mathcal{C}_A$) by invoking the *sub*(CA) operation, with the filter $CA \in \mathcal{C}_A \subseteq \mathcal{A}$ as parameter.

Once subscribed to these three filters, the alert infrastructure will notify of all matching alerts via the output operations *notify*(la), *notify*(ea) and *notify*(ca). All the notified alerts are processed and, depending on the clustering and synchronization functions, the cooperation manager can publish some global and external alerts by invoking the operations *pub*(ga) and *pub*(ea). Finally, it can revoke separately the active subscriptions by using the operations *unsub*(CA), *unsub*(EA) and *unsub*(LA).

The main task of the correlation manager is the execution of the alert correlation algorithm described in [4]. The correlation manager operates on the global alerts ($\mathcal{G}_A$) published by the local cooperation manager. To register its interest in these alerts, it invokes the *sub*(GA) operation, which takes the filter $GA \in \mathcal{G}_A \subseteq \mathcal{A}$ as parameter. Then, the alert infrastructure will notify of all matched alerts with the output operations *notify*(ga).

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. At last, it includes this information into the CorrelationAlert field of a new IDMEF message and publishes the correlated alert by invoking the *pub*(ca) operation, giving the notification ca as parameter. To revoke the subscription, it uses the *unsub*(GA) operation.

3.4 Evaluation

To evaluate the performance of the implemented alert communication infrastructure for our proposed architecture, we deployed a set of three analyzers publishing ten thousand messages, which are notified as local alerts through the communication infrastructure, and then processed and published in turn to three subscribed managers.

The throughput on the alert communication infrastructure is above 150 messages per second on a Intel-Pentium M (Centrino) processor 1400MHz with 512MB RAM (both, analyzers and managers, on the same machine running Linux 2.6.8, and using Java HotSpot Client VM 1.4.2 for the execution of the java based broker).

Both analyzers and managers are based on the *libidmef* C library [8], which is used to build and parse compliant IDMEF messages, and the xmlBlaster client C socket library [10], which supports access to xmlBlaster with asynchronous callbacks.

4 Conclusions

In this paper we presented an infrastructure to share alerts between the components of a prevention framework, that is targeted at detecting, as well as reacting to, distributed and coordinated attack scenarios, through the use of the publish/subscribe paradigm.

The information exchange between peers achieves a more complete view of the whole system, which is necessary to detect and react on the different actions of an attack, in an indirect manner.

We have also introduced and evaluated a concrete implementation of such mechanism based on xmlBlaster, an open source publish/subscribe message oriented middleware.

As future work we are considering the use of privacy mechanisms to address the exchanging of alerts in a pseudonymous manner, i.e. to provide the destination and origin information of alerts without violating the privacy of publishers and subscribers located on different domains.

Acknowledgments

The collaboration and discussions between Joaquín García, Fabien Autrel, and Frédéric Cuppens sharpen many of the arguments presented in this paper.

The work of Joaquín García and Joan Borrell is funded by the Spanish Government Commission CICYT, through its grant TIC2003-02041, and the Catalan Government Department DURSI, with its grant 2003FI-126. The work of Michael A. Jaeger is funded by Deutsche Telekom Stiftung. Gero Mühl is funded by Deutsche Telekom.

References

[1] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0 (third edition), w3c recommendation. `http://www.w3.org/TR/REC-xml/`, February 2004.

[2] J. Clark and S. DeRose. XML path language (XPath) 1.0, w3c recommendation. `http://www.w3.org/TR/xpath/`, November 1999.

[3] H. Debar, D. Curry, and B. Feinstein. Intrusion detection message exchange format data model and extensible markup language (xml) document type definition. Internet draft, January 2005.

[4] J. Garcia-Alfaro, F. Autrel, J. Borrell, S. Castillo, F. Cuppens, and G. Navarro. 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, October 2004. Springer-Verlag.

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

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

[7] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Spreinger-Verlag, 1992.

[8] A. C. Migus. IDMEF XML library version 0.7.3. `http://sourceforge.net/projects/lib-idmef/`, March 2004.

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

[10] M. Ruff. XmlBlaster: message oriented middleware. `http://xmlblaster.org/xmlBlaster/doc/whitepaper/whitepaper.html`, 2000.

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

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

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

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