Preventing Coordinated Attacks via Alert Correlation

J. Garcia*, F. Autrel†, J. Borrell*, Y. Bouzida‡,
S. Castillo*, F. Cuppens‡ and G. Navarro*

*UCCD-UAB, 08193 Bellaterra (Spain)
Email: {jgarcia,jborrell,scastillo,gnavarro}@ccd.uab.es

†ONERA-CERT, 31055 Toulouse (France)
Email: fabien.autrel@enst-bretagne.fr

‡GET-ENST-Bretagne, 35576 Cesson Sévigné (France)
Email: {yacine.bouzida,frederic.cuppens}@enst-bretagne.fr

***Abstract*— When attackers gain access to enterprise or corporate networks by compromising authorized users, computers, or applications, the network and its resources can be used to perform distributed and coordinated attacks against third party networks, or even on computers on the network itself. We are working on a decentralized scheme to share alerts in a secure multicast infrastructure to detect and prevent these kind of attacks. In this paper we present a collaborative framework that performs coordinated attack prevention. The detection and prevention process itself is done by a set of collaborative entities that correlate and assemble the pieces of evidence scattered over the different network resources. We also provide an example of how our system can detect and prevent a coordinated attack to demonstrate the practicability of the system.**

***Index Terms*— Intrusion Detection Systems, Publish-Subscribe Systems, Alert Correlation.**

I. Introduction

Despite the advances in network security technology, such as perimeter firewalls, authentication mechanisms and intrusion detection systems, networked systems have never been more vulnerable than today. The proliferation of Internet access to every network device, the increased mobility of these devices, and the introduction of network-enabled applications have rendered traditional network-based security infrastructures vulnerable to a new generation of attacks. Generally, these attacks start with an intrusion to some corporate network through a vulnerable resource and then launching further actions on the network itself or against third party networks. Once harmless hosts and devices have been compromised, they will become active parts in the deployment of new attacks against other networks if the administrator in charge for these resources cannot effectively disarm them.

The use of distributed and coordinated techniques in these kind of attacks 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 (DDoS), etc. These techniques are also useful to make their detection more difficult and, normally, these attacks will not be detected by solely considering information from isolated sources of the network. Different events and specific information must be gathered from all 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.

According to [7], we can define the term *attack* as a combination of actions performed by a malicious adversary to violate the security policy of a target computer system or a network domain. Therefore, we can define the *attack detection process* as the sequence of elementary actions that should be performed in order to identify and respond to an attack. An *intrusion detection system* (IDS) is the most important component in performing this process. As mentioned in [10], an IDS has to fulfill the requirements of accuracy (it must not confuse a legitimate action with an intrusion), performance (its performance must be enough to carry out real-time intrusion detection), completeness (it should not fail to detect an intrusion), fault tolerance (the IDS must itself be resistant to attacks) and scalability (it must be able to process the worst-case number of events without dropping information).

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 (called 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. The main difference between our proposal and other related work 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.

The rest of this paper is organized as follows. Section II presents some related work on the detection of distributed attacks. Our system is presented in Section III and its alert correlation mechanism is introduced in Section IV. The utilization of our system inside a real scenario is described in Section V. Finally, conclusions and further work are placed in the last section.

II. Related Work

Currently, there is a great number of publications related to the design of systems that detect and prevent coordinated and distributed attacks. The major part of them are designed as 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.

As shown in [2], centralized systems, such as DIDS [22], and NADIR [14], 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 GrIDS [23], Emerald [20], AAFID [2], and NetSTAT [25], 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 problems and fault tolerance vulnerabilities at the root level.

In contrast to these traditional architectures, alternative approaches such as Micael [9], IDA [1], Sparta [17], and MAIDS [13], propose the use of mobile agent technology to gather the pieces of evidence of an attack (which are scattered over arbitrary locations). 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. Furthermore, agents are also able to react very quickly when an intrusion has been discovered.

Mobile agent systems and mobile code may seem to be a promising technology to implement decentralized architectures for the detection of coordinated attacks, but the current systems present very simplistic designs and suffer from several limitations. For instance, in most approaches the use of agent technology and mobility is unnecessary and counterproductive. According to [16], mobile agents are used in these designs simply as data containers, a task that can be performed more efficiently by using a simple message passing. Furthermore, mobile agents introduce additional security risks and cause a performance penalty without providing any clear advantage. None of the proposals based on mobile agent technology seem to have a definitive implementation or any industrial application.

Some message passing designs, such as Quicksand [16] and Indra [15], try to eliminate the need for dedicated elements by introducing a message passing infrastructure. 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.

III. 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 entities which will be lodged inside the network, the system will prevent 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 several steps of a coordinated attack and to avoid it.

The design of our system has two main goals. The first one is to obtain a modular architecture composed of 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 they are located at, 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 have a complete uncoupled relationship between the different components that are these cooperative entities. Having accomplished this, we will be able to distribute these components according to the needs of each resource we want to disarm.

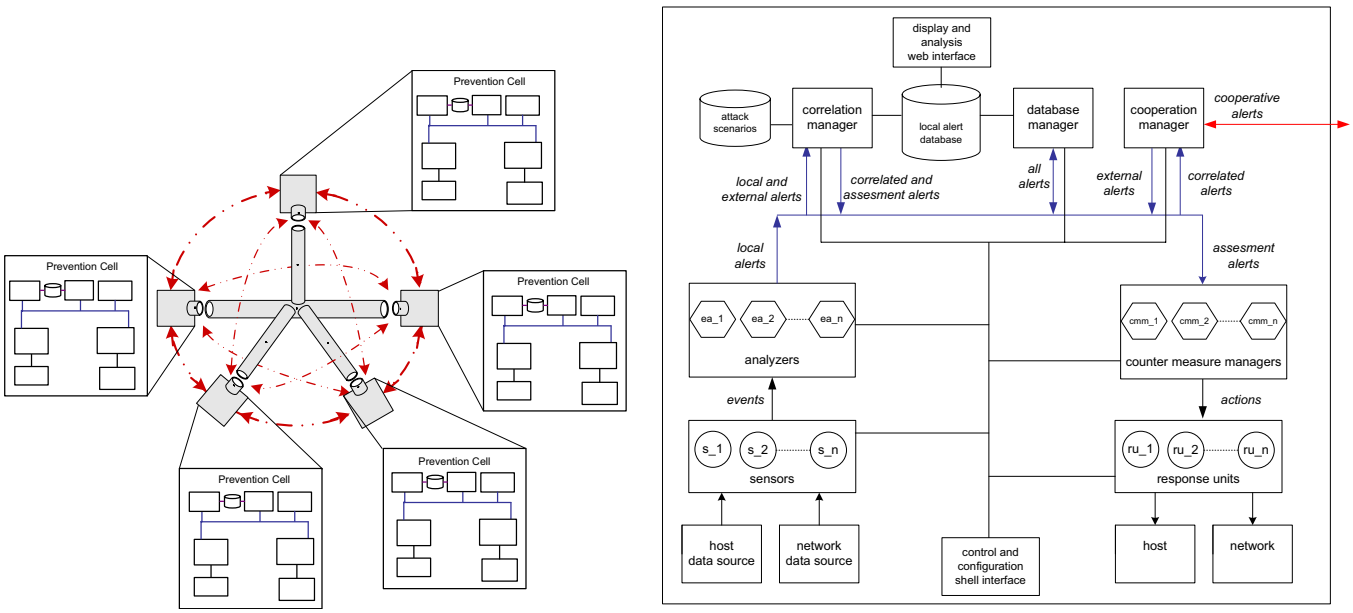
The remainder of this section is organized as follows. First, we present the essential features of the communication architecture of this system and the model used to design it. Then, we describe the elements that make up the different nodes of this architecture.

A. Multicast Communication Architecture

To achieve the first design goal listed above, a multicast architecture is proposed for the communication between the cooperative entities. Through this multicast communication architecture, each one of these entities, called prevention cells, will exchange a set of cooperative messages to collaborate in the decentralized detection process (Figure 1(a)). This architecture must also provide security mechanisms to avoid communication attacks and permit the identification of the different components (like the security mechanisms of the multicast infrastructure introduced in Section VI-D). To do that, we propose the use of a publish-subscribe model.

According to [12], a publish-subscribe system consists of brokers and clients that are connected to brokers. The brokers themselves form the infrastructure 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 this notification matches. If so, the message is delivered to this client.

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 publish. Some other advantages in using a publish-subscribe model for our proposal are the easy implementation of the add and remove operations for components, as much as the introduction of new kind of notifications, the registration of new listeners, and the modification of the set of publishers for a given type of notification.



(a) Message passing architecture based on prevention cells

(b) Basic scheme of a prevention cell

Fig. 1. Collaborative architecture based on prevention cells

B. Prevention Cells

Taking into account the advantages of the publish-subscribe model discussed above, this model 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. By using this model, all of them will be able to produce and consume messages on a secure shared bus.

The internal elements of each prevention cell 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 are three: *events* (between sensors and analyzers), *alerts* (between analyzers and managers), and *actions* (between managers and response units). These components, and the different messages exchanged between them (Figure 1(b)), are described below:

- *Sensors*, that look for suspicious data on the host or over the network where they are installed and publish this information to a specific event scope (where associated analyzers can subscribe). We propose the use of network based sensors and host based sensors.
- *Analyzers*, that 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. After that, they will publish these alerts at the corresponding scope (the local alert scope).
  We propose the use of misuse based analyzers, with a priori knowledge of sequences and activities of different attacks, and the use of anomaly based analyzers to identify malicious activity comparing the events listened against the representation of normal activities.
- *Correlation manager*, that listens for local and external alerts on their specific scopes and uses the data consumed against its associated coordinated attack scenarios. It will perform a higher correlation process and will be involved in the relative part of the correlation process explained in Section IV. It is also responsible for publishing correlated and assessment alerts.
- *Database manager*, that listens to all of the alert scopes to consume all the alerts produced inside and outside the prevention cell. It will store all these alerts on the local database where it is installed.
- *Cooperation manager*, that listens for cooperative alerts published outside the prevention cell where it is installed and publishes external alerts inside the prevention cell. Furthermore, it also subscribes to correlated alerts and publishes cooperative alerts outside the prevention cell.
- *Counter measure managers*, that 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 into the correct actions which will be sent to the associated response units.
- *Response Units*, that 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. We propose the use of network and host based response units.

IV. Correlation of Alerts

Correlating information held by multiple intrusion detection systems is an approach that has been discussed in several papers. However the goal aimed by those approaches are different and need to be explained.

With the rise of cooperative or distributed intrusion detection frameworks, the problem of reasoning on information coming from multiple sources spread across the monitored system is very important. Correlating this information allows to fulfill different goals, such as information redundancy and scenario detection.

The notion of alert correlation as the process of aggregating alerts related to the same event has been studied in [11], [24], and [4]. They define a similarity relationship between alert attributes to aggregate alerts. The second main approach of alert correlation as the process of detecting scenarios of alerts has been discussed in [19], [5], and [3]. In our proposal we use the latter approach, introducing the notion of alert correlation as the process of finding a set of alerts in the stream of intrusion detection alerts organized into a scenario. Our formalism is explained below.

A. Modeling Actions and Objectives

From the attacker point of view, the attack process can be seen as a planning activity [5]. The intruder can have some knowledge of the system he wants to attack, probably knowing the vulnerabilities present or software and hardware used. If the attacker has a limited knowledge about the targeted system, he can try to gather information by executing actions such as ports scans or using other vulnerability detection tools. Once the attacker has sufficient knowledge of the system to attack, he can define a set of reachable attack objectives.

From the point of view of the victim, those attack objectives constitute a violation of the security policy. In order to reach those attack objectives, the attacker select a set of actions constituting one or multiple scenarios of actions. Finally, from the detection point of view, we want to detect the coordinated attack by constructing scenarios of alerts corresponding to the scenarios of actions executed by the attacker. Hence, we have to model the set of actions available for the attacker and the set of attack objectives. Since we want to react to the detection of ongoing scenarios, we have to model the set of available counter measures.

We use the LAMBDA language [7] to model the actions of the coordinated attacks. LAMBDA provides a logical and generic description of actions, but we use it to model as well the attack objectives and the counter measures. A LAMBDA description of an action is composed mainly of the following attributes:

- *pre-condition*: defines the state of the system needed in order to achieve the action.
- *post-condition*: defines the state of the system after the execution of the action.

Let us consider the modeling of the *BIND Birthday Attack*. This coordinated attack tries to perform a DNS cache poisoning by sending a sufficient number of queries to a vulnerable DNS server based on the BIND software, while sending an equal number of false replies at the same time. A reason for the generation of multiple queries for the same domain name at the same time, could be an attacker trying to hit the needed transaction ID to perform a DNS cache poisoning. Since the transaction ID function is a pseudo-random function, we can supply the brute-force birthday attack based on the birthday paradox.

This attack will result in the storage of an illegal recursive query using the coordination of three techniques. First, a DoS attack to keep the authoritative DNS server from being able to reply. Second, a flooding of queries to an ISP DNS server asking for the IP address of the domain name to be hijacked. And third, a second flooding with the same number of replies formulated by spoofing the IP address of the authoritative DNS server (this way it looks like if these replies were sent from the legitimate nameserver). The attacker avoids the authoritative reply by the first action (the denial of service). If the attack is successful, the targeted ISP DNS will cache the spoofed record for the time indicated in the TTL section of the reply. At this point, the attack is over, but the effect persists for the time the ISP holds the phony record in its nameserver cache. The victim at the ISP is exposed to the spoofed information any time it makes a query for the domain name in question.

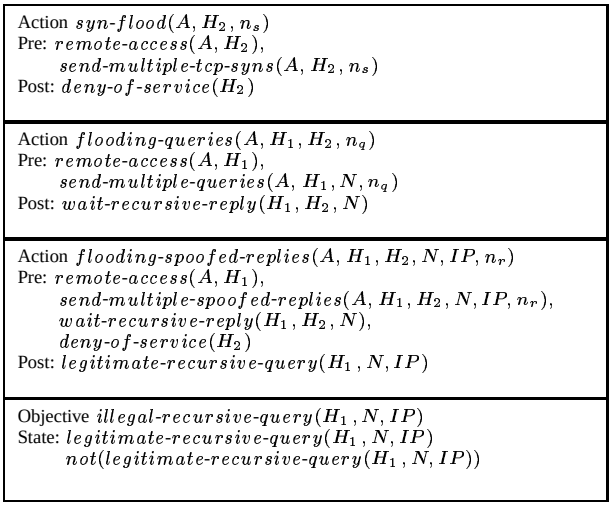


Fig. 2. Modeling the BIND Birthday Attack objective and actions

Figure 2 presents the models for each action that composes the BIND Birthday Attack scenario represented using the LAMBDA language. We also model the attack objective for this scenario as a condition on the system state.

B. Detecting Scenarios

In order to detect the coordinated attack scenario, we use the notion of correlation as the process of finding a set of alerts into the stream of alerts organized into a scenario. To do that, the correlation engine will perform action correlation and alert correlation:

- *Action Correlation* - Two actions A and B are correlated when the realization of A has a positive influence on the realization of B (given that A occurred before B). More formally, if $post(A)$ is the set of post-conditions

of action A and $pre(B)$ is the set of pre-conditions of action B, we say that A and B are *directly correlated* if the following conditions are satisfied:

$\exists E_a$ and E_b such that:

- $(E_a \in post(A) \wedge E_b \in pre(B))$ or $(not(E_a) \in post(A) \wedge not(E_b) \in pre(B))$
- E_a and E_b are unifiable through a most general unifier θ.

Similarly we define the notion of correlation between an action and an attack objective. In this case we correlate the post-condition of an action and the state condition of an objective. The attack objective is modeled as a condition on the system state (Figure 2).

- *Alert Correlation* - Once all the actions available for the attacker have been modeled, we can generate the set of unifiers between all the actions. This generation is done off-line. When an alert is received, we have to bind this alert to an action model and then check for a unifier between the new alert and the already received alerts.

  This set of unifiers is also used to anticipate the possible actions we may see after having observed the beginning of a scenario. Those hypothetic observations are called *virtual actions.*

C. Reacting on Detected Scenarios

Detecting the coordinated attack scenario is interesting but it does not prevent the attacker from reaching his objective. Therefore, 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:

- *Anti-correlation* - Two actions A and B are anti-correlated when the realization of A has a negative influence on the realization of B (given that A occurred before B). More formally, if $post(A)$ is the set of post-conditions of action A and $pre(B)$ is the set of pre-conditions of action B, we say that A and B are *directly anti-correlated* if the following conditions are satisfied:

  $\exists E_a$ and E_b such that:
  - $(not(E_a) \in post(A) \wedge E_b \in pre(B))$ or $(E_a \in post(A) \wedge not(E_b) \in pre(B))$
  - E_a and E_b are unifiable through a most general unifier θ.

From the modeling point of view, the models for counter measures are not different from the ones 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. This mechanism is provided by the correlation engine through the use of the hypothesis generation mechanism [3]. 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 anti-correlated with the virtual actions. This set of anti-correlated actions becomes the set of counter measures available for the hypothesis represented by the partially observed 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 $block\text{-}spoofed\text{-}connection(A, H_1, H_2)$
Pre: $spoofed\text{-}connection(A, H_1, H_2)$
Post: $not(spoofed\text{-}connection(A, H_1, H_2))$

Fig. 3. Modeling the BIND Birthday Attack counter measures

Figure 3 presents the models for each action representing the available counter measures for the BIND Birthday Attack 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 $deny\text{-}of\text{-}service(H_1)$. Similarly, the predicate $not(spoofed\text{-}connection(A, H_1, H_2))$ of action $block\text{-}spoofed\text{-}connection(A, H_1, H_2)$ is anti-correlated with the predicate $spoofed\text{-}connection(A, H_1, H_2)$ of attack objective $illegal\text{-}recursive\text{-}query(H_1, N, IP)$.

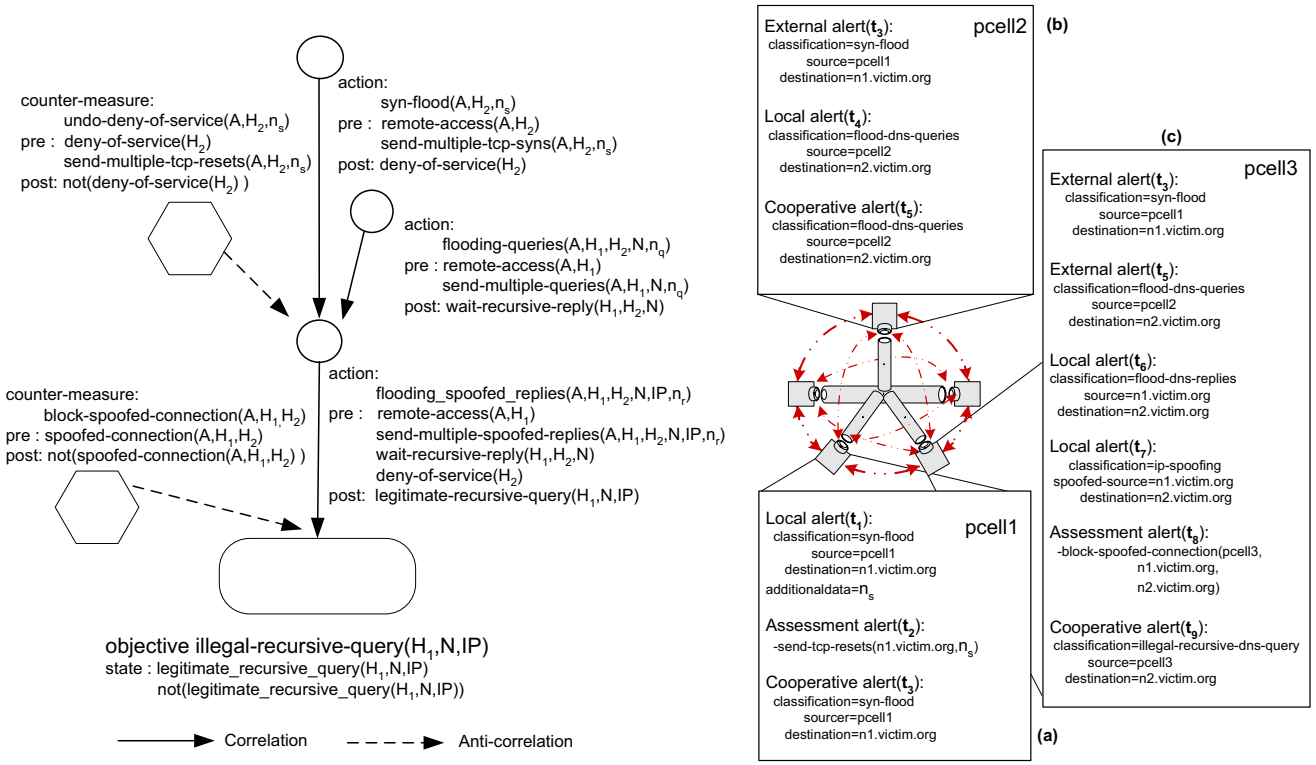
V. Preventing the BIND Birthday Attack

In this section we will discuss the prevention of the BIND Birthday Attack scenario introduced above by using the prevention cells system presented in this paper. This attack is a good example to demonstrate how the components of our architecture handle a possible coordinated attack.

The correlation and anti-correlation graph [6] for this coordinated attack is shown in Figure 4(a). In the first step of this model, A (the agent that performs the whole attack) floods a given host H_2. In the second step, A sends a flood of DNS queries to host H_1 to achieve, that the server process on this host will launch a recursive query to discover the IP address associated to the name N. Then, A starts flooding false recursive replies spoofing the IP address of H_2. Since H_2 is in a mute state, H_1 will never receive the authoritative reply. If one of the false replies has succeeded, H_1 will store the faked information in its cache.

The model of Figure 4(a) proposes two counter measures to prevent the coordinated attack. First, as soon as the host which is performing the SYN flooding DoS against H_2 would detect it, it will neutralize the attack by sending the same number of RST TCP packets to H_2 as SYN TCP packets having received. Second, as soon as the host where the third action (the flooding of spoofed replies to H_1) is detected, it blocks these spoofed connections.

To show how the components of our architecture would handle the coordinated attack model described in Figure 4(a), we consider the sequence of alerts described in Figure 4(b). We assume that an attacker targeting the network `victim.org`



(a) Correlation graph for the BIND Birthday Attack

(b) Sequence of alerts raised inside each prevention cell

Fig. 4. Preventing the BIND Birthday Attack by using the prevention cells system

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 4(b)). For each prevention cell we show the most relevant IDMEF compliant alerts [8] published and consumed by components of the cell. We have simplified quite a lot the information and format of each alert for clarity reasons. For the same reason we assume the correlation and anti-correlation graph for the BIND Birthday Attack is not stored in the attack scenario database of the other prevention cells. 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 recursive DNS query scenario. 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 flooding of DNS queries 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 flooding or DNS queries has been detected in *pcell2*.

Finally, the coordinated attack detection will be completed when the attacker tries the flooding of spoofed replies on the target system (`n2.victim.org`) from the host that lodges the prevention cell *pcell3*. The sensors from *pcell3* detect this flooding and produce the 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 recursive DNS query. This detection step will produce the assessment alert t_8 to block the flooding of spoofed connections. Furthermore, it also involves the production of the cooperative alert t_9 to notify the system that the illegal recursive DNS query scenario has been detected.

VI. Current Development

This section presents a brief overview of an implementation of our prevention system and that deploys all the basic components proposed in this paper. This platform has been developed for GNU/Linux systems in C and C++. Our implementation has been tested on different versions of Linux 2.4.x series and on the versions 2.9.x and 3.x of GNU's gcc compiler. 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. Below, we introduce the main components of our prototype.

A. Sensors and Response Units

Our prototype started with 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 implementation of the 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.

At this time, we have developed the following network based sensors and response units: a sensor to detect stealth scanning (*syns_s*), a sensor to detect IP spoofing (*spoof_s*), a sensor to detect buffer overflows on the packet payload (*bof_s*), a sensor to detect TCP connection establishments that will be used to infer connection chains (*conn_s*), three sensors to detect denial of service (DoS) attacks based on SYN, UDP and ICMP flooding (*sflood_s*, *uflood_s*, *iflood_s*) and, finally, a response unit capable of producing packet filtering (*pfilter_ru*).

The implementation of the host sensors is based on the interception of some system calls with the purpose of obtaining useful information in the search process of illicit or suspicious activities. On the other hand, the implementation of the host based response units uses the same idea to provide the needed mechanisms to prevent the associated action related with the step of the attack to avoid. We have finished the development of a sensor to monitor the execution of programs (*execve_s*), a sensor to detect which processes want to be finished (*kill_s*) and a response unit able to kill and protect host processes (*kill_ru*).

B. Communication of Events and Actions

The sensors provide the events to each prevention cell analyzer. On the other hand, the counter measure manager of each prevention cell provides the actions to the response units. As we already mentioned, sensors and response units work in kernel space. The complexity of these components and the limitation that supposes to work in a kernel scope entails to design them as daemon processes in user space. Thus, a specific communication mechanism between kernel space and user space is needed.

Among the diverse alternatives for performing the communication between kernel space and user space, we have chosen the *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 has been designed with focus on implementing protocols based on IP services, this mechanism can also be used as a standard interface to perform communication between 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.

C. Analyzers and Managers

Both the implementation of the analyzer and the counter measure components, as well as the other managers of each prevention cell, are 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 and produce information.

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 event watcher analyzer, 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.

D. Communication of Alerts

The communication between the analyzers and managers, inside each prevention cell as well as between the other prevention cells of our architecture, is performed by using the Elvin publish-subscribe system [21]. 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 using libelvin and e4xx, two portable C and C++ libraries for the Elvin client protocol. Additionally, each host with a prevention cell lodged inside will run an Elvin server to route all the alerts published inside each prevention cell.

Finally, 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 we guarantee confidentiality, integrity and authenticity of the cooperative alerts communicated between each prevention cell.

VII. Conclusions and Further Work

We have presented in this paper a decentralized solution for the detection and prevention of distributed and coordinated attacks from network resources. This system uses a secure multicast communication between different entities to avoid their participation in a coordinated attack against third party networks or even the local network. We have also outlined how our system can detect and prevent the BIND Birthday Attack, exploiting the distribution and coordination of the system components. Then, we have briefly discussed the implementation of a platform, which has been 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 time developed only for Linux 2.4, we plan to upgrade them to Linux 2.6.

As a further work, we are evaluating the possibility to incorporate the formal data model proposed in [18] in our approach. We are also making a more in-depth study of the IDMEF format [8] to solve unnecessary duplicated calculus inside each prevention cell. Finally, we will incorporate intrusion tolerant mechanisms to make our system more reliable when the host that lodges a prevention cell is infected.

Acknowledgments

We would like to thank Michael A. Jaeger for his comments on early drafts of this paper.

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] M. Asaka, A. Taguchi, and S. Goto. The implementation of IDA: An intrusion detection agent system. In *11th Annual FIRST Conference on Computer Security Incident Handling and Response (FIRST'99)*, 1999.

[2] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, Eugene H. Spafford, and Diego Zamboni. An architecture for intrusion detection using autonomous agents. In *ACSAC 1998*, pages 13–24, 1998.

[3] 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.

[4] F. Cuppens. Managing Alerts in a Multi-Intrusion Detection Environment. In *17th Annual Computer Security Applications Conference New-Orleans*, New-Orleans, USA, December 2001.

[5] 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)*, pages 806–817, Santiago, Chile, October 2002.

[6] F. Cuppens, S. Gombault, and T. Sans. Selecting appropriate countermeasures in an intrusion detection framework. In *Proceedings of 17th IEEE Computer Security Foundations Workshop*, Asilomar, Pacific Grove, CA, June 2004.

[7] 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.

[8] 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.

[9] J. D. de Queiroz, L. F. R. da Costa Carmo, and L. Pirmez. Micael: An autonomous mobile agent system to protect new generation networked applications. In *2nd Annual Workshop on Recent Advances in Intrusion Detection*, Purdue, IN, USA, September 1999.

[10] H. Debar, M. Dacier, and A. Wespi. *Towards a Taxonomy of Intrusion Detection Systems*. Computer Networks, 1999.

[11] H. Debar and A. Wespi. Aggregation and Correlation of Intrusion-Detection Alerts. In *Fourth International Workshop on the Recent Advances in Intrusion Detection (RAID'2001)*, Davis, USA, October 2001.

[12] 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.

[13] G. Helmer, J. Wong, M. Slagell, V. Honavar, L. Miller, R. Lutz, and Y. Wang. Software fault tree and colored petri net based specification, design and implementation of agent-based intrusion detection systems., 2002. Submitted to IEEE Transaction of Software Engineering.

[14] 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.

[15] R. Janakiraman, M. Waldvogel, and Q. Zhang. Indra: A peer-to-peer approach to network intrusion detection and prevention. In *Proceedings of IEEE WETICE 2003*, Austria, June 2003.

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

[17] C. Kruegel and T. Toth. Flexible, mobile agent based intrusion detection for dynamic networks. In *European Wireless*, Italy, February 2002.

[18] 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.

[19] P. Ning, Y. Cui, and D. Reeves. Constructing Attack Scenarios Through Correlation of Intrusion Alerts. In *proceedings of the 9th ACM conference on Computer and communication security*, pages 245–254, Washington DC, USA, 2002.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] A. Valdes and K. Skinner. Probabilistic alert correlation. In *Fourth International Symposium on Recent Advances in Intrusion Detection (RAID2001)*, pages 58–68, Davis, CA, USA, October 2001.

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