Fusing a Heterogeneous Alert Stream into Scenarios*

Oliver Dain
Massachusetts Institute of Technology
Lincoln Laboratory

Robert K. Cunningham
Massachusetts Institute of Technology
Lincoln Laboratory

14th September 2001

Abstract

An algorithm for fusing the alerts produced by multiple heterogeneous intrusion detection systems is presented. The algorithm runs in realtime, combining the alerts into scenarios; each is composed of a sequence of alerts produced by a single actor or organization. The software is capable of discovering scenarios even if stealthy attack methods, such as forged IP addresses or long attack latencies, are employed. The algorithm generates scenarios by estimating the probability that a new alert belongs to a given scenario. Alerts are then added to the most likely candidate scenario. Two alternative probability estimation techniques are compared to an algorithm that builds scenarios using a set of rules. Both probability estimate approaches make use of training data to learn the appropriate probability measures. Our algorithm can determine the scenario membership of a new alert in time proportional to the number of candidate scenarios.

Keywords: security, fusion, scenarios, correlation, intrusion detection

1 Introduction

Many security-conscious organizations protect their networks with multiple intrusion detection systems (IDSs). While this does improve attack detection rates, it also increases the work load on intrusion detection analysts; there are now more components to be monitored and false alarm rates tend to rise. Fusion systems attempt to alleviate these problems by collecting the alerts produced by all the systems on the network. The collected alerts can then be managed from a single location.

In this paper we propose a realtime algorithm to combine the alerts produced by multiple intrusion detection systems into scenarios. Each scenario is intended to indicate a sequence of actions performed by a single actor or organization. It is important to realize that a scenario constructed by this algorithm does not necessarily indicate malicious behavior. The purpose of the scenarios is simply to group alerts that share a common cause. The resulting scenarios give network defenders a more complete picture of the traffic on their network. Additionally we can assign false alarm probabilities to whole scenarios rather than individual alerts, and thus decrease false positive rates while increasing the sensitivity of the underlying sensors.

Each time a new alert is produced the algorithm must compare it to scenarios that have already been constructed. For each existing scenario the probability that the new alert belongs to this scenario must be calculated. We compare two techniques for producing the required probability estimates. Both approaches use training data to learn the appropriate probability measures. These techniques are compared to a naïve algorithm that builds scenarios using a set of hand coded rules. The algorithm runs in time proportional to the number of existing scenarios, and is capable of finding scenarios even if an intruder used stealthy attack methods such as forged source IP addresses or long latencies between attack components.

*This work was sponsored by the Department of Defense under Air Force contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Air Force.

As Edward Amoroso points out, very little research has been done on correlation or fusion[2]. Much of the work in this area has focused on collecting alerts from multiple detectors at a single location where they can be displayed, queried, and correlated[17]. The GrIDS system developed at University of California, Davis uses rule sets to combine alerts and network data into a graph structure capable of discovering large scale coordinated attacks[15]. Clifton and Gengo use data mining techniques to discover common sequences of alerts[4]. These alert sequences can, at the discretion of human operators, be filtered so they are no longer displayed to an analyst. The discovered sequences are not intended to be scenarios; they are simply sequences that appear frequently. Valdes and Skinner have studied correlation and scenario building from probabilistic detectors such as their EMERALD eBayes system[16]. Each alert and meta alert (scenario) in their system is represented by a vector of attributes. A similarity measure is applied to the vectors and those that exceed a threshold are joined together into a scenario. Our approach is similar to the one proposed by Valdes and Skinner but we use a different scenario construction algorithm and optimize our probability estimates so that the system forms scenarios like a human expert. Furthermore we have tested our system using output from both commercial and proprietary intrusion detection systems.

2 Fusion Approach

The fusion system develops scenarios as alerts are received from the component Intrusion Detection Systems. Thus, our task at any given instant is to determine the scenario membership of a new alert given the alerts we have already seen. The most flexible way to do this is to consider all possible ways of combining the alerts into scenarios each time a new alert is generated. Unfortunately this quickly becomes computationally intractable. To see this, consider the simpler problem of deciding which of n existing alerts belong in a scenario with a given new alert. Figure 1a shows possible scenario assignments for a new alert, C, given two existing alerts. We note that there are 4 possibilities. Figure 1b shows the 8 possible scenario assignments for a new alert, D, given three existing alerts. We note that we could add alert D to any of the scenarios enumerated in Figure 1a. Additionally D could be grouped with any of these scenarios with C removed. Thus the addition of each alert doubles the number of possible scenario constructions so that a new alert may be grouped with n existing alerts in 2^n ways. Since our task is not simply to construct a scenario in which to place a single new alert but rather to construct scenarios for all alerts the complexity of the problem is super-exponential. Clearly it is not possible to consider all such arrangements in realtime.

Our solution is to use an "atom model". Each time we receive a new alert from a sensor we compare it to the scenarios we have constructed thus far. For each existing scenario we calculate the probability that the new alert belongs to this scenario. The new alert is then assigned to the scenario that produced the highest probability score. If all the scores are below a threshold the new alert is not joined to any scenario and instead starts a new scenario. Once the alert has joined a scenario we do not consider changing this assignment. Hence the scenario is not divisible and is "an atom" - it may absorb other alerts and grow, but it can not be split. Although it may cause errors in the assembled scenarios, this simplification reduces the computational complexity from super-exponential in the number of alerts to linear in the number of candidate scenarios. Despite this simplification the system still performs well (see Section 6). Informal testing indicates that the system is able to construct scenarios from over 16,000 alerts in less than two seconds.

3 Architecture

The fusion system consists of three distinct components: the IDSs that produce the alerts, a database where the alerts are stored and the fusion software itself. The IDSs currently used by the fusion system are Internet Security System's RealSecure network sensor[11] and MIT Lincoln Laboratory's Battlefield Intrusion Detection System (BIDS), a host based system capable of finding known and novel attacks[6]. The alerts produced by these IDSs are converted to a standard form and written to a SQL database. The fusion system reads from the database, determines the scenario membership of any new alerts, and then saves the results back to the database. Although our research

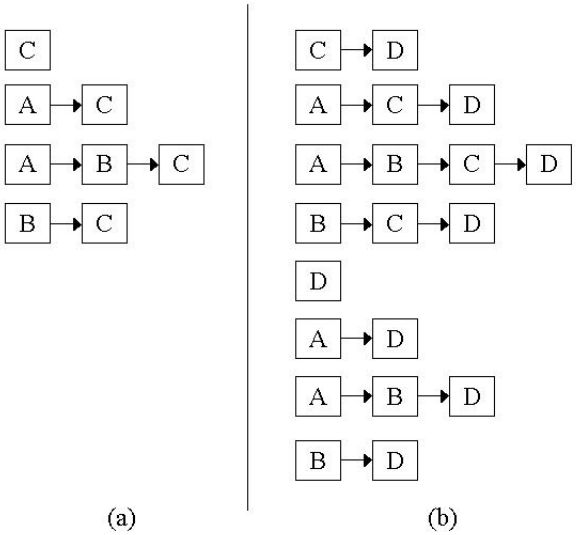

Figure 1: (a) Possible scenario assignments for a new alert, C, given 2 existing alerts, A, and B. (b) Possible scenario assignments for a new alert, D, given 3 existing alerts, A, B, and C.

has thus far focused on the two sensors mentioned, the algorithm would work with any sensor that could send its output to this database. Communication between the component IDSs and the database currently relies on custom "bridging" software. As vendors begin to support the the Internet Engineering Task Force Intrusion Detection Working Group's emerging Intrusion Detection Message Exchange Format[8] we will consider supporting this as our communication mechanism.

4 Definitions

	Decimal	Binary
Address 1	172.16.112.20	10101100.00010000.01110000.00010100
Address 2	172.16.112.40	10101100.00010000.01110000.00101000
Mask	255.255.255.192	11111111.11111111.11111111.11000000

$r = 26$

Table 1: r value calculation for addresses 172.16.112.20 and 172.16.112.40

RealSecure version 5.0 is capable of identifying over 250 different attack types. Because attackers can use multiple techniques to accomplish the same result[18] each of these alerts is assigned to one of five categories: *discovery*, *scan*, *escalation*, *denial-of-service* (DoS), and *stealth*. Alerts in the *discovery* category are indicative of network discovery activity (IP sweeps, DNS zone transfers, etc.). Alerts in the *scan* category indicate port scanning activity. The *escalation* category consists of privilege escalation type attacks (password guessing, buffer overflows etc.). The *DoS* bucket consists of attacks that prevent access to services (SYN floods, Smurf attacks, etc.). The *stealth* bucket contains alerts that indicate attempts to conceal identity (forged IP addresses, etc.).

As an attacker may be able to launch attacks from several different, legitimate IP addresses (the attacker may have access to several machines on a single subnet or may be running DHCP), it is useful to be able to quantify the proximity of two IP addresses. To this end we have defined a quantity we denote r. r is the maximum number

of 1 bits in an IPv4 subnet mask that could account for the two addresses. Thus $r = 32$ when the two IP addresses are identical, and $r = 0$ when there is no way the two addresses could be on the same subnet (see Table 1). Due to the prevalence of Classless Inter-Domain Routing (CIDR) [9] we allow two addresses that are on different class A, B or C subnets to have a value of r greater than 0 as long as some of their high order bits are the same. For example, the two class B addresses 130.114.5.20 and 130.115.100.8 differ in their second octet, thus they can't be on the same class B network. However, if an organization owned both the 130.114.x.x and 130.115.x.x networks they could treat them as one large subnet via CIDR. Therefore we would give these two addresses an r value of 15 since the first 15 bits in both addresses are the same.

5 Probability Assignment

Thus far we have specified how scenarios are formed given a measure of the probability that an event belongs to a given scenario. The following sections describe different approaches to the probability estimation problem. Section 5.2 describes a naïve approach, Section 5.3 describes a heuristic method, and Section 5.4 describes the application of traditional data mining techniques to this problem. Both the heuristic and data mining approaches make use of training data to optimize the probability estimates. The process by which data is collected and prepared for use is described in Section 5.1.

5.1 Data Sources and Use

Given a set of network alerts and the corresponding scenarios we could use this data to train the free parameters in our probability estimate algorithms to reproduce the scenarios in the data set. Unfortunately, the authors know of only one dataset that contains realistic network traffic and attack scenarios for which ground truth is known[10] and this dataset does not contain enough scenarios to be used for parameter estimation. However, if the fusion system could match the performance of a human expert this would be a valuable contribution. We therefore find scenarios in real network traffic by hand and use this data for optimization.

Every year at the DEF CON conference participants play "capture the flag"[1]. There are two types of participants in this game: those who run servers and those who try to break into them. The former group gets points for running more services (as this makes the host more vulnerable) and the latter group gets points for each server they compromise. The rules stipulate that a host is considered compromised only if the attacker can put a file containing his or her name in the root of the file system. Additional points are awarded for "style".

This is an excellent data source because it contains a large amount of attacks and scenarios. It is, however, an unusual data source. The volume and frequency of attacks is far greater than that experienced on most operational networks. Additionally, all hosts participating in the game were on the same subnet. Ordinarily such a large volume of attacks in a small time window emanating from a single subnet would be indicative of a coordinated attack by a single organization. While that is not the the case here, this type of activity is similar to what one might see in a cyber terrorism attack. For example, recently many Israeli web sites have been the target of coordinated attacks by Palestinian groups[14]. The type of traffic experienced by these sites may be similar to the traffic in this data set.

We believe that different tactical situations would produce different alert stream characteristics. For example, a "cyber war" would produce a very different alert profile than would normal traffic. A good fusion approach should be flexible enough to work in all such environments. In our case, the optimal parameters would change with the tactical situation, but these could be easily swapped via a configuration file. Thus far we have only analyzed data from the DEF CON 2000 hacker conference's "capture the flag" game. We are investigating other data sources that contain more typical traffic patterns. Results on these data sources will help us determine how robust our approach is to different tactical situations.

Two and a half hours of tcpdump data from the DEF CON conference was replayed on a network using Lincoln Laboratory's Netpoke tool. All traffic was sniffed by ISS RealSecure's network sensor (in its "out of the box" configuration). This produced 16,250 alerts. The alerts were mapped by the first author into eighty nine

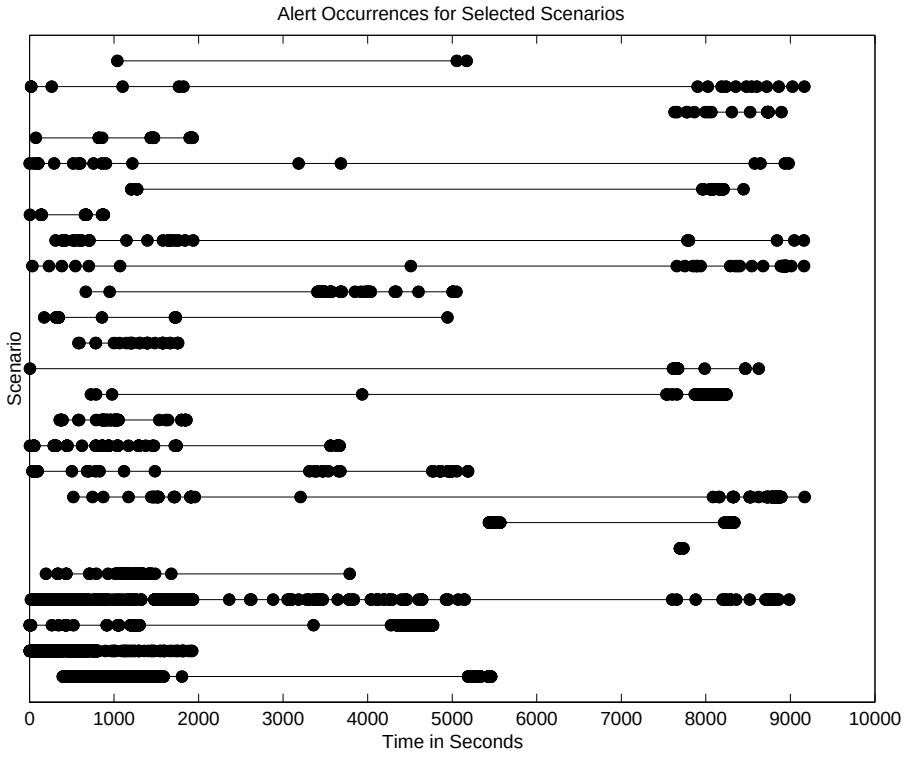

Figure 2: Alert occurrence times in the DEF CON data. The x-axis is time in seconds since data collection began. Each horizontal line depicts a single scenario. The endpoints of the line indicate when the scenario began and ended. The black circles indicate when an alert occurred. Depicted are the twenty five scenarios containing the largest number of alerts.

scenarios. The largest scenario contained 10,912 alerts. Twenty seven scenarios contained only a single alert. The average number of alerts per scenario was 183. Twenty one scenarios contained alerts from more than one source address. All the remaining scenarios contained alerts from a single source IP address. The length of time covered by the alerts in a scenario varied widely (see Figure 2). Often two consecutive alerts in a scenario were separated by hundreds of alerts belonging to other scenarios making scenario reconstruction difficult.

When hand tagging scenarios some idiosyncrasies of ISS RealSecure version 5.0 were discovered. In particular we noticed that the source and destination addresses reported by RealSecure were often the reverse of what would be expected. We expect the attacker's address to be listed as the source and the victim's address to be listed as the destination. The majority of RealSecure alerts follow this pattern, but some do not. [1] Classification features designed to account for these idiosyncrasies are discussed in Section 5.4.

Tagging the scenarios provides us with examples of data which should cause our probability estimate algorithms to produce a "join the scenario" decision (a high probability that the alert belongs to the scenario). Optimization requires negative ("don't join the scenario" decision) training examples as well as positive ones. In order to produce negative training examples a program was written that reproduces the decisions the computer would need to make given a set of data. This program takes a file which contains the alerts in the order in which

[1] It appears that RealSecure version 5.0 always reports *windows_access_error*, *netbus*, and *dns_length_overflow* alerts backwards while *dns_all*, *iphalfscan* and *udp_port_scan* alerts are only sometimes reported backward. The *dns_all*, *iphalfscan* and *udp_port_scan* alerts appear to be reported backward as an "echo" effect. They are only reported with the source and destination address backward shortly after the same alert has been reported correctly. Not every occurrence of one of these alerts is followed by another with the source and destination reversed.

they were produced. Each alert is tagged with a number indicating its scenario membership. The program loops through this file and writes one line of output for each decision the fusion system would have had to make. The output contains the "correct" decision and the statistics on which this decision must be based (time difference between alerts, r value of the IP addresses in question, etc.). Pseudo-code for this algorithm is included below:

- `WHILE there are still alerts in the file`
  - `Read the next alert from the file`
  - `FOR each scenario in memory`
    - `If ID of current scenario matches ID of new alert then positive training example, else negative training example`
    - `Generate and write out features`
  - `END FOR`
  - `Add new alert to correct scenario`
- `END WHILE`

Running this algorithm on the alerts in this dataset produced 1,001,436 training examples.

Once the training data has been generated it must be separated into training, validation and test data sets. The validation and test data sets are used to ensure that we are learning a function which will generalize well to other data (see [13] for an discussion of overfitting). In order to ensure accurate results the proportion of "join" and "don't join" examples must be equal in the train, test and validation data files. A script was written to randomly split the data into these three files (50% for training, 25% for evaluation, and 25% for testing) while preserving the prior probabilities.

5.2 Naïve Technique

The most obvious way to group alerts together is by source IP address. This naïve approach assumes that all attacks belonging to a single scenario share a common source address. Thus scenarios are formed by comparing the new alert to the most recent alert in the scenario. The new alert is added to the scenario if and only if the source IP addresses are identical.This technique was tried, but due to the tendency of RealSecure to reverse source and destination IP addresses it did not perform very well.

To compensate for this we applied some simple heuristics to correct for alerts whose source and destination address might be reversed. To deal with alerts whose source and destination addresses are always reversed we simply replaced the source address with the destination address before comparing it to other alerts. Other alerts are only sometimes reversed. This usually occurs as an "echo" effect where an alert is reported twice; first with the source and destination correct and then with the source and destination exactly reversed. We handled these alerts by producing a join decision when the new alert and the most recent alert in the scenario were of this type and their source and destination addresses were exactly opposite. Results of this approach are presented in Section 6.

5.3 Heuristic Technique

The technique discussed in this section extends work described earlier [7]. It attempts to model our intuitions about how to form scenarios. We believe that:

- Attacks often follow a logical progression. As such, certain sequences of alerts types would be more common than others.
- The time between alerts is a good predictor of scenario membership. Alert latencies vary with alert type.

- The range of source IP addresses in a group of alerts is a good predictor of scenario membership. The range of addresses one is likely to see is dependent on the alert type; some attacks lend themselves to IP spoofing and others do not.

We approximate these intuitions with a bigram model. Given a scenario and a new alert, probability is assigned by considering the most recent alert in the scenario and the new alert. Thus if our scenario contains three alerts, a_1, a_2, and a_3, and we get a new alert, a_{new}, we compare a_{new} to a_3 only. The probability that two alerts, from buckets i and j respectively, belong in the same scenario is the product of three quantities: l_{ij}, which accounts for the strength of the link between the two alerts, $\sigma_{ij}(\Delta t)$, which accounts for the time lag between the two alerts, and $R_{ij}(r)$, which accounts for the source IP addresses of the two alerts. All three quantities are between 0 and 1, thus their product is also between 0 and 1 (see Figure 3).

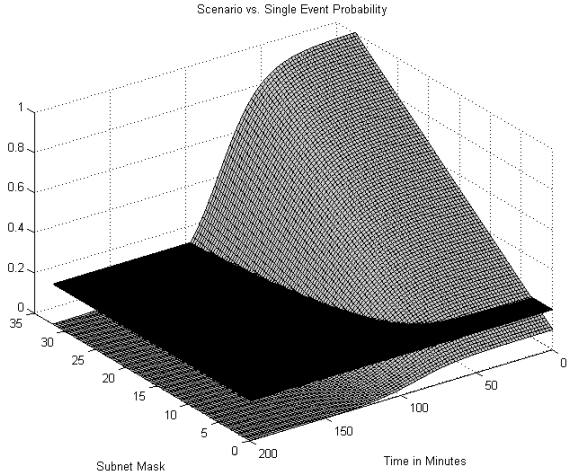

Figure 3: Decision surface for a single transition type. The height of the curve is the probability that two alerts in the given time span with the given r value belong in the same scenario. The black plane is the threshold below which the new alert would not join the scenario and would start a new scenario.

The first quantity, l_{ij}, is the probability that an alert from bucket i is followed by an alert from bucket j. For example the probability that a scan will be followed by an escalation might be higher than the probability than an escalation will be followed by a DoS because an attacker often needs to scan a system prior to gaining access to it while an attacker who has gained unauthorized access to a system is less likely to deny services to a network.

The second quantity, $\sigma_{ij}(\Delta t)$, accounts for the time between the alerts. It is a sigmoid function, defined by $\sigma_{ij}(\Delta t) = 1/\left(1 + e^{\alpha+\beta\Delta t}\right)$. The shape of the function (determined by the values of α and β) is different for each transition type (where transition type is defined by the buckets containing the two alerts). Thus the sigmoid representing the time between DoS attacks might fall off quickly (as we expect these alerts to occur in rapid succession) while the sigmoid representing the time between network discovery and privilege escalation might fall off rather slowly since an attacker might not launch an attack immediately after gathering information.

The final quantity, $R_{ij}(r)$, accounts for the IP address range of the two alerts. Let r be the maximum number of 1 bits in an IPv4 subnet mask that could account for the two IP addresses (as defined in section 4). $R_{ij}(r)$ is a function of r. The exact function depends on the buckets i and j. For each transition type values of $R_{ij}(r)$ are picked for r = 0, 8, 16, 24 and 32. The value of $R_{ij}(r)$ for other values of r is determined by linear interpolation. For example, since source addresses are often spoofed in a DoS attack we might expect $R_{ij}(r)$ to have a relatively high value for all values of r if $i = j =$ DOS.

To compensate for the idiosyncrasies of RealSecure the same heuristics used by the naïve approach were applied before r was calculated. $R_{ij}(r)$ is then calculated from this modified value of r. The following algorithm

is used to calculate r:

- `new_a = new alert`
- `r_a = most recent alert in the sceanario`
- `if always_reversed(new_a) and not always_reversed(r_a) then r = calc_r(new_a.destination, r_a.source)`
- `else if always_reversed(r_a) and not always_reversed(new_a) then r = calc_r(new_a.source, r_a.destination)`
- `else if sometimes_reversed(new_a) and sometimes_reversed(r_a) and r_a.source = new_a.destination and r_a.destination = new_a.source then r = 32`
- `else r = calc_r(new_a.source, r_a.source)`

For each transition type the model has the eight parameters described above. Since there are five bucket types there are $5^2 = 25$ transition types. Thus there are $8 \times 25 = 200$ total parameters. To optimize the parameters we converted our fusion code to work from a training file as discussed in Section 5.1 and compiled it into a shared library. We can then call the fusion code passing it a matrix of parameters. The fusion code reads each line in the training file and, using the parameters passed, produces the probability that the example read from the file should produce a "join" decision. Since the training file encodes the "right" answer we can calculate the squared error for this example (the "correct" probability is 0 if the example should have produced a "don't join" decision and 1 otherwise). When all the examples in the training file have been scored the sum of squared errors is returned. Using the constrained optimization routine fmincon in MATLAB[5] we can find the parameter values that minimize the sum of squared errors. With each iteration the evaluation data set was also scored and the sum of squared errors on this data set was tracked. This would allow us to stop the training early if overfitting became an issue.

5.4 Data Mining Techniques

We have also assigned probabilities using data mining techniques. Here the data is prepared as described in Section 5.1 and then passed to a standard machine learning algorithm. The use of machine learning algorithms allowed us to use additional features which may provide better predictive power. As with the heuristic technique, features which indicate the bucket membership of the new alert, the bucket membership of the most recent alert in the scenario, the time between the new alert and the most recent alert in the scenario, and the value of r when the new alert is compared to the most recent alert in the scenario were used. In addition, the following features were used:

- To expand our model from a simple bi-gram model, features which indicate the bucket membership of the 3 most recent alerts in the scenario were used.
- Since attackers often use the same tools or attack types many times features were included to indicate if any previous alerts in the scenario are the exact same type as the current alert and if the most recent alert in the scenario is the exact same type as the new alert.
- Attackers often focus their efforts on a single host . We therefore include a feature that indicates if the destination address of the new alert was the target of any other alerts in the scenario.
- An attack scenario may contain components with spoofed source IP addresses while other components of the attack may use the attackers real source IP. As such a feature whose value is the maximum value of r when the source IP of the new alert is compared to the source IP addresses of all the alerts in the scenario is included.
- Once a machine has been compromised it can be used as the source of further attacks. A feature whose value is the maximum value of r when the source IP of the new alert is compared to the destination IPs of all the alerts in the scenario is therefore included.

- Since RealSecure sometimes reports the source and destination IP addresses reversed features to account for this were included. One indicates if the new alert is "backward". Another indicates if the most recent alert in the scenario is "backward". Both of these features vary between 0 and 1 where 1 indicates that the given alert type is always reported backward by RealSecure and 0 indicates that the given alert type is never reported backward.
- As RealSecure alerts sometimes suffer from an "echo effect" where an alert is sent twice with the source and destination IP addresses of the two alerts exactly switched we included a feature which indicates if the source IP of the new alert matches the destination IP of the most recent alert in the scenario and the destination IP of the new alert matches the source IP of the most recent alert in the scenario.

6 Experimental Results

All experiments were conducted on the DEF CON data described in Section 5.1. A test data set containing 500,717 patterns was used to train the heuristic and data mining algorithms. An evaluation data set containing 250,358 patterns was used to monitor for over-fitting and to choose algorithm parameters. A test data set containing 250,361 patterns, which was not used for any other purpose, was used for final algorithm evaluation. All of the following results reflect performance on this data set.

6.1 Naïve Technique

Predicted

Target	Don't Join	Join	Totals
Don't Join	246,310	10	246,320
Join	246	3,795	4041
Totals	246,556	3,805	250,361

(a)

Predicted

Target	Don't Join	Join	Totals
Don't Join	100.00%	0.00%	100%
Join	6.09%	93.91%	100%

(b)

Table 2: Confusion matrices from Naïve approach on test data set . Two hundred forty six of the 4,041 patterns that should have produced a "join" decision (6.09%) incorrectly produced a "don't join" decision, and ten of the 246,320 patterns that should have produced a "don't join" decision incorrectly produced a "join" decision.

The performance of the naïve approach is presented in Table 2. This simple approach does remarkably well on this data set making only 256 errors on 250,361 patterns. It might seem that this performance is adequate. However we think it is important to improve on this performance for two reasons. First, this data set contains very few attempts at stealth. The capture the flag game rewards speed but no points are awarded for stealth. We expect that more realistic data sets would not be as amenable to such a simple technique. Second, the six percent of attackers who are missed by this simple technique are precisely the individuals we need to worry about. If an attacker can be discovered by our Naïve algorithm that attacker probably can't do much damage anyway. This simple technique does help us group all of the "script kiddie" attacks together which results in significant data reduction, but we would prefer an approach that could discover more sophisticated attacks as well.

6.2 Heuristic Technique

The two hundred parameters used by our heuristic approach were optimized to the data described in Section 5.1. Monitoring the performance on the evaluation data set revealed that overfitting was not a problem; probably

Target \ Predicted	Don't Join	Join	Totals
Don't Join	246,315	5	246,320
Join	452	3,589	4,041
Totals	246,767	3,594	250,361

(a)

Target \ Predicted	Don't Join	Join	Totals
Don't Join	100.00%	0.00%	100%
Join	11.19%	88.81%	100%

(b)

Table 3: Confusion matrices from heuristic approach on test data set. The algorithm tries to match the human decision. For example, 4,041 test examples should have produced a "join" decision. The algorithm correctly produced the "join" decision 3,589 times (88.81% of the time).

because this is not a very complex classifier. After optimizing the parameters needed by our heuristic approach we scored the test data set which contained 250,361 patterns. The results are depicted in Table 3.

The error rates for this algorithm are slightly higher than the error rates for the naïve approach even though the same features used by the naïve algorithm are used by this classifier. The problem is that we have constrained the possible set of decision surfaces this classifier can produce and a decision surface equivalent to that used by the naïve approach is not in possible. Additional features or a different set of possible decision surfaces would need to be considered by the classifier in order to improve its performance enough to be useful. As it is not immediately clear which features would be most useful and how these features should be combined with those we are already considering we will replace this technique with the more powerful data mining algorithms discussed in Section 5.4 in our future work.

Target \ Predicted	Don't Join	Join	Totals
Don't Join	246,316	4	246,320
Join	4	4,037	4,041
Totals	246,320	4,041	250,361

(a)

Target \ Predicted	Don't Join	Join	Totals
Don't Join	100.00%	0.00%	100%
Join	0.01%	99.99%	100%

(b)

Table 4: Confusion matrices from a decision tree on the test data set. The decision tree tries to match the human decision. For example, 4,388 examples should have produced a "don't join" decision. The RBF network produced the correct decision on 3,305 of these (99.98% of the time).

6.3 Data Mining Techniques

Radial basis function networks, multi-layer perceptrons and decision trees were applied to the training data using the features described in Section 5.4. The LNKnet data mining package [12] was used for all experiments. A decision tree with 36 non-terminal nodes produced the best performance. Confusion matrices for this classifier appear in Table 4. The tree was trained using the CART algorithm [3] until no errors were produced on the training set. Nodes were ordered by their effect on the error rate on the training data and then pruned, removing nodes which least affect the error rate first. The validation data set was used to choose the optimal number of nodes.

The error rate of the decision tree is extremely low. It performs significantly better than either of the other two approaches considered, catching the six percent of attack scenario missed by our naïve approach. As this six percent of attackers are probably the ones we should be concerned about this performance increase is significant. We suspect the improved performance is largely a result of the additional features available to the algorithm.

Those features used by the the tree as splitting criteria in non-terminal nodes indicate which features were most useful. These are listed below:

- Of the twenty features which indicate the bucket type of the new and three most recent alerts in the scenario only four were used as splitting criteria. The features used indicated if the new alert was a scan, if the new alert belonged to the stealth bucket, if the most recent alert in the scenario belonged to the stealth bucket, and if the second most recent alert was an escalation alert.
- Our measure r proved to be very useful. Recall from Section 4 that r is a number which indicates the similarity of two IP addresses. We can compare the new alert to the most recent alert in the scenario and produce r values four ways. We can compare the source of the new alert to the source of the most recent alert, the destination of the new alert to the source of the most recent alert, the source of the new alert to the destination of the most recent alert and the destinations of both alerts may be compared. All four features were used by the decision tree. Additionally a feature whose value is the maximum value of r when the source address of the new alert is compared to the source address of each alert in the scenario was used.
- A feature which indicates if the destination of the new alert was the destination of any previous alerts was used.
- The time between the new alert and the most recent alert in the scenario was used as a splitting criteria.
- A feature which indicates if the most recent alert was the same type as the new alert was used as was a feature which indicates if any alert in the scenario was the same type as the new alert.
- A feature which indicates if the new alert might have its source and destination addresses switched was used. This feature varies between 0 and 1 where 1 indicates that this type of alert is always reported reversed and 0.5 indicates that the alert has its addresses reversed about half of the time.

7 System Benefits

The system described provides great benefits to the security analyst. The value of the system was made clear when we began to examine the DEF CON 2000 data. Looking at the 16,250 alerts in this data set was initially overwhelming. It was extremely difficult to figure out what was happening on the network and which alerts represented activity that would be of concern to somebody trying to protect this network. It took the first author about fifteen hours to hand tag this data into scenarios. Once this was done there were only eighty seven scenarios to consider. The scenarios conveyed much more information that the original alert stream. A quick look at many scenarios was sufficient to infer the attackers intentions and tools. For example many scenarios consisted of scans for port 80 followed by a long string of HTTP specific exploits. These individuals were clearly running scripts that tried every HTTP exploit they knew of. The exploits attempted on every host were identical. Thus a network administrator who determined that her hosts were not vulnerable to these exploits could quickly determine that the entire scenario containing several hundred alerts was not a security risk. It might also allow her to preemptively check the security of other web servers once the the beginning of the scenario was seen.

While the benefits of this type of analysis are clear, it is not feasible for administrators to spend fifteen hours analyzing two and a half hours of network data. It is necessary to automate the analysis. The use of the atom model makes it possible to reproduce these decisions in real-time. Our system, using the binary tree discussed in Section 6.3, was able to accurately reproduce the author's analsis of the data in under two seconds!

The system would be even more valuable if it were able to assess the security risk of a scenario and update that risk assessment as the scenario changed. To this end we have begun work on a declarative language that administrators can use to specify risk assignment rules. This language will be the subject of a future publication.

8 Discussion and Summary

We have presented a scenario building approach to IDS fusion. We believe that the scenarios produced by our software convey more information to an analyst than the individual alerts. The scenario building process requires us to be able to produce an estimate that an alert belongs to a scenario. We have examined three approaches to this problem. Our analysis indicates that the data mining approach is superior to either of the other two considered. The data set used for this analysis was fairly simple. This allowed our naïve approach to perform better than we expect it would on more realistic data. Even on this relatively simplistic data the improvement offered by machine learning techniques is significant. We are in the process of obtaining a large amount of data from an operational military network so that we can assess our technique on more realistic traffic.

We intend to improve the system by integrating the security risk assessment language and adding a graphical user interface for viewing the constructed scenarios. Furthermore we intend to create graphical tools to ease the burden of hand tagging scenarios to produce training data.

References

[1] DEF CON 8 conference. Las Vegas, NV, 2000. www.defcon.org.

[2] E. Amoroso. *Intrusion Detection*. Intrusion.Net Books, Sparta, New Jersey, 1999.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Inc., Belmont, California, 1984.

[4] C. Clifton and G. Gengo. Developing custom intrusion detection filters using data mining. In *2000 Military Communications International Symposium*, Los Angeles, CA, October 2000.

[5] T. Coleman, M. A. Branch, and A. Grace. *Optimization Toolbox for Use with MATLAB*. The MathWorks, Inc., 1999.

[6] R. K. Cunningham, R. P. Lippmann, D. Kassay, S. E. Webster, and M. A. Zissman. Host-based bottleneck verification efficiently detects novel computer attacks. In *IEEE Military Communications Conference Proceedings*, Atlantic City, NJ, 1999.

[7] O. M. Dain and R. K. Cunningham. Building scenarios from a heterogeneous alert stream. In *Proceedings of the IEEE SMC Information Assurance Workshop*, West Point, NY, June 2001.

[8] B. S. Feinstein and G. A. Matthews. The Intrusion Detection Exchange Protocol (IDXP), August 2001. www.ietf.org.

[9] P. S. Ford, Y. Rekhter, and H.-W. Braun. Improving the routing and addressing of IP. *IEEE Network*, 7(3):10–15, May 1993.

[10] J. Haines, L. Rossey, and R. Lippmann. Extending the 1999 evaluation. In *DISCEX Proceedings*, June 2001.

[11] Internet Security Systems. RealSecure console user guide. Atlanta, GA, 2000. www.iss.net.

[12] R. P. Lippman, L. Kukolich, et al. LNKnet: Neural network, machine learning, and statistical software for pattern classification. *Lincoln Laboratory Journal*, 6(2):249–268, 1993.

[13] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[14] J. Schwartz. Hacker defaces pro-israel web site as the mideast conflict expands into cyberspace. *The New York Times*, November 3, 2000.

[15] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS-a graph based intrusion detection system for large networks. In *19th National Information Systems Security Conference Proceedings*, pages 361–370, October 1996.

[16] A. Valdes and K. Skinner. An approach to sensor correlation. In *Recent Advances in Intrusion Detection (RAID 2000)*, Toulouse, France, October 2000.

[17] S. Vasile. Automated intrusion detection environment (AIDE). In *Joint Aerospace Weapon Systems Support, Sensors, and Simulation Proceedings*, June 2000. http://www.adpansia.org/events/brochure/092/proceed.htm.

[18] B. J. Wood and R. A. Duggan. Red teaming of advanced information assurance concepts. In *DISCEX 2000*, Hilton Head, South Carolina, January 2000.