Decentralized publish-subscribe system to prevent coordinated attacks via alert correlation

Joaquin Garcia Alfaro, Fabien Autrel, Joan Borrell, Sergio Castillo, Frédéric Cuppens, Guillermo Navarro

Decentralized Publish-Subscribe System to Prevent Coordinated Attacks via Alert Correlation

Joaquin Garcia[1], Fabien Autrel[2], Joan Borrell[1], Sergio Castillo[1], Frederic Cuppens[3], and Guillermo Navarro[1]

[1] Universitat Autònoma de Barcelona, 08193 Bellaterra, Spain
{jgarcia,jborrell,scastillo,gnavarro}@ccd.uab.es
[2] ONERA-CERT, 2 Av. E. Belin, 31055 Toulouse, France
fabien.autrel@enst-bretagne.fr
[3] GET/ENST-Bretagne, 35576 Cesson Sévigné, France
frederic.cuppens@enst-bretagne.fr

Abstract. We present in this paper a decentralized architecture to correlate alerts between cooperative nodes in a secure multicast infrastructure. The purpose of this architecture is to detect and prevent the use of network resources to perform coordinated attacks against third party networks. By means of a cooperative scheme based on message passing, the different nodes of this system will collaborate to detect its participation on a coordinated attack and will react to avoid it. An overview of the implementation of this architecture for GNU/Linux systems will demonstrate the practicability of the system.

Keywords: Intrusion Detection, Publish-Subscribe Systems, Alert Correlation

1 Introduction

The use of distributed and coordinated techniques is getting more common among the attacker community, since it opens the possibility to perform more complex tasks, such as coordinated port scans, distributed denial of service, etc. These techniques are also useful to make their detection more difficult and, normally, these attacks will not be detected by exclusively considering information from isolated sources of the network. Different events and specific information must be gathered from all of these sources and combined in order to identify the attack. Information such as suspicious connections, initiation of processes, addition of new files, sudden shifts in network traffic, etc., have to be considered.

In this paper, we present an intrusion detection system which provides a decentralized solution to prevent the use of network resources to perform coordinated attacks against third party networks. Our system includes a set of cooperative entities (prevention cells) which are lodged inside resources of the network. These entities collaborate to detect when the resources where they are lodged are becoming an active part of a coordinated attack. Prevention cells must be able to prevent the use of their associated resources (where they are lodged in) to finally avoid their participation on the detected attack. Thus, the main difference between our proposal and other related tools is that each node that lodges a prevention cell is expected to be the source of one of the

different steps of a coordinated attack, not its destination. Traditional technology that prevents against these attacks remains rooted in centralized or hierarchical techniques, presenting an easily-targeted single point of failure.

The rest of this paper is organized as follows. Section 2 presents some related work dedicated to the detection of distributed attacks, whose contributions and designs have been used as the starting point of this work. Our system is presented in Section 3 and the use of our system inside a real scenario is described in Section 4. A first implementation of the system is presented in Section 5.

2 Related Work

Currently, there are a great number of publications related to the design of detection systems that detect and prevent coordinated and distributed attacks. The major part of these works are conceived like centralized or hierarchical systems that usually present a set of problems associated with the saturation of the service offered by centralized or master domain analyzers. Centralized systems, such as DIDS [16], process their data in a central node despite their distributed data collection. Thus, these schemes are straightforward as they simply place the data at a central node and perform the computation there. On the other hand, hierarchical approaches, such as Emerald [14], have a layered structure where data is locally preprocessed and filtered. Although they mitigate some weaknesses present at centralized schemes, they still carry out bottleneck, scalability and fault tolerance vulnerabilities at the root level.

Alternative approaches, such as Sparta [11], propose the use of mobile agent technology to gather the pieces of evidence of an attack. The idea of distributing the detection process to different mobile agents, has some advantages regarding centralized and hierarchical approaches. For example, these schemes keep the whole system load relatively low and the consumption of the needed resources takes place only where the agents are running. Unfortunately, these systems present very simplistic designs and suffer from several limitations. In most of these approaches the use of agent technology and mobility is unnecessary and counterproductive.

Message passing designs, such as Quicksand [10], try to eliminate the need for dedicated elements. 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. In order to be able to detect coordinated and distributed attacks, the different entities have to collaborate on the intrusion detection activities and cooperate to perform a decentralized correlation algorithm. These architectures have the advantage that no single point of failure or bottlenecks are inherent in their design.

3 Prevention Cells System

In this section we present the design of a system whose main purpose is to detect and prevent coordinated attacks. By means of a set of cooperative entities which will be lodged inside the network, the system will avoid the use of network resources to perform coordinated attacks against third party networks. The aim of this system is not to detect incoming attacks against these entities, but to detect when these nodes are the source of one of the different steps of a coordinated attack to avoid it.

The design of our system has two main goals. The first goal is to obtain a modular architecture composed by a set of cooperative entities. These entities will collaborate to detect when the resources where they are lodged are becoming an active part of a coordinated attack against the network where they are located or against a third party network. Once an attack has been detected, they must be able to prevent the use of their associated resources to finally avoid their participation on the detected attack. The second goal is to achieve a complete independent relationship between the different components which form these cooperative entities. In this case, we will be able to distribute these components according to the needs of each resource we want to disarm.

The remainder of this section is organized as follows. First, we present the essential features of the communication architecture of this system and the model used to design it. Then, we show the elements which make up the different nodes of this architecture. Finally, we introduce the mechanisms used by the cooperative nodes to perform the correlation of alerts.

3.1 Multicast Communication Architecture

To achieve the first design goal listed above, a multicast architecture is proposed for the communication between the different cooperative entities. Through this architecture, each one of these entities, called prevention cell, will exchange a set of cooperative messages to collaborate in the decentralized detection process. To do that, we propose the use of a publish-subscribe model where each prevention cell will be able to produce and consume messages on the shared multicast bus.

According to [8], in a publish-subscribe system the different components will produce messages and announce (or publish) them on a shared bus. Other components may listen to (or be subscribed to) these messages. Once listened, they will be consumed by the components. Components can be objects, processes, servers, applications, tools or other kinds of system runtime entities. The messages, or events, exchanged between these components may be simple names or complex structures. The key feature of this model is that components do not know the name, or even the existence, of listeners that receive events that they announce. Thus, some immediate advantages in using this model for our proposal are the relatively easiness to add or remove components, as much as the introduction of new kind of messages, the registration of new listeners, and the modification of the set of announcers for a given message.

3.2 Prevention Cells

Taking into account the advantages of the publish-subscribe model discussed above, it is also useful to achieve the independence between components that we have announced as the second goal. Thus, we also propose the use of the publish-subscribe model for the relationship between the internal elements of each prevention cell. All these internal elements have been proposed according to the basic components of any IDS, that is, sensors, analyzers, managers and response units. The messages exchanged between these components will be three: *events* (between sensors and analyzers), *alerts* (between analyzers and managers), and *actions* (between managers and response units).

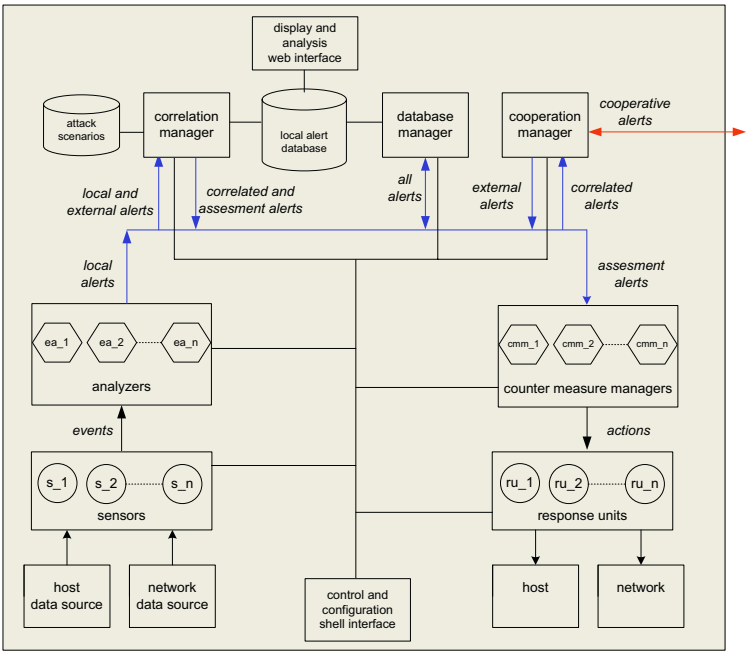

Fig. 1. Basic scheme of a prevention cell

These components, and the different messages exchanged between them, are shown in Figure 1.

- *Sensors*, which look for suspicious data over the host or over the network where they are installed and publish this information to a specific event scope. We propose the use of network and host based sensors.
- *Analyzers*, which listen to the events published by sensors, to perform a low level correlation process. Thus, these components will consume events and produce local alerts inside the prevention cell. We propose the use of misuse and anomaly based analyzers.
- *Correlation manager*, which listens for local and external alerts on their specific scopes and uses the data consumed against its associated coordinated attack scenarios. This component will perform a higher correlation process and will be involved in the relative part of the decentralized correlation process associated to the prevention cell where it is installed. It is also responsible for publishing correlated and assessment alerts.
- *Database manager*, which listens to all of the alert scopes to consume all the alerts produced inside and outside the prevention cell. Then, it will store all these alerts on the local database where it is installed.
- *Cooperation manager*, which listens for cooperative alerts published outside the prevention cell where it is installed and publishes external alerts inside the prevention cell. It also listens correlated alerts and publishes cooperative alerts outside the prevention cell.

– *Counter measure managers*, which listen for assessment alerts published by the correlation manager inside the prevention cell. These managers will be responsible for consuming the assessment alerts and transforming them to the correct actions that will be sent to the associated response units.
– *Response Units*, which take actions produced by their associated counter measure manager to initiate them. Each action is generated to prevent one of the different steps of the detected coordinated attack, and will be performed against the node where the prevention cell is installed. Like sensors, we also propose the use of network and host based response units.

3.3 Correlation of Alerts

The notion of alert correlation needs to be precisely defined since it has been presented in several articles but the definition differs from one article to another. Two main definitions have been given. The first one presents alert correlation as the process of aggregating attack detection alerts related to the same event. Alerts are aggregated in clusters of alerts [9, 1, 3] through the use of a similarity operator or function. This approach is called alert aggregation and fusion in [3]. The second definition presents attack detection alert correlation as the process of finding a set of alerts, organized into an attack scenario, into the stream of attack detection alerts generated by some IDS [13, 5, 4, 2].

In order to detect attack scenarios, each prevention cell includes a correlation manager that performs alert correlation by using the second definition introduced above. The chosen formalism is exposed in the following subsections.

Attack modelization. The attack process is modeled as a planning activity [4]. The intruder can use a set of actions. His goal is to find a subset of actions that can allow him to change the state of a system so that the attack objectives he has planned have been reached. In this final state the system security policy is infringed. The chosen approach and formalism is the same as [4]. Actions are represented by their pre and post conditions. Pre conditions correspond to the conditions the system state must satisfy to perform the action. Post conditions correspond to the effects on the system state of the action execution.

Scenario modelization. As exposed in [4] we do not need to explicit the scenario, we just have to model the actions composing the scenario. Then, correlation rules are generated from these models and used by the correlation engine to detect the scenario. Those correlation rules represent all the possible correlation between the actions available to the intruder.

Let us consider the scenario representation of a Mitnick attack. This attack tries to exploit the trust relationship between two computers to achieve an illegal remote access using the coordination of three techniques. First, a SYN flooding DoS attack to kept the trusted system from being able to transmit. Second, a TCP sequence prediction against the target system to obtain its following TCP sequence numbers. And third, an unauthorized remote login by spoofing the IP address of the trusted system (while it is in a mute state) and using the sequence number that the target system is expecting.

Action $syn\text{-}flood(A, H_1, n_s)$
Pre: $remote\text{-}access(A, H_1)$,
$send\text{-}multiple\text{-}tcp\text{-}syns(A, H_1, n_s)$
Post: $deny\text{-}of\text{-}service(H_1)$

Action $tcp\text{-}sequence\text{-}prediction(A, H_2, n)$
Pre: $remote\text{-}access(A, H_2)$,
$obtain(A, following\text{-}tcp\text{-}sequence(H_2, n))$
Post: $knows(A, following\text{-}tcp\text{-}sequence(H_2, n))$

Action $spoofed\text{-}remote\text{-}login(A, U, H_1, H_2, n)$
Pre: $remote\text{-}access(A, H_2)$,
$knows(A, following\text{-}tcp\text{-}sequence(H_2, n))$,
$deny\text{-}of\text{-}service(H_1)$,
$spoof\text{-}address(A, H_1, n, remote\text{-}login\text{-}connection(U, H_2))$
Post: $remote\text{-}login(A, U, H_2)$

Objective $illegal\text{-}remote\text{-}login(A, U, H_2)$
State: $remote\text{-}login(A, U, H_2)$
$not(authorized(remote\text{-}login(A, U, H_2)))$

Fig. 2. Modelling the Mitnick scenario

Action $undo\text{-}deny\text{-}of\text{-}service(A, H_1, n_s)$
Pre: $deny\text{-}of\text{-}service(H_1)$,
$send\text{-}multiple\text{-}tcp\text{-}resets(A, H_1, n_s)$
Post: $not(deny\text{-}of\text{-}service(H_1))$

Action $kill\text{-}remote\text{-}login(A, U, H_2)$
Pre: $remote\text{-}login(A, U, H_2)$
Post: $not(remote\text{-}login(A, U, H_2))$

Fig. 3. Counter measures for the Mitnick scenario

Figure 2 presents the models for each action that composes the scenario. The actions are represented using the LAMBDA language [6]. We must also modelize the attack objective for this scenario. The attack objective is modeled as a condition on the system state. Attack objective correlation rules are also generated to allow the correlation engine to correlate actions with attack objectives.

Counter measure management. Each prevention cell has its response units responsible for launching actions allowing the termination of ongoing scenarios. In order to detect when a counter measure must be launched, we use the anti-correlation mechanism defined in [2]. On the modelization point of view, the counter measures are not different from the models representing the set of actions available for the intruder. Actually a counter measure is an action C anti-correlated with another action A, i.e one of the predicates in its post-condition is correlated with the negation of one predicate in the pre-condition of action A.

Figure 3 presents the models for each action representing the available counter measures for the Mitnick scenario. The predicate $not(deny\text{-}of\text{-}service(H_1))$ in the post condition of action $undo\text{-}deny\text{-}of\text{-}service(A, H_1, n_s)$ is anti-correlated with the

predicate $deny\text{-}of\text{-}service(H_1)$. Also, the predicate $not(remote\text{-}login(A, U, H_2))$ of action $kill\text{-}remote\text{-}login(A, U, H_2)$ is anti-correlated with the predicate $remote\text{-}login(A, U, H_2)$ of attack objective $illegal\text{-}remote\text{-}login(A, U, H_2)$.

Detecting the scenario. The attack detection alert correlation mechanism allows to find a set of actions belonging to the same scenario and leading to an attack objective. However, we need a mechanism to be able to decide when to execute a counter measure once the scenario has been partially observed and that the next expected action can be blocked through an anti-correlated action.

This mechanism is provided by the correlation engine through the hypothesis generation mechanism [2]. Each time a new alert is received, the correlation engine finds a set of action models that can be correlated in order to form a scenario leading to an attack objective. This set of hypothesis is then instantiated into a set of virtual alerts. The correlation engine then looks for actions models that can be anticorrelated with the virtual actions. This set of anti-correlated actions forms the set of counter measures available for the hypothesis represented by the partially observed scenario.

A counter measure C for an action A must be executed before the action A occurs. If the correlation engine receives an alert for which a correlated virtual alert exists, it will notify the response units to execute the associated counter measure. For example, if the correlation engine receives an alert corresponding to the execution of $syn\text{-}flood(A, H_1, n_s)$, it will generate a virtual alert corresponding to $spoofed\text{-}remote\text{-}login(A, U, H_1, H_2, n)$. Since $undo\text{-}deny\text{-}of\text{-}service(A, H_1, n_s)$ is anti-correlated with $spoofed\text{-}remote\text{-}login(A, U, H_1, H_2, n)$ and that an occurrence of action $syn\text{-}flood(A, H_1, n_s)$ has been observed, the correlation engine notify the response units to execute $undo\text{-}deny\text{-}of\text{-}service(A, H_1, n_s)$ with the parameters extracted from the $syn\text{-}flood(A, H_1, n_s)$ alert.

4 Sample Prevention of a Coordinated Attack

In this section we will discuss the prevention of the Mitnick attack scenario introduced above by using the prevention cells system presented in this paper. Although the Mitnick attack is several years old, it is an excellent example to show how the components of our architecture handle a possible coordinated attack.

The correlation and anti-correlation graph for this coordinated attack is shown in Figure 4. In the first step of this model, *A* (the agent that performs the whole attack) floods a given host H_1. In the second step, *A* sends a TCP sequence prediction attack against host H_2 to obtain its following TCP sequence numbers. Then, by using these TCP sequence numbers, *A* starts a spoofed remote login session to the host H_2 as it would come from host H_1. Since H_1 is in a mute state, H_2 will not receive the RST packet to close this connection. If this third step has success, *A* will establish an illegal remote login session as user root to system H_2.

The model of Figure 4 also proposes two counter measures to prevent the coordinated attack. First, as soon as the host which is performing the SYN flooding DoS against H_1 would detect it, it will neutralize the attack by sending the same number of RST TCP packets to H_1 as SYN TCP packets it has send. And second, as soon as the

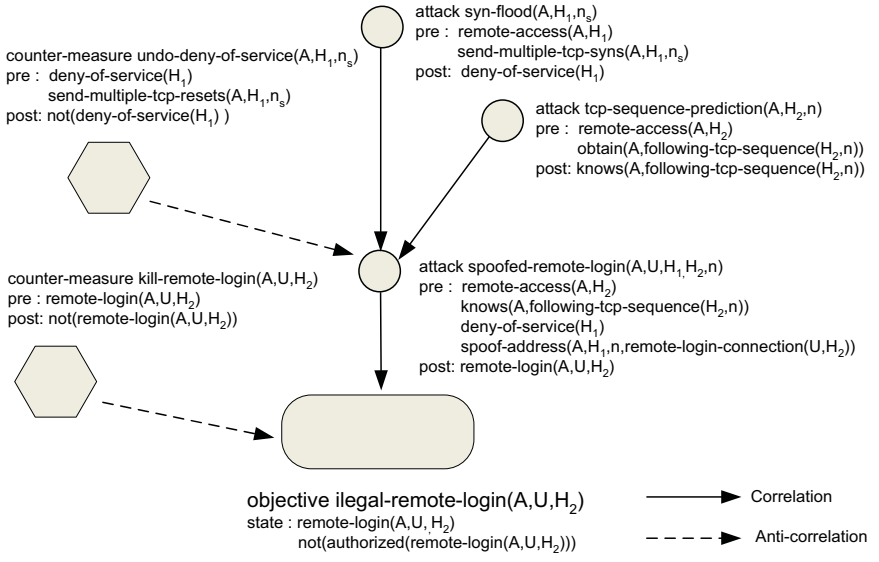

Fig. 4. Correlation and anti-correlation graph for the Mitnick attack

host where the third action (the spoofed remote login against H_2) is detected, it will kill the remote login process to avoid the illegal user access.

To show how the components of our architecture would handle the coordinated attack model described in Figure 4, we consider the sequence of alerts described in Figure 5. We assume that an attacker targeting the network `victim.org` will use resources from another corporate network to perform the coordinated attack. This corporate network is protected with our prevention cells system. The different parts of the attack are detected by three protection cells, named *pcell1*, *pcell2*, and *pcell3* (see Figure 5). For each prevention cell we show the most relevant IDMEF compliant alerts [7] published and consumed by components of the cell. We have simplified quite a lot the information and format of each alert for clarity reasons. We also assume the correlation and anti-correlation graph for the Mitnick attack is not stored in the attack scenario database of the other prevention cells for clarity reasons. Each alert is denoted with ordered identifiers t_i, which correspond to the *DetectionTime* field of the IDMEF alert format.

The first indication of the attack is detected by sensors from *pcell1*. The sensors detect the SYN flooding DoS, and generate the local alert t_1. This alert is received by the correlation engine of the cell, which in turn generates the assessment alert t_2 informing that the DoS needs to be neutralized. The assessment alert is observed by the counter measure manager of the prevention cell, which will signal a response unit to block the DoS. Then, by means of the cooperative manager, the prevention cell will send the cooperation alert t_3 to the other prevention cells of the system. This alert is received by the other prevention cells as an external alert notifying that a SYN flooding DoS attack against `n1.victim.org` has been detected and prevented in *pcell1*.

At this point, the prevention cell *pcell1* has prevented the DoS attack against the host `n1.victim.org`, which is the first step of the illegal remote login scenario.

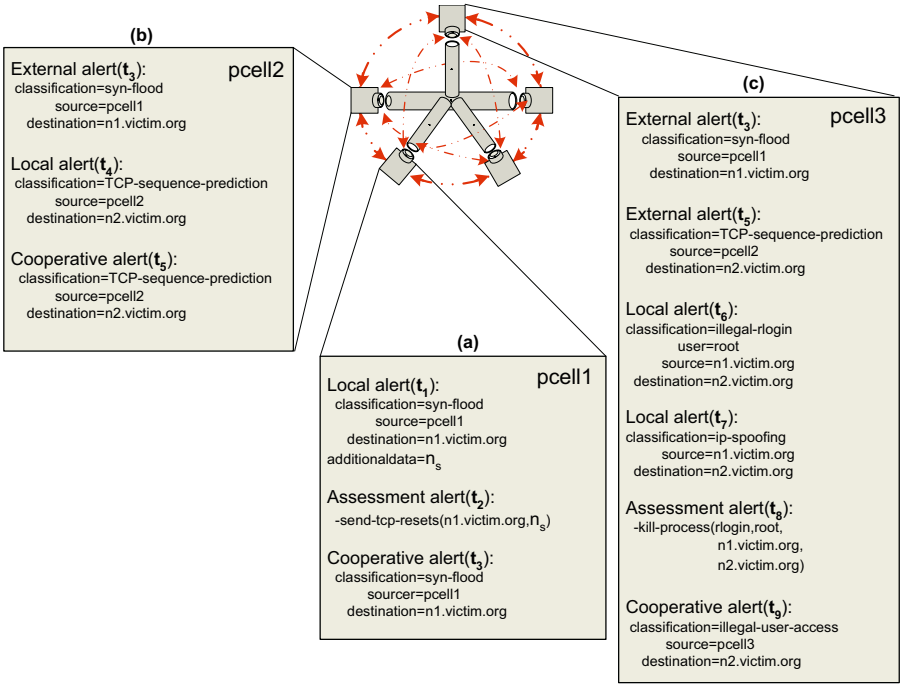

Fig. 5. Sequence of alerts raised inside each prevention cell

Nevertheless, we cannot ensure that the whole attack is frustrated. It is reasonable to assume that the attacker will try to use another resource not covered by the prevention cells system to commit the final attack. Thus, it is important to try to detect all the steps of the attack and to be able to correlate them in order to identify the whole attack.

The next step of the attack, a TCP sequence prediction attack against `n2.victim.org`, is detected by sensors of *pcell2* that publish it as the local alert t_4. The correlation manager of *pcell2* consumes the alert and produces a corresponding cooperative alert t_5. This alert is sent to the other prevention cells, making them aware that the TCP sequence prediction attack has been detected in *pcell2*.

Finally, the coordinated attack detection will be completed when the attacker tries the spoofed remote login on the target system (`n2.victim.org`) from the host that lodges the prevention cell *pcell3*. The sensors from *pcell3* detect a spoofed rlogin connection against the host `n2.victim.org` producing local alerts t_6 and t_7. These alerts, together with the external alerts t_3 and t_5, are correlated by the correlation engine of *pcell3*, resulting in the detection of the coordinated illegal user access. This detection step will produce the assessment alert t_8 to kill the remote login process executed. Furthermore, it also involves the production of the cooperative alert t_9 to notify the other prevention cells that the illegal remote login has been detected from nodes *pcell1*, *pcell2*, and *pcell3*, against the target `n2.victim.org` and its trusted system `n1.victim.org`.

5 Current Development

This section presents a brief overview of a platform which implements our publish-subscribe system and that deploys all the basic components proposed in this paper. This platform is currently being developed for GNU/Linux systems in C and C++. The combination of free high-quality documentation, development and network solutions provided by GNU/Linux operating systems eased the analysis of requirements and the development of this platform.

The main difference between our proposed system and other related tools, is that the node that lodges each prevention cell is expected to be the source of one of the different steps of a coordinated attack, not its destination. This fact implies some considerations in the analysis of requirements for both sensors and response units. First, the number of sensors and response units must be enough representative to detect and react against the different steps of the attack scenarios the system knows. Second, both analyzers and counter measure managers need a fast communication with sensors and response units to be able to gather or to provide events and actions.

Sensors and response units. In order to fulfill the requirements showed above, we started the development of this platform working on the design and implementation of a set of sensors and response units embedded in the Linux 2.4.x series as kernel modules. Even though, third party sensors and third party response units could easily be integrated in our platform.

The use of sensors and response units embedded as kernel modules involves a set of advantages. First, the privileged location of the modules within the operating system allows the platform to have access to all the necessary information in an efficient and trustworthy way. Second, the load produced by the exchange of information from kernel space to user space is reduced, transferring information only at the moment that an event is produced. As a result of this previous point, the throughput of analyzed patterns (e.g. network datagrams or executed commands) is maximized.

The implementation of the proposed network sensors and response units is based on the netfilter subsystem, a framework for packet manipulation that enables packet filtering, network address translation and other packet mangling on Linux 2.4.x and upper series. On the other hand, the implementation of the proposed host sensors and host response units is based on the interception of some system calls. In this manner, is possible to obtain useful information in the search process of illicit or suspicious activities and provide the needed mechanisms to prevent the associated action related with the step of the attack to avoid.

Communication of events and actions. The complexity of the analyzers and counter measure managers, as well as the limitation that supposes to work in a kernel scope, entails to design them like a daemon processes in user space. Thus, a specific communication mechanisms between kernel space and user space is needed. Among the diverse alternatives for performing this communication, we have chosen the use of *netlink sockets* to bind the proposed sensors and response units with the analyzers and counter measure managers. Netlink sockets is a Linux specific mechanism that provides connectionless and asynchronous bidirectional communication links. Although the use of netlink sockets is focused for implementing protocols of IP services, this mechanism

can also be used as a standard interface to perform a communication link between the kernel modules and user space processes. Netlink sockets allows us to use the well known primitives from the socket treatment, providing us transparency with the buffering mechanisms.

Analyzers and managers. The implementation of analyzers and managers is based on a plug-in mechanism to facilitate the development and the maintenance of the different features that these components will offer. Thus, through the use of netlink sockets, both the event watcher analyzer and the counter measure manager will consume or produce information. But, to generate this information, or to manage it, different plug-ins will be enabled or disabled. Some of these plug-ins will be launched in a multi-threading fashion.

The analyzer in charge for obtaining the events produced by the sensors, for example, will launch the different plug-ins to handle the events received from the sensors using this multi-threading mechanism. This way, it is possible to parallelize the gathering of the different events produced by the set of sensors. Other plug-ins, such as the one responsible for sending actions to the response units, the one responsible for managing external alerts and transform them to internal alerts, etc. will not need the use of this multi-threading mechanism to perform its work.

One of the plug-ins which will be present on all the analyzers and managers is the responsible for generating, parsing and communicating the IDMEF compliant alerts [7]. This plug-in is based on the library libidmef, an ANSI C library compliant with the IDMEF format and uses Libxml to build and parse IDMEF messages. The use of libidmef, besides to provide a free and easy library to develop our components, also makes it easy for third party managers and analyzers to communicate with the different components of our system.

Communication of alerts. The communication between the analyzers and managers, as much inside of each prevention cell as between the other prevention cells of our architecture, will be performed by using the Elvin publish-subscribe system [15]. Elvin is a network communication product that provides a simple, flexible and secure communication infrastructure. To be able to use the infrastructure offered by the Elvin publish-subscribe system, both the analyzers and the managers of our implementation have been developed by using libelvin and e4xx, two portable C and C++ libraries for the Elvin client protocol. On the other hand, each host with a prevention cell lodged inside will run an Elvin server to route all the alerts published inside each prevention cell.

To share the cooperative alerts produced by the different prevention cells in a secure multicast fashion, we use the federation and reliable local-area multicast protocol provided by Elvin, and other interesting features offered by this publish-subscribe system, such as fail-over and cryptographic settings. By using SSL at the transport layer, for example, we guarantee the confidentiality, integrity and authenticity of the cooperative alerts communicated between each prevention cell.

6 Conclusions and Further Work

We have presented the design of a publish-subscribe system for the detection and prevention of coordinated attacks from network resources. This system uses multicast com-

munication between different entities to avoid their participation in a coordinated attack against third party networks or even the local network. Our approach can be merged into any existing corporate network becoming a common framework for the prevention of coordinated attacks from these network environments. We have also outlined in this paper how our system can detect and prevent the Mitnick attack, exploiting the distribution and coordination of the system components.

We have briefly introduced the implementation of a platform, which is currently being developed and which implements the major part of the components of the architecture previously proposed for GNU/Linux systems. Although the detection and reaction components of this platform (sensors and response units implemented as Linux modules) are at this moment developed only for Linux 2.4, we plan to upgrade them to Linux 2.6 in very near future.

As further work, we will study the possibility to incorporate other alert correlation contributions in our work, such as the formal data model proposed in M2D2 [12] . We will also make a more in-depth study of the IDMEF format [7] to solve unnecessary duplicated calculus inside each prevention cell. Finally, we will study and incorporate current intrusion tolerant mechanisms to make our system more reliable when the host that lodges a prevention cells is infected.

Acknowledgments

The work of J. Garcia, J. Borrell, S. Castillo and G. Navarro has been partially funded by the Spanish Government Commission CICYT, through its grant TIC2003-02041, and the Catalan Government Department DURSI, with its grant 2001SGR-219.

References

1. D. Andersson, M. Fong, and A. Valdes. Heterogeneous sensor correlation: A case study of live traffic analysis. In *3rd Annual Information Assurance Workshop*, United States Military Academy, West Point, New York, USA, June 2002.
2. S. Benferhat, F. Autrel, and F. Cuppens. Enhanced correlation in an intrusion detection process. In *Mathematical Methods, Models and Architecture for Computer Network Security (MMM-ACNS 2003)*, St Petersburg, Russia, September 2003.
3. F. Cuppens. Managing alerts in a multi-intrusion detection environment. In *17th Annual Computer Security Applications Conference (ACSAC'01)*, New Orleans, Lousiana, December 2001.
4. F. Cuppens, F. Autrel, A. Miège, and S. Benferhat. Recognizing malicious intention in an intrusion detection process. In *Second International Conference on Hybrid Intelligent Systems (HIS'2002)*, Santiago, Chile, October 2002.
5. F. Cuppens and A. Miège. Alert correlation in a cooperative intrusion detection framework. In *IEEE Symposium on Security and Privacy*, Oakland, USA, 2002.
6. F. Cuppens and R. Ortalo. LAMBDA: A language to model a database for detection of attacks. In *Third International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, Toulouse, France, 2000.
7. D. Curry, H. Debar, and B. Feinstein. Intrusion detection message exchange format data model and extensible markup language (xml) document type definition. Internet draft, January 2004.

8. D. Garlan, S. Khersonsky, and J. S. Kim. Model checking publish-subscribe systems. In *Proceedings of the 10th International SPIN Workshop*, Portland, Oregon, USA, May, 2003.
9. K. Julich. Using root cause analysis to handle intrusion detection alarms. *ACM journal name*, 2:111–136, October 2002.
10. C. Kruegel. *Network Alertness - Towards an adaptive, collaborating Intrusion Detection System.* PhD thesis, Technical University of Vienna, June 2002.
11. C. Kruegel and T. Toth. Flexible, mobile agent based intrusion detection for dynamic networks. In *European Wireless*, Italy, February 2002.
12. B. Morin, L. Mé, H. Debar, and M. Ducassé. M2D2: a formal data model for intrusion alarm correlation. In *Proceedings of the 5th Recent Advances in Intrusion Detection (RAID2002)*, Zurich, Switzerland, October 2002.
13. P. Ning, Y. Cui, and D. S. Reeves. Analyzing intensive intrusion alerts via correlation. In *Fifth International Symposium on Recent Advances in Intrusion Detection (RAID2002)*, pages 74–94, Zurich, Switzerland, October 2002.
14. P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, pages 353–365, October 1997.
15. B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of the third annual technical conference of AUUG 1997*, pages 243–255, Brisbane, September 1997.
16. 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.