Winter 12-5-2004

Security Alert Management in E-Business Networks

Allan Lam

Pradeep Ray

Security Alert Management in E-Business Networks

Allan Lam, Pradeep K. Ray

School of Information Systems, Technology and Management,
University of New South Wales, Sydney, Australia
p.ray@unsw.edu.au

ABSTRACT

Security management has become a major concern in today's e-business systems due to ever-increasing attacks on enterprise servers. This has led to the increasing sophistication of network security tools and systems in e-business networks that involve a number of organizational entities cooperating over computer communication networks. Many large organizations are outsourcing the management of e-business networks. This paper examines the problem of security management in the context of an Management Service Provider (an organization that provides remote management of e-business networks). Existing security tools (e.g., Intrusion Detection Systems (IDS)) assist us in detecting attempts by unauthorized users to get access to networked information resources. However, the management of IDSs offers some interesting challenges (e.g., false alerts). This paper presents a policy-based management framework to solve this problem.

Keywords: Outsourced IT management, IDS Alerts, Management Service Provider

1. INTRODUCTION

E-business systems allow organizations to focus on their core strengths, while outsourcing many aspects of the business to business partners in a value chain. For example, many e-businesses now outsource the management of their IT networks to specialist organizations [7]. The increasing ferocity of cyber-attacks is making organizations think seriously of more and more sophisticated security management solutions. Organizations now have firewalls, anti-virus, authentication, and Intrusion Detection Systems (IDS). However, many of these facilities are either unused, or under-utilised due to problems, such as false alarms in IDS [1]. This paper presents a security management strategy for large e-business networks from the perspective of outsourced management.

The paper is organized as follows. The Section 2 starts with a background of current intrusion detection systems and the need for frequent management interventions. This is followed in Section 3 by our proposed architecture for Security Alert Management (based on management policies) that helps reduce frequent false alarms. Section 4 presents a case study of the implementation of this architecture in a large organization providing management services to e-business networks in the Asia-pacific region. Finally, Section 5 concludes the paper.

2. BACKGROUND

Many network security technologies require regular human intervention in order to receive the best protection. Consider firewalls and intrusion-detection systems (IDSs). Typically, these devices are setup so that they log information to a database. It is then up to the security engineer to analyse these logs and to identify network attacks or suspicious behaviour if, or while, they are occurring. Unfortunately, this can be very time consuming.

Logs are generated and typically contain information for the following three events:
A packet that is allowed to pass through a firewall.
A packet that is blocked (denied access) at a firewall.
A packet that satisfies one of the signatures in the IDS database.

Thresholding rules are then applied to the above events to generate security alerts. Unfortunately, because they work on data from firewalls and IDSs, they also inherit the major problem with them, e.g., the generation of numerous false positives [2].

This research reported in this paper aims to develop a framework whereby security alerts, in particular false or routine alerts, can be analysed and responded to automatically, saving precious employee time and the employer's money. Routine alerts are those that we wish to appear but which we do not want to analyse. False and routine alerts are composed of many events. Our proposed framework is called Security Alert Management System (SAMS). This is a Policy-based management framework as discussed in [4], [5] and [6].

3. SECURITY ALERT MANAGEMENT SYSTEM (SAMS)

SAMS uses the term policy to refer to a particular rule that defines a false or routine alert. Every policy is instantiated from a policy type. A policy type represents a set of policies that possess a similar underlying idea/pattern. For example, consider the following policy types:

- If pattern A appears at least Z% of the time in the alert, then acknowledge this false/routine alert.
- If pattern A appears at least Z% of the time in the

alert, AND if a pattern B appears at least Y% of the time in the alert, then acknowledge this false/routine alert.

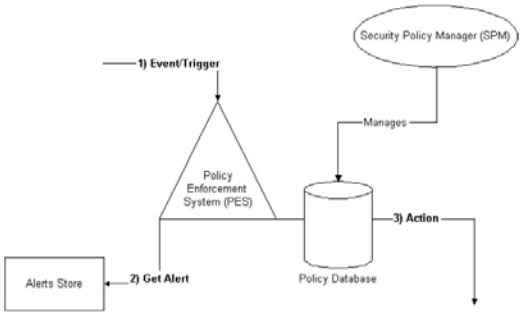

Figure 1: SAMS framework

The above policy types can be used to instantiate the following policies:

- If the source IP address 127.0.0.1 appears 90% of the time in the alert, then acknowledge this false/routine alert.
- If the source IP address 127.0.0.1 appears 40% of the time in the alert, AND the destination IP address 234.22.12.120, then acknowledge this false/routine alert.

The policy types then really have no constraints, other than their realisation into code, which will be discussed later.

3.1 SAMS Framework

The framework presented in Figure 3 provides a design for the development of a system that analyses security alerts. It is called the Security Alert Management System (SAMS) framework. At a high level of detail, SAMS will wait for an event or trigger to start the alert analysis process. Once this event has been received, the alerts are retrieved from an alerts store. Each alert is then analysed with its corresponding policy, which is determined by the domain that the alert belongs to. Each policy will determine whether or not the alert is a false alert or a routine alert. The policy will also determine what action to carry out based on that analysis.

The simple framework of SAMS reflects the nature of the problem, which is to analyse and respond to security alerts. However, its simplicity hides the powerful inner workings of the policies. The more verbose or complex your false or routine alerts, the more complicated your policies become. As shown in Figure 1, the framework is made up of three main components, the Policy Enforcement System (PES), the Policy Database and the Security Policy Manager (SPM). These will be described in the following three sections.

3.2 Policy Enforcement System (PES)

The PES does three things, as shown in Figure 2 with a triangle:
A) Intercept an event/trigger. This event/trigger can be a mouse click or a timed event and invokes the PES process.
B) When the PES is invoked, alerts are read from whatever format that they are stored as.
C) Each alert, based on its domain is then compared to its corresponding policies found in the policy database. If a policy is found that matches the alert then the action for that alert is automatically carried out.

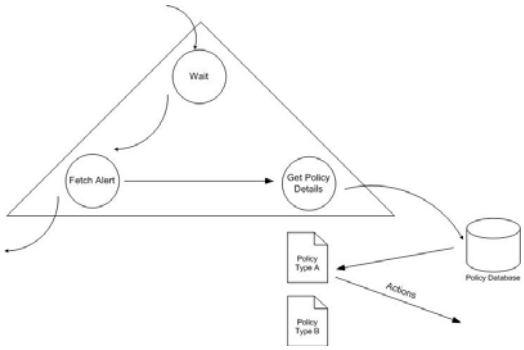

Figure 2: PES process

The PES 'waits' for the event/trigger to start the process. Upon receiving this, it will 'fetch alerts' from the alerts store. Finally, it will 'get the policies' for this alert from the policy database. The policies that it retrieves are based on the alert domain. The policies are compared to the alerts and if one is found to "match", then the appropriate 'actions' are carried out. The last step is actually a little more involved, which will now be discussed. Recall from the beginning of section 3 that we wish to define different policy types. Each policy type will define a false or routine alert. The PES will actually store the policy type definition, represented as 'Policy Type A' and 'Policy Type B' in the above diagram. The policy database will then store the particulars for the policy type definitions. For example, the PES may have the following as 'Policy Type A':
If pattern A appears at least Z% of the time in the alert, then acknowledge this false/routine alert.

The policy database may then have a table such as the following:

alert_domain	pattern_A	percentage_Z
1	Shellcode.*	90

So if an alert that belongs to the alert_domain 1 is encountered, then the corresponding pattern (Shellcode.*) and percentage (90) will be replaced in the policy type. Encountering such an alert will apply this policy to it:
If pattern Shellcode.* appears at least 90% of the time in the alert, then acknowledge this false/routine alert.
If the alert satisfies this policy, then it will be

acknowledged. Note that the pattern supports regular expressions. By separating the policy type and its specific details, we allow a policy to be applied to various domains and alerts, without producing repetitive policies. As the policy types need to be coded, this also reduces code redundancy.

In summary, the PES has been designed so that it stores the programming logic to analyse alerts. It in fact contains the skeleton of the logic. The remainder of the programming logic is obtained from the policy database. The reason for this separation is that many alerts have similar analysis. For example, many alerts are analysed based on what event occurs the most often. So, instead of replicating the code for this analysis across all domains, we can save disk space and redundancy by storing only the skeleton of the code and storing the finer details in the database. Creating new policy types therefore involve coding them into the PES.

PES is the only component that integrates with a current thresholding system.

3.3 Policy Database

This represents the database of policies. As described in the previous section, it will store the details of the policies in a policy table. Each domain is also represented in the Policy Database. This only makes sense since it is natural and easy to link domains with their particular policies via table joins. A schema such as the following can be employed for security alert management systems developed from SAMS (Figure 3).

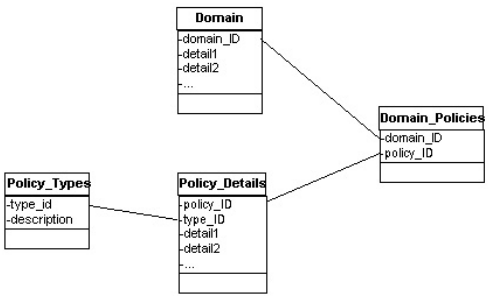

Figure 3: Policy Database schema

The table Domain will contain information about that particular domain. Policy_Details will contain information about all policies that have been created for the system and Domain_Policies is a lookup table matching the policies to the particular domains. So to retrieve the details for a particular policy the following query might be performed:
select p.detail1, p.detail2 from domain_policies d, policy_details p where d.domain_ID=1 and d.policy_ID=p.policy_ID.

The attributes returned from the above query can then be substituted into the policy type definition defined in the PES. The Policy_Types table helps with this task by associating a policy instance in the table Policy_Details to the policy type definition in the PES.

3.4 Security Policy Manager

The SPM is the GUI management tool that will allow you to manage the Policy Database. It will allow you to enable, disable or edit policies for your defined domains. The SPM will present the information from the database in an easy to read format allowing rapid creation of the policies.

As discussed in the beginning of section , policies are instantiated from policy types. Hence, when we enable a policy, we are actually creating an instance of a policy type. Every instance is created by inserting the details for the policy into the policy database, discussed in the previous section. The policy created is then applied to a selected domain.

When we disable a policy, we are actually removing an instance of the policy from the policy database. This involves removing various rows from the policy database to remove both the policy instance and its association with a particular domain. Note that when we disable a policy, we do not actually remove the policy type definition from the PES. We are only removing an instance of it, which is defined in the policy database.

Editing a policy allows us to edit the details of a policy instance. For example, consider the following policy:
If pattern Shellcode.* appears at least 90% of the time in the alert, then acknowledge this false/routine alert.
The user could modify this policy by changing the pattern or changing the percentage value. Note also that editing a policy does not edit the policy type definition. Editing a policy only edits a policy instance. Editing a policy will perform a database ‘update’ query on the policy_details table.

Note that whenever a new policy type is created in the PES, the SPM must also be updated so that it recognises it. The SPM must be updated so that one can enable, disable and edit policies based on the new policy type.

4. CASE STUDY

This section will use the SAMS framework to develop a working system. The system will be used on a thresholding system already in place at a company X. The company has asked its name to be withheld. The company is a leading IP networking and communications solutions provider in the Asia-Pacific region.

Of interest to this paper is the orginsation’s security division. The security engineers, among other tasks,

monitor the alerts from various firewalls and IDSs for various customers. The security division has implemented a thresholding system, which analyses events from firewalls and IDSs and based on predefined thresholding levels will generate alerts. These alerts are displayed to security engineers via an internal intranet web page.

The SAMS framework helps design a system that automatically acknowledges false and routine alerts produced from the thresholding system. According to the engineer, on average it takes one minute to analyse one security alert. On an average day for the organisation's security division, they receive 300 alerts. This means that every day, 300 minutes is used to analyse false alerts. If there are 230 working days per year, that equates to 1150 hours or 143 working days that are spent on analysing alerts.

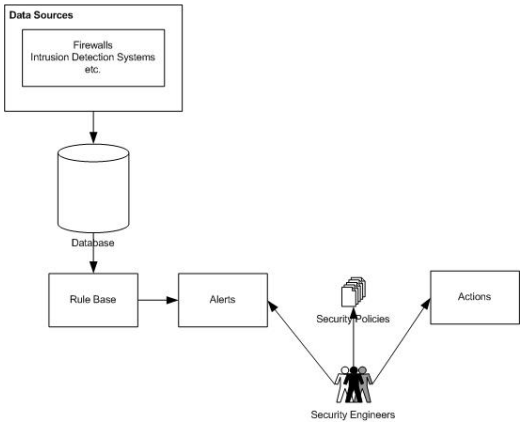

Figure 4: Current security architecture of company X

4.1 Current Security Architecture

Before we design a SAMS system, we must first analyse the current architecture to see where it fits in. The following diagram shows the current data flow of the organisation's security division:
The data sources log their events to a PostgreSQL database. The thresholding system rule base then generates the alerts. Security engineers will analyse these alerts and perform the required actions. As indicated Figure 4, the security engineers are actually using policies when they are analysing the alerts. That is, the security engineers are applying certain rules for the analysis that they carry out.

4.2 Design

This Section presents the design of the proposed system based on SAMS (Figure 5). However, we first need to identify how our alerts are being stored.

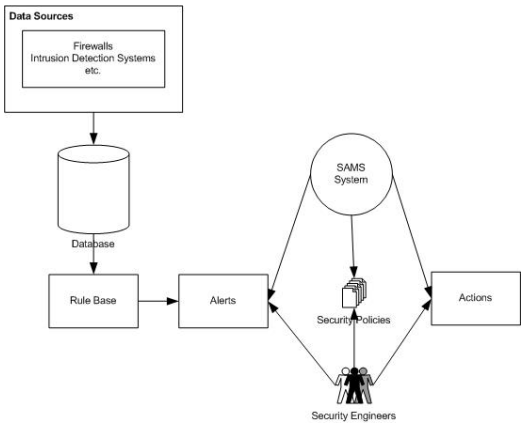

Figure 5: Proposed SAMS Architecture

The PES will therefore perform a similar task. It will read in the alerts from a HTML file and In the case of the organization's thresholding system, the alerts are stored as HTML files. Security engineers analyze alerts by entering an intranet web site. This web site provides the alerts and the events that make up in the alerts in a tabular format. It is then up to the security engineer to analyze these and acknowledge that they have been analysed. PES will only acknowledge the alerts if the policies in its database match the alerts.

The next step is to identify the domains. In the case of thresholding system, these will be the customers. Since the organisation manages customer networks, it is only logical that we distribute policies according to this domain. The existing database already has this information in a table called 'customers'. The schema for this table is as follows:

```
customers(id, name, fullname, heat_id,
gnc_name, speech_name)
```

Hence for our SAMS system we will need to create these tables:

```
policy_details(policy_id, description, …)
cust_policies(cust_id, policy_id)
policy_types(type_id, description)
```

The full schema of policy_details will be worked out later.

Our final step is to develop the SPM. Because the existing thresholding system is web-based, the SPM designed here will also be web-based. Similarly, because the existing thresholding system is developed in Perl, our SAMS system will also be developed in Perl.

4.3 Implementation

After discussion with the organisation's security staff, it was determined that the main policy type that should be created first, was one that specified a pattern that was most prevalent. That is, in the analysis of every alert, false and routine alerts tended to have a pattern whereby certain elements would be repeated. As an example of such an alert, refer to the following that has been used throughout this report:

- If pattern Shellcode.* appears at least 90% of the

time in the alert, then acknowledge this false/routine alert.

This is the first policy type that was coded into PES and which the SPM will recognise. Hence, after this decision was made, the schema for the table policy_details is as follows:
policy_details(policy_id, devicetype_id, type_id, details)

The important attributes are described here:

- type_id: this is a foreign key to the policy_types table and associates a policy with its policy type.
- details: this will be a string that contains the details for a policy. Since we are implementing the above policy type, this string will contain the pattern to look out for and the percentage that this pattern appears in the alert. More precisely, it will contain what specific details of an alert contain the pattern, what the pattern is and the percentage value.

When the user is creating a policy, they must first select which components of an alert make up the "key". The key is a combination of the attributes in the above table that the policy type definition will concentrate on when it is analysing it.

The SPM was then designed so that it could properly manipulate the policy database. As mentioned, all development work for this system was done in Perl. The existing thresholding system was developed in Perl, so for compatibility issues, it was logical to select this as the development technology.

4.4 Results

A successful SAMS system was developed on the development network. The following table shows how many alerts could have been automatically acknowledged if the SAMS system was put into production.

Time (for 26 November 2003)	Number of alerts that could have been automatically acknowledged
00:00 to 01:00	6
01:00 to 02:00	3
02:00 to 03:00	3
03:00 to 04:00	3

This is but a four-hour sample of the number of alerts that could have been automatically acknowledged. It would equate to 15 minutes of saved alert analysis time. For another random sample:

Time (for 27 November 2003)	Number of alerts that could have been automatically acknowledged
13:00 to 14:00	7
14:00 to 15:00	9
15:00 to 16:00	5
16:00 to 17:00	6

The above would have saved 27 minutes of alert time. These alerts are all based on the same policy type specified in the previous section. Due to the nature of networks it is hard to predict how many alerts on average could be automatically acknowledged with this policy type. But it is clear from the above samples that significant time can in fact be saved.

Future work will involve the creation of policy chains, introduction of policy priorities and the development of a policy type creator, since with the current framework, it is quite cumbersome to add other policy types. Also it is necessary to integrate the SAMS framework with the network and systems management framework [3].

5. CONCLUSION

In this paper, we have proposed a new framework, called SAMS, for the analysis of security alerts in e-business networks. This framework incorporates foundational concepts in policy-based management to solve the problem of false alerts that threatens to render many security tools (e.g., intrusion detection systems) useless.

The framework that is proposed in this paper is free from many constraints on the policy definitions, unlike in the case of existing management policy frameworks. In fact, policies defined in this framework are constrained by the author's ability to code them. This is hopefully a first step towards a framework that may become universally used in any system that generates alerts.

The time and money wasted in analysing false and routine alerts is great as sproven by our case study figures. A system developed from SAMS can also be viewed as an "engineer-in-a-box". Because the security engineer is telling the SAMS system what constitutes false/routine alerts, the system is really acting like a security engineer. As a consequence, if the security engineer decides to leave the organization for which he/she works, then the knowledge base of the engineer will be retained.

Such policy-based management frameworks are likely to be used more increasingly for security management in future.

ACKNOWLEDGEMENT

This work was partially supported by the Australian

Research Council (ARC) SPIRT Grant #C00107103.

REFERENCES

1. Axelsson, S., "Intrusion Detection Systems: A Survey and Taxonomy", Technical Report of the Department of Computer Engineering, Chalmers University of Technology, Sweden, March 2000
2. Blackman,D., "Intrusion Detection is failing: Enter Intrusion Management", July 2002, http://www.itsecurity.com/papers/pentasafe1.htm
3. Distributed Management Task Force, Inc. (DMTF), "Common Information Model (CIM) Specification", version 2.2, June 14, 1999, http://www.dmtf.org/spec/cims.html
4. N. Dulay et al. A Policy Deployment Model for the Ponder Language (2001)
5. B. Moore, J. Strassner, E. Elleson, "Policy Core Information Model -- Version 1 Specification", Feb 2001, ftp://ftp.rfc-editor.org/in-notes/rfc3060.txt
6. Proctor,P., "The Practical Intrusion Detection Handbook", Prentice Hall, 2001
7. Ray, P., "Integrated Management from e-Business Perspective: Concepts, Architectures and Methodologies", Kluwer Academic/ Plenum Publishers, Januray 2003
8. P. Yarng, P.Ray and D. Maher, "Profiling Cyber Attacks using Alert Regression Profiles", IEEE Globecom2003 Symposium on Network Security, San Francisco, Nov 2003