Dynamic and Risk-Aware Network Access Management

Lawrence Teo[*] Gail-Joon Ahn[†] Yuliang Zheng[*]
Laboratory of Information Integration, Security, and Privacy (LIISP)
University of North Carolina at Charlotte
9201 University City Blvd, Charlotte, NC 28223, USA
http://www.sis.uncc.edu/LIISP/
{lcteo,gahn,yzheng}@uncc.edu

ABSTRACT

Traditional network security technologies such as firewalls and intrusion detection systems usually work according to a static ruleset only. We believe that a better approach to network security can be achieved if we use quantified levels of risk as an input. In this paper, we describe a dynamic access control architecture which uses risk to determine whether to allow or deny access by a source connection into the network. A simulation of our architecture shows favorable and promising results.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*security and protection*; C.2.3 [**Computer-Communication Networks**]: Network Operations—*network management, network monitoring*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*authentication, invasive software, unauthorized access*

General Terms

Security, management, algorithms

Keywords

Dynamic access control, network management, risk, risk awareness, role

1. INTRODUCTION

Traditional network security technologies, particularly firewalls and intrusion detection systems (IDSs), usually work in a static manner. By "static", we mean that they do not change their behavior in response to certain events. Consider a typical packet-filtering firewall. It first reads the rulesets as specified by the administrator. When packets arrive, it compares the header and content of the packet with the previously defined rules. If the packet is allowed according to the ruleset, the firewall would allow the packet to go through; conversely, if the rules do not allow the packet to go through, the packet is denied. The point that we can observe here is that the firewall either allows or denies the packet based on an "all-or-nothing" approach.

Let us consider another example. Suppose we have a signature-based IDS installed in the network. Like the firewall, the IDS initially reads in signatures defined and customized by the administrator. Then, as packets arrive into the network, the IDS either logs the connections as intrusions or legitimate traffic. A connection is either an intrusion or is legitimate; there can be no "in-betweens."

We believe that a static decision-making approach based on static input, as described earlier, is not sufficient to achieve full network security. This is evident when we examine the problems posed by the static approach in everyday situations. Firewalls still allow malicious traffic to enter the network. For example, a common firewall policy is to allow connections through port 80, thus allowing malicious programs like Code Red to attack web servers. Intrusion detection systems still produce false positives. There is no way to change this unless the static firewall ruleset or IDS signatures are updated manually by the administrator.

The inefficiency of the static approach is clearly evident when we examine the fastest computer worm to date – the SQL Slammer/Sapphire worm [9] that spread around the world in just 10 minutes on January 25, 2003. The worm exhibited highly unusual traffic which was easily recognizable. However, human intervention was still required to update firewalls and routers to prevent the worm from spreading further. Now that high speed worms are no longer a theoretical threat, it is clear that human-mediated counter-responses, as required by the static approach, is no longer feasible [13].

To address these problems, there is a need to use a dynamic approach to determine threats. A dynamic approach would be able to intelligently decide if a source is a threat or not, based on dynamic conditions in the network, as opposed to static rules. This enables us to allow the "in-betweens."

[*]also affiliated with: Calyptix Security Corporation, P.O. Box 561508, Charlotte, NC 28256, USA. http://www.calyptix.com/

[†]The work of Gail Ahn has been partially supported by the grants from National Science Foundation (NSF-CCR-0124873) and National Security Agency (MDA904-02-1-0218).

The dynamic approach would use risk as an input to adapt to varying network conditions. In contrast, static rules may not be relevant in certain conditions (recall the Code Red example). Risk refers to how much or how little a source can be trusted.

The need to use risk and a dynamic approach is especially evident when we draw a few examples from the real world. Risk is generally determined using two criteria in the real world: measures and signs. A measure here refers to a quantifiable amount that represents risk. Suppose a person would like to apply for a bank loan. In order to determine whether the applicant can be trusted to repay the loan or not, the bank looks at the credit rating, which is a measure of how much risk the applicant possesses. Another example can be seen when a student applies for a scholarship. The student's Grade Point Average (GPA) or similar measure would highly affect the student's chances of being awarded the scholarship. The other criterion for determining risk has to do with signs. Suppose a salesperson is attempting to sell a product to a customer. Signs such as the salesperson's appearance, communication skills, and dressing would all play a part in determining whether the sale would be successful or not.

In our project, we aim to build a security architecture that uses a dynamic access control scheme to perform risk-aware network security management. This system would use the "in-betweens" approach as opposed to the "all-or-nothing" approach. Our objective is to examine the feasibility of using a dynamic access control scheme to perform network security management. We also investigate the use of *risk* as a more effective way to identify threats. Based on these objectives, we aim to design a new approach to complement existing firewalls and intrusion detection systems. It should be stressed that we are choosing to make this a complementary approach rather than a replacement for firewalls and IDSs. The reason is that it is unlikely that organizations would replace their existing network security mechanisms with something entirely new. Therefore, we are introducing a new entity into the network, not to replace current security systems, but to complement them.

The rest of the paper is organized as follows. Section 2 describes the background and related work. Section 3 describes the dynamic access control architecture. We discuss the policy specification and policy enforcement in Sections 4 and 5 respectively. A simulation of the architecture, along with the results, are described in Section 6. A discussion of issues is given in Section 7. Ongoing and future work is suggested in Section 8, before the conclusion in Section 9.

2. BACKGROUND AND RELATED WORK

The notion of dynamic access control can be traced back to early work done by Thomas and Sandhu [14, 15, 16]. Traditional access control mechanisms employ a subject-object view. Thomas and Sandhu argued that this view is no longer sufficient for newer systems. In order to cater for distributed and multi-tiered applications, access control has to be performed at a higher level of abstraction. They called their approach task-based access control.

Another project named Seraphim [10] is a generic, theoretical framework for dynamic policy specification and enforcement. The difference between Seraphim and our work is that Seraphim is a generic framework, while we are concentrating more specifically on the network environment.

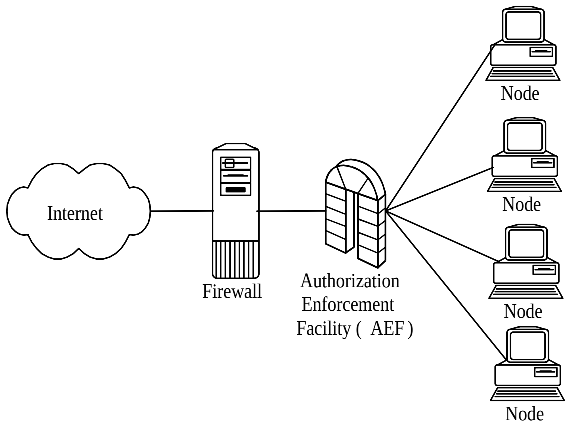

Figure 1: Conceptual view of the network.

Knorr [6] did some work on performing dynamic access control using petri net workflows. There are other related projects on dynamic access control in different contexts. For example, Lin, Lee, and Chang [7] described a dynamic access control mechanism in the area of information protection systems. Harn and Lin [3] and Yen and Laih [18] developed dynamic access control schemes that enforce user authentication, based on public key cryptography.

3. ARCHITECTURE

This section describes the dynamic access control architecture that enables risk-aware network access management. We will use the conceptual view of the network in Figure 1 to describe the architecture. The diagram shows a typical network with a firewall and various nodes behind the firewall. Please note that this is a packet-filtering firewall. However, we have now introduced a new entity called the Authorization Enforcement Facility (AEF) which would be used to realize the ideas described in this paper. More specifically, the AEF is mainly used for policy enforcement and enable risk-aware network access management.

Before we describe the responsibilities of the AEF, we will first discuss why risk-aware network access is important in the context of this architecture. Let's begin with a discussion of the role of a typical firewall in an organizational network. The firewall's main role is to filter traffic based on an administrator-defined ruleset. For example, the ruleset may specify that access to certain network ports is not allowed, or that external packets with internal IP addresses arriving from the Internet would not be allowed.

Although this is the standard function of the firewall at present, there are still critical issues with it that need to be resolved. One significant problem is that although certain traffic may look legitimate, they may actually be malicious. Such traffic would still bypass the firewall. Therefore, although traffic may be allowed to pass through by the firewall, we still cannot be totally sure about its intentions. A clear indication of this problem is evident when we look at common attacks today, such as worms like Code Red. Code Red is successful in penetrating packet-filtering firewalls, because it attacks webservers with traffic looking like regular HTTP connections to port 80 (which is a commonly allowed port in most organizations). Other types of attacks which can bypass firewalls by posing as legitimate traffic are rogue web services and malformed packets (for example, TCP packets with non-standard flag combinations) that are not filtered by the firewall ruleset.

With risk-aware network access management, we can mitigate this problem by examining the traffic to determine how risky it is, even if it looks legitimate. This is the function of the AEF. The AEF analyzes incoming traffic and determines the amount of risk associated with each source. If the risk is low, the AEF would allow the connection to reach its intended destination node. If the risk is high, the AEF would deny access to the destination node.

Another important point needs to be made before we move on to the rest of this section. A firewall is normally concerned with the authorization of traffic. The firewall ruleset describes what the input packets can access. However, from the standpoint of this paper, it is probably more accurate to think of the firewall as performing *authentication* instead of authorization. The firewall "authenticates" the traffic and ensures that only traffic that is recognized as legitimate passes through. The AEF, on the other hand, would perform *authorization.* The AEF only works with traffic that has been authenticated by the firewall. The AEF determines whether these already-authenticated connections are actually allowed to access their intended destinations or not. The four key points are:

- The firewall authenticates input traffic, by ensuring that only traffic that is allowed by the ruleset can pass. The firewall performs *authentication.*

- The AEF only works with traffic that has been authenticated by the firewall. The AEF is concerned with *authorization.*

- The AEF performs risk-aware network access management, by determining the risk with each source connection and allowing or denying it to access its destination nodes based on its risk.

- All traffic must pass through the AEF for authorization.

3.1 Elements

This section describes the elements in the architecture. We will first describe nodes, services, and sources. This is followed by a discussion of node values and service values. We then show how roles relate to the nodes and services.

3.1.1 Nodes, Services, and Sources

To protect a network using risk-aware network access management, we first need to determine what our assets are, and which ones we try to protect. For the purposes of this paper, the assets to be protected are nodes and services.

The first basic element in the architecture is a *node.* We use the term "node" in the same way that it is used in networking; it refers to a particular machine or device in the network. For example, a node may be a workstation, web server, mail server, laptop, and so on.

The second element is a *service.* A service refers to the network and Internet services provided by the node. For example, a webserver node may offer services such as HTTP, SSH, and FTP.

The nodes and services act as the destinations for incoming traffic. For example, a source S may wish to connect to the HTTP service on destination node A.

We use the term *source* to denote a generic input into the architecture. For example, a source may be a packet, connection (a series of related packets), or a stream. The definition of source is intentionally left generic so that it can be used to represent different kinds of input, as the need arises.

3.1.2 Node Values and Service Values

In a typical organization, not all nodes are equal. An administrator would take more care to set up security measures on an important node such as a mission critical production webserver, than s/he would on an employee workstation. So, how do we determine if one node is more valuable than another?

This question is addressed by the next element: the *node value.* A node value is a measurable quantity that states how valuable a node is. This concept is best explained using an example. Suppose the node value is measured on a scale of 1 to 10, where 1 represents least importance and 10 represents critical importance. Nodes such as unimportant anonymous FTP servers or honeypots would be assigned a node value of 1, which means that they are not important. In contrast, nodes such as production servers would be assigned a high node value like 9 or 10 to reflect their importance. The specific value to assign to each node is dependent on the organizational security policy.

Like nodes, each service can be assigned a *service value*, which states how valuable the service is. It should be noted that the service value is dependent on the node value. This is required, since the value of a particular service on node A may be different from that of node B. For example, the SSH service may be considered more important on a production webserver than it is on an internal mailing list server. The actual way in which the node value and service value are used to determine threats will be explained in a later section of this paper.

3.1.3 Roles

If we have many nodes and services on a network, especially in a large organization, it may be very difficult to assign values to nodes and services throughout the network. Suppose a large international organization has 50 production webservers and 25 development webservers. Consider the scenario when the decision is made to increase the value of the HTTP service on every production webserver, and decrease the value of the HTTP service on every development webserver. If this happens, the administrator would have to locate every single production webserver and development webserver and change the HTTP service values. The complexity increases greatly when there are more servers and similar decisions.

To reduce the complexity of node management and to facilitate assignment of values to nodes and services, we introduce the concept of the role. This is somewhat similar to the meaning of a role in Role-Based Access Control (RBAC) [11]. However, the important distinction is that we are using the role to keep track of node values and service values instead of permissions.

A role consists of four parts: a name, a node value, services offered by the node, and service values. This concept

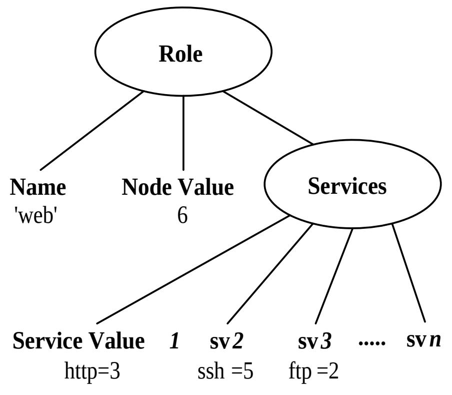

Figure 2: Example of a 'web' role.

is illustrated in Figure 2. Figure 2 shows a role named *web* which has a node value of 6. Part of the services offered by the *web* role are HTTP, SSH, and FTP. These services are assigned the service values 3, 5, and 2 respectively. This means that the SSH service is considered most valuable, followed by the HTTP service and FTP service.

Using a role such as *web*, we can facilitate assignment of node values and service values to nodes. For example, if we have 5 nodes and we assign them the *web* role, each node will have the node values and service values defined by the *web* role immediately. This is much more efficient than assigning the node values and service values to each individual node manually.

3.2 Threat Levels

To determine the risk associated with each source, the risk must be available as a quantifiable measure. The measure we use for the source is called the *threat level*. A threat level represents how suspicious the source is. This measure can be likened to the real world metaphors of the credit rating and GPA that were discussed in the introduction.

We can specify the threat level of a source using the threatLevel() function. For example, we can specify the threat level for the source 192.168.1.1 using the notation:

$$\text{threatLevel}(192.168.1.1) = 2.45$$

The threatLevel() function returns the threat level of the source. In this case, the threat level of the source 192.168.1.1 is 2.45. The idea is that the more suspicious the source is, the higher its threat level will be. The threat level is meant to be dynamic – it will change dynamically based on risk that is perceived in the network.

3.3 Thresholds

Each node value and service value has an associated *threshold*. The threshold represents the tolerance of the node or service to suspicious events. For example, we can specify that the threshold of a node value of 6 is 4, using the nodeThreshold() function in this notation:

$$\text{nodeThreshold}(6) = 4$$

The higher a node value or service value is, the lower its threshold will be. This means that the more valuable a node or service is, the lower its tolerance to suspicious events. This is required because we do not want highly valuable nodes and services to receive suspicious or malicious traffic. Unlike the threat level which is dynamic, the threshold is static.

3.4 Actions

As discussed earlier, the threat level changes dynamically. However, there has to be a set of criteria that determines how the threat level changes. In our architecture, the threat level will change based on events. The operation that actually changes the threat level is known as an *action*.

An action has two purposes: the first is to adjust the threat level, and the second is to act as a countermeasure that is triggered as a result of an event. For a typical action, we can define the event that would trigger the action, and how much we should adjust the threat level if that event were to occur. For example, in natural language, we can require an action to "increase the threat level by 5% if we encounter a network packet with buffer overflow shellcode."

The new threat level is calculated based on the previous threat level. The threat level $t\ell$ at time $i+1$ is determined by the threat level at time i. This is specified as:

$$t\ell_{i+1} = t\ell_i \ op \ adj$$

op is the mathematical operator, and can be any element in the set $< +, -, \times, \div >$. *adj* is the amount that will be used to adjust the threat level. For example, the following statement would increase the threat level by 10%:

$$t\ell_{i+1} = t\ell_i \times 1.1$$

The following statement would decrease the threat level by 0.3:

$$t\ell_{i+1} = t\ell_i - 0.3$$

At present, we have not specifically described exactly how the actions would be performed. In the actual implementation, the actions would be performed by software agents residing in the AEF and nodes themselves. These agents need to cooperate in order to perform the action. This is especially true for actions that trigger countermeasures (for example, to gather information about DNS records in the DNS server and correlate it with the network packets seen by the AEF).

3.5 Notation

We will use the following notation to describe the elements in our architecture:

N	A set of nodes, $\{n_1, ..., n_i\}$.
S	A set of services, $\{s_1, ..., s_j\}$.
V	A set of node values, $\{v_1, ..., v_k\}$.
W	A set of service values, $\{w_1, ..., w_l\}$.
R	A set of roles, $\{r_1, ..., r_m\}$.
A	A set of actions, $\{a_1, ...a_n\}$.

The notation for the functions we use to describe the architecture are as follows::

role(i)	Returns the role of node i.
services(i)	Returns the set of services for node i.
roleServices(m)	Returns the services of role m.

nodeServices(i)	Returns the services of node i. Equivalent to roleServices(role(i)).
nodeValue(i)	Returns the node value of node i.
serviceValue(i,j)	Returns the service value of service j on node i.
nodeThreshold(i)	Returns the threshold of node i.
serviceThreshold(i,j)	Returns the threshold of service j on node i.
nvThreshold(v)	Returns the threshold of node value v.
svThreshold(w)	Returns the threshold of service value w.
nodes()	Returns all nodes $n_1 ... n_i$.
nodeActions(i)	Returns all actions of node i.
threatLevel(i)	Returns the threat level of source i.

4. POLICY SPECIFICATION

This section describes how the policy is specified in the architecture. The policy is divided into two sub-policies: a static policy and a dynamic policy. As their names imply, the static policy does not change, while the dynamic policy changes. This section describes both types of policies, and what is contained in each. A diagram showing how these policies are related is shown in Figure 3.

Before we proceed with the discussion of static and dynamic policies, it is important to understand the rationale behind why the policy is divided into two sub-policies. The architecture needs to address two objectives: the first is to formalize a method to define nodes, services, their respective values, and roles; the second is to keep track of the threat levels of sources and ensure that they do not exceed the preset thresholds. It can be observed that the first objective and second objective have one main difference: the requirements for the first objective do not change frequently, while the requirements for the second changes frequently in real-time. Based on this observation, we divide the policy into two cleanly separated sub-policies. The static policy handles the first objective and is specified manually by the administrator, where it may be changed from time to time. The second objective, on the other hand, would be addressed by the dynamic policy. Unlike the static policy, the dynamic policy is initialized and specified by the machine. It should be noted that the dynamic policy does use certain input from the static policy as part of the AEF's decision-making process.

We would also like to stress that the meanings of static and dynamic in this context are defined from the viewpoint of the AEF. Even though the role memberships may be dynamic, we assume that they are static when we apply it for our purpose of network access control.

4.1 Static Policy

The static policy is specified by the administrator. The static policy is like a regular policy, which does not change until a modification to the policy is required. If this happens, then the policy is modified manually by the administrator.

The static policy consists of six parts: constraints, roles, node-role assignment, the threshold table, services, and actions. We will now describe each part in turn.

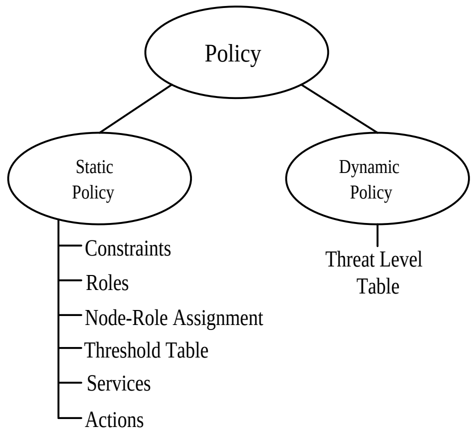

Figure 3: Static and dynamic policies.

With the exception of constraints, the static policy is defined using statements that begin with a declarative keyword. The statement accepts several arguments after the keyword. The general schema is:

```
keyword(
    arg1 = val1,
    arg2 = val2,
);
```

This statement uses the keyword `keyword` to define a rule, and assigns values `val1` and `val2` to the variables `arg1` and `arg2` respectively. The full policy specification schema is given in Figure 4.

4.1.1 Constraints

It is important to have flexible constraints in order to support emerging applications [5]. In our architecture, constraints are used to restrict the way in which the policy is specified. Constraints are needed to ensure that the semantics of the policy are correct. For example, we can use constraints to enforce the minimum node value to be 1 and the maximum node value to be 10. The following list shows six predefined constraints that are used for our architecture:

$C1$	$\max(V) = \max(W)$
$C2$	$\min(V) = \min(W)$
$C3$	$\lvert V \rvert = \lvert W \rvert$
$C4$	serviceValue(i, j) $\leq$ nodeValue(i), for any service j on node i
$C5$	svThreshold(w) $\leq$ nvThreshold(v), if $v = w$
$C6$	$\min(\text{threatLevel}(s)) \geq 0.1$

Constraint $C1$ states that the maximum node value must be defined to be the same as the maximum service value. Conversely, constraint $C2$ ensures that the minimum node value must be equal to the minimum service value. Constraint $C3$ ensures that the number of elements in the set of node values V have to be equal to the service values W. Constraints $C1$, $C2$, and $C3$ are motivated by the same reason: to ensure consistency and uniformity in the measures used for both nodes and services. If the measures are not uniform, the analysis of risks based on these measures may become complex and result in an inaccurate assessment.

Constraint $C4$ ensures that all service values for a given node are always less than the node value of that node. This

General schema:

```
keyword(
    arg1 = val1,
    arg2 = val2,
);
```

where

```
  keyword   =  role|assign|action|services|actions|meta
 argument   =  name|nodevalue|addr|role|dpe|cost|pattern|pass_op|pass_tla|traffic
    value   =  1|...|n
```

Role statement:

```
role(
   name              =  string,           // role name
   nodevalue         =  integer,          // node value
   service_descrip   =  embed services(), // services structure
);
```

Services statement:

```
services(
   name   =  value, // service name (one or  many)
),
```

Node-role assignment statement:

```
assign(
   addr              =  string,          // node address
   role              =  string,          // node role name
   dpe               =  string,          // dynamic policy enforcement flag
   actions_descrip   =  embed actions(), // actions structure
);
```

Actions statement:

```
actions(
   name = string,  // action name
   embed name()   // list of action names
),
```

Node threshold declaration statement:

```
nodevalue(
   integer   =>  integer,  // node value -> node threshold
);
```

Service threshold declaration statement:

```
servicevalue(
   integer   =>  integer,  // service value -> service threshold
);
```

Action declaration statement:

```
action(
   name      =  string,           // action name
   cost      =  string,           // cost of performing action
   pattern   =  integer range,    // patterns to look for
   pass_op   =  +|-|*|/           // operator to use in threat level adjustment
   pass_tla  =  real,             // amount to adjust
   traffic   =  int|ext|both,     // traffic type
);
```

Meta declaration statement:

```
meta(
   minimum_tl        =  real,     // minimum threat level
   decrease_trigger  =  integer,  // number of sources before default decrease
);
```

Figure 4: Policy specification schema.

is to ensure that services are able to be accessed as long as the nodes are accessible.

Constraint $C5$ states that the service threshold for each service value v must be less than the node threshold for each node value w, if $v = w$. For example, if service threshold for a service value of 10 must be less than or equal to the node threshold for a node value of 10. This is done for the same reason as that for constraint $C4$.

Constraint $C6$ ensures that the minimum threat level for all sources are always greater than or equal to 0.1. 0.1 is chosen as the minimum threat level instead of 0 for reasons of simplicity; this reduces the need to address problems such as division by zero when adjusting the threat level.

Apart from these constraints, additional constraints can be specified if required. This flexibility would be useful if we need to fine-tune the policy specification to support specific organizational objectives. For example, a military environment has different needs compared to a commercial environment. In such scenarios, the ability to add or modify constraints would be beneficial. The same flexibility would also be useful when exploring new forms of policies and algorithms based on this architecture.

4.1.2 Roles

The first part of the static policy specifies the roles that can be assigned to the nodes. As discussed earlier, roles facilitate management and reduce the complexity of assigning node values and service values to many nodes. In the static policy, the role is given a name, a node value, and a list of service and service value pairs. The role is declared using the `role` keyword. For example, the following statement declares the *web* role, giving it a node value of 4. The services ssh, ftp, and telnet are given service values 4, 2, and 1 respectively. The node value and service values must conform to Constraint $C4$.

```
role(
    name = "web",
    nodevalue = 4,
    services(
        ssh = 4,
        ftp = 2,
        telnet = 1,
    );
);
```

4.1.3 Node-Role Assignment

The node-role assignment specifies how the roles are assigned to the nodes. This assignment is declared using the `assign` keyword. The arguments to the declaration are the source address, the role name, the dpe flag (which stands for dynamic policy enforcement), and the actions that the node is “interested” in.

The following example shows a node with the address 192.168.0.1 being addressed to the *web* role. It states that dynamic policy enforcement should be carried out on this node, and the node is interested in the actions `ping`, `oscheck`, and `dnscheck`.

```
assign(
    addr = "192.168.0.1",
    role = "web",
    dpe = true,
    actions(
        ping, oscheck, dnscheck
    );
);
```

4.1.4 Threshold Table

The threshold table defines the thresholds for node values and service values. The thresholds for the node values and service values are related, and are restricted by constraints $C1$, $C2$, $C3$, and $C5$ from Section 4.1.1. The following example shows the threshold table for node values. In this table, the node value 10 has a threshold of 2, while the node value 9 has a threshold of 4, and so on.

```
nodevalue(
    10 => 2,
    9 => 4,
    8 => 6,
    7 => 8,
    6 => 10,
    5 => 12,
    4 => 14,
    3 => 16,
    2 => 18,
    1 => 20,
);
```

The service threshold table for service values use the same syntax as the node threshold table. The difference is that the `servicevalue` keyword is used instead.

```
servicevalue(
    10 => 2,
    9 => 3,
    8 => 4,
    7 => 5,
    6 => 6,
    5 => 7,
    4 => 8,
    3 => 9,
    2 => 10,
    1 => 11,
);
```

4.1.5 Services

The services policy is specified using a file with the same format as the `/etc/services` file found in UNIX systems. We are using this since `/etc/services` is a standard and widely-known format on UNIX systems. This file defines services and port number assignments. We use the standard on services and port assignments set by the Internet Assigned Numbers Authority (IANA) [4].

4.1.6 Actions

Each action is defined using six arguments: name, cost, pattern, pass_op, pass_tla, and traffic. The *name* acts as an identifier for the pattern. *Cost* states how much the action would cost to execute (in terms of resources such as time, disk storage, network resources, etc.). For example, it would be more costly to execute an action like correlating traffic between nodes, than it would be to simply ping the source of a connection. At present, however, cost is not used in our simulation, but it is specified here as an argument that can be used in the future. The *pattern* argument specifies the pattern to look for in the source. Once a pattern match is encountered, the action would be triggered and executed. An example of a pattern may be an IDS signature or a certain value in a particular field in an IP packet. It is intentionally left generic at this stage.

After being triggered, the action would have to adjust the threat level. The *pass_op* and *pass_tla* fields define the operator and threat level adjustment value that would be used to adjust the threat level. The threat level will only be adjusted if a pattern match is found. In order not to overcomplicate matters, we do not provide an equivalent operator and threat adjustment level if there is no pattern match (if there are such arguments, they would be called *fail_op* and *fail_tla* respectively).

The last argument is *traffic*, which can be specified as either *int* for internal, *ext* for external, or *both* for both internal and external traffic. This argument instructs the AEF about the type of traffic to look for – internal, external, or both.

The following statement defines an action named "ping" using the `action` keyword:

```
action(
    name = "ping",
    cost = 1,
    pattern = 35-38,
    pass_op = +,
    pass_tla = 0.1,
    traffic = both,
);
```

This action incurs a cost of 1 unit, and looks for patterns in the range of 35 to 38 (in our simulation, we are just using integers to represent traffic patterns. In an actual implementation, we would use a more formal language to represent these patterns). If the pattern matches are found, the action would increase the threat level of the source by 0.1 ($t\ell = t\ell + 0.1$). This pattern looks for sources in both the internal and external network, since the traffic argument is set to *both*.

Apart from the usual actions, the action policy specification has to begin with a meta section which would be declared using the `meta` keyword. The meta section specifies two arguments. *minimum_tl* is the minimum threat level to assign to a new source. This value has to conform to constraint $C6$. The second argument is the *decrease_trigger* field, which specifies how many legitimate sources should be allowed before we trigger the special default_decrease action. The default_decrease action is used to decrease the threat level. The *decrease_trigger* field and the default_decrease action are both used to reward sources which continuously exhibit good behavior. The following is an example of how the meta section can be used to specify a minimum threat level of 0.1 and a *decrease_trigger* of 10 sources.

```
meta(
    # Minimum threat level
    minimum_tl = 0.1,

    # Decrease only after 10 sources of
    # good behavior (no actions triggered)
    decrease_trigger = 10,
);
```

The special default_decrease action is specified in the same way as regular actions, except that its name has to be "default_decrease". The following example specifies that the default_decrease action should reduce the threat level by 1%.

```
# Decreases threat level as a reward for good behavior.
action(
    name = "default_decrease",
    cost = 0,
    pattern = 0,
    pass_op = *,
```

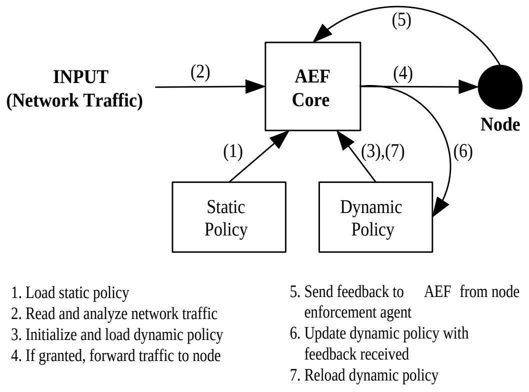
INPUT (Network Traffic)
(2)
AEF Core
(4)
(5)
Node
(1)
(3),(7)
(6)
Static Policy
Dynamic Policy
1. Load static policy
2. Read and analyze network traffic
3. Initialize and load dynamic policy
4. If granted, forward traffic to node
5. Send feedback to AEF from node enforcement agent
6. Update dynamic policy with feedback received
7. Reload dynamic policy

Figure 5: Inner workings of the AEF.

```
    pass_tla = 0.99,
    traffic = both,
);
```

4.2 Dynamic Policy

Unlike the static policy which is specified by the administrator, the dynamic policy is utilized by the computer. As its name implies, the dynamic policy automatically changes based on certain conditions.

Compared to the static policy, the dynamic policy is relatively simple. The dynamic policy consists of a threat level table, which is a two-column table that keeps track of sources and their current threat levels. In the implementation of our experimental prototype for the simulation, this table was implemented as a hash table that takes in the traffic mode (internal or external) of the source and the source address, and maps to the threat level of the source:

$$H\colon\ (\textit{traffic\_mode},\textit{src\_addr}) \rightarrow \text{threatLevel}(\textit{src})$$

Upon the trigger of an action on a source, the threat level of the source would be adjusted. The new value of the threat level would then be reflected in the hash table.

5. POLICY ENFORCEMENT

As described earlier in Section 3, we have introduced a new entity called the Authorization Enforcement Facility, or AEF. This section describes how the policy is enforced using the AEF. We show how the policy specified in Section 4 is loaded by the AEF, and how the AEF makes decisions about whether to allow or deny access from a source into the network.

A high-level flow of the inner workings of the AEF is shown in Figure 5. First, the AEF loads the static policy. Then, as traffic comes in, the AEF would read and analyze each source (recall that source is a generic term for a packet, connection, or stream). As the first source arrives, the AEF would initialize and load the dynamic policy. The AEF then determines the risk associated with the source. Access is determined at this point. If the risk is low, access is granted to the destination node and service.

Before we commence our discussion on policy enforcement, it is important to first understand the relationship between policy specification and policy enforcement. This relationship is illustrated by the way in which the static policy is used by the dynamic policy. When the AEF loads the static policy, it would also load the sub-policies under the static policy, which are the constraints, roles, node-role assignments, thresholds, services, and actions. Based on this information, nodes are assigned roles, and both nodes and services are given their specified thresholds. The node-role mapping and threshold table are kept in the AEF's internal memory. The threat level table in the dynamic policy is also initialized by the AEF, which would assign initial threat levels (the minimum) to all sources. Policy enforcement is generally performed like this: when new sources arrive at the network, the AEF would examine them to see whether they match any patterns defined by the actions in the policy specification. If they do, then the action would increase the threat level. When the threat level increases beyond the thresholds allowed by the static policy, access would be denied. In all other cases, access is allowed.

Now that the basic idea of how policy specification assists enforcement has been addressed, we will discuss, in greater detail, how the AEF actually determines whether access is allowed or denied. We then describe how we measure suspicion in order to trigger actions.

5.1 Determining Access

Recall the discussion of real world metaphors in the introduction, where we described the credit rating and GPA as measures to determine the risk associated with individuals. Determining access for a source using our architecture follows the same concept. Like the credit rating and GPA, threat levels and thresholds are the measures used to determine risk in this architecture.

We stated earlier that each node value and service value has an associated threshold, which states how tolerant the node or service is to suspicious events. Also, each source has a threat level which shows how suspicious the source is. These two measures are meant to be used together to determine risk, which in turn is used by the AEF to allow or deny access to the source.

If a source continuously exhibits suspicious behavior, the AEF would increase its threat level via the actions. As long as the threat level stays below the node threshold or service threshold, access would be granted to the source. If the threat level of the source increases to such an extent that it meets or exceeds the threshold of the destination node or service, the AEF would deny the source from accessing the destination.

In summary, access to a node i and service j from source k is allowed if and only if the following two equations are fulfilled:

$$\text{threatLevel}(k) \leq \text{nodeThreshold}(i)$$
$$\text{threatLevel}(k) \leq \text{serviceThreshold}(i, j)$$

To get a better idea about how this works, consider the scenario where a malicious source is trying to access two nodes: a node H that is more valuable, and another node L that is less valuable. Since H is more valuable, both its node threshold and service threshold would be lower than those of L. Initially, the threat level of the source would be low, and therefore access to both H and L would be granted (since the threat level has not exceeded the thresholds yet). As the threat level grows, it would exceed H's thresholds first, since the thresholds are lower than that of L's. When this happens, access to H and its services would be denied. Later, when the threat level increases beyond the thresholds of L, access to L and its services would be denied too.

So far, we have only looked at the scenario where the threat level increases. In the real world, traffic from a source may be legitimate most of the time, and malicious at other times. Therefore, the threat level has to decrease from time to time in order to make it realistic. We have already described the *default_decrease* action in Section 4.1.6. Using the *default_decrease* action, we can decrease the threat level after a certain amount of legitimate connections, as specified in the static policy.

However, an attacker may attempt to exploit this mechanism by introducing a lot of legitimate traffic to hide her attacks. By doing this stealthily, the attacker can make the AEF reduce the threat level of its source, while launching attacks stealthily in the background. Since the source's threat level is low, access to most nodes would still be granted. To counter this threat, we use a conservative-decrease aggressive-increase policy. This means that we would decrease the threat level by a small conservative amount when legitimate connections are encountered, but at the same time, we would increase it by a large amount when malicious traffic is present. Using this method, the AEF can frustrate the attacker who is trying to attack the network stealthily.

By using risk to determine access, we can dynamically allow or deny access to many sources. This is what we mean by allowing the "in-betweens" which we mentioned in the introduction.

5.2 Measuring Suspicion

So far, we have discussed how access is granted or denied based on the threat levels and thresholds. We will now discuss the criteria that are used by actions to increase the threat level of a source. In other words, how do we measure suspicion?

In the introduction, we used the example of a salesperson, where we said that the salesperson's ability to make a sale depends on "signs" such as appearance, communication skills, and dressing. In our architecture, these signs are represented by the patterns which the actions look for. In Section 4.1.6, the concept of patterns was left generic. In order to explain how suspicion is measured, we will now discuss examples of more specific patterns that are looked for by actions to adjust the threat level.

Since we are primarily working with the network domain, we will give examples of patterns that can be found in the network domain as follows:

- Malformed packets – Illegal TCP flag combinations, invalid field values, corrupted packets.
- Malicious/abnormal packet content – Buffer overflow shellcode in the packet payload.
- Anomalous behavior – Unusual behavior from sources that deviate from the normal profile. For example, suppose a sales agent who normally uses the system from 9am to 5pm logs in at 3am one night. This exhibits anomalous behavior.

Apart from these patterns, we can also use active countermeasures to gauge whether a source is behaving suspiciously. For example, we can use programs to enable a node to ping the source of the connection whenever the node receives traffic. If the source of the connection does not respond, there

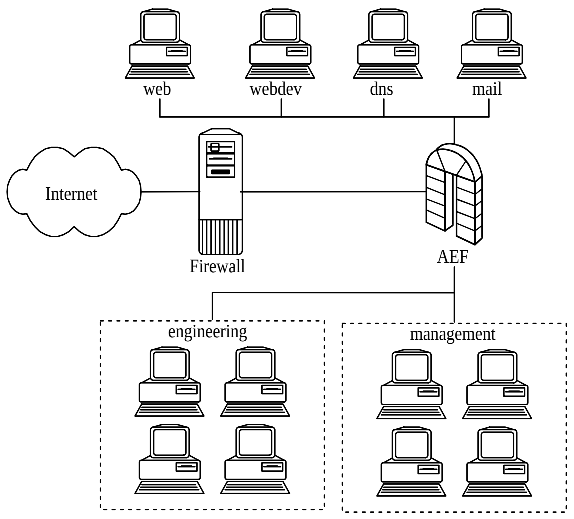

Figure 6: The simulated network.

is a slight chance that it is spoofed traffic. Of course, there may be other reasons too, such as a firewall blocking the source from receiving ping packets, or the source itself disabling ping packets. When we execute a countermeasure like this, we have to take these issues into consideration. If the source of a connection is an internal node, and is known to block ping packets, then the action may be configured to disregard the problem that it is not responding. On the other hand, if it is an external source, we cannot be entirely sure – therefore, a ping action can be configured to increase the threat level for this source by just a small value.

6. SIMULATION

A simulation was used to investigate the feasibility of the architecture. We decided to perform a simulation before developing an actual implementation of the architecture, since a simulation would be able to provide us with advance information on potential design and implementation issues.

We focused on simulating the network of a small organization, shown in Figure 6. In the diagram, the role of each node is displayed under the node. The network consists of a server farm consisting of four servers: a production web server (web), a development web server (webdev), a DNS server (dns), and a mail server (mail). We also created two two fictitious departments in the organization called engineering and management. There are four workstations in each department. Note that *engineering* and *management* are also the names of the roles for the workstations.

In our discussion in Section 3, we mentioned that all traffic has to pass through the AEF in order to be authorized for access to the destination nodes (see also Figure 1). In accordance with this requirement, we designed the simulated network such that all traffic from any node has to pass through the AEF in order to reach its destination node.

6.1 Dataset

The simulation assumes 10,000 source connections. We used both internal and external hosts (or sources) in our simulation. There were 8 internal hosts, where 4 are workstations assigned the *engineering* role, and the other 4 workstations are assigned the *management* role. We used 8 external hosts with non-internal IP addresses.

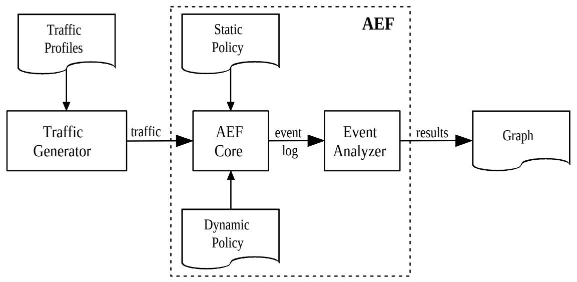

Figure 7: The flow of events in the simulation.

In the simulation, we are trying to assess if the sources would be granted or denied access by the AEF based on their risk. Therefore, to make it realistic, we designed the simulation such that the servers would only receive traffic and not send out traffic. In other words, the servers act only as destination nodes and not as source nodes.

We defined 6 actions which look for varying patterns and have differing threat adjustment values. The minimum threat level (and also the initial threat level) for each source was specified to be 0.1. The *default_decrease* action was configured to be triggered after 10 legitimate connections from the source. Upon its activation, the threat level for the source would be decreased by 1%.

6.2 Experimental Prototype

An experimental prototype was developed to perform the simulation. The prototype was built using Perl 5.6 on the Linux 2.4 platform. There are three components in the prototype: the traffic generator, the Authorization Enforcement Facility (AEF), and the event analysis component.

The traffic generator reads in information about the source hosts (both internal and external) and traffic profiles. It then generates traffic based on these input data. The generated traffic is later passed to the AEF. The AEF loads the static policy and parses the traffic sent to it by the traffic generator. As parsing is done, the dynamic policy is updated continually to adjust the threat level based on the actions defined in the static policy. All events triggered by the AEF are logged into an event log. The event log is then processed by the event analyzer, which generates graphs based on the logs. This simulation flow is shown in Figure 7.

6.3 Results

We will now explain the simulation results. We used three traffic generation profiles in the simulation: *normal*, *suspicious*, and *highly malicious*. The graphs representing the results for each of these profiles are shown in Figures 8, 9, and 10 respectively. There are many elements that are represented in these graphs. The graph shows the threat level of a source and thresholds of all destination nodes and services. In order to make things clear, we arrange the thresholds of destination nodes and services by groups, instead of showing every threshold for every node and service. For example, suppose a source connects to 5 destination nodes. The first two nodes share the same threshold of 6, the third has a threshold of 7, and the last two nodes share a threshold of 8. Instead of showing thresholds for all 5 destination nodes,

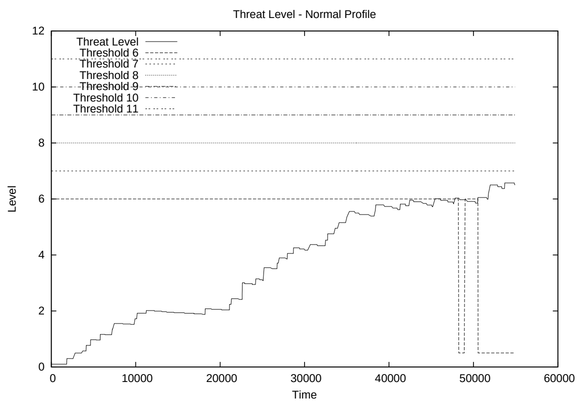

Figure 8: Normal profile.

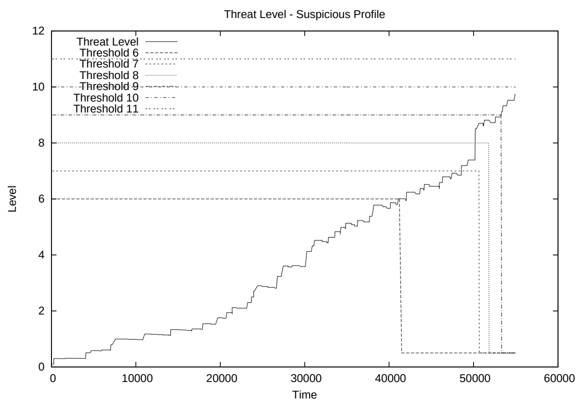

Figure 9: Suspicious profile.

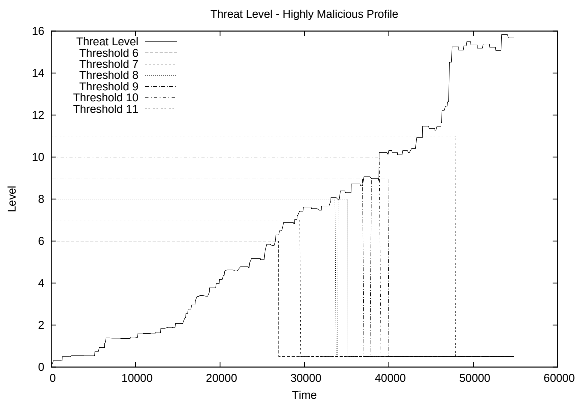

Figure 10: Highly malicious profile.

we just show thresholds 6, 7, and 8 to represent them.

When access is denied to a source for nodes and services with a certain threshold, we represent it in the graph by showing that the node or service threshold as having dropped to a level of 0.5. If access is granted again, the threshold is increased to its original level.

It should be noted once again that a node or service with a high value would have a low threshold. In our simulations, the thresholds range from 6 to 11. This implies that nodes and services represented by threshold 6 have the highest value, while nodes and services at threshold 11 have the lowest value. In each graph, the solid line represents the threat level, while dotted lines show the various thresholds.

We will first examine the results for the normal traffic generation profile (Figure 8). The solid line is the threat level of the source. As it increases, access is still granted to all nodes and services of all thresholds, until it reaches a level of 6 and above. At this point, the AEF denies access for the source to nodes and services with a threshold of 6. This is shown by the drop of the Threshold 6 line to 0.5. As a result of the *default_decrease* action, the threat level temporarily decreases to a value below 6. This allows destinations with Threshold 6 to be accessed again – hence, we see the Threshold 6 line resuming its original level of 6. The threat level increases again, and Threshold 6 drops yet once more. All this while, the threat level never increases to 7 and beyond. Therefore, destinations with thresholds 7 to 11 are always accessible by the source. Since the destinations with Threshold 6 are of the highest value, it is expected that access to these destinations would be denied even when the threat level is still relatively low.

The next graph in Figure 9 shows the suspicious profile. Like before, as the threat level increases, the access to destinations with thresholds are denied. This time, destinations with thresholds 6 to 9 are gradually denied access as the threat level of the source increases. Destinations with thresholds 10 to 12 (the destinations with the lowest values) are still accessible.

Lastly, the graph in Figure 10 represents the highly malicious profile. In this graph, the threat level increases very aggressively, due to the source introducing a lot of malicious traffic. This behavior triggers many actions, which in turn increases the threat level. As a result of this, all destinations are denied access gradually as their thresholds are exceeded.

There is one issue that needs to be pointed out with regard to the threat level and the threshold lines in the graphs. In a few cases, such as the Threshold 11 line in Figure 10, it looks as though access is still granted to the destinations even when the threat level has exceeded a level of 11. The reason why this happens is because at this point, the source has not connected to the destinations with a threshold of 11 yet – it was connecting to other destinations. However, once the source attempts to connect to a destination of threshold 11, it was denied immediately. This is shown by the drop in the Threshold 11 line at around time unit 48,000.

The results can be summarized as follows. As noted earlier, the more valuable a node or service is, the lower its threshold (the lower its tolerance to suspicious events). As the threat level increases, the access to the more valuable nodes and services will be denied to the source first. This is followed by the denial of access to less valuable nodes and services. This demonstrates that our dynamic access control scheme can determine whether access to nodes and services should be allowed or denied by taking risk as the input.

7. DISCUSSION

So far, we have demonstrated the feasibility of the architecture using a simulation. In order for the architecture to be deployed into a real network environment, however, a few issues still need to be resolved. These issues are discussed in this section.

For further testing and evaluation of the architecture, we need to develop or use a traffic generator that performs realistic traffic generation. The tool needs to be highly customizable to allow fine-grained control over how traffic is generated. This would allow us to test various types of traffic patterns and how actions respond to them.

In this paper, we have described policy enforcement as a process, where we discussed how the AEF determines access and measures suspicion. In the future, we would also describe policy enforcement from a more general, architectural view. The approach we are investigating would divide policy enforcement into three components: policy parser, policy decision, and policy management. The policy parser component would preprocess the policy specification and extract parameters for policy decision and management. Our overall policy specification schema may need to be improved and extended to accommodate future requirements.

One potential concern with the AEF architecture is that it may be a bottleneck. While this may seem to be the case at first glance, it can be noted that the AEF is similar to a firewall that is positioned between the Internet and the organizational network. Since modern day firewalls carry out fairly complex and highly advanced tasks, we believe the AEF should also be able to perform the same level of functionality without becoming a bottleneck. Furthermore, there are commercial products like ForeScout ActiveScout [2] that are installed between the firewall and the Internet, but are not seen as bottlenecks. Having said that, it is possible that the AEF may encounter performance issues during future development. Should this happen, two approaches that would be worth exploring are the development of efficient algorithms for the AEF, as well as the implementation of the AEF in hardware for high-speed performance.

We need to be wary about attacks against the architecture itself. As a finite state machine, the AEF may be susceptible to certain issues that affect finite state machines. At this point, we have identified two broad types of attacks that may potentially be used to attack the architecture. The first type of attack is where an attacker attempts to launch a denial-of-service attack against the AEF by introducing a lot of malicious traffic from spoofed source addresses. This would increase the threat levels of the real sources and eventually these sources would not be able to access the network. We believe that this problem can be solved, especially when the AEF is used in conjunction with other network security technologies. Another countermeasure is to improve the algorithms used by the AEF to include more advanced methods to detect and reduce this threat. For example, the AEF can be designed to immediately drop network packets that are known to be spoofed. This is similar to how firewalls with stateful packet inspection handle spoofed packets. In the AEF, we could drop such packets without increasing the threat level of that source, thus mitigating this threat. We believe that a practical solution lies somewhere in between these two approaches, and we are looking carefully into the issue.

The second type of attack is where an attacker introduces legitimate traffic as noise to hide stealthy malicious attacks. We have suggested one mechanism to counter this threat of stealthy attacks, which is the conservative-decrease aggressive-increase policy discussed in Section 5. However, more sophisticated attacks from this category may require more advanced defense scheme. As such attacks also affect other finite state machines apart from the AEF, the study of countermeasures against them may present itself as an interesting research problem for future study. At present, we are in the process of conducting further simulations of the AEF in order to develop better algorithms to counter this type of attack.

Communication between the AEF and the nodes is an issue too. In order to carry out actions, an application-level communication protocol needs to be designed and used between the AEF and the nodes. There is the issue of clock skew, which may affect certain countermeasures and actions. This can be addressed by using time synchronization protocols like the Network Time Protocol (NTP) [8]. Another issue is the security of messages that are exchanged using this protocol. To achieve confidentiality and preserve integrity in these AEF-node communications, we would need to use cryptographic techniques to secure the messages.

Lastly, there is the issue of role hierarchy. In our present architecture, no role hierarchy is assumed. However, an actual network in an organization may benefit from role hierarchy. For example, a department may require all its webservers to have certain services and service values, while a smaller unit within that department may have their own webserver which would need to inherit those services. Since we are using roles to represent values as opposed to permissions in regular RBAC, a few ways of allowing role hierarchy need to be considered.

8. ONGOING AND FUTURE WORK

Based on the issues from our discussion in Section 7, we are currently carrying out the deployment of the architecture into a network environment. Although we are extending the ideas described in this paper into an actual network environment, we also envision a few other areas where the architecture can be used. In this section, we discuss our current progress as well as a few ideas on how this architecture can be applied.

One point we mentioned earlier in the paper is that the AEF complements firewalls and IDSs. We are presently working on a few approaches that can be used to perform true integration of the architecture into firewalls and IDSs. An interesting property of this architecture is that it can be integrated with firewalls and IDSs using a few techniques. We see three key approaches that can potentially be used. First, the AEF functionality can be integrated into a firewall itself, thus transforming a packet-filtering firewall into a dynamic firewall that allows or denies packets based on intelligent decisions using risk. Another approach is to integrate it with an IDS, whereby the IDS would use the feedback provided by the AEF to update its signatures. The third approach is to introduce the AEF as a separate entity altogether. Each of these approaches have their advantages and disadvantages. Inclusion of the technology into firewalls and IDSs would require an organization's existing technology to be changed. Using the AEF as a separate component may avoid the need for this, but would require a deployment of an additional device in the network.

So far, we have determined risk using threat levels and thresholds, which are tightly related to the network layer. Using input from the environment may give us interesting ways of determining risk. For example, if we know that a certain environment, say a department, has certain machines with special properties, we can use this knowledge to develop an environment profile which can be used as an input to determine risk more intelligently. Likewise, we can define certain features in roles that can be used as input at a higher level of abstraction, using a role profile.

The architecture uses the threat level measure extensively. The threat level can result from different combinations of threat levels, increments (or decrements), and actions. There may be a concern about whether all combinations that result from one threat level actually represent the same amount of risk or not. More simulations need to be performed to obtain the optimal values and measures to represent the risks.

8.1 Specific Actions

Up to this point, the actions have been described in a rather generic manner. Actual deployment in a network environment would require more specific methods to define patterns, perform pattern matching, and execute countermeasures. We will now discuss a few specific examples of actions and countermeasure strategies that we are presently considering. Please note that these approaches may require software agents to be running on the destination nodes. These agents would communicate with the AEF to enable it to carry out the actions.

One action that may be worth pursuing would be to obtain the passive fingerprint [12] of the operating system of the source. Based on this OS fingerprint, we can examine future incoming packets from the same source to see whether they fit the criteria of packets for that operating system or not. If they do not match the criteria, something suspicious may be happening. OS fingerprints can also be used to identify network scans that originate from certain scanning programs which are known to be only available on certain platforms. Of course, there are issues with this, such as sources with customized operating system kernels that have been patched to announce false OS fingerprints, and sources with multiple operating systems installed. We cannot be sure whether the fingerprint is correct or not. At the same time, these fingerprints may be useful to detect attacks and hence we should not dismiss them totally. Information like the OS fingerprint is especially appropriate for the AEF architecture, which, as we mentioned earlier in the paper, allows "in-betweens."

The AEF can also potentially make more intelligent decisions if information from servers in an internal network is available. Let us say that we have access to the internal network's DNS server. Suppose Node A accessed Node B (a webserver) on the HTTP port for the *first time*. If DNS records do not show that Node A accessed the DNS server to obtain Node B's numeric IP address, there may be reason to believe that this connection is suspicious. The reason is, most of the time, a user would enter the host name of the webserver in the web browser in order to access its web pages. This requires the DNS server to be queried in order to resolve the host name to its numeric equivalent, especially if the webserver is queried for the first time. An Internet worm, on the other hand, would normally access numeric IP addresses directly without querying the DNS server.

Another action is to use high-level heuristic rules to determine if a network connection is normal or not. We can examine whether a typical connection is conforming to Internet standards and rules. For example, is the source port incrementing for every new connection that a source is initiating? Does the connection look like a port scan? Has the same TCP sequence number appeared recently?

We can also consider the bandwidth that is being consumed by the source and match it with the type of traffic that is expected by the source. For example, we do not expect high bandwidth utilization for normal web traffic compared to applications like video streaming. If normal traffic such as web traffic uses bandwidth utilization that is equivalent to that of multimedia streaming in a short period of time, something may be wrong.

Other factors that we are considering for use as actions include the use of IDS signatures, investigation of user behavior (especially on an internal network), logging incoming and outgoing data per node, and basic countermeasures to gain intelligence from the source. From the examples of actions that we have seen so far, one observation can be noticed: the AEF architecture makes it easy for new forms of actions to be used. If we require new types of countermeasures in the future, action modules can be developed for them and these modules can be "plugged" into the architecture. We are currently creating and designing more actions and plan to carry out further experiments and simulations using these actions soon.

8.2 Other Potential Applications

Although the architecture presented is specific to the network domain, it can be applied to other areas as well. One such area is load balancing. Suppose we have a pool of web servers, with different degrees of load at different times. If we have an AEF-like component in the network, a new HTTP connection can be directed to the appropriate web server based on certain criteria, such as where the source connection is from and the amount of load in the current web servers.

Yet another area where this architecture can be applied is in the area of reputation-based systems. In an example of a reputation-based system, a node would assign a reputation to neighboring nodes, and decide whether to send data based on the neighboring node's current reputation. Our notions of threat levels and thresholds can potentially be used to support such systems.

Since the earliest work in information security, it has been recognized that the greatest threat is from insiders. In particular, defending against an insider who attempts to abuse his computer privileges is significant to the security of the nation's critical infrastructures. Despite the fact that potentially catastrophic consequences of insider threat continues to be confirmed by security breaches in the real world, numerous issues on how to combat these threats are yet to be fully addressed by the research community [1, 17]. We believe that the AEF can help with the design of a practical and effective insider threat mitigation system.

To see how it might work, recall that access to the servers on an internal network can provide intelligence for the AEF to make better decisions. This would provide the AEF with the capability to function as an insider threat mitigation system. As an example, if we have an internal network where all communications are analyzed by the AEF, we would be able to detect the abuse of certain resources by employees. To be more specific, consider the case where the employee might try to access an unauthorized webserver containing sensitive information. The AEF could be configured to automatically block future similar attempts to nullify or minimize the impact of the abuse.

9. CONCLUSION

We have described a new approach to secure networks in the form of a dynamic access control architecture. This architecture uses risk as a way to determine threats in a network environment. We view it as a complementary technology to firewalls and IDSs. Our simulation results show that it is feasible, and we are currently working on developing and deploying the architecture in a real network environment.

Acknowledgments

We would like to thank the anonymous reviewers of this paper for their helpful comments and suggestions.

10. REFERENCES

[1] M. D. Abrams, J. Heaney, O. King, L. J. LaPadula, M. Lazear, and I. M. Olson. Generalized framework for access control: Towards prototyping the ORGCON policy. In *Proceedings of the 14th National Computer Security Conference*, Washington, D.C., October 1991.

[2] ForeScout. ActiveScout. World Wide Web, 2002. http://www.forescout.com/activescout.html.

[3] L. Harn and H. Lin. Integration of user authentication and access control. In *IEE Proceedings-E*, volume 139, number 2, pages 139–143, 1992.

[4] Internet Assigned Numbers Authority. Port numbers. World Wide Web. http://www.iana.org/assignments/port-numbers.

[5] T. Jaeger. On the increasing importance of constraints. In *Proceedings of the 4th ACM Workshop on Role-Based Access Control*, pages 33–42, Fairfax, VA, October 1999.

[6] K. Knorr. Dynamic access control through petri net workflows. In *Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC)*, New Orleans, LA, December 2000.

[7] C. H. Lin, R. C. T. Lee, and C. C. Chang. A dynamic access control mechanism in information protection systems. *Journal of Information Science and Engineering*, 6(1):25–35, March 1990.

[8] D. L. Mills. Network Time Protocol (version 3) specification, implementation and analysis. RFC 1305, March 1992.

[9] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. The spread of the Sapphire/Slammer worm. Technical report, January 2003. http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html.

[10] P. Naldurg and R. H. Campbell. Dynamic access control policies in Seraphim. Technical Report UIUCDCS-R-2002-2260, Computer Science Department, University of Illinois at Urbana-Champaign, February 2002.

[11] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[12] L. Spitzner. Know your enemy: Passive fingerprinting. World Wide Web, March 2002. http://project.honeynet.org/papers/finger/.

[13] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium (Security '02)*, San Francisco, CA, August 2002.

[14] R. K. Thomas and R. S. Sandhu. Towards a task-based paradigm for flexible and adaptable access control in distributed applications. In *Proceedings on the 1992-1993 Workshop on New Security Paradigms*, pages 138–142, Little Compton, RI, 1993.

[15] R. K. Thomas and R. S. Sandhu. Conceptual foundations for a model of task-based authorizations. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 66–79, Franconia, NH, June 1994.

[16] R. K. Thomas and R. S. Sandhu. Task-based authorization controls (TBAC): A family of models for active and enterprise-oriented authorization management. In *Proceedings of the IFIP WG11.3 Workshop on Database Security*, Lake Tahoe, CA, August 1997.

[17] D. Verton. Insider threat to security may be harder to detect, experts say. *Computerworld*, April 12, 2002.

[18] S.-M. Yen and C.-S. Laih. On the design of dynamic access control scheme with user authentication. *International Journal of Computers and Mathematics with Applications*, 25(7):27–32, 1993.