A Data Mining and CIDF Based Approach for Detecting Novel and Distributed Intrusions

Wenke Lee[1], Rahul A. Nimbalkar[1], Kam K. Yee[1], Sunil B. Patil[1], Pragneshkumar H. Desai[1], Thuan T. Tran[1], and Salvatore J. Stolfo[2]

[1] Department of Computer Science, North Carolina State University, Raleigh, NC 27695, USA
wenke@csc.ncsu.edu
http://www.csc.ncsu.edu/faculty/lee
[2] Department of Computer Science, Columbia University, New York, NY 10027, USA
sal@cs.columbia.edu

Abstract. As the recent distributed Denial-of-Service (DDOS) attacks on several major Internet sites have shown us, no open computer network is immune from intrusions. Furthermore, intrusion detection systems (IDSs) need to be updated timely whenever a novel intrusion surfaces; and geographically distributed IDSs need to cooperate to detect distributed and coordinated intrusions. In this paper, we describe an experimental system, based on the Common Intrusion Detection Framework (CIDF), where multiple IDSs can exchange attack information to detect distributed intrusions. The system also includes an ID model builder, where a data mining engine can receive audit data of a novel attack from an IDS, compute a new detection model, and then distribute it to other IDSs. We describe our experiences in implementing such system and the preliminary results of deploying the system in an experimental network.

1 Introduction

As network-based computer systems play increasingly vital roles in modern society, they have become the targets of our enemies and criminals. The security of a computer system is compromised when an intrusion takes place. An intrusion can be defined as "any set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource" [3]. Intrusion prevention techniques, such as encryption, authentication (e.g., using passwords or biometrics), and defensive programming, have been used to protect computer systems as the first line of defense. However, intrusion prevention alone is not sufficient because as systems become ever more complex yet security is still often the after-thought, there are always exploitable weakness in the systems due to design and programming errors, or various "socially engineered" penetration techniques. For example, after it was first reported many years ago, exploitable "buffer overflow" security holes, which can lead to an unauthorized root shell, still exist in some recent system software. Furthermore, as illustrated by recent Distributed

Denial-of-Service (DDOS) attacks launched against several major Internet sites where security measures are in place, the protocols and systems that are designed to provide services (to the public) are inherently vulnerable to attacks such as DOS. Intrusion detection can be used as another wall to protect network systems because once an intrusion is detected, e.g., in the early stage of a DOS attack, response can be put into place to minimize damages, gather evidence for prosecution, and even launch counter attacks.

Intrusion detection techniques can be categorized into misuse detection and anomaly detection. Misuse detection systems, e.g., IDIOT [7] and STAT [4], use patterns of well-known attacks or weak spots of the system to match and identify known intrusions. For example, a signature rule for the "guessing password attack" can be "there are more than 4 failed login attempts within 2 minutes". The main advantage of misuse detection is that it can accurately and efficiently detect instances of known attacks. The main disadvantage is that it lacks the ability to detect the truly innovative (i.e., newly invented) attacks. Anomaly detection systems, e.g., IDES [12], flag observed activities that deviate significantly from the established normal usage profiles as anomalies, i.e., possible intrusions. For example, the normal profile of a user may contain the averaged frequencies of some system commands used in his or her login sessions. If for a session that is being monitored, the frequencies are significantly lower or higher, then an anomaly alarm will be raised. The main advantage of anomaly detection is that it does not require prior knowledge of intrusion and can thus detect new intrusions. The main disadvantage is that it may not be able to describe what the attack is and may have high false positive rate.

In 1998, the Defense Advanced Research Project Agency (DARPA) sponsored the first Intrusion Detection Evaluation [11] to survey the state-of-the-art of research in intrusion detection. The results indicated that the research systems were much more effective than the leading commercial systems. However, even the best research systems failed to detect a large number of new attacks, including those that can lead to unauthorized user or root access. It is very obvious that the enemies, knowing that intrusion prevention and detection systems are installed in our networks, will attempt to develop and launch new attacks. By definition, misuse detection techniques are ineffective against new intrusions. While it is critical that we develop effective anomaly detection algorithms to detect novel attacks, it is also very important that we develop a mechanism where once a novel attack is detected (necessarily as an anomaly at first) its behavior is analyzed and a specific detection model is built and widely distributed. That is, we need to turn a novel attack into a "known" one as quickly as possible so that appropriate detection and response mechanisms are in place in a timely manner.

The recent DDOS attacks pose a serious challenge to the current de facto practice where an IDS is only concerned with its local network environment, without communication with other IDSs in the Internet. As described by Dittrich [2], a DDOS attack is normally accomplished by first breaking into hundreds (and even thousands) of poorly secured machines around the Internet and installing packet generation "slave" programs on these compromised systems. Remote "master" programs (or the attacker) control these slave programs to send packets of various types to a target host on the network. Even though each slave can just send malicious packets in an amount small enough to be considered "acceptable", the resulting flood, due to a huge number of such slaves getting involved simultaneously, can effectively shut the target system out of normal operation for periods ranging up to several hours. Although an IDS on a target system may detect a DOS, it alone cannot determine that there are many compromised systems in the Internet and that they are used to launch the attack. In other words, the localized approach can only detect a small part of the large-scale distributed attack, and can thus suggest only limited (and often useless) response. On the other hand, if IDSs in the Internet have a communication framework where they can exchange attack information, then upon detecting a DOS, an IDS can broadcast the attack instance to other IDSs, which can in turn activate specific modules (if they are not already running) to look for and kill the slave programs (if there are any in their local environments) responsible for the DDOS attack.

Our research aims to develop techniques for detecting novel and distributed intrusions. In this paper, we describe an experimental system, based on the Common Intrusion Detection Framework (CIDF) [17], where geographically distributed IDSs can communicate with each other by following the protocols defined in CIDF. For example, the IDSs can exchange attack information that includes attack source, method, behavior, and response, etc. Moreover, upon detecting a novel attack, an IDS can send the relevant audit data to a "model builder", which in turn automatically analyzes the data and computes a new detection model specifically for the attack, and distributes the model to other IDSs for local customization/translation and installation.

The rest of the paper is organized as follows. We first briefly describe the data mining technologies that enable the model builder. We then give an overview of the specifications of CIDF. We next describe the design and implementation of our experimental system. We then describe the experiments of using our system to detect (new) DDOS attacks. We compare our research with related work, and conclude the paper with a discussion of future research directions.

2 MADAM ID: a Data Mining Approach for Building ID Models

Currently, building an IDS is a labor-intensive knowledge engineering task where "expert knowledge" is codified as the detection models, i.e., the misuse detection rules or the measures on system features for normal profiles. Given the complexities of today's network systems, expert knowledge is often incomplete and imprecise; as a result, IDSs have limited effectiveness (i.e., accuracy). Further, since the development process is purely manual, updates to IDSs, due to new attacks or changed network configurations, are also slow and expensive.

We have been researching and developing a more systematic and automated approach for building IDSs. We have developed a set of tools that can be applied to a variety of audit data sources to generate intrusion detection models. We call the collection of these tools MADAM ID (Mining Audit Data for Automated Models for Intrusion Detection) [8, 10]. The central theme of our approach is to apply data mining programs to the extensively gathered audit data to compute models that accurately capture the actual behavior (i.e., patterns) of intrusions and normal activities. This approach significantly reduces the need to manually analyze and encode intrusion patterns, as well as the guesswork in selecting statistical measures for normal usage profiles. The resultant models can be more effective because they are computed and validated using large amount of audit data. Results from the 1998 DARPA Intrusion Detection Evaluation [11] showed that the detection models produced by MADAM ID had one of the best performances (i.e., with the highest true positive rates while keeping the false alarm rates within the "tolerable" ranges) among the participating systems, most of which were knowledge engineered.

The main elements of MADAM ID include the programs for computing activity patterns from audit data, constructing features from the patterns, and learning classifiers for intrusion detection from audit records processed according to the feature definitions. The process of using MADAM ID is shown in Figure 1.

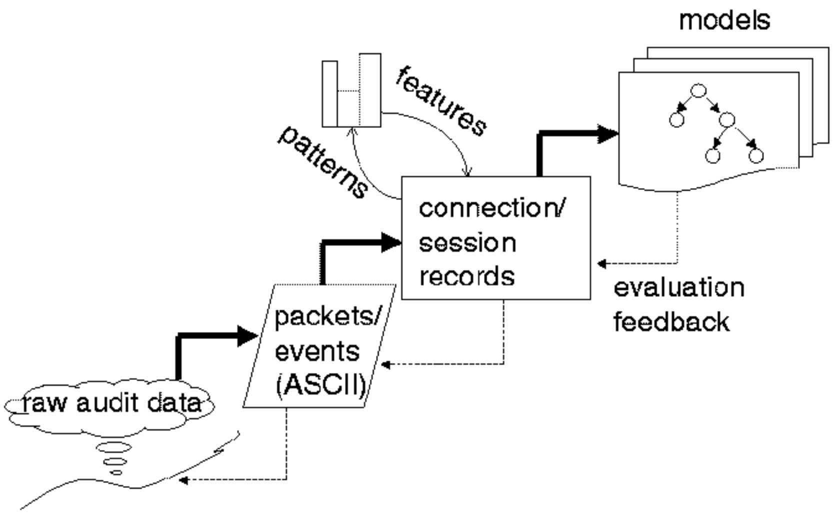

Fig. 1. The data mining process of building ID models.

The end product of MADAM ID is a set of classification rules that can be used as intrusion detection models. We consider intrusion detection as a classification problem because ideally we want to classify each audit record into one of a discrete set of possible categories, i.e., normal, a particular kind of intrusion, or anomaly. Given a set of records, where one of the features is the class label (i.e., the concept), classification algorithms can compute a model that uses the most discriminating feature values to describe each concept. An example rule output from RIPPER [1], a classification rule learner, is the following:

```
pod :- wrong_fragment >= 1, protocol_type = icmp.
```

This "ping of death" rule uses a conjunction of two conditional tests, that each checks the value of a feature, e.g., "wrong_fragment". Before we can apply classification algorithms, we need to first select and construct the right set of system features that may contain evidence (indicators) of normal or intrusions. In fact, feature selection/construction is the most challenging problem in building IDSs, regardless the development approach in use.

We exploit the temporal and statistical nature of network traffic and apply data mining programs to compute patterns for feature constructions. As shown in Figure 1, raw audit data, e.g., tcpdump [5] data of network traffic, is first processed into packet/event level ASCII data, which is further summarized into connection/session level records. Each record is defined by a set of basic and general-purpose features, e.g., start time, duration, source and destination hosts and ports, number of bytes transferred, and a flag that indicates the protocol-level behavior of the connection (e.g., SF for normal SYN and FIN), etc. Data mining algorithms that are optimized for audit data are then applied to compute various activity patterns from the audit records, in the forms of per-host and per-service frequent sequential patterns [9]. Given a set of extensively gathered normal audit data and another audit data set that includes an intrusion instance, we can compare the patterns from the normal data and the "intrusion" data to identify the "intrusion-only" patterns, i.e., those that exhibit only in the "intrusion" data. These patterns are then parsed to construct appropriate features that are predictive to the intrusion. For example, a pattern of "SYN flood" is shown in Table 1. Accordingly, the following features are constructed for "SYN flood": a count of the connections to the same destination host in the past 2 seconds, and among these connections, the percentage that are to the same service, and the percentage that have the S0 flag. In prior work [8], we showed that these constructed features have high information gain, and can therefore improve the accuracy of the classification rules.

The process of applying MADAM ID to build intrusion models, as shown in Figure 1, involves multiple steps and iterations. For example, poor performance of a model often suggests that additional features need to be constructed, and/or additional kinds of patterns need to be computed, etc. We have developed a process-centered approach to automate this process: the completion of one step automatically triggers the next step in the process; and heuristics are used to automatically tune the programs of each step, via parameter selection, to achieve performance improvement over the previous iteration. The resulting "process-centered" system can service real-time "model building" request by accepting

Table 1. An example of intrusion patterns

Frequent episode	Meaning
(flag=S0, service=http, dst_host=victim), (flag=S0, service=http, dst_host=victim) → (flag=S0, service=http, dst_host=victim) [0.93, 0.03, 2]	93% of the time, after two *http* connections with *S*0 flag (i.e., only one SYN packet is sent) are made to host *victim*, within 2 seconds from the first of these two, the third similar connection is made, and this pattern occurs in 3% of the data

audit data from an IDS and iterating through the process to compute a desired detection model.

3 An Overview of CIDF

In 1997, a group of research projects funded by DARPA began a collaborative effort called the Common Intrusion Detection Framework (CIDF). The motivation of CIDF was to provide an infrastructure that allows intrusion detection, analysis, and response (IDAR) systems and components to share information about distributed and coordinated attacks.

A major design goal of CIDF is that IDAR systems can be treated as "black boxes" that produce and consume intrusion-related information. According to the roles the IDAR components play in CIDF, they can be categorized as the event generators (E-boxes), analysis engine (A-boxes), response engines (R-boxes), and databases (D-boxes). All four kinds of CIDF components exchange data in the form of Generalized Intrusion Detection Objects (GIDOs), which are represented via a standard common format, defined using the Common Intrusion Specification Language (CISL) [18]. A GIDO encodes the fact that some particular events happened at some particular time, or some analytical conclusion about a set of events, or an instruction to carry out an action.

Given that there is a wide variety of intrusion-related information, CISL needs to be flexible and extensible. The main language construct of CISL is the general-purpose S-expression [16]. S-expressions are simply recursive groupings of tags and data. An example S-expression is:

```
(FileName '/etc/passwd')
```

This S-expression simply groups two terms FileName and '/etc/passwd' together. The advantage of S-expressions is that they provide an explicit association between terms, without limiting what those terms and their groupings might express. In CISL, intrusion-related data is expressed as a sequence of S-expressions with two or more elements. The first element always indicates how to interpret the data that follows, i.e., it is a tag that provides semantic "clue" to the interpretation of the rest of the S-expression. For this reason, these tags are called Semantic IDentifiers, or SIDs for short. As an example, the report of an event "user Joe deleted /etc/passwd" can be expressed as:

```
(Delete
      (Initiator
           (UserName 'Joe')
      )
      (FileSource
           (FileName '/etc/passwd')
      )
)
```

A set of CIDF APIs is provided for encoding and decoding GIDOs. Encoding a GIDO involves first translating the S-expression into a corresponding tree-like structure, then encode the structure into a sequence of bytes. Decoding the byte sequence back into a tree structure simply reverses the above procedure. Each SID code indicates, in a bit of the first byte, the type of argument that the SID takes: an elementary data, an array, or a sequence of S-expressions. The parser then interprets the succeeding bytes accordingly. The tree can be printed in S-expression format for further processing, i.e., extracting the intrusion-related data, by the CIDF component.

CIDF also provides a matchmaking service, i.e., a matchmaker, through which CIDF components can make themselves known to other components, and to locate communication "partners" with which they can share information, and request or provide services. The matchmaker supports feature-based lookup by grouping the CIDF components based on their capabilities. Communications in CIDF need to be as secured as possible because the intrusion-related data being transmitted is obviously critical to the well beings of the IDAR systems. The matchmaker thus provides authenticated and secured communications between CIDF components by acting also as a Certificate Authority (CA). The CIDF messages, i.e., GIDO packets, can include authentication headers and can be encrypted.

4 MADAM ID as a Modeling Engine in CIDF

Researchers have laid a lot of groundwork in defining CISL, GIDO encoding and decoding APIs, and the communication protocols, including matchmaking and authentication, between CIDF components. Research projects or experiments related to CIDF tend to focus on how two IDSs communicate, e.g., by exchanging attack and response data. In addition to studying how IDSs can cooperatively detect distributed attacks in real-time, we are interested in using CIDF to facilitate the distribution of new intrusion detection models so that "novel" attacks will have a very short "life span". That is, we study how a model builder, e.g., MADAM ID, can receive attack data, then rapidly and automatically produce appropriate models, and distribute them to IDSs for installations.

4.1 Design Considerations

When an IDS detects an attack, it can broadcast the instance report to other IDSs, which in turn can check whether the same attack is launched against their local environments or that some local suspicious activities may have caused the attack to other environment(s). Such cooperation among IDSs is well supported by CIDF. The challenge is to build up and maintain a set of SIDs and a dictionary on their possible values that can be used to accurately describe attack scenarios.

Introducing modeling service into CIDF is not as straightforward as it may seem. Note that an "analysis engine" of CIDF has limited capabilities in that it draws a conclusion from event data (i.e., whether and what kind of intrusion has occurred) and can even suggest a response, but it does not provide a detection method. When modeling service is available, upon detecting a new intrusion (as an anomaly), an IDS encodes the relevant audit data (e.g., network traffic within a time window) into GIDOs and transmits the GIDOs to the "modeling engine", i.e., the process-centered MADAM ID system. MADAM ID then performs pattern mining, feature construction, and rule learning using the audit data extracted from the GIDOs. MADAM ID keeps a large store of "baseline" normal patterns so that "intrusion patterns" can be easily identified for the reported attack/anomaly. It also keeps a large amount of "historical" attack data and patterns so that if the reported attack is old attack or a slight variant, an "updated" rule can be produced by training on the combination of "historical" and "new" data. We see here that a modeling engine includes the functionalities of an analysis engine and database.

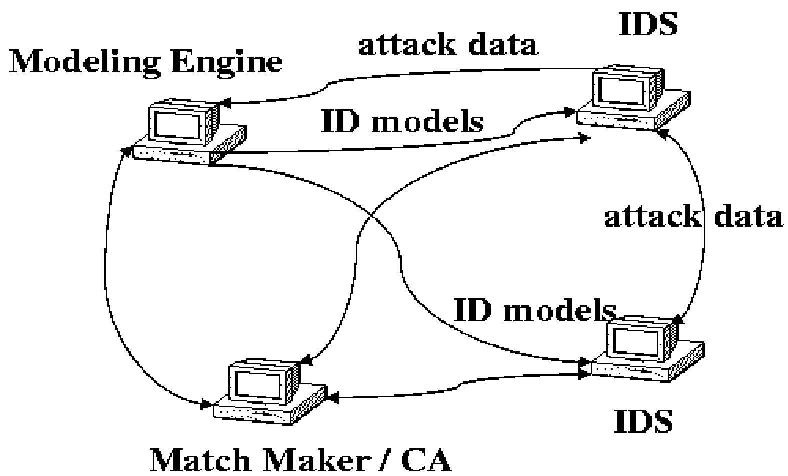

Fig. 2. A CIDF Architecture.

The most challenging issue in adding a modeling engine into CIDF is the encoding of features and rules into GIDOs. A set of SIDs need to be defined to express the computational instructions for each feature and rule so that an IDS receiving a "ID model GIDO" can automatically parse the instructions and generate local execution modules. Each rule can also include the "accuracy" (i.e. confidence) measurement so that an IDS can decide whether to accept or reject the rule. An updated rule will also have the special tag "updated" so that old rule can be replaced. Figure 2 shows the architecture of CIDF where IDSs can share attack information with each other, and send attack data to the modeling engine, which in turn computes and distributes new detection models. A matchmaker is responsible for hooking up the IDSs with each other and with the modeling engine, and for facilitating authenticated and secured communications between CIDF components.

4.2 Implementation of an Experimental System

We implemented an experimental system, based on CIDF, where MADAM ID is the modeling engine, and Bro [14] and NFR [13] are the two real-time IDSs. We also implemented a system that acts as both a matchmaker and a CA. We described our experiences here.

The Modeling Engine As shown in Figure 1, MADAM ID normally starts the model building process from raw audit data, but can also directly use the processed connection/session records. In either case, an IDS needs to supply audit data to MADAM ID. We used a new SID AuditData that at the encoding end specifies a (local) file that contains the audit data, and another tag AuditDataSize that specifies the size of the audit data. When encoding an S-expression that contains these two tags, the audit data size is recorded in the GIDO object, followed by the content of the audit data file, so that the receiving end has sufficient information to accurately extract the audit data.

To encode features and rules in GIDOs, we first enumerated a set of "essential" (i.e., boot trap) features that all IDSs must know how to compute. These features include: source host, source port, destination host, protocol type, service, duration, flag, etc. We assigned each feature a unique id, and introduced a number of new SIDs for specifying features and their values. For example, the feature condition "flag has value S0" is represented as the following S-expression:

```
(FeatureCond
   (FeatureID 'flag')
   (Cond
        (CondOperator 'equal')
        (CondValue 'S0')
   )
)
```

MADAM ID constructs new features as functions (i.e., some operations or computations) of some (existing) feature conditions for the connections that satisfy certain constraints. The feature construction operators include count, percent, average, etc. The data constraints include same (e.g., same destination host or same-service), different, time window (e.g., 2 seconds), etc. We used a number of new SIDs to define new features. As an example, one of the "SYN flood" features, "for the connections to the same destination host in the past 2 seconds, the percentage that have the S0 flag", is expressed as the following S-expression:

```
(FeatureDef
   (FeatureID 'S0_rate')
   (Constraint
       (ConstraintType 'same')
       (ConstraintValue 'destination host')
   )
   (Constraint
       (ConstraintType 'time')
       (ConstraintValue '2 seconds')
   )
   (Operation
        (Operator 'percent')
        (FeatureCond
            (FeatureID 'flag')
            (Cond
                (CondOperator 'equal')
                (CondValue 'S0')
            )
        )
)
```

An intrusion detection rule, which is simply a sequence of conjuncts on feature conditions, can then be specified using the following form:

```
(DetectionRule
  (AttackType ...)
  (FeatureDef ...) [optional]
  (FeatureCond ...)
)
```

That is, the expression first specifies the type of attack (i.e., an intrusion name) that can be detected by this rule. It then describes the definitions of any new features used in the rule, followed by a sequence of feature conditions.

We can see that as long as the IDSs and the modeling engine understand the same vocabulary, i.e., the set of SIDs and the specific values (i.e., the features, operators, constraints, etc.), new intrusion detection models can be expressed and parsed unambiguously. When an IDS rejoins CIDF after a period of absent, its vocabulary needs to be updated, by exchanging GIDOs with the modeling engine.

In our experimental system, MADAM ID listens to a socket for incoming audit data, and sends new intrusion detection models to all IDSs. A "wrapper", which consists of function calls to CIDF APIs, along with data dictionaries of the features, operators, constraints, etc., is added to MADAM ID so that audit data can be extracted and detection models can be encoded.

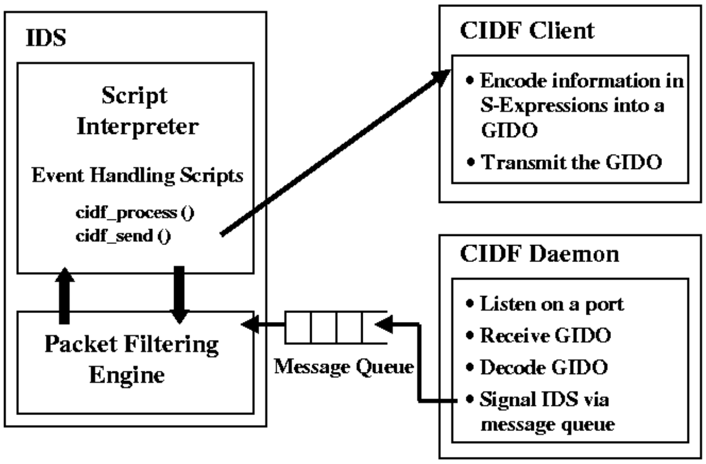

Fig. 3. The CIDF interfaces of IDS.

The IDSs We used Bro and NFR in our experimental system for two reasons. First, they are "programmable" IDSs because both filter network traffic streams into a series of events, and execute scripts, e.g., Bro policy scripts or NFR N-codes, which contain site-specific event handlers, i.e., intrusion detection and handling rules. Since the event handling scripts are interpreted by the IDSs, adding new detection rules does not require re-building the IDSs. Such feature facilitates the fast updates of IDSs when new detections models are distributed. The second consideration is a practical one: we have been using these two real-time IDSs for various experiments in the past couple of years, and have the source codes of both systems.

As illustrated in Figure 3, for each IDS, we implemented a CIDF daemon, which is responsible for receiving and decoding GIDOs, and a CIDF client, which is responsible for encoding and sending GIDOs. Upon execution, the CIDF Daemon creates a Shared Message Queue that allows it to communicate with the IDS whenever new messages arrive. This queue will be constantly monitored by the IDS (more details to follow). The CIDF Daemon will be listening on a port, e.g., 3295 (0x0CDF), for incoming GIDOs. Upon connection from a client (i.e., a CIDF Client), the daemon verifies that the received message is a valid GIDO, decodes it to S-expressions. The decoded data is then queued into the Shared Message Queue to signal the IDS for further processing.

The packet filtering engine of the each IDS was modified in order to support communication with the CIDF Daemon. Since "connection-finished" is a condition checked by a packet filtering engine, the probing of the Shared Message Queue is scheduled to occur whenever the engine processes a connection-finished event. Upon receiving the message (i.e., the S-expressions), the packet engine queues an event along with the S-expressions into the Event Queue that it shares with the script interpreter. CIDF-related events e.g., interpreting the S-expressions to take appropriate actions, are handled by the cidf_process script function. A "model interpreter" is implemented as a separate process, invoked by the cidf_process function, to parse the S-expressions that describe a new intrusion detection rule. The generated local intrusion handlers can then be inspected by human experts and loaded into the IDS. Note that the reason that incoming data, from other CIDF components, have to come through the packet engine to reach event handlers is that not all IDSs supports the functionality by which the interpreted script can load data from local file system into the program space (of an IDS).

The CIDF Client provides the means for an IDS to encode and send GIDOs to other CIDF components. Whenever there is information (e.g., audit data, intrusion alerts, etc) to be sent, the request is serviced by the "cidf_send" script function. cidf_send firsts construct S-expressions and writes the data to a local file. It then invokes the CIDF Client process and passes the filename to it. The CIDF Client parses the S-expressions in the file and encodes them into a GIDO. The GIDO is in turn sent to the modeling engine or other IDSs. An example S-expression on a "SYN flood" attack instance is the following:

```
(Attack
          (AttackSpecifics
               (AttackID 0x00000004 0x00000000)
          )
          (When
               (Time 953076974.391314)
          )
          (Message
               (Comment 'SYN flood')
               (SourceIPV4Address 181.162.15.74)
               (TCPSourcePort 1551/tcp)
               (DestinationIPV4Address 152.1.205.85)
               (TCPDestinationPort 80/tcp)
               (TCPConnectionStatus 1)
          )
)
```

The Matchmaker and CA The CIDF matchmaker is used by the components to locate "partners" with whom they can communicate with. For our implementation, the matchmaker is integrated with the CA server, i.e. they are the same process. The matchmaker maintains a list of all CIDF components and their roles (i.e. modeling engine, event engine, etc). New components and roles can be added as desired. When a component needs to communicate with another component, it can do so in either of two ways. If it knows the IP address or hostname of the other component, then it can contact that component directly for communication. In this case, our matchmaker does not play any active role. However, if the component only knows that it wants to communicate with a CIDF component with certain capability, it sends this request to the matchmaker along with any other criteria. The matchmaker then looks up the list of IDSs that satisfy the criteria specified and returns the "matched" list to the component. If no suitable match is found, then a "matchmaking failure" message is returned.

The CA provides authentication of components and enable their secured communications using RSA public key encryption technology. When components A and B need to establish a secured communication, they need to first authenticate each other: A first uses its private RSA key to sign a random nonce generated and given to it by B; it then returns this singed nonce along with its own random nonce to B, who verifies the signature on it using A's public RSA key, signs the A's nonce using its own private RSA key and sends the signed nonce back to A; A then verifies the signature using B's public RSA key. Once the components authenticate each other, they can use their RSA keys to establish a per-session secret key. DES (Data Encryption Standard) secret key algorithm is then used to encrypt the data transmission between the components. Before encrypting any data to be sent to its peer, a component compresses it using the lossless data compression algorithm. Similarly the peer must uncompress the data after decrypting it. Compression not only reduces the size of the data being transmitted, but also provides an added layer of data confidentiality.

Certificates signed by CA are used by the components to exchange their public RSA keys in the first place. A certificate contains the following fields: the ID of the CIDF component (i.e. the IP of the host it is running on); the component's public RSA key; the current time stored as a timestamp. A certificate is considered as valid (i.e. not yet expired) if its timestamp is within the past 90 minutes. We assume that all components know the public RSA key of the CA in advance. This is needed to authenticate the CA and also verify the signature on certificates signed by the CA. This public RSA key must be distributed to the components by some other means such as manual entry.

5 Experiments

We deployed our experimental system in our campus network where Bro, NFR, and MADAM ID are in separate subnets. We conducted a series of experiments to test the limits and strength of this system. We describe some of the findings here.

Our first set of experiments was designed to test the interoperability of the CIDF components. The results were then used to fine-tune the implementation of our system. For example, we simulated "SYN flood" attack against the subnet that Bro was monitoring. Bro detected the attack and sent a GIDO describing the instance to NFR. We compared the S-expressions on both sending (i.e. Bro) and receiving (i.e. NFR) ends to verify that they matched. In another experiment, we took out the "SYN flood" from Bro so that it could not detect the attack as a specific intrusion, but rather an anomaly based on the unusual traffic statistics caused by the attack. Bro then sent then tcpdump data to MADAM ID for a new intrusion detection model. We verified that a new rule was computed and distributed to both Bro and NFR. Both IDSs were able to translate the rule into their local script functions.

Our second set of experiments was designed to test the limit of the modeling engine. In a set of "timing" experiments, we measured how long it took for MADAM ID to compute a detection model after it received the audit data. We found that the results had a wide range, from mere seconds to a few hours. Upon detailed analysis, we discovered that the automated iterative process of mining data, constructing features, computing classifiers, and evaluating performances requires more guidance in its heuristic settings of parameters for the programs. Otherwise, the exhaustive search can be very slow. For example, detecting "SYN flood" may require patterns of "same destination host and same service" be computed and compared (with normal patterns), while "Port-scan" requires "same destination host and different service" patterns. We are studying whether it is feasible for an IDS to include a "high-level" description of the attack behavior in the GIDO so that the modeling engine can have a better chance of setting the correct parameters for the data mining algorithms. In a set of "modeling" experiments, we tried to discover whether some new models could not be expressed as S-expressions because of our current assumptions that every IDS "understands" the same boot-trap set of features and that new features can be expressed using the existing features. We indeed found that for intrusions that require new pre-processing from raw audit data (hence new boot-trap features), it is virtually impossible to express the complex and delicate data pre-processing logic in S-expressions. For example, "teardrop" attacks require special processing of IP fragments, which can only be expressed in high-level programming language such as C/C++. In fact, human experts must get involve in the modeling process when such new data pre-processing is required (since without these new features defined by human experts, the automatic data mining process will not be able to produce a good model and will terminate when it uses up the allotted CPU time in the model building process). We are studying how to distribute new data pre-processing codes in CIDF.

Our third set of experiments was intended to verify that the IDSs could work together to detect distributed attack. We used the Tribal Flood Network (TFN) DDOS attack tool for our experiment. Bro monitored the subnet of the target host and detected the attack, which was launched by several slave programs, each running in a separate host, in the subnet monitored by NFR. When NFR received the attack instance report from Bro, it parsed the GIDO and recognized that a SYN flood seemed to have originated from its subnet (by checking the SourceIPV4Address SID). It then activated a N-code that detects and blocks the "control message" (ICMP echo reply packets) sent by the attack "masters" to the "slaves". It also launched a search for the slave programs, by informing a special daemon on each system in its subnet to look for and kill any running process with a name that matches any in the list of known attack programs. Our results showed that within a few seconds after Bro detected the attack, NFR was able to kill all attacking programs.

6 Related Work

EMERALD [15] provides architecture to facilitate enterprise-wide deployment and configuration of intrusion detectors. A "resolver" is used to combine the alarms from the distributed detectors to make a determination of the state of the (entire) network. This is certainly a right direction for detecting coordinated attack against the enterprise. The resolver technology can also be utilized for event analysis in a CIDF environment. This scope of the system is limited to an enterprise. We are more interested in the problem of how IDSs can collaborate over the Internet, and more importantly, how to automatically produce and distribute new intrusion detection models for "novel" attacks.

Kephart et al. [6] outlined a system architecture where anti-virus systems across the Internet can subscribe to a centralized virus modeling server to receive fast updates whenever a new virus is discovered and a new anti-virus module is produced. This is very similar to our idea of adding the modeling engine to CIDF. Our system has the additional capability of facilitating the IDSs to exchange attack information to detect distributed intrusions.

7 Conclusion and Future Work

In this paper, we discussed the need for new techniques to detect novel intrusions as well as distributed attacks. We proposed to add a modeling service to CIDF so that IDSs can not only exchange attack data to detect distributed intrusions, but also receive detection models once a new attack method surfaces. We described the underlining technologies of our approach, namely, MADAM ID, a data mining framework for automatically building intrusion detection models, and CIDF, a framework for IDAR components to collaborate. We discussed the design and implementation of an experimental system, which uses MADAM ID as the modeling engine, and Bro and NFR as the real-time IDSs. Although our experiments are still preliminary, the promising results showed that these components can interoperate to detect distributed attacks, and can produce and distribute new intrusion detection models.

As for future work, we plan to first conduct more extensive and robust experiments. We will install the components of our experimental system to separate domains over the Internet for a new set of "timing" experiments. We will also run an extensive set of attacks, for example, those that are generated by DARPA "red team", to test whether our system can indeed achieve a better detection performance over a single system.

We will continue to develop the underlying technologies of our system. In particular, we will investigate how to improve the automated process of building intrusion models, and how to encode and distribute features and rules that require detailed system knowledge and are beyond the scope of current CISL.

8 Acknowledgments

This research is supported in part by grants from DARPA (F30602-96-1-0311). Our work has benefited from in-depth discussions with Matt Miller and Dave Fan of Columbia University, and Jim Yuill, Felix Wu, and Doug Reeves of North Carolina State University.

References

1. W. W. Cohen. Fast effective rule induction. In *Machine Learning: the 12th International Conference*, Lake Taho, CA, 1995. Morgan Kaufmann.
2. D. Dittrich. Distributed denial of service (ddos) attacks and tools. http://staff.washington.edu/dittrich/misc/ddos/.
3. R. Heady, G. Luger, A. Maccabe, and M. Servilla. The architecture of a network level intrusion detection system. Technical report, Computer Science Department, University of New Mexico, August 1990.
4. K. Ilgun, R. A. Kemmerer, and P. A. Porras. State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, March 1995.
5. V. Jacobson, C. Leres, and S. McCanne. *tcpdump*. available via anonymous ftp to ftp.ee.lbl.gov, June 1989.
6. J. O. Kephart, G. B. Sorkin, M. Swimmer, and S. R. White. Blueprint for a computer immune system. Technical report, IBM T. J. Watson Research Center, Yorktown Heights, New York, 1997.
7. S. Kumar and E. H. Spafford. A software architecture to support misuse intrusion detection. In *Proceedings of the 18th National Information Security Conference*, pages 194–204, 1995.
8. W. Lee. *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems*. PhD thesis, Columbia University, June 1999.
9. W. Lee, S. J. Stolfo, and K. W. Mok. Mining audit data to build intrusion detection models. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, New York, NY, August 1998. AAAI Press.
10. W. Lee, S. J. Stolfo, and K. W. Mok. A data mining framework for building intrusion detection models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, May 1999.
11. R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunninghan, and M. Zissman. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, January 2000.

12. T. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P. Neumann, H. Javitz, A. Valdes, and T. Garvey. A real-time intrusion detection expert system (IDES) - final technical report. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, February 1992.
13. Network Flight Recorder Inc. Network flight recorder. http://www.nfr.com, 1997.
14. V. Paxson. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.
15. P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *National Information Systems Security Conference*, Baltimore MD, October 1997.
16. R. Rivest. S-expressions. Internet-Draft draft-rivest-sexp-00.txt, expired 1997.
17. S. Stainford-Chen. Common intrusion detection framework. http://seclab.cs.ucdavis.edu/cidf.
18. B. Tung. The common intrusion specification language: A retrospective. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, January 2000.