

IDS Interoperability and Correlation Using IDMEF and Commodity Systems

Nathan Carey, Andrew Clark and George Mohay

Information Security Research Centre,
Faculty of Information Technology,
Queensland University of Technology,
GPO Box 2434, Brisbane,
Queensland, 4001
{n.carey, a.clark, g.mohay}@qut.edu.au

Abstract. Over the past decade Intrusion Detection Systems (IDS) have been steadily improving their efficiency and effectiveness in detecting attacks. This is particularly true with signature-based IDS due to progress in attack analysis and attack signature specification. At the same time system complexity, overall numbers of bugs and security vulnerabilities have increased. This has led to the recognition that in order to operate over the entire attack space, multiple IDS must be used, which need to interoperate with one another, and possibly also with other components of system security. This paper describes an experiment in IDS interoperation using the Intrusion Detection Message Exchange Format for the purpose of correlation analysis and in order to identify and address the problems associated with the effective use and management of multiple IDS. A study of the process of intrusion analysis demonstrates the benefits of multi-IDS interoperation and cooperation, as well as the significant benefits provided by alert analysis using a central relational database.

Keywords: Intrusion Detection, Data Analysis, Correlation, Interoperability, Network Management

1 Introduction

Intrusion Detection Systems (IDS) have evolved significantly over the past two decades since their inception in the early eighties [6]. The simple IDS of those early days were based either upon the use of simple rule-based logic to detect very specific patterns of intrusive behaviour or relied upon historical activity profiles to confirm legitimate behaviour. In contrast, we now have IDS which use data mining and machine learning techniques to automatically discover what constitutes intrusive behaviour and quite sophisticated attack specification languages which allow for the identification of more generalised attack patterns.

IDS are still however commonly characterised according to a two-fold taxonomy involving detection method on the one hand and placement on the other. Classification by detection method relates to signature-based vs. anomaly-based IDS while classification by placement relates to host vs. network based IDS. Many IDS can be described in this way, with the exception of emergent hybrid IDS, which may use multiple placement types or detection methods.

Signature based IDS constrain the range of attacks detected using specific detection patterns in return for acceptable detection error rates, while anomaly based IDS cover the entire attack space by looking for anomalies within their data source at the cost of increased error rates.

Host based IDS or HIDS, use host logs and host event records to provide a record of current activity which can then be analysed. Network based IDS or NIDS, typically use packet headers and packet level information, sometimes even packet payload, as their working data. Due to the speeds required in modern networks, NIDS are typically signature based.

With a wide range of IDS and applications, it is inevitable that individual IDS have their own areas of specialisation and effectiveness [7]. As a result, with the broad range of systems and networks in use today, many systems use multiple IDS which brings with it the associated challenge of achieving a consistent approach to IDS management and analysis of IDS alerts.

The Intrusion Detection Working Group (IDWG) emerged following previous work done by the Common Intrusion Detection Framework (CIDF), both efforts attempting to standardise and formalise the work of cooperating IDS. The IDWG efforts so far have focussed on two standards, a data exchange format and a protocol for communication. The Intrusion Detection Message Exchange Format (IDMEF) built on the experience of CIDF, but given the increased deployment of XML for specifying protocols across the Internet used XML for expressing the two protocol requirements. The IDMEF Data Type Definition (DTD) is currently in version 1.0, and provides a rich and extensible alert representation for a broad range of IDS applications.

The motivation and background for our research is described in more detail in Section 2. Section 3 describes the detailed design of the prototype software that has been developed for interoperability and alert correlation, as well as testing procedures and experiments performed with the prototype. Section 4 then presents a case study of an experimental attack analysis, while Section 5 addresses future work and conclusions.

2 Related Work and Motivation

Our work is related to work by Vigna et. al.[10], Doyle et al. [5], Valdes and Skinner [9], Cuppens [1] and Debar and Wespi [4]. 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 mechanisms used are different. The STAT framework [10] provides the ability to perform dynamic configurability of STAT components and uses Java to aid in portability. The MAITA [5] project reflects similar goals, but utilises a more complex architecture to support interoperability and uses trend templates as opposed to STATL (used by the STAT framework) for the specification of chains of events. 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 use features in the Tivoli Enterprise Console to perform correlation, and focus on the abilities of an management system to reduce the amount of data presented to an administrator. Recent work by Cuppens [1, 2] 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. Our work, by comparison focuses on a simple framework built of commodity and free software in order to produce practical alert correlation across heterogenous commodity IDS. Our work also differs in concepts to do with implementation, such not using stored procedures, the nature of interaction between analysis and signatures, and the format and implementation of signatures themselves.

We use the following terms and definitions inherited from both recent work cited above, but we define them again here for convenience.

Attack: A series of steps taken by an attacker to achieve an unauthorized result; any attempt to gain knowledge of or penetrate a system; includes scanning, probing, mapping, etc. ¹

Alert: An alert is a warning message generated by an IDS. It may indicate an attack or suspicious event.

Complex Attack: An attack that may comprise multiple steps and generates multiple alerts. This definition is based on that by Cuppens [1].

Attack Signature: An attack signature is the specification of a pattern of alerts whose occurrence would indicate a complex or multi-step attack. In our work, signatures consist of a combination of groups and/or sequences of alerts, in any order.

Alert Cluster: An alert cluster is a group of alerts that are related by one or more common features, such as time, source or destination. This relationship may be logical, mathematical or based on statistical grouping. We use this definition of cluster as opposed to some others e.g., in the work of Cuppens, who uses 'cluster' to term grouped alerts corresponding to a single instance of an attack.

¹ <http://www.cert.org/security-improvement/modules/m06.html>

Our work has been motivated by the previous work above and attempts to address the application of IDS data analysis within a simple architecture based upon commodity IDS. As a result, our goals are:

- to provide standardised IDS interoperability;
- to provide a capability for multiple levels of analysis of multi-IDS alerts;
- to provide centralised management of IDS.

Interoperability is important in order to provide a proper base for information sharing amongst IDS and possibly other system security components e.g., between IDS and firewalls, or host logging information. This allows us to exploit the use of heterogeneous IDS and other components to provide identification and notification of a wider range of alerts than is possible with homogeneous IDS.

The need for alert analysis arises from the two main tasks performed by an IDS administrator - to identify individual attacks and group them by characteristics, and the ability to investigate a specific stored alert to discover other alerts that may be related.

While the first task is simple data aggregation and representation, the second relies on correlation, the selection of sets of alerts that could be related, and induction by ascertaining which alerts in the set of relevant alerts should be investigated further.

Centralised management of IDS enables us to integrate the features of interoperability and global IDS data analysis (in this case correlation) into a product that is both useful and efficient. While interoperability and correlation can be achieved without requiring a central management framework, centralised management facilitates system extensibility with regard to incorporation of new tools, and efficiency with regard to removal of redundant or superfluous functionality.

In particular correlation provides us with the potential ability to see beyond the actual alerts themselves, and determine trends, find patterns and infer relationships between alerts - capabilities which a single sensor is unlikely to have, and can be far more easily and comprehensively performed with multiple IDS.

3 Prototype for IDS Interoperability

One of the goals of our research has been to produce a prototype that draws upon standardised communication and alerting formats, as well as making use of commodity systems. Our approach is to convert native alerts from Snort and Dragon to the IDMEF format, store these IDMEF alerts in a central database, and perform analysis on the data both with native SQL queries and custom algorithms. We feel this commodity IDS solution suits the 'real world' concerns of many IDS users, rather than being customised for particular IDS, which ties usage to particular products. The decision to use common IDS unfortunately means that some aspects of an alert may be ignored in favour of maintaining a common 'core' set of information provided by the majority of IDS. The exact ways in which this effects our prototype are discussed later in the paper.

There are a selection of free tools for IDS analysis, dealing almost exclusively with Snort alerts. Examples include ACID, Demarc and SnortReport. The aim of our prototype is to provide the functionality of these free tools as well as more sophisticated event processing and more detailed correlation capabilities. Some systems, including some commercial offerings, provide administrators with a web browser interface to manage their systems. While this may be appropriate for some systems, the range of features we desired required a full GUI, with the ability to run on multiple OS. Two IDS were chosen for the prototype, due to their suitability both in terms of OS platforms and alert format. These two IDS were Snort and Enterasys Dragon.

After first looking at the possibilities of the native Snort alert format for simple centralised IDS alert storage (due to the tools currently available for Snort mentioned above), it was decided, for interoperability reasons, to look at the capabilities of the emerging IDMEF standard.

While IDMEF is a very expressive and extensible format, our usage is limited to the common set of information available from virtually any IDS, that of alert classification (or alert name), time, source and target. Omitted information that could be considered useful with more widespread IDS support includes priority of the alert, host names and DNS names of hosts, and the impact of the alert.

3.1 The architecture

The prototype architecture consists of three major components: the Alert Agent (AA), the Control Unit (CU) and the Administrative Console (AC). These three components interact in a manner so as to provide a simple architecture and allow for the majority of applications to be integrated easily and effectively.

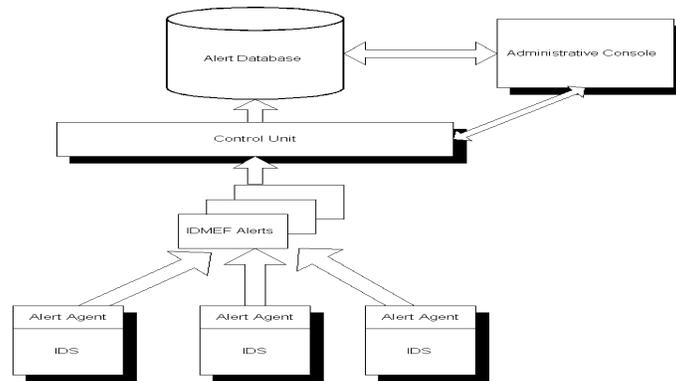


Fig. 1. The Architecture

Alert Agent. The Alert Agent performs translation and communication of IDMEF alerts to the CU. It can utilise either a native IDS IDMEF output format such as the Snort output plugin, or translate to IDMEF from the native IDS format. Generally ‘native’ (IDS generated) IDMEF messages should be considered a more accurate data source than converted messages, due to the increased amount of information that can be gathered at alert time versus conversion later, but the basic information required in an IDMEF message can be met by virtually any commonly used IDS alert format.

Control Unit. The recipient of all IDMEF messages is the Control Unit. The CU is responsible for alert processing and storage, as well as implementing any real-time alert analysis (not implemented at this time). When an alert is received it is queued for storage in the database. This queue is buffered which allows our prototype to handle influxes of alerts, such as in the case of a DoS attack. When storing, the alert is converted to the SQL tables by a custom IDMEF-DBMS mapping module and a JDBC connection pool. This allows IDMEF alerts to be stored in a normal relational database, in this case a Postgres database, aiding in portability and the effectiveness of later searches on alert data. The database contains all the information stored in the IDMEF alert, though only a small amount of this is generally used in our analysis. With the addition of more support from IDS vendors to supply the additional information possible in an IDMEF alert, analysis could include alert priority and impacts in order to perform better tracing of possible cause and effect relationships between alerts.

Administrative Console. This application allows the administrator to gain access to the alerts stored in the database. In the current off-line method, the simplest form of correlation, the aggregation and sorting of alerts, is performed by SQL queries on the database. Complex correlation (where mathematical patterns maybe involved) is performed by Java algorithms working on the produced database result set. The process by which data is analysed is discussed further below. The Administrative Console is designed to be the only device actually reading the database, though there could be multiple consoles running simultaneously. The

console also has rights to alter information inside the database if required, to perform tasks such as delete old records, merge alert information or change alert data to add extra information.

3.2 Data Analysis

Figure 2 shows the four step process we use to analyse IDS data and shows the waterfall nature of the stages and the information flow between them. This model builds on concepts mentioned in related work discussed above. We use this model to systematically reduce the workload and volume of information in later stages, and to concentrate the amount of relevant data to be used in the correlation and induction stages. The basic process of alert analysis uses the stages below to group alerts into clusters which are related in some way, either by an attack signature or by a pattern, such as time difference or grouping by time. How these relations are developed is described below.

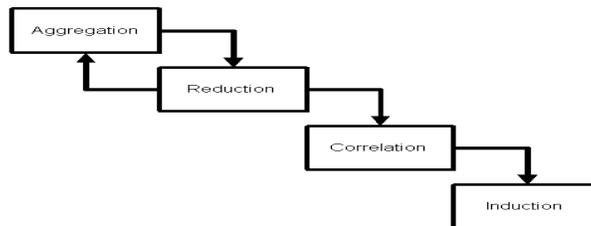


Fig. 2. Data Flow Diagram

Stage 1: Data Aggregation. At this point, alerts are grouped by time, classification, source and destination using similarity metrics. These metrics currently use simple factors such as time period, time difference between alerts (to detect sequences), IP differences and matches on alert classification, host or IDS type.

Stage 2: Data Reduction. After determining which alerts can be grouped together, we remove redundant alerts. This can be performed by merging alert groups based on pre-defined criteria (such as ignoring certain types of messages from a particular IDS) and deleting the originating alerts, or using signatures for patterns of alerts to delete. Examples of this could be low-level alerts of no interest, alerts not appropriate for the environment, such as IIS attacks on an Apache web server, or duplicate alerts. The goal is to remove that information that is definitely not relevant, therefore while the process of removing alerts is simple, the logic can become quite complex. Reduction in this iterative stage fashion is not currently used in our system, though the major functionality is implemented off-line using SQL.

Stage 3: Data Correlation. With the alerts reduced to a more relevant subset, the software considers which alerts may be correlated. Correlation is currently based on time, IP, location, and name, but in principle could be virtually any field deemed relevant in determining an attack. Examples of this could be noticing sequences in fields like ports (such as in the case of a portscan), or even the process of an attack on a HIDS (such as might occur in a resource exhaustion attack). The relationship may be based directly on a signature, or by the system watching for certain abnormal operation, such as the above cases of sequences of ports opened by a single host (for a portscan) or a large amount of processes opened by a single user (for the above host attack). The exact nature of what the system can determine as 'abnormal operation' is yet to be defined.

Stage 4. Data Induction. At this point in analysis, the software will extrapolate from the data either to determine events that may have occurred, or would be likely to occur. Induction uses more complex analysis than does correlation but the same general principles apply. The major difference is that induction is used to extrapolate or interpolate information from the dataset, rather than simply determine relationships. Examples of this could be predicting the next stage of an attack, deducing missing components of an attack, or determining possible spread of an attack through the network. In our current prototype, induction is not yet included, though will be closely tied to the 'abnormal operation' aspect of correlation.

3.3 Attack Signatures

Attack specification languages such as LAMBDA [3], and CISL [8] provide the ability to define very complex relationships between *events* themselves. However, our only requirement at this stage is that the signatures be able to describe the relationships between the *alerts* generated by IDS as opposed to events. In order to best solve our specific problems, we have developed our own signature format for patterns of alerts. Our signature scheme is designed to be simple yet still have the capability to define reasonably complex patterns. It is used only to represent alert relationships, and as such requires less complexity and expressiveness than the above systems. The general requirements for our signatures are:

- The signature must contain information about the conditions on the alerts comprising the signature. This includes whether the alerts are a sequence or unordered group, any temporal constraints on the interval between alerts, any spatial constraints (i.e., same host or subnet) and the timeout for expiry of the signature.
- Signatures should be able to generate an alert themselves or be able to trigger new monitoring processes, so multiple signatures can be combined in a sequence or hierarchy if necessary.

A signature may be of two types:

1. For signatures which define alert sequences, the monitor process attempts to find a match for the first item only, and when triggered by such a match, a new monitor process is spawned which looks for the next step in the sequence. This means a sequence of n alerts could potentially have $n-1$ monitor points. A sequence signature monitor expires when the interval between individual steps is reached, or the overall signature timeout is reached.
2. For signatures which define alert groups, the monitor must check for each item in the group. For this reason, groups should be kept small, or significant processing could be required. When an alert within the set is matched, it is flagged so that it will not be matched against again. A group signature monitor expires when a timeout occurs on the interval between alerts. A signature-wide timeout is not needed, as functionally, this would be equivalent to the timeout between alerts.

One property of our system, is that signatures can specify what operations to perform on the alerts which contributed to the signature (e.g. store, delete, merge), as well as any responses to be effected. This relatively simple system specification can accommodate many conceivable relationships of interest by being able to express signature composition using multiple signatures, without storing redundant information. An example of our signatures being used to describe a multi-step attack across two different IDS in multiple stages is included in the Evaluation and Testing section.

We currently use a first-fit system to determine matching on signature monitors, which means that the first signature which matches an alert will prevent any other signatures from matching it. This has implications for signature ordering and the construction of signatures, but is not a significant impediment at this stage of research. Investigation into other methods for matching, such as 'best-fit' or 'multi-fit' approaches may be used later if needed.

3.4 Discussion

The architecture is purposely minimalistic. Other approaches propose more sophisticated architectures which are likely to be difficult to manage and maintain. We have purposely not included a comprehensive communication mechanism or information sharing protocols. As our system is a testbed for data analysis, more

complex issues of distributed monitoring, information sharing policy and complex data channels would distract from the focus. We consider our simple communication to be an adequate model in many cases, especially in small to medium sized implementations, or those already using secure communication channels (such as a separate ethernet network dedicated to IDS) or IDWG’s communication protocol, the Intrusion Detection eXchange Protocol (IDXP).

The current system is implemented in an off-line context, in order to properly evaluate the capabilities of the system for centralised monitoring and analysis without the distraction of requiring real-time operation. We have nonetheless endeavoured to include consideration of performance and eventual real-time implementation while developing the architecture.

4 Evaluation and Testing

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 versus the operation required by a human administrator. This helped us distinguish those elements of the system that were useful, and identified algorithms used in human analysis that could be replicated in our prototype.

4.1 Testing

The testing procedure was developed both to test the operation of the system itself, and to evaluate the success of the methodology for data analysis. This entailed the careful construction of a test network to enable the evaluation of analysis across network boundaries as well as investigation of the abilities of one IDS to reinforce or invalidate the data of another. Figure 3 shows the testing framework for the software.

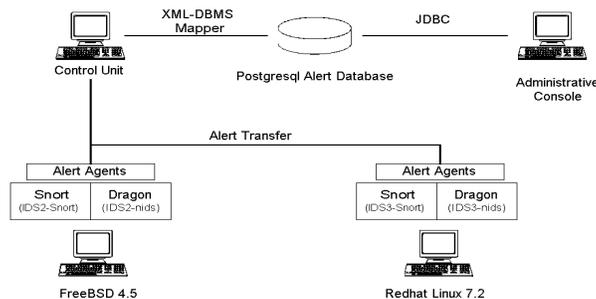


Fig. 3. Testing Framework

Two ‘client’ machines are used, each hosting Snort, Dragon Squire 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 Postgres Alert Database, and stores all alerts received in the database. This is then accessed from a separate machine acting as the Administrative Console. All the machines were placed on a switch, to separate traffic into distinct network segments.

A suite of attacks capable of being launched across multiple machines was selected, consisting of two DDoS tools, stick and tfn2k (Case 2 below), as well as multiple variations of nmap scan (Case 1 below) and a multi-stage attack described by Cuppens [3] (Case 3 below). A vulnerability scanner and two attack tools were also run, but not described below as the alerts could not be correlated directly with the attacks themselves. The Cuppens attack 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>`

The attacks were scripted in order to satisfy reproducibility concerns, and the session was logged in TCPDUMP for later analysis. Initial experiments had used TCPReplay and TCPDUMP to reproduce attack sets, but the sophistication of Dragon and Snort meant that some data ‘normal’ replayed (such as obsolete TCP SYN/ACK sequences) would trigger alerts, and the incorrect session generation meant that some attacks were (correctly) ignored as erroneous. Because of these problems, we used shell scripting to automate the attacks. Using scripting meant that the attacks could be performed quickly and repeatably for multiple iterations if required and avoid the problems of stale packets with TCPDUMP.

4.2 Correlation Analysis

After running the three attacks (Case 1, Case 2 and Case 3) below, we commenced correlation by querying how many alerts had been logged. The four IDS, IDS2-dragon, IDS3-dragon, IDS2-snort and IDS3-snort produced a total of 8799 alerts. A list of alerts per IDS shows IDS2-dragon had 7 alerts, while IDS2-snort had 3737, IDS3-dragon 3784 and IDS3-snort 1271.

Case 1: TCP Scan. Looking at the abnormally low number of alerts on IDS2-dragon, we observed that it contained three unique types of alert, one ‘HEARTBEAT’, one ‘SSH:VERSION-1’, and five ‘TCP-SCAN’. The TCP-Scan alerts are potentially dangerous, and have timestamps of 1:22:52, 1:22:55, 1:23:10, 1:23:14 and 1:24:18. Based on the similarity in times we can infer three separate events have occurred. By searching through the database for time periods +/- 5 seconds around these groups of alerts on other IDS, we observed that many scan-related alerts (such as ‘ICMP Ping NMAP’ and ‘Scan Proxy’ messages from Snort and a ‘TCP-SCAN’ message from Dragon) occurred across both IDS2 and IDS3 in the 10 second periods around 1:22:50 and 1:23:10, and that the third time (1:24:18) is the result of separate traffic. Considering there were 18 alerts (including a normal HEARTBEAT message from IDS3-Dragon) from the four different IDS (2 IDS2-Dragon, 11 IDS2-Snort, 3 IDS3-Dragon, 3 IDS3-Snort) in the first period, and 11 alerts (2 IDS2-Dragon, 7 IDS2-Snort, 1 IDS3-Dragon, 1 IDS3-Snort) in the second, we can safely conclude that two separate large-scale scans occurred.

Case 2: DoS Attack. As noted above, the other alerts around 1:24:18 alert did not correspond to a TCP-Scan. From 1:24:18 to 1:24:49 3551 alerts from varying IP’s logged as ‘BAD-TRAFFIC’ requests on IDS2-Snort. From 1:24:38 to 1:24:48 1246 ‘BAD-TRAFFIC’ alerts were received on IDS3-Snort. Interestingly, IDS3-Dragon logs 3758 alerts in this time, indicating that Dragon on FreeBSD may contain DoS protection in the kernel network code or the FreeBSD version of Dragon itself. Large amounts of varying source alerts could indicate either a spoofed single host DoS attack, or a bone-fide multi-host DoS attack. As there were 1625 unique IP’s, all of which had the last IP quad as ‘0’, one could reasonably assume a spoofed DoS attack has taken place.

Case 3: Cuppens Attack. Without extra events to analyse, by looking at the relative levels of alerts on both IDS, we note that IDS3 has more alerts than IDS2, and that Snort logs less alerts than Dragon. The large amounts of alerts for this host indicate a higher chance of attack, therefore warrants further investigation. In much the same way as Case 1 above, we can look at the types of alerts on IDS3 and relate these to each other in order to identify attacks.

Using the smaller dataset on IDS3, and using a query to show a list of unique alert names, Snort logs 9 different types of alert: 2 types of ‘BAD_TRAFFIC’, plus ‘FINGER root query’, ‘ICMP PING NMAP’, ‘RPC portmap listing’, ‘RPC portmap request mountd’, ‘RSERVICES rlogin root’, ‘SCAN Proxy (8080)

attempt' and 'X11 outbound client connection detected'. Looking at the occurrences of each of these, 1260 come from BAD TRAFFIC alerts (related to the DoS attack, above), two occurrences of the SCAN Proxy alert, two occurrences of ICMP PING NMAP, three occurrences of RPC portmap listing and one of FINGER root query, RPC portmap request mountd and RSERVICES rlogin root. Looking at the times of the alerts, 1:23:46 contains two instances of RPC portmap listing, as well as the RPC mountd and RSERVICES Login attempt, comes 30 seconds before the DoS attack, and contains the majority of types of alerts the IDS logged - this makes the alerts from this particular time suspicious.

A search for alerts from all IDS for 10 seconds around this time gives us 16 entries confined to IDS3, 5 from Snort and 11 from Dragon. This period adds the "FINGER root query" from Snort mentioned earlier, plus a FINGER:ROOT, RSH:ROOT and 8 DYNAMIC-TCP messages from Dragon. The FINGER and RSH:ROOT messages match those of Snort, while the DYNAMIC-TCP messages match the RPC messages of Snort. These alerts constitute a match to the sequence that indicate Cuppens multi-stage attack. In this case Dragon's alerts allow us to validate the results of Snort. Note that both systems missed the 'cat ++' stage - it is at this point the addition of a host-based IDS would help to increase detection scope and effectiveness. Notice that this attack can be identified from others simply by looking for anomalous alerts and correlating information between IDS.

This attack can also be detected using signatures. In our case, we note that "FINGER root" and "FINGER:ROOT" are equivalent, so create a signature of the name "DUAL IDS Finger Root Detection" consisting of both of these alerts, and we create another with "RSH:ROOT" and "RSERVICES rlogin root" called "DUAL IDS Remote Root Login Detection". If we combine these in a sequence with the RPC messages from Snort, as well as a single occurrence of "DYNAMIC-TCP" (to cover the Dragon-Snort RPC overlap) we create a signature for the combination of these alerts that allows the checking of detection across heterogeneous IDS. This allows for better error rates due to the reduced likelihood of false positives across both IDS simultaneously.

Summary The end result is that we are able to identify signatures of certain large-scale attacks across time and space, as well as the anomalies which may indicate suspicious activity with simple SQL queries and a small amount of extra analysis. We replicated the experiment of Cuppens [1] "multi-stage" attack using Snort and Dragon instead of Snort and E-trust, without complicated attack signatures.

We see that the levels of data reduction and aggregation are included in the SQL query generation, while alert correlation occurs within an outside process. The current system iterates through the data either in the manner shown above looking for abnormal events, or by searching for matches to simple static signatures. It cannot currently extrapolate or interpolate information, but that functionality is the subject of further research.

As well as demonstrating the process of an attack analysis, this example also shows the advantages of cross-IDS and cross-locational analysis. In all cases, the overlap of IDS alerting provided an increase in information regarding the nature of the attack, and correlating across location helped to identify the scale of the attack.

5 Conclusions

Our aim in developing the prototype has been to implement alert correlation across multiple real-world IDS. We have achieved successful development of such a prototype to support IDS interoperability using several concepts that build upon previous work, as well as new concepts, such as simple signature specification and grouping of alerts. This has demonstrated proof of concept of the ideas used in the research and it now remains to deploy the prototype in a wider range of real-world situations, in particular the context of a wider range of attacks and background traffic, in order to better explore the architecture's potential for analysis using the concepts of aggregation, reduction, correlation and induction. Further work will address optimisation of the algorithms used in the architecture, and deployment of the system in a real-time environment.

References

1. Frédéric Cuppens. Managing alerts in a multi-intrusion detection environment. In *Annual Computer Security Applications Conference (ACSAC 2001)*, Sheraton New Orleans, Louisiana, USA, December 10-14 2001. ACM.
2. Frédéric Cuppens and Alexandre Miège. Alert correlation in a cooperative intrusion detection framework. In *2002 IEEE Symposium on Security and Privacy (S&P '02)*, pages 187–202, Berkeley, CA, USA, May 12 - 15 2002.
3. Frédéric Cuppens 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)*, volume 1907 of *LNCS*, pages 197–216, Toulouse, France, October 2-4 2000. Springer.
4. Herve Debar 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)*, volume 2212 of *LNCS*, pages 85–103, Davis, CA, October 2001. Springer-Verlag.
5. Jon Doyle, 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. IEEE.
6. John McHugh. Intrusion and intrusion detection. *International Journal of Information Security*, 1:14–35, 2001.
7. Marcus Ranum. Coverage in intrusion detection systems. Technical report, NFR Security, Inc, 26th June 2001.
8. Brian Tung. 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. IEEE.
9. Alfonso Valdes and Keith Skinner. Probabilistic alert correlation. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*, volume 2212 of *LNCS*, pages 54–68, Davis, CA, October 2001. Springer-Verlag.
10. G. Vigna, 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)*, volume 2212 of *LNCS*, pages 69–84, Davis, CA, October 2001. Springer-Verlag.