11 Monitoring and Intrusion Detection

Ideally, an application of the techniques of *control and monitoring* is perfectly established: a security policy specifies exactly the wanted permissions and prohibitions; administrators correctly and completely declare the policy, which subsequently is fully represented within the computing system; and the control and monitoring component can never be bypassed, and it enforces the policy without any exception. As a result, all participants are expected to be confined to employing the computing system precisely as intended. Unfortunately, reality often differs from the ideal; for instance, the following shortcomings might arise:

- The security policy is left imprecise or incomplete.
- The declaration language is not expressive enough.
- The internal representation contains flaws.
- The enforcement does not cover all access requests.
- Administrators or users disable some control facilities for efficiency reasons.
- Intruders find a way to circumvent the control and monitoring component.

Some shortcomings might stem from careless or even malicious behavior of various persons involved in the overall process of building a secure computing system. However, we cannot blame only the responsible participants, but should recognize the inherent difficulties in fully meeting the ideals. Moreover, there are further, intricate difficulties. For example:

- In general, as indicated by undecidability results, control privileges and information flow requirements are computationally difficult to manage.
- For the sake of efficiency, information flow requirements can only be roughly approximated by access rights.
- A user might need some set of specific permissions for his legitimate obligations, but not all possible combinations of the permissions are seen to be acceptable.
- A user might exercise his permissions excessively and thereby exhaust the resources of the computing system.
- A user might exploit hidden operational options that have never been considered for acceptable usage.

Summarizing, we have to face the above shortcomings and difficulties and related ones, and, admitting an imperfect reality, have to prepare for them with additional protection mechanisms. Fortunately, the generic model of local control and monitoring, suitably extended, already provides a useful basis:

- Access requests are intercepted and thus can be documented persistently in the *knowledge base* on the *usage history*. This feature can be extended to *logging* further *useful data* about computing activities, including data that is only indirectly related to a malicious user's requests and thus is not subject to circumvention.
- In addition to deciding on each individual access request, the control and monitoring component can *audit* and *analyze* the data on request sequences and other recorded activities available. These actions are aimed at searching for *intrusions*, i.e., patterns of unexpected or unwanted behavior, and reacting as far as is possible or convenient.

Clearly, such additional secondary mechanisms cannot achieve perfection either, since otherwise we could construct perfect primary control mechanisms right from the beginning. Thus these secondary mechanisms should be designed to work *complementarily*, aiming at narrowing the gap left by the primary mechanisms.

11.1 Intrusion Detection and Reaction

11.1.1 Tasks and Problems

A *control and monitoring component* primarily permits or prohibits requests for controlled objects, aiming at enabling the needed usages of objects and at preventing undesirable accesses to system resources by all kinds of participating subjects. At least conceptually, the "needed usages" and "undesirable accesses" are precisely specified by a normative *security policy*, leading to *formal semantics* for *access decisions*.

Secondarily, and mainly complementarily, a control and monitoring component can be extended to provide *intrusion detection* and *reaction*. This functionality aims at the following:

- *identifying*, on-the-fly, activities of potentially *malicious users* who might try to employ system resources in an *undesirable way*, or *recognizing*, after the fact, that such an unfortunate event has already occurred;
- *responding* appropriately, as far as possible, once an intrusion has been detected.

Again, we need a notion of "undesirable accesses" or of "employment in an undesirable way". It could also be useful to have a notion of the positive counterpart, namely "acceptable usages" or "tolerable usages". Unfortunately, however, such notions are now much more difficult to define than in the context of access control:

- On the one hand, in order to achieve our eventual aim of automating intrusion detection, we have to come up with an algorithmic version of these notions.

- But, on the other hand, we somehow have to anticipate situations in which the precise security policy, which is used normatively for access decisions, is already on the way toward failing to fully serve its purpose.

In the following, we shall refer to such notions as an *intrusion defense policy*. There are various suggestions for determining such an intrusion defense policy as a reasonable compromise between the seemingly incompatible requirements – to be both purely algorithmic, and sufficiently flexible and adaptable to various circumstances. Below, we tentatively outline some common features of these suggestions.

For this outline, the computing system under consideration is assumed to be abstractly defined by its possible *states* and its possible *state transitions*. A *behavior* of the collection of all users, or, if identifiable, of a single user, can then be represented by a *trace*, i.e., a sequence of state transitions. For simplicity, we present our exposition mainly in terms of such traces. Alternatively, or for some variations complementarily or even necessarily, we can express anything more statically oriented in terms of system states or, better, in terms of suitably extended states that also capture at least a part of the preceding *usage history*. Thus most of the following considerations about sets of behaviors (or traces) can be translated into considerations about states as well.

As a base set, we postulate the set of "possible" behaviors. This set is intended to capture all operational options that can be exercised by a fictitious version of the computing system considered, whereby no restrictions are enforced by security mechanisms. A subset of possible behaviors is seen as *acceptable*, and another subset as *violating*. While the participating subjects continue to engage in their behaviors, i.e., request and activate state transitions, the control and monitoring component continuously checks whether the transitions are remaining within the "acceptable behaviors" or whether they are going to approach a "violating behavior". Figure 11.1 displays a rough visualization of the sets involved and their supposed inclusion relationships.

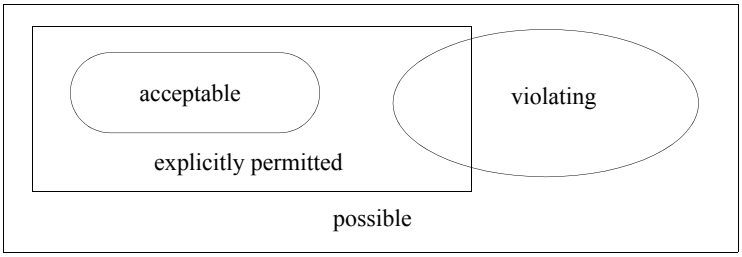

Fig. 11.1. Inclusion relationships of the relevant sets of behaviors (or of states)

Obviously, we want to be able to *separate* "acceptable" behaviors from "violating" ones. In particular, no behavior should be classified both as acceptable and as violating. Furthermore, "acceptable behaviors" should definitely be permitted according to the security policy for access control. However, at least in general, access control will allow more behaviors than just the "acceptable" ones: access control can be understood as enforcing a syntactic specification of what might happen, but only a minor fraction of the syntactically permitted behaviors can be considered *semantically acceptable*.

One of the main reasons for this troublesome discrepancy stems from the fact that access control traditionally deals mostly with *one-step* transitions only, whereas "acceptability" is assigned to *sequences* of transitions. Moreover, beyond keeping track of histories, the notion of "acceptability" typically exploits further information that is not used for access control, for example information that is related to the "semantics" of requests or "contents" of messages. Consequently, in general, the set of "semantically acceptable" behaviors is strictly included in the set of behaviors syntactically permitted by access control.

Under the precondition of a user or a collection of users having behaved "acceptably" and not in a "violating" way so far, access control features are aimed at checking a *request* for a state transition for whether it would *immediately* result in a "violating behavior", as implicitly declared by the precise security policy. And, if applicable, these features directly *prohibit* the crucial transition. As a result, the property of behaving "acceptably" and not in a "violating" way is hoped to be maintained as an invariant.

In contrast, there are two features of intrusion detection that work on different issues. The first feature explores whether an inspected state transition could possibly be a *dangerous step towards* reaching a "violating behavior". In this framework, it may well happen that a sequence of permitted requests, each of which appears to be harmless according to the precise security policy, nevertheless may end in a "violating behavior" according to the intrusion defense policy. The second feature investigates whether a "violating behavior" has *already been reached*. In both cases, appropriate *reactions* are due. Apparently, for these features, we need a notion of something like a *distance* from the already observed behavior or an anticipated behavior to the set of "violating behaviors".

Among other features, this distance should also provide indications that can be used to estimate the number and kind of permitted requests required to leave the "acceptable behaviors" or to reach a "violating behavior" from the present one. Such an estimation could be based on two kinds of presumably available knowledge:

- first, the mismatch between the design of the granting of permissions and prohibitions and the specification of "acceptable behaviors" and "violating behaviors"; and
- second, some patterns of "most likely" behaviors of friendly and malicious users. Both kinds of knowledge should be dynamically updated on the basis of past experience.

The features sketched can also be viewed as moves in a strategic game. One player is the control and monitoring component, and the adversary player is the collection of all users or some single user, as far as single users are identifiable. The goal of the control and monitoring component is to confine transitions to "acceptable behaviors" and not to reach a "violating behavior" or, even more strongly, to keep the already shown behavior "far away" from being extendable to a "violating behavior", whereas the adversary player (potentially or actually) tries the opposite, namely to complete a "violating behavior".

11.1.2 Simple Model

Figure 11.2 shows a simple model of intrusion detection and reaction. The main components form a feedback loop as follows:

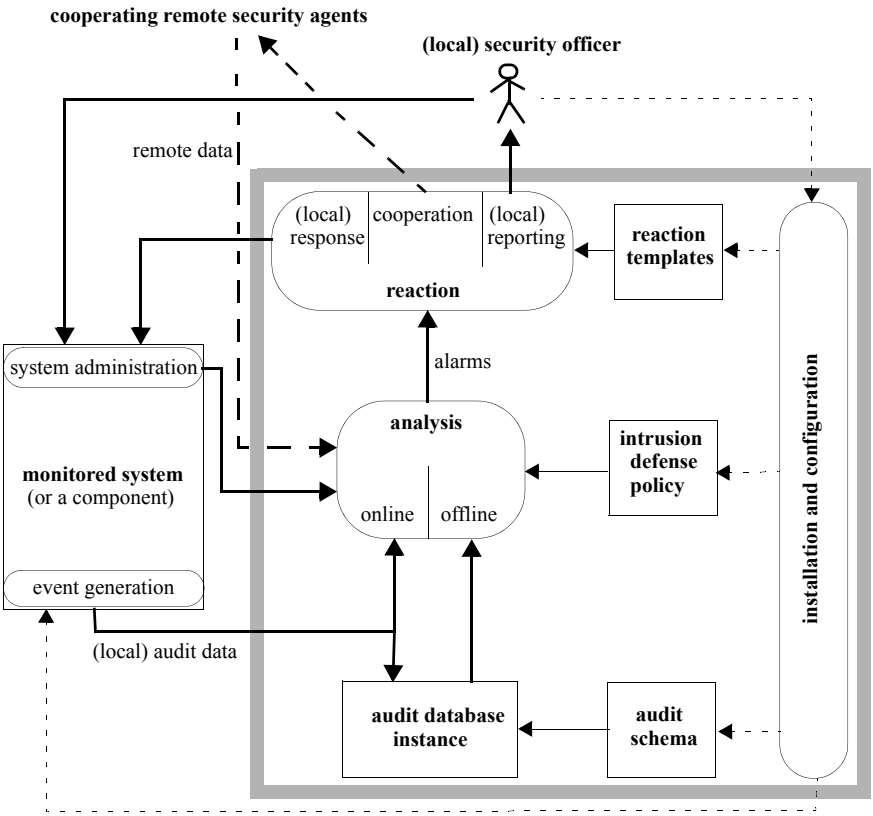

Fig. 11.2. A simple model of intrusion detection and reaction

- The *event generation*, appropriately implanted in the *monitored computing system*, delivers local *audit data* to the monitoring system, to be immediately processed online, or later offline.
- The *audit database instance* constitutes the intermediately stored audit data gathered for offline analysis.
- The *analysis* component directly inspects the currently delivered audit data in an *online* mode, or examines a larger amount of audit data *offline*. Besides the local audit data, the analysis component usually needs some basic *configuration data* about the monitored system, and, if applicable, the analysis is enhanced by *remote data* stemming from cooperating *remote security agents*. If the analysis component detects suspicious behaviors or states, it raises *alarms*.
- The *reaction* component deals with alarms in basically three ways. The reaction can be a purely algorithmically generated *local response* that intervenes in the monitored system, say by modifying some administration parameters or, in the extreme case, by totally closing down the system. Alternatively or additionally, *local reporting* to a human *security officer* is performed. The security officer, in turn, can then intervene in the monitored system, if this is considered necessary or convenient. If intrusion detection is done in *cooperation* with *remote security agents*, appropriate messages are sent out.

At initialization time or when an update is due, these main components have to be installed and configured appropriately. Therefore the local security officer provides the needed information: an *audit schema* for storing audit data, an *intrusion defense policy* for guiding the analysis efforts, and *reaction templates* for dealing with alarms. Additionally, the security officer has to implant one or several suitably configured *event generators* into the monitored system.

The *intrusion defense policy* appears to constitute the most crucial information, although it might be equally important to closely coordinate all information provided. Basically, the intrusion defense policy formally describes the intuitive notions of "acceptable" and "violating", and in this way it defines a specific *classification task* for the analysis component: on the basis of available data, the analysis component has to algorithmically decide whether – or in more sophisticated cases, to what extent – the actually observed behavior within a monitored system or the actually observed sequence of states is *evaluated* to be "acceptable" or "violating" (with respect to the given security defense policy).

Typically, the *security officer* determines the intrusion defense policy on the basis of previous experience. Thus, before the first installation or before a reconfiguration, some *learning* of the notions of "acceptable" and "violating" is required. This learning either can be done purely manually, or can be at least partially supported by algorithmic tools. In the latter case, such a *learning tool* is given *training data*, which might stem from two sources: the training data can be gained from logging suitable aspects of the specific system to be monitored, or it may be taken from publicly available or proprietary repositories of examples of "acceptable" or "violating" behavior.

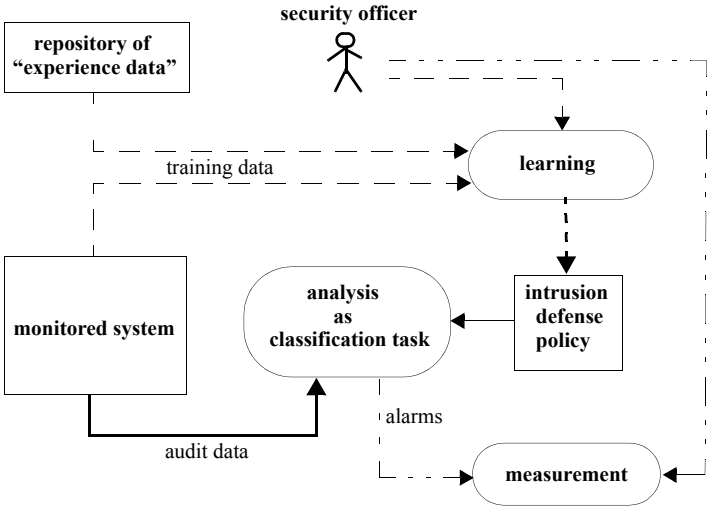

Fig. 11.3. Learning, operation and measurement for an intrusion defense policy

Additionally, the notions learnt have to be suitably formally represented in a compressed way such that the classification task of the analysis component is best supported, from the point of view, mainly, of *effectiveness* and *efficiency*. Furthermore, at an appropriate time, the actual achievements have to be measured. The *learning phase*, the *operation phase* and the *measurement phase* of an intrusion defense policy are schematically outlined in Figure 11.3.

The *effectiveness* of an analysis component guided by an intrusion defense policy is usually measured in terms adapted from the field of *information retrieval*. Basically, the measurement relates experimental items to (more or less fictitious) ideal items. The *experimental items* are taken from controlled experiments that observe the outputs of the analysis component for a given monitored system, i.e., the *alarms* raised that denote supposedly "violating" behaviors. The *ideal items* constitute a priori (more or less fictitious) knowledge about the "real status" of the behaviors underlying the experiments. For each such behavior, there are four possibilities in principle:

- The analysis component raises no alarm (it classifies the behavior as "acceptable"), and the "real status" of the behavior is indeed acceptable.
- The analysis component raises an alarm (it classifies the behavior as "violating"), and the "real status" of the behavior is indeed violating.
- The analysis component raises an alarm (it classifies the behavior as "violating"), but the "real status" of the behavior is actually acceptable. In this case, the analysis component raises a *false* alarm and, accordingly, the classification result is said to be a *false positive*.

- The analysis component raises no alarm (it classifies the behavior as "acceptable"), but the "real status" of the behavior is actually violating. In this case, the analysis component fails to generate a *correct* alarm and, accordingly, the classification result is said to be a *false negative*.

Clearly, one would like to prevent both of the last possibilities, where the analysis fails. In practice, however, there is usually a trade-off. Roughly speaking, in order to avoid false positives, the analysis has to justify alarms by very strong reasons, thereby potentially allowing more false negatives. Conversely, in order to avoid false negatives, the analysis has to raise alarms not only for strong reasons but also in questionable cases, thereby potentially allowing more false positives.

Efficiency is mandatory for online analysis. In this case, basically, the analysis component has to perform the classification task in linear real time. This requirement suggests that one should internally represent an intrusion defense policy as a pair of suitably enhanced finite automatons or by means of closely related computing abstractions, to be used as *recognizers* for "violating" and "non-acceptable" inputs.

11.2 Signature-Based Approach

The signature-based approach contributes to representing *violating behaviors* and constructing a corresponding recognizer. More specifically, long-term observation and evaluation of violating behaviors have led to a large collection of samples of known attacks on a computing system. Intuitively, a *signature* is a formal representation of a known *attack pattern*, preferably including its already seen or merely anticipated variations, in terms of generic *events*, instances of which can be generated by the *event generation* and reported in the form of *audit data*.

More formally, in the simplest case, a *signature* σ is given as a finite time-ordered sequence of abstract events taken from a finite event space Σ; accordingly, we can consider a signature as a word over the event space seen as an alphabet, i.e., $\sigma \in \Sigma^*$. The *event space* is determined by the layer where the event generation is located. Typically, the event generation is coupled to an internal interface of the monitored system where requests or more general messages are easily interceptable and can be inspected in some way. For example, in the layer of an operating system, events might be *system calls* to the kernel; in the layer of a network system, events might be *packet moves*; and in the layer of some application, events might be *method invocations*. Depending on the actual location, intrusion detection systems are sometimes classified as *host-based* or *network-based*.

Still considering the simplest case for the sake of conciseness, the *analysis component* has two inputs:

- a fully known signature $\sigma \in \Sigma^*$, as the intrusion defense policy;
- an eventwise supplied behavior $\beta \in \Sigma^\infty$, as the (ongoing) recorded activities within the system, given as a (possibly) infinite word of audited events.

The basic *classification task* is then to determine whether and where "the signature σ *compactly occurs* in the behavior β", i.e., to find *all* position sequences for β that give the signature σ such that each prefix cannot be completed earlier (or some similar property holds). Accordingly, the analysis component must provide a corresponding recognizer, which should raise an alarm for *each* such compact occurrence of σ in β. An abstract example is shown in Figure 11.4.

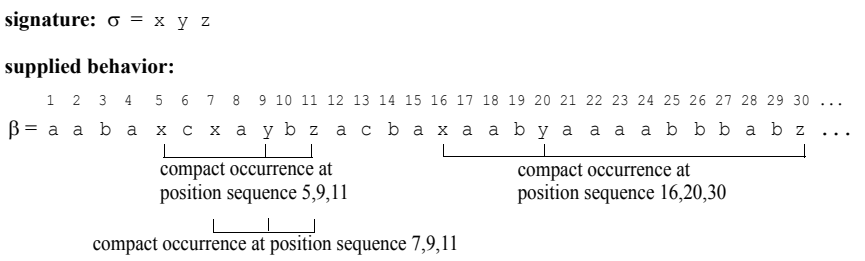

Fig. 11.4. The compact occurrences of a signature in a supplied behavior

Clearly, an actual analysis component should be much more sophisticated, dealing at least with the features sketched in the following.

In principle, a compact occurrence of the signature σ can be spread widely. Accordingly, while checking whether "σ compactly occurs in β", the recognizer must memorize and handle each detected occurrence of a prefix of σ in β until the prefix has been completed. In practice, in particular for the sake of efficiency, one would like to "forget" non-completed prefixes after a while.

A basic concept for doing so is to declare explicit *escape conditions* for terminating an initialized occurrence. As a simple implementation, one can employ a *sliding window* of some appropriate length *l* for the behavior β, and to search only for occurrences of the signature that fit into one window position. As a trade-off, all more widely spread occurrences are missed in the hope that they do not reflect an "actual attack". For the example shown in Figure 11.4, if we restricted the window to a length of 10, then the occurrence at position sequence 16,20,30 would be ignored.

Depending on the intercepting interface, in most cases events appear to be structured and composed of several items. As a typical example, a *parameterized event* $e[\ldots, A_i : v_i, \ldots]$ might consist of an event type *e* and a list of specific attribute–value pairs $A_i : v_i$. In the case of system calls, *e* would be an operator, and the $A_i : v_i$ would denote the arguments. Correspondingly, a *parameterized signature* would be a sequence of parameterized events, where some or all values might be replaced by variables.

When one is searching for the compact or otherwise suitably restricted occurrences of the signature in the supplied behavior, the values in the signature have to

match the audited values, whereas the variables in the signature are *bound* to the pertinent audited values. Thus each detected occurrence of a prefix of the parameterized signature is linked to a *binding list* for variables, and once a variable is bound for a detected prefix, the binding also applies to the tail of the signature, i.e., audited values there have to match the bound values. Notably, the recognizer must maintain a partially instantiated signature instance for each detected occurrence of a prefix. Figure 11.5 shows a simple abstract example.

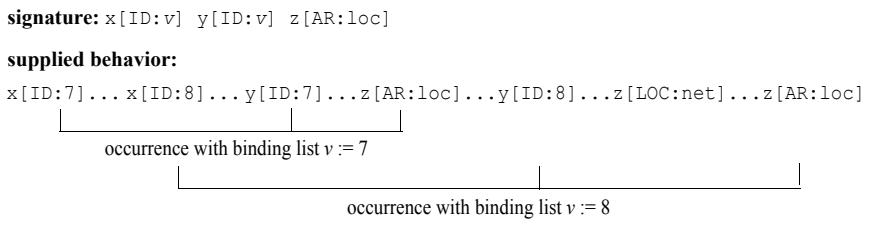

Fig. 11.5. Detected occurrences with binding lists

In order to capture variations of an attack, several closely related event sequences might be represented concisely as a directed acyclic graph (dag) built from events. Given a signature in this form, the recognizer has to search for a compact occurrence for any path from some start event to some end event within the supplied behavior.

Finally, in general, "violating" behavior is described by hundreds of known attacks, and thus by a large number of signatures that the analysis component has to handle in parallel.

Summarizing, and subject to many variations, the signature-based approach consists of the following steps:

- In the *learning phase*, an administrator – possibly assisted by a tool – models the known attacks by an intrusion defense policy, specified as a set of parameterized dag-like signatures using some appropriate formal language. Then a suitable tool transforms the specified policy into an integrated collection of recognizers that become a part of the analysis component.
- In the *operation phase*, the recognizers instantiate the given signatures according to the prefixes and their bindings for variables, as detected in the supplied behavior within a sliding window, and raise an alarm whenever an instantiation has been completed.
- In the *measurement phase*, the effectiveness might be improved by revising or refining the policy or by enlarging the length of the sliding window, and the efficiency might be improved by optimizing the collection of recognizers and by diminishing the length of the sliding window. To deal with the apparent trade-off regarding the sliding window, its length might also be adapted dynamically.

11.3 Anomaly-Based Approach

The anomaly-based approach contributes to representing *acceptable behaviors* and constructing a corresponding recognizer for non-acceptable behaviors. More specifically, with some precautions, a large collection $N \subset \Sigma^*$ of actual behaviors, i.e., sufficiently long event sequences generated by the event generator as audit data in the past, is supposed to constitute a representative sample of "acceptable" behaviors. Clearly, in particular, the precautions have to ensure that there are no hidden attacks, and that the sample covers the full range of the real users' activities.

Roughly speaking, a recognizer is then constructed that is trained to let each collected behavior $\sigma \in N$ pass, and sufficiently similar behaviors as well, but raises an alarm for all other behaviors. Intuitively, the former behaviors are seen as supposedly *normal* and thus "acceptable", whereas the latter behaviors are seen as deviating from the norm and thus *anomalous*, giving rise to an alarm.

Briefly outlined, and subject to many variations, the anomaly-based approach thus consists of the following steps:

- In the *learning phase*, an administrator gathers a sample set N of supposedly normal behaviors, and selects a length l of a *sliding window* on the behaviors. Then, a suitable tool for *machine learning*, for example a neural network, is employed to construct an efficient finite-automaton-like recognizer for anomalous parts of behaviors.
- In the *operation phase*, the recognizer searches for anomalous parts in the supplied behavior within the sliding window, raising an alarm whenever such a part has been detected.
- In the *measurement phase*, the effectiveness might be improved by adapting the sample set N or by enlarging the length of the sliding window, and then reconstructing the recognizer. The efficiency might be improved by optimizing or even smoothing the recognizer, for example by letting some additional behaviors pass or by diminishing the length of the sliding window, again facing a trade-off with effectiveness.

11.4 Cooperation

In a distributed system, each site might apply intrusion detection and reaction for its own purposes. Additionally, the administrators of the various sites might agree to cooperate in order to more effectively protect against attacks spanning more than one site. A *distributed denial-of-service* (*DDoS*) *attack* is an example, where some malicious participants try to flood some sites with too many requests such that these sites' expected services degrade or even are no longer available.

Any *cooperation* requires making some of the local information available to the remote partners. Figure 11.2 indicates a natural approach: some of the local alarms are forwarded to the cooperating agents and, vice versa, the communicated remote alarms are treated as further input data for the analysis component. Basically, the

analysis component then has to perform the additional subtask of correlating its own (more) *elementary alarms* with the received (more) *elementary alarms* in order to produce a (more) *complex alarm*. A complex alarm should inform the reaction component about the overall situation regarding a suspected attack, and thereby facilitate an effective response.

Evidently, although producing useful local elementary alarms is already a tedious and often only unsatisfactorily accomplished task, correlating them into informative complex alarms is even more challenging. Nevertheless, standardization has laid a foundation, and recent work has outlined a heuristic method for achieving this ambitious goal. In the following, we briefly sketch what the steps of such a method are intended to achieve:

- *Normalization* maps local alarms to a common format with common semantics and mandatory features, for example the attack's start time, end time, source and target.
- *Fusion* discards obvious duplicate alarms generated by different sites observing the same activity.
- *Verification* identifies irrelevant alarms and false positive alarms, for example by judging a current alarm regarding a knowledge base of "known harmless situations".
- *Thread reconstruction* gathers together subsequent alarms describing attacks that originate from an attacker residing at the same source, on the same target.
- *Session reconstruction* correlates alarms that describe events on the network and in a host, for example by exploiting knowledge about how network transport protocol port numbers are mapped to the identifiers of the processes on the host listening on the respective ports, and knowledge about the parent–child relationships of the processes on the host.
- *Focus recognition* integrates alarms describing attacks where a given attacker attacks many targets or where a given target is attacked by many sources, for example distributed denial-of-service attacks.
- *Multistep correlation* combines alarms suspected to constitute a complex attack, for example on the basis of a suitable collection of complex signatures.
- *Impact analysis* and *alarm prioritization* determine the suspected effect of an attack in order to prioritize the respective alarm accordingly, for example on the basis of an asset database, which measures the importance of the services offered and models their dependencies.

11.5 Bibliographic Hints

Amoroso [15], Bace [24] and Marchette [333] have authored recent books about intrusion detection. Current work on cooperation is summarized and extended by Ning/Jajodia/Wang [374], emphasizing the communication needs of an "abstraction-based approach", and by Kruegel/Valeur/Vigna [301], elaborating the heuristic

correlation method outlined above. Taxonomies that survey the various approaches have been proposed by Debar/Dacier/Wespi [157] and Axelsson [22].

Examples of the signature-based approach are presented by Mounji/et al. [359], Ilgun/Kemmerer/Porras [264, 265], Lindquist/Porras [323], Eckmann/Vigna/Kemmerer [181, 491], Lee/Stolfo [312] and others. Castano/Fugini/Martella/Samarati [118] give an introduction to some early approaches to anomaly detection. More specific work is reported by Forrest/Hofmeyr/Longstaff [214], D.Denning [165], Javitz/Valdes [278], Ko/Ruschitzka/Levitt [294], Lane/Brodley [310], Wespi/Dacier/Debar [498], Michael/Ghosh [351] and others. Cooperation is considered by Huang/Jasper/Wicks [261], Bass [31], Ning/Xu/et al. [372, 373] and others.

Axelsson [23] argues that the false-alarm rate limits the effectiveness of intrusion detection in principle. McHugh [340] critically examines the actual achievement of intrusion detection systems. Iheagwara/Blyth/Singhal [263] report on a more recent performance study for high-speed networks.

Monitoring raises many more specific problems. As an example, Julisch [287] treats the problem of identifying the root causes underlying the bulk of reported alarms. As another example, Fischer-Hübner/Sobirey/et al. [454], Büschges/Kesdogan [112], Lundin/Jonsson [328], Biskup/Flegel [67, 68, 208, 209, 210], Xu/Ning [504] and others investigate the problem that individuals might have conflicting security interests, for example, on the one side, accountability for the sake of pursuing attackers, and on the other side, non-observability or anonymity for the sake of privacy.

http://www.springer.com/978-3-540-78441-8

Security in Computing Systems
Challenges, Approaches and Solutions
Biskup, J.
2009, XXVIII, 694 p., Hardcover
ISBN: 978-3-540-78441-8