Dynamic Response in Distributed Firewall Systems

Mohamed Taibah, Ehab Al-Shaer and Hazem Hamed

Multimedia Networking Research Laboratory
School of Computer Science, Telecommunications and Information Systems
DePaul University, Chicago, USA
Email: mtaibah@cs.depaul.edu

***Abstract*— Firewalls are currently the prominent defense against network attacks. These devices can play a crucial role in preserving the wellbeing of commercial as well as personal networks. However, the correct configuration of firewalls is hardly a trivial task, especially in distributed environments. A variety of anomalies can affect the proper functioning of firewalls. This paper discusses possible firewall anomalies in the single and distributed firewall cases. A formalization of the rule anomaly discovery problem is presented. As an application of the anomaly discovery algorithm, we overview an autonomous defense system to counter Internet worms. General components of such system are presented in a general envisioned design. Several research problems are presented in the context of such system.**

I. Introduction

The mitigation of network security threats has become a vital necessity for networks of all sizes today. The continuously increasing trend of targeted and random network attacks shows no sign of slowing down. Company private networks are especially at risk, partially because of the catastrophic consequences a malicious breach of network security may cause from a business perspective. Traditionally, network designers have responded to that threat with a diverse array of network security architectures. Firewalls, however, remain the single most important element at the backbone of any network security system. Correctly placing and configuring firewalls is at the essence of providing security to private networks.

This task, however, can be as complex as it is crucial, particularly when rule changes are made under tight time constraints, and by different administrators. Distinct firewall rules may interact in a manner that yields unexpected and undesired results. These rule anomalies can hinder network performance or may even create holes in the defense against attacks. The complexity of the firewall anomaly problem prompted researchers to seek a formal solution that can be automated. It is important here to realize two different cases, the case of a single firewall, and that of a distributed firewall system. Although both cases share similarities, the latter is by far the more complex.

The ability to correctly configure firewall systems based on a formal solution for the anomaly problem, may clear the way for researchers to develop safe algorithms to dynamically reconfigure firewalls based on emerging attacks. The goal is to design algorithms that can dynamically reconfigure or ”reposture” a network security system to withstand or even avert an emerging automated worm attack for example.

This report will overview solutions being developed at the Multimedia Networking Lab (MNLab) at DePaul University for the firewall anomaly problem. It will also attempt to overview a possible application for this formal solution in the area of dynamically re-posturing a security system based on sensed worm attacks.

II. Firewall Rule Anomaly Examples

Firewalls control network traffic -at the secured domain boundary- based on a set of filtering rules. In their simplest form, these rules are based on the packets 5-tuple (i.e. protocol, source address, source port, destination address, and destination port fields). A firewall either denies or allows a packet based on the values in these fields. An example of a firewall rule is: {*tcp, 140.192.37.*, Any, *.*.*.*, 80, Accept*}. In this rule, all TCP traffic coming from any port in sub-domain 140.192.37.*, and going to port 80 (http) on any IP address will be allowed to pass through the firewall. An ordered list of such rules constitutes the firewall policy. When processing a packet, a firewall sequentially goes through its list of rules until it finds a rule that matches the packets' 5-tuple and then it performs the action associated with that rule. If none of the rules is matched by the packets 5-tuple, the policy's default is automatically activated. In most cases, a ”deny” default is recommended. An example of a possible firewall policy is shown in figure 1.

Rule anomalies can arise in a variety of cases. For example, consider rules 3 and 6 in the firewall policy in figure 1. A packet with the 5-tuple {tcp, 140.192.37.61, 1234, 161.120.33.40, 80} matches rule 3 as well as rule 4. This causes ambiguity about which rule was meant to handle this packet. Obviously, the preceding rule is the one that will always be triggered in any such case. In the case of a distributed firewall system -or inter-firewall anomaly- , consider the firewall policies shown in figure 2. Assume that a packet is coming from the Internet to domain $D1.1$. The packet is part of a TCP stream going from port 4000 on host 189.124.32.60 to port 25 on host 140.192.22.10. On FW_0, this stream is allowed based on rule 3, however, FW_1 will deny this packet -in fact the whole stream- because of its ”deny all” default. This is an anomaly because either FW_0 is allowing through a stream that it should deny, or FW_1 is denying a stream that it should allow through.

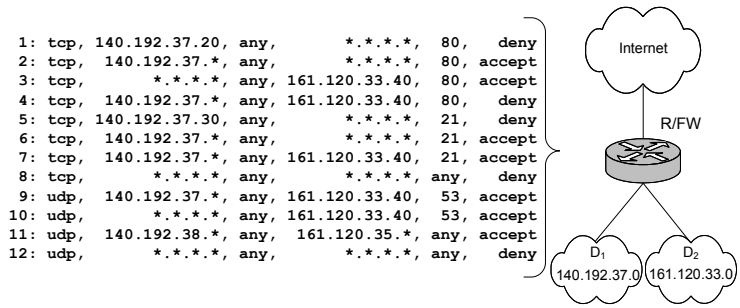

Figure 1. An example of a firewall policy

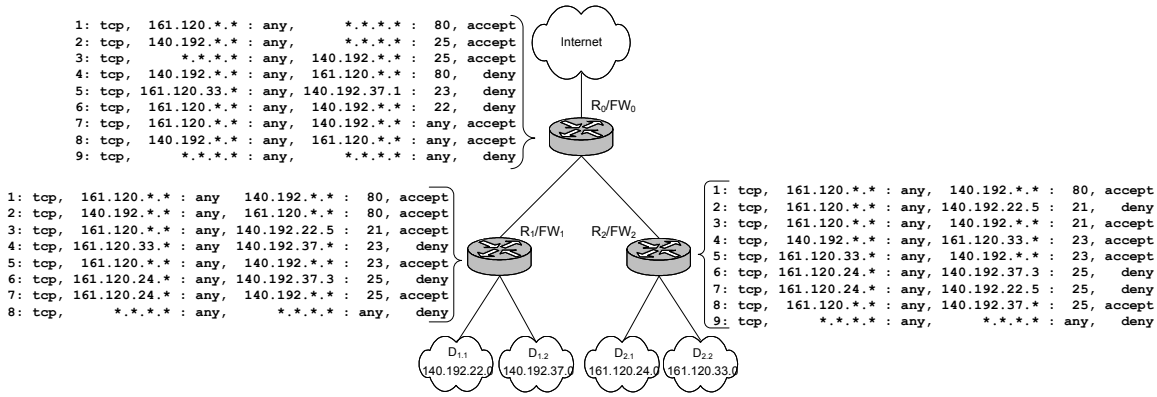

Figure 2. An example of distributed firewall policies

The examples given for firewall rule anomalies are instances of a more general set of possible anomalies. The formalization of this concept is discussed in the next three sections.

III. Firewall Policy Modelling

Modelling of firewall rule relations is necessary for analyzing the firewall policy and designing management techniques such as anomaly discovery and policy editing. In this section, we formally describe a model of firewall rule relations.

A. Formalization of Firewall Rule Relations

A useful model must cover all the relations that can relate packet filtering rules. In this section we define all the possible relations that may exist between filtering rules, and we show that no other relations can exist. We determine the relations based on comparing the network fields of filtering rules as follows.

Definition 1: Rules R_x and R_y are *completely disjoint* if every field in R_x is not a subset nor a superset nor equal to the corresponding field in R_y. Formally, $R_x \Re_{CD} R_y$ iff

$$\forall i : R_x[i] \not\bowtie R_y[i]$$
$$\text{where } \bowtie \in \{\subset, \supset, =\},$$
$$i \in \{\text{protocol, s\_ip, s\_port, d\_ip, d\_port}\}$$

Definition 2: Rules R_x and R_y are *exactly matching* if every field in R_x is equal to the corresponding field in R_y. Formally, $R_x \Re_{EM} R_y$ iff

$$\forall i : R_x[i] = R_y[i]$$
$$\text{where } i \in \{\text{protocol, s\_ip, s\_port, d\_ip, d\_port}\}$$

Definition 3: Rules R_x and R_y are *inclusively matching* if they do not exactly match and if every field in R_x is a subset or equal to the corresponding field in R_y. R_x is called the *subset match* while R_y is called the *superset match*. Formally, $R_x \Re_{IM} R_y$ iff

$$\forall i : R_x[i] \subseteq R_y[i]$$
$$\text{and } \exists j \text{ such that } : R_x[j] \neq R_y[j]$$
$$\text{where } i, j \in \{\text{protocol, s\_ip, s\_port, d\_ip, d\_port}\}$$

Definition 4: Rules R_x and R_y are *partially disjoint* (or *partially matching*) if there is at least one field in R_x that is a subset or a superset or equal to the corresponding field in R_y, and there is at least one field in R_x that is not a subset and not a superset and not equal to the corresponding field in

R_y. Formally, $R_x \Re_{PD} R_y$ iff

$\exists i, j$ such that $R_x[i] \bowtie R_y[i]$ and $R_x[j] \not\bowtie R_y[j]$
where $\bowtie \in \{\subset, \supset, =\}$,
$i, j \in$ {protocol, s_ip, s_port, d_ip, d_port}, $i \neq j$

Definition 5: Rules R_x and R_y are *correlated* if some fields in R_x are subsets or equal to the corresponding fields in R_y, and the rest of the fields in R_x are supersets of the corresponding fields in R_y. Formally, $R_x \Re_C R_y$ iff

$\forall i : R_x[i] \bowtie R_y[i]$ and
$\exists j, k$ such that $: R_x[j] \subset R_y[j]$ and $R_x[k] \supset R_y[k]$
where $\bowtie \in \{\subset, \supset, =\}$,
$j, k \in$ {protocol, s_ip, s_port, d_ip, d_port}, $j \neq k$

The following theorems show that these relations are distinct, i.e. only one relation can relate R_x and R_y, and complete, i.e. no other relation between R_x and R_y could exist. The complete proofs for the theorems are presented in [2].

Theorem 1: Any two k-tuple filters in a firewall policy are related by one and only one of the defined relations.

Theorem 2: The union of these relations represents the universal set of relations between any two k-tuple filters in a firewall policy.

B. Firewall Policy Representation

We represent the firewall policy by a single-rooted tree called the *policy tree* [2]. The tree model provides a simple representation of the filtering rules and at the same time allows for easy discovery of relations and anomalies among these rules. Each node in a policy tree represents a network field, and each branch at this node represents a possible value of the associated field. Every tree path starting at the root and ending at a leaf represents a rule in the policy and vice versa. Rules that have the same field value at a specific node will share the same branch representing that value.

Figure 3 illustrates the policy tree model of the filtering policy given in Figure 1. Notice that every rule should have an action leaf in the tree. The dotted box below each leaf indicates the rule represented by that branch in addition to other rules that are in anomaly with it as described later in the following section.

IV. Intra-Firewall Anomalies

The ordering of filtering rules in a centralized firewall policy is crucial in determining the filtering policy, because of the sequential manner in which packets are matched against rules. If filtering rules are disjoint, the ordering of the rules is insignificant. However, it is very common to have filtering rules that are inter-related. In this case, if the relative rule ordering is not carefully assigned, some rules may be always screened by other rules producing an incorrect policy. Moreover, when the policy contains a large number of filtering rules, the possibility of writing conflicting or redundant rules is relatively high.

An intra-firewall policy anomaly is defined as the existence of two or more filtering rules that may match the same packet or the existence of a rule that can never match any packet on the network paths that cross the firewall [2]. In this section, we classify different anomalies that may exist among filtering rules in one firewall.

A. Intra-Firewall Anomaly Classification

We can formally describe the type of anomalies that may exist in a firewall policy as follows:

1) Shadowing anomaly: A rule is shadowed when a previous rule matches all the packets that match this rule, such that the shadowed rule will never be activated. Formally, rule R_y is shadowed by rule R_x if:

$$R_x[\text{order}] < R_y[\text{order}], R_x \Re_{\text{EM}} R_y, R_x[\text{action}] \neq R_y[\text{action}]$$
$$R_x[\text{order}] < R_y[\text{order}], R_y \Re_{\text{IM}} R_x, R_x[\text{action}] \neq R_y[\text{action}]$$

For example, Rule 4 in shadowed by Rule 3 in Figure 1. Shadowing is a critical error in the policy, as the shadowed rule never takes effect. This might cause accepted traffic to be blocked or denied traffic to be permitted.

2) Correlation anomaly: Two rules are correlated if they have different filtering actions, and the first rule matches some packets that match the second rule and the second rule matches some packets that match the first rule. Formally, rule R_x and rule R_y have a correlation anomaly if:

$$R_x \Re_{\text{C}} R_y, R_x[\text{action}] \neq R_y[\text{action}]$$

Rule 1 is in correlation with Rule 3 in Figure 1. Note that if the order of these rules is reversed, the policy in effect changes.

3) Generalization anomaly: A rule is a generalization of a preceding rule if they have different actions, and if the first rule can match all the packets that match the second rule. Formally, rule R_y is a generalization of rule R_x if:

$$R_x[\text{order}] < R_y[\text{order}], R_x \Re_{\text{IM}} R_y, R_x[\text{action}] \neq R_y[\text{action}]$$

Rule 2 is a generalization of Rule 1 in Figure 1. These two rules imply that all HTTP traffic that is coming from the address 140.192.37.* will be accepted, except the traffic coming from 140.192.37.20. This anomaly should be treated as only a warning to confirm that the administrator *means* to exclude this specific traffic from the general rule.

4) Redundancy anomaly: A redundant rule performs the same action on the same packets as another rule such that if the redundant rule is removed, the security policy will not be affected. Formally, rule R_y is redundant to rule R_x if:

$$R_x[\text{order}] < R_y[\text{order}], R_x \Re_{\text{EM}} R_y, R_x[\text{action}] = R_y[\text{action}]$$
$$R_x[\text{order}] < R_y[\text{order}], R_y \Re_{\text{IM}} R_x, R_x[\text{action}] = R_y[\text{action}]$$

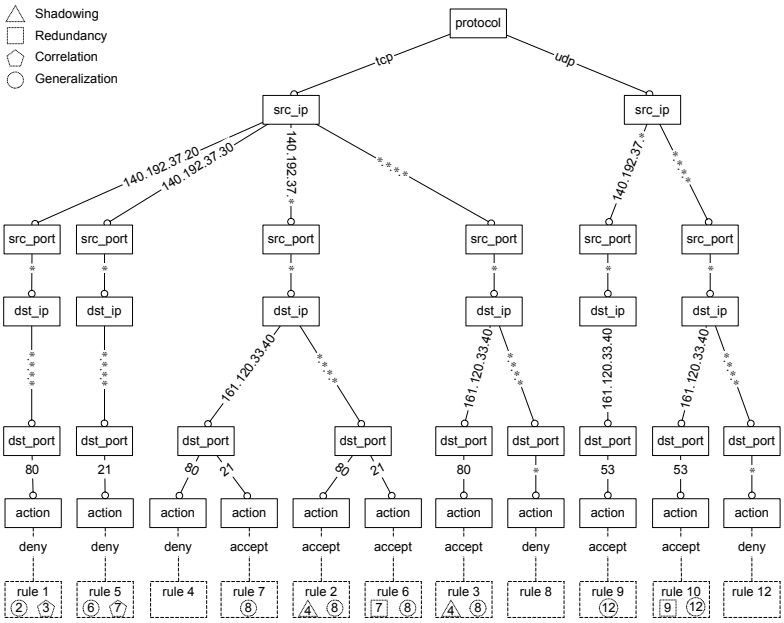

Figure 3. The policy tree for the firewall policy in Figure 1.

Whereas rule R_x is redundant to rule R_y if:

$$R_x[\text{order}] < R_y[\text{order}], R_x \Re_{\text{IM}} R_y, R_x[\text{action}] = R_y[\text{action}]$$
$$\text{and} \quad \nexists R_z \text{ where } R_x[\text{order}] < R_z[\text{order}] < R_y[\text{order}],$$
$$R_x\{\Re_{\text{IM}}, \Re_{\text{C}}\}R_z, R_x[\text{action}] \neq R_z[\text{action}]$$

Referring to Figure 1, Rule 7 is redundant to Rule 6. Redundancy is considered an error in the firewall policy because a redundant rule adds to the size of the filtering rule list, and therefore increases the search time and space requirements of the packet filtering process [5].

5) Irrelevance anomaly: A filtering rule in a firewall is irrelevant if this rule cannot match any traffic that might flow through this firewall. This exists when both the source address and the destination address fields of the rule do not match any domain reachable through this firewall. In other words, the path between the source and destination addresses of this rule does not pass through the firewall. Thus, this rule has no effect on the filtering outcome of this firewall. Formally, rule R_x in firewall F is irrelevant if:

$$F \notin \{n : n \text{ is a node on a path from } R_x[\text{src}] \text{ to } R_x[\text{dst}]\}$$

Referring to Figure 1, Rule 11 is irrelevant because the traffic that goes between the source (140.192.38.*) and the destination (161.120.35.*) does not pass through this firewall. As in irrelevancy, this unnecessarily adds to the size of the policy.

B. Implementation of the Intra-Firewall Anomaly Discovery Algorithm

The state diagram in Figure 4 illustrates intra-firewall anomaly discovery states for any two rules, R_x and R_y, where the two rules are in the same firewall and R_y follows R_x. For simplicity, the address and port fields are integrated in one field for both the source and destination. This reduces the number of states and simplifies the explanation of the diagram.

Initially no relationship is assumed. Each field in R_y is compared to the corresponding field in R_x starting with the protocol, then source address and port, and finally destination address and port. The relationship between the two rules is determined based on the result of subsequent comparisons. If every field of R_y is a subset or equal to the corresponding field in R_x and both rules have the same action, R_y is redundant to R_x, while if the actions are different, R_y is shadowed by R_x. If every field of R_y is a superset or equal to the corresponding field in R_x and both rules have the same action, R_x is potentially redundant to R_y, while if the actions are different, R_y is a generalization of R_x. If some fields of R_x are subsets or equal to the corresponding fields in R_y, and some fields of R_x are supersets to the corresponding fields in R_y, and their actions are different, then R_x is in correlation with R_y. Identifying irrelevant rules requires the knowledge of the network connectivity. Discovering this intra-firewall anomaly is discussed later in Section V-C along with the inter-firewall anomaly discovery algorithm. If none of the preceding cases occur, then the two rules do not involve any anomalies.

The basic idea for discovering anomalies is to determine if any two rules coincide in their policy tree paths. If the path of a rule coincides with the path of another rule, there is a potential anomaly that can be determined based on intra-firewall anomaly definitions in Section IV-A. If rule paths do not coincide, then these rules are disjoint and they have no anomalies. The detailed description of the intra-firewall anomaly discovery algorithm is available in [1]. Applying the

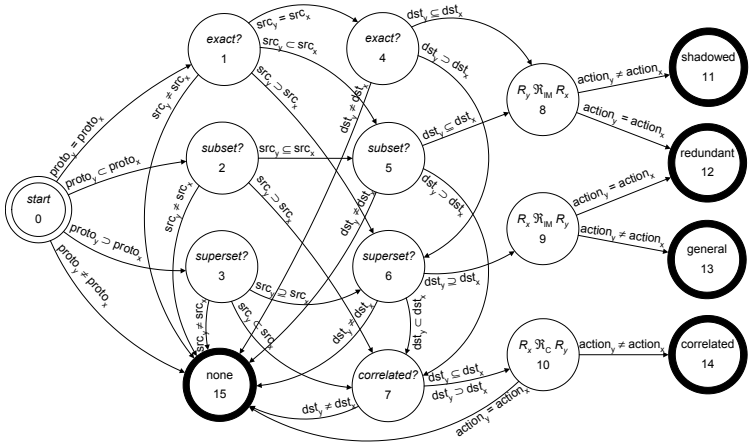

Figure 4. State diagram for detecting intra-firewall anomalies for rules R_x and R_y, where R_y comes after R_x.

algorithm on the rules in Figure 1, the discovered anomalies are marked in the dotted boxes at the bottom of the policy tree in Figure 3. Shadowed rules are marked with a triangle, redundant rules with a square, correlated rules with a pentagon and generalization rules with a circle.

V. Inter-Firewall Anomalies

A. Inter-Firewall Anomaly Definition

In general, an inter-firewall anomaly may exist if any two firewalls on a network path take different filtering actions on the same traffic. We first illustrate the simple case of multiple cascaded firewalls isolating two network sub-domains where the firewalls are installed at the routing points in the network.

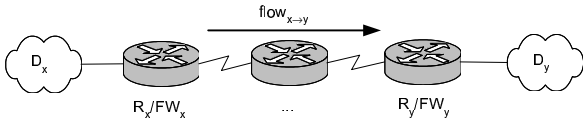

Figure 5. Cascaded firewalls isolating domains D_x and D_y.

Referring to Figure 5, we assume a traffic stream flowing from sub-domain D_x to sub-domain D_y across multiple cascaded firewalls installed on the network path between the two sub-domains. At any point on this path in the direction of flow, a preceding firewall is called an *upstream firewall* whereas a following firewall is called a *downstream firewall*. The closest firewall to the flow source sub-domain (FW_x) is called the *most-upstream firewall*, while The closest firewall to the flow destination sub-domain (FW_y) is called the *most-downstream firewall*.

Using the above network model, we can say that for any traffic flowing from sub-domain D_x to sub-domain D_y an anomaly exists if one of the following conditions holds:

1) The most-downstream firewall accepts a traffic that is blocked by any of the upstream firewalls.
2) The most-upstream firewall permits a traffic that is blocked by any of the downstream firewalls.
3) A downstream firewall denies a traffic that is already blocked by the most-upstream firewall.

On the other hand, all upstream firewalls should permit any traffic that is permitted by the most-downstream firewall in order that the flow can reach the destination.

B. Inter-Firewall Anomaly Classification

In this section, we consider anomalies in multi-firewall environments. Our classification rules are based on the basic case of cascaded firewalls illustrated in Figure 5, assuming the network traffic is flowing from domain D_x to domain D_y. Rule R_u belongs to the policy of the most-upstream firewall FW_x, while rule R_d belongs to the policy of the most-downstream firewall FW_y. We assume that no intra-firewall shadowing or redundancy exists in any individual firewall. As illustrated in Section IV-A, this implies that every "deny" rule should be followed by a more general "accept" rule, and the default action of unspecified traffic is "deny".

1) Shadowing Anomaly: A shadowing anomaly occurs if an upstream firewall blocks the network traffic accepted by a downstream firewall. Formally, rule R_d is shadowed by rule R_u if one of the following conditions holds:

$$R_d \Re_{\mathrm{EM}} R_u, R_u[\text{action}]=\text{deny}, R_d[\text{action}]=\text{accept} \quad (1)$$

$$R_d \Re_{\mathrm{IM}} R_u, R_u[\text{action}]=\text{deny}, R_d[\text{action}]=\text{accept} \quad (2)$$

$$R_u \Re_{\mathrm{IM}} R_d, R_u[\text{action}]=\text{deny}, R_d[\text{action}]=\text{accept} \quad (3)$$

$$R_u \Re_{\mathrm{IM}} R_d, R_u[\text{action}]=\text{accept}, R_d[\text{action}]=\text{accept} \quad (4)$$

Intuitively, in cases (1) and (2), the upstream firewall completely blocks the traffic permitted by the downstream firewall. Rules ($2/FW_2$, $3/FW_1$), and Rules ($8/FW_1$, $4/FW_2$) in Figure 2 are examples of cases (1) and (2) respectively. In cases (3) and (4) the upstream firewall partially blocks the traffic permitted by the downstream firewall. Rules ($7/FW_2$, $7/FW_1$), and Rules ($5/FW_2$, $5/FW_1$) in Figure 2

are examples of cases (3) and (4) respectively.

2) Spuriousness Anomaly: A spuriousness anomaly occurs if an upstream firewall permits the network traffic denied by a downstream firewall. Formally, rule R_u allows spurious traffic to rule R_d if one of the following conditions holds:

$$R_u \Re_{\mathrm{EM}} R_d, R_u[\text{action}]=\text{accept}, R_d[\text{action}]=\text{deny} \quad (5)$$
$$R_u \Re_{\mathrm{IM}} R_d, R_u[\text{action}]=\text{accept}, R_d[\text{action}]=\text{deny} \quad (6)$$
$$R_d \Re_{\mathrm{IM}} R_u, R_u[\text{action}]=\text{accept}, R_d[\text{action}]=\text{deny} \quad (7)$$
$$R_d \Re_{\mathrm{IM}} R_u, R_u[\text{action}]=\text{accept}, R_d[\text{action}]=\text{accept} \quad (8)$$
$$R_u \Re_{\mathrm{IM}} R_d, R_u[\text{action}]=\text{deny}, R_d[\text{action}]=\text{deny} \quad (9)$$

In cases (5) and (6), the rule R_u in the upstream firewall permits unwanted traffic because it is completely blocked by R_d in the downstream firewall. Examples of these cases are Rules (2/FW_1, 4/FW_0), and Rules (2/FW_1, 9/FW_2) in Figure 2 respectively. In cases (7) and (8) part of the traffic allowed by rule R_u in upstream firewall is undesired spurious traffic since it is blocked by rule R_d in the downstream firewall. Examples of these cases are also found in Rules (5/FW_2, 4/FW_1), and (3/FW_2, 3/FW_1) in Figure 2 respectively. Case (9) is not as obvious as the previous cases and it needs further analysis. Since we assume there is no intra-firewall redundancy in the upstream firewall, the fact that R_u has a "deny" action implies that there exists a superset rule in the upstream firewall that follows R_u and accepts some traffic blocked by R_d. This occurs when the implied "accept" rule in the upstream firewall is an exact, superset or subset match (but not correlated) of R_d. Rules (5/FW_0, 4/FW_1) in Figure 2 are an example of this case.

3) Redundancy Anomaly: A redundancy anomaly occurs if a downstream firewall denies the network traffic already blocked by an upstream firewall. Formally, rule R_d is redundant to rule R_u if, on every path to which R_u and R_d are relevant, one of the following conditions holds:

$$R_d \Re_{\mathrm{EM}} R_u, R_u[\text{action}]=\text{deny}, R_d[\text{action}]=\text{deny} \quad (10)$$
$$R_d \Re_{\mathrm{IM}} R_u, R_u[\text{action}]=\text{deny}, R_d[\text{action}]=\text{deny} \quad (11)$$

In both of these cases, the deny action in the downstream firewall is unnecessary because all the traffic denied by R_d is already blocked by R_u in the upstream firewall. In Figure 2, Rules (6/FW_2, 6/FW_1), and Rules (9/FW_2, 6/FW_0) are examples of cases (10) and (11) respectively.

4) Correlation Anomaly: A correlation anomaly occurs as a result of having two correlated rules in the upstream and downstream firewalls. We defined correlated rules in Section III-A. Intra-firewall correlated rules have an anomaly only if these rules have different filtering actions. However, correlated rules having any action are always a source of anomaly in distributed firewalls because of the implied rule resulting from the conjunction of the correlated rules. This creates not only ambiguity in the inter-firewall policy, but also spurious, and shadowing anomalies. Formally, the correlation anomaly for rules R_u and R_d occurs if one of the following conditions holds:

$$R_u \Re_{\mathrm{C}} R_d, R_u[\text{action}]=\text{accept}, R_d[\text{action}]=\text{accept} \quad (12)$$
$$R_u \Re_{\mathrm{C}} R_d, R_u[\text{action}]=\text{deny}, R_d[\text{action}]=\text{deny} \quad (13)$$
$$R_u \Re_{\mathrm{C}} R_d, R_u[\text{action}]=\text{accept}, R_d[\text{action}]=\text{deny} \quad (14)$$
$$R_u \Re_{\mathrm{C}} R_d, R_u[\text{action}]=\text{deny}, R_d[\text{action}]=\text{accept} \quad (15)$$

An example for case (12) is
R_u : tcp, 140.192.$*$.$*$, any, 161.120.33.$*$, 80, accept
R_d : tcp, 140.192.37.$*$, any, 161.120.$*$.$*$, 80, accept

In this example, effectively, the correlative conjunction of these two rules implies that only the traffic coming from 140.192.37.* and destined to 161.120.33.*. will be accepted as indicated in the following implied rule R_i
R_i : tcp, 140.192.37.$*$, any, 161.120.33.$*$, 80, accept
This means that other traffic destined to 161.120.*.* will be shadowed at the upstream firewall, while spurious traffic originating from 140.192.*.* will reach the downstream firewall.

For case (13) the example is
R_u : tcp, 140.192.$*$.$*$, any, 161.120.33.$*$, 80, deny
R_d : tcp, 140.192.37.$*$, any, 161.120.$*$.$*$, 80, deny

In this case, the resulting action at the downstream firewall will deny the traffic coming from 140.192.37.* and destined to 161.120.33.*. The implied filtering rule R_i will be
R_i : tcp, 140.192.37.$*$, any, 161.120.33.$*$, 80, deny
This means that other traffic originating from 140.192.*.* will be shadowed at the upstream firewall, while spurious traffic destined to 161.120.*.* may reach the downstream firewall. A possible resolution for cases (12) and (13) is to replace each of the correlated rules with the implied filtering rule R_i.

The example for case (14) is
R_u : tcp, 140.192.$*$.$*$, any, 161.120.33.$*$, 80, accept
R_d : tcp, 140.192.37.$*$, any, 161.120.$*$.$*$, 80, deny

This example shows that the resulting filtering action at the upstream firewall permits the traffic that is coming from 140.192.37.* and destined to 161.120.33.*. However, the same traffic is blocked at the downstream firewall, resulting in spurious traffic flow. To resolve this anomaly, an extra rule R_i should be added in the upstream firewall prior to R_u such that it blocks the spurious traffic as follows
R_i : tcp, 140.192.37.$*$, any, 161.120.33.$*$, 80, deny

As for case (15), the example is
R_u : tcp, 140.192.$*$.$*$, any, 161.120.33.$*$, 80, deny
R_d : tcp, 140.192.37.$*$, any, 161.120.$*$.$*$, 80, accept

This example shows a different situation where the resulting filtering action at the upstream firewall will block the traffic that is coming from 140.192.37.* and destined to 161.120.33.*. However, because this traffic is accepted at the

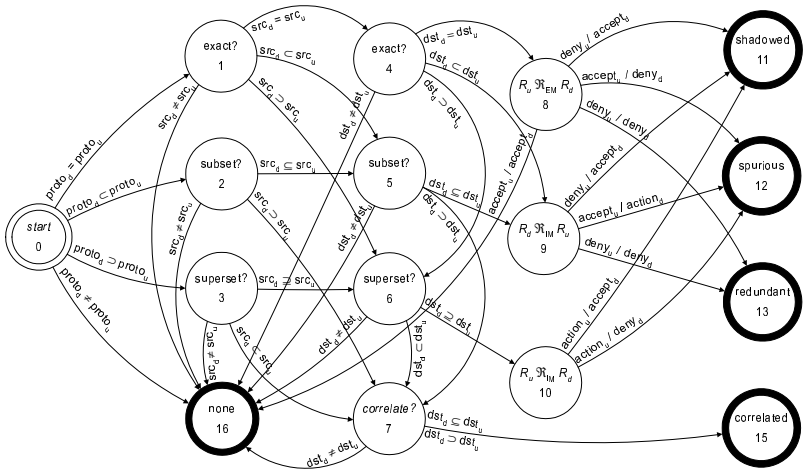

Figure 6. State diagram for inter-firewall anomaly discovery for rules R_u and R_d, where R_u belongs to the upstream firewall and R_d belongs to the downstream firewall.

downstream firewall, R_d is shadowed by R_u. To resolve this anomaly, an extra rule R_i should be added in the upstream firewall before R_u to avoid the shadowing anomaly as follows

R_i : tcp, 140.192.37.$*$, any, 161.120.33.$*$, 80, accept

In the following theorem, we show that the anomaly cases we presented above are covering all the possible inter-firewall anomalies. A complete proof of the theorem is provided in [2].

Theorem 3: This set of anomalies represent all filtering anomalies that might exist between any two rules rules each in a different firewall.

C. Implementation of the Inter-Firewall Anomaly Discovery Algorithm

This algorithm finds the rule relations described in Section V-B and discovers the anomalies between filtering rules in two or more connected firewalls. In Figure 6 we show the state diagram of the inter-firewall anomaly discovery algorithm. The figure shows the anomaly discovery for any two rules, R_u and R_d, where R_u is a rule in the upstream firewall policy, and R_d is a rule in the downstream firewall policy. For simplicity, the address and port fields are integrated in one field for both the source and destination. At the start state, we assume no relationship between the two rules. Each field in R_d is compared to the corresponding field in R_u starting with the protocol then source and destination addresses and ports. Based on these comparisons, the relation between the two rules is determined, as well as the anomaly if it exists. For example, if R_u is found to inclusively match R_d (State 10), then R_d is partially shadowed if its action is "accept" (State 11), or R_u is spurious if the action of R_d is "deny" (State 12).

Since more than two firewalls may exist between sub-domains in an enterprise network, the inter-firewall anomaly discovery process should be performed on all firewalls in the path connecting any two sub-domains in the network. For example, in Figure 2, inter-firewall anomaly analysis is performed on (FW_1, FW_0) for all traffic that goes between $D_{1.2}$ and the Internet, on (FW_2, FW_0) for all traffic that goes between $D_{2.2}$ and the Internet, and on (FW_1, FW_0, FW_2) for all traffic that goes between $D_{1.2}$ and $D_{2.2}$. Although we use a hierarchical network topology example, this analysis can be performed on any network topology as long as there is a fixed route between source and destination sub-domains.

Intuitively, inter-firewall anomaly discovery is performed by aggregating the policy trees presented in Section III-B for all the firewalls isolating every two sub-domains in the network. The algorithm takes as an input the list of network paths between sub-domains. For each path, we determine all the firewalls in the traffic flow. Then for every firewall in the path, we first run the intra-firewall anomaly discovery algorithm described in Section IV-B to ensure that every individual firewall is free from intra-firewall anomalies. Next, we build the policy tree of the most upstream firewall and then add into this tree the rules of all the consecutive firewalls in the path. During this process, only the rules that apply to this path (have the same source and destination) are selected and marked. Eventually, as a result of applying the algorithm on all the network paths, the rules that potentially create anomalies are reported. In addition, any rule left unmarked is reported as an irrelevant rule anomaly as it does not apply to any path in the network. The complete description of the inter-firewall anomaly discovery algorithm is provided in [1].

As an example, we apply the inter-firewall anomaly discovery algorithm on the example network in Figure 2. We start by identifying the participating sub-domains in the network given the network topology and routing tables. The domains in the figure are $D_{1.1}$, $D_{1.2}$, $D_{2.1}$, $D_{2.2}$ in addition to the global Internet domain. The Internet domain is basically any address that does not belong to one of the network sub-domains. Afterwards, we identify all the possible directed paths between any two sub-domains in the network and

determine the firewalls that control the traffic on that path, and we run the algorithm on each one of these paths. According to the figure, the algorithm analyzes 20 distinct paths for inter-firewall anomalies and produces the anomalies indicated in Section V-B.

VI. Dynamic Configuration of Firewall Rules

Algorithms have been given in [2] to automate the detection of both types of anomalies. Based on these algorithms it is possible to safely automate the reconfiguration of firewall policies based on emerging perceived threats. In particular, it would be of great value to develop a security system that can re-posture itself in response to the detection of a worm attack. The target of the re-posturing is to minimize, if not stop the worm infection. At a high level, this is a typical control problem. The technique devised above -the detection of firewall rule anomalies- will constitute the essence of safe actuating of control actions. In addition to actuators, a control system requires sensors, to estimate the state of the system; and a controller, to select which action to apply in response to a perceived state. A conceptual realization is shown in figure 7. In the following sections, we will discuss our current vision for the realization of each function. The goal is to show examples of the existing solutions, or developments in that area, and how they are related to our envisioned autonomous system.

VII. State Estimation

To correctly respond to a worm threat, the system must first correctly sense the threat. From a control perspective, this is analogous to the function of a sensor. An element that monitors the activity of all hosts on the network is needed in order to be able to closely estimate the networks' state. There must also be a mathematical basis for quantifying the threat level associated with every host and/or connection.

As for the sensing function, a wide range of research papers have been published featuring proposed techniques to sense a network attack. Some of this research focuses on detecting attacks from the perspective of a single host, while other research focuses on sensing attacks from the network perspective. The network resource that detects illicit activity is normally falls under the category of Intrusion Detection Systems (IDS). IDS's typically come in two types: anomaly-based, and signature-based. Signature-based IDS's are only effective against well known worms, while anomaly-based IDS's are somewhat effective against new worm threats.

A. Signature-based Intrusion Detection

A signature is a set of strings that a worm uses in its code. Once a worms signature is known, it is fairly easy to detect a worm in transit using a signature-based IDS. The shortcoming of this approach is that it is incapable of detecting new worm infections. Another problem lies in the existence of polymorphic worms. These are worms that use a polymorphic engine to generate a new looking string each time in order to elude IDS's.

A basic limitation of signature-based systems is that signatures are generated manually. This means that the speed by which a signature can be generated is limited by human limits. Finding and distributing a signature for a new worm normally consumes enough time for the worm to have infected most of its target population. As will be mentioned later, there are emerging technologies that hold some promise in automating this process.

B. Anomaly-based Intrusion Detection

Anomaly-based intrusion detection is intended to discover new unknown attacks. In the context of worm attacks, anomaly-based detection is designed to detect new worm infections for which no signature is known yet. Anomaly-based systems rely on forming a profile of normal activity in a particular network during a "training" period. Then the instantaneous traffic profile is compared against that normal profile to detect any abnormality. A kin of anomaly-based detection is misuse-based detection. The difference between the two is in the reference of comparison. While anomaly-based systems compare traffic to the normal profile, misuse-based systems compare traffic to the profile of an attack. In other words, anomaly-based systems trigger an alarm if the traffic profile moves away from that of normal behavior, while misuse-based systems trigger an alarm if the traffic profile moves closer to an attack profile. The performance of these systems normally depend on setting a threshold for the "distance" that traffic is allowed to deviate/come within from normal/attack behavior.

The training of anomaly/misuse-based systems, threshold setting techniques, distribution of the detection function, in addition to other related issues are all open research issues. Advancements in these areas will bring us closer to an autonomous defense system to counter worms.

C. Other Related Technologies

Other research has been done in developing network resources that do not precisely fall under the category of IDS's. These developments use heuristics to detect intrusions, and some of them even work to slow down or stop them, but they do not formally fall under the categorizations of IDS's. Some of the research that focused on host-based detection (detect whether or not this host is infected) include [9] and [6]. In [9], a simple throttling technique is used for all outgoing connections on a host. The throttling process monitors the rate r at which the host initiates new connections. To determine the "newness" of the connection, a history of the last n connections is kept. If a process on that host tries to initiate a connection to a new destination, that request is queued to insure that the rate of new connections does not exceed r. Given that a worm tries to connect to as many *new* hosts as possible in as little time as possible, the rate at which the queue grows can be used as an indicator of a worm infection. Both n and r are configurable to suit different hosts, services and applications. In [6], the fact that most infecting mechanisms depend on forcing the processor to return from a function call to the

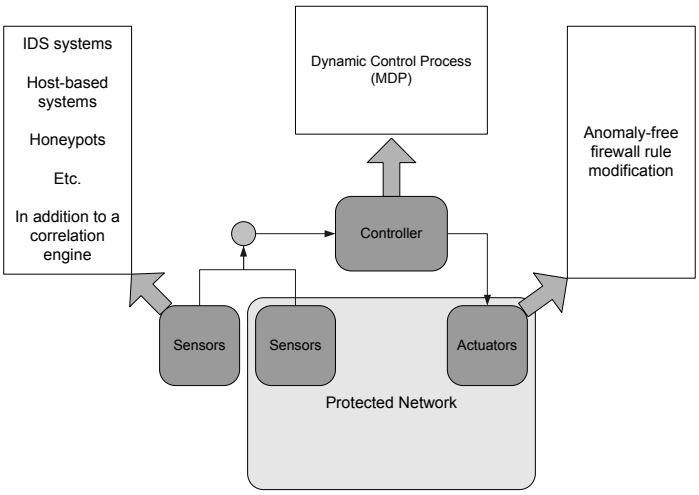

Figure 7. General worm control system architecture

address of injected code is utilized. All return addresses are computed statically before run time, then compared to the return addresses at run time to insure they are the same. A difference in return addresses indicates a suspicious operation, and triggers termination of the violating process. Several other host-based detection and control techniques exist, and may be utilized in a general worm control framework. The output of such host-based detectors can be incorporated into the input given to the controller of the autonomous defense system.

Other research has focused on network-based sensing of attacks. This category includes a very important emerging network resource, namely honeypots. The idea of honeypots is to provide an attacker with a decoy machine and learn about the method of attack. In the context of worms, there is a potential to learn many things about how a worm gains access to the network, what services it uses, the rate at which it propagates, as well as a lot of other information about the worm. A honeypot is assigned all the unused addresses in a network. A typical worm tries to infect computers by connecting to randomly generated IP addresses. Thus, there is normally a good possibility that it will try to infect a honeypot. By definition, all communication to a honeypot is suspicious.

As honeypots are a quickly emerging technology, considerable research wirk is being done on finding new applications, and improvements for them. An example that ties into our objective is given in [3]. The authors propose a technique by which the generation of worm signatures can be automated. In the context of our autonomous control system, this can constitute an important ”learning” function for the system. If the system can efficiently and automatically learn the signatures of *new* worms, the signature pool in signature-based IDS’s can be swiftly updated, and attacks can be stopped with minimal damage. Another example of the potential of this emerging technology is demonstrated in [7]. In this paper, rather than generating worm signatures, software patches are automatically generated in response to an attack. The system learns about the exploit that the worm uses from the honeypot, then automatically generates a patch that is applied to the vulnerable system. In this way the system is automatically immunized to the worm attack. The success rate achieved in [7] is 82%, with a maximum patch generation time of 8.5 seconds.

D. Estimate Generation and the Correlation Function

As will be discussed in section VIII, the controller must have an accurate estimate of which hosts are healthy and which hosts are infected. Input from different systems (IDS’s, throttling, DOME, honeypots...etc.) must be represneted numerically, and given proper weight based on the fidelity of the specific source system. The goal is to find a statistical model by which we can minimize the probability of a false positive or a false negative from the combined input of all the sensing techniques employed. This model should provide us with a risk factor associated with a host/network, based on the collective inputs. A step in that direction was made in [8]. In that paper, the function threatlevel() is proposed such that threatlevel(IP) evaluates to a numerical value that corresponds to the threat level posed by that IP address.

A basic function that will be included in this module is the correlation function. To detect a certain intrusion with a level of confidence, separate events occurring on all hosts will have to be correlated. For example, assume that host A is suspected of being infected (say it was involved in a communication with a honeypot). Let us now assume that host A communicates with host B. Immediately after that, host B starts acting suspiciously. In that case, this series of events should be correlated and information must be deduced based on that. In other words, the threat level of host A must be modified based on its apparent infection of host B. This correlation function is essential to accurate state estimation.

VIII. Controller

The controller function is extremely dependent on having a clear view of the system state. The obvious analogy is that of a car driver being dependent on having a clear view the cars position on the road. Assuming that the state estimator is capable of supplying such an accurate image to the controller, the controller must steer the system to a desired state. If an intrusion (worm activity in specific) is detected, the controller will need to calculate the best response available. The most basic and crucial response will be in the form of dynamic firewall rule modification. To implement this, the system will have to decide where to apply the rule modification (which firewall?), what kind of modification is needed(which rule to insert or delete or edit?), and whether other actions may be needed. A flexible controller design must be able to choose between several actions based on a set of parameters. For example, the response should be different for different levels of risk returned by the state estimation module. The response may also change if the system does not respond to a previous action ordered by the controller.

Examples of actions that a controller may choose from include:

- Dynamically quarantine a host that acts suspiciously for a short period of time T. When T has elapsed, release the host and monitor its activity. If the activity is still suspicious, the controller may choose to quarantine it for a longer period of time, or to take a harsher action - depending on the threat level- [10].
- Statically quarantine a suspicious host by inserting firewall rules that deny any traffic coming from it.
- In case there was suspicion that other hosts that came in contact with the infected host have become infected, it is possible for the controller to take precautionary action by quarantining these hosts as well.
- A suspicious host with a low threat level can be prevented from accessing vital network servers as an extra precaution.
- In addition to firewall rule modification, a technique such as throttling can be used to dynamically modify the allowed connection rate in response to perceived elevated risk associated with a given host.
- In all cases, administrators should be alerted to the threat so that proper manual action can be taken.

At this stage the choice of activity should be selected such that the problem is manageable. A wide selection of actions will complicate the controller function and make it too slow to respond. Thus, the dynamic modification of firewall rules constitutes the most reasonable starting point.

An interesting statistical control technique is the Markov Decision Process (MDP). In [4] it was used in a host-based worm control system. Our current vision is to generalize this technique to control network-based autonomous defense.

A brief overview of MDPs will be presented next. Some understanding of this powerful control mechanism is necessary to understand its role in our autonomous defense system. The MDP is a statistical control model that is composed of the following:

1) Decision epochs: The instances of time at which the controller makes decisions. In a discrete environment, decisions are made at all decision epochs. A decision by the controller is a choice of action from a set of available actions. In our case, the decision may involve quarantining a host, or a whole subnet for example.
2) States: A set of possible states in which the system can be. These states are one of the factors that a controller considers when making a decision. In our case, the state might consist of a threat level vector in which each host is represented as an element. To make the number of states finite, we might need to limit the granularity of the threat level measure.
3) Actions: A set of actions that the controller can perform to influence the system. This set may be state dependent, in which case the set of actions changes from one state to the other, or may be state independent. As mentioned above, the actions in our case include quarantining a host or a subnet for a specific amount of time or permanently (until the problem is resolved).
4) Rewards: At any state, the choice of action by the controller accrues a specific reward. This measure is used to model the decision process as an optimization problem. The goal in that optimization problem is to find the set of actions that will maximize the total reward. Here again, the reward function may be state dependent. The reward in our case might be measured as an index that balances the drop in threat level with the loss of functionality. A good measure of immediate rewards is essential for MDPs.
5) Transition probabilities: At each state, the controller has to have a sense of which state the system will end up in if an action A is chosen. This information is provided by the transition probability. As with the rewards and actions, the transition probability can be state dependent. In other words, the transition probabilities answer the questions: If I'm in state S_1, and I take action A_m, what is the probability that I end up in state S_n, where n and m range over the set of available states and actions respectively.

The ultimate goal of the MDP powered controller is to arrive at a policy (a sequential set of actions)by which the total reward is maximized. If the reward function is modelled correctly, this should correspond to minimizing risk, while preserving functionality of the system. Arriving at such a controller is another objective of this research effort.

IX. Safe Actuators

As mentioned before, our actuation will consist of modules that allow the controller to modify firewall rules. And interface can be defined for these modules for the controller to interact with. One of the core components in these modules must be an implementation of the anomaly discovery algorithm. If we allow the controller to edit firewall rules without checking for

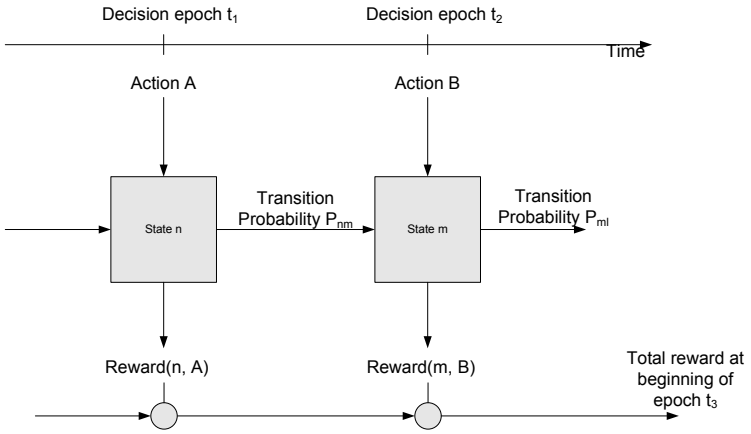

Figure 8. Markov decision process model

anomalies, the system will end up having security holes and/or run inefficiently. In order to fully automate this, an algorithm for anomaly resolution will have to be designed. This will allow the actuators to automatically resolve any anomalies discovered.

X. Summery and Future Work

To summarize, this paper discussed the evolution of our research effort at the MNLab. After succeeding in finding firewall rule anomaly discovery algorithms, our efforts are aimed towards extending our research to viable applications that utilize this algorithm. The current vision is to work towards developing an autonomous defense system to counter Internet worm attacks. To achieve that goal, many research issues must be addressed. As was mentioned throughout the paper, these research issues include:

- Developing suitable intrusion detection systems for the accurate detection of worms. This is directly related to the state estimation problem.
- Research in the area of threat level assessment to numerically combine and represent the outcome of intrusion detection devices. This is also part of the state estimation problem.
- Efficient modelling of the worm control problem as an MDP.
- Developing algorithms to resolve anomalies in firewall rules to allow for automatic modifications.

As our research develops, a subset of these problems might prove challenging and valuable enough to warrant limiting our research to that subset. In any case, advancement in any of the aforementioned areas will be a significant advancement towards autonomous worm defense systems.

References

[1] E. Al-Shaer and H. Hamed. ”design and implementation of firewall policy advisor tools”. Technical Report CTI-TR-02-006, DePaul CTI, August 2002.

[2] E. Al-Shaer and H. Hamed. ”discovery of policy anomalies in distributed firewalls”. In *Proceedings of IEEE INFOCOM’04*, March 2004.

[3] Christian Kreibich and Jon Crowcroft. ”honeycomb: Creating intrusion detection signatures using honeypots”. In *CCR Paper Comment, Discussion, and Update Forum*, volume 34. ACM Press, 2004.

[4] O. Patrick Kreidl and Tiffany M. Frazier. ”feedback control applied to survivability: A host-based autonomic defense system”. *IEEE TRANSACTIONS ON RELIABILITY*, 53(1):148–166, March 2004.

[5] R. Panko. *Corporate Computer and Network Security*. Prentice Hall, 2003.

[6] Jesse C. Rabek, Roger I. Khazan, Scott M. Lewandowski, and Robert K. Cunningham. ”detection of injected, dynamically generated, and obfuscated malicious code”. In *ACM CCS Workshop on Rapid Malcode (WORM’03)*, Washington, DC, October 2003.

[7] Stelios Sidiroglou and Angelos D. Keromytis. ”countering network-worms through automatic patch generation”. *IEEE security and privacy*, 2004.

[8] Lawrence Teo, GailJoon Ahn, and Yuliang Zheng. ”dynamic and riskaware network access management”. In *Proceedings of the ninth ACM symposium on Access control models and technologies*. ACM Press, 2003.

[9] M. Williamson. ”throttling viruses: Restricting propagation to defeat malicious mobile code”. Technical Report HPL-2002-172, Information Infrastructure Laboratory, HP Laboratories Bristol, June 2002.

[10] Cliff C. Zou, Weibo Gong, and Don Towsley. ”worm propagation modeling and analysis under dynamic quarantine defense”. In *ACM CCS Workshop on Rapid Malcode (WORM’03)*, Washington, DC, October 2003.