

# Building Scenarios from a Heterogeneous Alert Stream

Oliver M. Dain and Robert K. Cunningham

*Abstract*— We describe a realtime algorithm for combining the alerts produced by several heterogeneous intrusion detection sensors into scenarios. Each scenario represents a sequence of actions performed by a single actor or organization. Our algorithm, which is probabilistic in nature, can determine the scenario membership of a new alert in time proportional to the number of candidate scenarios. It is capable of finding scenarios even if an intruder has used stealthy attack methods such as forged source IP addresses or long latencies between attack components.

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

## I. INTRODUCTION

Security conscious organizations are beginning to use multiple intrusion detections systems to protect their computer networks, with the result that many related portions of a single attack are detected by different sensors. While this approach may improve attack detection rates, it also tends to increase the false alarm rate. Additionally, multiple sensors may increase the work load of intrusion detection analysts as there are now more components to be monitored. Fusion systems aim to alleviate these problems by combining the alerts from multiple heterogeneous systems and displaying the results at a single easily monitored location.

In this paper we propose a real time 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 robust 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. Our algorithm is probabilistic in nature, running 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

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.

Oliver M. Dain and Robert K. Cunningham are with Massachusetts Institute of Technology Lincoln Laboratory, 244 Wood Street, Lexington, MA 02420

components.

As Amoroso points out, “correlation processing has not been the focus of much research in computer and network security”[1]. 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[2]. 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[3]. Valdes and Skinner have studied correlation and scenario building from probabilistic detectors such as their EMERALD eBayes system[4]. 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.

## II. FUSION APPROACH

Our system develops scenarios as they occur on the network. 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 newly arrived alert with the previous alerts. Unfortunately this quickly becomes computationally intractable. In fact, if we have  $n$  alerts and we see a new alert we must consider  $2^n$  combinations of alerts. To see this imagine we have only 3 alerts in memory when we see the new alert. We will call the 3 existing alerts A, B and C, and the new alert E. There are  $2^3 = 8$  ways we can arrange these alerts into scenarios. These possibilities are enumerated in Table I. If we now assume that there were 4 alerts in memory, A, B, C, and D, when our new alert, E, was produced we see that the number of possibilities has doubled to 16. Looking at Table II it is clear that each new alert in memory doubles the number of possibilities that must be considered. Thus the growth is exponential. Clearly it is not possible to consider all such arrangements in a realtime system.

Our solution is to use an “atom model”. Each time we receive a new alert from a sensor we compare it to each scenario. For each existing scenario we calculate the probability

AE	ABE	ABCE	ACE
BE	BCE	CE	E

TABLE I  
POSSIBLE ARRANGEMENTS OF 4 ALERTS

AE	ABE	ABCE	ACE
BE	BCE	CE	E
ADE	ABDE	ABCDE	ACDE
BDE	BCDE	CDE	DE

TABLE II  
POSSIBLE ARRANGEMENTS OF 5 ALERTS

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. This simplification reduces the computational complexity from super-exponential in the number of alerts to linear in the number of candidate scenarios although it may cause errors in the assembled scenarios. For example, three alerts from a scenario might be necessary to provide enough evidence for us to join them together. Since we only consider scenario formation one alert at a time we would never group such alerts.

### III. ARCHITECTURE

The fusion system consists of three distinct components: the intrusion detection systems (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, the RealSecure server sensor, and MIT Lincoln Laboratory’s Battlefield Intrusion Detection System (BIDS), a host based system capable of finding known and novel attacks[5]. 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 and the scenario membership of any new alerts is determined. Scenario membership information is then written back to the database. Although our research has thus far focused on the three sensors mentioned, the algorithm would work with any sensor that could send its output to this database.

### IV. DEFINITIONS

RealSecure is capable of identifying over 250 different attack types. Because attackers can use multiple techniques to

accomplish the same result[6] each of these alerts is assigned to one of 5 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 2 addresses. Thus  $r = 32$  when the 2 IP addresses are identical, and  $r = 0$  when there is no way the 2 addresses could be on the same subnet (see Table III). Due to the prevalence of Classless Inter-Domain Routing (CIDR) we allow 2 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 2 class B addresses 130.114.5.20 and 130.115.100.8 differ in their 2nd 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 2 addresses an  $r$  value of 15 since the first 15 bits in both addresses are the same.

### V. PROBABILITY ASSIGNMENT ALGORITHM

Thus far we have specified how scenarios are formed given a measure of the probability that an event belongs to a given scenario. This section describes how these probability measures are determined. We use 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 3 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 1).

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

	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 III

$r$  VALUE CALCULATION FOR ADDRESSES 172.16.112.20 AND 172.16.112.40

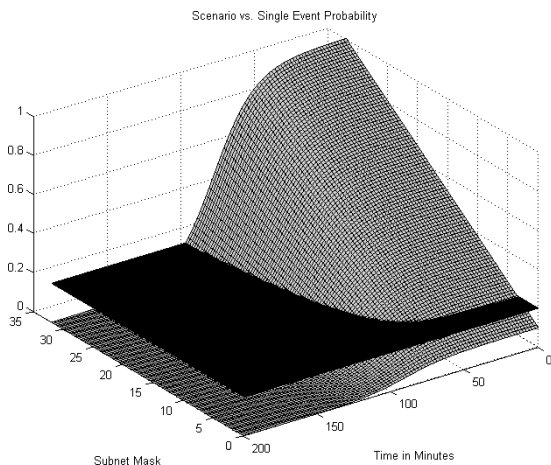


Fig. 1. Decision surface for a single transition type. The height of the curve is the probability that 2 alerts in the given time span with the given  $r$  value belong in the same scenario. The black plane is the threshold below which the new alert would not join the scenario and would start a new scenario.

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  $\sigma_{ij}(\Delta t) = 1 / (1 + e^{\alpha + \beta \Delta t})$ . The shape of the function (determined by the values of  $\alpha$  and  $\beta$ ) 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 2 IP addresses (as defined in section IV).  $R_{ij}(r)$  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  $i = j = \text{DOS}$ .

For each transition type the model has the 8 parameters described above. Since there are 5 bucket types there are  $5^2 = 25$  transition types. Thus there are  $8 \times 25 = 200$  total parameters. To optimize these we used data as described in Section VI.

## VI. DATA SOURCES AND USE

Given a set of network alerts and the corresponding scenarios we could set the probability estimate parameters 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[7] 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. While this is a labor intensive task it only needs to be performed once.

We used tcpdump data from the DEF CON 8 hacker conference’s “capture the flag” game for this task[8]. Every year at the DEF CON conference participants play “capture the flag”. There are 2 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. Clearly that is not

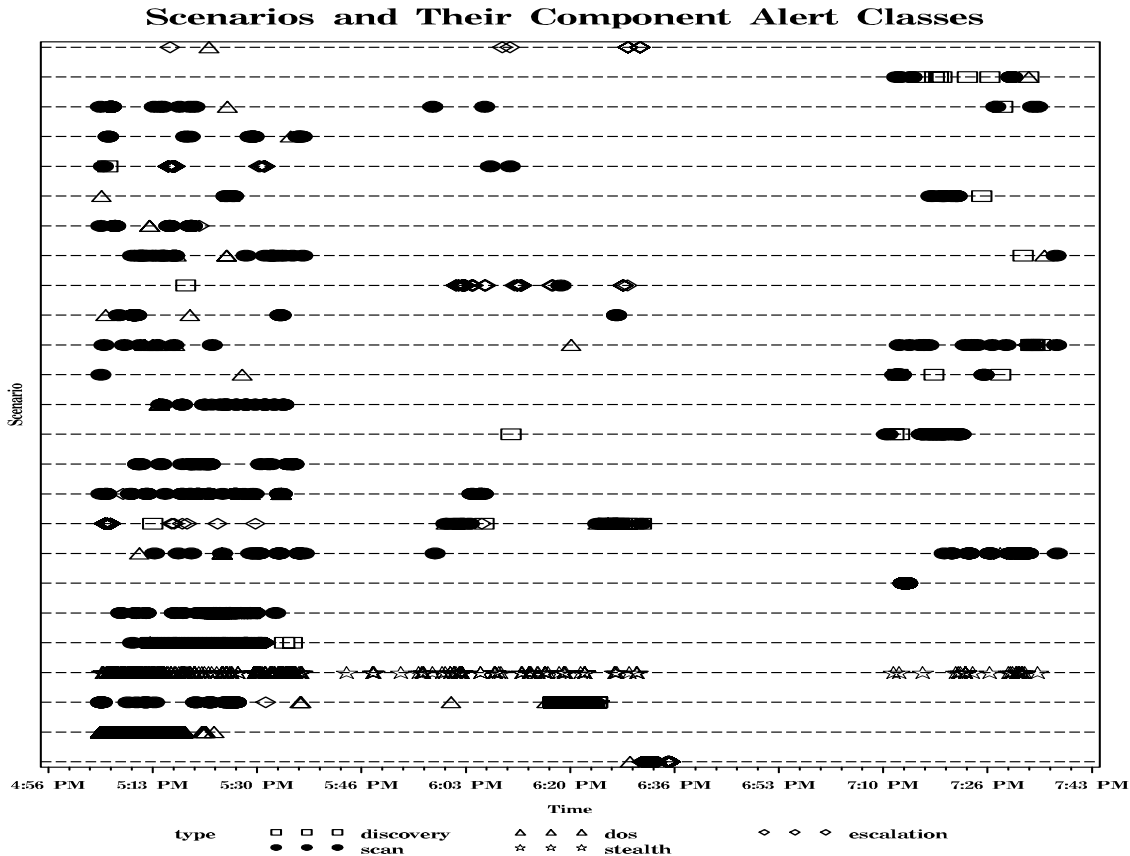


Fig. 2. Times of Selected Scenarios. The number of alerts in the scenario is indicated on the vertical axis. For example, there were 1,368 alerts in the scenario at the bottom of the chart. The first alert in this scenario occurred at about 5:15 pm and the last at about 5:40 pm.

the case here.

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.

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 tagged into eighty nine 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.

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 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
      - ▷ Positive training example
    - \* ELSE
      - ▷ Negative training example
    - \* END IF
      - \* 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 [9] 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 3 files (50% for training, 25% for evaluation, and 25% for testing) while preserving the prior probabilities.

We converted our fusion code to work from a training file 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[10] we can find the parameter values that minimize the sum of squared errors.

## VII. EXPERIMENTAL RESULTS

After optimization we scored the test data set which contained 250,361 patterns. The results are depicted in tables IV and V.

The algorithm does an excellent job of matching human performance, but there is still room for improvement. Note that with 16,250 alerts and an error rate of 11.19% we will still incorrectly fail to join an alert to a scenario approximately 1,818 times. Once we have made one mistake the scenario is “corrupted” and we are more likely to make further mistakes. Nevertheless, when we used these parameters to produce scenar-

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

TABLE IV

CONFUSION MATRIX FROM TEST DATA SET. THE ALGORITHM TRIES TO MATCH THE HUMAN DECISION. FOR EXAMPLE, 246,320 TRAINING EXAMPLES SHOULD HAVE PRODUCED A “DON’T JOIN” DECISION. THE ALGORITHM PRODUCED THE CORRECT DECISION ON 246,315 OF THESE.

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

TABLE V

CONFUSION MATRIX PERCENTAGES. THE ALGORITHM CORRECTLY PRODUCED A “JOIN” DECISION 88.81% OF THE TIME. THE “DON’T JOIN” DECISION WAS CORRECTLY PRODUCED ALMOST 100% OF THE TIME.

ios from DEF CON data the results were quite good. There were some instances where alerts that we would have joined into scenarios were not combined and some instances where alerts were incorrectly put into scenarios, but overall results were good enough to be of value to a situation analyst.

## VIII. DISCUSSION AND SUMMARY

We have presented a probabilistic approach for fusing alerts from multiple intrusion detection sensors into scenarios. Our results indicate that by combining a few simple features we can produce decisions that closely resemble those made by a human analyst.

Despite our good results we believe that more work could be done in this area. We have begun to explore other approaches to the probability estimation problem including traditional data mining techniques. This would allow us to use additional features which may provide better predictive capability. We hope to further reduce the amount of incorrect scenario assignments. Once an alert has been assigned to the wrong scenario this scenario is “corrupted”. The system is then more likely to make errors in the future. We intend to investigate ways of mitigating these problems.

## REFERENCES

- [1] Edward Amoroso, *Intrusion Detection*, Intrusion.Net Books, Sparta, New Jersey, 1999.
- [2] Sandra Vasile, “Automated intrusion detection environment (AIDE),” in *Joint Aerospace Weapon Systems Sup-*

- port, *Sensors, and Simulation Proceedings*, June 2000, <http://www.adpansia.org/events/brochure/092/proceed.htm>.
- [3] 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*, October 1996, pp. 361–370.
  - [4] Alfonso Valdes and Keith Skinner, "An approach to sensor correlation," in *Recent Advances in Intrusion Detection (RAID 2000)*, Toulouse, France, October 2000.
  - [5] Robert K. Cunningham, Richard P. Lippmann, David Kassay, Seth E. Webster, and Marc A. Zissman, "Host-based bottleneck verification efficiently detects novel computer attacks," in *IEEE Military Communications Conference Proceedings*, Atlantic City, NJ, 1999.
  - [6] Bradley J. Wood and Ruth A. Duggan, "Red teaming of advanced information assurance concepts," in *DISCEX 2000*, Hilton Head, South Carolina, January 2000.
  - [7] Joshua Haines, Lee Rossey, and Rich Lippmann, "Extending the 1999 evaluation," in *DISCEX Proceedings*, June 2001.
  - [8] "DEF CON 8 conference," Las Vegas, NV, 2000, [www.defcon.org](http://www.defcon.org).
  - [9] Tom M. Mitchell, *Machine Learning*, McGraw-Hill, 1997.
  - [10] Thomas Coleman, Mary Ann Branch, and Andrew Grace, *Optimization Toolbox for Use with MATLAB*, The MathWorks, Inc., 1999.