Evaluating the Impact of Automated Intrusion Response Mechanisms

Thomas Toth and Christopher Kruegel
Technical University Vienna, Austria
Institute for Information Systems
Argentinierstrasse 8, A-1040 Vienna, Austria
{ttoth, chris}@infosys.tuwien.ac.at

Abstract

Intrusion detection systems (IDSs) have reached a high level of sophistication and are able to detect intrusions with a variety of methods. Unfortunately, system administrators neither can keep up with the pace that an IDS is delivering alerts, nor can they react upon these within adequate time limits. Automatic response systems have to take over that task. In case of an identified intrusion, these components have to initiate appropriate actions to counter emerging threats. Most current intrusion response systems (IRSs) utilize static mappings to determine adequate response actions in reaction to detected intrusions. The problem with this approach is its inherent inflexibility. Countermeasures (such as changes of firewall rules) often do not only defend against the detected attack but may also have negative effects on legitimate users of the network and its services. To prevent a situation where a response action causes more damage that the actual attack, a mechanism is needed that compares the severity of an attack to the effects of a possible response mechanism. In this paper, we present a network model and an algorithm to evaluate the impact of response actions on the entities of a network. This allows the IRS to select the response among several alternatives which fulfills the security requirements and has a minimal negative effect on legitimate users.

1. Introduction

The constant increase of attacks against networks and their resources causes the necessity to protect these valuable assets. Although well-configured firewalls provide good protection against many attacks, some services (like `HTTP` or `DNS`) have to be publicly available. In such cases, a firewall has to allow incoming traffic from the Internet to these services without restrictions. As a matter of fact, the programs implementing these services are often complex and old pieces of software. This inevitably leads to the existence of programming bugs which can be exploited by skilled intruders.

Intrusion detection systems (IDSs) are security tools that are used to detect traces of malicious activities which are targeted against networks and their resources. IDSs are traditionally classified as anomaly or signature based. Signature based systems like Snort [10], STAT [12] or NetSTAT [13, 14] act similar to virus scanners and look for known, suspicious patterns in their input data. Anomaly based systems watch for deviations of actual from expected behavior and classify all 'abnormal' activities as malicious.

As signature based designs compare their input to known, hostile scenarios they have the advantage of raising virtually no *false alarms* (i.e. classifying an action as malicious when in fact it is not). For the same reason, they have the significant drawback of failing to detect variations of known attacks or entirely new intrusions.

Because of the ability to detect previously unknown intrusions a number of different anomaly based systems have been proposed. Depending on their source of input data, they are divided into host based and network based designs.

Host based anomaly detection systems can focus on user or program behavior. User profiles are built from login times and accessed resources (e.g. files, programs) or from timing analysis of keystrokes [4, 1]. Unfortunately, user behavior is hard to predict and can change frequently. Additionally, such systems cannot react properly when network services get compromised as no single user profile can be associated to a daemon program.

As a consequence, the focus was shifted from user to program behavior. The execution of a program is modeled as a set of system call sequences [6, 5] which occur during 'normal' program execution. When the observed sequences deviate from the expected behavior the program is assumed to perform something unintended, possibly because of a successful attack (e.g. buffer overflow).

Most network based anomaly detection systems [8, 9] only model the flow of packets. The source and destination IP addresses and ports are used to determine parameters like the number of total connection arrivals in a certain period of time, the inter-arrival time between packets or the number of packets to/from a certain machine. These parameters can be used to reliably detect port scans or denial-of-service (DOS) attempts. In [7], an anomaly based method has been introduced that analyzes the packet payload to identify malicious content used in exploits.

Intrusion response systems (IRSs) take over after signs of an intrusion are identified and either record the attack or attempt to actively counter it. Although IRSs are tightly coupled with the ID systems themselves and are as important as these in defending against threats, not much research effort has been put into their study. Therefore, intrusion response, in most cases, remains a manual process which has to be performed by the system administrator [2].

Current intrusion response systems can be divided into notification, manual response and automatic response systems.

The majority of IRSs operate as notification systems, which means that they simply display or forward output delivered by the IDS (e.g. incident data) to the system administrator. Usually, urgent notification is realized via e-mail or text message services over a mobile phone.

Manual IRS allows the administrator to manually launch countermeasures against a detected intrusion by choosing from a predetermined set of response mechanisms. This might allow the administrator to harden the firewall or to change router configurations to disallow malicious traffic. Manual response can help to cut off denial-of-service attacks [11] but is also beneficial in the case that the system detects a hacker who has just obtained access to a certain host. Such systems support an administrator by offering ready-to-apply reconfiguration mechanisms in order to quickly secure the system. Nevertheless, a person has to determine which methods are appropriate.

The two categories listed above are not proactive in countering an intrusion. Even when signs of an intrusion have been detected, countermeasures are not triggered automatically and defending the network remains a task for the system administrator. This opens a time window of vulnerability between the point when the intrusion has been detected and the point when the first countermeasure is launched. The size of this time window can range from seconds to hours (e.g. during night times or weekends).

According to [3], the success rate of an intruder rises with the time he can work undisturbed. This interesting study reports that a skilled attacker can perform an intrusion with a 80% success rate if he is given 10 hours time before any response is launched.

In contrast to the two approaches shown above, automatic response systems attempt to choose appropriate countermeasures without human intervention. This allows to dramatically reduce the size of the vulnerability window. Most current systems implementing automatic response mechanisms use simple decision tables to determine how to react in the case of identified attacks. More sophisticated variants such as Cooperating Security Managers [15] and Emerald [9] apply expert systems to perform that task.

Decision tables are an inherently inflexible mechanism because they allow only a static mapping between intrusions and the corresponding response actions and do not take possible negative side effects of countermeasures into account.

In order to provide optimal responses, all possible situations would have to be encoded in that static table. As this is clearly infeasible (often situations are not known to the person creating that table), one has to fall back to default mechanisms when encountering new situations.

Additionally, it is only feasible to build the static mapping table for small networks. In such cases, an operator can perform the analysis of (all) threat scenarios and determine the table entries manually. If the network is large and the network services become more and more intertwined, hidden dependencies cause the generation of the mapping table to be more and more cumbersome and error prone.

Another severe problem are false positives (i.e. the IDS raises an unjustified alarm). When the corresponding countermeasure in case of an incorrect alert is executed by the IRS, legitimate users may be negatively effected. Consider a firewall reconfiguration which prohibits incoming connections to a certain service which is needed by users outside the network (a nightmare for e-commerce sites).

Emerald and CSM mitigate the drawbacks of a static mapping table in case of a false alarm by including severity and confidence metrics into their response process. The confidence metrics describes the belief of the system that detected evidence is the indication of a real intrusion. The severity metrics rates all response mechanism according to their (potential) negative side effects on legitimate network operations. Measurements with a high severity level are only allowed when the confidence in an attack is high enough. A quite similar idea is described in [2] where the determination of an appropriate response function is done with the consideration of the expected false positive rate of the underlying IDS. If the IDS is expected to have a low false positive rate, the IRS is more likely to invoke severe response actions.

Such metrics work well when the response action does not interfere with many other services or when the response does not last longer than a reasonable small amount of time. Unfortunately, responses with long-term effects may seriously hamper regular users from performing their tasks. Current response systems do not take normal operation into account and have no notion of dependencies of services between each other. Large networks contain many hosts, each running several services which might require other services, making a thorough manual analysis very difficult. The dependencies among the services are usually very complex, involving a number of different protocols. While some services are critical for remote services or users, others may be unavailable for some time without causing problems.

The proposed severity and confidence metrics are a first step into the right direction, as they attempt to estimate the negative side effects of response mechanism. Nevertheless, they do not include the analysis of the actual network topology or services. In addition, these mechanisms are not flexible in assigning priorities to certain services according to their importance for regular tasks. While the webserver of an e-commerce site has to remain operational at all times, this might not be necessary for normal companies. The cost of having a service off-line due to a response action might be higher than the threat of the attack. Current models do not allow to model such requirements.

In this paper, we present a network model to evaluate the effect of intrusion response mechanisms to the operation of network services, thus enabling the IRS to choose the best alternative from a set of possible alternatives. Our model takes into account the network topology and the dependencies between different entities to capture the consequences of responses more accurately. Based on this model, an evaluation function can estimate the impact of various responses and select the one with the expected minimal negative consequences.

We define a modeling language to specify the resources with their dependencies as well as response actions and their impact on the availability of resources. By including user requirements into our network model, different sites can tailor the responses according to their needs.

The next two chapters discuss our requirements and the network model itself. Section 4 describes the evaluation function to calculate the effects of responses, while Section 5 presents details about the implementation. Then we provide some measurements obtained from our prototype. Finally, we outline further research and conclude.

2. Model Requirements

This section elaborates on the requirements that we have identified for our network model to be able to accurately calculate the effects of responses. In the following sections, we focus on responses that can reconfigure the firewall, enable or disable user accounts and modify the status of processes running on a host (i.e. restarting network services, terminating malicious programs).

- **Flexibility** The model has to be able to cope with different network topologies and must be able to express the dependencies among resources themselves and between services and users. As there should be no artificial restrictions, our model should not have to rely on simple mapping tables for calculating response effects. Instead, the actual situation of the network needs to be reflected in the model to be able to determine the effects of response actions accurately.

  Not all resources or users have the same relevance for the operation of the network – this fact has to be expressible in our model. A resource that is only utilized by low priority entities is obviously less important than one used by a mission critical entity. The importance of a resource can vary dynamically - even by the time of day.

- **Dynamic Model** The model has to be dynamic to be able to track changes in the environment (caused by response actions). A reconfiguration of the firewall has to be reflected in the model, as well as changes in the availability of services due to their (de)activation by a response.

- **Efficiency** In order to be useful, the evaluation function needs to be evaluated quickly. IRSs have to respond fast in order to keep the time window of vulnerability small. The model should make the design of an efficient evaluation function possible.

- **Ease of Use** In a large network, there are many dependencies between different entities. In order to make administration of our proposed system easy, not all of them should have to be entered explicitly by an administrator. The majority of dependencies can be determined automatically by the analysis of transitive relations and the network topology. Only basic relationships at a high level (e.g. this host needs access to a `DNS` server) should have to be specified.

  The model should be intuitive and comprehensive in the sense that it resembles the facts of the real world. A smooth integration of entities and their dependencies is necessary to achieve this goal.

- **Minimization of Negative Impact** The model should be able to help the IRS determine which response action to use. In the case that more than one response action is available, the one which has the least negative effects on the whole system should be chosen.

3. Network Model

The following section introduces our network model that is used to calculate the effects of response actions. First, the elements, which are included in our model, are identified. The basic elements, as explained in more detail below, are services provided by hosts, users of the network, the underlying communication infrastructure and firewall rules that are currently in effect.

Then, we specify how direct and indirect dependencies between entities are represented. After that the algorithm that operates on this model to determine the effects of responses is explained.

3.1. Modeled Elements

Networks are complex structures that include many elements which are heavily related and dependent on each other. For our model, the following elements are relevant (in the following explanation, system users and resources are together referred to as *entities*).

- **Resources** Resources describe network services offered by hosts. They build the basic building blocks of our network model and can depend on other resources to various degrees. A resource is a network service which is provided by a process on a host. Examples of resources/services are `DNS`, `NFS`, `NIS`, `HTTP` or `FTP`. A process provides resources to others by listening on a predefined port to which other processes or users can submit requests. Requests are processed and a reply is sent back to the originator of the query. In our model, only resources that are used by other entities have to be included, and processes running at a host without providing services to external entities are not considered to be resources.

- **System Users** Users have to perform their tasks by utilizing the provided resources, therefore they have to be part of the model as well. Users can assign different levels of importance to resources.

- **Network Topology** The network topology has an important role for the evaluation process because it determines the communication framework utilized between different resources.

- **Firewall Rules** The installed firewall rules effect the availability of resources/services of the protected network. Dependencies between two resources located in the same subnet are not affected by response effects that modify firewall rules. In the case that the communication path from one resource to another leads through a firewall, its rules obviously influence the availability of that resource.

3.2. Entity Dependencies

This section explains the different types of dependencies between entities.

Definition: An entity, which needs a service that is provided by another entity to be fully operational, is called *dependent* on that entity. The relation between these two entities is called a *dependency*. Among the different entities which are distributed over the hosts of a network, there are many dependency relationships. While some entities do not need other ones to be fully operational (stand alone machine without network connections for running editors), most do (e.g. a mail-server to let the user send and receive e-mail, a `DNS` server to allow `DNS` name resolution or an `HTTP` server for accessing web pages).

An entity is considered to be *available* for a dependent one if (a) communication between both is possible and (b) the entity providing the service is functional (i.e. the process providing the service is running). Communication between two entities is possible if (i) there is a route provided by the underlying network topology between both and (ii) all hosts on the route permit the traffic between them.

Figure 1 shows the dependencies between the two users Anne and Customer, two `HTTP` servers and a `DNS` as well as an NFS server. The entities are expressed as annotated boxes while the dependency relationships are expressed as arrows.

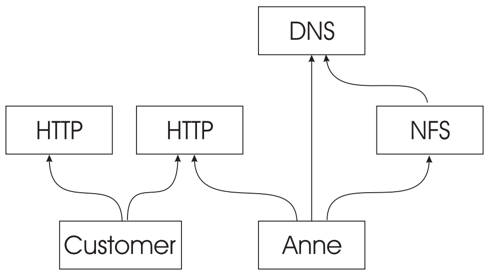

Figure 1. Resource Dependencies

Definition: The dependency between two entities might be *direct* or *indirect*.

A direct dependency is a dependency that is given to the model manually (via configuration files - the grammar to model direct dependencies as well as an example are given in the Appendix). These are the dependencies of entities on various services. An example would be a user that uses the `DNS` service to resolve `DNS` names (e.g. user Anne in Figure 1).

As described above, we consider the network topology and the firewall rules as part of our network model. While the network topology is the glue between the resources by providing communication paths, the firewall rules can be viewed as a method for imposing constraints on these paths by (dis)allowing certain traffic. The network topology and the firewall rules introduce new artificial dependencies between entities and their needed resources. This is caused by the fact that information exchange has to take place over routers and must be allowed by all firewall rules that are in effect on the communication path (i.e. host personal firewalls, router firewalls). These artificial dependencies are called *indirect dependencies*. Indirect dependencies are determined automatically by analyzing the network topology (which is encoded in routing tables) as well as firewall rules. Indirect dependencies are identified and evaluated during the phase in which the propagation of network traffic is simulated. They can be seen as a precondition for fulfilling a direct dependency. All the indirect dependencies that are imposed by nodes being on the path between the depending hosts have to be fulfilled to fulfill the direct dependency. If a node creates an indirect dependency that is not fulfilled (e.g. the packet is filtered at this node), a direct dependency relying on it can never be fulfilled too. Indirect dependencies would be immediately introduced in the example shown in Figure 1 if the `DNS` server is located in a different subnet than Anne or if personal firewalls are installed on any of the just mentioned hosts.

The example in Figure 2 shows a network that consists of four subnets and the external Internet (in the top left corner). The direct dependencies are identical to those shown in Figure 1. However, note the indirect dependencies between the gateways that connect the different subnets. In this figure, routing information as well as firewall rules are omitted for the sake of simplicity.

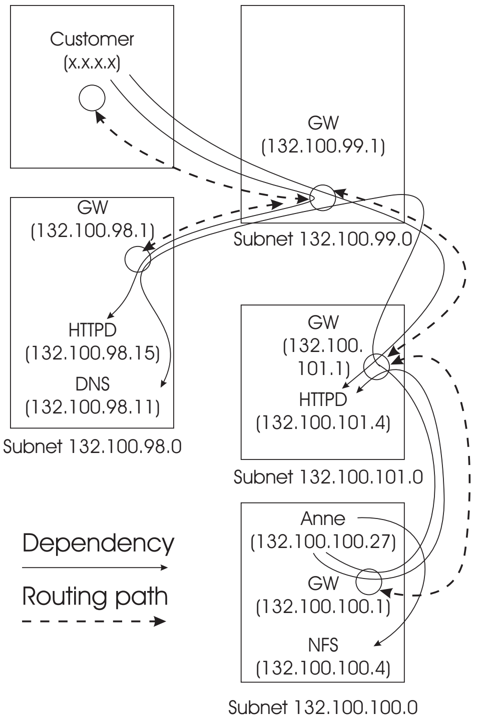

Figure 2. Topology and Entity Dependencies

4. Impact Evaluation

A *response action* is a set of operations that can be utilized to avert a certain threat. The basic operations, called *response items*, are basic steps like installing or removing firewall rules, killing and restarting processes or user account en-/disabling. Response actions are initiated by the IRS in response to an intrusion which is detected by an IDS. Because a number of different response actions might achieve the desired result, it is the task of the IRS to choose the one with the least impact. The determination of the impact (or effect) of response actions is done with the help of the current network model by an *impact evaluation function*. The set of response actions that have already been applied (and which have lead to the current state of the network) are called *response configuration*.

A single response action can affect entities either directly or indirectly. A direct effect is witnessed when a needed service becomes unavailable (e.g. due to stopping of services or disabling of user accounts). An indirect effect is experienced when the direct effect on one entity reduces the service that this entity can provide to another one, thereby affecting entities that are unrelated at first glance but which are indirectly dependent.

If a response action hits an entity, it will not be able to perform its task with the same quality or speed as before. The *degree* of a dependency describes in how far the operation of an entity is affected if the resource, which it depends on, is no longer available. The introduction of a degree of dependency can be best motivated by the following example. Consider a user that uses his machine mainly to surf the Internet. In our network model, the entity (representing this user) will depend a lot on the availability of the `DNS` server and the `HTTP` server (dependencies with high degrees), but not on the NFS server. On the other hand, a user editing files on the remote NFS machine will mainly need this service to accomplish his work (dependency with a high degree).

An entity will usually depend on several resources in the network. These relationships do not necessarily have to be trivial. For some entities, it is sufficient to have access to at least one of a set of (similar) services (called 'OR-dependency') while others need access to all of them (called 'AND-dependency'). Our model is capable of expressing both types of relationship as well as combinations of them.

The following example describes a user with her dependencies. User Anne requires access to the NFS server as well as to the `HTTP` server. Additionally, she needs access to one of the two domain name servers DNS1 and DNS2. These relationships can be denoted in a *dependency tree* as shown in Figure 3.

The *capability* $c(r)$ of an entity r is a value ranging from 0.0 to 1.0 and describes in how far a resource can perform its work given the current response configuration, compared to

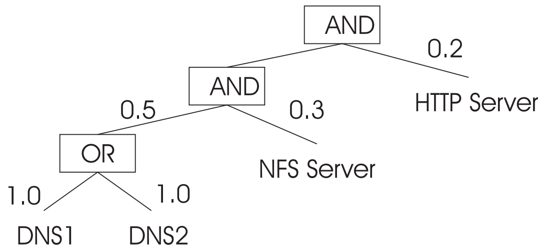

Figure 3. Dependency tree

the situation where all needed resources are available. The calculation of this value is based on the underlying network model given its current state (with services that might have already been disabled) and the entity's dependency tree. When the capability value is determined for an entity, the communication paths to all the resources that it depends on are examined. This allows the evaluation function to take the current routing and packet filter (firewall) rules into account. When specifying a dependency tree for a certain entity, one must make sure that the capability of the entity is 1.0 when all resources are available. The following paragraphs explain how a capability value is determined for an entity.

4.1. Capability Calculation

- **Entity does not depend on other entities:** In this case only the current condition of the entity determines its capability.
  - Entity provides service: In this case the capability is set to 1.0.
  - Entity does not provide service: The capability is set to 0.0.
- **Entity depends on other entities:** Here, a recursive algorithm that performs a depth-first search on the dependency tree is utilized to determine its capability. The types of the nodes of the dependency tree determine which formulas are used to aggregate the capability values obtained from the subtrees below. The intermediate nodes of the graph can be either of the type `AND` or `OR`, while the leafs represent entities. $func(left)$ and $func(left)$ denotes the capability of the left/right link of a node, multiplied with the dependency degree. c describes the capability value that is derived for the intermediate node. The items of the following list denote the different node types.
  - **Entity** In this case, the value c for the leaf node is set to the current capability of this entity.
  - **OR** $c = max(func(left), func(right))$
  - **AND** $c = func(left) + func(right)$

To make this evaluation process efficient, no cyclic dependencies may be present which makes it possible to determine a fixed evaluation order in which each entity has to be evaluated only once. The order can simply be generated through expanding all dependency trees. During this operation, all leaves in a dependency trees are substituted with their dependency trees. No cyclic dependencies are allowed, therefore the trees have a bounded size. The evaluation order is then determined by the trees themselves. The elements at the bottom of the trees have to be evaluated first, and then the ones one level up in the tree can be evaluated.

The *capability reduction* $cr(r)$ of a resource r is the value $1 - c(r)$.

The *penalty cost* for an entity is a value representing the cost when this entity becomes unavailable. The penalty-cost $p(r)$ of a resource can be calculated with the formula below.

$$p(r) = cr(r) * penalty \tag{1}$$

where the penalty is a user-defined constant that reflects the importance of an entity.

As an example, consider the penalty for the web server of an e-commerce site. Here, the penalty will be extremely high as it is necessary to have a running web server to stay in business. On the other hand, the penalty for the same service (web service) of a normal company will be usually lower. Downtimes are clearly acceptable in that case.

4.2. Cost Optimization

Consider the situation where a threat or an intrusion is identified. There are often a variety of possibilities where and how a response action can be deployed. Nevertheless, the choice of the actual response item or response locations can have tremendous impact on the usability of the whole system.

Response actions that effect the system's security in similar ways (i.e. that counter a certain threat) are called *alternatives*. Ideally, a response system can determine a number of adequate response actions which all provide the same level of security. In this case, the response action with the least impact should be chosen. Assume a situation in which a denial-of-service (DOS) attack against the `HTTP` server `132.100.101.4` in Figure 2 is detected. The response system might then decide to prevent outside traffic to this machine either at the gateway to the Internet or at the gateway located on the same subnet as the `HTTP` server.

Usually, choosing the best alternative is a difficult task. But by determining the impact (i.e. penalty cost) of a response action on all entities of the network (using our model and evaluation function), the one with the lowest negative effect can be selected.

Minimizing the penalty cost of new response actions: This can be easily done when the IRS determines and presents appropriate response actions to our system. Each response action is simply added temporary to the model, the model is evaluated and the overall penalty cost (which is the sum of all penalty costs) is determined. The response action with the lowest penalty cost can then be chosen by the IRS, as shown in Figure 4. Obviously, the actually launched response actions have to be added permanently to the model (more precisely, to the response configuration) in order to keep it up-to-date with the actual state.

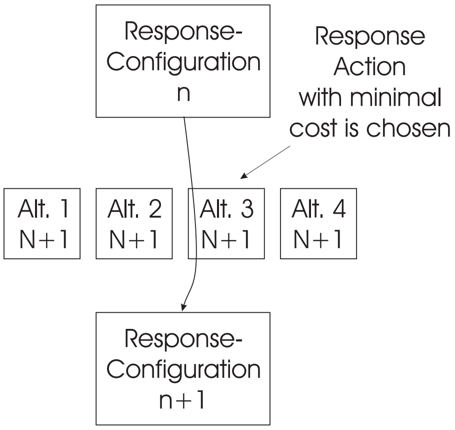

Figure 4. Response Configurations

Minimizing the overall penalty cost: When the response with the least impact is chosen in every step (the local optimum), the overall response configuration might not be globally optimal (see the example from the Appendix). Finding a globally optimal response configuration is not trivial and a number of previous actions might have to be 'rolled' back. All alternative combinations of response actions (stored in the history) have to be re-evaluated to find the scenario which has the least overall penalty cost.

5. Implementation

We implemented a prototype of the network model and the evaluation function on Linux 2.4.18 using C. The parser to process the grammar used to specify direct dependencies (as shown in the Appendix) has been realized using `flex` and `bison`. The routing tables from all relevant routers of the network as well as the firewall rules are imported into the model at startup time.

In addition, the paths from each entities to all entities that it depends on are pre-calculated. These paths are stored as a list of hosts where only the permissions (rules) of intermediate firewalls have to be checked. This is possible, because we assume that routing tables remain unchanged during regular network operation. Also the order in which entities have to be evaluated is pre-calculated using the method explained in Section 4.1.

After this initialization phase the prototype is then able to process the requests of an external IRS component or requests stored in files. We provide an API to the IRS component to evaluate the effects of response actions and to modify and update the response configurations.

Usually, the model is re-evaluated completely (i.e. the capabilities for all entities are recalculated) when a new response is examined. The evaluation of the model can be optimized, however, when only a simple firewall rule should be added. In this case, only a small part of the whole model is affected by the response action, and therefore only entities which have dependencies that lead over the modified firewall have to be evaluated. The rest of the model remains untouched and needs no re-evaluation.

As mentioned before, we currently support the update of firewall rules, the killing and restarting of processes and the disabling/enabling of user profiles at hosts. These are the most important long-term response actions and our network model can be utilized to calculate their impact.

6. Evaluation

The presented model allows us to determine the effects of firewall and process based intrusion responses. We proposed an evaluation mechanism that utilizes external information describing dependencies between resources in a network as well as their importance to different users to obtain an impact value for different response actions. In this section the computational complexity of the evaluation is investigated.

6.1. Theoretical Considerations

In order to evaluate the efficiency of this model, we have to investigate the different operations that are involved.

An optimized data structure is built during the initialization phase. Each resource contains information about the resources that it depends on their respective dependencies. The fixed evaluation order of the entities and the precalculated paths between entities allow a fast evaluation of the complete model.

The insertion and removal of a temporary response actions (the one which is currently examined) into the model is a crucial step because it has to be performed every time the impact of new a response action needs to be calculated. For the actual evaluation of the impact itself, the precalculated paths are utilized. This allows one to only take the effects of firewall rules and the availability of resources into account, making local optimization (i.e. finding the best response action among a set of alternatives) very efficient.

Finding a globally optimal response configuration is harder. It requires an exhaustive search of all possible combinations of alternatives in the *response history*, which is comprised by all response actions that have been suggested by the IRS. Although this search can be optimized too, it is still an expensive operation where the number of possibilities increases exponentially with the length of the history of the response actions that have been evaluated so far (shown in Figure 5).

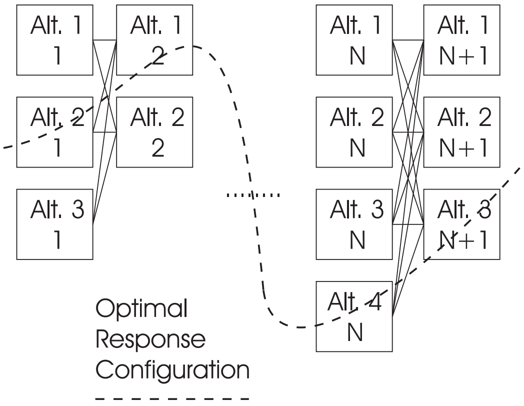

Figure 5. Response history and globally optimal response configuration

6.2. Performance Results

We obtained performance measurements to support the claims of our theoretical results. The execution times of different tasks performed by our prototype evaluation engine have been determined. We used a model with 35 resources which were heavily depended on each other - the dependency trees had a depth of up to eight. These resources were distributed over five subnets. We evaluated the impact of thirteen different response actions that consisted of up to ten firewall rule changes, user accounts and process status modifications.

The average number of alternatives in each test was 2.5384. The performance measurements (listed in Table 1 below) have been collected on a Pentium III machine with 550 MHz and 512 MB RAM. For each local optimization step, only up to eight scenarios had to be evaluated where we could make use of partial evaluation. The global evaluation, on the other hand, had to completely evaluate 5184 response configurations in order to determine an optimal solution.

The results show that evaluating different response actions can be done quickly. This is caused by the fact that only crucial resources are modeled and that optimized data structures are used during the evaluation process. While the complete entity capability evaluation is suitable for real time response, a complete global optimization may take longer, depending on the length of the response history and the number of alternative response actions. This is caused by the fact that many alternatives in a long history of response actions lead to an explosion of the number of sequences that have to be tried. While this seems to be undesirable at first glance, one has to realize that no real time performance is needed for this task. Even if the model requires a minute to find a globally optimal response configuration with an adequate level of security, the result is still beneficial. The security of the system has to be achieved first, then, in a second step, the usability can be improved.

Insertion and deletion	0.0255 ms
Complete entity capability evaluation	0.915 ms
Global optimization step	34.358 s

Table 1. Performance Results

7. Conclusion and Future Work

We have presented a network model together with an evaluation function that can be consulted by an intrusion response component to determine the response action which yields the minimal negative impact on deployed network resources and their users. The effects of ‘severe’ responses and their impact on the usability of the whole system can be estimated.

We propose a network model that takes network topology, firewall rules, services and users into account and supports both, dependencies among entities within the network and those to and from outside users. This allows us to determine the costs of disabling crucial resources in a response function. The evaluation mechanism which determines the negative impact exhibits good performance properties, especially the variant that determines the best action among a set of possible alternatives.

Future work will extend the network model and the cost functions. Instead of deriving the capability of an entity from static dependency weights on various services, more sophisticated functions could be utilized. Usually penalty costs are not constants, but they are a function of time, and our model could be extended to do so. Work will also concentrate on improving the global optimization step which is computationally expensive now (because an exhaustive search is performed). Priority queues and dynamic programming might help in speeding up that process.

References

[1] D. Anderson, T. Frivold, A. Tamaru, and A. Valdes. *Next Generation Intrusion Detection Expert System (NIDES)*. SRI International, 1994.

[2] C. A. Carver, J. M. D. Hill, and U. W. Pooch. Limiting Uncertainty in Intrusion Response. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, United States Military Academy, West Point, June 2001.

[3] F. Cohen. Simulating Cyber Attacks, Defenses, and Consequences. http://all.net/ journal/ntb/simulate/simulate.html, May 1999.

[4] D. Denning. An intrusion-detection model. In *IEEE Symposium on Security and Privacy*, pages 118–131, Oakland, USA, 1986.

[5] L. Eschenauer. Imsafe. http://imsafe.sourceforge.net, 2001.

[6] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. In *Proceedinges of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.

[7] C. Krügel, T. Toth, and C. Kerer. Service specific anomaly detection for network intrusion detection. In *Symposium on Applied Computing (SAC)*. ACM Scientific Press, March 2002.

[8] P. G. Neumann and P. A. Porras. Experience with emerald to date. In *1st USENIX Workshop on Intrusion Detection and Network Monitoring*, pages 73–80, Santa Clara, California, USA, April 1999.

[9] P. A. Porras and P. G. Neumann. Emerald: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th NIS Security Conference*, October 1997.

[10] M. Roesch. Snort - lightweight intrusion detection for networks. In *USENIX Lisa 99*, 1999.

[11] D. Sterne, K. Djahandari, B. Wilson, B. Babson, D. Schnackenberg, H. Holliday, and T. Reid. Autonomic Response to Distributed Denial of Service Attacks. In *Proceedings of 4th International Symposium, RAID 2001*, Davis, CA, USA, October 2001.

[12] G. Vigna, S. Eckmann, and R. A. Kemmerer. The STAT Tool Suite. In *Proceedings of DISCEX 2000*, Hilton Head, South Carolina, January 2000. IEEE Computer Society Press.

[13] G. Vigna and R. A. Kemmerer. NetSTAT: A Network-based Intrusion Detection System. In *14th Annual Computer Security Applications Conference*, December 1998.

[14] G. Vigna and R. A. Kemmerer. NetSTAT: A Network-based Intrusion Detection System. *Journal of Computer Security*, 7(1):37–71, 1999.

[15] G. B. White, E. A. Fisch, and U. W. Pooch. Cooperating security managers: A peer-based intrusion detection system. *IEEE Network*, pages 20–23, January/February 1996.

8. Appendix

8.1. Model Language Grammar

```
Resource:         <header> <properties>
                  'requires' [<depends>] ';'
header:           resourceName 'is' <type>
                  [number] <locations>
type:             ( 'SERVICE' | 'USER' )
locations:        'at' (IPAddress/subnetmask
                  | Hostname)
properties:       '{' (<property> ';')+ '}'
property:         Attribute '=' Value
depends:          ( <compounddepend> ['or'])+
compounddepend:   '(' resourceName <location>
                  [<degree>]['and'
                  <depends>]+ ')'
degree:           <number>
```

The grammars for importing router configurations and firewall rules as well as the API for the IRS have been omitted here because of lack of space.

8.2. Example

This section shows the network model of the simple example that has been introduced above in Figure 2 written in our grammar.

After that, we describe how the effects for the two different response actions are calculated and demonstrate that the order in which response actions are selected is crucial for the final result. Our example presents a situation where the selected responses, though locally optimal, do not lead to the best global result. Only an expensive global optimization which involves brute forcing all possible combinations can assert that a setup with a minimal global penalty cost is reached.

```
DNS is service at 132.100.98.11 53 udp
  { processName="bind";};

HTTP is service at 132.100.98.15 80,
  at 132.100.101.4 80
  { processName="httpd"; };

NFS is service at 132.100.100.4 2049
  { processName="nfsd"; };

anne is user at 132.100.100.27 { cost=5000; }
  requires (DNS at 132.100.98.11 53 udp 0.4)
   and
     ( NFS at 132.100.100.4 2049 0.4
       and  HTTP at 132.100.101.4 80 0.2 ));
```

```
customer is user at !132.100.0.0/255.255.0.0
  tcp { cost= 100000; }
  requires (HTTP at 132.100.101.4 80 1.0
    or HTTP at 132.100.98.15 80 1.0 );

// here routing entries & fw rules start
// subnet *.98
{
132.100.98.1 132.100.98.0 0.0.0.0
  255.255.255.0 eth1;
132.100.99.16 132.100.99.0 0.0.0.0
  255.255.255.0 eth0;
132.100.98.1 0.0.0.0 132.100.99.1
  255.255.255.0;
}
// subnet *.99
{
132.100.99.1 132.100.99.0  0.0.0.0
  255.255.255.0 eth0;
132.100.99.1 0.0.0.0 132.101.27.1
  255.255.255.0;
132.100.98.1 132.100.98.0 0.0.0.0
  255.255.255.0 eth1;
132.100.101.1 132.100.100.0
  132.100.101.79 255.255.255.0 ;
132.101.27.34 132.101.27.0  external
  0.0.0.0 255.255.255.0 eth3;
}
// subnet *.100
{
132.100.100.1 132.100.100.0 0.0.0.0
  255.255.255.0 eth1;
132.100.100.1 0.0.0.0 132.100.101.1
  255.255.255.0;
132.100.101.14 132.100.101.0 0.0.0.0
  255.255.255.0 eth0;
}
// subnet *.101
{
132.100.101.1 132.100.101.0 0.0.0.0
  255.255.255.0 eth0;
132.100.100.27 132.100.100.0 0.0.0.0
  255.255.255.0 eth1;
132.100.99.34 132.100.99.0 0.0.0.0
  255.255.255.0 eth2;
132.100.101.79 0.0.0.0 132.100.99.1
  255.255.255.0;
}

// Response actions to check
// First response action
{
 // Alternative 1-A
 { insertfwrule at 132.100.101.1:
    fw forward -i eth2 -j deny;
 }
 // Alternative 1-B
 { insertfwrule at 132.100.99.1:
    fw forward -sourceIP  !132.100.0.0
    -sourceNm 255.255.0.0 -destIP
    132.100.100.0 -destNm 255.255.255.0
    -j deny;
   insertfwrule at 132.100.99.1:
    fw forward -sourceIP  !132.100.0.0
    -sourceNm 255.255.0.0 -destIP
    132.100.101.0 -destNm 255.255.255.0
    -j deny;
 }
}
{ // Second response action
 // Alternative 2-A
 { insertfwrule at 132.100.99.1:
    fw forward -destIP 132.100.98.0
    -destNm 255.255.255.0 -j deny;
  }
}
```

Initially the response configuration does not contain any firewall rules (for the sake of simplicity). An ID system detects an attack coming from the Internet towards the machine `132.100.101.4`, which is running the `HTTP`-server. The IRS finds out that there are two ways to protect this server (labeled *alternative 1-A* and *alternative 1-B*). The IRS then requests the evaluation function to calculate the effects of both response actions. Alternative 1-A is inserted temporary into the model and the capability of all entities is determined, finding that it results in a reduced capability for the user Anne (because she will not be able to access the `DNS` server anymore) leading to a penalty cost of 2000. The capability for the entity Customer is not reduced because it can use one of two alternative `HTTP` servers. Even if one of them is not accessible the availability of the other one is sufficient and no penalty cost has to be assigned to this resource. The total penalty cost for this alternative is therefore 2000.

The evaluation of alternative B reveals that the capability of both, the customer and Anne are not reduced, resulting in a total penalty cost of 0.0. This means that the IRS will clearly use this variant, because it has a lower penalty value.

The ID system then detects another attack in the network `132.100.98.0/24`, for which the IRS finds only one response action (namely alternative 2-A). As there are no alternatives to this response action, the optimal choice can be determined easily. Unfortunately, together with alternative 1-B, the capability of the customer drops to 0.0 which results in a total penalty cost of 100000 for this variant.

When alternatives 1-A and 2-A would have been chosen, the total penalty cost would have been only 2000. This emphasizes the importance of global optimization. Notice that the reconfiguration will not change the security but increases the availability of important services.