Integrated Access Control and Intrusion Detection (IACID) Framework for Secure Grid Computing

Tatyana Ryutov, Clifford Neuman, and Li Zhou
Information Science Institue
University of Southern California
Marina del Ray, CA 90292
{tryutov, bcn, zhou}@isi.edu

Abstract

Traditional Intrusion Detection Systems (IDSs) work in isolation from access control for the application the systems aim to protect. The lack of coordination and inter-operation between these components prevents detecting sophisticated attacks and responding to ongoing attacks in real time, before they cause damage. Another disadvantage is a large number of false positives. Reports of attacks can trigger response actions (e.g., termination of the offending connections). Thus an inaccurate IDS decision may result in disruption of service to legitimate users. Therefore, successful intrusion detection requires accurate and efficient models for analyzing application, system and network audit data and real time response to the attacks.

To address the need for accurate and effective intrusion detection and response for secure Grid computing we developed an Integrated Access Control and Intrusion Detection (IACID) framework. The heterogeneity of Grid resources calls for policy-based detection and response to attacks to accommodate various security levels of resources and different security policies for specific users and environments.

Our approach is based on specifying security policies extended with the capability to identify intrusions and suspicious behavior at the application level and guiding the audit data collection, intrusion detection and response. Policy enforcement is performed by the GAA-API (a component of the IACID system) that monitors access requests to vulnerable Grid applications evaluates and enforces the policies. The system detects and responds to intrusions by comparing access request patterns against the security policies and communicating with different detectors in order to build a coherent picture of attacks.

Key words: Intrusion Detection, Access Control, Security Policies, Computing Grids, and Grid applications

1 Introduction

The recent history of attacks against the information systems and widespread vulnerabilities indicate that security threats have dramatically escalated in speed, impact and frequency. Grid infrastructures like Globus [11]; Legion [12] and Condor [13] are particularly vulnerable to intrusions and require an adequate level of security for users, data and resources. Grids are open environments; compromise of a single resource may provide unauthorized access to data and services from other systems. Grids are vulnerable to large-scale attacks that may cause disruption of the Grid services. Thus, it is essential for Grids to support prevention, detection and automatic response to intrusion attempts. To increase the survivability of Grids, we need to develop automated defenses capable of rapid and intelligent responses to the attacks.

Most traditional intrusion detection systems (IDSs) [2] take either a network- or a host-based approach for recognizing and responding to attacks. These systems look either for attack signatures, specific patterns that indicate malicious or suspicious intent or deviation from a normal profile (anomaly) that indicates an attack. A network-based IDS looks for these patterns and anomalies in network traffic. A host-based IDS looks for attack signatures and anomalies in system audit trails or application logs.

In most cases intruders exploit vulnerabilities and misconfigurations in application servers to break into a system. The application-level attack can enter a system through the same open "door" in the perimeter defenses used by legitimate users. Therefore, these attacks are difficult to detect.

Current network- and host-based IDSs work in isolation from access control for the application the systems aim to protect. The lack of coordination and inter-operation between these components prevents detecting sophisticated attacks and responding to ongoing attacks in real time, before they cause damage. Another disadvantage of currently available IDSs is a large number of false positives (IDS reports an attack when none has occurred). Reports of attacks can trigger response actions (e.g., termination of the offending connections). Thus an inaccurate IDS decision (a false alarm) may result in disruption of service to legitimate users. Therefore, successful intrusion detection requires accurate and efficient models for analyzing a large amount of application, system and network audit data and real time response to the attacks.

To address this, we apply dynamic policy techniques to support cooperative, accurate and cost effective intrusion/misuse detection and response capabilities. For the purpose of this project we concentrate on the audit data that contains distinctive evidence of legitimate and intrusive user behavior. At a high level, intrusion detection is an audit data classification problem. An IDS extracts pieces of evidence from audit data, uses some model or a set of rules to reason about this evidence, and classifies each audit record into normal, particular kind of intrusion, or anomaly. The most challenging problem is extraction and selection of system features to be examined by the IDS.

Our approach differs from other work done in the area by concentrating on dynamic policy techniques that guide data collection and analysis. The policies recognize when applications operate outside their normal range and provide inputs to aid intrusion detection and response mechanisms. This will support detecting complex and subtle attacks and fine-tuning detection services. In particular, it allows one to selectively audit events that are most likely to identify suspicious behavior; find more attacks, and reduce the number of false positives. A major contribution of the proposed work is the development of techniques that consider an organizational and application-level security policies in order to customize intrusion detection capabilities to meet the security needs of a particular application.

As a part of the GridSec (*Grid Security*) project at USC/ISI, we developed the IACID framework. IACID detects and responds to intrusions by comparing access requests against security policies and communicating with multiple detectors to build a coherent picture of attacks.

Our approach is based on a generic access control mechanism – GAA-API [6], [7], [8], and [9] that can be used by a number of different applications with no modifications to the code of the mechanism. In contrast, traditional application-based IDSs (e.g., AppShield [10] and Emerald [1]) are hard to manage and deploy, as one is required for each type of critical application. An access control mechanism is ideally situated to apply application level knowledge about policies and activities to identify suspicious behavior and apply appropriate responses.

2 Background Work

In the completed DEFCN (Dynamic Policy Evaluation for Containing Network Attacks) project at USC's Information Science Institute, funded by DARPA, we developed an access control framework [6] that is sensitive to network threat conditions. The project has developed the GAA-API [7], a middleware API for generic authorization and access-control. The implementation is based on expanding the policy evaluation mechanism with the ability to generate real time actions, such as checking a current system threat level and sending a notification.

An *authorization policy* regulates access to objects. An *object* is a target of requests, e.g., files and hosts. An *access right (or operation)* is a particular type of access to a protected object, e.g., read or writes. A *condition* describes a context in which each access right is granted. The conditions provide support for monitoring and updating internal system structures and their runtime behaviors.
Pre-conditions specify what must be true in order to grant the requested operation.
Request-result conditions must be activated whether the authorization request is granted or whether the request is denied.
Mid-conditions specify what must be true during the execution of the requested operation.
Post-conditions specify what must be true on the completion of the operation execution. The post-conditions are used to activate post execution actions, such as logging and notification whether the operation succeeds/fails.
The GAA-API provides a general-purpose execution environment in which policies (that specify access rights with optional set of associated conditions) are evaluated. The GAA-API evaluates the policies using the current system state.
The enforcement of the policies is partitioned into three successive phases. During each phase only the specified set of all conditions in the policy is evaluated. *S*a indicates whether the request is authorized (T), not authorized (F) or uncertain (U). Sm. indicates the evaluation status of the mid-conditions (T/F/U). Sp indicates the evaluation status of the post-conditions (T/F/U). The policy enforcement process is shown in Figure 1.

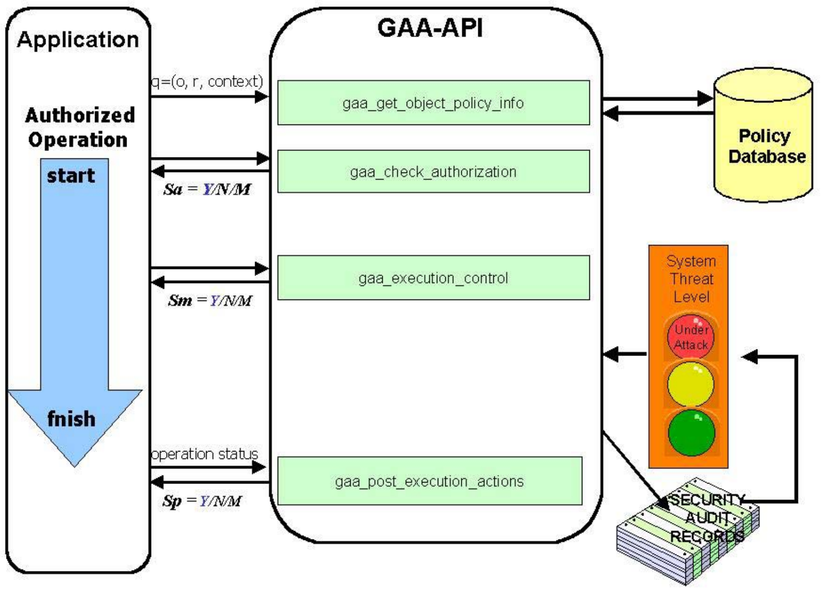


Figure 1. The Three-phase Policy Enforcement.

The *access control phase* starts with receiving a request q = (o, r,context) to access object o, requested operation r and contextual information (e.g., user identity, current time). First, the gaa_get_object_policy_info function is called to obtain the security policy associated with the object. If no relevant policy was found, the authorization status is set to F and the request is rejected.

Next the **gaa_check_authorization** function is called to evaluate pre- and request-result- conditions. If there are no pre-conditions, the authorization status is set to T. Otherwise, the pre-conditions are evaluated and the result is stored in the authorization status Sa. If the request-result conditions are present in the policy, the conditions are evaluated and the intermediate result is stored in variable X. The conjunction of the X and Sa is stored in the authorization status Sa. If authorization is not granted ($Sa \neq T$), the request is rejected.

The *execution control phase* consists of starting the operation execution process and calling the **gaa_execution_control** function. If mid-conditions are found, the conditions are evaluated. Some mid-conditions are evaluated just once, other mid-conditions are evaluated in a loop until either the operation finishes or any of the mid-conditions fails. In the latter case, the operation execution is suspended and the reactive actions are started. The mid-conditions can be returned unevaluated to be enforced by application. The result is stored in Sm.

During the *post-execution action phase*, the **gaa_post_execution_actions** function is called. The operation execution status (indicating whether the operation succeeded/failed) is passed to the **gaa_post_execution_actions**. If no post-conditions are found, the Sp is set to T, otherwise the post-conditions are evaluated and the result is stored in Sp.

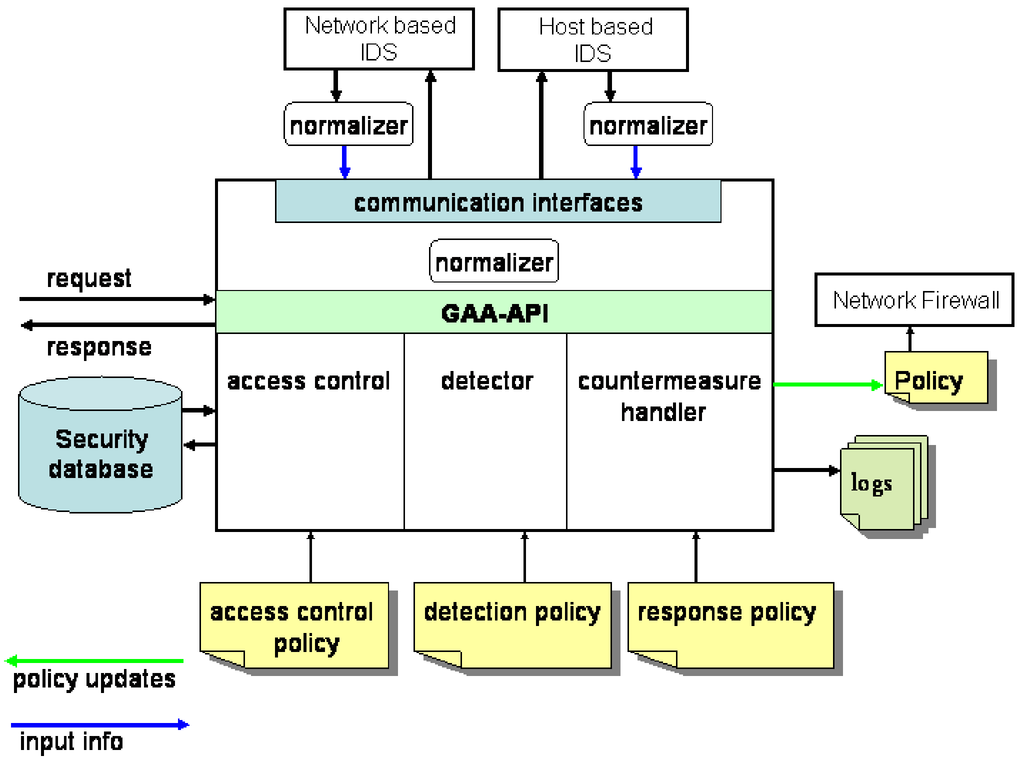
The purpose of dynamic adaptation to threat conditions is to enable the use of policies that tighten or loosen defenses in response to network attacks or the perceived likelihood of an attack at a particular point of time. The DEFCN project supports dynamic adaptation to network threat conditions communicated by an IDS.

The efforts in the DEFCN project were not to detect intrusions, but to accept information from IDSs. This information is used in evaluation of conditions to activate and deactivate particular policy entries that are used to mediate access to application-level objects. In this effort, we will specifically feed data to the distributed intrusion detection and response systems specified in Section 3 and to accept alerts from this system and use those alerts in determining authorized actions.

If response to an attack scenario must be fine-grained, i.e. short of shutting down all connections into a system completely, yet still comprehensive enough to protect against attacks that originate from multiple points on the network, then higher-level software must adapt behavior in response to a perceived threat. This support must be at the application level in servers because often only the servers themselves have knowledge of the operations that are requested, and the objects to be manipulated by the request

3 The IACID framework

We assume network-based IDSs are placed at routers and firewalls; that host-based IDSs may be located anywhere on end hosts; that application level IDSs are typically placed with the application server. The entire architecture is depicted in Figure 2.



The IADIC architecture consists of the following primary components:
Security database provides information to the rest of the system. This information is collected from various sources and can be of many types, including:

1. User activity profiles describe "normal" user behavior and are used in anomaly detection to detect suspicious activities that deviate from the profiles.

2. Misuse signatures and intrusion scenarios describe known techniques used by attackers to penetrate the target system.

3. Application audit records monitor and log user activity.

4. Attack reports generated by other IDSs. As application-level IDS monitors events at the user level of abstraction, the IDS usually can not detect Trojan Horses or other software tampering attacks. Therefore, cooperation and sharing of information among application-, host- and network-based IDSs is important. It allows correlation of intrusion reports to aid in situation awareness and may help to detect and contain intrusions that cross multiple network and domain boundaries.

5. Customized response recommendations provide a description of the response to the attack based on the type of incident, domain-specific characteristics and location of an event within the network. The recommendations describe the actions required for incident response to contain and control the attack. The actions may include generating alarms, adapting the behavior of the application, and adjusting the activity profiles to incorporate new values in order to keep the profiles up to date.

Intrusion detector determines the presence of an attack based on event streams and input from the security database. The engine can consist of an anomaly detector or/and a misuse detector. These components can be fairly generic and used for a number of applications. However, the database of known intrusion scenarios and attack patterns should be customized for different applications. Next section provides more detailed description of this component.

Countermeasure handler If the detector determines the result to be suspicious, the handler will take the corrective actions to prevent malicious actions from being executed.

A security policy includes three parts:
1. An ***access control policy*** specifies authorizations checked by the access control module.

2. A ***detection policy*** specifies the information to be analyzed by the detector. This information may include database of attacks and activity profiles, parameters of an access request and information obtained from monitoring the execution of the requested operation and a status (success/failure) of the completed operation.

3. A ***response policy*** defines actions to be performed by the countermeasure handler for incident response. The IADIC system will support denying requested access; affecting execution of the requested operation (e.g., suspending or killing a process); generating alarms and audit records; updating firewall rules and so on.

Access control module recognizes and mediates access requests forwarded to the module for approval.

Communication interfaces If the IADIC system and IDS are to cooperate, there must be some mechanism for the two systems to communicate with one another. Since the services will have to explicitly specify the information to be passed and returned they will need bi-directional interfaces. The researchers will design a policy controlled interface for establishing a subscription-based communication channels to allow IADIC system and IDSs to communicate, either by having information directed from one service to another (pushed) or requested by one service from another (pulled).

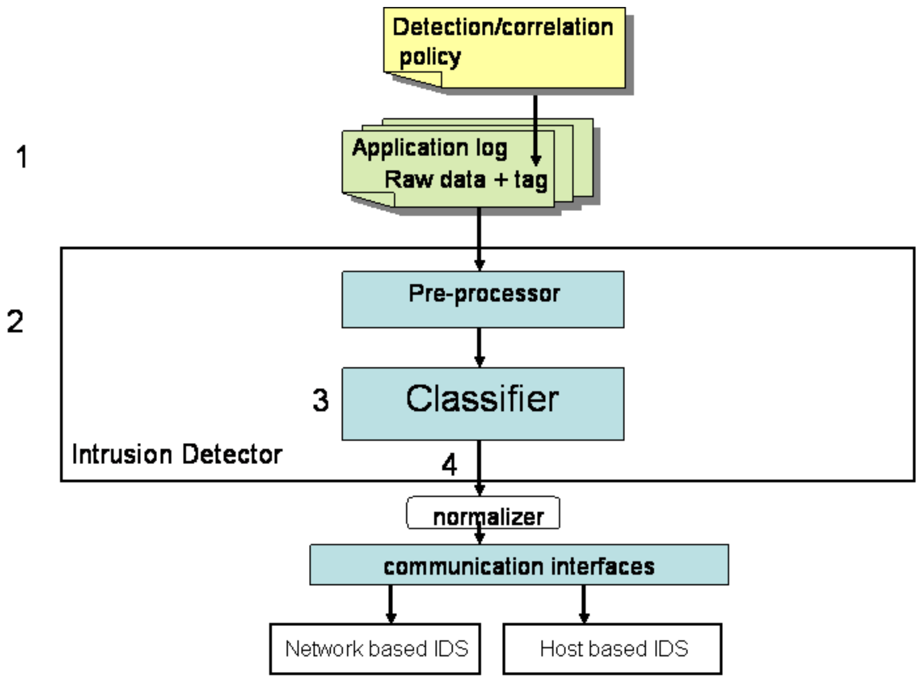
Data Normalizer processes the input and output data and syntactically converts it to a format understood by the IADIC system (input data) or particular intrusion detection system (output data). In the case of detectors with open source code, this is done by modifying the code accordingly; otherwise, we will use wrappers to translate the native output of the detectors into a common format.

3.1 IACID Intrusion Detector

The researchers developed a generic IACID architecture, in which application-level intrusion detection and response are integrated components. This architecture is based on the existing GAA-API implementation framework developed during the DEFCN project, which provides a basic access control framework extended with the ability to adapt to a state generated by intrusion detection engines.

The IACID system supports both real time and delayed attack detection. The real time detection catches malicious requests based on attack signatures (e.g., SQL manipulation attacks). More complex and multistage attacks that require extensive processing can be done periodically.

The IACID supports the following tasks illustrated in Figure 3.



1. *Adaptive collection and initial encoding of audit data at run time.* A well-known problem with audit is that too much data is collected to be efficiently and accurately analyzed for intrusions. To alleviate this problem, the IACID supports adaptive auditing and initial encoding of data. The audit data is used to derive profiles that describe typical behavior of users working with different applications. For example, the correlation between *command* and *argument* in the grid application command history data of a user, can serve as the basis for constructing normal usage profiles. The data collection is conditioned on different characteristics, including:
    a. *time.* In an office system, data collection can include office hours during weekdays and few first and last days of a month (to cover monthly operations). Whereas nighttime and weekends are better suited for looking for suspicious activity.
    b. *system load.* When system is busy, the data collection can be suppressed, whereas light system load can allow reporting data to build user profiles.

   c. *access identity.* The behavior of particular users can be of interest: new users, for whom no profile exists, untrusted users, such as guests, and so on.
   d. *type of requested right.* The usage of certain operations can be of particular interest, e.g., system shutdown.

   Each audit record is assigned a specific tag. This tag is used to convert the audit data to an input to a classifier. The amount, detail and characteristics of the data are controlled by the policies and depend on system state. The tag will be either attached to an audit condition in policy (EACL entry) and is logged with the data or will appear in a condition that sends requests to other classifiers.

2. *Adaptive pre-processing of the audit data.* The collected audit data is unstructured and is not suitable for classification. The raw audit data has to be transformed into the format required by the data analysis techniques. A component of the IACID performs data preprocessing and select a small number of important features from the available data. The system produces different outputs according to the tag stored with each audit record thus supporting adaptive selection of the most influential system features. The data pre-processing is done off line periodically.

3. *Adaptive data analysis.* Based on the information observed by the access control component and encoded in the policy, the IACID intrusion detector provides advise/predictions for (classifiers, e.g., which classification algorithm to use (may depend on the data), association rule set changes).

4. *Cooperative intrusion detection.* The IACID system helps to support data mining over the entire knowledge and data sources of the system. IACID provides hints for correlation of reports produced by different IDS at several levels: packet, host and application to obtain combined evidence of intrusion. Attacks may be related by attacker, type of attack, objectives, sites, or timing.

   For example, IACID detects a buffer overflow attacks by checking that the length of input to a CGI script exceeds the threshold defined in the policy. This may indicate a buffer overflow attack. Before applying serious countermeasures (e.g., blacklisting the IP address) the system has to check for IP address spoofing. IACID can ask classifier that processes packet logs (e.g., tcpdump logs) to find packets where *source IP = destination IP = my local domain*. Another way is for IACID to send a request to a classifier that examines host-level logs on the apparent source machine to find an entry for initiating the remote access. If the IP spoofing attack has succeeded, one may get a log entry on the victim machine showing a remote access. However, on the source machine there will be no corresponding entry for initiating that remote access. This information will be encoded in the policy and can be represented as association rules or a set of SQL queries.

3.2 Cooperative Intrusion Detection

The high false alarm rate is one of the most serious problems of the current ID technology. More sophisticated analysis that integrates diverse sources of available data is needed. For complex attacks, information from one sensor is unlikely to detect suspicious activity. Thus, the next task is to explore how access control and intrusion detection components can cooperate to improve the overall effectiveness of system protection mechanisms. The data extracted from an application at the access control time can be supplemented with data from network- and host-based IDSs to detect attacks not visible at the application level.

IACID to IDSs Interactions describe information that the IACID system reports to various IDSs. This information is used locally by modules that implement the application-level intrusion detector, and may be forwarded to other IDSs for analysis.
The IACID system reports:

1.Ill-formed access requests
Because the IACID system processes access requests by applications, the system can apply application-level knowledge to determine whether the request is properly formed. Ill-formed access requests may signal an attack. For example, consider an application that issues queries to a database. It is assumed that the application makes bug-free database queries. Errors in access requests may indicate that someone has compromised the application and is performing ad hoc queries against the database.

2.Accesses requests with abnormal parameters
The system can report requests with parameters that: (i) violate site policy, for example root modification of the file /etc/hosts.equiv that adds a '+' to the file. (ii) abnormally large. For instance, a Sendmail buffer overflow attack sends a message that contains a MIME header line that is inappropriately large.

3.Denied access
The system can report even a single instance of access denial to sensitive system objects. If, for instance, a process originating from a web browser or a mail program attempts to write to system files, that is most likely a malicious behavior. The system can report attempts to access non-existent hosts on a network, which could indicate network scanning or mapping activity and attempts to use critical commands.

4.Exceeding threshold conditions
Statistical techniques are often approximated by thresholds, particularly when it is not practical to develop full profiles or when speed is an issue. Threshold detection is based on determining which events indicate intrusion. Examples of events that can be controlled by the threshold detectors and reported by the IACID system include: the number of failed login attempts within a given period of time, the cutoff levels for network traffic, disk or CPU usage, and the number of successful system shut downs within a given period of time.

5.Incidents
The IACID system can report detected application-level attacks. The report may include application specific threat characteristics, such as: description of attack, attack severity rating, attack type, attack confidence value, attack correlation propositions, and defensive recommendations.

6.Suspicious application behavior
The system can report unusual application behavior such as creating files.

IDSs to IACID Interactions
The IACID system can request a network-based IDS to report, for example, indications of address spoofing. This information can be used in addition to the application-level attack signatures to further reduce the false positive rate and avoid DoS attacks. This will be particularly important for applying pro-active countermeasures, such as updating firewall rules and dropping connections.

The IACID system can request information for adjusting policies, such as values for thresholds. When implementing a threshold detector, the obvious difficulty is choosing the threshold number and a time interval of the analysis for a particular event. The threshold values may depend on many factors and can be determined by a host-based IDS and communicated to the IACID system.

3.3 Security Policy

One of the weaknesses of current IDSs is the absence of a security policy. These systems rely on built-in algorithms that usually are not flexible enough to adapt to changing system security profile. In anomaly-based systems, any behavior that is inconsistent with a profile is judged to be suspicious. Similarly, misuse detection systems always report misuse when a match to an attack pattern is discovered. By developing techniques to drive the detection of intrusion/misuse from a policy specialized for the site and application being monitored, fine-tuning of intrusion/misuse capabilities becomes possible.

A major part of the proposed work is the development of techniques that consider an organizational and application-level security policies in order to customize intrusion detection capabilities to meet the security needs of a particular application.

Determining a comprehensive security policy, identifying events that are violations of the policy, then evaluating this information in the context of what is important to an operation can be difficult. We are exploring ways to formulate a security policy governing the actions of intruders. A policy can be defined in terms of acceptable and unacceptable access patterns to protected resources. For example:

- *A Closed World policy* states that everything that is not explicitly authorized is unacceptable and may indicate suspicious behavior.
- *An Open World policy* defines that everything that is explicitly denied is unacceptable and may indicate suspicious behavior.
- *A Mixed World policy* may recognize some explicitly authorized access patterns as suspicious. The policy may authorize such access only on a closely monitored basis.

If the organizational requirements are relatively straightforward, i.e., they dictate the need for SSH connections and web access, (a firewall blocks all other traffic), and it is likely that any successful intrusions will be SSH- or HTTP-based. In this case it is useful to audit all successful connections. An effective policy should define the subsets of activity that are authorized at the application layer. For example, it might be possible to define a minimum set of SSH commands that are allowed and then define the presence of all other commands as a violation of policy. For the web server one may be able to define a set of strings contained in the HTTP request that indicate suspicious behavior.

Detecting intrusions requires either knowledge of possible intrusions or knowledge of the known and expected behavior of a system. To support both anomaly- and misuse-based analysis, the researchers will propose a set of conditions to characterize the activity profiles and identify patterns corresponding to known attacks. Some possible profile and threshold characteristics include:

- History of invoked operations that describe user activities on a system.
- The way the resources are accessed, e.g., location, time intervals.
- The resource usage characteristics, such as CPU, memory, disc space.
- The amount of data being accessed and the rate of accessing the data.

Other characteristics will be added in the course of the research process. These characteristics can be represented as conditions that describe user or system normal behavior. Failure of any of these conditions (or a combination of conditions) may signal suspicious behavior. For example, access is requested at unexpected times or unusual locations. Accessing abnormally large number of records in a database may signal a suspicious activity such as attempted data inference. Opening abnormally large number of simultaneous connections to an application server may indicate a DoS attack because it starves the number of available sockets, disallowing new connections.

4 Implementation Status

Under the GridSec project we are extending the GAA-API software tools to support fine-grain access control and data mining for threat tracking and policy update in using Grid resources.

Some grid applications are provided on the web through grid portals, specific web sites etc.
Web interface can become an interface for users to access grid services providing an environment that hides the complexity of heterogeneous, distributed back end. For example, users can submit and collect results for their jobs on remote resources through a web interface.

Unfortunately web servers are attractive targets for attackers seeking to steal or destroy data, deny user access, or embarrass organizations by changing Web site contents. The Web servers are an easy target for outside intruders because the servers must be publicly available around the clock. In order to penetrate their targets, attackers may exploit well-known service vulnerabilities. A Web server can be subverted through vulnerable CGI scripts, which may be exploited by metacharacters or buffer overflow attacks.

Motivated by the importance of web servers for Grids, multitude of Web server vulnerabilities, and generally unsatisfactory server protection, we have integrated the GAA-API with the Apache Web server. The API evaluates HTTP requests and determines whether the requests are allowed and if they represent a threat according to a policy. The policy enforcement takes three phases:

1. Before the requested operation (e. g., display an HTML file or run a CGI program) starts- to decide whether this operation is authorized.

2. During the execution of the authorized operation- to detect malicious behavior in real-time (e. g., a process consumes excessive system resources).

3. After the operation is completed- to activate post execution actions, such as logging and notification whether the operation succeeds or fails (e. g., alerting that a particular critical file was written can trigger a process to check the contents of the file).

By being integrated with the Web server and having the ability to control the three processing steps of the requested operation, the GAA-API can respond to suspected intrusion in real-time before it causes damage, whether it is site defacement, data theft, or a DoS attack.

We are planning to integrate the GAA-API into **GridSite** [14] – a grid aware web application. GridSite can operate as a file server, as a web host with dynamic content, and as a grid host with Grid Services. GridSite provides a library for manipulating grid credentials (conventional X.509 certificates and Globus GSI proxies), and XML Grid Access Control Lists (GACL) [15]. We are interested in exploring how the GAA-API can interpret policies expressed in GACL and the benefits of such integration.

4.1 Adaptive collecting and encoding of audit data

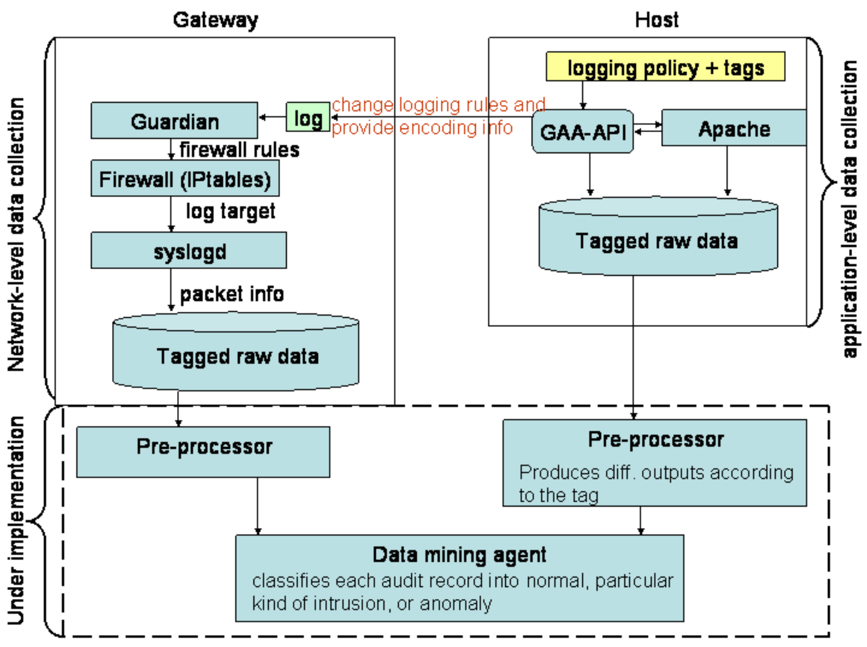
The access control decisions provide useful information to intrusion and misuse detectors, making dynamic, fine-grained responses feasible. In particular, policies may control the level of detail of the audit records generated. Thereby the audit overhead is reduced until intrusion detection engine notices that something is amiss, though not necessarily what it is. In addition, application-level feedback supported by the policies can help find more attacks and reduce the number of false-positives.

The main idea is to utilize the application level information available to the IACID system to guide the auditing and intrusion detection and response. The expert knowledge about access request characteristics (such as target objects, requested access rights and parameters), access control decisions and application vulnerabilities will help to extract the most relevant features and chose the efficient detection rules.

One of the major functions that an intrusion detection system should have is to interact with the firewall at the gateway. For instance, when a suspicious host is detected at the application-level, IACID may need to log all packets from that IP address at the gateway for the use of data mining agents. Moreover, after identifying the host as untrustworthy, IACID may also need to block the host at the gateway.

We have extended the GAA-API with a mechanism that dynamically updates the gateway firewall (iptables) rules as the system threat level changes. We next describe our implementation of the adaptive auditing and initial encoding of audit data.

Since the GAA-API only exists transiently as an application asks for an access control decision, it is insufficient to keep the firewall status continuously and make an overall control for all the GAA-API integrated applications within the system. Therefore, we use Guardian [16], a daemon script written in Perl, to serve as an agent between the GAA-API integrated applications and the firewall. The overall architecture of this design is shown in Figure 4.


Gateway
Host
Network-level data collection
application-level data collection
Under implementation
logging policy + tags
Guardian
log
change logging rules and provide encoding info
GAA-API
Apache
firewall rules
Firewall (IPtables)
log target
syslogd
packet info
Tagged raw data
Tagged raw data
Pre-processor
Pre-processor
Produces diff. outputs according to the tag
Data mining agent
classifies each audit record into normal, particular kind of intrusion, or anomaly

We use log target option with iptables command that turns on kernel logging of matching packets. When this option is set for a rule, the Linux kernel will log specified information on all matching packets. To illustrate how we use “log target” in our system, consider the following policy (expressed in EACL format and evaluated by the GAA-API):

```
1: neg_access_right appname *
2: pre_cond_check_suspicions .... ....
3: rr_cond_log_target guardian "on:failure/tag/source_IP/warning"
```

As the GAA-API finds suspicious host at application level (line 2), it will reject this request (line 1) and in further issues a log target rule for the gateway (line 3). This rule includes the tag (discussed in Section C.1) that will be logged with data, the IP address that the suspicion comes from, etc.

On the gateway machine, a Guardian daemon will periodically check the log file. If a new rule is found, it will first check whether the same rule has already been applied. Since it's very common for an application to identify the same suspicion more than once, or, for several applications to identify the same suspicion independently, the Guardian must remove the redundancy. If he rule is not redundant, the Guardian issues the following command to the firewall (iptables):

```
iptable -A INPUT -p tcp -j LOG --source-port="211.100.7.101" --log-prefix= "tag" --log-
level=warning
```

This command will enforce the firewall to log the information of all TCP packets coming from IP address 211.100.7.101. Here is an example of packet-level information that is logged:

```
Feb 2 12:29:17 localhost kernel: tag IN=eth0 OUT=
MAC=08:00:46:72:fe:0d:00:04:80:29:84:00:08:00 SRC=211.100.7.101 DST=128.9.168.64
LEN=60 TOS=0x00 PREC=0x00 TTL=47 ID=0 DF PROTO=TCP SPT=80
```

We are currently exploring the next two steps described in Section 3.1. In particular, we are exploring the types of information that should be encoded in the tag, how this information can help the preprocessor to adaptively select the most influential system features, and how it can provide some fine tuning of the association rule and frequent episode algorithms. The association rules algorithm determines relationships between fields in the audit records, and the frequent episodes algorithm models sequential patterns of audit events [3], [4], and [5].

5 Conclusions

Conventional security tools, such as firewalls, intrusion detection systems, and access controls, do not address the issue of application vulnerabilities. These mechanisms work in isolation from each other and from application they aim to protect. In addition, the mechanisms have separate, non-coordinated policy management

To address the need for accurate and effective intrusion detection and response for secure Grid computing we developed an Integrated Access Control and Intrusion Detection (IACID) framework. Our approach is based on specifying security policies extended with the capability to identify (and possibly classify) intrusions and suspicious behavior at the application level and guiding the intrusion detection and response. Policy enforcement is performed by the IACID system that monitors access requests to vulnerable applications evaluates and enforces the policies.

An IACID is well placed to act as logger and intrusion detection system as it receives just security relevant information at the abstraction level of the application. The advantages of looking for the attacks at the application level include the ability to access decrypted information about a request. A request transported to the application through an encrypted channel is not visible to a network-based IDS.

Fast attacks on IDS (that seek to exploit application's vulnerability before the IDS can apply counter measures) will not succeed because the system processes access requests by applications and the application waits for the result.

The information generated by the IACID at access control time can supplement the information obtained by network- and host-based IDSs. Thus, the approach proposes bi-directional, policy-controlled interface between the IACID system and network- and host-based IDSs. This allows the IDSs to communicate with the components they protect. While instrumenting the system to feed information to IDSs has its costs, the researchers believe that such communication can provide a number of potential advantages:

- Support detection of attacks not visible at the application level, but not detectable without the application-level knowledge.

- Reduce the rate of false positives. By integrating diverse sources of available data on several levels (application, host and network) the system will provide a level of accuracy on prevention decisions.

- Correlate sequences of events within the context of an application behavior that will also reduce the potential for false positives.

- Increase the quality and reduce the volume of data to feed into the detection systems. The IACID system can perform preprocessing steps before submitting data. Several distinct security events that emanate from the same attack can be combined into a single alert being generated, therefore, reducing the number of events that must be handled by an IDS or intrusion analyst. For example, the system can combine attack alerts from different sources but directed at the same target to get a complete attack scenario. The preprocessing steps might be implemented at the application layer through enforcement of policies that can determine related events - a task that often requires application-level knowledge. Furthermore, IACID can guide the audit data gathering and selection of relevant system features presented in the audit records to compute classifiers that can recognize anomalies and known intrusions.

***Acknowledgements*:** We thank the financial support from NSF Grant No. ACI-0325409.

References

[1] M. Almgren and U. Lindqvist.
Application-Integrated Data Collection for Security. *Proceedings of the Fourth International Symposium on the Recent Advances in Intrusion Detection (RAID'2001),* number 2212 in LNCS, pages 22-36, 2001.

[2] R. Bace and P. Mell.
Intrusion Detection Systems. NIST Special Publication on Intrusion Detection Systems. *National Institute of Standards and Technology,* August, 2001.

[3] W. Lee, S. Stolfo,
Data Mining Approaches for Intrusion Detection, in 7thUsenix Security, 1998.

[4] William W. Cohen. 1995 *Fast effective rule induction*, In Proceedings of the Twelfth International Conference on Machine Learning, Lake Taho, California, Morgan Kauffman.

[5] D. Barbara, J. Couto, and S. Jajodia. ADAM: A testbed for exploring the use of data mining in intrusion detection. SIGMOD Record, 30(4):15-24, 2001.

[6] T. V. Ryutov, B. C. Neuman, D. Kim and L. Zhou.

Integrated Access Control and Intrusion Detection for Web Servers IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 9, September 2003.

[7] T. V. Ryutov, B. C. Neuman, and D. Kim

Dynamic Authorization and Intrusion Response in Distributed Systems

In Proceedings of the 3rd DARPA Information Survivability Conference and Exposition (DISCEX III), Washington, D.C. April 22-24, 2003.

[8] T. V. Ryutov and B. C. Neuman.
The Set and Function Approach to Modeling Authorization in Distributed Systems. *In Proceedings of the Workshop on Mathematical Methods and Models and Architecture for Computer Networks Security*, May 2001, St. Petersburg Russia.

[9] T. V. Ryutov and B. C. Neuman.
The Specification and Enforcement of Advanced Security Policies.
Proceedings of the Conference on Policies for Distributed Systems and Networks (POLICY 2002), June 5-7, 2002.

[10] Sanctum, Inc. http://www.sanctuminc.com/

[11] I. Foster and C. Kesselman.
Globus: A metacomputing infrastructure toolkit.
International Journal of Supercomputer Applications, 1997.

[12] M. Lewis and A. Grimshaw.
The Core Legion Object Model.
Fifth IEEE International Symposium on High Performance Distributed Computing, 1996.

[13] M. Litzkow, M. Livny, and M. Mutka.
Condor - a hunter of idle workstations.
In Proceedings of the 8th International Conference on Distributed Computing Systems, 1988.

[14] GridSite http://www.gridpp.ac.uk/gridsite/

[15] Grid Access Control Language http://www.hep.man.ac.uk/website/gacl.html

[16] Guardian Active Response for Snort http://www.chaotic.org/guardian/