A Protection Scheme For Security Policies In Ubiquitous Environments Using One-Way Functions

Håkan Kvarnström[1], Hans Hedbom[2] and Erland Jonsson

Department of Computer Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden

{hkv, hansh, erland.jonsson}@ce.chalmers.se

Abstract

This paper addresses the problem of protecting security policies and other security-related information in security mechanisms and products, such as the detection policy in an Intrusion Detection System (IDS) or the filtering policy in a firewall. Unauthorized disclosure of the such information is particularly serious, since it might reveal the fundamental principles and methods for the security and protection of the whole system or network, which is much more far-reaching that the protection of the target system or security mechanism itself. This problem is especially noticeable in ubiquitous environments where a possible large number of nodes need knowledge about the security policy of their domain. In order to avoid this risk we suggest that security information should be protected using one-way functions and the paper suggests a basic scheme for protecting stateless policies. A stateless policy is a policy that only takes the current event into consideration when decisions are made and not the preceding chain of events. Thus, the process of comparing events towards the policy, i.e. making decisions, could be done in much the same way that passwords are hashed and compared in UNIX systems. However, one important distinction is that security policies contain a certain variability that must be handled and a method for this is discussed. The suggested scheme is very basic and has certain drawbacks as regards practical implementation, but does still clearly demonstrate the protection principle. We expect further research to result in extended methods that are more suitable for practical design.

---

[1] The author is also with Telia Research AB, SE-123 86 Farsta, Sweden.

[2] The author is also with the Department of Computer Science, Karlstad University, SE-651 88 Karlstad, Sweden.

Keywords: intrusion detection systems, detection policy, protection schemes, one-way functions.

1 Introduction

The protection of computers and information systems is vital for the success of virtually every enterprise. Distributed system architectures connecting a large number of computers raises questions on how to better protect the information and resources of these systems. Traditionally, access-control services such as firewalls [3, 4], are used to control access to systems and services. However, the use of access-control components only, could present a single-point-of-failure. A flaw in an access-control component could lead to loss or theft of information or computer resources by allowing an intruder to circumvent existing security measures. Intrusion detection systems (IDS) [13] is a technology that attempts to detect unauthorized activities and suspicious events (behavior), that is, events that violate the effective security policy for a certain domain. Intrusion detection systems provide a second line of defence, allowing intrusions to be detected in the event of a breach in the perimeter defence. In addition, intrusion detection systems allow misuse or suspicious behavior of users to be detected.

The security of the IDS itself is important for several reasons. First, it is obvious that the functionality of the IDS, i.e. its ability to operate as expected, depend greatly on the security of the IDS itself. i.e. its ability to resist attacks. If an intruder succeeds in mounting an attack against the IDS, either by taking over the IDS, corrupting its input data or its detection policy, or in other ways fool the IDS, it will no longer give alarms for attacks launched against the target system. In a report by Newsham and Ptacek [15], several successful attacks against existing intrusion detection systems are identified. Further, the information contained within the IDS (e.g. audit data) may be misused by an intruder to gain knowledge about the target system (e.g. weaknesses, protocols used, etc.) that would indeed facilitate attacks. However, the most serious perspective, especially for large, distributed systems, is that the IDS, or other security mechanism, could contain information such that the unauthorized disclosure of it would endanger the security of the whole computer network. Examples are overall security properties, lists of trusted hosts and information on unprotected vulnerabilities.

The argumentation above holds for firewalls as well, or indeed for any security mechanism that stores policies or other security-related information. However, in the following we will mainly discuss IDSs. One main issue for an IDS is how to conceal its policy. This could be achieved by using encryption, as discussed by Neuman [11] for the NIDES system [12]. His method requires storing keys on the local host, which virtually means that anyone that gains access to the host also gains access to the key and thereby the policy.

Hiding the key does not really help in the long run as Shamir and van Someren demonstrates in [14]. Beside this, we could be stuck with the problem of distributing keys to a possibly vast number of hosts, if the IDS or the firewall is distributed or cooperating with other IDSs or firewalls.

Another way of protecting the policy could be to use one-way functions. To our knowledge this has not been previously discussed. Thus, this paper presents a first attempt towards this type of protection. It suggests a method for protecting stateless policies using one-way functions. We are fully aware of the fact that stateless policies only can be used to describe a limited set of decision rules, but are nevertheless convinced that it could serve as a starting point for discussing the issue and for constructing schemes to handle more complex types of policies for firewalls and IDSs.

2 The need for policy protection

2.1 Policies

Security mechanisms, such as IDSs and firewalls are equipped with a decision function e.g when to send an alarm in the IDS case or whether or not to let traffic through in the firewall case. In order to make those decisions the systems uses some form of rule set (or rule base) that serves as the basis for the decision. We call this rule set the detection policy in the IDS case and the filtering policy in the firewall case. When there is no need to separate the two we will collectively refer to them as a policy. Examples of policies are:

- Rules for access-control to objects (e.g. access control lists)
- Misuse signatures
- Statistical user or system normal behavior

The policy is usually expressed in a description language describing the *event* or *combination of events* that is considered inappropriate (or in the firewall case: usually appropriate). In this paper, we define an event as an occurence of a single activity which is registered and stored in an audit-log. The mechanisms presented in this paper are not limitied to events of a certain type or having certain characteristics, as long as they can be coded or described as a (binary) string. A typical event would be an audit-log record describing user activity (e.g. accessing a web-page, logging into a service etc.). Table 1. shows five events generated as a result of remote logins to a server host.

Jun 24 18:19:42:jellybean sshd[1084049]: log: Connection from 192.168.0.1 port 56722
Jun 24 18:21:12:jellybean sshd[3472342]: log: Connection from 192.168.0.244 port 16239
Jun 24 19:29:14:jellybean sshd[1265421]: log: Connection from 192.168.0.123 port 54346
Jun 24 20:19:01:jellybean sshd[9934742]: log: Connection from 192.168.0.220 port 16222
Jun 24 21:45:41:jellybean sshd[1124234]: log: Connection from 192.168.0.11 port 201

Table 1. Five events generated by remote logins to the host "jellybean".

For example, a rule in a filtering policy could state that connections between the internet and the intranet are only allowed if they originate from the intranet and a rule in a IDS could express that logfile entries containing the string "login successful" and a time stamp between 11 pm to 4 am indicates a possible intrusion.

A typical intrusion detection systems rule-base can be divided in two parts. The first part consists of a set of well-known (standard) attack signatures, often provided by the supplier of the IDS. This part is updated regularly similar to virus scanners. The second part of the rule-base consists of site-specific threats and vulnerabilities, which may be unique to the target system one wishes to protect. For example, the target system could be configured to run old versions of software for compatibility with legacy systems or use applications and protocols known to be vulnerable to attack such as NFS and NIS. In the next section, we show that the latter type of rule-base impose a threat to the target systems.

2.2 The need for protection

The main reason for using an IDS is to improve the security of the target systems by adding a second layer of defence. However, there are inherent security concerns also for this second layer as it may introduce new security risks [7, 8]. These concerns involve the protection of information and information flow within the IDS and how the information can be used for illicit purposes. In this section we will discuss the security properties of an IDS and argue that the confidentiality aspect for the detection policy is by far the most important requirement.

2.2.1 Confidentiality and integrity of audit data

Audit data generated by the target systems within an IDS domain may contain sensitive information not to be disclosed outside the domain. The sensitive information might be about users and target systems as well as application related data. In some cases, the mere existence of an event may be confidential as it reveals some form of activity. Nor should audit data be subject to insertion, deletion or alteration as demonstrated in a paper by Ptacek and Newsham [15]. On the other hand, breach of confidentiality or integrity of audit

data presents less a risk than for policies. The result of a confidentiality or integrity breach can often be limited to a missed detection or a false alarm.

2.2.2 Confidentiality and integrity of the policy

Similarly, an attacker may break into an intrusion detection system and disable the detection mechanism so that target system attacks can be launched without being disclosed. However, if and when the attack eventually is detected, the system can be taken offline, its integrity restored, and finally brought back into operation. The damage is limited to the loss of detection capability over a period of time. On the contrary, if the rule-base is disclosed during the attack the attacker can learn about inherent vulnerabilities of the target system as well as of the whole distributed system, a knowledge that would still remain even when the integrity of the IDS is restored. The inherent vulnerabilities of the target system may not be possible fix and hence, permanent damage has occurred. Thus, the main reason for our concern about the confidentiality of the rule-base is that a breach of confidentiality is irreversible and cannot be undone.
As an example, consider a target system running NFS to share files between clients. A policy rule could be defined to detect the use of NFS (UDP) packets containing certain strings (e.g. /etc/passwd) between a certain NFS client and a dedicated server. If such a policy rule were stored in cleartext in the rule-base, an attacker would learn about the inherent vulnerability that exists and possibly exploit it to gain access to the information in the system. However, this particular attack signature is target specific and an attacker may not discover the vulnerability when conducting a vulnerability scan of known attacks.

Our experience when applying intrusion detection systems is that the rule-base containing target-specific attack signatures increases rapidly as the complexity (e.g. number of hosts and services) of target systems grows. This is different from the standard attack rule-base, which is not affected by the complexity of the target system.

3 Architectural implications

Increased use of network encryption and virtual private networks providing end-to-end security between systems, makes it hard for intrusion detection systems to monitor events in the target system or in the network. One solution to this problem is to execute the detection system on the end-systems where the encryption terminate (e.g. a personal IDS). This means that the security policy will be distributed and a distributed intrusion detection architecture (IDA) [7] results. It is clear that the IDS ability to protect its detection policy is highly dependent on the intrusion detection architecture.

An IDS is distributed when the different components of the detection system are distributed in some respect. Figure 1 illustrates a distributed intrusion detection architecture where components are located in different domains.

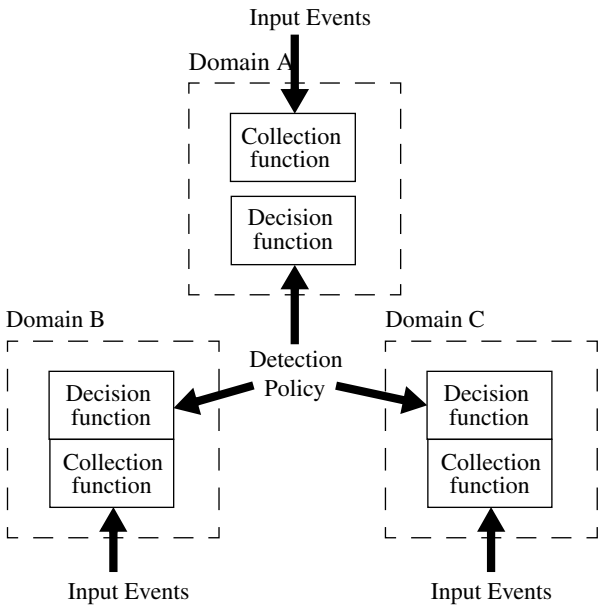

Figure 1. A distributed intrusion detection architecture (IDA)

In a strictly centralized system, the detection policy is known only to a small part of the system. However, in a distributed intrusion detection architecture, the policy is distributed to all of the decision functions that participate in the system (as described by Figure 1). In addition, the end-systems are not dedicated for the purpose of detecting intrusions (i.e. many other applications execute on the systems) which alters the threat model for the IDS. An end-system such as an office PC are often under control of the user which makes it harder to enforce the security policy in practice. Thus, the detection policy may be known to a possibly large number of entities. This implies that the security of the policy is dependent of the security of all the entities that have knowledge of the policy and of the security of the distribution channel.

4 Protecting a policy using one-way functions

One way of solving the problem of how to protect the policy could be to use strong one-way functions. In this section we will discuss the concept of one-way functions and discuss how they can be used in order to protect stateless policies.

4.1 The concept of one-way functions and commitment

One-way functions are one of the fundamental building block of cryptography. Many security mechanisms providing security services such as authentication, integrity protection and confidentiality depend on the unique properties provided by one-way functions. Informally, a one-way function [6] is a function $f\colon S \rightarrow T$ where S and T are any two sets, such that:

(1) for any $x \in S$, $f(x)$ is "easy" to compute,

(2) given the information that $f(x) = y$, there is no "feasible" way of computing x for any reasonable large proportion of the y belonging to T,

(3) for any $\mathfrak{r}, z \in S$, $f(x) \neq f(z)$. This property states that the function must be collision free, in the sense that no two values of S must result in the same y belonging to T

Assuming that the set S is sufficiently large, an exhaustive search will be computationally infeasible and thereby impractical. The UNIX password protection scheme [10, 5] is an example of a security mechanism making use of one-way functions. It provides confidentiality of the users' passwords, thus preventing disclosure of the passwords even though the password file itself is disclosed. By calculating a one-way hash (using the DES-encryption scheme) of a user's password, it is protected from disclosure but the resulting hash can still be used to verify that a user entered a correct password during the authentication process. This is achieved by calculating a one-way hash using the password provided during authentication and thereafter comparing the resulting hash with the stored hash for that particular user. If the two hashes are identical, the user must have entered the correct password and the authentication succeeded. Given the list of all one-way hashes for the users of the UNIX system, the only way of retrieving the cleartext password is to perform an exhaustive search over the entire domain of clear-text passwords. However, in practice, the number of possible passwords are often limited by the fact that human users normally choose words and phrases as passwords (e.g. names, the make of cars, or other easy to remember words) instead of random strings. Thus, making the search easier.

Several candidate one-way functions have been proposed. Factoring and the discrete logarithm problem are two well-know "hard" mathematical problems that often are used to create one-way functions.

4.2 Protection principle

Informally, the policy classifies an incoming event (or sets of events) into predefined categories such as *legal events, intrusion attempts, intrusions* etc. In its simplest form a secu-

rity policy states what patterns or signatures of events that are authorized/unauthorized. A *default accept* policy would search for events having a certain signature and classify those as unauthorized whereas a *default deny* policy makes the assumption that all events are unauthorized except those that explicitly matches a defined signature. Most rule-based IDS have taken a default accept standpoint due to the difficulty of defining all authorized events while most firewalls have a default deny policy, only letting through the events matching the policy.

The following simple example show how signatures can be used to detect policy violations in a default accept policy. Consider a set of input events $x_1, x_2 \ldots x_n \in X$ all of which are represented by k-bit binary strings. Further, the set $u_1, u_2 \ldots u_m \in U$ is the set of "signature strings" that identifies an unauthorized event. Whenever $x_i = u_j$ where, $i \leq n$, $j \leq m$ the event being analyzed matches a detection signature and an alarm is raised. The detection scheme is fairly simple as it only involves comparing events over X with all strings in U searching for identical pairs. Now consider the set $u_1', u_2' \ldots u_m' = f(u_1), f(u_2) \ldots f(u_m) \in U$ where f is any cryptographically strong one-way function. For each signature, a hash is calculated and stored. Due to the inherent properties of the one-way function, it is hard to deduce any $u \in U$ given $f(u) \in U'$. Thus, U' can be made publicly available without compromising the secrecy of U. The computational effort to successfully deduce $u \in U$ is on average equal to an exhaustive search of $1/2$ of the domain of U. Thus, assuming $u_1, u_2 \ldots u_m \in U$ where $|u|$, $u \in U$ has a binary length of k bits. The computational effort to find $u \in U$ given $f(u) \in U'$ would (in average) require 2^{k-1} operations.

Detecting policy violation in the default accept case is a straightforward process of applying the same one-way function to all input events $x \in X$ and compare the resulting values to the stored values of U'. If a match is found, the input event is an unauthorized event and an alarm is raised. Many intrusion detection systems utilizes simple string matching to find unauthorized patterns in the input data. For example, an IDS could search a UNIX syslog looking for strings containing the pattern "su root failed" which would indicate a failed attempt to gain administrative privileges on the system. By using one-way functions to hide the string patterns, it is virtually impossible for an intruder to find out about the detection policy of the system.

Policy violations in the default deny case can be handled in a similar manner as discussed above. The only difference in this case is that U, and thus U', will contain allowed events and actions are taken if $f(x)$, $x \in X$ is not a member of U'.

5 Handling variabilities

In a normal case one input value will lead to a unique output value from the one-way function. This means that even a small change in the input value will generate a totally different result. This is a desired property in traditional use of one-way functions,but it is undesirable in the policy case. In this section we will discuss why we need some way of handling variabilities in the stateless policy case and give some ideas on how this could be achieved using different methods. We will also elaborate on what we believe to be the shortcomings and merits of the different methods.

5.1 Why do we need to handle variabilities?

When describing rules in a policy it is useful to be able to handle variables or to express intervals. This means that the protection scheme for the policy must also be able to handle this variability in some way. For example, it might be desirable to state in the policy that certain actions are forbidden for every user or that a given user is not allowed to login to a number of network addresses. If we divide the information in fields there will be fields containing variables in both the cases above. In the first case the value in the forbidden action field will be constant but the value in the user field is variable and in the second case the value in the user field is constant but the value in the network field is variable. In the latter case it is also useful to state an interval, i.e this is the range of network addresses that are allowed or disallowed.

5.2 A first approach

One naive approach to handle variability might be to apply the one-way function on every combination of the constant field or the hashed field. This would, however, lead to a very large collection of values that need to be compared. In small systems this might be tolerable, but in large systems it would be unacceptable. The next approach would be to skip the variable fields and just apply the one-way function on the fixed fields. This solves the problem in the first case, where we do not care who the user is, i.e all possible values of the user field is considered illegal. However, it will not solve the problem in the second case above since not all of the possible values in the network field might be considered illegal. Besides, it also has a serious side effect. By applying the function on some fields and not on others we are giving away information on which fields we are interested in and that in turn might, under certain circumstances, point an intruder towards which attacks we are looking for.

5.3 Using fuzzy commitment to handle intervals.

Fuzzy commitment is a a way of doing commitment suggested by Ari Juels and Martin Wattenberg [9]. It is essentially a method that accepts a certain amount of fuzziness (i.e variability) in the witness[1] used. Its main use is in the area of authentication by means of biometrics (e.g. fingerprints etc.) where it is almost impossible to get exactly the same result from two consecutive scans of the same object. For example a thumb is newer placed in the same way twice on a thumb scanner. Juels et al. show that by using a combination of error correction methods and one-way functions one can create a commitment scheme that will accept different witnesses as long as they only differ up to a controllable threshold. In essence the method works as follows: Assume that we have a witness x and an error correction function f that corrects codewords from the set C. Now, choose a $c \in C$ and calculate $d = x - c$. Commit c by using a strong one-way function and store d. To decommit c a user has to give a witness x' that is sufficiently close to x so that the correction function f can correct $x' - d$ to c. The amount of fuzziness accepted is thus dependent on number of errors accepted by the correction function and on the value d.

Fuzzy commitment could be used to handle intervals in the following way. The basic idea is to let all the values in an interval hash to the same cryptographic hash-sum. In this case we do not really need a proper error correction function but merely a function that groups values together. Such functions could be used in fuzzy commitment, e.g. Juels et al [9] uses a lattice rounding function rather than a proper error correcting function as an example of how the scheme works. The lattice rounding function in their example is a function that rounds points in the plane to the nearest multiple of 100x100. If we interpret the values in the variable fields as integers we could use a function that truncates values downwards to the nearest multiple of the selectable integer i, e.g. if $i = 10$ then $0 \ldots 9 \rightarrow 0$, $10 \ldots 19 \rightarrow 10$ and so on. Lets call this function g. Such a function could easily be generalized by making i an parameter to the function (e.g. $g(i, x) = i\lfloor x/i \rfloor$). Let the witness x represent the lower end of the interval and let i be the width of the interval. Choose randomly a $c \in C$ where $C = \{\forall c \in C : ix\}$, where x is an integer. Calculate all other parameters according to the rules above. To test later whether an obtained witness x' is within the interval the calculations described above for decommitment is used exchanging f with g.

Example (commitment). Assume that we want to commit the interval $234 \ldots 243$. In this case $i = 10$ since the width of the interval is 10. The set C will be all integer multiples of 10 and we randomly choose $c = 410$ from this set. Since the lower end of the interval is

[1] The witness is the value that is to be committed or the value that is compared with a committed value after it has been transformed with a strong one-way function.

234 $x = 234$ and $d = x - c = 234 - 410 = -176$. c is then committed using a strong one-way function.

Example (decommitment). Assume that the commitment in the example above has been made. Further assume that we want to find out if 240 is with in the interval i.e $x' = 240$. We first calculate $c' = x' - d = 240 - (-176) = 416$. By applying the function g on 416 getting 410 as the result. We then commit 410 using the same strong one-way function as in the example above and compare the results.

5.4 The shortcomings of using fuzzy commitment.

The big disadvantage when using fuzzy commitment is that the error correction function will give the codeword in the clear as output. With this value and the known value of the variable d it is very easy to calculate the interval. The implications of this is that one lucky guess will reveal the whole interval. This is a small problem if the set of possible values is very large and the intervals are narrow since the probability of guessing a corest value is very small and an exhaustive search is highly time consuming. However, in most "real world" cases of specifying intervals for detection purposes the set of possible values will be small and/or the intervals will be very wide. Therefore, the probability of a lucky guess is high and it is relatively easy to perform an exhaustive search. In this light fuzzy commitment by itself is not a perfect solution to the interval problem. In section 5.5 we discuss a method to make it harder to do this deduction.

5.5 Making value deduction harder.

The possible input values to the one-way function are usually few enough to permit an exhaustive search. This is basically due to the fact that we are dividing the input stream into fields and separately applying the one-way function to each field, but also because we in essence are dividing the possible values of the fields into equivalence classes and thereby making it easier to find one input value that maps to a valid output value. Of course, one could make the division of the data in such a way that the input fields would be large enough. However, we believe that in most applications there are natural divisions based on the format of log entries and other data structures in the system. The individual fields in these structures usually have a small domain of possible input values, but the combined structure in itself has a much larger domain. By applying the function repeatedly in a tree-like manner it should be possible to use the input domain of the structure as the input domain of the resulting one-way function and still be able to handle variabilities in the structure as previously discussed. The method works as follows:

1. Calculate the individual fields of the structure by applying an appropriate method, i.e a one-way function on the fixed fields and a method for handling variability on the vari-

able fields. Please not that the method for handling variablility includes the application of a one-way function.

2. Concatenate the resulting values pair-wise and apply a one-way function to each of the pairs
3. Repeat step 2 until only one value remains. This is the value that is compared with the policy.

The process above is graphically described in Figure 2. It could be generalized by always applying the method for variablilty and restricting the variability for the fixed fields e.g setting the intervall lenght to 1 if the fuzzy commitment approach is used.

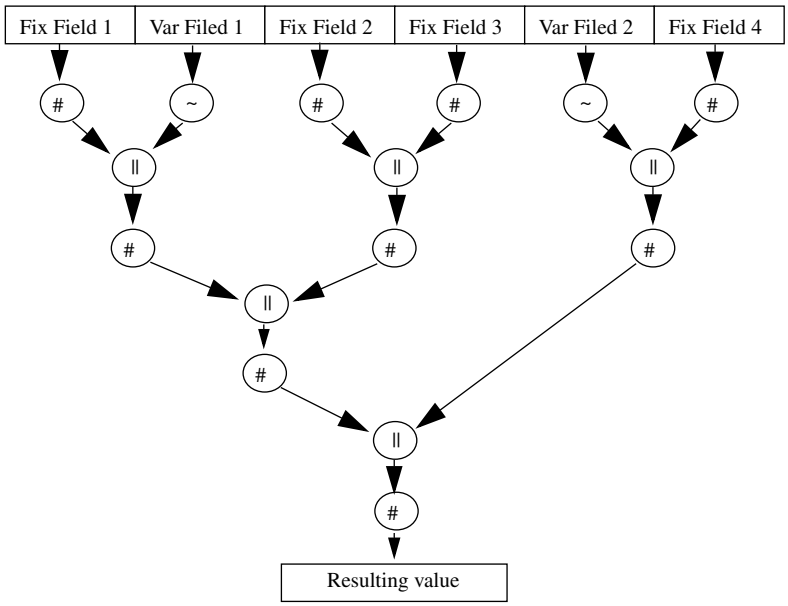

Figure 2. Method one for handling variabilities

If all the fields in the structure are variable or if the structure is small the domain might still be to small. However, it is, in any case, larger than the input domain of the individual fields.

An alternative to the method described above could be the following:

1. Calculate the individual fields of the structure by applying an appropriate method (i.e a one-way function on the fixed fields and a method for handling variability on the vari-

able fields).

2. Concatenate all the resulting values and apply a one-way function to the concatenated value.

This process is graphically described in Figure 3.

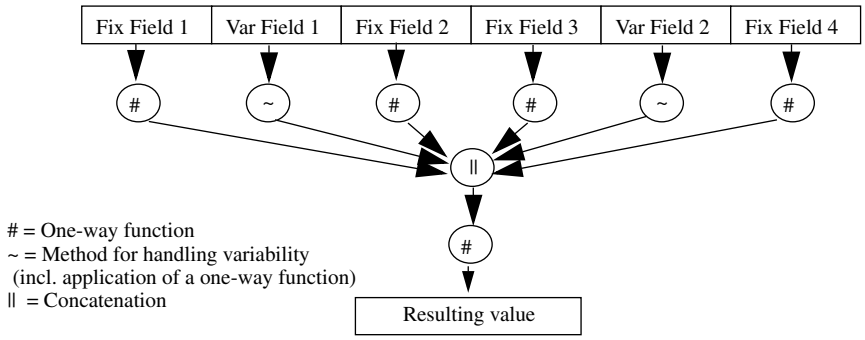

Figure 3. Method two for handling variabilities

The big difference between the first approach and the second approach, besides a reduction in the use of one way functions, is that the former can be used to do iterative matching. Iterative matching in this case would be to try and find a match in the policy whenever a one-way function is applied in Figure 2. Thus, matches could be found for individual fields or groups of fields using the same general algorithm, i.e the matching algorithm indicates a policy violation whenever a match is found in any level of the tree. By doing this, and constructing the sequence of fields in an appropriate way, it would actually be possible to leave out fields in some cases without explicitly stating this in the matching algorithm.

The following example illustrates how the audit-logs in Table 1 on page 4 can be used for detection and how variability is achieved. A template is constructed which will be used for filling in information contained in the logs.

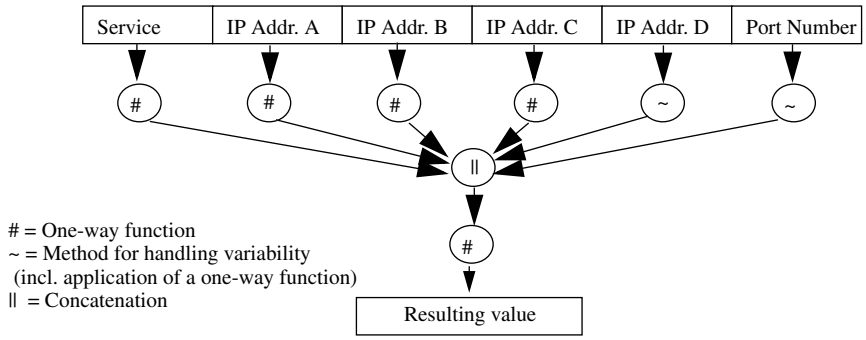

Figure 4. Template for detecting remote login policy violations

In Figure 4. four fixed fields and two variable fields are used to encode an event for remote login attempts. The service field contain the name of the service "sshd". The first three IP-address fields (A-C) contain fixed values for the subnet in which the IDS reside. The variable IP-address field (D) contains the range of hosts for which remote login is a policy violation. The last variable field, "Port Number" enables us to disallow login attempts from certain portnumbers such as non-privileged ports (above 1024) on UNIX systems. During the detection phase, the input events are encoded one by one using the template and compared to the set of stored hash values indicating a policy violation.

6 Discussion and future work

There are a number of open questions regarding the protection of policies using one-way functions. Below, we will discuss some of them and give indications of future work.

First, we have only discussed the protection of stateless policies, i.e. policies that only takes the current event under consideration. We are fully aware that this presents a certain limitation, but we do feel that the idea of this type of policy protection has a fundamental value that is worth presenting. A more elaborated protection scheme for stateful policies could be used to describe more complex threats and attacks and we are indeed in the process of carrying through the work to develop such a scheme.

Second, we have discussed the need to handle variability and suggested solutions on how to handle intervals but not on how to handle general variables. In order to solve that problem we would probably need a strong cryptographic one-way hash function that generically and controllably would hash different values to one hash value. Basically this function needs to divide the possible values of the variable into equivalence classes based on the values that we are interested in and then hash the individual members of each

equivalence class into one hash value. In order to be generic this function must be able to take the relation that defines the equivalence classes as a parameter and it must be possible to define arbitrary relations without giving the individual values. We do not know of such a function, neither do we know if it is possible to construct it. There are collisionful one-way hash-functions described in the literature e.g. [6, 1, 2]. However, they do not solve the problem discussed above.

Finally, there is the question of performance. Some of the systems, e.g. IDSs in large computer systems or firewalls in high capacity networks, need to be able to handle large amounts of information in a limited time. The performance of an IDS described in this paper is likely to be significantly lower than a traditional signature-based intrusion detection system. We do not claim this scheme to be efficient, but instead focus on confidentiality properties and the feasibility to protect the policy from unauthorized disclosure. In a real-life system, traditional signature detection techniques could be used to detection well-known attacks and complemented using this technique for attacks that are target specific. This would limit the decrease in performance to a minimum.

7 Conclusion

This paper discusses the problem of protecting the security policy of a security mechanism, such as an intrusion detection system or a firewall operating distributed or ubiquitous environents. We have stated that the policy contains sensitive information that could be misused by an attacker in order to avoid detection or to render the detection system useless. Still worse, it could provide information that would facilitate intrusions into the target system or even extend to logically connected systems within or outside of the actual network. Therefore, the policy is crucial for the function of the security mechanism and the system it is supposed to protect.

We have suggested that one-way functions could be used as a means to protect the policy. However, this approach has certain shortcomings. Foremost of these is the fact that normal one-way function schemes can only be used on constant values, so that even small variations in input values give totally different output values. This is a desirable property for the ordinary use of one-way functions. In our case, however, we would like for a complete equivalence class of data to be hashed into one specific hash value and we have suggested a clustering method based on fuzzy commitment that would accomplish this for some, but not all, types of variability in the input data.

The drawback of clustering is that it increases the probability of guessing a correct match or performing an exhaustive search. It is also a fact that the very nature of events such as intrusions puts a bound on the possible cases, thereby making it easier to make a good guess. To counter this we have suggested a method that expands the possible domain

by grouping values together and repeatedly applying one-way functions in a tree-like manner, thereby making guessing and exhausitve searches more difficult.

References

1. S.Bakhtiari, R. Safavi-Naini and J. Pieprzyk. On the Weakness of Gong´s Collisionful Hash Function. Journal of Universal Computer Science, vol 3, no.3, pp 185-196, Springer Pub. Co, 1997.
2. S.Bakhtiari, R. Safavi-Naini and J. Pieprzyk. On Selectable Collisionful Hash Functions. The Australian Conference on Information Security and Privacy, LNCS No. 1172, pp 287-292, Springer Pub. Co., 1996.
3. D.B. Chapman and E.D. Zwicky. Building Internet Firewall. O'Reilly & Associates, Inc. September, 1995.
4. W.R. Cheswick and S.M. Bellovin. Firewalls and Internet Security: Repelling the Wily Hacker. Addison-Wesley. 1994.
5. D.C. Fieldmeier and P.R. Karn. UNIX password security - ten years later. Advances in Cryptology CRYPTO 89, LNCS 0302-9743; 435 pp:44-63, Springer cop., 1990.
6. L. Gong. Collisionful keyed hash functions with selectable collisions. Information Processing Letters 55, pp 167-170, Elsevier, 1995.
7. H. Hedbom, H. Kvarnström, E. Jonsson. Security Implications of Distributed Intrusion Detection Architectures, In Proceedings of the 4th Nordic Workshop on Secure IT systems - Nordsec 99, pages 225-243; Stockholm, Sweden.
8. H. Hedbom, S. Lindskog, E. Jonsson. Risks and Dangers of Security Extensions. In *Proceedings of IFIP Working Conference on Security and Control of IT in Society-II, SCITS-II*, Bratislava, Slovakia, June 15-16, 2001. To appear
9. A. Juels and M. Wattenberg. A Fuzzy Commitmen Scheme. In Proceedings of the second ACM conferens on computer and communication security CCS´99. Singapore, 1999.
10. R. Morris and K. Thompson. Password security: A case history. Communications of the ACM, 22(11):594-597, November 1979.
11. P.G. Neumann, "Architectures and formal representations for secure systems", Final Report; SRI Project 6401; Deliverable A002, 1995.
12. Next-generation Intrusion Detection Expert System (NIDES) - A Summary, SRI, Computer Science Laboratory, 1995.
13. S. Northcutt. Network Intrusion Detection : An Analyst's Handbook. New Riders. 1999.

14. A. Shamir, Nico van Someren, “Playing hide and seek with stored keys”, Weizmann Institute of Science, Israel; nCipher Corporation Limited, England, 1998
15. T. H. Ptacek, T. N. Newsham, “Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection”, Secure Networks, Inc.
16. S Staniford-Chen, B Tung, P Porras, Cliff Kahn, D Schnackenberg, R Feiertag, M Stillman, The Common Intrusion Detection Framework - Data Formats, Internet Draft, September, 1998.