AN EXPERIENCE-BASED INCIDENT RESPONSE SYSTEM

G. Capuzzi, E. Cardinale, I. Di Pietro, L. Spalazzi
Dipartimento di Ingegneria Informatica, Gestionale e dell’Automazione
Università Politecnica delle Marche
60100 Ancona
E-mail: {capuzzi,cardinale,di Pietro,spalazzi}@diiga.univpm.it

ABSTRACT

This paper presents a tool for attack detection, attack identification and attack response. These activities have received a great attention by the research community and by several organizations (e.g., ISO/IEC and CERT). Nevertheless, most of the work focuses on the detection and the identification of intrusions instead of attack identification and response. In our work, an intrusion is a detectable atomic action performed by an attacker against a given target, whereas an attack may go through several phases. Each phase involves different methods and different goals. Therefore, according to our meaning, an attack is a sequence of intrusions.

In our approach, the attack identification and response can be fulfilled in four distinct phases. The first phase deals with intrusion detection. This means collecting data from several sensors on the network and on computers, e.g., log files of operating systems and system servers, firewalls, (network-, host-, application-based) IDSs. The second phase deals with alarm correlation. This means correlating all the data collected in the previous phase to the end of providing an attack description in terms of sequence of events as complete as possible. The third phase deals with identification by means of an experience-based model. This means: the tool has a case memory that contains past attacks described in terms of their features and their corresponding event sequences. This allows us to have a tool capable of identifying an attack on the base of its similarity with previous experiences, and learning new kinds of attacks. The fourth phase deals with response. We can link to each past attack an appropriate response plan, then, after the attack identification, we can reuse (after an appropriate adaptation) the plan associated to the recognized attack. This allows us to have a tool capable of identifying an attack on the base of its similarity with previous experiences, and eventually learning new kinds of attacks.

INTRODUCTION

What makes a security manager better than another one? The trivial answer is: her/his experience. The greater her/his experience, the greater it is the number of attacks examined in the past that can be recognized if they occur again. The greater her/his experience, the greater it is the number of past attacks to which the security manager has found an appropriate responses, the greater it is the possibility of finding a response to a new attack. This is the focus of our work. We propose a system that, at first, supports the security manager in all her/his activities related to attack response (detection, identification, reporting and response) and, second, is an experience based system. Let us explain this idea depicting two different scenarios. In the first scenario, let us suppose to have an attack similar to a past attack: our system has to recognize it like a new occurrence of the past attack. If in the past we have successfully responded to this attack with certain actions, it is reasonable to suppose that these actions may be applied to the new attack as well, even if after an appropriate adaptation. Our system has to retrieve the past plan and adapt it to the current attack. This adapted plan is presented to the security manager that decides whether it has to be modified and/or executed. Part of plan actions must be executed by security manager or other human operators; the rest must be executed by computers. Therefore, our system has to overview plan execution, automatically run the part of the plan that that must be executed by computers, and monitor results. In the second scenario, let us suppose to have an attack that never occurred. Even in this case, our system is able to select past attacks from its database (with their response plans) that share some (few) characteristics with the current attack. The security manager can thus build a response plan for the current attack on the basis of these past response plans. The description of the new attack and the related response plan may be retained and thus improving system experience.

The paper is structured as follows. Section provides a system overview and the related work. In Section , we discuss attack detection, in other words: sensing, normalization, and correlation. The case memory is described in Section . Section deals with the entropic filter that aims to reduce the number of false alarms. Section deals with attack identification. In Section , we describe the structure of the response plan and the adaptation process. Some Conclusions are drawn in Section .

SYSTEM OVERVIEW AND RELATED WORK

A network and computer security incident is “any adverse event whereby some aspect of computer secu-
The architecture of an experience-based system for attack identification.
of the retrieved past attack. We should need only few adaptations. We use a simple adaptation algorithm before executing the plan. This plan is returned to the security manager who checks and corrects it. After its validation, the security manager can allow plan execution. A plan is a sequence of actions, each action can be an action that must be automatically executed by a software, or an action that must be performed by the security manager. After the execution stage, the system stores the plan updating the case memory. In this way, the system is able to learn from new cases and to improve the effectiveness against new attacks. There are tools that work in this phase, but they have a different function. They provide response plans quite different from ours, consisting of generic advices to create a security plan for the network or several advices to avoid insecure behaviours. Our system provides response plans consisting of detailed actions to assess the incident, restore the previous status, and improve the security level of the network. We have just an example where the returned plans are similar to our kind of plans (see [17]). That system is based on classical AI planning. When it receives alerts from IDSs, it establishes what are the goals to satisfy in order to react to the attack. It has a set of actions described in terms of preconditions and affect and combines them in order to reach the goal from the current situation. This solution is flexible when we have never recorded attacks. Nevertheless, classical AI planning has been recognized to be a complex task (usually it is undecidable). It is based on the assumption to have “a complete theory” able to describe all the possible actions we need to use. Unfortunately, this is usually true only for toy examples. In our system, we prefer to trust the experience. Indeed, an experience-based approach is known to be extremely useful in the diagnosis and the management of several different kinds of emergency [10], [3], [9], [13]. For instance, the work of [10] deals with alarm correlation (i.e., with identification), even if not with response. Furthermore, the system deals with fault alarms, not intrusion alarms, and the goal is to obtain a fault tolerant network, not a secure network. The work of [3] and the work of [9] deal with intrusion detection. There the goal is to exploit learning to improve detection and avoid the need of frequently updating the database of known attacks. In our work, the goal is the improvement of attack identification and response by means of learning. We have just an example of application to incident response [13]. There the goal is to improve detection and to avoid the need of frequently update the database of known attack.

ATTACK DETECTION

Attack detection is fulfilled in three steps: intrusion detection, alarm normalization, alarm correlation.

Intrusion detection consists of sensing all the alerts that can provide us information about possible attacks. This mainly relies on sensing data, such as data obtained by monitoring traffic on a network, activity logs stored on a computer, or system state. We use both signature recognition and anomaly detection sensors.

Sensed data include a lot of irrelevant information, and cause difficulty for efficient and accurate attack detection. For example, the log file of an operating system contains a vast range of log messages, only few of them deal with intrusions, the rest deals with normal system activities or errors. In other words, we have an event log everytime a device is mounted or unmounted, everytime the clock must be synchronized, or everytime a system error occurs. Furthermore, each sensor has its own output data format. Therefore, the collected data must be reduced in number and ordered in a single specific format. For this reason, for each sensor we have developed a module to have a preliminary filtering of sensed data and to normalize them in a specific format. The preliminary filter is composed by a service-based white-list: applications not included in the white-list are not monitored. For example, we do not need to take into account alarms related to FTP when the FTP service is not active on that host. Concerning the normalization, for alarm we consider the following parameters: the tipology of the sensor that produced the alert, the timestamp, the destination port, the source IP, the target IP and the related message. We developed different modules in C++ and Python, one for each sensor, to normalize its output in the specified format. Normalized alerts represent the input of the correlator.

The correlator is the module demanded to correlate alerts coming from different sensors and producing an attack descriptor list, composed of sequences of alerts. It also reduces redundant alerts, by a fusion process. This module is a variant of the correlator proposed by [1]; we only considered the following subsystems:
- fusion: it correlates several instances of the same event detected by different sensors; for instance, a malformed packet sent to the DMZ subnet should be detected by the perimetral sensor and by a sensor on the DMZ
- session reconstruction: it correlates alerts, detected by host intrusion detection and network intrusion detection systems; for instance, an attack launched against a web server should be detected by a NIDS and by a HIDS installed on the web server
- focus recognition: it correlates one to many attacks (for instance, a ports scanning), consisting of alerts that have the same source and different targets and many to one attack (for instance a DDoS attack), consisting of alerts that have the same target and different sources
- thread reconstruction: it links all the alerts of the same attack, looking for alerts from the same source to the same target

In addition, we did not consider the IDMEF format for alerts description.

CASE MEMORY

In the case memory, each case consists of two components: a static component and a dynamic component. The dynamic component is given by the list of
correlated alerts that form the attack. This part is compared with the current attack in order to identify which kind of attack is occurring. The static component contains the type and a description of the attack, its response plan and other information (in the future its priority). After that the attack has been identified, this part can be used to prepare an appropriate incident report and response plan. In Figure 3, we reported a partial representation of attacks in the case memory. Notice that, in the case memory are represented in an "abstract" way. Let us explain this concept with an example. Figure 2; reports an attempt of opening a backdoor by a trojan horse. It consists of six events: a portscanning, four fingerprinting attempts and a trojan sent through the network. For each event, the report of the current attack includes: attack type, sensor, source and target IP addresses, sensor, and event description (i.e., the intrusion type). Let us suppose that in the past we had an event of the kind SNMP trap tcp with a given IP number (say 192.168.0.3) as target address. One of the current events is an SNMP trap tcp on a different IP number (say 172.16.113.84). It seems quite natural to consider these two events similar (they have the same event type), even if these two events are not identical (they have a different target). Event abstraction is the tool to find similarities without taking into account irrelevant details. In short, it consists of substituting some values as source and target with their type. For example, if the IP number 172.16.113.84 is the address of a web server, we can substitute the number with the keyword webserver. Formally, let $e = \langle \text{eventType}, \text{sensor}, \text{source}, \text{target} \rangle$ be an event, then $Abs(e) = \langle \text{eventType}, \text{sensorType}, \text{sourceType}, \text{targetType} \rangle$ is the abstraction of $e$. $Type(e) = intrusionType$ is the event type of $e$. Let $I = \langle e_1, \ldots, e_n \rangle$ be an attack, then $Abs(I) = \langle Abs(e_1), \ldots, Abs(e_n) \rangle$ is the corresponding abstraction of $I$ and $Type(I) = \langle Type(e_1), \ldots, Type(e_n) \rangle$ the corresponding sequence of event types of $I$. The attack abstraction mechanism could be based on a file or table representing the description of the actual network in the form of $\langle \text{host}, \text{IP} : \text{host}, \text{description} \rangle$ pairs.

**ENTROPY-BASED FILTERING**

Alert correlation allows us to analyze complex attack descriptors rather than single events. The advantages of this method are the ability of recognizing stateful attacks (composed of several alerts) and the availability of an abstract description of a generic attack, independent from a specific network configuration. However, this fashion leads to drawbacks, as well. The correlation module does not perform any analysis on the attacks it produces, thus it produces several false positives in the output list. For this reason, we developed a filtering module, whose aim is to reduce the number of false positives. An index of significance is needed to recognize trivial attacks. In our approach, we base this index on the well-known information theory. We define this index in two steps. First, we define a weight for each type of event. Second, we define a weight for a sequence of events.

The event weight is a function of the frequency: the lowest the frequency is, the highest the weight. An index of significance is needed to recognize trivial attacks. In our approach, we base this index on the well-known information theory. We define this index in two steps. First, we define a weight for each type of event. Second, we define a weight for a sequence of events.
(based on the occurring frequency) that such event type occurs and, thus, we can compute its entropy as follows:

\[ w_t = -\log_2 p(t) \]  

(1)

Concerning the weight of an attack, it is useful to notice that attacks can be roughly divided in two categories:
- significant event attacks: in which few relevant events are enough to recognize a particular attack pattern. To this category belong attacks based on malformed packets. IDS are very sensible to unexpected packets formats. In this case the attack could be composed of a single event (e.g., the LAND attack has the same value in the source and target fields of IP header).
- multitude event attacks: characterized by a large number of events, rather than their type. An attack can be performed with a big number of common and legal packets. This is the case of flooding DoS attacks. For example, a SYN flood DoS attack is done by sending a lot of TCP SYN packets to a web server. SYN packets are perfectly legal and they use to flow between client and server in the three-way-handshake phase. A DoS attack is detected if too many SYN packets come from spoofed sources. A system that analyzes single events would not be able to detect this kind of attacks. Therefore, referring to these two categories, we defined two indices of significance for attacks evaluation: the stateless (or low) information content and the stateful (or high) information content. The former is the average of event weights in attack sequence, while the latter takes into account the multiplicity of each event type, which raises the score of stateful attacks. In formulas:

\[ I(A)_{\text{low}} = \frac{\sum_{\forall e_\in A} w_e}{N_A} \quad I(A)_{\text{high}} = \frac{\sum_{\forall e_\in A} w_e \cdot (1 + \alpha \cdot n_A(e))}{N_A} \]  

(2)

where
- \( N_A \) = number of alert types in attack \( A \)
- \( n_A(e) \) = number of alerts of type \( e \) in attack \( A \)
- \( \alpha \) = multiplicity weight (0 < \( \alpha \) < 1)

Filter module uses two different threshold parameters: \( LT \) (low threshold) and \( HT \) (high threshold), for stateless and stateful attacks information level evaluation, respectively. Attacks can be classified in attacks that we can ignore (trivial attacks) and attacks that we must take into account (serious attacks):
- \( I(A)_{\text{low}} < LT \) and \( I(A)_{\text{high}} < HT \) ⇒ TRIVIAL
- \( I(A)_{\text{low}} \geq LT \) ⇒ SERIOUS (stateless)
- \( I(A)_{\text{high}} \geq HT \) ⇒ SERIOUS (stateful)

The filter passes to the next module (the attack recognizer module) only the serious attacks. The two thresholds can be manually set by the security manager, or automatically set by the system. The filter has two techniques to set the thresholds, a run-time mode and a training mode. The first set the thresholds everyday an attack occurs, basing of the difference between the informative content of the new attack and the average informative content of the case memory. The second set the thresholds each epoch: an epoch consists of a predefined (by the security manager) number of attacks.

**ATTACK IDENTIFICATION**

The output of the entropic filter is a list of abstract attacks, each attack composed by a sequence of events. The filter eliminated most of false positives. Now, the attack identification module has to retrieve past attacks similar to the attack that passed the filter. The retrieval can be achieved by means of an appropriate similarity function. In this paper, we propose the following four:

**Definition 1:** Let \( I_c = (e_{c,1}, \ldots, e_{c,n}) \) be the current attack, let \( I_k = (e_{k,1}, \ldots, e_{k,n}) \) be the attack of the k-th case in the case memory, let \( \text{Type}(I_c) \) and \( \text{Type}(I_k) \) be the sequence of event types of \( I_c \) and \( I_k \), respectively. Let \( n_t^{(c)} (n_t^{(k)}) \) be the number of how many times the event type \( t \) occurs in \( \text{Type}(I_c) \) (\( \text{Type}(I_k) \)). Let \( m_t^{(c)} = \begin{cases} 1 & \text{if } t \in \text{Type}(I_c) \\ 0 & \text{otherwise} \end{cases} \quad m_t^{(k)} = \begin{cases} 1 & \text{if } t \in \text{Type}(I_k) \\ 0 & \text{otherwise} \end{cases} \)

be boolean functions that are true when \( I_c \) (\( I_k \) respectively) has at least an event whose type is \( t \). Let \( w_t \) the entropy of the event type \( t \). Then:

\[ F_1(I_c, I_k) = \frac{\sum_{\forall t \in \text{Type}(I_c)} \min(n_t^{(c)}, n_t^{(k)})}{\sum_{\forall t \in \text{Type}(I_c)} n_t^{(c)}} \]
\[ F_2(I_c, I_k) = \frac{\sum_{\forall t \in \text{Type}(I_c)} \min(m_t^{(c)}, m_t^{(k)})}{\sum_{\forall t \in \text{Type}(I_c)} m_t^{(c)}} \]
\[ F_3(I_c, I_k) = \frac{\sum_{\forall t \in \text{Type}(I_c)} w_t \cdot \min(n_t^{(c)}, n_t^{(k)})}{\sum_{\forall t \in \text{Type}(I_c)} w_t \cdot n_t^{(c)}} \]
\[ F_4(I_c, I_k) = \frac{\sum_{\forall t \in \text{Type}(I_c)} w_t \cdot \min(m_t^{(c)}, m_t^{(k)})}{\sum_{\forall t \in \text{Type}(I_c)} w_t \cdot m_t^{(c)}} \]

\( F_1(I_c, I_k) \) depends on how many abstract events \( I_c \) shares with \( I_k \). It is the ratio between the events that \( I_c \) shares with \( I_k \) (with their multiplicity) over the number of events in \( I_c \). On the other hand, \( F_2(I_c, I_k) \) depends on the number of event types shared by \( I_c \) and \( I_k \). In other words, it considers one event per type. For example, let \( \text{Type}(I_c) = (A, B, C, D) \) the current attack and \( \text{Type}(I_k) = (A, B, C, D) \) the \( k \)-th attack in the case memory, where \( A, B, C, D \) are the event types of the corresponding events. When we apply \( F_1 \), we obtain 0.75, as result; otherwise, applying \( F_2 \) we obtain 1. This is due to the fact that it counts each event type once. Notice that, \( F_3 (F_4) \) is similar to \( F_1 (F_2) \), but it also takes into account the entropy of each event type. As a consequence, these functions have a discriminating power better than \( F_1 \) and \( F_2 \).
Definition 2: Let $I_e$ be the current attack, let $KB=$\{I_1, \ldots, I_n\} be a set of past attacks, then $Sim_x(I_e, KB) = \mathcal{T} \in KB$ is the most similar past attack of $I_e$ (according to the similarity function $\mathcal{F}_x(\ldots)$), and it is defined as follows:

$$F_2(I_e, 7) = \max_{I_e \in KB} F_0(I_e, I_i)$$

According to this definition, the procedure to find the most similar past attack is based on pattern matching. When a new attack occurs, the pattern abstracted by this attack (i.e., the abstract sequence of alerts) is compared with patterns in the case memory (the abstract sequence of past alerts) until a match is found. Consider the attack reported in Figure 2 and the case memory depicted in Figure 3. Let us suppose to use $F_1(\ldots)$ as similarity function, then we obtain that $F_1(I_e, Case1) = 0$, $F_1(I_e, Case2) = 0.5$, $F_1(I_e, Case3) = 0.67$, and $F_1(I_e, Case4) = 0.83$. Therefore, we select the attack Case4 as the most similar of the current one. On the other hand, applying $F_2(\ldots)$, we obtain $F_2(I_e, Case1) = 0$, $F_2(I_e, Case2) = 0.5$, $F_2(I_e, Case3) = 0.67$, and $F_2(I_e, Case4) = 0.83$. Finally, let us consider the weight in Figure 3. Applying the other retrieval functions, we obtain $F_3(I_e, Case1) = 0$, $F_3(I_e, Case2) = 0.44$, $F_3(I_e, Case3) = 0.63$, and $F_3(I_e, Case4) = 0.88$ and $F_4(I_e, Case1) = 0$, $F_4(I_e, Case2) = 0.44$, $F_4(I_e, Case3) = 0.63$, and $F_4(I_e, Case4) = 0.88$.

RESPONSE PLAN

The response plans we have used for our experiments are compliant with CERT recommendation and ISO17799. Indeed, each plan unfolds in three successive phases: in the first phase it gathers details of the current attack (incident assessment); in the second phase it improves the security of the network; in the last one it restores the normal status. As we can see in Figure 5, the first phase of the response plan extracts information related to the current attack: the name of the process, the numbers of the ports, the file that opens not reliable connections, etc. The second phase of the plan improves the security of the system, killing bad processes, updating system and security tools, installing patches, closing bad connections or suspicious ports, etc. Finally, the third phase aims to reach a normal condition using backup copies to restore services, software and files.

From the point of view of our system, the retrieved response plan simply consists of a sequence of abstract actions. Each abstract action must be translated in a concrete action. In order to do that, for each abstract action we have a set of concrete actions each of them related to a given host, operating system, environment, etc. Therefore, for each concrete action, we have appropriate preconditions to be satisfied. In other words, a concrete action can be used as translation of a given abstract if and only if its preconditions are satisfied. As a consequence, a concrete plan consists of a sequence of concrete actions with true preconditions. Let us explain this by an example. In Figure 4, the first column has the abstract actions of the plan, the second column has the related concrete action and the third column has the preconditions. Indeed, the first two actions aim to check the current process list comparing it with a reliable list, in order to detect bad processes. In the second column, we have the concrete actions for Windows OS and Linux OS. The system selects the second one, because the machine runs a Linux OS. The same process is applied to the other actions and the result plan is in Figure 5. As we said our plans respect these three phases suggested by CERT, but the adaptation algorithm does not consider them.

When a new attack occurs, the retrieval module searches for the closest past case in the case memory, and returns its linked plan. This plan is related to the similar attack, hence it is useful for the security manager: it consists of several advices to block the attack and restore the normal status. The plan is executed semi-automatically: before executing each action, the system asks a confirmation to the security manager, who can accept or deny the execution. If he denies the execution, he can change the action correct and execute it. After the execution, the new case with the correct plan are retained in the case memory: in this way, the system learns from new cases to improve the effectiveness.

CONCLUSIONS

Our work deals with an incident response system and the security manager activity. In this paper, we presented a tool for detecting and identifying complex attacks (a sequence of alerts that have the same goal), presenting a semi-automatic response plan to the security manager.

It is an innovative system because it covers all the phases of an attack response security system: from identification to response. Attack identification has been recognized as one of the most crucial activities, if we want to have a possibility of responding appropriately to an attack (e.g., see [11], [4]). Obviously, the attack identification process strongly depends on the previous experience of the security manager, but our system may help him with its identification system, proposing appropriate response plans and automating operations that can be.

Furthermore, every time a new attack is detected, the security manager must learn it. This model of the identification process relies on the observation that, as can be noticed in the DARPA experiments [12], it is infrequent to have twice the same attack, but it is very common to have several “similar” attacks. These three characteristics, experience-based reasoning, similarity-based retrieval and learning are the base of the proposed tool. Furthermore, our system is able to propose response plans based on responsive actions adopted for past similar attacks. Even if the kind of attack is new, the system is still able to propose, to the security manager, a response plan. If this plan is evaluated as appropriate, it can also be retained to improve system experience. Where it is possible, plans are automat-
<table>
<thead>
<tr>
<th>Abstract Actions</th>
<th>Concrete Actions</th>
<th>Preconditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>check the process list</td>
<td>taskmgr.exe</td>
<td>OS=Microsoft Windows</td>
</tr>
<tr>
<td>check the connection list</td>
<td>netstat &gt;&gt; ports.txt</td>
<td>OS=Microsoft Windows, Linux</td>
</tr>
<tr>
<td>kill illegal processes</td>
<td>badprocts.txt &gt;&gt; kill -9</td>
<td>OS=Linux</td>
</tr>
<tr>
<td>restore the normal status of the network</td>
<td>use backup copies to restore services</td>
<td>backup and backup utility</td>
</tr>
</tbody>
</table>

**Fig. 4. The response plan linked to the case4.**

| Response plan | 
|------------------|------------------|---------------|
| ps -aux >> proc.txt | badproctls.txt >> kill -9 | IOs=Linux |
| ps -aux >> proc.txt | ps -aux >> proc.txt | IOs=Linux |
| yum update | use backup copies to restore services | backup and backup utility |
| update opponents | use backup copies to restore corrupted softwares | backup and backup utility |
| update firebox | use backup copies to restore damaged files | backup and backup utility |

**Fig. 5. The adapted plan.**

I actually executed, after the security manager validation. From preliminary experiments done on the whole tool, it arises that a tool based on the experience, and capable of learning, capturing what a security manager usually does is quite hard to realize, but the preliminary results are promising. As future work, we will present experimental results.

**REFERENCES**


