Attack Signature Matching and Discovery in Systems Employing Heterogeneous IDS

Nathan Carey, George Mohay and Andrew Clark
Queensland University of Technology
nl_carey@yahoo.com.au, g.mohay@qut.edu.au, a.clark@qut.edu.au

Abstract

Over the past decade, Intrusion Detection Systems (IDS) have improved steadily in the efficiency and effectiveness with which they detect intrusive activity. This is particularly true with signature-based IDS due to progress with intrusion analysis and intrusion signature specification. At the same time system complexity, overall numbers of bugs and security vulnerabilities have been on the increase. This has led to the recognition that in order to operate over the entire attack space, multiple heterogeneous IDS must be used, which need to interoperate with one another, and possibly also with other components of system security. This paper describes our research into developing algorithms for attack signature matching for detecting multi-stage attacks manifested by alerts from heterogeneous IDS. It describes also the testing and preliminary results of that research, and the administrator interface used to analyze the alerts produced by the tests and the results of signature matching.

1. Introduction - Related Work and Motivation

Intrusion Detection Systems (IDS) have evolved significantly over the past two decades since their inception in the early 1980s [1]. The simple IDS of the early days were based either upon the use of simple rule–based logic to detect very specific patterns of intrusive behaviour or upon historical activity profiles to confirm legitimate behaviour. In contrast, we now have IDS which use data-mining and machine-learning techniques [2] for the dynamic compilation of new intrusion signatures and which allow for quite general expressions of what may constitute intrusive behaviour. Other modern IDS may use a mixture of sophisticated statistical and forecasting techniques to predict what is legitimate activity.

Alert correlation is one of the central issues currently challenging intrusion detection research. Recent work by Morin *et al* [3] provides a formal model - *M2D2* - for alarm or alert correlation. The M2D2 model comes at a time when researchers and implementers are grappling with the need for reduction of the number of false positives, and indeed that is one of the focal points emphasised in the paper by Morin et al. Our work is related to previous work by Vigna et. al. [4], Doyle et. al. [5], Valdes and Skinner [6], Cuppens [7] and Debar and Wespi [8]. None of this work, including ours, attempts to incorporate the four dimensions of M2D2.

The work by Vigna et. al. uses the STAT model to provide a framework for dynamic configurability of IDS sensors and uses Java to aid in portability. The work by Doyle et. al. with MAITA reflects similar goals, but utilises a more complex architecture to support interoperability and uses trend templates for the specification of chains of events as opposed to the STATL attack scenario language used in the STAT work. Valdes and Skinner use a probabilistic approach to perform correlation of information from multiple sensors, and focus on the concept of 'threads' to maintain links between alerts. Debar and Wespi [8] use features in the Tivoli Enterprise Console to perform aggregation and correlation of alerts, and focus on the abilities of a management system to reduce the amount of data presented to an administrator. Recent work by Cuppens ([7], [9]) which focuses on commodity IDS, and uses a central database for alert aggregation and analysis is the most similar to our own approach. Cuppens uses a Prolog database and static signatures for attack detection, together with stored procedures to perform aggregation of alerts to reduce redundancy. The signature set is defined using the LAMBDA syntax, which enables the specification of very complex event relationships.

While each of these systems has its own special objectives, they have in common that in each case use is made of a central alert store that captures alerts from multiple sensors in order to assist the overall intrusion detection process. Some of the systems use IDMEF for communication, and some utilise purpose built IDS platforms, rather than relying on commodity IDS for alert data. While all perform some sort of analysis on the data, the analysis mechanisms used are different.

Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 2003)
1063-9527/03 $17.00 © 2003 IEEE

Our work too is pitched at the correlation of alerts emanating from heterogeneous IDS using a central alert store that captures alerts from multiple sensors [10]. In contrast however, it adopts a bottom up approach to the problem and sets out to develop a simple framework built out of commodity and free software in order to produce practical alert correlation across heterogenous commodity IDS. There are some other, more minor, differences too with regard to implementation, such as not using stored procedures, the nature of the interaction between alert correlation and signatures, and the relatively simple format and implementation of the multi-step attack signatures themselves. The two major goals of our work have been to design and implement:

1. *signature-based attack detection* which matches attack signatures with an alert stream comprising alerts from heterogeneous IDS and uses off the shelf software and a minimum of specialised components to do so, and
2. *an experimental platform* to allow experimentation with new signature matching algorithms and mechanisms.

To achieve these goals, the work has involved the development of an *alert or cluster analysis capability* to facilitate the analysis and correlation of alerts from heterogeneous IDS, and a *visualization capability* to complement and assist the alert analysis capability and to facilitate signature discovery. The alert or cluster analysis was discussed in our paper [10], the visualization features are new.

Interoperability is a crucial part of alert analysis, providing a common platform for alerts arriving from heterogeneous IDS and possibly other system security components such as firewalls and host logs. We employ IDMEF for this purpose and this allows us to exploit the use of heterogeneous IDS to provide identification and notification of a wider range of alerts than is possible with homogeneous IDS and thus perform cross-sensor signature-based attack detection.

While the ultimate goal of IDS and inter-IDS alert correlation is to provide real-time warning or even reaction to perceived alerts and attacks, there are outstanding issues to be solved relating to IDS interoperability and attack recognition which are most easily and productively tackled in an offline or post hoc context. Our focus has therefore been on the development of an offline platform for studying attack signature matching and discovery techniques. This enables us to concentrate on experimentation with signature specification and matching techniques and avoid being distracted unnecessarily by real-time performance considerations. For the future, this will enable us to study outcomes such as how the error rate of false negatives and false positives varies with circumstances.

The system also includes a security administrator interface with access to both the signature detection and the alert analysis and visualization functions. The interface also provides access to a number of system configuration parameters thus providing the basis for studying the different outcomes (e.g., error rates) resulting from different experimental scenarios.

The paper is structured as follows. This section has reviewed related work, including the authors' previous work upon which the work described here is founded, and provides also the rationale for the work described in the paper. Section 2 examines the detailed design of our attack signature specification and the detection algorithms we have developed. Section 3 discusses an example of a specific attack signature that illustrates our approach while Section 4 describes the administrator interface which provides access to both the signature detection and alert analysis capabilities of our system, together with visualization features. Section 5 on *Testing and Evaluation* discusses the attack set used to test and evaluate our system and presents the preliminary results achieved with respect to that attack set. This is followed by the conclusions and summary of further work in Section 6.

2. Signature Specification and Matching - Detailed Design

Our earlier paper [10] focused on IDS interoperability, system architecture and alert cluster analysis, the subsequent work described in this paper relates to the signature specification and matching techniques that we have developed and details of these appear immediately below.

2.1. Signature Specification

The correlation of alerts whether from the one IDS over time or across several heterogeneous IDS provides two significant benefits:

- *signature-based attack detection* - it allows known multi-step attacks to be detected by matching alert streams against the set of known attack signatures, and
- *attack and signature discovery* - it allows for the discovery of new attacks by identifying hitherto unseen alert relationships, and the discovery and incorporation of new attack signatures into the set of known attack signatures.

The alert correlation work most closely related to ours is that by Cuppens and Miege [7], their work is however more ambitious than ours and has somewhat different objectives. It uses a sophisticated model based around first order logic for expressing signatures for attacks and for representing alerts, and has the ultimate objective of anticipating the future steps of an attacker. The signature set is defined using the LAMBDA syntax, which enables the specification of very complex event inter-relationships.

Our goal of producing a simple yet functional system for use in the context of heterogeneous IDS flows through into the specification and usage of signatures themselves. Attack specification languages such as LAMBDA [11] and CISL and S-expressions [12] provide the ability to define very complex relationships between events themselves.

However they do not satisfy the goal of a simple method of specification. We on the other hand set out simply to recognize known multi-step attacks from an alert stream containing alerts from heterogeneous IDS, and to be able to learn new attacks to include into the signature set and to do both of these with simple, off-the-shelf components. The signature specification notation has thus been kept intentionally minimal.

Our attack signatures are couched in terms of *alert elements* which consist of the following two alert characteristics:

1. *alert name* (the unique name of the alert), and
2. *alert time* (the time the alert was detected).

In addition, we include with each alert element some additional attributes as follows:

3. *timeout* (inter-alert timeout) – this allows the expiry of signature (and hence attack) matching based upon expiry of the expected maximum time between successive alerts,
4. *optional/mandatory components* - the alerts specified within an attack signature are by default **mandatory**, and the absence of a mandatory alert within the timeout period will flag absence of the attack and expire the matching activity. An alert may however be marked as **optional**, and absence of an optional alert will not of itself lead to the conclusion that the attack is absent. An optional alert has an alert weight associated with it and a necessary condition for the signature to *fire* (i.e., for the signature as a whole to have been matched by alerts from the alert stream, thus signifying that an attack has been detected) is that the sum of the weights of matched optional alerts equals or exceeds a specified optional alert weight threshold associated with the signature.
5. *alert repeat factor* - signatures include the facility for associating an alert repeat factor with an alert element, this is the threshold number of repeats (default of one) that needs to be achieved for an alert to satisfy the alert element. This does not affect the semantics of alert element weights.

The signatures themselves have five attributes that are pivotal to the signature specification and matching algorithm:

i. *IP mask* – this is the network mask (denoted by a number of bits) which determines a valid set of targets for this particular signature,
ii. *signature priority* - the signature priority governs the order in which alerts are matched against candidate attack signatures. Given the use of 'single-fit' of an alert from the alert stream to candidate signatures (see below), the priority order in which candidate signatures are matched against the alert stream is critical;
iii. *overall signature timeout* – this allows the expiry of signature (and hence attack) matching based upon expiry of the total expected maximum time for an attack to complete (to be effective this must be less than the sum of the previously mentioned individual alert timeouts),
iv. *threshold weight* – this is the threshold proportion of optional alerts (see below) that needs to be satisfied for the signature match to have been be deemed successful and the signature to have fired. This is achieved by associating a weight with each optional alert forming part of the signature and setting a minimum total or threshold weight that needs to be satisfied; and
v. *action on firing* – typically this includes noting to the administrator's console that the signature has fired and may include re-insertion of a named *synthetic alert* into the alert stream.

Optional elements of an Attack Signature refer to an alert which does not necessarily have to be matched for that signature to fire (in contrast to *mandatory* alert elements). Such elements have an associated weighting. When such an optional alert element *is* matched, then its weighting is accumulated into the 'optional_alert_accumulated_weight' (initially zero) for that active signature. For a signature to fire, all mandatory alert elements must have been matched, and the value of its 'optional_alert_accumulated_weight' must equal or exceed the threshold firing weight. The latter has a default of zero, which assumes that the signature contains no optional alert elements. Arbitrary threshold and element values enable the specification of situations such as 'two out of three alerts' or 'alert xxxx plus three other alerts'.

Note that the mandatory/optional, repeat factor and inter-alert timeouts are all optional for a signature. For ease of use, a signature can be initially specified with only the signature name, basic timeout, and using a group signature in order to provide a simple method of selecting all of the component alerts within a given time period.

2.2. Signature Composition

An attack signature may be one of two basic types:

- a *sequence* of alerts, or
- a *group* of alerts.

An attack signature consisting of an alert sequence is constructed as an ordered list, so that the signature is matched or fired when the final and all previous mandatory alert elements are satisfied in the order listed and when the threshold weight of optional alerts has been reached. An attack signature consisting of an alert group is constructed as a set such that the signature is matched or

fired when all the alert element items are matched, irrespective of order, and when the threshold weight of optional alerts has been reached. More complex attack signatures are composed by combining these two basic types of signatures by the concept of *synthetic alerts*, a concept which appears elsewhere in previous works (e.g., Cuppens [7]). When a basic attack signature type is fired, it can have two outcomes:

- notification of the occurrence of an attack, and/or
- insertion of a *synthetic alert* into the alert stream.

Such a synthetic alert may itself then be matched with higher order signatures allowing the security administrator to construct attack signatures of arbitrary complexity.

2.3. Signature Matching Algorithm

Before considering the more significant design details of our signature-matching algorithm, we present the following definitions:

- *the active signature set* - this is the set of all partially matched signatures, it is the set of all those signatures which have matched at least one alert and not yet timed out or completed.
- *the open signature set* - this is initialised at start up to be the set of all attack signatures, it then grows to include the active signature set
- *the candidate alert set* - this is the set of alerts comprised of:
  i. the first unmatched mandatory alert plus every immediately preceding unmatched optional alert in the case of a sequence signature, for all sequence signatures in the open signature set; and
  ii. every unmatched alert in the group in the case of a group signature, for all group signatures in the open signature set
- *the per-alert candidate signature list* - this is the priority-ordered list of open signatures 'waiting' for the alert. Signatures will typically appear in several such lists:
  - in the case of sequence signatures, the signature will appear in the signature list for its next unmatched mandatory alert, and in the signature lists of each unmatched optional alert preceding its next unmatched mandatory alert
  - in the case of a group alert, the signature will appear in the signature list for each of its unmatched alerts

We now consider the details of our signature-matching algorithm.

2.3.1 Single-fit Signature-matching We have implemented a 'single-fit' strategy for our matching algorithm, as opposed to a 'multi-fit' strategy. With a single-fit strategy of matching alerts to signatures, the current alert from the alert stream is matched against the candidate signature list associated with that alert, in particular with the first signature in the list i.e., the highest priority signature in the list, and this signature then 'consumes' that alert. 'Multi-fit' on the other hand allows the current alert to contribute provisionally to multiple eligible signatures simultaneously until the first of these fires. At that time, the alert's contribution to the other signatures is withdrawn and they are rolled back to their previous status. This roll-back allows multiple threads of execution to run in parallel without affecting any other thread.

Multi-fit has some advantages, the main one being that it leaves open the possibility of matches with all possible signatures until one fires and does so without necessarily burdening the administrator with details of signature priority (though priority may still be used). At the same time it has the disadvantage of increasing the complexity of the matching algorithm, where multiple roll-back strategies may be required to produce a 'best-fit' for a given stream of alerts.

We have opted to use the single-fit algorithm and repeated experiments (given ours is currently an off-line platform) to perform this role, rather than focus on the algorithm for matching in exhaustive detail.

Successive experiments have enabled us to adjust the relative priority ordering of signatures used in the single-fit matching process and thus identify the alternative signature matching outcomes that can result with different signature priorities or ordering of alerts in the alert stream. Single-fit has also enabled direct analysis on how basic pattern matching is insufficient to describe the relationships between alerts in complex attacks when attack state, multiple attacks, alert overlap and out-of-sequence alerts are considered.

As single fit is, at it's heart, basic sequential pattern matching, additional logic needed to be developed to enable the specification of priority, mandatory/optional, weighting and IP address filtering. These features were considered required functions to properly describe many of the attacks that can occur over the network. Indeed, this area of analysis of state-based attack detection mechanisms is considered an undeveloped area in IDS research. Implementation of a multi-fit strategy and comparisons between it and the single fit algorithm in practice is considered for future work.

2.3.2 Duplicate Attacks Our system detects overlapped, duplicated attacks of the one identical type as the open signature set is initialised at start up to include all attack signatures and this is augmented with additional partially matched signatures as and when a new partial match occurs. This feature is configurable at set up time to be switched on or off for the purposes of experimentation and

so that it is possible for the administrator to forego the feature if desired. In some cases, detecting overlapped identical attacks is not necessarily more valuable than detecting just the one instance of the attack.

2.3.3 Slow Attacks, Timeouts and Alert Propagation The system prevents knowledge of the timeouts used by signatures from being exploited by an attacker. That is, an attacker could attempt to subvert the system essentially by launching a slow attack or a number of such attacks. We guard against this situation by deploying two countermeasures:

- *detection of overlapped, duplicate attacks* - the first is the configurable detection of overlapped, duplicate attacks as discussed above,
- *alert propagation* - the second is our treatment of signature priority and the propagation of alerts between signatures. As mentioned previously, matching of the current alert to the open signature set takes place through the candidate signature list for that alert. There is a configurable option for that candidate signature list to include not just the bona fide candidate signatures for that alert, let us call it $alert_{current}$, but also all those active signatures that have recently matched an earlier duplicate of the same alert, say $alert_{previous}$. Any active signature, whose most recent previous alert match had been to such a duplicate $alert_{previous}$, will also appear in the candidate alert list.

The significance of the last point is that a signature that had previously matched $alert_{previous}$, is now expecting a completely different alert *different-alert*, but is about to time out due to expiry of the inter-alert time out mechanism will be refreshed with the new alert $alert_{current}$, that is the new alert $alert_{current}$ will replace the previous older one $alert_{tprevious}$, thus preventing a possible signature time out. This of course may retrospectively contradict the previous inter-alert time out constraint but has the benefit of guarding against detection avoidance through manipulation of signature time out. The replaced alert is then propagated to the next candidate signature in the list and the process continues recursively. Since the list is ordered by signature priority, we ensure that the highest priority open signatures are given maximum chance of firing. The three factors which govern how alerts are matched against signatures are, in order:

1. the alert type,
2. the priority order of the signatures in the candidate signature list, and
3. IP masking constraints associated with the alert element.

3. Signature Specification and Matching – An Example

We provide here an example of an attack signature in order to demonstrate our approach. This example is used in the experiments described in Section 5. This particular attack is the same as the one used in work by Cuppens [7]. In fact, this particular attack requires four separate signatures. Three of these signatures generate synthetic alerts, as described above, which are matched by the fourth signature.

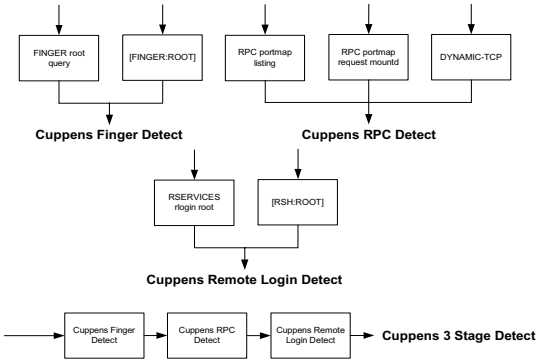

Figure 1: Cuppens Attack Signature

The two IDS used in our experiments, Snort and Dragon, produce different alerts when detecting this attack. These alerts can be related to each other based on their content. Snort produces the following specific pattern of alerts:

- *FINGER root query*;
- two occurrences of *RPC portmap listing*;
- *RPC portmap request mountd*; and
- *RSERVICES rlogin root*.

While Dragon produces the following alerts:

- *FINGER:ROOT*;
- three *DYNAMIC-TCP* alerts; and
- a *RSH:ROOT* alert.

Figure 1 shows the structure of each of the four signatures used in detecting this attack. The "FINGER root" alert generated by Snort and the "FINGER:ROOT" alert generated by Dragon are equivalent. These alerts are grouped in the first signature, which we label "CUPPENS Finger Detect". The second signature, labelled "CUPPENS RPC Detect", is made up of the "RPC portmap listing" and "RPC portmap request mountd" alerts generated by Snort and also includes the "DYNAMIC-TCP" alert generated by Dragon. Multiple instances of the "RPC portmap listing" and "DYNAMIC-TCP" alerts are detected by setting the alert repeat factor to a value greater than one. The "RSERVICES rlogin root" (Snort) and "RSH:ROOT" (Dragon) together form the third signature, labelled "CUPPENS Remote Login Detect". Finally, the fourth signature, which represents the entire Cuppens attack, is comprised of the first three synthetic alerts.

In this particular case, the format of the entry in the file used to store signatures is in the form of four signatures, shown here in basic form (no individual alert replication, timing or weighting information) using a comma-delimited format:

1. CuppenFinger, group, newalert, CUPPENS Finger Detect, 32, 5, 0, 2, FINGER root query, [FINGER:ROOT]
2. CuppenRemoteLogin, sequence, newalert, CUPPENS Remote Login Detect, 32, 20, 0, 2, RSERVICES rlogin root, [RSH:ROOT]
3. CuppenRPC, group, newalert, CUPPENS RPC Detect, 32, 30, 0, 3, RPC portmap listing, RPC portmap request mountd, [DYNAMIC-TCP]
4. Cuppen3Stage, sequence, newalert, CUPPENS 3 Stage Detect, 32, 20, 0, 3, CUPPENS Finger Detect, CUPPENS RPC Detect, CUPPENS Remote Login Detect

We note for the purposes of illustration using the first signature as an example, that CuppenFinger, is a group signature spawning a 'new alert' of 'CUPPENS Finger Detect', it has a netmask of 32, a timeout of 5 seconds, a weighting threshold of zero (i.e. not used) and a priority of 2 out of 5, with 1 being the highest.

The first three signatures above correspond to the first three signatures in Figure 1(the top two signatures, and the middle signature), and the fourth corresponds to the bottom signature in Figure 1 which combines the first three signature synthetic alerts into a second-level signature. This is achieved by using the alerts produced by the 'new alert' option in each of the first three signatures.

4. The Administrator Interface

The administrator interface (Figure 2) provides the security administrator with access to both the signature detection and alert visualization and analysis functions.

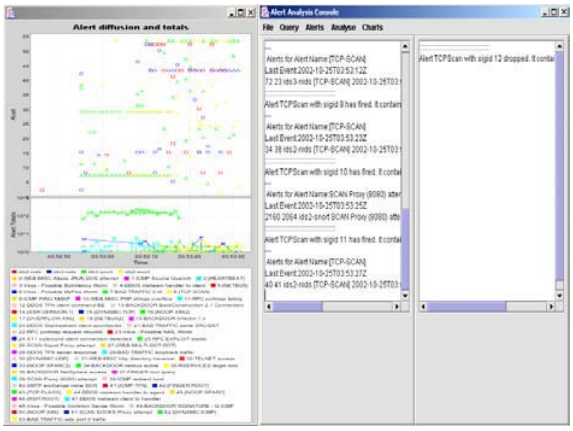

Figure 2: Administrator Interface Screenshot

The Administrator Interface was developed to provide an interface to the following features:

1. *signature-based attack detection* - execute signature-based attack detection on the alert stream from the database with options:
   - display the list of signatures that fired, and
   - display the list of signatures that expired before firing
2. *an experimental platform* to allow experimentation with new signature matching algorithms and mechanisms viz., a system configuration capability to configure a number of system parameters including signature priorities, duplicate attack detection, alert propagation
3. *an alert or cluster analysis capability* to facilitate the analysis and correlation of alerts from heterogeneous IDS
4. *a visualization capability* to complement and facilitate the above alert analysis capability in order to facilitate signature discovery; in particular to provide a summary graphing analysis of alerts with information on IDS, time, and alert name, and provide access to detailed information across a range of attributes down to the individual alert level.

Feature 1 provides a display of the outcome of the attack signature detection that was discussed earlier in Section 2 (right panel in Figure 2). This lists the names of the attacks that were detected and in each case allows the administrator to display the signature itself and the particular alerts that contributed to it. It also displays 'incomplete' attacks and is included in order to assist the administrator in developing better or more complete attack signatures for inclusion in the attack signature set. Feature 2 has been discussed in earlier sections of the paper. Feature 3, together with our architecture for interoperability, is discussed in our earlier paper [10].

We focus now on the remaining feature, feature 4, the provision of visualization and summary graphing information with access to detailed alert information (this can be seen in the left side of Figure 2). The intention of this feature is to provide the security administrator with the opportunity to examine alert traffic in general with a view to identifying unexpected traffic not diagnosed by the signature detection and to identify and evolve new attack signatures for inclusion in the attack signature set. The display presents the following information:

- time (x-axis),
- sensor (icon),
- alert type (top graph, y-axis), and
- alert number (bottom graph, y-axis).

This information is presented in two co-located graphs, one showing alert type, sensor and time (the upper graph), and the other showing the numbers of alerts within the discrete time periods (the lower graph). Placing the mouse over an item in the upper graph shows the specific name, IDS and time of the alert(s) that are indicated. Clicking on

the item brings up detailed alert information. The lower graph uses a logarithmic scale on the y-axis. The display becomes particularly useful when a DoS or IDS overloading attack occurs, as it allows the administrator to see patterns in the alert data, and elements within the graph can be clicked on to gather information about the alerts which correspond to each item in the graph. Visualization of the relationship between signatures and surrounding alerts is also particularly useful.

When looking at the alerts that are attributed to a signature, it is possible to view the other alerts that have occurred at about the same time, and a graph can show this information over a large time period, without becoming overly complex.

There are certain attacks that cannot currently be specified in our signatures, while they can be observed in the graphs provided. The graphing functionality can be used to identify trends and series of alerts that do not correspond to signatures. For example, if a large number of alerts is observed on a single host, one might assume that an attack is taking place. By clicking on individual items in the upper graph, the administrator can drill down and see the alerts within that item, and look at the features of those alerts. Certain attacks such as DoS or IDS alert spoofing attacks can be observed visually by looking at the amount, type and IDS of alerts that have been stored.

The general advantage of the prototype GUI over other approaches is that the user is never confronted with any sort of flat text file containing alerts. Alerts are shown either in the graphs provided (where the user can identify each alert type by the legend at the bottom of the screen) or by investigation of the alerts that have been matched to signatures. Alerts with high occurrences as well as those with low occurrences are easily identified and can be investigated by clicking on the graph. The graph also identifies which alerts are produced by which sensor, which is useful for tracking trends either of attacks moving within the network, attacks that are only detected by a particular IDS, or alerts that may constitute false alarms. In practice, this makes the identification and analysis of alert streams with large numbers of alerts much quicker than some other approaches.

5. Testing and Evaluation

In order to properly gauge the usefulness of our system, we experimented with various attacks and profiled the operation of our system on a set of attack data. The testing procedure was developed both to test the operation of the system itself, and to evaluate the success of the methodology with regard to attack detection outcomes. This entailed the construction of a test network consisting of two 'client' machines, each hosting Snort and Dragon NIDS. The alerts from these systems are then interpreted by IDS Alert Agents, and sent to the Control Unit on a separate third machine. The Control Unit is co-located with a PostgreSQL Alert Database on a third host, and stores all alerts received in the database. This is then accessed from a separate fourth machine acting as the Administrative Console. The machines were placed on a switch, to separate traffic into distinct network segments.

5.1. Attack Detection Outcomes

In order to test the prototype, a suite of attacks with different characteristics was selected. The attacks utilised a number of different attack tools: two DoS-type tools, *stick* and *tfn2k*, the port-scanner *nmap*, the vulnerability scanner *Vetescan*, and two exploit tools which attack network management protocols. Four distinct attacks were run:

Attack 1. A simple nmap stealth scan, with two phases: one with nmap running alone, and one running with the IDS alert spoofing tool stick run at the same time to obfuscate the nmap alerts.

Attack 2. This attack consists of running Vetescan and the two exploit tools on both hosts.

Attack 3. The Cuppens attack [7], which consists of 6 steps, of which only five (all except step 5) are detectable by Snort and Dragon:

1. finger root@target
2. rcpinfo <target>
3. showmount <target>
4. mount <target directory>
5. cat "++" < .rhost
6. rlogin <target>

Attack 4. An attack using the DoS tool tfn2k, first against one host, and then against both hosts, in order to determine the difference in amounts of alerts, if any.

The attacks were scripted in order to satisfy reproducibility concerns, and the session was logged using TCPDUMP for later analysis. Using scripting meant that the attacks could be performed quickly and were repeatable for multiple iterations if required.

The attacks were run in four different scenarios: sequentially; sequentially with background traffic; simultaneously; and simultaneously with background traffic. These scenarios enabled us to test the performance of the detection algorithms in very different circumstances, with differing levels of alert dispersion and intensity. The background traffic was injected onto the network while the attacks were running. This was achieved using the TCPREPLAY utility, in order to make the testing process reproducible.

As shown in Table 1, in all except two instances, each of the attacks was detectable using our signature-based attack detection approach. Attack 2 was not detected in either of the cases where the attacks were run simultaneously. This was because higher priority alerts, such as those resulting from Attacks 1 and 3, have priority over the alerts that the Attack 2 signature required. The original alerts were not propagated by the detection engine

due to the exploit signature either not being active (and therefore not acknowledging the need for that alert), the exploit signature timing out before the alert could be reassigned, or the alert giving rise to a new instantiation of the higher priority signatures as they were needed for a group signature. However, the signatures could be matched if the priority for Attack 2 was increased to be higher than Attacks 1 and 3.

Table 1: Summary of results for each scenario tested

Signature	TCP-Scan	Exploit	Cuppens	DoS Attack
Sequential	Detect	Detect	Detect	Detect
Sequential w/background	Detect	Detect	Detect	Detect
Simultaneous	Detect	Detect*	Detect	Detect
Simultaneous w/background	Detect	Detect*	Detect	Detect

* This was not detected when run normally. This attack was only detected when its signature was rated at a higher priority than TCP-Scan and Cuppens (not the case by default in testing).

We also performed experiments to determine the effect of different signature parameters on the matching operation. The results of those experiments are presented in the following sections.

5.2. Other Test Results

5.2.1. Mandatory/Optional/Weighting As described in Section 2.1, an alert appearing in a signature can be tagged as optional, in which case it is tagged also with an associated weight. In the absence of optional alerts, a signature will fire – assuming no timeouts have occurred - when all its (mandatory) alerts have been matched by the alert stream. With optional alerts, it is possible to construct signatures whose firing depends not only upon the matching of the regular or mandatory alerts forming part of the signature, but also upon the matching of a sufficient sub-set of optional alerts. This is achieved by specifying a signature threshold for the cumulative optional alert weight and specifying individual alert weights for each optional alert. Note that only the first alert of the type contributes to the cumulative optional alert weight. We performed the following experiments varying the threshold weight to test the performance of this feature.

The Attack 1 (TCP-Scan) signature consists of three alerts in a group, "TCP-SCAN", "Scan Proxy (8080)" and "ICMP Ping NMAP". It is illustrated in Figure 3 below.

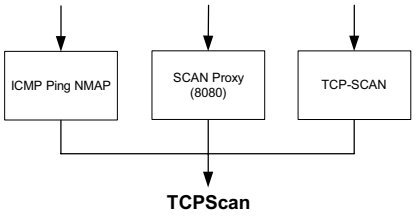

Figure 3: Attack 1 (TCP-Scan) Signature

Attack 1 can be specified as a '2 out of 3' example, where if two of the signature's alerts are detected, the TCP-Scan signature should be fired. This can be accomplished by setting each item to optional and each alert's weighting to be 1, with a threshold of 2. Table 2 gives the results of experiments relating to modifying the threshold value for optional alerts. (The attack resulted in 6 TCP-SCAN, 3 SCAN PROXY and 2 ICMP PING NMAP alerts).

The testing consisted of four cases:

Case 1. All Mandatory – Testing how many times the signature will match when requiring all alerts.

Case 2. All Optional, Threshold 1 – Testing how the signature will match when 1 out of 3 is required.

Case 3. All Optional, Threshold 2 – Testing how the signature will match when 2 out of 3 is required.

Case 4. All Optional, Threshold 3 – Testing how the signature will match when 3 out of 3 is required (effectively identical to Case 1).

The number of expected and actual matches for each case is listed in Table 2.

Table 2. Number of alerts generated for different threshold weights (M-Mandatory, O-Optional)

Case	1	2	3	4
Threshold Weight	N/A	1	2	3
TCP-SCAN	M	O	O	O
SCAN Proxy (8080)	M	O	O	O
ICMP PING NMAP	M	O	O	O
# of expected matches	2	11	5	2
# of actual matches	2	11	4*	2

* An out-of-sequence alert affected this result, meaning practical results did not meet theoretical expectations.

This feature should be useful in specifying random attacks such as spoofing attacks, portions of DoS attacks

or even scanning activity.

5.2.2. Alert Repeat Factor The alert repeat factor is useful for signatures designed to detect attacks which consist of multiple instances of the same alert. A good example of such an attack is a denial of service attack, such as our Attack 4.

Attack 4, being a DoS attack, has a large number of alerts normally associated with it, however the alerts that appear are of three types, shown in Figure 4.

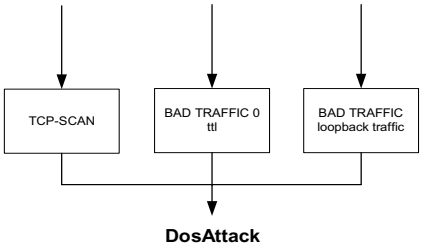

Figure 4: Attack 4 (DosAttack)

If a signature was used that required only a single instance of each of these alerts, it would fire tens of times per second. The alert repeat factor is provided in order to allow an administrator to better configure how the system responds to these sorts of attacks.

With a well-tuned alert repeat factor, the number of times a signature matching occurs can be reduced. Table 3 shows the relative numbers of the three different alerts within our simultaneous scenario.

Table 3. Results for various alert repeat factors for Attack 4

	Total Alerts	Start Time	End Time
TCP-SCAN	6	3:52:56	3:53:27
BAD TRAFFIC 0 ttl	4895	3:52:55	3:53:17
BAD TRAFFIC loopback traffic	67	3:52:55	3:53:16
Total	**4968**	**3:52:55**	**3:53:27**

It should be noted that the distribution of these alerts can vary significantly, with the "BAD TRAFFIC 0 ttl" alert *on average* occurring 73 times more than the "BAD TRAFFIC loopback traffic" alert but in practice this number varies from 20-100 times.

Table 4 shows the results of varying the number of repeated alerts in a single DosAttack instance.

Table 4. Results for various alert repeat factors for Attack 4

Case	1	2	3	4	5	6
TCP-SCAN (Optional)	1	1	1	1	1	1
BAD TRAFFIC 0 ttl	1	1	1	1	10	90
BAD TRAFFIC loopback traffic	1	2	3	68	1	1
# of expected matches	67	33	22	0	67	54
# of actual matches	66	32	21	0	65	44

The results indicate that the number of signature matches in this case is largely dependent upon the "BAD Traffic loopback traffic" alert. When the repeat factor for this particular alert is increased the overall number of signature matches is reduced. The high values for the "BAD TRAFFIC 0 ttl" alert repeat factor indicate that there are large numbers of this particular alert being generated. The administrator is able to tune the desired number of alerts of a particular type required in order for the signature to match thus allowing the number of times a signature fires to be reduced.

5.2.3. Signature Priority Signature priority comes into play when two signatures are awaiting the arrival of the same alert. Examples are our signatures for Attacks 1 and 2, which both utilise the alert "SCAN Proxy (8080)", and Attacks 2 and 3, which both utilise the "RPC portmap listing" alert. When the attacks were run sequentially, Attack 2 was successfully detected. However, during the simultaneous attack runs, the Attack 2 signatures were denied access to mandatory alert components. This is because the required alerts were consumed by higher priority signatures, namely those for Attacks 1 and 3. This illustrates the importance of selecting suitable priority values, especially when using our "single-fit" strategy.

This scenario could be avoided (possibly at the expense of larger numbers of false-positive matchings) by employing a "multiple-fit" strategy. Further research is required to analyse these issues in greater detail. It should be noted that, as expected, we were able to cause the signature for Attack 2 to match by giving it a higher priority that those for Attacks 1 and 3. Of course, in this instance the matching of Attacks 1 and 3 is inhibited.

5.2.4. IP Mask A limitation of the test network meant that the specification of an IP mask in the signatures could not be tested. For this reason we were not able to perform any experiments using the IP mask signature component. For the purposes of illustration we show how the IP mask may be useful for signatures when detecting Attack 4. A denial of service attack may be directed at a range of addresses on the victim network, impacting availability of the entire

network. By specifying a suitable IP mask in the signature, rather than a single target address, the effectiveness of the signature can be improved since a single signature can accumulate alerts generated for multiple target hosts.

6. Conclusions and Future Work

Our results to date have demonstrated a viable signature-based attack detection scheme that utilises commodity components. The signature structure which we propose is sufficiently flexible to detect a variety of complex attack scenarios. Through practical experimentation with the signature constraints, including mandatory/optional components with weighting, alert repeat factors and signature priorities it has been shown that this approach allows for the specification of accurate, yet fault-tolerant, signatures. The system we have developed was not intended for real-time analysis of alerts. Instead it has proven to be a useful test-bed for the development of complex attack signatures through experimentation with the various signature parameters in an off-line environment.

The results are preliminary and more comprehensive testing is needed using a wide range of multi-step attacks, with alert information from both HIDS and NIDS, in order to establish performance with respect to error rates and to more firmly establish that the signature specification notation we have developed is sufficiently expressive.

The signature notation we have developed promises to be useful for alert aggregation and reduction. In particular, the alert repeat factor acts as a tool for alert reduction (via aggregation). More experiments must be conducted to fully explore the benefits of this approach for reducing the overall number of alerts generated by the system.

A multiple-fit signature-matching algorithm is currently being investigated. Multiple-fit needs to incorporate multiple simultaneous alert-matching threads in order to maintain concurrent matching in situations where alerts match multiple attack signatures. In this way an alert will potentially be matched to multiple signatures. In simplistic terms, this concurrent matching will then fire when the first contributing signature is completely matched, followed by rollback with respect to the other contributors, although this too presents issues with respect to relative priorities (severity) of the competing potentially matching signatures.

Finally, further work is needed with respect to visualization to assist in analysing alerts and developing attack signatures. Additional features that have been identified for inclusion in the alert graphic tool to date are:

- graphing with and without the alerts attributable to signatures,
- graphing of the alerts only attributable to signatures, and
- the specific graphing of items such as pre-defined clusters.

8. References

[1] McHugh, J., "Intrusion and intrusion detection", *International Journal of Information Security*, 1:14– 35, 2001.

[2] Lee, W. and S. Stolfo, "Data Mining Approaches for Intrusion Detection", *Proc. 1998 7th USENIX Security Symposium*, http://www.cs.columbia.edu/~sal/hpapers/USENIX/usenix.html, last visited Aug. 2002.

[3] Morin, Benjamin and Ludovic Me, Herve Debar and Mireille Ducasse, "M2D2: A Formal Data Model for IDS Alert Correlation", In the *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection* (RAID2002), Zurich, October 2002, volume 2516 of LNCS, pages 115-137, Springer-Verlag. 16

[4] Vigna, G. and R.A. Kemmerer and P. Blix. "Designing a Web of Highly-Configurable Intrusion Detection Sensors". In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection* (RAID 2001), Davis, CA, October 2001. Volume 2212 of LNCS, pages 69–84, Springer-Verlag.

[5] Doyle, Jon and Isaac Kohane, William Long, Howard Shrobe, and Peter Szolovits, "Agile monitoring for cyber defense". In *Proceedings of the DARPA Information Survivability Conference and Exposition* (DISCEX-II), Anaheim, California, June 12-14 2001.

[6] Valdes, Alfonso and Keith Skinner. "Probabilistic alert correlation". In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection* (RAID 2001), Davis, CA, October 2001. Volume 2212 of LNCS, pages 54–68, Springer-Verlag.

[7] Cuppens, Frederic and Alexandre Miege. "Alert correlation in a cooperative intrusion detection framework". In *2002 IEEE Symposium on Security and Privacy* (S&P'02), Berkeley, CA, USA, May 12 - 15 2002, pages 187–202.

[8] Debar, Herve and Andreas Wespi. "Aggregation and correlation of intrusion detection alerts". In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection* (RAID 2001), Davis, CA, October 2001. Volume 2212 of LNCS, pages 85–103, Springer-Verlag.

[9] Cuppens, Frederic.. "Managing alerts in a multi-intrusion detection environment" In *Annual Computer Security Applications Conference* (ACSAC 2001), New Orleans, Louisiana, USA, December 10-14, 2001.

[10] Carey, N., G. Mohay, and A. Clark. "IDS Interoperability and Correlation Using IDMEF and Commodity Systems" In *Proceedings of 4th International Conference of Information and Communications Security (ICICS)*, Singapore, December 2002. Volume 2513 of LNCS, pages 252-264, Springer-Verlag.

[11] Cuppens, Frederic and Rodolphe Ortalo. "Lambda: A language to model a database for detection of attacks". In *Third International Workshop on Recent Advances in Intrusion Detection* (RAID 2000), Toulouse, France, October 2-4, 2000. Volume 1907 of LNCS, pages 197–216, Springer-Verlag.

[12] Tung, Brian, "The common intrusion specification language: A retrospective." In *Proceedings of the DARPA Information Survivability Conference and Exposition* (DISCEX), 2000, pages 3–11, Hilton Head, SC, 2000.