MSc. Thesis Report

The Meta-Alert Correlation Engine

Advisors : dr. M.E.M. Spruit (TU Delft), ir. R. Prins (Fox-IT)

Melanie Rose Rieback (1113410)

Faculty of Information Technology and Systems
Technical Informatics
Delft
July 11, 2003

The Meta-Alert Correlation Engine (MACE)

Part I – Introduction

1. Introduction

1. 1 Introduction

In the last half century, computers have gone from science fiction to having an important place in global business and society. Computers nowadays are prevalent in high- and low-tech business, education, leisure, medicine, and an almost unimaginable number of other applications. People have come to trust computers to manage their most personal matters – from careers and finances, to communication with family members. Of course, in exchange for that trust, we expect that computers will keep this information private, and away from the eyes of the curious and the meddling.

Intrusion Detection Systems are an important piece of technology employed in keeping computers and their data secure. These systems are a powerful tool for detecting suspicious or anomalous behavior on either the host or network level. However, security professionals have a tough challenge coping with the vast amounts of data (and false positives) that are produced by Intrusion Detection systems. IDS data often comes from multiple sensors, spanning multiple possibly geographically separated networks. There may also be data from multiple brands or types of (ex. host- or signature-based) ID Systems, as well as data from other tools such as traffic and bandwidth monitors. This complexity results in the fact that Intrusion Detection Systems require constant monitoring and maintenance, and that these systems are therefore not easily used by the layman. My thesis project, and this resulting report, will highlight a new approach to help reduce the impact of this fundamental problem.

1. 2 Thesis Objective

My thesis begins with the following two liberal requirements from my advisor:

- Investigate the various methods of intrusion detection, and come up with something that works better.

- Make sure that what you develop reduces the number of false positives and false negatives

Armed with that, I was otherwise completely free to create what I wanted. At the very beginning of my literature study, (April, 2002) I defined my long term objectives as such:

I will develop an intrusion detection algorithm or heuristic, and implement this in a software package. I will create this algorithm based upon the study of suspicious network activity, both in the context of a honeynet, and in a normal live network.

The Meta-Alert Correlation Engine was created to satisfy this objective.

This thesis report itself serves two main functions. First it explores the problem domain, giving a broad view of the current state of Intrusion Detection, and explaining the need for a Metaalert Correlation system. Secondly, this report describes the entire development cycle that has accompanied realization of this project: requirements, designing, implementation, and of course, the final testing and results.

1. 3 Structure of this Report

- **Part I** – Introduction. You are reading this section right now. This section is intended to give a short introduction, an oversight of the thesis requirements, and a brief glimpse into the content of the rest of this report.

- **Part II** – Background. This section gives a broad introduction to the field of Intrusion Detection. It discusses many of the various methodologies and techniques of intrusion detection, providing comparisons and evaluations of the techniques' effectiveness. This section also highlights some of the existing problems and limitations within the field of Intrusion Detection that still need to be overcome.

- **Part III** – Requirements. This section describes some of the general, technical, and user requirements that were established to guide the development of this project.

- **Part IV** – Environment. This section describes the general Intrusion Detection System setup that I am using, as well as the testing setup (a.k.a. where does my sample data come from.) The software development environment is also described, illuminating the tools and platforms that were utilized in conjunction with this project.

- **Part V** – System Design. This section gives an overview of the architectural design of the Meta-Alert Correlation Engine, as well as an in-depth look at each of the components.

- **Part VI –** Remote Modules – This section describes some of the extra modules available with MACE, that provide extra input to aid with metaalert correlation.

- **Part VII –** Metaalert Algorithms. This section describes the "intelligent algorithms" that are used by the Expert System to filter and correlate large amount of Intrusion Detection into a small number of critical metaalerts.

- **Part VIII** – Results and testing. This section details the results and effectiveness of the Meta-Alert Correlation Engine in various circumstances.

- **Part IX** – Conclusion. This section gives a summary of the entire project, and draws conclusions about the worthwhileness of the solution that has been created.

1.4 Acknowledgements

I would like to thank many people that have helped me in various ways throughout this thesis project. First and foremost, I would like to thank my two advisors: Marcel Spruit (TU Delft) and Ronald Prins (Fox-IT). I would also like to thank Lolke Boonstra from the Dienst Technische Ondersteuning (DTO) for giving me access to the TU Delft DuNET network. Thanks to my all of my coworkers at Fox-IT – especially Erwin Fok (for his IDS experience), and Jeremy Butcher (programming tips), Eric Eekhof (network/firewall/system support), Joost Bijl (PHP tips), and Paul Bakker (penetration test tips).

Last but not least, I want to thank my parents, David and Eileen Rieback, the rest of my family, and my wonderful boyfriend René Butter for emotional support and love throughout this entire project. They have listened patiently to more details about this thesis project than they could have possibly been interested in. Thanks for everything!

Part II – Background

2. Introduction to Intrusion Detection

Intrusion detection (ID) is a blanket term for detecting inappropriate, harmful, or anomalous activity on a computer or network. As we saw in the last part of this report, there are many sources of intrusion information that can be used to illuminate what actually happened during a system compromise. However, there are also "real time" intrusion detection systems, that use current process-, network-, and state information to determine if unusual activity is currently in progress.

There are many different sorts of intrusion detection methods, ranging from the historical to the modern. In this part of the report, we will take a look at the different methods and philosophies of detecting intrusive or unusual events. This includes looking at both the theory behind intrusion detection, as well as considering current products on the market.

3. Types of Intrusion Detection

3.1 A Generic Intrusion Detection Model

Dorothy Denning, in 1987, created one of the first theoretical models of intrusion detection. The model is very generic, using templates to represent various system elements, and it is forerunner to the current intrusion detection technology that we use today. Denning describes a "general-purpose intrusion-detection expert system" called IDES, that uses rule-based pattern matching to determine anomalous user behavior based upon profiles of typical user behavior. This model, in order to be independent of system or intrusion type, heavily uses templates to represent the various parts of the system.[4] According to Dorothy Denning's paper, "An Intrusion Detection Model", the model has six components[4]:

- **Subjects:** Initiators of activity on a target system – normally users
- **Objects:** Resources managed by the system files, commands, devices, etc..
- **Audit Records:** Generated by the target system in response to actions performed or attempted by subjects on objects – user login, command execution, file access , etc…
- **Profiles:** Structures that characterize the behavior of subjects with respect to objects in terms of statistical metrics and models of observed activity. Profiles are automatically generated and initialized from templates.
- **Anomaly records:** Generated when abnormal behavior is detected
- **Activity rules:** Actions taken when some condition is satisfied, which update profiles, detect abnormal behavior, relate anomalies to suspected intrusions, and produce reports.

The following figure depicts the architecture of Dorothy Denning's generic intrusion detection model: [1]

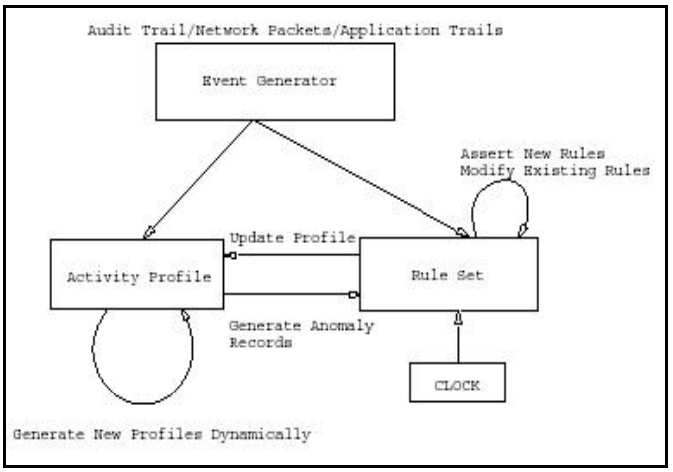


Figure 1 – Dorothy Denning's Generic Intrusion Detection Model

The intrusion detection model has several capabilities that modern intrusion detection systems currently possess. It can generate statistical models, such as the Mean and Standard Deviation Model, Multivariate Model, or Markov Process Model. It is capable of breaking down audit records into atomic system calls, and reporting information on both a periodic and an event-caused basis. Additionally, the model has capabilities of automating such tasks as: responses to anomalous events ("activities"), and adding new users or objects.[4]

3.2 Anomaly Based Intrusion Detection

Anomaly-based intrusion detection systems look for perturbations or deviations in normal behavior on a computer system, suggesting the presence of attacks, system errors, etc.. Most anomaly detection techniques use probabilistic algorithms to analyze data collected in various logs on the system. However, these algorithms must have a baseline "normal activity profile", to know if system activity is unusual. Based upon this profile, all system states that vary by a statistically significant amount are flagged as suspicious events. Figure 2 shows a diagram of a typical anomaly detection system.[3]

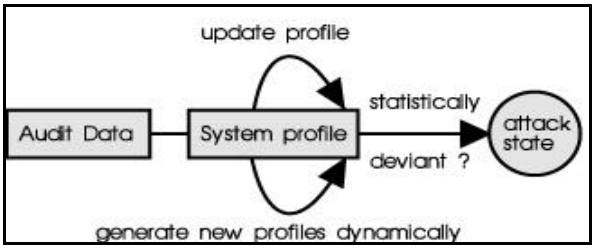


Figure 2 – Typical Anomaly Detection System

Anomaly-based intrusion detection leads to two major problems. Anomalous activities that are not intrusive are sometimes incorrectly labeled as intrusive, causing a "false positive". On the other side of the coin, not all intrusive activities are anomalous, so some intrusive activities might not be labeled at all, causing a "false negative". False negatives are a far more serious problem than false negatives, since false positives can always be later sorted through.[3]

Therefore, anomaly detection systems become a matter of selecting statistical thresholds that can eliminate baseline "noise" (acceptable anomalous user activity) without accidentally ignoring intrusive activities. Anomaly detection systems also periodically update their activity profiles, so that the system can "evolve" with changing (acceptable) user behavior. A disadvantage to this kind of system is that maintaining profiles and performing statistical analysis can be computationally expensive.[3]

3.2.1 Statistical Anomaly Detection

Statistical anomaly detection is focused upon the generation and maintenance of behavior profiles. While initial profiles are generated for each subject, the system gradually and continuously updates the profiles based upon a number of behavioral factors. Some of these factors may be: activity type, CPU usage, network connections, etc.. Part of the difficulty of using statistical anomaly detection systems is selecting which factors to monitor in order to accurately measure intrusive activities. These factors can be determined partially by past experience, and can statically or dynamically modified in the system to try to reduce false negatives and positives.[3]

If statistical behavior profiles are updated, these statistical anomaly detection systems will adaptively "learn" the behavior of each user. This can be both a positive and negative characteristic. While the system can create a more insightful profile of user behavior, it can also be gradually trained by intruders to give false negatives to malicious intrusive activities. This statistical method also has a disadvantage in that it cannot take advantage of relationships between or the chronology of events.[3]

3.2.2 Feature Selection

As mentioned in the previous section, selection of features to monitor poses a problem in anomaly-based intrusion detection. Only a subset of heuristically chosen measures detect actual intrusions, and this appropriate subset depends largely on the type of intrusions that will occur. Additionally, different operating systems often require different combinations of feature information to detect intrusion.[1]

The determination of an optimal combination of features can be done in several ways. One possible way is to use a "brute force" method, trying every possible combination of features to find an optimum. This exhaustive search is inefficient for large numbers of features, as the number of possible subsets is equal to the power set of the total number of factors. Another method is to use genetic learning to adapt the space to the right combination of features. This approach generates an initial set of features, that is

gradually refined by a learning scheme that uses the genetic operators of crossover and mutation. Subsets of features that have a low probability of intrusion detection are weeded out, and are replaced by genetic operators that can yield stronger subsets of features. The efficiency of this approach, as opposed to the use of brute-force, is largely determined by the size of the feature space, and well as the predictability measures of the various subsets.[1]

3.2.3 Predictive Pattern Generation

Anomaly-based intrusion detection systems can use rule-based sequential pattern analysis to predict future events, based upon known events that the system has already witnessed. In "An Introduction to Intrusion Detection", Aurobindo Sundaram illustrates this approach by giving the following rule:[3]

E1 - E2 --> (E3 = 80%, E4 = 15%, E5 = 5%)

This means that if we have event E1, followed by event E2, E3 may occur with an 80% probability, E4 with 15%, and E5 with 5%.

There are several advantages to this approach, as opposed to the more traditional statistical methods. First, this approach provides more support to the system in the early stages of the learning period. Since we begin with a standard set of rules and probabilities, it is easier to find intruders that are attempting to "train" the system. However, despite this initial ruleset, this system retains all of the needed flexibility and adaptiveness, as the rule based patterns are continuously updated and refined, weeding out lower quality patterns. Another advantage is that, since the rulesets are continuously maintained, anomalous activities can be detected and reported immediately upon happening.[3]

The predictive pattern generation method also has some disadvantages. One of the problems is that intrusion scenarios must be described by one of the rules in order to be detected as intrusive. This gives an interesting dilemma of what to do with unknown events, that are not described by any existing rules. They can be flagged as intrusions, causing them to be added to the database, but increasing the number of false positives. On the other hand, they can also be ignored, increasing the number of false negatives. In the normal functioning of the system, events that match the left side of a rule, but deviate statistically from the right side, are flagged as intrusive.[3]

3.2.4 Neural Networks

Neural Networks are a form of multiprocessor computer system that are characterized by the following elements:[5]

- simple processing elements
- a high degree of interconnection
- simple scalar messages

- adaptive interaction between elements

The idea is that each element in the neural network behaves like a biological neuron in the human brain, accepting a large number of inputs, and producing a single output to many other neurons. These networks are trained, by adjusting weight values, to recognize or reproduce certain patterns.[5] This functionality makes them well-suited for recognizing anomalous events in intrusion detection.

The neural network is trained to predict a user's next action, based upon a history (or training set) of past actions. After the training period generates a user profile, the network then compares actual user input with the profile. This produces a statistical deviation value, flagging any anomalous behavior. Some advantages of neural networks are that they deal well with noisy data, their ease of adaptivity to new users, and their lack of statistical assumption about the underlying data. Some disadvantages are that a considerable amount of time is required to train the network, and that an intruder can train the network to produce false negatives if he has access during it's learning phase.[3]

3.2.5 Baysian Classification

A new technique, Bayesian classification, classifies unaffiliated data into a number of data classes, using Bayesian statistical techniques. These techniques try to determine an optimal number of classes, grouping users with similar profiles, and than assigning a probabilistic membership function to each new user in the system. Bayesian classification is a kind of statistical intrusion detection, heavily relying upon computation of probabilistic values to classify observed behavior.[1]

This approach is still new, and has not yet been implemented and tested. Therefore, it is not yet clear how well the method handles chronological and incrementally increasing data. Additionally, being a statistical technique, it suffers from some of the same disadvantages as other statistical anomaly based intrusion detection systems – namely, finding good threshold values, and intruders gradually "training" the system.[1]

3.2.6 Limitations of Anomaly Detection

First, anomaly-based intrusion detection systems tend to be computationally expensive. However, the largest problem inherent with anomaly detection systems is the assumption that the set of intrusive activities is a subset of anomalous activity. In practice, this is not necessarily true. Attackers can take over a user's account, using the same kinds of commands that the user would normally use. On the other hand, individual users sometimes display anomalous behavior without actually pursuing any kind of intrusive behavior.[1] False negatives and false positives. This is one of most persistent unresolved problem of intrusion detections systems of all kinds.

3.3 Misuse Detection

Misuse-based intrusion detection systems compare all activities on a computer system to a library of known attacks, looking for intrusive behavior. These systems are a bit like virus detection systems – using attack "signatures" to store the details of various attack patterns. Signatures must be able to match both attacks and variations upon known attacks, without causing false positives by accidentally matching non-intrusive events. Misuse detection schemes can only detect *known* attack patterns – therefore one of the main issues in misuse detection is creating a comprehensive library of known attacks.[3]

A typical misuse detection system is shown below in Figure 3.[3]

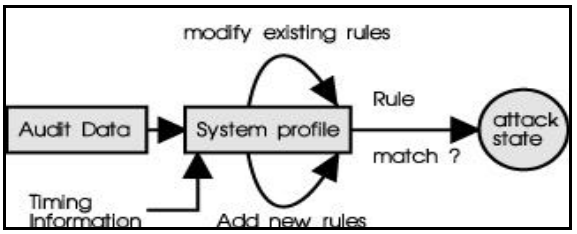


Figure 3 – Typical Misuse Detection System

Misuse-based intrusion detection can be used together with anomaly-based intrusion detection, to make a more robust intrusion detection system. This combined system could detect attacks that each method would miss individually.[1]

3.3.1 Pattern Matching

Pattern matching-based intrusion detection works by comparing events on a computer system against libraries of known intrusions. Known attacks and problems are represented in various "patterns" to be matched by the system. These patterns may be composed of individual events, sequences of events, thresholds, and combinations or these using boolean operators (AND/OR/NOT). Attack patterns are usually compiled from sources of information security knowledge, such as CERT advisories, and corporate and individual experiences. Unlike a virus scanner, pattern matching engines do not need to be updated for new attacks – only for new *classes* of attacks. Patterns should be defined generally enough to match variations on common hacks and security problems. These patterns should ideally match attack classes, regardless of which software contains the security hole.[2]

Pattern Matching intrusion detection has the following advantages:[2]

- Events monitored are only monitored if they match a pattern appropriate for the computer system. This means that if the IDS is on a web server, if won't have to check the patterns for a mail server.

- Pattern matching is more efficient than statistical analysis, due to the absence of floating point calculations.

Pattern Matching intrusion detection has the following disadvantages: [2]

- Scalability and performance depend upon the size and architecture of the pattern database. For large databases, this is more of a problem.

- Pattern databases are difficult to extend with new attack signatures, since the format is not standardized.

- Patterns, while more flexible for catching new attacks than virus scanners, still need to be updated frequently. If the database is not updated, new attacks may not be caught by the system.

- Machine learning is not utilized in pattern matching systems. It could theoretically be added by IDS vendors - but it has never been done. If added, Artificial Intelligence could add new patterns to the database as new attacks are "learned".

- Attacks may be difficult to translate from natural language into a pattern. Therefore, new patterns must be extensively tested to guarantee that attacks are detected, and that false positives are not produced.

3.3.2 Conditional Probability

This method is similar to probabilistic analysis in anomaly-based intrusion detection. The largest difference of misuse-based conditional probabilistic analysis is that the data analyzed is external events, instead of anomaly measures. We can find the conditional probability by using the following formula:[1]

$$P(\text{Intrusion} \mid \text{Event Pattern}) = P(\text{Event Pattern} \mid \text{Intrusion}) \frac{P(Intrusion)}{P(EventPattern)}$$

3.3.3 Expert Systems

Expert systems consist of a predetermined rulebase that encodes known intrusion and attack scenarios into rules. Events are then activated by actions on the system that match one of the rules in the ruleset. The rule database can be changed for different operating systems and computer uses (ex. mail/web server.) One of the main difficulties with expert systems is that, similar to misuse patterns, the rules must be formulated and tested by security experts. Additionally, new rules must remain consistent with the interdependencies between other rules in the ruleset. Addition and deletion to the rulebase can be performed automatically by other intrusion detection methods, such as anomaly-based statistical methods. One system that integrates anomaly detection with a

misuse-based expert system is NIDES (Next Generation Intrusion Detection Expert System), produced by SRI. The misuse- and anomaly detection portions of NIDES work together to flag intrusions that would be missed by the individual components.[3]

3.3.4 State Transition Analysis

In state transition analysis systems, attacks are represented as a sequence of transitions between states on a target system. Each of the states also has associated requirements that must be satisfied in order to allow transition. These requirements are illustrated as arcs between different states. An advantage of state transition analysis is that event types are independent of system-type, and are built directly into the model. A disadvantage is that attack patterns are limited to sequences of events, instead of being able to represent more complex types of attacks. Also, there is no easy way to find partial matches to a given attack pattern.[1]

3.3.5 Keystroke Monitoring

Keystroke monitoring, as the name suggests, captures keystrokes on a terminal or computer and analyzes them for known attack patterns. The analysis of keystrokes is entirely independent from the analysis of the underlying application programs, so this technique could be augmented with other intrusion detection methods. One disadvantage to keystroke monitoring is that definable user aliases (as offered in several of the prominent UNIX shells) can defeat this technique. To avoid this, alias expansion and semantic analysis should included in the keystroke analysis process.[3]

3.3.6 Model-based

In model-based misuse detection, attack scenarios are inferred by looking at other observable activities. This scheme consists of three separate modules. The "anticipator" uses scenario models (knowledge bases with intrusion scenario specifications) to try and predict the next activity that will occur. The "planner" turns this behavioral prediction into a sample audit trail. Lastly, the "interpreter" searches the actual audit trails for data similar to that generated by the planner. The system uses these three modules to calculate intrusion probabilities, and an alert is sounded when the attack likelihood percentages bypass a given threshold.[3]

The model-based approach has some advantages:[3]

- Since the planner and interpreter only deal with a limited number of system activities, noise in the audit data is reduced, and general performance increases.
- If there is indeed an attacker, the system can predict his next move.

It also has some disadvantages:[3]

- Intrusion scenario patterns must be easily recognized
- Patterns must always occur in the actual attack (or we get false negatives)

- Patterns cannot be associated with normal system function (or we get false positives).

3.3.7 Limitations of Misuse Detection

Misuse detection has several primary limitations, the largest of which is that it can only look for attack patterns that are already known within the system. New attack classes are constantly developing, and misuse detection can do nothing to stop them until an attack signature has been developed. Another limitation is the difficulty in capturing all possible variations on an attack-class in an attack signature. Hackers constantly come up with new ways to hide old attacks, and the attack signatures may not always detect these variations. This leads to a perpetual game of catch-up, as hackers develop new attacks and variations thereof, and misuse detection experts develop new signatures after a number of computer systems have already been hacked.[1]

Another problem with misuse detection is that it can be very demanding of system resources. Audit trails can not record information for every program or process variable since the space and memory limits are quickly exceeded by the flood of information. Less resource intensive deductive methods can be used, such as predicting the future values of system variables. However, this oftentimes requires intrusive hacks, worked into the program source code in order to access the internal variables. (This may also exceed memory limits.) Additionally, these predictions are often inaccurate, leading to false positives, false negatives, or both.[1]

Finally, some techniques cannot be reliably detected by both anomaly- and misuse-based intrusion detection. Some of these techniques include passive sniffing, and IP address spoofing, that causes the attack events to seem to originate from a different source location. The whole approach of intrusion detection relies on the integrity of event data.[1]

4. Network vs. Host Based Intrusion Detection

Besides the division between anomaly- and misuse based intrusion detection, there are also two other commonly used ID categories: network- and host based. Many intrusion detection products focus upon only one category, while a few integrated products combine them. We will discuss and compare each category of intrusion detection in the following chapter.

4.1 Network Based Intrusion Detection

Network-based intrusion detection typically uses a network adapter, running in promiscuous mode, to "sniff" all of the raw data packets from the network segment in real-time.[6] These sniffed packets are then sent to the device driver, which subsequently sends the packets to the Intrusion Detection System for analysis.[2] The Intrusion Detection System commonly uses the following techniques to look for an attack:[6]

- Pattern, expression, or bytecode matching
- Exceeding frequency or threshold limits
- Correlation of minor events
- Statistical anomaly detection

Once an attack is detected, system administrator, automatically taking action (such as dropping the network connection), and initiating expanded logging features for later forensic and legal analysis.[6]

To effectively detect attacks, network sensors need to be placed in strategic locations along the network. For example, the first node after the router in the subnet is ideal for catching inbound subnet packets. The network sensors are also commonly placed along gateways between different subnets, or immediately behind the firewall in enterprise systems.[2]

Network IDS's should not affect network performance, although they can sometimes become overwhelmed themselves by the bandwidth on large networks. This potentially troublesome situation can cause dropped packets, resulting in false negatives.[2]

4.2 Host Based Intrusion Detection

Host-based Intrusion Detection, first implemented in the early 1980's, is the manual or automatic monitoring of changes in security logs and the filesystem on a single computer. When any file changes, the IDS checks to see if the new entry or file matches a known attack pattern. If such a match occurs, the system responds by notifying the system administrator, or by carrying out other automatic responses. (ex. drop carrier)[6]

As time goes on, host-based intrusion detection gets increasingly sophisticated. While the host-based IDS still uses audit logs, a greater amount of the functionality and analysis tool are automated. Host-based intrusion detection systems have also started integrating new technologies, such as cryptographic checksums and simple network-based intrusion detection additions, such as port access alerts. The responses of host-based intrusion detection systems are also becoming increasingly timely, as automation allows the frequency of polling intervals to increase.[6]

4.3 Comparison

4.3.1 Network-Based ID

Here are some of the advantages of using network-based intrusion detection systems:[6]

1. **Low cost of ownership** – Network based intrusion detection systems usually use a limited number of sensors in the network, creating less installation and

management costs. Additionally, there are a number of free open-source network-based IDS's available on the market.

2. **Detects attacks that host-based systems miss** – network-based IDS can analyze packet headers and contents. Looking at packet headers enables detection of many IP-based attacks (ex. DOS, teardrop attacks). Additionally, the packet contents can be scanned for "payloads", or specific commands that are used in various attacks. Host-based IDS can not detect any of these attacks until after they somehow damage or alter the filesystem.

3. **More difficult for an attacker to remove evidence –** Network based IDS captures information from the network packet stream real-time, making it difficult for a hacker to alter the captured data, and "cover his tracks". Host-base IDS use audit trails, which can be subsequently modified by the attacker if they are not copied to another computer system quickly enough. This information makes it easier to capture and persecute the hacker.

4. **Real-time detection and response** – Network based IDS can recognize attacks while they are occurring, allowing a quick response from the system administrator. Detecting attacks real-time often minimizes the damage that a hacker causes, and can stop compromises before they actually occur. Host based systems cannot respond to attacks until suspicious changes to system logs and the file system have already taken place. By this time, it may be too late, and the damage might have already been done. Real-time intrusion detection also gives the system administrator the opportunity to perform surveillance on the attacker (ex. in the context of a honeypot or honeynet.)

5. **Detects unsuccessful attacks and malicious intent –** Network based IDS can gather information about what happens outside of the firewall. In this way, the network-based IDS can detect unsuccessful attempts to penetrate system security, while host-based IDS would not have noticed it, since none of the files on the system itself were changed.

6. **Operating system independence** – Network-based IDS are less dependent upon the specific operating system type than host-based IDS. Additionally, many of the popular network-based IDS products have been ported to various OS's, or have been written using a platform-independent interface. (ex. Java)

4.3.2 Host-Based ID

Here are some of the advantages of using host-based intrusion detection systems:[6]

1. **Verifies success or failure of an attack** – Host-based IDS use logs of events or changes that actually occurred on the target system. While network-base IDS may provide early-warning capabilities, host-based IDS can actually verify to

what extent the attack was successful (or damaging). This decreases the number of false positives.

2. **Monitors specific system activities** – Host-based IDS monitors specific events and activities on the system, including file accesses, permissions changes, new files, and privilege violations. Network-based IDS do not provide such a low level of detail on system activities. Host-based IDS also log the activities of users with administrator permissions, including the addition/deletion/modification of user accounts. Host-based IDS also monitor changes to the security policy itself, and can sometimes stop the installations of trojans, backdoors, or viruses as soon as it is detected.

3. **Detects attacks that network-based systems miss** – Host-based IDS may detect attacks that network-based IDS may miss. These attacks include attacks launched from the console of the server itself, or other attacks that never have a reason to be sent across the network.

4. **Well-suited for encrypted and switched environments –** Host-based IDS can reside on various hosts across a switched and encrypted environment, avoiding deployment challenges of network-based IDS in such environments. These host-based systems can reside on as many critical hosts as are needed. Additionally, network-based IDS sometimes have problems identifying attacks that are carried encrypted through a network. Host-based IDS overcome this limitation, because by the time they see the data stream, it has already been decrypted.

5. **Near-real-time detection and response –** Although host-based IDS is not quite as "real-time" as a network-based IDS, it can still come quite close to real-time if implemented correctly. Host-based systems, instead of using predefined intervals, can check the content and status of logfiles immediately after modification, using system interrupts to alert the IDS. There is still a delay between logfile modification and host-IDS recognition, but in many cases, it is small enough to prevent the attack from progressing any further.

6. **Requires no additional hardware** – Host-based IDS can reside on existing system infrastructure, so no extra hardware is necessary to purchase or maintain. This can significantly cut costs compared to network-based IDS.

7. **Lower cost of entry** – Host-based IDS tend to be less expensive to deploy than network-based IDS. This is due to both the requirement for new hardware, and the general cost of the IDS system itself (not counting open-source IDS's).

4.3.3 Integrated Approach

The ideal solution for many computer systems is to use an integrated system – having both host- and network-based IDS components present. This integration is ideal because,

as seen in the last sections, each type of IDS has strengths and weaknesses. An integrated solution allows for a "best of both worlds" solution, that offers greater network resistance to attacks, and greater flexibility in implementing the system's security policy.[6]

Figure 4 lists some of the advantages that each type of IDS has to offer a total integrated IDS solution:[6]

	Network-Based IDS	Host-Based IDS
Notification	Alarm to Console	Alarm to Console
	E-Mail Notification	E-Mail Notification
	SNMP Trap	SNMP Trap
	View Active Session	
Storage	Log Summary (Reporting)	Log Summary (Reporting)
	Log Raw Network Data	
Active	Kill Connection (TCP Reset)	Terminate User Login
	Re-Configure Firewall	Disable User Account
	User Defined Action	User Defined Action

Figure 4 – Integration of Network- and Host Based IDS Functionality

5. Limitations of Intrusion Detection

After having examined many different types of Intrusion Detection Systems and their methodologies, here are some broad limitations that the ID field as a whole still needs to address:[1]

- **No Generic Building Methodology:** Building Intrusion Detection Systems is still difficult and expensive because of a lack of structured methodology. Lack of agreement on intrusion detection techniques and tools hinders the development of such a methodology.

- **Efficiency:** Many Intrusion Detection methods are computationally expensive, and need extensive system profiles and libraries of attack signatures. Also, some kinds of Intrusion Detection Systems are also implemented using expert systems, causing a high runtime overhead and limiting the representation of possible relationships between events.

- **Portability:** Most intrusion detection systems have been written for single environments, and have proved difficult to port to others. This limits the reuse and retargeting of intrusion detection systems. It is difficult to get rid of these specific system customizations, as some are currently necessary to detect certain kinds of attacks in certain computing environments.

- **Upgradability:** It is often difficult to integrate newer and better intrusion detection techniques with older existing intrusion detection systems.

- **Maintenance:** Maintenance of intrusion detection systems requires a high level of security knowledge from the administrator. Some specialized non-security knowledge necessary may include expert system rule language, and statistical calculating methods. This specialized knowledge makes administering intrusion detection systems difficult and costly for the average system administrator.

- **Performance and Coverage Benchmarks:** There are not many realistic sets of intrusion and vulnerability data, nor many published reports of intrusion detection coverage on various systems. Vendors tend to treat intrusion coverage qualitatively, since it is very difficult to predict new attacks and their frequency in large organizations.

- **No Good Way to Test:** There is no standard way of testing intrusion detection systems. Potential attacks are difficult to simulate, and the lack of a common audit trail format makes it difficult to compare the performance of existing systems in common attack scenarios.

Part III – Requirements

6. General Requirements

As mentioned in the introduction, the primary requirements of this thesis are ideally the following:

- Investigate the various methods of intrusion detection, and come up with something that works better.

- Make sure that what you develop reduces the number of false positives and false negatives

7. Metaalert Functionality

During the early design process of this project, my colleagues at Fox-IT and I had a "brainstorming session" to come up with a list of desired functionality regarding Metaalert creation. The following list is a collection of desired (but not fully implementable in the short-term) Metaalert correlation behavior:

- **Potentially successful attacks –** A metaalert should be generated when an attack is directed towards a machine that is vulnerable for that specific attack
- **Slow typing metaalert –** A metaalert that can detect if someone is providing hand-typed input for a service (ex. while logged in to a port) that normally accepts batch input from programs.
- **Recognized combinations of (meta-)alerts –** A metaalert that recognizes combinations of other alerts/metaalerts that identify a specific break-in scenario. (Ex. Portscan + known exploit against vulnerable server + high levels of new outgoing traffic.)
- **Multi-sensor Alerts –** A metaalert generated when several sensors belonging to the same company/organization are targeted. A multi-sensor alert should be suppressed if the alerts are also recognized as a port scan.
- **Too many unique signatures –** A metaalerts is generated when too many unique signatures are generated within a specified time-interval.
- **Tool/Worm specific metaalerts –** Specific groups of alerts are often indicative of activity of a single tool/exploit/worm. A metaalert should provide this summary.
- **Alert followed by traffic policy violation** – An alert followed by a traffic policy violation should generate a higher priority metaalert.
- **Seldom seen signatures –** Some alerts appear so infrequently that they are a sure sign that something suspicious is going on. This should generate a metaalert.
- **Slow portscans –** A metaalerts should be generated when a set number of metaalerts come from one source in a "long" time interval.

- **Victim Specific Port Scans –** A metaalert should be generated when more than a certain number of ports are scanned on a specific victim machine in a set time interval.
- **Short SSH Sessions –** A number of short prematurely broken-off SSH sessions may indicate that someone is trying to "brute force" the password.
- **Possible covert channels** – Generate a metaalert when replies for certain protocols (ex. ICMP) appear without the presence of the associated request.
- **Anomalous protocol behavior –** Generate a metaalert when a server suddenly and unexplicably starts using a new protocol that was previously infrequently or never used.
- **Arpwatch metaalerts** – New MAC/IP address combinations may indicate spoofing or wardriving.
- **Flow-based metaalerts –** Alerts should be accepted that provide information about bandwidth, protocols, ports, and anomalous activity.

It is desirable that the Meta-Alert Correlation Engine can accept input from several different sources (i.e. NIDS alerts, syslog alerts, nessus input, bandwidth monitors, anomaly detection IDS alerts, etc..) Also, it would be ideal to eventually integrate alerts from the following other tools: HTTP Insertion Processors, URL analyzers, Statistical Analysis Modules (page hashes, page size, n-gram analysis, content-type), and modules to detect anomalous database activity.

8. Technical Requirements

While not too many of the technical details are "official" requirements, I have strived to follow certain technical guidelines in creating the Meta-Alert Correlation Engine. These guidelines include:

- Object oriented and modular system design
- Use of STL, and other data structures that are already validated by the developer community.
- Thoughtful coding style (according to GNU coding standards)
- Use of an interoperable alert/metaalert communications interface
- Eventual portability

I have also worked to make the system as easy-to-use as possible for the end-user. It is not expected that the end user will have advanced computer skills, so most of the system processing and management should be accessible via interfaces such as the WWW Management interface.

Part IV – Project Environment

9. Intrusion Detection Setup

9.1 Snort

Snort is an open-source, lightweight, network intrusion detection system.[12] There are three main ways in which Snort can be used: as a packet sniffer, a packet logger, and a Network Intrusion Detection System (NIDS). Snort's sniffer mode simply reads the packets from a network interface and sends them in a continuous stream to the console. Snort's packet logger mode logs all of the sniffed packets to the disk. Snort's Network Intrusion Detection mode is the most complex (and perhaps useful) configuration, enabling Snort to analyze network traffic for matches against a predefined rule set describing suspicious "intrusive" activity. Snort is also capable of performing actions based upon what it sees.[10]

There are a number of advantages to using Snort as a network Intrusion Detection System:[12]

- Snort is open source (GNU General Protection License)
- Snort is free.
- Snort's rule language is easy to use
- Rules can be customized to detect new exploits and to utilize specific properties of a network that is being monitored.
- Snort (and the rulebase) is actively supported by a large user community.
- Snort is available for many various OS's, including *NIX, and Windows.

9.2 MySQL

MySQL is an open-source SQL database that is developed, distributed and supported by the company MySQL AB. MySQL is a relational database management system. "Relational" means that data is stored in separate tables, as opposed to keeping all of the data in one centralized location. Tables are then linked by defined references, allowing this relational data to be easily combined through use of "Structured Query Language" or SQL.[13]

The following list describes some of the main advantages of using MySQL as a database management system:[13]

- MySQL software is Open Source (GNU General Protection License)
- MySQL, written in C and C++, has been tested with a broad range of different compilers, and has been demonstrated to work on many different platforms.
- Uses GNU Automake, Autoconf, and Libtool for portability.
- MySQL has APIs for C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, and Tcl.
- MySQL offers multiple thread and multiple CPU support.
- MySQL is efficient. It uses fast B-tree disk tables with index compression.

MySQL is also one of the three database systems (MySQL, Postgres, Oracle) that is fully supported by Snort and ACID (see section 9.4)

9.3 Stunnel

Stunnel is an open-source program that encrypts network connections using the Secure Sockets Layer (SSL). By using Stunnel, non-SSL aware programs and protocols can encrypt sent data without requiring any changes to the source code. Stunnel does not provide the cryptographic code itself – it uses external SSL libraries such as OpenSSL or SSLeay instead. Both of these SSL libraries are capable of strong (128 bit) cryptography, and Stunnel uses the highest strength available to both the client and the server. Stunnel works by receiving encrypted data on a specific port, and sending it to the SSL server. Then, the decrypted data is sent to an arbitrary port on that or another machine.[14]

Stunnel supports the following functionality:[14]

- SSL client
- SSL server
- Server and client side certificate verification
- TCP wrapper support
- IDENT lookups
- SMTP protocol negotiation
- Source address rewriting (transparency)
- Restricting allowed SSL ciphers

I use Stunnel to encrypt the communications between Snort and the MySQL database. I do this so that attackers cannot glean sniffed information (like usernames and passwords) from watching the Snort output as it is transported to the database machine.

9.4 ACID

Analysis Console for Intrusion Databases

ACID, the Analysis Console for Intrusion Databases is an open-source PHP-based analysis engine that can search through and summarize a database of security events generated by various IDS's (including Snort), firewalls, and network monitoring tools.[15] The ACID graphical user interface is shown in Figure 5.

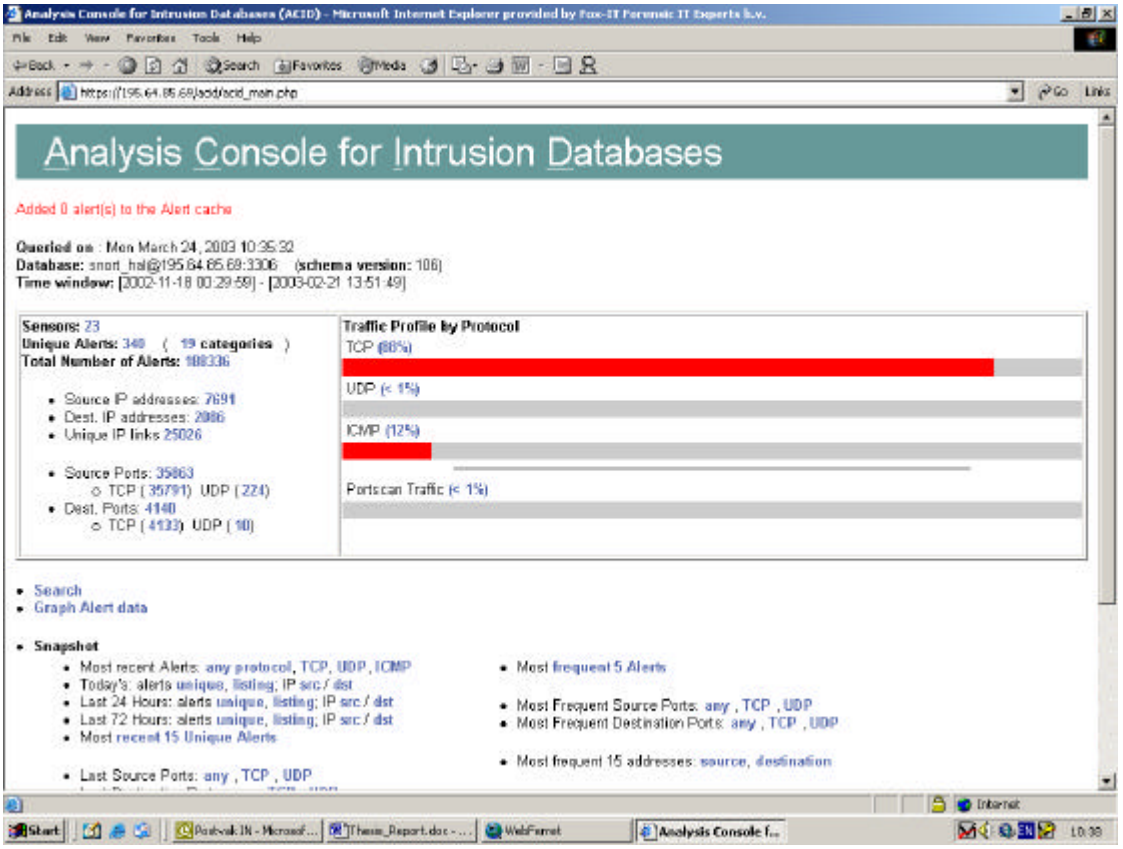


Figure 5 – ACID Graphical User Interface

ACID has the following features:[15]

- **Query-builder and search interface.** Alerts can be searched based upon specified alert meta information (ex. signature, detection time) as well as by the underlying network statistics (ex. source/destination address, ports, payload, or flags).
- **Packet viewer (decoder).** ACID will graphically display the layer-3 and layer-4 packet information for the alerts.

- **Alert management.** Alerts can be logically grouped into incidents (alert groups), and they can be deleted, exported to email, or archived to another alert database by a single click on the Graphical User Interface.
- **Chart and statistics generation.** ACID can create charts based on time, sensor, signature, protocol, IP address, TCP/UDP ports, or classification.
- **Adaptability.** ACID has the ability to analyze data in a wide variety of formats. Tools exist for Snort alerts, tcpdump binary logs, and logsnorter, ipchains, iptables, and ipfw logs.
- **Open Source.** ACID is freely available under the GNU General Protection License.

I use ACID to graphically view the alerts that originate from the Snort alert databases.

10. Testing Environment

10.1 DUNET-Database

The Technical University of Delft (TU Delft) has a 2x10 Gbps network that connects various research and academic faculties, student houses, and various companies to the Internet. The connections within the DUNET are provided with level-2 switches, and the network is divided into several virtual subnets.[22] The DUNET Technical Support Service (Dienst Technische Ondersteuning) was kind enough to grant me access to a spanport along a busy junction of the network, and with the assistance of system administrator Lolke Boonstra, a 1 Gbps sensor was attached to the spanport. The sensor was running Snort, and was sending the alerts to an external MySQL database, via an encrypted TCP connection, using Stunnel. Figure 6 shows a general view of the DUNET network.

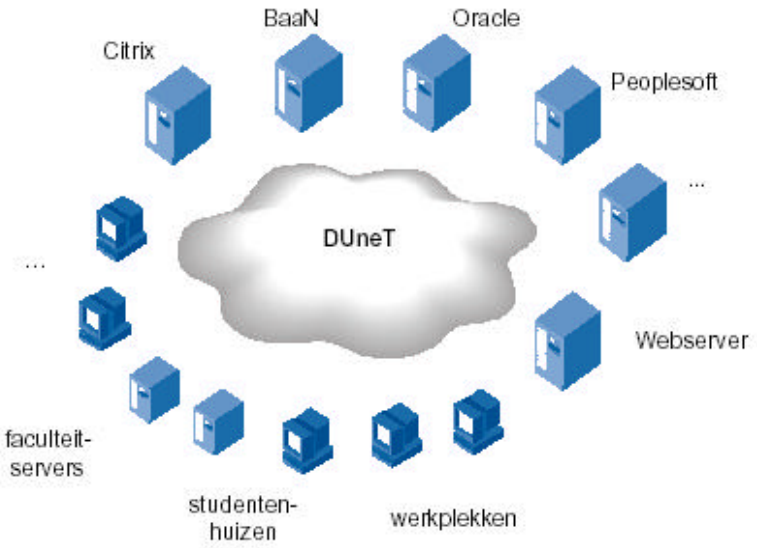


Figure 6 – DUNET Network Overview [22]

Alerts were collected over a period of several months, and more than 2 million Intrusion Detection alerts were eventually stored in the MySQL database. These alerts were used as a raw input to help devise some of the correlation algorithms that are implemented in the MACE tool.

10.2 Fox-IT Hal Database

Fox-IT (Forensic IT Experts), the Information Security company that I am working in conjunction with on this project, was also kind enough to grant me access to their databases of client intrusion detection data. Thanks to Ronald Prins and Erwin Fok, I received a supply of Snort databases with real intrusion data. This intrusion data was also analyzed for intrusions by a human Security Operations Center (SOC) operator, making the data instrumental in creating the meta-alert correlation algorithms.

11. Software Development

11.1 Operating System

11.1.1 OpenBSD

The OpenBSD project produces a free, open-source, multi-platform 4.4BSD-based UNIX-like operating system. OpenBSD is built with an emphasis upon portability, standardization, correctness, proactive security and integrated cryptography. OpenBSD supports binary emulation of most programs from SVR4 (Solaris), FreeBSD, Linux, BSD/OS, SunOS and HP-UX.[16]

The following list describes some of the main advantages of using OpenBSD as a development platform:[16]

- **Open-Source** – OpenBSD is available under the Berkeley license.
- **Portable** - OpenBSD runs on 10 different commonly-used hardware platforms.
- **Secure** - OpenBSD has undergone a 10-member 1.5-year long comprehensive source code security audit.
- **Cutting Edge** - OpenBSD has strong ongoing development in many areas, providing access to emerging technologies with an international community of programmers and end-users.

I am doing all of my software development and testing using OpenBSD 3.2.

11.2 Programming Languages

11.2.1 C++ (w/ STL)

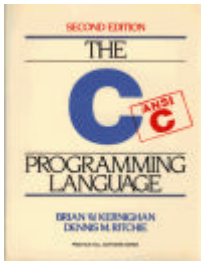
Bjarne Stroustrup, while working at AT&T, developed C++ in order to add object oriented constructs to the C language. Object oriented technology was new at the time, and the primary goal of C++ was to preserve the efficiency of C, while offering this new functionality. A well written C++ program reflects elements of both object oriented programming style and classic procedural programming. C++ is an extendable language in which we can define new data types and "objects" in such a way that they act like part of the standard language. C++ is well suited for large scale software development. [18]

The Standard Template Library (STL) is a C++ library of container classes, algorithms, and iterators. It provides a generic and robust library of the basic algorithms and data structures of computer science. Almost every component in the STL is implemented as a template. Like many class libraries, the STL includes *container* classes: classes whose purpose is to encapsulate other objects. The STL includes the following container classes: vector, list, deque, set, multiset, map, multimap, hash_set, hash_multiset, hash_map, and hash_multimap. STL container classes are used in much the same way as you would use

your own data strucutres, except that it manages details like dynamic memory allocation automatically.[17]

I have chosen to use C++ as my primary programming language for this project because its object oriented technology assists me in designing and implementing a strongly modular system. I have chosen to make heavy use of the Standard Template Library, because implementation of basic data structure is not my primary objective, and because my own implementation of these data structures would be more likely to be susceptible to security “holes”, or subtle programming errors that can compromise the integrity of the system.

11.2.2 C



The C programming language was developed at AT&T, originally intended as an operating system for the PDP-11 series of computers (which later developed into UNIX). The primary goal of C is operating efficiency. C was originally defined by the classic text “The C Programming Language”, by Kernigan and Ritchie, and this was the standard used by all C programmers up until recently. The ANSI standard for C was approved in December 1989, and this is now the official standard for programming in C.[18]

I use C in a few parts of my program – specifically, the sections that interface with external libraries written in C (CLIPS, libxml, libidmef) or sections utilizing dynamically loadable plugins.

11.2.3 PHP/HTML

HyperText Markup Language (HTML) is a non-proprietary format based upon SGML, and is the dominant publishing language of the World Wide Web. HTML uses tags (such as <h1> and </h1>) to structure text into headings, paragraphs, lists, hypertext links, and other structures. In addition to text, multimedia, and hyperlink features, HTML also supports multimedia options, scripting languages, style sheets, printing facilities, and documents that are accessible to users with disabilities. HTML strives towards the internationalization of documents, with the goal of making the Web truly World Wide.[20]

The “PHP: Hypertext Preprocessor" (PHP) is a general-purpose scripting language that is specifically designed for Web development and easy embedding into HTML. Its syntax is similar to C, Java, and Perl, and the main goal of the language is to allow web developers to write dynamically generated webpages quickly.[19]

The following list describes some of the major features of PHP:

- Open-Source (PHP License)
- HTTP authentication
- Cookies
- Handling file uploads
- Using remote files
- Connection handling
- Persistent database connections
- Safe mode
- Command line interface

I am using HTML and PHP to create the WWW Management Interface. HTML was an obvious choice for creating web content of any sort. I used PHP for the dynamic content because the extensive collection of built-in libraries provide easy-to-use APIs for much of the functionality that I needed. (ex. graphics libraries, socket libraries, DB connections).

11.3 Build and Distribution System

11.3.1 GNU Autotools

GNU AUTOCONF, AUTOMAKE, AND LIBTOOL

The GNU Autotools: Autoconf, Automake, and Libtool are tools for simplifying and automating the compilation and distribution process of software. The Autotools assist with the task of creating portable software – they provide a mechanism that can automatically detect hardware/software on various systems, so that the software can adapt it's configuration and functionality accordingly (usually using config.h files and #DEFINES).[21]

The three parts of the GNU Autotools suite are:[21]

- **Autoconf** - Performs tests to discover system characteristics before the package is compiled. The source code can then adapt to these differences.

- **Automake** – Generates 'Makefiles' that automatically conform to a number of "best practice" standards. The organization of a given package is the tool input. Automake also performs dependency tracking between source code files.

- **Libtool** - A command line interface to the compiler and linker. Libtool is used mostly to generate static and shared libraries that are platform independent.

I use the GNU Autotools to handle the compilation, linking, and distribution-tarball packaging of my project's source code.

Part V – System Design

12. Overview

The Meta-Alert Correlation Engine is a collection of several independently operating modules, connected via TCP communications channels. Figure 7 gives a general architectural overview of the MACE system.

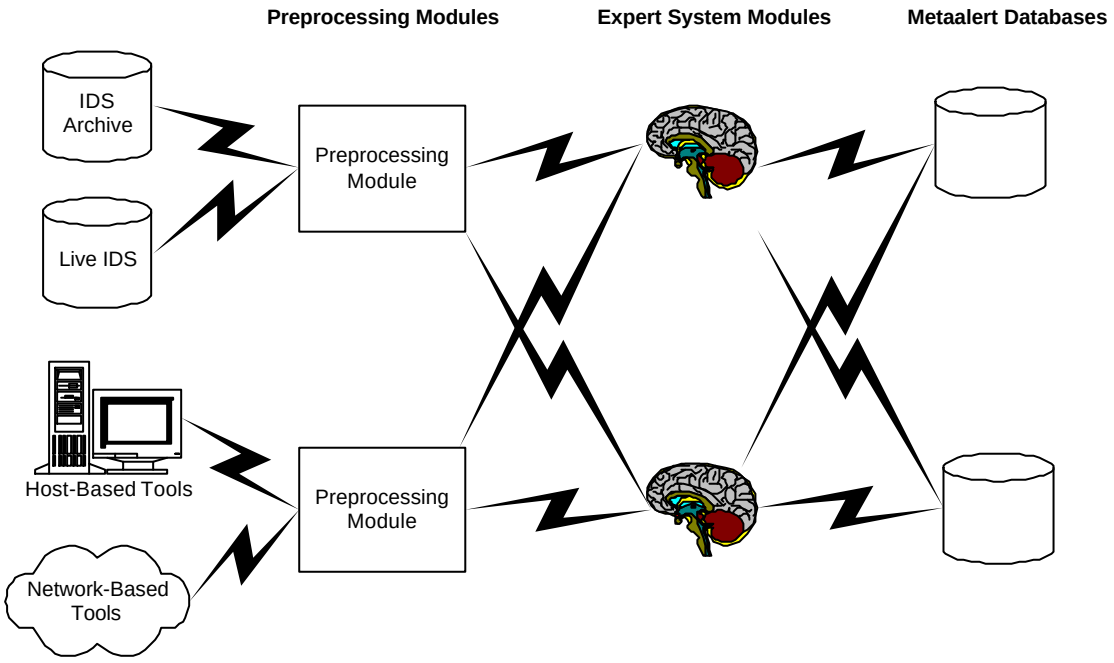


Figure 7 – MACE System Overview

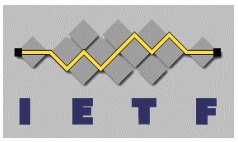
The rest of the chapters in this section describe each of these modules (and the underlying communications mechanism) in detail.

13. Communications Mechanism

The underlying communications between all the internal and the remote modules comprising the Meta-Alert Correlation Engine make use of the Intrusion Detection Message Exchange Format.

13.1 Intrusion Detection Message Exchange Format

13.1.1 The Intrusion Detection Working Group



The Internet Engineering Task Force (IETF) is an umbrella organization that supervises the architectural oversight and continued development of the Internet. The IETF is a widely successful organization, as they have created and maintained most of the protocols in the TCP/IP suite. The IETF is divided into a number of subject areas, and further into "working groups", that tackle specific problems related to the Internet.[25]

The Intrusion Detection Working Group (IDWG) is an IETF working group that exists with the purpose of defining data formats and exchange procedures to facilitate intrusion detection information management, correlation, and response. Led by Mike Erlinger and Stuart Saniford-Chen, the IDWG has produced a number of documents that are intended to become industry (and possibly Internet) standards.[7] These documents include:

- **Requirements document** - describes the high-level functional requirements for communications between intrusion detection systems and management systems. This document also provides a number of examples.[7]

- **Common Language Specification** – Describes the proposed "standard" data format. This document specifies the Intrusion Detection Message Exchange Format (IDMEF) DTD.[7]

- **Framework Document** – Specifies protocols that are best used for transporting communications using the data format. This document highlights the use of the Intrusion Detection Exchange Protocol (IDXP), a protocol that uses a "TUNNEL" profile to exchange IDMEF information between multiple Block Exchange Extensible Protocol (BEEP) peers.[7]

13.1.2 Rationale for using IDMEF

I have chosen to use the Intrusion Detection Message Exchange Format to represent the alert- and meta-alert data within the Meta-Alert Correlation System. This decision was based upon my belief that interoperability is the first step towards solving the Intrusion Detection correlation problem. There are currently many varied sources of intrusion detection data (NIDS, HIDS, anomaly detection, etc..) that are all equally valid and useful for establishing the presence of intrusive activity. However, because they all use various proprietary formats for representing their data, it is difficult to bring this data together for the purposes of management and correlation. By using IDMEF, I believe that it will be easier to extend MACE in new directions in the future, to include new sources of

intrusion detection data, and to easily manage and re-analyze the meta-alerts produced by the MACE system itself.

On a more philosophical note, I believe that IDMEF will find widespread acceptance within the next 5 years. The IDMEF is not yet accepted as an RFC, but the first few reference implementations are now appearing – in C and Perl, and the first few Intrusion Detection Systems are starting to experiment with using IDMEF to represent their output (ex. Snort, Prelude). Only by using a standard will a data format actually become a standard in the "real world". I wish to support this standardization process by including IDMEF within the Meta-Alert Correlation Engine.

13.2 IDMEF++

13.2.1 libidmef

Libidmef is the first reference implementation of the Intrusion Detection Message Exchange Format (IDMEF), created by Joey McAlerney from Silicon Defense and Adam Migus from NAI Labs.[26] It builds upon libxml, the XML C library developed in conjunction with the Gnome project. While this library is still in an early stage of existence at the time of this writing (version 0.7.2), libidmef has thoughtfully implemented the basic data structures and parser functionality, based upon the continuously-evolving IDMEF common language documentation.

13.2.2 libidmef++

Libidmef, while a huge step in the right direction, was not precisely what I needed to interface with the C++ Meta-Alert Correlation Engine. To fill this need, I developed my own C++ wrapper around libidmef, called libidmef++. Libidmef++ provides an object-oriented interface to libidmef, offering a more intuitive interface for C++ programmers that need to automatically generate IDMEF alerts. Libidmef++ also handles the connectivity problems that surface when connecting C++ code with an existing C library.

All of the Meta-Alert Correlation Engine modules use libidmef++ to represent (meta-) alert data that is transmitted through TCP communication sockets.

14. Preprocessing Module

14.1 Introduction

The Meta-Alert Correlation Engine aims to reduce the number of false positives generated by intrusion detection systems. The presence of false positives is one of the largest problems for Security Operations Center (SOC) personnel, as a large percentage of data that causes IDS alerts is simply random junk that happened to trigger an IDS signature, or large volumes of worm attacks that might not even threaten the computers

that are being monitored. A human security expert usually has little trouble separating the "junk" from the meaningful alerts. However, this still presents a huge workload for a single operator to deal with.

The MACE Preprocessing Module attempts to automatically deal with this problem. The preprocessing module is meant to filter out the obvious false positives, that a human operator would otherwise have to spend time looking at.

14.2 Architectural Design

The MACE Preprocessing Module consists of a series of dynamically loadable plugins that can filter out specific alerts based upon specific alert characteristics (ex. signature, IP address, ports, packet content, etc...) Dynamically loadable plugins were chosen as the means to represent this simple filtering information, so that end-users can easily modify or add new plugins to their MACE system, in accordance with their needs. Plugins could be periodically downloaded from a central repository, much in the same way that other security tools (like Snort or Nessus) use for keeping up-to-date with the latest security issues.

The Preprocessing Module receives IDMEF encoded alerts via TCP communications channels, originating from intrusion detection systems (ex. Snort) or remote modules (ex. ArpMonitor, Bandwidth Monitor). The module then activates the appropriate plugin to process that alert, and the system than determines whether the alert should be filtered out or not. If the alert is not filtered out, it is sent on via TCP to the next step, the Expert System Core (see chapter 14.)

Since all of the incoming alerts are represented by the IDMEF, new plugins could be created for the Preprocessing Module, to provide preprocessing support for other IDS systems that are not yet supported.

14.3 Example Preprocessing Plugin

14.3.1 Simple Filtering Example

Snort rule #885 looks for the presence of the string "/bash" in packet content. This rule may alert the SOC about a hacker that is trying to get a bash shell on a target machine. The actual rule is shown below:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-CGI bash access";flow:to_server,established; uricontent:"/bash"; nocase; reference:cve,CAN-1999-0509; reference:url,www.cert.org/advisories/CA-1996-11.html; classtype:web-application-activity; sid:885; rev:6;)
```

However, sometimes other things activate this rule by accident. I've seen alerts with a packet content of:

```
GET /nos/nieuws/images/buitenland/165/bashir_bakar_abu.jpg HTTP/1.1
Accept: */*  Referer:
http://www.omroep.nl/nos/nieuws/hoofdpunten/hoofdpunten.html  Accept-
Language: nl  Accept-Encoding: gzip, deflate  User-Agent: Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.0)  Host: www.omroep.nl
Connection: Keep-Alive
```

or

```
GET /syndicaat/pics/BasHoekstra.jpg HTTP/1.1  Accept: */*  Referer:
http://spike.oli.tudelft.nl/syndicaat/index.cfm?ID=2  Accept-Language:
nl  Accept-Encoding: gzip, deflate  If-Modified-Since: Sun, 19 May 2002
15:40:25 GMT  If-None-Match: "5a14c87e4bffc11:395d"  User-Agent:
Mozilla/4.0 (compatible; MSIE 5.5; Windows 98; Wanadoo cable)  Host:
spike.oli.tudelft.nl  Connection: Keep-Alive
```

These are both examples of packets that match an attack signature, but that are clearly recognized as benign immediately upon inspection.

Therefore, we may want to use the preprocessing module to filter out alerts, triggered by snort alert #885, that have a character with the value [a-z,A-Z] directly after the string "/bash", thus rendering the "bash call" ineffective.

14.3.2 Sample Plugin Code

Here is sample plugin code to solve the simple filtering example described above:

```
/*
 * plugin_SID_885.cxx
 *
 * Plugin to preprocess Snort alert SID #885
 *
 */

#include <iostream.h>
#include <stdio.h>

#include "libidmefpp.h"


/* Don't use C++ name mangling for exported symbols */
extern "C" {

/* These are the symbols to be exported */
#define plugin_init     libpluginSID885_LTX_plugin_init
#define plugin_run      libpluginSID885_LTX_plugin_run


/* Information about our plugin */
#define NAME "Plugin SID #885"
```

```
#define DESCRIPTION "This plugin performs the preprocessing on alert
885"
#define AUTHOR "Melanie Rieback"


/* Plugin initialization function */
int plugin_init() {
  cout << "Initializing plugin SID #885\n";

  return 0;
}

/* Plugin initialization function */

int plugin_run(idmef_object *Idmef_Object) {
  cout << "Running plugin SID #885\n";

  // Declare a few variables
  string::size_type pos = 0;
  char next_letter;
  int ret_val;

  // We need to create a copy of the alert with a new pointer to
  // avoid problems with C++ compiler name-mangling
  idmef_object *Idmef_Object2 = new idmef_object();

  // Generate the new Idmef_Object by reconstructing from the
  // xml string
  string Temp = string(return_xml_string(Idmef_Object));
  parse_xml_string((char *)Temp.c_str(),Idmef_Object2);

  // Get the packet data from the alert
  string Temp = string(Idmef_Object2->get_idmef_message()->
  get_idmef_alert(0)->get_idmef_additionaldata(0)->get_data();

  // Search for the location of the string "/bash" in the data
  pos = Temp.find ("/bash",0);

  // Store the value of the next letter after the string "/bash"
  next_letter = Temp[pos+5];

  // Check if the value of next_letter is [a-z,A-Z]
  if (isalpha((int)next_letter)) {

    /*
     * next_letter IS [a-z,A-Z].  Therefore, we can filter this
     * alert out
     */
    ret_val = 2;
  }
  else {

    /*
     * next_letter is NOT [a-z,A-Z].  Therefore, this might be a
     * real attack
     */
    ret_val = 0;
```

```
  }


  /* Possible return values:
   *
   * 0 - Alert is okay.  Don't delete.
   * 1 - Error during plugin execution.  Don't delete.
   * 2 - Delete this alert
   */

  // Free up some memory
  Temp.~string();
  delete Idmef_Object2;

  // Return our result
  return ret_val;
}

}
```

15. Expert System Core

15.1 Introduction

15.1.1 Introduction to Expert Systems

Expert systems are programs that try to emulate human expertise and problem solving abilities through use of a technique called "rule-based" programming. Rule-based programming makes use of heuristics, or "rules of thumb", to specify actions to perform when specific patterns of data are encountered.[9] An example rule is shown below:

```
(defrule match-dunet-ip-address
  (system-info (my-ip-address ?ip_addr))
  (idmef_address (mid ?mid_value) (index ?address_index) (address
    ?ip_addr))
  (idmef_node (mid ?mid_value) (index ?node_index) (Address
    ?address_index))
  (idmef_target (mid ?mid_value) (Node ?node_index))
=>
  (assert (assert-ids-specific-fact ?mid_value)))
```

Rules are composed of an *if* portion and a *then* portion. The *if* portion of a rule is a series of patterns which specify the data that causes the rule to "activate". The *then* portion is a set of actions to be executed upon this activation.[9]
The above rule, when provided with a fact describing the type of intrusion detection system, and a fact describing an attack itself, will *assert* or add a new fact to the system, containing information about the intrusion detection system, the type of attack, alert CID, and destination IP address. The *if* portion of the rule is specified by what appears before the arrow (=>), and the *then* portion is what appears afterwards.

One of the primary requirements of the Meta-Alert Correlation Engine is to reduce the number of false positives that human IDS operators have to weed through. Expert systems provide an appropriate means of simulating this human ingenuity and experience, utilizing comprehensive knowledge bases, and constant yet adaptable sets of rules to sort and correlate the Intrusion Detection alert data as it appears.

15.1.2 Introduction to CLIPS

CLIPS is an expert system tool which provides a complete environment for the construction of rule and/or object based expert systems. Created in 1985, by NASA's Johnson Space Center, CLIPS is now widely used throughout the government, industry, and academia. Some of its main features are:[9]

- **Knowledge Representation:** CLIPS handles a wide variety of knowledge with support for three different programming paradigms: rule-based, object-oriented and procedural. CLIPS allows complex systems to be modeled as modular components (which can be easily reused to model other systems or to create new components). CLIPS also provides capabilities similar to those found in languages such as C, Java, Ada, and LISP. [9]

- **Portability:** CLIPS was written in C for portability and speed and runs on a whole number of operating systems. Some of these include: Windows 95/98/NT, MacOS X, and *nix. Additionally, CLIPS comes with a complete set of source code, which can be tailored to meet a user's specific needs.[9]

- **Integration/Extensibility:** CLIPS can be embedded within external source code, called as a subroutine, and integrated with languages such as C, Java, FORTRAN and ADA.[9]

- **Verification/Validation:** CLIPS contains features that support the verification and validation of expert systems that are created using the system. This support includes support for modular design and partitioning of the knowledge base, static and dynamic constraint checking, and analysis of rule pattern semantics to locate inconsistencies that could prevent a rule from firing.[9]

- **Fully Documented:** CLIPS comes with extensive documentation including reference manuals, beginning and advanced user guides, and detailed architecture guides.[9]

- **Open Source:** CLIPS is maintained as open source software, and is freely available under the GNU General Protection License.[9]

15.2 Architectural Design

15.2.1 Overview

The CLIPS Expert System core is the part of the Meta-Alert Correlation Engine that is primarily responsible for the correlation and prioritization of incoming Intrusion Detection alerts. The data flow of the system is illustrated in Figure 8:

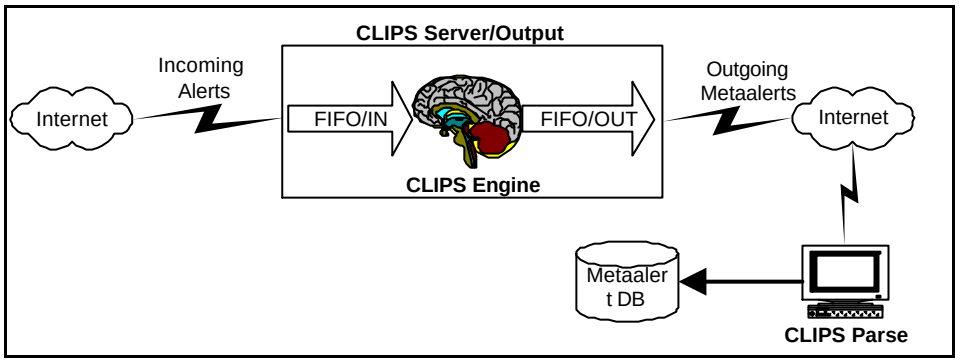


Figure 8 – Architectural Overview of CLIPS Expert System Core

The CLIPS Expert System Core consists of a number of separate components, connected via TCP communications channels that are located on either the same computer, or on another computer elsewhere on the network or Internet. First, the alerts arrive via TCP from primary sources (IDS's) or secondary sources (preprocessing modules). Then, the CLIPS Server reads and parses the incoming alerts, and sends them to the CLIPS Engine via a named pipe (FIFO) on the system. The CLIPS Engine uses intelligent algorithms and correlation techniques to produce a number of Meta-alerts. (The specific algorithms will be discussed in detail in Chapter VI). Output of the CLIPS Engine is then sent by the CLIPS Output module via another named pipe to a socket, and the CLIPS Parse Module reads this output, extracting the meta-alerts, and sending them to the Meta-alert Database.

The upcoming sections in this chapter will describe each of the aforementioned components of the CLIPS expert system core in detail.

15.2.2 CLIPS Engine

The CLIPS Engine is simply an unmodified version of CLIPS Version 6.20, with standard input/output routed to the two FIFOs.

The CLIPS engine is fed with a variety of commands, mostly originating from the WWW management interface (commands and rule definitions), or from the primary or secondary alert sources (fact definitions).

Figure 9 shows a typical screenshot of the CLIPS commandline interface.

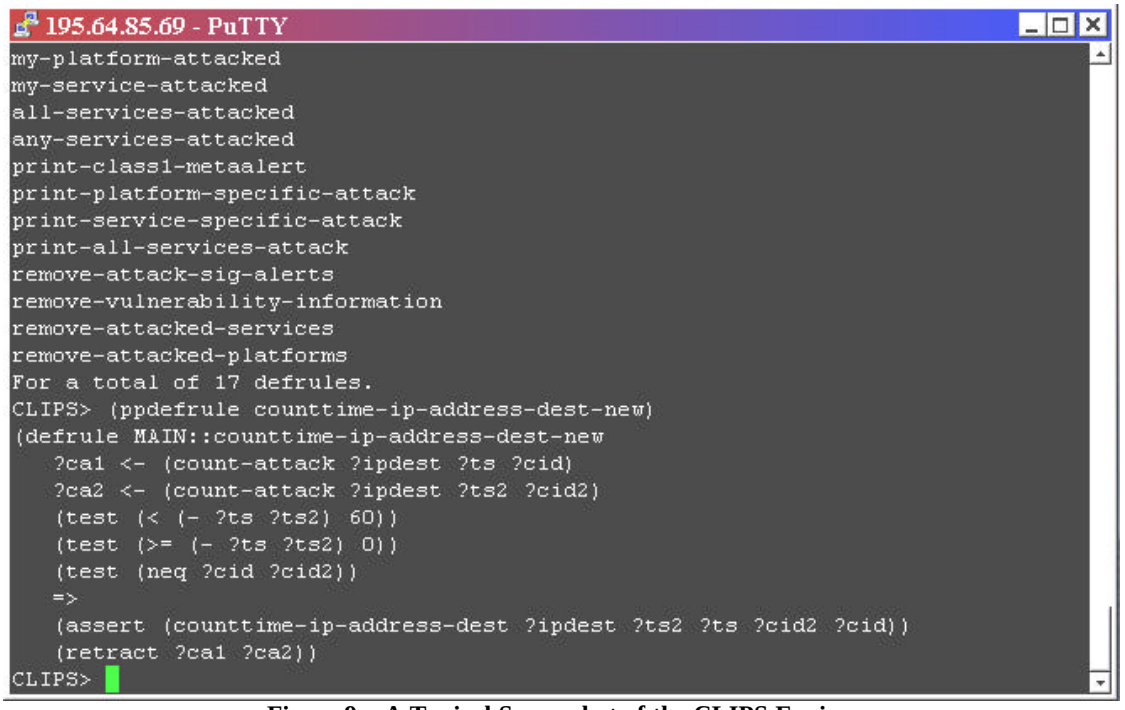


Figure 9 – A Typical Screenshot of the CLIPS Engine

The specific rules and facts (intelligence) utilized by the CLIPS engine will be discussed in great detail in Section VI of this report.

15.2.3 CLIPS Server/Output

The CLIPS Engine is serviced by independent client/server processes that handle the socket communications, and pass alerts (and other information) across the filesystem via named pipes on the operating system. This functionality is provided by two modules: CLIPS Server and CLIPS Output. The CLIPS Server module accepts incoming alerts and commands via a TCP connection on Port 49000 (configurable). The CLIPS Server then passes this input, via a FIFO, to the CLIPS Engine. The CLIPS Output module receives the CLIPS Engine output via a FIFO, and it attaches this output to TCP Port 49001 (configurable), so that other modules can simultaneously connect and read this information. Keep in mind that things like port numbers are easily reconfigurable.

An overview of the CLIPS Server/Output module functionality is shown in Figure 10.

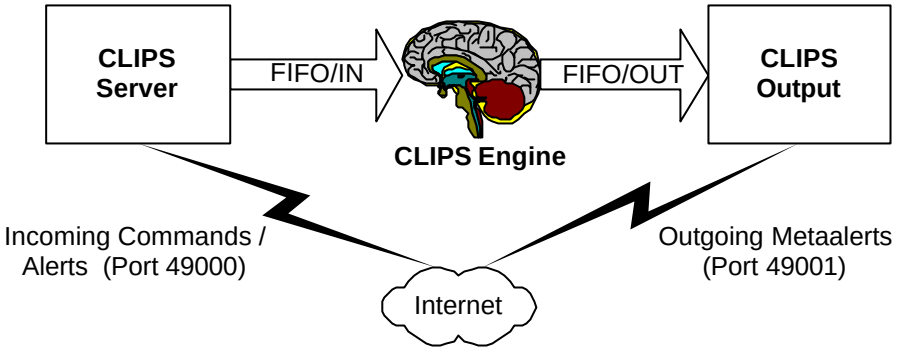


Figure 10 – CLIPS Server/Output Module Functionality

15.2.4 CLIPS Parser

The CLIPS Parser Module reads the CLIPS Engine output, by connecting to the CLIPS output module. This module parses out the meta-alerts as they are generated, and it sends them to the appropriate database(s). The CLIPS Engine produces output that is already in IDMEF format. Here is an example Metaalert, as generated by CLIPS:

```
CLIPS> <Fact-44>
CLIPS> Metaalert:<?xml version="1.0"?><!DOCTYPE IDMEF-Message PUBLIC "-
//IETF//DTD RFC XXXX IDMEF v1.0//EN" ""><IDMEF-Message
version="1.0"><Alert ident="136092"><Analyzer analyzerid="2"
class="snort"><Node><name>unknown:bge0</name></Node></Analyzer><CreateT
ime ntpstamp="0xc2a14226.0x0">2003-06-
23T09:08:54Z</CreateTime><DetectTime ntpstamp="0xc1a4b37c.0x0">2002-12-
13T19:29:00Z</DetectTime><Source
interface="bge0"><Node><Address><address>130.161.180.56</address></Addr
ess></Node><Service><port>3729</port><protocol>tcp</protocol></Service>
</Source><Target
interface="bge0"><Node><Address><address>130.161.180.55</address></Addr
ess></Node><Service><port>139</port><protocol>tcp</protocol></Service><
/Target><Classification origin="vendor-specific"><name>NETBIOS NT NULL
session</name><url>530</url></Classification><AdditionalData
type="string" meaning="Packet
Payload">000000B6FF534D42730000000018038000002AABC6B4F918DA6300000000FE
CA000000000D75008400041132000000000000000100000000000000D40000004700000
0000000570069006E0064006F007700730020004E00540020003100330038003100000
0000570069006E0064006F007700730020004E00540020003400 2E0030000000000004F
F000000000001002700005C005C004F00530043004100520030004E00540031005C0049
00500043002400000049504300</AdditionalData></Alert></IDMEF-Message>
```

The CLIPS parser reads this as input, and sends the meta-alert via TCP to the appropriate meta-alert and/or viewer databases.

16. Primary Alert Modules

The Primary Alert Modules are responsible for collecting the information from the primary sources of alerts. These primary sources may include Network Intrusion

Detection Systems (NIDS), Host-Based Intrusion Detection Systems (HIDS), or other tools that produce alerts based upon some kind of suspicious or anomalous activity. These Primary Alert Modules are less generic than the rest of the Meta-Alert Correlation Engine because they must read and interpret alert data that is stored in a potentially proprietary format. The Primary Alert Modules are responsible for reading and converting (live or archived) IDS alert data into IDMEF format. Upon this conversion, the module sends the IDMEF alerts, via TCP, to the Preprocessing Module for initial filtering.

As time goes on, and as the Meta-Alert Correlation Engine matures, extra primary alert modules can be written. This would allow MACE to correlate and manage alerts from a wide number of primary input sources.

16.1 Snort Primary Alert Module

The Meta-Alert Correlation Engine has a Primary Alert Module to read alerts from a Snort database.

This Primary Alert Module keeps a record of the last alert processed from the database, and it automatically processes the alerts incrementally until they have been processed to completion. This module can be activated from a cron script to process newly-generated alerts upon a predefined time interval. A MySQL connection is used (via a generalized MACE API library) to query the appropriate alert from the database. Then, the alert is converted into IDMEF format, using libidmef++. Finally, the alert is sent via TCP to the desired Preprocessing Module. (The IP address/port of the destination is configurable, so the alert could alternately be sent directly to the CLIPS Server.)

17. Metalert Database

I am currently using a MySQL database to hold the meta-alert data (although a new MACE API library could be written in the future to enable use of other databases.) The Metaalert database, as it is currently implemented, is structured as shown in Figures 10-20.

The first ER diagram, Figure 11, shows the relationship between the idmef_message, idmef_alert, and idmef_heartbeat tables.

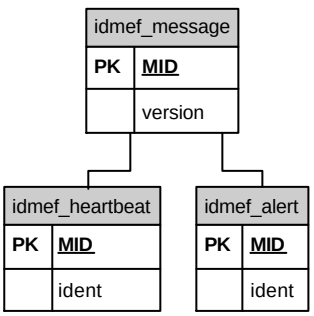


Figure 11– Metaalert Database ER Diagram (Part I)

The second ER diagram, Figure 12, shows the relationship between the following tables: idmef_alert, idmef_analyzer, idmef_source, idmef_target, idmef_time,idmef_assessment, idmef_classification, idmef_additionaldata, idmef_toolalert, idmef_correlationalert, and idmef_overflowalert.

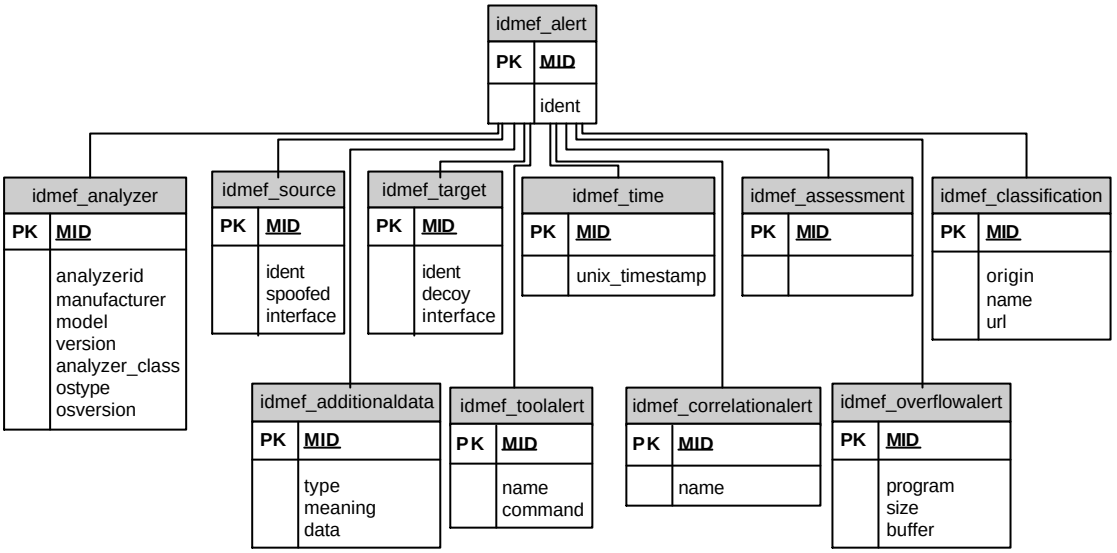


Figure 12– Metaalert Database ER Diagram (Part II)

The third ER diagram, Figure 13, shows the relationship between the following tables: idmef_heartbeat, idmef_analyzer, idmef_time, and idmef_additionaldata.

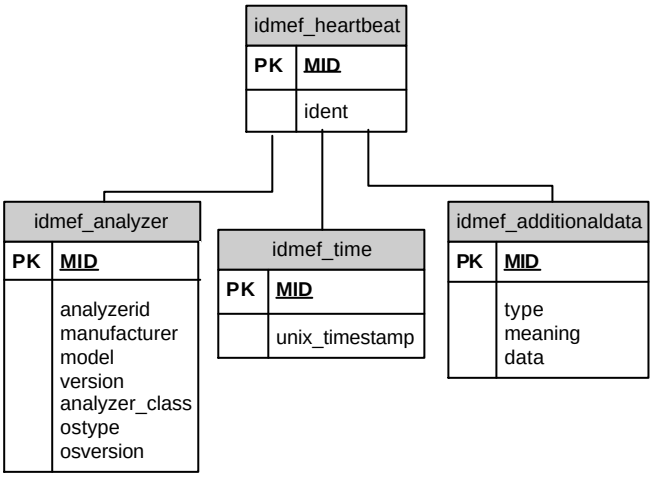


Figure 13– Metaalert Database ER Diagram (Part III)

The fourth ER diagram, Figure 14, shows the relationship between the following tables: idmef_user, idmef_userid

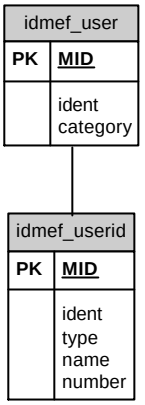


Figure 14– Metaalert Database ER Diagram (Part IV)

The fifth ER diagram, Figure 15, shows the relationship between the following tables: idmef_service, idmef_snmp_service, idmef_webservice

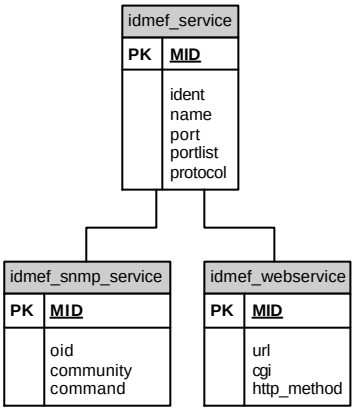


Figure 15– Metaalert Database ER Diagram (Part V)

The sixth ER diagram, Figure 16, shows the relationship between the following tables: idmef_target, idmef_process, idmef_node, idmef_user, idmef_service, idmef_filelist

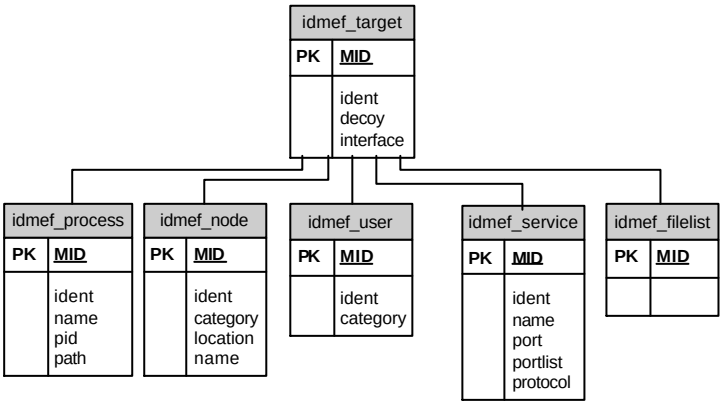


Figure 16– Metaalert Database ER Diagram (Part VI)

The seventh ER diagram, Figure 17, shows the relationship between the following tables: idmef_source, idmef_process, idmef_node, idmef_user, idmef_service

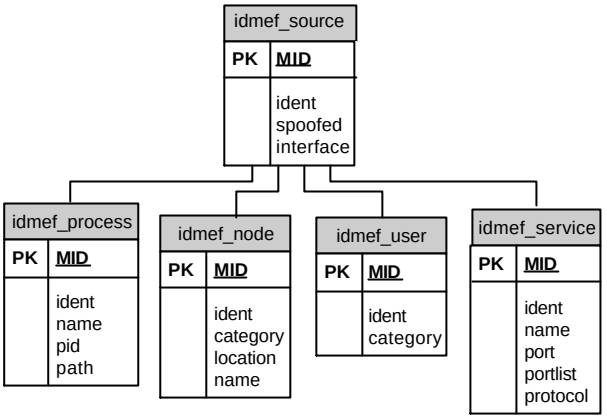


Figure 17– Metaalert Database ER Diagram (Part VII)

The eighth ER diagram, Figure 18, shows the relationship between the following tables: idmef_source, idmef_process, idmef_node, idmef_user, idmef_service

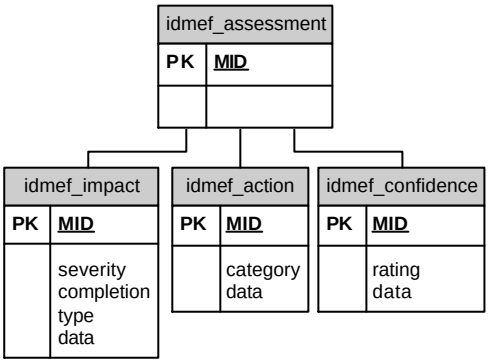


Figure 18– Metaalert Database ER Diagram (Part VIII)

The ninth ER diagram, Figure 19, shows the relationship between the following tables: idmef_analyzer, idmef_node, idmef_process

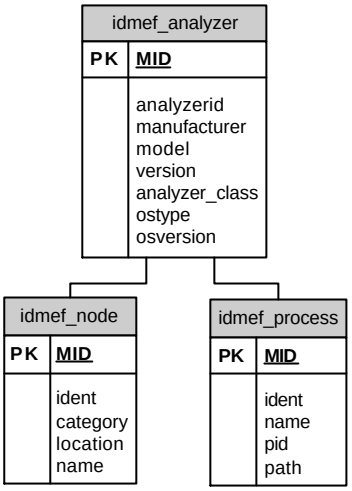


Figure 19– Metaalert Database ER Diagram (Part IX)

The tenth ER diagram, Figure 20, shows the relationship between the following tables: idmef_correlationalert, idmef_toolalert, idmef_alertident

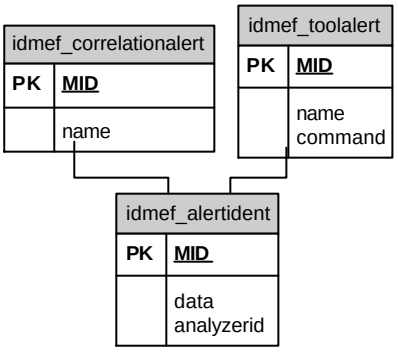


Figure 20– Metaalert Database ER Diagram (Part X)

The eleventh ER diagram, Figure 21, shows the relationship between the following tables: idmef_webservice, idmef_process, and Args

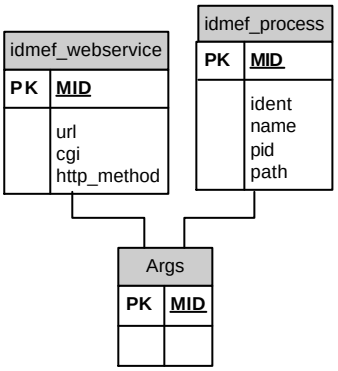


Figure 21– Metaalert Database ER Diagram (Part XI)

18. WWW Management Interface

The Meta-Alert Correlation Engine, while designed as a number of loosely interconnected modules, needs a centralized and intuitive management interface for the users. The average computer user cannot be expected to, and would probably not be interested in, learning the internals of such a system. Therefore, all of the various modules: The Primary Alert Module, Preprocessing Module, CLIPS Engine, and Meta-Alert Database, should all be accessible via a single easy-to-use access point.

For this purpose, I have created a WWW Management Interface. This interface, written with a combination of HTML and PHP, uses TCP to bring the user in contact with each of the separate modules. The WWW Management Interface currently offers the user the ability to enter commands directly into the CLIPS Engine, and to query meta-alerts from the Meta-alert Database.

In the future, I want to add a higher level interface here for each of the components. For the CLIPS Engine, the user should ultimately be able to configure the expert system rules, and specify their monitored machines' platform and service information from this WWW Interface. The Primary Alert Modules should also receive information from the Management Interface controlling their execution. The dynamically loadable plugins of the Preprocessing Module could also ideally be modifiable via this web interface. The Meta-Alert database should also ideally be accessible via an ACID-style interface, that can display (meta-) alerts from an IDMEF Meta-Alert database format, in place of using the current Snort-specific format.

A screenshot from the current WWW Management interface is shown in Figure 22.

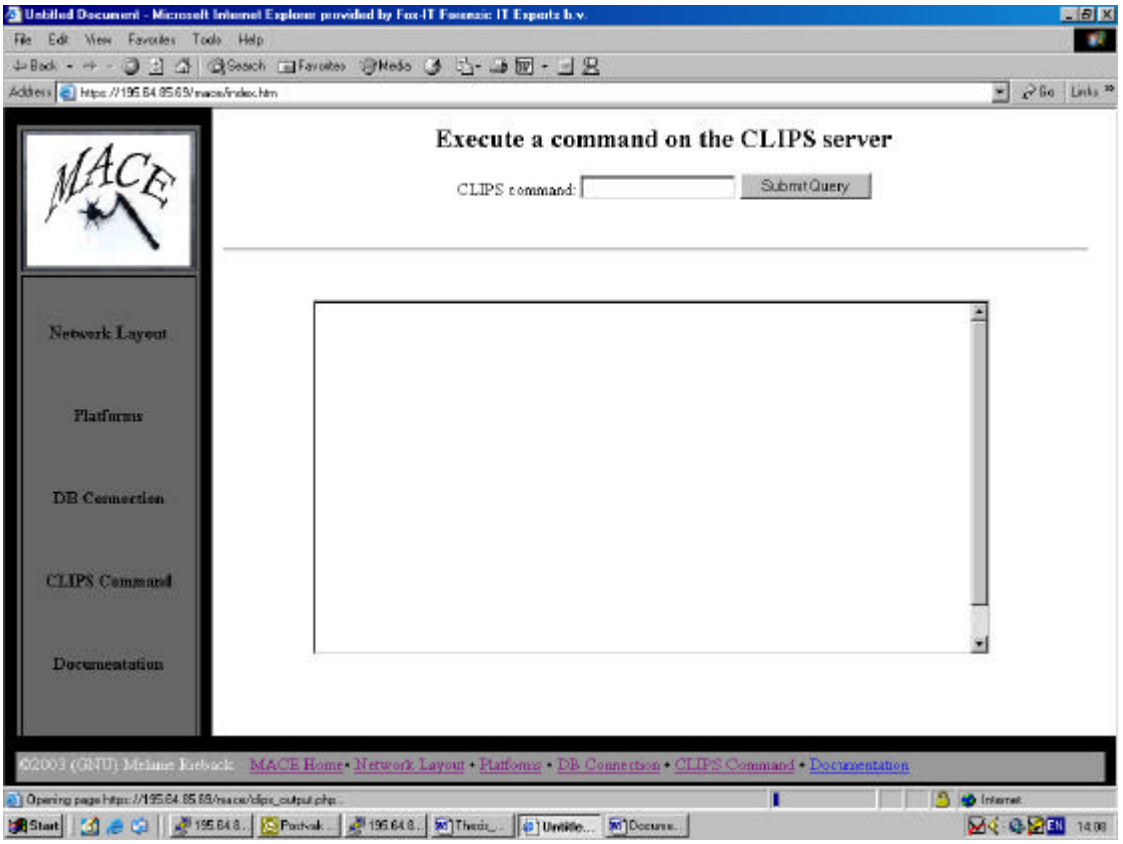


Figure 22 – MACE WWW Management Interface

Part VI – Remote Modules

19. Overview

Just as Primary Alert Modules are available to expand the number of potential MACE input sources, Remote Modules are also available to provide extra customized input to aid meta-alert correlation. Remote Modules are simply independent programs that perform some kind of monitoring function, and generate alerts in an IDMEF format, sending them to a specified IP address/port.

The Meta-Alert Correlation Engine currently has two remote modules available: the ARP Monitor, and the Bandwidth Monitor. These two modules will be discussed in the upcoming two chapters.

20. ARPMonitor

20.1 Address Resolution Protocol

Before packets can be sent between any two machines, each machine must know the data-link address of the "next hop router" along the path to the specified IP address. Address Resolution Protocol is an Internet Protocol, specified by RFC 826, that provides this mapping between 32-bit IP addresses and data link addresses (ex. MAC addresses.) ARP Requests and Replies are sent across the network to update the "ARP caches" of machines on the network, so that they store the most recent IP/data-link address pairs.

Every time a new machine is attached to a network, or a hardware address changes, an ARP Reply will eventually convey the new information to the rest of the computers on the network. This information can be useful to assist with IDS correlation. If a new MAC address appears unexpectedly on an otherwise static network, this may signify the presence of a new and unwanted machine. Similarly, a changed IP/MAC address pair may indicate that an attacker is spoofing one of the IP addresses on the network.

The Lawrence Berkley National Laboratory (LBNL) has created a program called arpwatch, as part of the tcpdump suite of programs, that is capable of monitoring these IP/data-link address pairings.[27] I have created a smaller program, called ARPMonitor, that is directly modeled after arpwatch. ARPMonitor, while sharing much of the same functionality, is much smaller than arpwatch, forgoing the bundled SNMP applications and sending IDMEF-formatted alerts to an IP address/port in place of emailing the alerts, as arpwatch does.

20.2 Tool Design

ARPMonitor also uses libpcap to sniff ARP traffic from the sensed (or specified) network interface. ARP replies are specifically sniffed from the network traffic, and the source/destination IP/data-link address information is parsed from the IP/ARP encapsulated packets. A MySQL database holds the "cache" of ARP/MAC address pairs, updating the information for an IP address every time that a new or changed MAC address appears. Upon new IP address entries, or changed MAC address entries, the ARPMonitor generates and sends an IDMEF-formatted alert to a specified IP address/port for further processing.

20.3 Example ARPMonitor Alert

Here is an example of a "changed MAC address" alert that is generated and sent by the ARPMonitor:

```
<IDMEF-Message version="1.0">
  <Alert ident="alert_id_001">
    <Analyzer analyzerid="analyzer_id_001"/>
    <CreateTime ntpstamp="0xc24132eb.0xdb5caf2d">2003-04-
11T12:26:19Z</CreateTime>
    <Source>
      <Node>
        <Address category="ipv4-addr">
          <address>195.64.85.69</address>
        </Address>
        <Address ident="old mac address" category="mac">
          <address>0:4:76:dd:31:38</address>
        </Address>
        <Address ident="new mac address" category="mac">
          <address>0:4:76:dd:31:38</address>
        </Address>
      </Node>
    </Source>
    <Target>
      <Node>
        <Address category="ipv4-addr">
          <address>195.64.85.69</address>
        </Address>
      </Node>
    </Target>
    <Classification>
      <name>Arpmonitor</name>
      <url> </url>
    </Classification>
    <AdditionalData type="string" meaning="old timestamp">
     "Friday, April 4, 2003 11:33:42 +0200"
    </AdditionalData>
    <AdditionalData type="string" meaning="new timestamp">
```

```
    "Friday, April 4, 2003 11:33:42 +0200"
   </AdditionalData>
  </Alert>
</IDMEF-Message>
```

21. Bandwidth Monitor

21.1 Tool Design

The bandwidth monitor is a tool that measures the bandwidth on a network, split out per source/destination port or IP address. The tool uses the libpcap library to sniff packets from the detected (or specified) network interface, until a SIGALM is activated, after a specified number of seconds. The bandwidth monitor can then be periodically activated by a cron script, allowing the bandwidth monitor to sniff traffic in certain time intervals, (ex. 60 seconds every 10 minutes). This feature is intended to decrease the load demand of the bandwidth monitor while running on a network.

The sniffed packet statistics are then sent to a centralized database, using the MACE MySQL API, where they are statistically analyzed to detect anomalous bandwidth activity (See next section.) Any suspicious bandwidth activity will generate an IDMEF formatted alert, which will be send to MACE for further processing.

21.2 Limit-Based Bandwidth Analysis

I have not implemented this part of the tool yet, but I have collected some requirements about how this part of the tool will work. Alerts will be generated in the following situations:

- unknown IP addresses generate load to/from a server, in excess of a maximum limit (limit configurable)
- known IP addresses generate too much load on a server, in excess of a maximum limit (limit configurable)
- known IP addresses generate load at certain time intervals, where so much activity is unexpected (time interval / bandwidth limits configurable).

The tool should read destination and source ip addresses out of a configuration file, together with values that specify how much data per minute is acceptable over a specified time interval. The tool should constantly run on the network, so as not to give false negatives.

This “limit-based” bandwidth analysis would be most useful on networks that are time-sensitive (i.e. traffic is much lower outside of office hours.)

[The requirements for this tool were gathered by speaking with Erwin Fok, SOC Operator with Fox-IT.]

21.3 Statistical Bandwidth Analysis

This part of the tool has also not been implemented yet. However, the following chapter describes some of the statistical techniques that I intend to use when I do implement it.

The first step in statistically analyzing data is determining a baseline of "normal" activity, that new network activity can be compared to. In order to establish this baseline, we must be confident that the sample size is large enough to reflect a desired level of accuracy, and that our baseline pattern is distinct enough to serve as a tool for comparison. We can determine these that these conditions are true by calculating "confidence intervals".[23]

Confidence intervals are calculated in the following manner:

- Calculate the standard deviation of the data. This can be calculated using the following formula:[24]

$$\sqrt{\frac{r(1-r)}{N}}$$

- The general form of a confidence interval, also called a Z-interval (for given confidence level C) is represented by the formula:[24]

$$\overline{X} \pm ZValue\frac{s}{\sqrt{N}}$$

- The Zvalue corresponds to a desired confidence percentage, that can be found in a Z-Table, such as the following:[24]

Confidence	C	Z Value
90%	.9	1.645
95%	.95	1.96
99%	.99	2.575

In this manner, we can calculate whether our baseline is reliable within statistically "confident" limits.

Upon establishing a statistical baseline, we can then generate alerts based upon deviations from this baseline, as determined by user specific bandwidth deviation limits (also to be stored in a file).

Part VII – Metaalert Algorithms

22. Overview

The following sections in this chapter discuss some of the meta-alert correlation algorithms that are used within the CLIPS Engine of the Meta-Alert Correlation Engine.

23. Data Structures

23.1 Introduction to Deftemplates

The CLIPS Expert System offers a construction called a **define template**, or deftemplate for short, that aids in writing rules for facts that have pre-defined structures. Deftemplates are similar to *structs* available in several high-level programming languages. Deftemplates contain lists of known fields, also known as "slots", that have a preassigned field name, a data type, and a place to hold single or multiple data values (in "multislots").[28]

The CLIPS Engine of the Meta-Alert Correlation Engine heavily uses deftemplates to represent the major data structures that are used to hold the Intrusion Detection and and monitored systems' information.

23.2 IDMEF Alert Templates

Intrusion Detection alerts are logically represented within CLIPS by a series of deftemplates, that provide a template to represent the various hierarchies of IDMEF information. Here is an example CLIPS deftemplate for the top-level IDMEF message:

```
(deftemplate idmef_message
"Define a template for holding IDMEF message information"
  (slot mid             ; The metaalert ID
  (type STRING)
  (default "UNKNOWN"))  ; Sets the value to "UNKNOWN", if none is
                        ; provided
  (slot index           ; The metaalert index #
  (type STRING)
  (default "UNKNOWN"))  ; Sets the value to "UNKNOWN", if none is
                        ; provided
  (slot version         ; IDMEF message version number
  (type STRING)
  (default "UNKNOWN"))  ; Sets the value to "UNKNOWN", if none is
                        ; provided
  (multislot Alert      ; Indices of Alerts contained within this IDMEF
                        ; message
  (type STRING)
  (default "UNKNOWN"))  ; Sets the value to "UNKNOWN", if none is
                        ; provided
```

```
  (multislot Heartbeat  ; Indices of Heartbeats contained within this
                        ; IDMEF message
  (type STRING)
  (default "UNKNOWN"))) ; Sets the value to "UNKNOWN", if none is
                        ; provided
```

The idmef_message deftemplate contains a number of slots: mid, index, version, Alert, and Heartbeat. These slots are responsible for holding the various pieces of information from the IDS alerts, after they are parsed out of the IDMEF-format alerts.

There are also deftemplates that represent the following IDMEF entities in CLIPS:

idmef_address, idmef_time, idmef_classification, idmef_userid, idmef_user, idmef_snmp_service, idmef_webservice, idmef_service, idmef_process, idmef_node, idmef_fileaccess, idmef_linkage, idmef_inode, idmef_file, idmef_filelist, idmef_source, idmef_target, idmef_impact, idmef_action, idmef_confidence, idmef_assessment, idmef_alertident, idmef_additionaldata, idmef_analyzer, idmef_toolalert, idmef_overflowalert, idmef_correlationalert, idmef_alert, and idmef_heartbeat.

An example IDMEF Object could be asserted in the CLIPS as follows:

```
(assert
  (idmef_node (mid "2-129941") (index "1") (ident "UNKNOWN") (category
    "UNKNOWN") (location "UNKNOWN") (name "unknown:bge0") (Address
    "UNKNOWN"))
  (idmef_analyzer (mid "2-129941") (index "2") (analyzerid "2")
    (manufacturer "UNKNOWN") (model "UNKNOWN") (version "UNKNOWN")
    (class "snort") (ostype "UNKNOWN") (osversion "UNKNOWN") (Node "1")
    (Process "UNKNOWN"))
  (idmef_time (mid "2-129941") (index "3") (ntpstamp "0xc29af37f.0x0")
    (datetime "2003-06-18T14:19:43Z") (unix_timestamp 1055945983))
  (idmef_time (mid "2-129941") (index "4") (ntpstamp "0xc1a48805.0x0")
    (datetime "2002-12-13T16:23:33Z") (unix_timestamp 1039796613))
  (idmef_address (mid "2-129941") (index "5") (ident "UNKNOWN")
    (category "UNKNOWN") (vlan_name "UNKNOWN") (vlan_num "UNKNOWN")
    (address "217.83.14.216") (netmask "UNKNOWN"))
  (idmef_node (mid "2-129941") (index "6") (ident "UNKNOWN") (category
    "UNKNOWN") (location "UNKNOWN") (name "UNKNOWN") (Address "5"))
  (idmef_service (mid "2-129941") (index "7") (ident "UNKNOWN") (name
    "UNKNOWN") (port "1027") (portlist "UNKNOWN") (protocol "tcp")
    (Webservice "UNKNOWN") (Snmpservice "UNKNOWN"))
  (idmef_source (mid "2-129941") (index "8") (ident "UNKNOWN") (spoofed
    "UNKNOWN") (interface "bge0") (Node "6") (User "UNKNOWN") (Process
    "UNKNOWN") (Service "7"))
  (idmef_address (mid "2-129941") (index "9") (ident "UNKNOWN")
    (category "UNKNOWN") (vlan_name "UNKNOWN") (vlan_num "UNKNOWN")
    (address "130.161.180.142") (netmask "UNKNOWN"))
  (idmef_node (mid "2-129941") (index "10") (ident " UNKNOWN")
    (category "UNKNOWN") (location "UNKNOWN") (name "UNKNOWN") (Address
    "9"))
  (idmef_service (mid "2-129941") (index "11") (ident "UNKNOWN") (name
```

```
    "UNKNOWN") (port "1080") (portlist "UNKNOWN") (protocol "tcp")
    (Webservice "UNKNOWN") (Snmpservice "UNKNOWN"))
  (idmef_target (mid "2-129941") (index "12") (ident "UNKNOWN") (decoy
    "UNKNOWN") (interface "bge0") (Node "10") (User "UNKNOWN") (Process
    "UNKNOWN") (Service "11") (Filelist "UNKNOWN"))
  (idmef_classification (mid "2-129941") (index "13") (origin "vendor-
    specific") (name "SCAN SOCKS Proxy attempt") (url "615"))
  (idmef_additionaldata (mid "2-129941") (index "14") (type "string")
    (meaning "Packet Payload") (data "NULL"))
  (idmef_alert (mid "2-129941") (index "15") (ident"129941") (Analyzer
    "2") (Createtime "3") (Detecttime "4") (Analyzertime "UNKNOWN")
    (Source "8") (Target "12") (Classification "13") (Assessment
    "UNKNOWN") (Correlationalert "UNKNOWN") (Toolalert "UNKNOWN")
    (Overflowalert "UNKNOWN") (Additionaldata "14"))
  (idmef_message (mid "2-129941") (index "16") (version "1.0") (Alert
    "15") (Heartbeat "UNKNOWN"))
)
```

Please note that it is not required to specify values for all of the slots when asserting a fact that uses a deftemplate. The value "UNKNOWN", in this case, is used as a default when values are not explicitly provided.

23.3 System Info Template

Computer systems to be monitored are logically represented by the following "system-info" deftemplate:

```
(deftemplate system-info
"Define a default template for holding our systems information"
  (slot my-ip-address       ; IP address for this machine
    (type STRING)
    (default "N/A"))        ; Sets the value to "N/A", if none is
                            ; provided
  (multislot my-operating-system  ; Operating system types in use
    (type STRING)
    (default "N/A"))        ; Sets the value to "N/A", if none is
                            ; provided
  (multislot my-services    ; The types of services available
    (type STRING)
    (default "N/A")))       ; Sets the value to "N/A", if none is
                            ; provided
```

The system-info deftemplate contains the following of slots: my-ip-address, my-operating-system, and my-services. Note that my-services is actually a multislot, which means that multiple services can be stored in the template for one computer system.

A "system-info" fact can be asserted in the system as follows:

```
(assert (system-info (my-ip-address "2191646306") (my-operating-system
"Windows 2000")(my-services "Quicktime 5.02" "Powerftp 2.24")))
```

24. Attack / Vulnerability Correlation

24.1 Initial Alert Generation Rules

The Meta-Alert Correlation Engine uses a series of *if-then* expert system rules to transform the "system-info" facts and the incoming "attack" facts, into a meaningful alert that indicated that the targeted system might be vulnerable to the detected attack.

In designing these Expert System rules, I am trying to build something as generic as possible, that can work with several brands and types of IDS alerts. Therefore, I have some alert generation rules that deal with conversions between generic and IDS-specific formats.

First, a rule is needed so that every time a new IDMEF alert appears in the system, an attack-specific correlation fact will be generated. This fact is defined as follows:

```
; Every time that an IDMEF alert appears in the system, assert an
; attack-specific correlation fact for it

(defrule assert-attack-specific-fact
  (assert-attack-specific-fact ?my_mid)
  (idmef_classification (mid ?my_mid) (url ?attacktype))
  (idmef_analyzer (mid ?my_mid) (class ?my_class))
  =>
  (assert (attack-sig ?my_class ?attacktype ?my_mid))
)
```

24.2 Alert Conversion Rules

Once an IDS-specific alert is present in the Expert System, we want to convert it from a proprietary format to a more general one. Intrusion Detection Systems use identifiers to identify the type and class of attacks. There are also a few alert repositories, namely Bugtraq and CVE/CAN, that attempt to standardize attack ids, enabling free and commercial IDS systems to use a common language in describing intrusive events.

The Meta-Alert Correlation System uses a body of alert conversion rules to convert between several proprietary and standardized alert id values. The objective is to ultimately convert every alert id to an IDSS X-Force id, since MACE uses their comprehensive list of attacks/vulnerabilities to perform the final vulnerability correlation. (See section 24.3.)

24.2.1 Bugtraq Conversion Rules

Bugtraq is one of the best known repositories of IDS alert information. Conversion rules are pulled directly from the website, which is shown in Figure 23.

Figure 23 – Bugtraq Database Screenshot

Scripts automatically generate CLIPS rules that can convert to/from Bugtraq ids. A sample rule is shown below:

```
(defrule convert-bugtraq-bid-300
  (attack-sig "bugtraq" "bid300" ?mid)
  =>
  (assert
    (attack-sig "xforce" "xforce2265" ?mid)
  )
)
```

24.2.2 CVE/CAN Conversion Rules

CVE/CAN is another one of the most widely used and respected repositories of IDS alert information. A screenshot of the website is shown in Figure 24.

Scripts automatically generate CLIPS rules that can convert to/from CVE/CAN ids. A sample rule is shown below:

```
(defrule convert-cve-CAN-2001-0466
  (attack-sig "cve" "CAN-2001-0466" ?mid)
  =>
  (assert
    (attack-sig "xforce" "xforce6319" ?mid)
  )
)
```

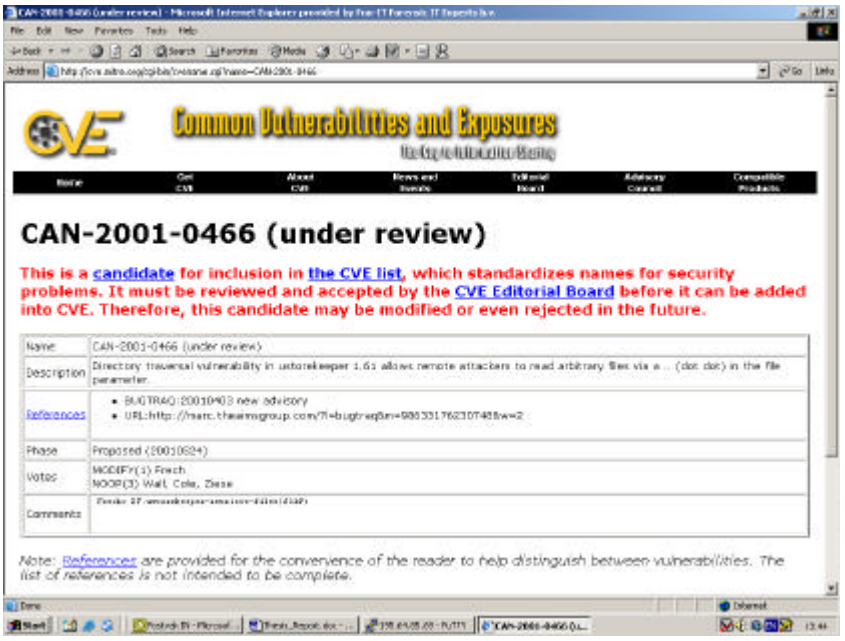


Figure 24 – CVE/CAN Database Screenshot

24.2.3 Snort Conversion Rules

Snort is a very commonly used open-source Intrusion Detection system. It maintains its own repository of rules, as shown in Figure 25.

Scripts automatically generate CLIPS rules that can convert to/from Snort ids. A sample rule is shown below:

```
(defrule convert-snort-sid-1106
  (attack-sig "snort" "sid1106" ?mid)
  =>
  (assert
     (attack-sig "bugtraq" "bid1431" ?mid)
     (attack-sig "cve" "CAN-2000-0590" ?mid)
  )
)
```

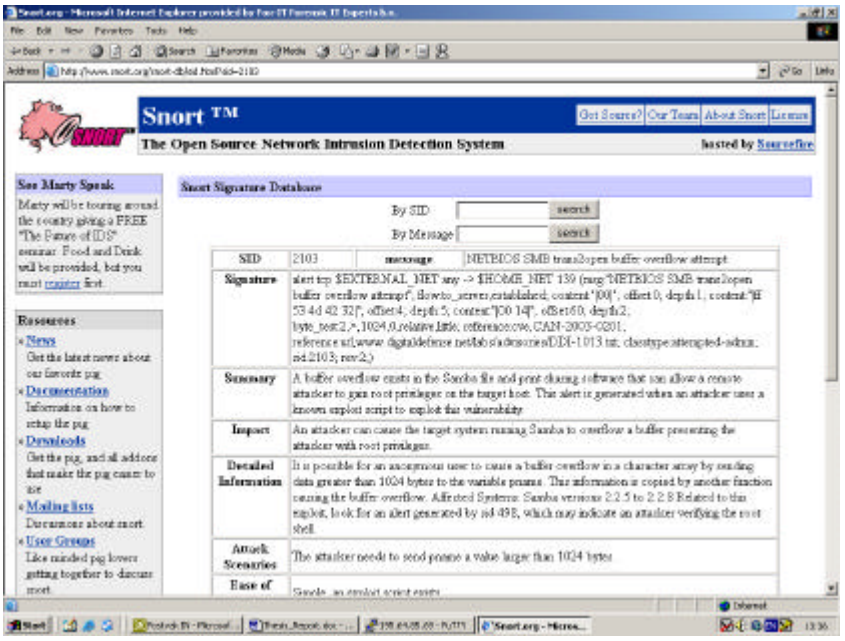


Figure 25 – Snort Database Screenshot

24.2.4 Whitehats Conversion Rules

Whitehats (also called arachNIDS) is another publicly maintained IDS alert repository. A screenshot of the website is shown in Figure 26.

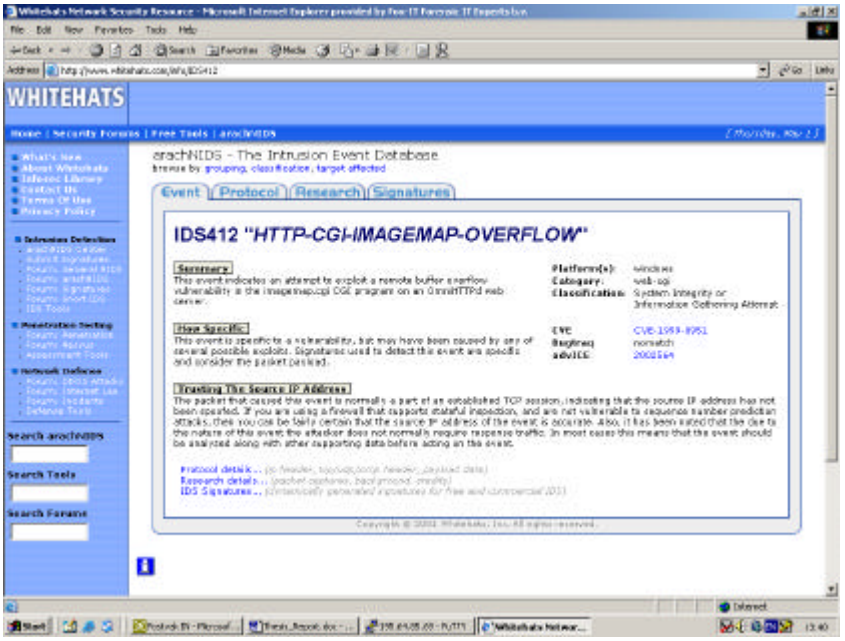


Figure 26 – Whitehats Database Screenshot

Scripts automatically generate CLIPS rules that can convert to/from Whitehats ids. A sample rule is shown below:

```
 (defrule convert-whitehats-IDS-412
  (attack-sig "whitehats" "IDS412" ?mid)
  =>
  (assert
    (attack-sig "cve" "CVE-1999-0951" ?mid)
    (attack-sig "advICE" "advICE2002564" ?mid)
  )
)
```

24.3 Vulnerability Conversion Rules

The X-Force Database is a commercial IDS alert repository, maintained by Internet Security Systems (ISS). A screenshot of the X-Force website is shown in Figure 27.

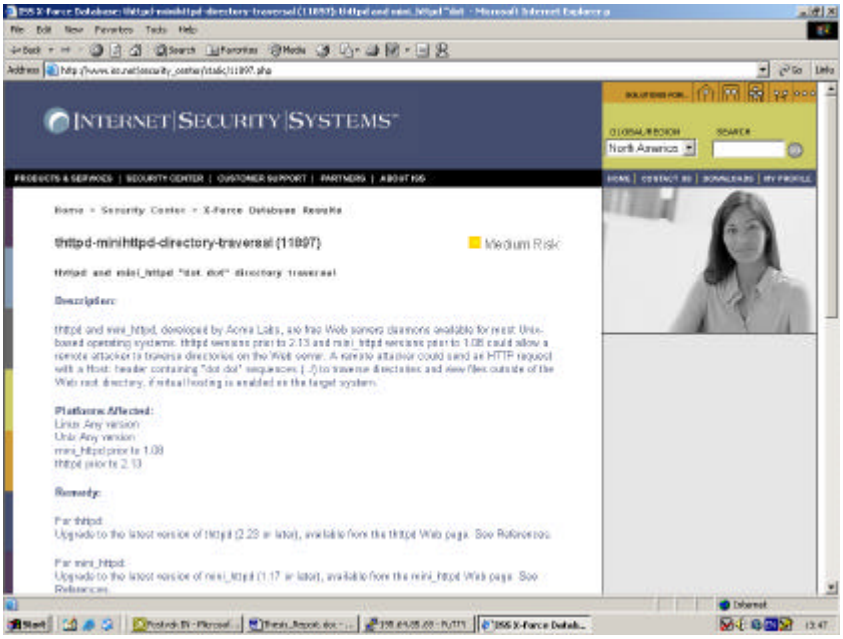


Figure 27 – X-Force Database Screenshot

The X-Force database is the final step of the IDS alert conversion. The Meta-Alert Correlation Engine uses the X-Force alert ids to perform the vulnerability correlation because after a lengthy investigation, the X-Force database has proven to have the most detailed and complete lists of attack types vs. affected platform/services.

Scripts parse data directly taken from the X-Force website, and automatically generate CLIPS rules that can convert between X-Force id's and vulnerable platforms and services. A sample rule is shown below:

```
(defrule define-vulns-xforce-10056
  (attack-sig "xforce" "xforce10056" ?mid)
  =>
  (assert
```

```
    (vulnerable "Amavis Virus Scanner 0.2.1-r2 and earlier"
"xforce10056" ?mid)
    (vulnerable "Linux Any version" "xforce10056" ?mid)
    (vulnerable "Unix Any version" "xforce10056" ?mid)
  )
)
```

24.4 Aggregation Conversion Rules

The Meta-Alert Correlation System also requires a ruleset to convert between "aggregated" platform/service descriptions, and single platform/service descriptions (as would be used in the system-info facts). Two sample rules are shown below:

```
(defrule aggregate-vulnerabilities-Kazaa-Any-Version
  (vulnerable "Kazaa Any version" ?mid)
  =>
  (assert
    (vulnerable "Kazaa 1.3" ?mid)
    (vulnerable "Kazaa 1.7.1" ?mid)
    (vulnerable "Kazaa 2.0.2" ?mid)
  )
)

(defrule aggregate-vulnerabilities-Cvs-1-11-4-and-earlier
  (vulnerable "Cvs 1.11.4 and earlier" ?mid)
  =>
  (assert
    (vulnerable "Cvs 1.10.8" ?mid)
    (vulnerable "Cvs 1.11" ?mid)
    (vulnerable "Cvs prior to 1.10.7-9" ?mid)
  )
)
```

These rules are important, because if the X-Force Database determines that "Kazaa Any version" is vulnerable, the computers running Kazaa versions 1.3, 1.7.1, and 2.0.2 all need to be considered for a potential meta-alert.

These rules (especially aggregations using "and earlier", "and later", etc..) are more difficult to automatically generate with scripts than the other rules because it requires an ever-growing list of which versions of various platforms/services are available, as well as because the naming and capitalization tends to be much less consistent within the various alerts. Improvements in this situation would unfortunately probably require an effort to standardize the terminology and capitalization used by the X-Force website itself.

24.5 Metaalert Generation Rules

24.5.1 Target Comparison Rules

Once a particular attack has been converted to a list of potentially affected platforms/services, the target computer must be checked to see if it is running any of the potentially vulnerable platforms and/or services. The two CLIPS rules below demonstrate how this target platform/service comparison is done:

```
; Platform specific attack is targeting one of our monitored computers
(defrule my-platform-attacked
  (system-info (my-operating-system $?osbefore ?myOS $?osafter) (my-ip-
     address ?ip_addr))
  (vulnerable ?myOS ?mid_value)
  (idmef_address (mid ?mid_value) (index ?address_index) (address
    ?ip_addr))
  (idmef_node (mid ?mid_value) (index ?node_index) (Address
    ?address_index))
  (idmef_target (mid ?mid_value) (Node ?node_index))
  =>
  (assert (generate-metaalert-for-mid ?mid_value))
)

; Service specific attack is targeting one of our monitored computers
(defrule my-service-attacked
  (system-info (my-services $?servbefore ?ServVal $?servafter) (my-ip-
    address ?ip_addr))
  (vulnerable ?myServ ?mid_value)
  (test (neq (str-index (lowcase ?myServ) (lowcase ?ServVal)) FALSE))
  (idmef_address (mid ?mid_value) (index ?address_index) (address
    ?ip_addr))
  (idmef_node (mid ?mid_value) (index ?node_index) (Address
    ?address_index))
  (idmef_target (mid ?mid_value) (Node ?node_index))
  =>
  (assert (generate-metaalert-for-mid ?mid_value))
)
```

Note that the platform rule checks for exact platform type matches (ex. Windows 95) while the service rule checks for the existence of the service as a substring in the attack type (ex. The string “ftp” in “Ws_FTP Server 3.1.1"”). If the potentially vulnerable platform/service is found on the target, than a metaalert will be generated for that particular IDMEF alert.

24.5.2 Metaalert Generation Rules

Metaalerts are actually generated within MACE (i.e. read by the CLIPS Parser, encoded in IDMEF format, and sent to the Metaalert Database), when the line of CLIPS output is preceded by the word “Metaalert”. This means that the final Metaalert generation rules, triggered upon some discovered or newly-generated fact in the system, produce a line of output that will activate the CLIPS Parser. A metaalert generation rule for an IDMEF message is shown below:

```
;; Rule to print out an idmef-message, given the mid
;;---------------------------------------------------------------------

(defrule print-idmef-message-using-mid
  ?control-fact <- (print-message-with-mid ?arg_mid)
  (idmef_message (mid ?arg_mid) (version ?arg_version) (Alert $?alerts)
    (Heartbeat $?heartbeats))
=>
  ; Retract the control fact that activated this rule
  (retract ?control-fact)

  ; Print out the XML Header information
  (printout t "<?xml version=\"1.0\"?>")
  (printout t "<!DOCTYPE IDMEF-Message PUBLIC \"-//IETF//DTD RFC XXXX
    IDMEF v1.0//EN\" \"\">")

  ; Print out an inital statement
  (printout t "<IDMEF-Message")

  ; Print out the IDMEF Message Version number
  (if (neq ?arg_version "UNKNOWN")
    then
    (printout t " version=\"" ?arg_version "\""))

  ; Print out a closing tag
  (printout t ">")

  ; Assert a fact that will cause a final idmef-message tag to be
  ; generated
  (assert (generate-idmef-message-end-tag))

  ; Print out each of the heartbeats
  (loop-for-count (?cnt 1 (length$ ?heartbeats)) do
    (if (neq (nth$ ?cnt ?heartbeats) "UNKNOWN")
      then
      (assert (print-heartbeat-with-mid ?arg_mid (nth$ ?cnt
        ?heartbeats)))))

  ; Print out each of the alerts
  (loop-for-count (?cnt 1 (length$ ?alerts)) do
    (if (neq (nth$ ?cnt ?alerts) "UNKNOWN")
      then
      (assert (print-alert-with-mid ?arg_mid (nth$ ?cnt ?alerts))))))
```

The meta-alert is generated when a fact like this is asserted into the system:

```
(print-message-with-mid "2-129941")
```

24.6 Example

Let's assume that we have 20 systems that we are monitoring and several alerts flying by per second. It's tough for the administrator to keep tabs on everything, so he uses his knowledge of the network (and perhaps also nessus/nmap) to create a list of operating

systems and services running per computer. Among the other computers, one of them is a Windows 2000 machine, running Ws FTP, Zone Alarm 3.0, and a Socks Proxy Server. This information is represented in CLIPS as following:

```
CLIPS> (assert (system-info (my-ip-address "130.161.180.142") (my-
operating-system "Windows 2000 Professional") (my-services "Ws_FTP
Server 3.1.1" "Zonealarm Pro 3.0" "Socks 5-v1.0r10")))
```

Alerts are read out of the Snort database that is attached to the network, and the following alert appears:

```
CLIPS> (assert
  (idmef_node (mid "2-129941") (index "1") (ident "UNKNOWN") (category
    "UNKNOWN") (location "UNKNOWN") (name "ep0") (Address
    "UNKNOWN"))
  (idmef_analyzer (mid "2-129941") (index "2") (analyzerid "2")
    (manufacturer "UNKNOWN") (model "UNKNOWN") (version "UNKNOWN")
    (class "snort") (ostype "UNKNOWN") (osversion "UNKNOWN") (Node "1")
    (Process "UNKNOWN"))
  (idmef_time (mid "2-129941") (index "3") (ntpstamp "0xc29af37f.0x0")
    (datetime "2003-06-18T14:19:43Z") (unix_timestamp 1055945983))
  (idmef_time (mid "2-129941") (index "4") (ntpstamp "0xc1a48805.0x0")
    (datetime "2002-12-13T16:23:33Z") (unix_timestamp 1039796613))
  (idmef_address (mid "2-129941") (index "5") (ident "UNKNOWN")
    (category "UNKNOWN") (vlan_name "UNKNOWN") (vlan_num "UNKNOWN")
    (address "217.83.14.216") (netmask "UNKNOWN"))
  (idmef_node (mid "2-129941") (index "6") (ident "UNKNOWN") (category
    "UNKNOWN") (location "UNKNOWN") (name "UNKNOWN") (Address "5"))
  (idmef_service (mid "2-129941") (index "7") (ident "UNKNOWN") (name
    "UNKNOWN") (port "1027") (portlist "UNKNOWN") (protocol "tcp")
    (Webservice "UNKNOWN") (Snmpservice "UNKNOWN"))
  (idmef_source (mid "2-129941") (index "8") (ident "UNKNOWN") (spoofed
    "UNKNOWN") (interface "bge0") (Node "6") (User "UNKNOWN") (Process
    "UNKNOWN") (Service "7"))
  (idmef_address (mid "2-129941") (index "9") (ident "UNKNOWN")
    (category "UNKNOWN") (vlan_name "UNKNOWN") (vlan_num "UNKNOWN")
    (address "130.161.180.142") (netmask "UNKNOWN"))
  (idmef_node (mid "2-129941") (index "10") (ident " UNKNOWN")
    (category "UNKNOWN") (location "UNKNOWN") (name "UNKNOWN") (Address
    "9"))
  (idmef_service (mid "2-129941") (index "11") (ident "UNKNOWN") (name
    "UNKNOWN") (port "1080") (portlist "UNKNOWN") (protocol "tcp")
    (Webservice "UNKNOWN") (Snmpservice "UNKNOWN"))
  (idmef_target (mid "2-129941") (index "12") (ident "UNKNOWN") (decoy
    "UNKNOWN") (interface "bge0") (Node "10") (User "UNKNOWN") (Process
    "UNKNOWN") (Service "11") (Filelist "UNKNOWN"))
  (idmef_classification (mid "2-129941") (index "13") (origin "vendor-
    specific") (name "SCAN SOCKS Proxy attempt") (url "615"))
  (idmef_additionaldata (mid "2-129941") (index "14") (type "string")
    (meaning "Packet Payload") (data "NULL"))
  (idmef_alert (mid "2-129941") (index "15") (ident"129941") (Analyzer
    "2") (Createtime "3") (Detecttime "4") (Analyzertime "UNKNOWN")
    (Source "8") (Target "12") (Classification "13") (Assessment
    "UNKNOWN") (Correlationalert "UNKNOWN") (Toolalert "UNKNOWN")
    (Overflowalert "UNKNOWN") (Additionaldata "14"))
  (idmef_message (mid "2-129941") (index "16") (version "1.0") (Alert
```

```
    "15") (Heartbeat "UNKNOWN"))
)
```

CLIPS produces the output:

```
Metaalert:<?xml version="1.0"?><!DOCTYPE IDMEF-Message PUBLIC "-
//IETF//DTD RFC XXXX IDMEF v1.0//EN" ""><IDMEF-Message
version="1.0"><Alert ident="129941"><Analyzer analyzerid="2"
class="snort"><Node><name>unknown:bge0</name></Node></Analyzer><CreateT
ime ntpstamp="0xc29af37f.0x0">2003-06-
18T14:19:43Z</CreateTime><DetectTime ntpstamp="0xc1a48805.0x0">2002-12-
13T16:23:33Z</DetectTime><Source
interface="bge0"><Node><Address><address>217.83.14.216</address></Addre
ss></Node><Service><port>1027</port><protocol>tcp</protocol></Service><
/Source><Target
interface="bge0"><Node><Address><address>130.161.180.142</address></Add
ress></Node><Service><port>1080</port><protocol>tcp</protocol></Service
></Target><Classification origin="vendor-specific"><name>SCAN SOCKS
Proxy attempt</name><url>615</url></Classification><AdditionalData
type="string" meaning="Packet
Payload">NULL</AdditionalData></Alert></IDMEF-Message>
```

This metaalert is produced because the Socks Proxy service is threatened. The CLIPS Parser reads this output, and adds the metaalert to the meta-alert and/or viewer database.

25. Intermediate Fact Removal

25.1 Analysis of Example

As Meta-Alert Correlation algorithms run, they will produce a tremendous amount of "intermediate" facts, that represent a transient change in state, while performing the necessary conversion steps that lead to a meta-alert. These extra facts must be removed from the system to keep the system performance at an optimal level.

If we look back at the example in section 24.6, the automatic vulnerability correlation produces the following intermediate facts as a new alert appears in the system:

```
f-4     (attack-sig "snort" "sid1888" "123")
f-5     (attack-sig "bugtraq" "bid5427" "123")
f-6     (attack-sig "cve" "CAN-2002-0826" "123")
f-7     (attack-sig "xforce" "xforce9794" "123")
f-8     (vulnerable "Socks Any version" "xforce9794" "123")
f-9     (vulnerable "Windows 2000 Any version" "xforce9794" "123")
f-10    (vulnerable "Windows NT Any version" "xforce9794" "123")
f-11    (vulnerable "Windows XP" "xforce9794" "123")
f-12    (vulnerable "Windows NT 3.5" "xforce9794" "123")
f-13    (vulnerable "Windows NT 3.51" "xforce9794" "123")
f-14    (vulnerable "Windows NT 4.0" "xforce9794" "123")
f-15    (vulnerable "Windows NT 4.0 Enterprise" "xforce9794" "123")
f-16    (vulnerable "Windows NT 4.0 Option Pack" "xforce9794" "123")
```

```
f-17    (vulnerable "Windows NT 4.0 SP1" "xforce9794" "123")
f-18    (vulnerable "Windows NT 4.0 SP2" "xforce9794" "123")
f-19    (vulnerable "Windows NT 4.0 SP3" "xforce9794" "123")
f-20    (vulnerable "Windows NT 4.0 SP5" "xforce9794" "123")
f-21    (vulnerable "Windows NT 4.0 SP6" "xforce9794" "123")
f-22    (vulnerable "Windows NT 4.0 SP6a" "xforce9794" "123")
f-23    (vulnerable "Windows NT 4.0 TSE" "xforce9794" "123")
f-24    (vulnerable "Windows NT 4.0 beta" "xforce9794" "123")
f-25    (vulnerable "Windows 2000 Advanced Server" "xforce9794" "123")
f-26    (vulnerable "Windows 2000 Beta" "xforce9794" "123")
f-27    (vulnerable "Windows 2000 Datacenter Server" "xforce9794"
"123")
f-28    (vulnerable "Windows 2000 Professional" "xforce9794" "123")
f-29    (vulnerable "Windows 2000 SP1" "xforce9794" "123")
f-30    (vulnerable "Windows 2000 SP2" "xforce9794" "123")
f-31    (vulnerable "Windows 2000 SP3" "xforce9794" "123")
f-32    (vulnerable "Windows 2000 Server" "xforce9794" "123")
f-33    (vulnerable "Windows 2000 Terminal Services" "xforce9794"
"123")
f-34    (platform-specific-attack "xforce9794" "123")
f-35    (service-specific-attack "xforce9794" "123")
```

This forward chaining of facts allows us to see the exact mechanism with which the Vulnerability Correlation algorithm works. However, if these intermediate facts are not removed from the system after the meta-alert is produced, it will clutter the CLIPS Engine, and hurt the performance of the system.

The important question to consider is: if the meta-alert is (or is not) produced, which of these facts do we want to keep in the system afterwards?

The answer that I suggest is the following: it depends on whether we intend to perform further correlation with these intermediate facts. If we do not need them anymore, we delete them immediately.

It's pretty safe to say that we will not be needing the intermediate facts that state the vulnerable platforms/services (f-8 through f-33), so we can retract these from the system. We probably also don't want the attack-sig conversion rules (f-4 through f-7) any more, so we can retract these as well. In most cases, assuming that if we dynamically add new rules, they are not required to back-correlate with past alerts, we can also remove the initial alert from the system.

If we decide to further correlate the platform- and service-specific-attacks with other (meta-)alerts, we might keep facts f-34 and f-35 a bit longer in the system. However, the correlation rules that use the platform- and service- specific attacks as input are then responsible for removing these facts from the system after using them, so perhaps a second metaalert will be produced, and then the second set of correlation rules will delete facts f-34 and f-35 from the system as well.

The moral of the story is that we only keep facts in the system that represent an intermediate state that might lead to a new meta-alert correlation, but that is still waiting

for further necessary input. This is the only way to keep the system state maintainable when sending a huge number of IDS alerts through, over long periods of time.

25.2 Fact Retraction Rules

Here are the rules that we need to add to perform the intermediate fact retractions from the last section:

```
; Remove basic attack alerts
(defrule remove-basic-alerts
  (declare (salience -1))
  ?ba <- (attack)
  =>
  (retract ?ba))

; Remove attack-sig correlation alerts
(defrule remove-attack-sig-alerts
  (declare (salience -1))
  ?ba <- (attack-sig $?)
  =>
  (retract ?ba))

; Remove vulnerability information
(defrule remove-vulnerability-information
  (declare (salience -1))
  ?ba <- (vulnerable $?)
  =>
  (retract ?ba))

; Remove attacked service information
(defrule remove-attacked-services
  (declare (salience -1))
  ?ba <- (attack-service $?)
  =>
  (retract ?ba))

; Remove attacked platform information
(defrule remove-attacked-platforms
  (declare (salience -1))
  ?ba <- (attack-platform $?)
  =>
  (retract ?ba))
```

Note that these rules have a "salience" level of –1. Salience is the priority value of rules on the agenda, that determines the order in which the rules will be fired. CLIPS assigns rules a default salience value of 0, where the possible salience values range from –10,000 to 10,000.[28]

The salience value of –1 assures that the rest of the meta-alert correlations will occur BEFORE the facts are removed from the system. The salience values of the meta-alert correlation rules themselves, since we did not define them explicitly, have the default

value of 0. Therefore all of the meta-alert correlation rules will be fired first, before the first rule fires that leads to an intermediate fact being retracted.

26. Portscan Correlation (Alert Counting)

Another example of how CLIPS can be used for correlation is to create Portscan meta-alerts.

26.1 Simple IP Address Counting

26.1.1 Initial Assertions

The first kind of Portscan correlation that we can perform is automatically looking for a certain number (or multiple) of alerts originating or targeting a particular IP address. This basically entails maintaining a count of alerts coming from/to a particular IP address.

Maintaining the count of source/destination IP addresses begins by asserting a *count-ip-address-src* or *count-ip-address-dest* fact, holding a count of zero, every time that a new alert enters the system. This occurs using the following rules:

```
(defrule count-ip-address-src-new
  (idmef_address (mid ?mid_value) (index ?address_index) (address
    ?ip_addr))
  (idmef_node (mid ?mid_value) (index ?node_index) (Address
    ?address_index))
  (idmef_source (mid ?mid_value) (Node ?node_index))
  =>
  (assert (count-ip-address-src ?ip_addr 0)
          (control-ip-src ?ip_addr)))


(defrule count-ip-address-dest-new
  (idmef_address (mid ?mid_value) (index ?address_index) (address
    ?ip_addr))
  (idmef_node (mid ?mid_value) (index ?node_index) (Address
    ?address_index))
  (idmef_target (mid ?mid_value) (Node ?node_index))
  =>
  (assert (count-ip-address-dest ?ip_addr 0)
          (control-ip-dest ?ip_addr)))
```

Note that we also assert facts called *control-ip-src* and *control-ip-dest*, along with the *count-ip-address-* facts. These facts are called “control facts”. Control facts are intermediate facts that are used solely to enable and disable the firing of rules. In this case, these control facts are used to keep rules from refiring every time a fact is retracted that which caused earlier firing of the rule.

26.1.2 Counting With Control Facts

If we have two *count-ip-address-src/dest* facts with the same IP address, we will want to combine them to reflect the current count. This is achieved by having rules that retract the original facts (with the lower count value), and reasserting new facts with a count value one higher than then highest value of the two counts that we are combining. Since we only want this combination to occur once per newly asserted alert, we also require the presence of a control fact as a requirement to fire. After the count combination rule fires once, the control fact is then asserted.

The following two rules illustrate the use of count combination facts for source/destination IP addresses:

```
(defrule count-ip-address-src
  ?count-ip-src <- (count-ip-address-src ?ipsrc ?count)
  ?count-ip-src2 <- (count-ip-address-src ?ipsrc ?count2&0)
  ?control-fact <- (control-ip-src ?ipsrc)
  =>
  (retract ?count-ip-src ?count-ip-src2 ?control-fact)
  (assert (count-ip-address-src ?ipsrc (+ 1 (max ?count ?count2)))))

(defrule count-ip-address-dest
  ?count-ip-dest <- (count-ip-address-dest ?ipdest ?count)
  ?count-ip-dest2 <- (count-ip-address-dest ?ipdest ?count2&0)
  ?control-fact <- (control-ip-dest ?ipdest)
  =>
  (retract ?count-ip-dest ?count-ip-dest2 ?control-fact)
  (assert (count-ip-address-dest ?ipdest (+ 1 (max ?count ?count2)))))
```

26.1.3 Reporting Multiple Attacks

Now that we have facts in the system that keep count of the number of source/destination IP addresses, we want to produce a meta-alert every time that a desired number of alert originate from a specified source/destination IP address.

As an example, the following two rules produce meta-alerts every time that 10 attacks appear from a source/destination IP address.

```
  (count-ip-address-dest ?ipdest ?count)
  (test (eq (mod ?count 10) 0))
  =>
  (printout t "Metaalert:" ?count " attacks targeting destination IP
address:")
  (printout t ?ipdest crlf))

  (count-ip-address-src ?ipsrc ?count)
  (test (eq (mod ?count 10) 0))
  =>
  (printout t "Metaalert:" ?count " attacks from source IP address:")
  (printout t ?ipsrc crlf))
```

26.1.4 Comments

While it is nice to know that CLIPS can be used to maintain such counts of alerts, it is actually not the most useful feature that MACE has to offer. Because one count fact will exist in the system per source/destination IP address, this kind of counting will tend to internally clutter the system if a reasonable amount of traffic is being monitored. Additionally, a SOC operator is usually not interested in knowing every 10 times (or 100 times) that an alert appears from a given source/destination IP address. This is especially true when hundreds or thousands of alerts appear in connection with an IP address. Therefore, this kind of counting is best reserved for keeping tabs on a single IP address of interest.

26.2 Counting Using Time Intervals

Another feature of interest to a SOC operator might be knowing when a given number of attacks F, target a single IP address in a specified time interval T. This sort of counting can be performing using sliding time intervals.

26.1.1 Initial Interval Assertion

This kind of counting begins in much the same way as with the simple counting of IP addresses. We want to assert a count-attack fact every time that an attack enters the system. Note that no control facts are asserted by these initial assertions, because the later count combination rules do not fire more than once as the facts are retracted and reasserted. (See next section.)

This initial assertion is performed by the following rule:

```
(defrule counttime-attack-aggregation-fact
  (idmef_address (mid ?mid_value) (index ?address_index) (address
    ?ip_addr))
  (idmef_node (mid ?mid_value) (index ?node_index) (Address
    ?address_index))
  (idmef_target (mid ?mid_value) (Node ?node_index))
  (idmef_time (mid ?mid_value) (index ?createtime)(unix_timestamp
    ?ts))
  (idmef_alert (mid ?arg_mid)(Createtime $?ctbefore ?createtime
    $?ctafter))
=>
  (assert (count-attack ?ip_addr ?ts ?mid_value)))
```

Once this rule is in the system, we will then want to create an interval fact, that corresponds to a specific destination IP address, and stores the beginning and ending timestamps of the alert, along with the list of alert identifiers that identify alerts falling into this time interval.

This time interval is generated from the initial count-attack fact with the following rule, that creates intervals from alerts targeting an IP address that occur within 60 seconds of each other:

```
(defrule counttime-ip-address-dest-new
  ?ca1 <- (count-attack ?ipdest ?ts ?mid)
  ?ca2 <- (count-attack ?ipdest ?ts2 ?mid2)
  (test (< (- ?ts ?ts2) 60))
  (test (>= (- ?ts ?ts2) 0))
  (test (neq ?mid ?mid2))
  =>
  (assert (counttime-ip-address-dest ?ipdest ?ts2 ?ts ?mid2 ?mid))
  (retract ?ca1 ?ca2))
```

We can see that the initial count-attack facts are retracted after the time interval, and that the timestamps are stored in low/high order in the new interval fact.

26.1.2 Interval Combination

Now that we have the initial time intervals of size 2, we want to combine them into larger time intervals, when they overlap with other existing time intervals but are still within the desired time interval T.

First, we need a rule that can check, every time that a new alert is asserted, whether is falls within the two timestamps of an existing *counttime* set. If this is the case, it would need to retract the old *counttime* set, and assert a new one that includes the new alert identifier in the set.

This functionality is performed with the following rule:

```
(defrule counttime-ip-address-dest
  (idmef_address (mid ?mid_value) (index ?address_index) (address
    ?ip_addr))
  (idmef_node (mid ?mid_value) (index ?node_index) (Address
    ?address_index))
  (idmef_target (mid ?mid_value) (Node ?node_index))
  (idmef_time (mid ?mid_value) (index ?createtime)(unix_timestamp
    ?ts3))
  (idmef_alert (mid ?arg_mid)(Createtime $?ctbefore ?createtime
    $?ctafter))
  ?ct <- (counttime-ip-address-dest ?ip_addr ?ts2 ?ts $?mids)
  (test (<= ?ts3 ?ts))
  (test (>= ?ts3 ?ts2))
  (test (eq (member$ ?mid $?mids) FALSE))
  =>
  (retract ?ct)
  (assert (counttime-ip-address-dest ?ip_addr ?ts2 ?ts ?mids ?mid)))
```

26.1.3 Interval Reduction

As we execute these rules, we will find that counttime sets will be produced that are exact subsets of each other. For example, we might see the two sets:

```
(counttime-ip-address-dest "193.67.146.17" 1044842787 1044842788 "2-
97309" "2-97310" "2-97311")
```

and

```
(counttime-ip-address-dest "193.67.146.17" 1044842787 1044842788 "2-
97309" "2-97310")
```

In this case, we will clearly want to remove the second set. We can perform this reduction with the following rule:

```
(defrule counttime-ip-address-dest-remove-subsets
  ?ct <- (counttime-ip-address-dest ?ipdest ?ts ?ts2 $?mids)
  ?ct2 <- (counttime-ip-address-dest ?ipdest ?ts3 ?ts4 $?mids2)
  (test (neq ?ct ?ct2))
  (test (eq (subsetp $?mids $?mids2) TRUE))
  =>
(if
  (<= (length$ ?mids) (length$ ?mids2))
then
  (retract ?ct)
else
  (retract ?ct2)
))
```

Note that it is also a desirable thing to combine and reduce intervals that are not subsets, but that contain overlapping timestamps, still falling within a total time interval of T.

26.1.4 Generating Metaalerts

Now that we are keeping a record of sliding time intervals for all alerts targeting a particular destination IP address, we will want to generate a meta-alert every time that the desired frequency F is reached within this time interval. We can do this using the following rule:

```
(defrule counttime-ip-address-dest-reached-frequency
  ?ct <- (counttime-ip-address-dest ?ipdest ?ts2 ?ts $?cids)
  (test (>= (length$ $?cids) 20))
  =>
  (retract ?ct)
  (printout t "Metaalert: Alerts exceeded frequency threshold:" ?cids
crlf))
```

26.1.5 Comments

This type of counting is probably more interesting for a SOC operator than the Simple IP Address Counting that was described earlier. The values of T and F are configurable, potentially on an IP address basis. However, with this kind of counting, we also run the

risk of cluttering the system with a large amount of *count-attack* and *counttime* interval facts. A good scheme to control this would be to use fact retraction rules to remove the time intervals, not only as meta-alerts are produced, but also as they stay in the system a certain amount of time without generating a metaalert.

My general attitude for Portscan detection and alert counting is that it is probably better to use an external procedural program to do the portscan detection (ex. The Snort portscan detection preprocessor), and then to send that output to MACE, using IDMEF for further processing. However, if the SOC operator wants to do it within MACE, it is possible, although caution much be exercised in implementing the expert system rules to prevent the CLIPS Engine from becoming too cluttered with partially-fulfilled intermediate facts.

Part VIII – Results and Testing

27. Overview

The upcoming section of this report will discuss system-testing of MACE using the Snort-Hal and DUNET-database data sets. The first section will take a quick look at the purely technical performance of MACE. Next, there is a slightly more elaborated description of the two data sets than before, and a discussion ensues detailing some of the various filtering / correlation techniques and their relative effectiveness. Lastly, we will look at some disadvantages of the presented correlation techniques.

28. Technical Testing

The following chapter discusses the testing process for some of the purely *technical* aspects of the Meta-Alert Correlation Engine.

28.1 Memory Testing

The following section discusses the Meta-Alert Correlation Engine's memory usage. This section, besides providing some memory-usage statistics, also highlights one of the tools that was used to check and improve MACE's memory performance.

28.1.1 Boehm's Garbage Collector

The primary function of the Boehm-Demers-Weiser "garbage collector" is to report memory objects that were allocated but never deallocated, and that are no longer accessable to the program. This works by offering a macro replacement for the C *malloc/free* and C++ *new/delete* function calls. The garbage collector works differently from most "counting leak" detectors, which verify that all allocated objects are eventually deallocated by process exit time. With the garbage collector, "permanent" data structures that are used and accessable throughout the program are not reported as "leaks" and are not required to be deallocated at the end – a potentially useless activity that often triggers large amounts of paging. The garbage collector uses the mark-sweep algorithm, providing incremental and generational collection under OS's with the right kind of virtual memory support. (Linux/Windows/OpenBSD/etc..)[29]

I am using the Boehm-Demers-Weiser conservative garbage checker to check MACE memory usage, and to find (and fix) memory leaks.

28.1.2 Memory Usage Statistics

MACE, including all of the various components, leaks about .05 M of memory every second. With 140 M of memory (plus 1024 M of swap memory) in my computer, this rate of leakage has not yet become a problem for my testing. Once MACE becomes a bit

more stable of a program, (that runs in the background for longer periods of time), I will go back and tune the memory performance a bit more.

28.2 CPU Utilization

I am developing and testing the Meta-Alert Correlation Engine on a 1U Rackmount Dell Pentium 3, with a 1 Gig processor, and 6250 Mb internal memory.

The following entry shows a typical *NIX 'top' display on this machine when MACE is correlating the DUNET test data:

```
processes:  2 running, 42 idle
CPU states: 45.5% user, 18.4% nice, 34.1% system,  0.0% interrupt,
2.0% idle
Memory: Real: 100M/141M act/tot  Free: 106M  Swap: 108M/1024M used/tot

PID USER   PRI NICE  SIZE   RES STATE WAIT  TIME    CPU COMMAND
8683  root  61    0   24M   25M   run    -   2:37  44.78% clips_engine
18863 mysql  2    4  398M   12M sleep  poll 89:43  19.68% mysqld
21574 root   2    0   84K  716K sleep netio  0:13  11.47% clips_output
28705 root   2    0 3136K 4316K sleep netio  0:06   5.91%get_snort_aler
31876 root   2    0 4084K 4104K sleep netio  0:01   0.98% mace_server
11279 root   2    0  772K 1528K sleep netio  0:01   0.83% clips_server
21191 root   2    0 3596K 4472K sleep netio  0:01   0.10% clips_parse
```

As we can see, MACE is using half of the total CPU power of this machine. The CLIPS Engine uses a bit less than 50% of that CPU power, and the MySQL daemon another 20%. The other 30% is used by the other 5 components of MACE. The size of the CLIPS Engine and MySQL daemon values are actually quite variable, depending on how much data is being queried from the database, and on how the correlation itself is being carried out.

There are some rules of thumb that the MACE user can use to keep CPU usage low:

1. **Do not leave excess facts in the expert system.**
   Nothing will bring the system to a grinding halt as fast as using expert system rules that are sloppily written, and that leave all kinds of data in CLIPS after it's no longer needed. (The more matches that the inference engine makes, the more CPU power is necessary.)

2. **Do not use a graphical Meta-Alert viewer if the database is too full.**
   For a smaller amount of alerts (a few hundred thousand), the graphical viewer gives an excellent response time. However for larger amounts of alerts (millions), the graphical user interface executes some extremely long queries, that eat up lots of CPU power. If the Meta-Alert database ever becomes problematically too large, this can be mitigated by splitting up the database into multiple smaller databases.

3. **Filter out alerts with the Preprocessing Module if possible.**
   Both the Preprocessing Module and the CLIPS Engine are capable of filtering out primary alerts. However, using the Preprocessing Module saves a lot of processing power. First, it gets the alert "out of the system" at an earlier stage, eliminating the need for processing in the other MACE components. Secondly, the preprocessor uses procedural methods to perform the filtering checks. This is generally more streamlined than performing the same checks within the context of the expert system.

29. A Closer Look at the Datasets

29. 1 Fox-IT Snort-Hal Dataset

As mentioned in Section 10.2, Fox-IT (Forensic IT Experts) has granted me use of their client intrusion detection databases for testing and correlation purposes. The following sections describe the composition of this data, and the techniques that I used to filter through and correlate this data.

29.1.1 IDS Setup

The input data from the Fox-IT Hal machine consists of Snort NIDS alerts, taken from multiple sensors. The alerts are generated with the following Snort rulesets:

```
bad-traffic, exploit, scan, finger, ftp, telnet, rpc, rservices, dos,
ddos, dns, tftp, web-cgi, web-coldfusion, web-iis, web-frontpage, web-
misc, web-client, web-php, sql, x11, netbios, misc, attack-responses,
oracle, mysql, snmp, smtp, imap, pop3, nntp, other-ids, web-attacks,
backdoor, shellcode, policy, info, virus, chat, multimedia, p2p,
experimental, local
```

Additionally, the following preprocessors are used to generate extra information:

```
frag2, stream4: detect_scans, disable_evasion_alerts,
stream4_reassemble, http_decode: 80 unicode iis_alt_unicode
double_encode iis_flip_slash full_whitespace, bo: -nobrute
telnet_decode, portscan: $HOME_NET 4 3 portscan.log, portscan-
ignorehosts: $PORTSCAN, conversation: allowed_ip_protocols all, timeout
60, max_conversations 32000
```

29.1.2 Data Composition

The Snort-Hal database consists of 188344 total Snort + preprocessor alerts, collected within the span of approximately a month.

188204 of the alerts are generated straight from the Snort rulesets, without the aid of a preprocessor. Figure 28 shows the top 20 most-frequently-occurring Snort signatures, along with their frequency within the Snort-Hal database.

Signature Name	Frequency
WEB-MISC net attempt	49786
MISC Tiny Fragments	13368
WEB-IIS cmd.exe access	13077
WEB-MISC robots.txt access	10626
SCAN SOCKS Proxy attempt	9050
ICMP PING NMAP	8363
WEB-MISC http directory traversal	7682
SCAN Proxy (8080) attempt	7365
WEB-IIS multiple decode attempt	6764
P2P GNUTella GET	6724
SCAN Squid Proxy attempt	5159
ICMP Destination Unreachable (Communication Administratively Prohibited)	3763
BAD TRAFFIC tcp port 0 traffic	3369
ICMP superscan echo	3243
WEB-IIS scripts access	2760
ICMP PING CyberKit 2.2 Windows	2309
POLICY FTP anonymous login attempt	2167
EXPERIMENTAL WEB-CLIENT javascript URL host spoofing attempt	1801
WEB-MISC ?open access	1795
WEB-IIS ISAPI .ida access	1670

Figure 28 – Top 20 Non-Portscan Fox-IT Alert Signatures

The last 140 Snort alerts are generated by preprocessors, mostly indicating the presence of portscans. Portscan preprocessors (such as spp_portscan2) generate alerts that contain information in approximately the following format:

```
Portscan detected from 65.37.167.226: 21 targets 21 ports in 3 seconds
0 NULL 1 1
```

29. 2 DUNET-Database Dataset

29.2.1 IDS Setup

The input data from the DUNET-database machine consists of Snort NIDS alerts, taken from a single sensor on the DUNET spanport. Alerts are generated using with the following Snort rulesets:

```
bad-traffic, exploit, scan, finger, ftp, telnet, rpc, rservices, dos,
ddos, dns, tftp, web-cgi, web-coldfusion, web-iis, web-frontpage, web-
misc, web-client, web-php, sql, x11, netbios, misc, attack-responses,
oracle, mysql, snmp, smtp, imap, pop3, nntp, other-ids, web-attacks,
backdoor, shellcode, policy, info, virus, chat, multimedia, p2p,
experimental, local
```

No preprocessors are used on the DUNET sensor.

29.2.2 Data Composition

The DUNET database consists of 4005587 total Snort alerts, collected within the span of about 3-4 months.

Figure 29 shows the top 20 most-frequently-occurring Snort signatures, along with their frequency within the DUNET-database.

Signature Name	Frequency
SNMP trap udp	626498
ICMP Large ICMP Packet	536033
WEB-IIS scripts access	490063
NETBIOS NT NULL session	390689
SNMP AgentX/tcp request	294589
SHELLCODE x86 NOOP	270347
ICMP PING speedera	248427
ORACLE select union attempt	223967
ICMP Destination Unreachable (Communication Administratively Prohibited)	176123
WEB-IIS cmd.exe access	127335
ICMP L3retriever Ping	65521
SHELLCODE x86 unicode NOOP	55137
ORACLE all_constraints access	53057
SHELLCODE x86 inc ebx NOOP	49992
WEB-ATTACKS /bin/ps command attempt	37363
ATTACK RESPONSES 403 Forbidden	35431
ICMP Source Quench	34942
WEB-CGI archie access	32770
ICMP PING WhatsupGold Windows	23563
SNMP public access udp	23095

Figure 29 – Top 20 DUNET Alert Signatures

Since the preprocessors were turned off, there are no portscan alerts.

30. Filtering and Correlation Testing

The upcoming chapter describes the process that was used to filter and correlate the data from each of the datasets, starting with the Fox-IT Snort Hal dataset.

30.1 Simple NIDS Signature Filtering

After my first discussion with Erwin Fok, the Fox-IT SOC Operator, he told me that there are a number of IDS alerts that Snort generates to the database, for the sake of documentation and completeness, but that he never looks at because they generate large amounts of false alarms and do not pose a real potential threat to the systems that Fox-IT is monitoring.

The Snort rules that are always ignored are the following:

Snort ID	Snort Signature Description
469	ICMP PING NMAP
474	ICMP superscan echo
478	ICMP Broadscan Smurf Scanner
485	ICMP Destination Unreachable (Communication Administratively Prohibited)
524	BAD-TRAFFIC tcp port 0 traffic
553	POLICY FTP anonymous login attempt
615	SCAN SOCKS Proxy attempt
618	SCAN Squid Proxy attempt
620	SCAN Proxy \(8080\) attempt
648	SHELLCODE x86 NOOP
881	WEB-CGI archie access
882	WEB-CGI calendar access
895	WEB-CGI redirect access
971	WEB-IIS ISAPI .printer access
1002	WEB-IIS cmd.exe access
1057	WEB-MISC ftp attempt
1062	WEB-MISC nc.exe attempt
1201	ATTACK-RESPONSES 403 Forbidden
1213	WEB-MISC backup access
1214	WEB-MISC intranet access
1242	WEB-IIS ISAPI .ida access
1243	WEB-IIS ISAPI .ida attempt
1256	WEB-IIS CodeRed v2 root.exe access
1287	WEB-IIS scripts access
1288	WEB-FRONTPAGE /_vti_bin/ access
1322	BAD-TRAFFIC bad frag bits
1390	SHELLCODE x86 inc ebx NOOP
1394	SHELLCODE x86 NOOP
1425	WEB-PHP content-disposition
1432	P2P GNUTella GET
1560	WEB-MISC /doc/ access
1841	WEB-CLIENT Javascript URL host spoofing attempt
1881	WEB-MISC bad HTTP/1.1 request, Potentially worm attack

Figure 30 – Snort Rules filtered out by the First Sweep

Plugins were created to automatically filter out each of these Snort alerts. After running the alerts from the Snort Hal database through MACE, with these plugins activated, only 108649 of the original 188344 alerts were remaining. (42% reduction)

30.2 NIDS Host/Vulnerability Correlation

Some alerts with specific signatures cannot be automatically filtered out, just because they generate a large amount of data. If an alert with a particular signature poses a potential threat to the computer(s) that it is targeting, it doesn't matter how many false positives may occur.. a more careful look is warranted!

This is where the signature vs. vulnerability matching (described in Chapter 24) can be useful, creating meta-alerts to bring special attention to alerts with a higher probability of indicating an actual break-in.

30.2.1 Collecting Vulnerability Information

There are several manners of collecting vulnerability information. One specific method is to use a vulnerability scanner. These 'scanners' can run through a sequence of IP addresses, checking each computer for open ports and services. Some popular vulnerability scanners include Nmap and Nessus.

Fox-IT runs periodic vulnerability scans on their client machines. Figure 31 shows the results of a periodic Nessus scan on the Fox-IT demilitarized zone network segment (dmz).

Sensor	Hostname	IP Address	Protocol	Port	Service
nessus-dmz		195.64.85.113	udp	123	ntp
nessus-dmz		195.64.85.113	tcp	22	ssh
nessus-dmz		195.64.85.116	tcp	113	auth
nessus-dmz		195.64.85.116	tcp	22	ssh
nessus-dmz		195.64.85.122	tcp	21	ftp
nessus-dmz		195.64.85.122	tcp	25	smtp
nessus-dmz		195.64.85.122	tcp	22	ssh
nessus-dmz		195.64.85.68	tcp	22	ssh
nessus-dmz	DUNET-database	195.64.85.69	tcp	22	ssh
nessus-dmz		195.64.85.70	tcp	1723	pptp
nessus-dmz		195.64.85.71	tcp	22	ssh
nessus-dmz		195.64.85.74	tcp	443	https
nessus-dmz		195.64.85.74	tcp	22	ssh
nessus-dmz		195.64.85.74	tcp	80	www
nessus-dmz		195.64.85.75	tcp	22	ssh
nessus-dmz	mail2.fox-it.com	195.64.85.66	tcp	53	domain
nessus-dmz	mail2.fox-it.com	195.64.85.66	tcp	53	domain
nessus-dmz	mail2.fox-it.com	195.64.85.66	udp	53	domain
nessus-dmz	mail2.fox-it.com	195.64.85.66	tcp	25	smtp
nessus-dmz	mail2.fox-it.com	195.64.85.66	tcp	25	smtp
nessus-dmz	mail2.fox-it.com	195.64.85.66	tcp	22	ssh
nessus-dmz	mail2.fox-it.com	195.64.85.66	tcp	22	ssh

Figure 31 – Nessus Vulnerability Scan Results on Fox-IT DMZ

We see that this Nessus scan reveals information about the DMZ machines (hostname/ip address) and it provides a list of their open (or obviously filtered) TCP/UDP ports.

I have extensive information about the machines on the 130.161.180. and 130.161.181. subnets of the DUNET network, via an nmap scan performed by Lolke Boonstra. I will

use the DUNET-database data in the upcoming examples to demonstrate how to perform vulnerability / attack correlation.

30.2.2 Creating Vulnerability Facts

The nmap scans from the DUNET-database contain entries that look like something like the following example:

```
Interesting ports on ns1.tudelft.nl (130.161.180.1):
(The 6 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
22/tcp     open        ssh
23/tcp     open        telnet
25/tcp     open        smtp
Remote operating system guess: Sun Solaris 8 early acces beta through
actual release
Uptime 126.318 days (since Fri Oct 18 08:27:17 2002)
```

While the information from the nmap scan may not be 100% accurate, this gives us enough information to perform correlation between attacks and their target computers. (This is also approximately the same information that a hacker has at his/her disposal, so even if some of the information here is false, we can still get an insight about how seriously a hacker is attempting to break in.)

The results of the Nmap scan can be used to automatically create expert system facts. The above shown nmap example can be represented by the following expert system fact:

```
(deffacts define-dunet-network-info-subnet-180-example
  (system-info
    (my-ip-address "130.161.180.1")
    (my-operating-system
     "Sun Solaris 8" "Solaris 8" "Solaris Any version"
    )
    (my-services "ftp" "ssh" "telnet" "smtp" "http")))
```

In order to perform the alert correlation in the upcoming section, I created one expert-system fact per Nmap entry for each of the DUNET machines.

30.2.3 Results of Vulnerability Correlation

After entering the platform / service expert system facts into CLIPS, I processed a portion of the DUNET-database alerts with MACE, using the vulnerability / attack correlation scheme described in Chapter 24.

In the following section, I have run portions of the DUNET-database alerts through MACE. I have not used the entire collection of DUNET alerts for the following reason: it would take an entire workweek to run all of the ~4 million DUNET-database alerts through the MACE system. I will do this at a later point in time, when a more stable

user interface is created (ACID, which I have temporarily borrowed as a GUI, leaves database connections open, resulting in infrequent but spontaneous MySQL aborts). In the meanwhile, I have performed the correlation and testing and smaller subsets of the total DUNET-database dataset.

These are the results of one example MACE run: When running the DUNET-database alerts through the system over the time-span of 6 hours and 50 minutes, the system processed 222469 alerts, and generated 493 meta-alerts. (The system processed about 9 raw IDS alerts per second). This is a 99.78% reduction in alert data.

Some of the "platform-matching" meta-alerts that were created are:

- NetBIOS NT NULL session - (Matched when targeting Windows NT machines)
- SNMP AgentX/tcp request – (Matched when targeting Solaris 8 machines)
- WEB-FRONTPAGE _vti_rpc access – (Matched when targeting Windows NT machines)
- WEB-IIS File permission canonicalization – (Matched when targeting Windows NT machines)
- WEB-IIS view source via translate header – (Matched when targeting Windows NT machines)

The following "service-matching" meta-alerts were created:

- SNMP public access udp – (Matched when targeting snmp)
- WEB-COLDFUSION administrator access - (As a test, I added "Coldfusion" to some of the DUNET computer service profiles. These were indeed picked out and correlated into meta-alerts by MACE.)

Figure 32 shows a screenshot of ACID, displaying the MACE-generated metaalerts:

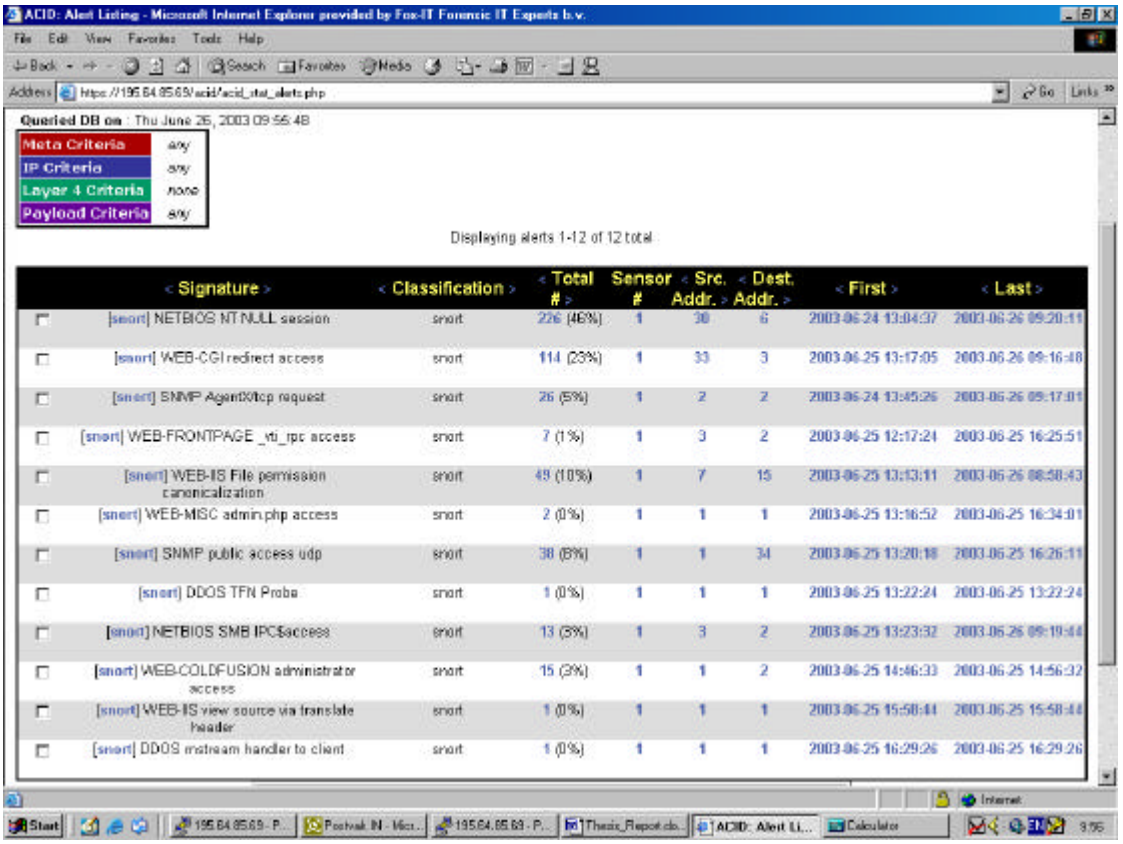


Figure 32 – Using ACID to Display Meta-Alerts

(Note: I'm only using ACID on a temporary basis to display the meta-alerts. It is obviously not designed to represent IDMEF, host-based or other tool-based alert data. Within the upcoming months, I will design my own PHP-based user interface, that represents things based upon the alerts' IDMEF-representation in the Metaalert database. However, my future web interface was not realizable in time for submission of this Masters thesis report.)

30.2.4 Target Vulnerability Reports

MACE output can also be used to generate reports specifying the platforms and services that are targeted the most frequently. Figure 33 shows a list of the top 10 most-frequently targeted services, as seen in a sample of the DUNET-database attacks:

Service Targeted	*Frequency*
Php	27.5%
Microsoft IIS 5.0	15.0%
Microsoft IIS 4.0	15.0%
Common Gateway Interface (CGI) Any version	13.8%
Coldfusion Any version	10.0%
Formmail Any version	8.8%
Snmpv1 Any version	2.5%

Red Hat Powertools 7.1	2.5%
Oracle Enterprise Manager	2.5%
Hp OpenView Network Node Manager	2.5%

Figure 33 – Ten Most-Frequently Vulnerable Services (DUNET-database)

Figure 34 shows an example list of the top 20 most-frequently targeted platforms, as seen in a sample of the DUNET-database attacks:

Platform Targeted	*Frequency*
Windows NT Any version	7.0%
Windows 2000 Any version	7.0%
Windows 98	6.3%
Windows 95	6.3%
Windows for Workgroups 3.11	5.9%
Windows Any version	5.9%
Samba Any version	5.9%
Os/2 Any version	5.9%
Linux kernel 2.0.x	5.9%
Suse Linux	4.4%
Red Hat Linux	4.4%
Mandrake Single Network Firewall	4.4%
Mandrake Linux Corporate Server	4.4%
Mandrake Linux	4.4%
Debian Linux	4.4%
Caldera OpenLinux Workstation	4.4%
Caldera OpenLinux Server	4.4%
Trustix Secure Linux	4.1%
Engarde Secure Linux	4.1%

Figure 34 – Twenty Most-Frequently Vulnerable Platforms (DUNET-database)

31. Filtering and Correlation Problems

At first appearances, the data reduction rate is fantastic (over 99%), and the generated alerts, when traced back through the MACE logs, actually correlate to known vulnerabilities in the monitored systems. However, this is not all cause for celebration. Besides the few false positives that haven't been correlated away, there is the much larger problem of false negatives in the metaalert data. The next few sections will explain some of the problems that still need to be overcome in order to make Attack / Vulnerability meta-alert correlation effective.

31.1 Insufficient Platform/Service Information

In my opinion, one of the largest problems with attack / vulnerability correlation is that we can only create meta-alerts for machines that we have information about. This point may sound obvious, but I didn't realize how large of a problem this would be until I looked at the Fox-IT client vulnerability-scan data.

The vast majority of the client machines that were "Nessus" (vulnerability) scanned are either behind a firewall (blocking the scans), or Fox-IT does not have permission to run vulnerability scans on these machines. (The client is sometimes worried that the vulnerability scan may crash or otherwise disrupt critical machines on the network.) The only network segment that Fox-IT had adequately scanned was Fox-IT's own demilitarized zone (see Figure 29). However, due to low network traffic levels, not a single of the Snort "attacks" correlated with the correct platform/service. Therefore, not a single meta-alert was produced.

If a SOC operator is relying on MACE as his/her primary source of information, this presents a large problem. Additionally, this lack of information / customer cooperation is also apparently not exclusively Fox-IT's problem. I learned from conversations with other third-party SOC operators, that this difficulty is encountered frequently, and that creativity and supplementary tools are required to gather this elusive information. Additionally, this lack of platform/service context information not only hinders automated monitoring, but it causes problems with manual correlation efforts as well.

I believe that the platform/service information can be filled in through other means, that a client may be more agreeable to. For example, "passive" methods can be used to determine the host platforms and services. Since a SOC already has permission from the client to sniff the network traffic, a program can always be used to monitor the port traffic and to record the service banners that get sent across the wire. A database can then store this information, and upon request, automatically convert it into the expert system rules.

It appears to me that some kind of "big-picture" solution needs to be created, to collect and store current information about managed computers and networks, as well as collecting and managing information about Intrusion Detection Alerts themselves. Additionally, I believe that host-based monitoring (and host-based platform/service information collection tools), along with other methods of intrusion detection (anomaly-based IDS, bandwidth analysis, traffic-policy violation analysis, portscan analysis, etc..) can be used to help to fill in this incomplete picture.

31.2 Insufficient Signature Conversion Information

Another problem that I have encountered, that has caused the undesirable filtering-out of crucial alerts, is incomplete information with regards to cross-correlation with vulnerabilities and more standardized alert identifiers (ex. CVE). I have used Snort as my primary alert source until now, and there are some serious gaps in the chain of information conversion that is required to produce meta-alerts.

For an example, we can take a look at our source for Snort alert conversion information: the Snort webpage. Figure 35 shows a not-so-uncommon example of Snort rule documentation:

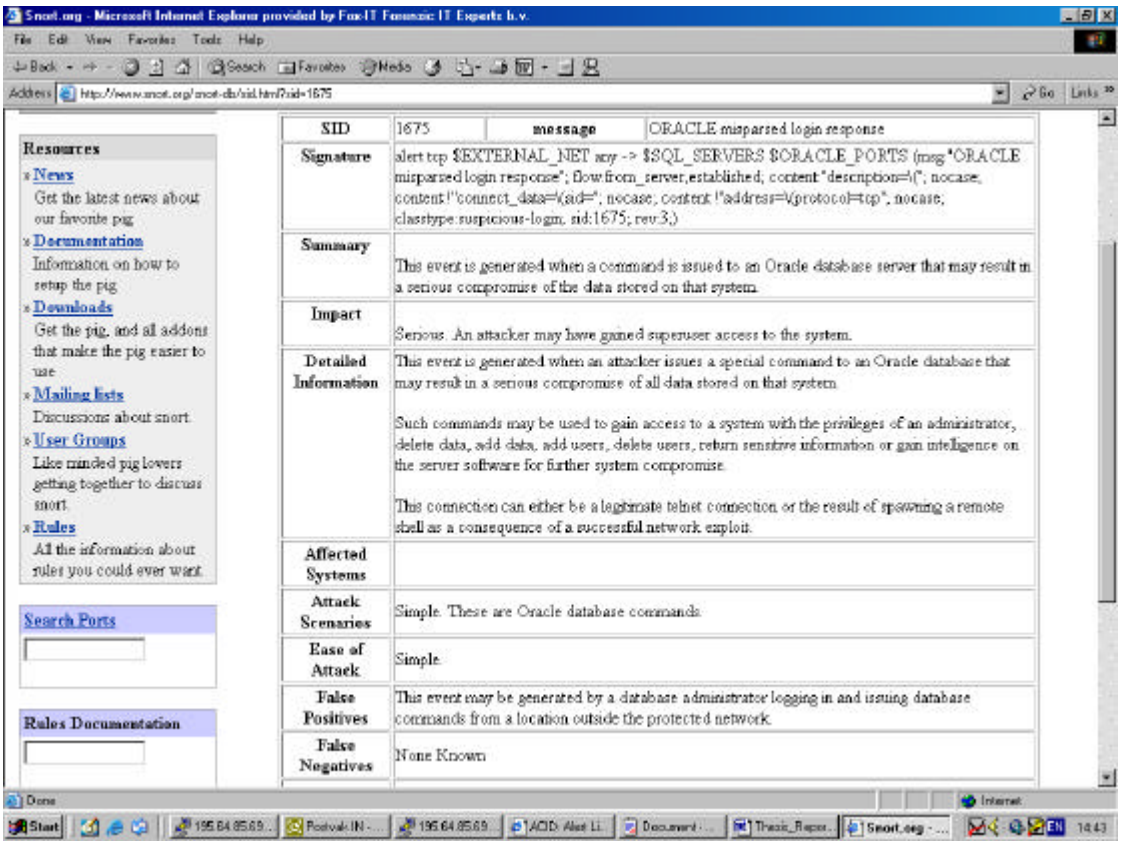


Figure 35 – Snort.org– Oracle Misparsed Login Response Rule Documentation

This figure shows the documentation for the "ORACLE misparsed login response" rule. It is pretty clear upon first inspection by a security analyst what the vulnerable service here is (Oracle). However, it is much more difficult for an automated system to extract this same information.

Here are the only references to that could lead us to Oracle in this rule description:

- **Summary / Detailed Information** - This event is generated when a command is issued to an Oracle database server that may result in a serious compromise of the data stored on that system.
- **Affected Systems –** (left blank)
- **Attack Scenarios** - Simple. These are Oracle database commands.
- **Corrective Action** - Use a firewall to disallow direct access to the Oracle database from sources external to the protected network.
- **References –** (left blank)

Unless we write a system that can extract the names of platforms/services from prose descriptions, we will not be able to use this Snort alert to produce a metaalert. Or, if someone (several people) make a very directed effort towards adding more Reference information to the rule descriptions in the Snort.org rule base, that might also improve the situation. This kind of coordinated effort is arguably simpler than writing a program that can parse prose descriptions, but it is still a considerable amount of work. This would also need to happen for any other signature-based Primary Alert sources (Dragon, ISS, Cisco, etc..) that are used as input to MACE. Naturally the proprietary signature-based IDS systems will correlate more cleanly using their own products (ex. ISS RealSecure IDS/ ISS Internet Scanner). However, we can really only use various commercial primary alerts together in a meaningful way if we can somehow correlate back to Open Standard ID's like those from CVE or Bugtraq. I'm hoping that this situation will improve in the upcoming years as attack / vulnerability correlation becomes a more established technique in the Intrusion Detection arena.

32. Future Research Directions

I believe that progress can be made in Intrusion Detection by advancing research and technologies in the following areas:

- Integration of existing IDS technologies. Many of the purely "academic" IDS methodologies (state transition models, neural networks, statistical analysis, genetic algorithms) have not yet been used in conjunction with the more widely accepted IDS methodologies (signature-based, host-based).

- Continued research in how Expert Systems can be used to simulate human reasoning in the correlation of IDS alerts. (MACE provides a ready-made vehicle for this.)

- Improved collection and usage of "contextual" information, describing the monitored networks and computers. (Ex. platform/services, machine purpose, owner, importance).

- Enable existing (and future) IDS technologies to work with Open Standards, like the Intrusion Detection Message Exchange Format (IDMEF).

- Development of more intuitive data presentation tools that use correlated and integrated data to present a more holistic view of the entire situation.

Part IX – Conclusion

Intrusion Detection has come a long way in the last decade, but it still has much further to go in the next. We have a wide array of Intrusion Detection tools at our disposal, that excel at their own specific tasks: finding signatures in traffic; highlighting anomalous traffic activity, unconventional protocol usage, changes in system logs, etc.. However, most of these tools are written to be solutions in and of themselves. The current host of existing IDS tools were created by people spanning the globe, who may have never had the intention of allowing their tools to work in conjunction with others as part of a larger solution. Interoperability is the next big step, and I believe that a sort of "holistic" analysis, with a greater inclusion of situational and social context, is required for a computer to make sense of the barrage of small technical details.

I believe that the Meta-Alert Correlation Engine has succeeded in providing a solid groundwork for continuing work in this area. While an enormous amount of work remains to be done, I think that MACE's use of the Intrusion Detection Message Exchange Format and its reasoning facility (CLIPS) will provide a useful tool for both the integration of separate IDS techniques, as well as for the analysis of how well these techniques can actually work together.

Construction of a "larger-picture" is utterly necessary. After all, for most people, this all reduces down to a simple question anyways:

Were we hacked or not?

Part X – References

1. Sandeep Kumar, Ph.D. Thesis, Purdue University, August 1995. Classification and Detection of Computer Intrusions.
2. Terry D. Escamilla, Intrusion Detection, Network Security Beyond the Firewall, John Wiley & Sons, 1998.
3. Aurobindo Sundaram, "An Introduction to Intrusion Detection", http://www.acm.org/crossroads/xrds2-4/intrus.html
4. Dorothy Denning, "An Intrusion Detection Model", IEEE Transactions on Software Engineering, Vol. SE-13, No. 2, February 1987, 222-232.
5. Leslie Smith, "An Introduction to Neural Networks", University of Sterling, http://www.cs.stir.ac.uk/~lss/NNIntro/InvSlides.html
6. "Network- vs. Host-based Intrusion Detection", Internet Security Systems, http://documents.iss.net/whitepapers/nvh_ids.pdf
7. "Intrusion Detection Exchange Format (idwg) Charter", http://www.ietf.org/html.charters/idwg-charter.html
8. M. Wood, Internet Security Systems, and M. Erlinger, Harvey Mudd College. "Intrusion Detection Message Exchange Requirements" (Internet Draft). October 22, 2002. http://www.ietf.org/internet-drafts/draft-ietf-idwg-requirements-10.txt
9. Gary Riley, "What is CLIPS?", http://www.ghg.net/clips/WhatIsCLIPS.html
10. Chris Green, "Snort Users Manual", http://www.snort.org/docs/writing_rules
11. Roman Danyliw, "SnortDB Database Schema", http://www.snort.org/docs/snortdb.png
12. John Bull, "Snort's Place on Windows 2000", http://www.snort.org/docs/snort-win2k.htm
13. David Axmark and Michael Widenius, MySQL AB, "MySQL Manual: General Information", http://www.mysql.com/documentation/mysql/bychapter/manual_Introduction.html
14. Stunnel Homepage", http://www.stunnel.org/
15. Roman Danyliw, "Analysis Console for Intrusion Databases (ACID)", http://www.andrew.cmu.edu/~rdanyliw/snort/snortacid.html
16. "OpenBSD FAQ: Introduction to OpenBSD", http://www.openbsd.org/faq/faq1.html
17. "The Standard Template Library: Introduction", http://www.sgi.com/tech/stl/stl_introduction.html
18. Coronado Enterprises, "C++ Tutorial: Introduction", http://www.coronadoenterprises.com/tutorials/cpp/cpp_intro.html
19. Stig Sæther Bakken, Alexander Aulbach, et. al., "PHP Manual", http://www.php.net/manual/en/
20. W3C, "HTML 4.0 Specification", http://www.w3.org/TR/1998/REC-html40-19980424/
21. Gary V Vaughan, Ben Elliston, Tom Tromey and Ian Lance Taylor, "Autoconf, Automake, and Libtool", http://sources.redhat.com/autobook/autobook/autobook_toc.html
22. Ronald Prins and Rens de Wolf, Fox-IT, "Interviewverslag DUNET Topologie"

23. Children's Mercy Hospitals and Clinics, "Stats: Query, Confidence Interval", http://www.cmh.edu/stats/size/confid.asp
24. Dr. Steve Blythe, Statistics and Modeling Science, University of Strathclyde, "Interval Estimates: Confidence Interval", http://www.stams.strath.ac.uk/~steve/53202/confint1.pdf
25. "Overview of the IETF", http://www.ietf.org/overview.html
26. Silicon Defense, "libidmef", http://www.silicondefense.com/idwg/libidmef/index.htm
27. "LBNL's Network Research Group", http://www-nrg.ee.lbl.gov/
28. Joseph Giarratano, Ph.D., "CLIPS User Guide", http://www.ghg.net/clips/download/documentation/usrguide.pdf
29. Hans Boehm and Alan J. Demers, "A garbage collector for C and C++", http://www.hpl.hp.com/personal/Hans_Boehm/gc/leak.html

Part XI - Appendix (Source Code)

Class Definitions

Arpmonitor

arp_stats

```
00068 class arp_stats {
00069   string ip_address_src;
00070   string ip_address_dest;
00071   string mac_address_src;
00072   string timestamp;
00087 };
```

IDMEF++

idmef_action

```
00511 class idmef_action {
00513   string category;
00514   string data;
00526 };
```

idmef_additionaldata

```
00586 class idmef_additionaldata {
00588   string type;
00589   string meaning;
00590   string data;
00604 };
```

idmef_address

```
00017 class idmef_address {
00019   string ident;
00020   string category;
00021   string vlan_name;
00022   string vlan_num;
00023   string address;
00024   string netmask;
00045 };
```

idmef_alert

```
00705 class idmef_alert {
00707   string ident;
00708   list <idmef_analyzer *> Analyzer;
00709   list <idmef_time *> Createtime;
00710   list <idmef_time *> Detecttime;
00711   list <idmef_time *> Analyzertime;
00712   list <idmef_source *> Sources;
00713   list <idmef_target *> Targets;
00714   list <idmef_classification *> Classifications;
00715   list <idmef_assessment *> Assessment;
```

```
00716   list <idmef_correlationalert *> Correlationalert;
00717   list <idmef_toolalert *> Toolalert;
00718   list <idmef_overflowalert *> Overflowalert;
00719   list <idmef_additionaldata *> Additionaldata;
00765 };
```

idmef_alertident

```
00568 class idmef_alertident {
00570   string data;
00571   string analyzerid;
00583 };
```

idmef_analyzer

```
00606 class idmef_analyzer {
00608   string analyzerid;
00609   string manufacturer;
00610   string model;
00611   string version;
00612   string analyzer_class;
00613   string ostype;
00614   string osversion;
00615   list <idmef_node *> Node;
00616   list <idmef_process *> Process;
00644 };
```

idmef_assessment

```
00545 class idmef_assessment {
00547   list <idmef_impact *> Impact;
00548   list <idmef_action *> Actions;
00549   list <idmef_confidence *> Confidence;
00566 };
```

idmef_classification

```
00067 class idmef_classification {
00068   string origin;
00069   string name;
00070   string url;
00085 };
```

idmef_confidence

```
00528 class idmef_confidence {
00530   string rating;
00531   string data;
00543 };
```

idmef_correlationalert

```
00687 class idmef_correlationalert {
00689   string name;
00690   list <idmef_alertident *> Alertidents;
00703 };
```

idmef_file

```
00343 class idmef_file {
00345   string ident;
```

```
00346   string category;
00347   string fstype;
00348   string name;
00349   string path;
00350   string create_time;
00351   string modify_time;
00352   string access_time;
00353   string data_size;
00354   string disk_size;
00355   list <idmef_fileaccess *> Fileaccesses;
00356   list <idmef_linkage *> Linkages;
00357   list <idmef_inode *> Inode;
00394 };
```

idmef_fileaccess

```
00268 class idmef_fileaccess {
00270   list <string> Permissions;
00271   list <idmef_userid *> Userid;
00285 };
```

idmef_filelist

```
00397 class idmef_filelist {
00399   list <idmef_file *> Files;
00410 };
```

idmef_heartbeat

```
00768 class idmef_heartbeat {
00770   string ident;
00771   list <idmef_analyzer *> Analyzer;
00772   list <idmef_time *> Createtime;
00773   list <idmef_time *> Analyzertime;
00774   list <idmef_additionaldata *> Additionaldata;
00796 };
```

idmef_impact

```
00488 class idmef_impact {
00490   string severity;
00491   string completion;
00492   string type;
00493   string data;
00509 };
```

idmef_inode

```
00314 class idmef_inode {
00316   string change_time;
00317   string number;
00318   string major_device;
00319   string minor_device;
00320   string c_major_device;
00321   string c_minor_device;
00341 };
```

idmef_linkage

```
00289 class idmef_linkage {
```

```
00291   string category;
00292   string name;
00293   string path;
00294   list <idmef_file *> File;
00311 };
```

idmef_message

```
00799 class idmef_message {
00801   string version;
00802   list <idmef_alert *> Alerts;
00803   list <idmef_heartbeat *> Heartbeats;
00819 };
```

idmef_node

```
00241 class idmef_node {
00243   string ident;
00244   string category;
00245   string location;
00246   string name;
00247   list <idmef_address *> Addresses;
00266 };
```

idmef_object

```
00014 class idmef_object {
00015 list <idmef_message *> Messages;
00091 };
```

idmef_overflowalert

```
00667 class idmef_overflowalert {
00669   string program;
00670   string size;
00671   string buffer;
00685 };
```

idmef_process

```
00210 class idmef_process {
00212   string ident;
00213   string name;
00214   string pid;
00215   string path;
00216   list <string> Arg;
00217   list <string> Env;
00239 };
```

idmef_service

```
00176 class idmef_service {
00178   string ident;
00179   string name;
00180   string port;
00181   string portlist;
00182   string protocol;
00183   list <idmef_webservice *> Webservice;
00184   list <idmef_snmp_service *> Snmpservice;
00208 };
```

idmef_snmp_service

```
00132 class idmef_snmp_service {
00134   string oid;
00135   string community;
00136   string command;
00150 };
```

idmef_source

```
00412 class idmef_source {
00414   string ident;
00415   string spoofed;
00416   string interface;
00417   list <idmef_node *> Node;
00418   list <idmef_user *> User;
00419   list <idmef_process *> Process;
00420   list <idmef_service *> Service;
00446 };
```

idmef_target

```
00448 class idmef_target {
00450   string ident;
00451   string decoy;
00452   string interface;
00453   list <idmef_node *> Node;
00454   list <idmef_user *> User;
00455   list <idmef_process *> Process;
00456   list <idmef_service *> Service;
00457   list <idmef_filelist *> Filelist;
00486 };
```

idmef_time

```
00048 class idmef_time {
00050   string unix_timestamp;
00064 };
```

idmef_toolalert

```
00646 class idmef_toolalert {
00648   string name;
00649   string command;
00650   list <idmef_alertident *> Alertidents;
00665 };
```

idmef_user

```
00111 class idmef_user {
00113   string ident;
00114   string category;
00115   list <idmef_userid *> Userids;
00130 };
```

idmef_userid

```
00088 class idmef_userid {
00090   string ident;
00091   string type;
```

```
00092   string name;
00093   string number;
00109 };
```

idmef_webservice

```
00152 class idmef_webservice {
00154   string url;
00155   string cgi;
00156   string http_method;
00157   list <string> Arg;
00174 };
```

Libmace

database_query

```
00022 class database_query {
00024   string hostname;
00025   string db_name;
00026   string user_id;
00027   string password;
00028   string query_string;
00029   vector <string> query_results;
00044 };
```

protocol

```
00012 class protocol {
00013   list<string> protocol_name;
00014   list<int> protocol_num;
00026 };
```

tcp_socket

```
00028 class tcp_socket {
00029   int sockfd;
00030   int listenfd;
00031   int connfd;
00032   pid_t childpid;
00033   struct sockaddr_in servaddr;
00034   struct sockaddr_in cliaddr;
00035   string sendline;
00036   int serv_port;
00037   string serv_ip_address;
00038   void sig_chld(int);
00066 };
```

MACE Preprocessing

plugin_loader

```
00031 class plugin_loader {
00032   lt_dlhandle handle;
00033   char *filename;
00034   char *module_path;
00036   list<string> plugin_SID;
00046 };
```

PAM Snort

sensor_list

```
00025 class sensor_list {
00026   list <string> sensor_id;
00027   list <string> sensor_last_cid;
00028   list <string> sensor_max_cid;
00045 };
```

Non-Member Function Descriptions

Arpmonitor

arpmonitor.h

```
00091 int main (int, char **);
```
Main function.

```
00092 void initiate_timed_sniff(char*, const char*, int, int);
```
Initiate network sniffing for a specified time interval.

```
00093 pcap_t* open_pcap_socket(char* device, const char* bpfstr);
```
Open the libpcap socket.

```
00094 void capture_loop(pcap_t* pd, int packets, pcap_handler func);
```
Start the packet capture loop.

```
00095 void parse_packet(u_char *user, struct pcap_pkthdr *packethdr,
00096                    u_char *packetptr);
```
Parse the captured packet.

```
00097 int extract_arp_info(struct pcap_pkthdr *packethdr, u_char
*packetptr, arp_stats *Arp_Stats);
```
Extract the ARP information from the packet.

```
00098 void send_arp_packet_to_db(arp_stats *Arp_Stats);
```
Send the arp information to the database.

```
00099 void remove_arp_info_from_db(arp_stats *Arp_Stats);
```
Remove the old arp information from the database.

```
00100 int query_matching_arp_information(arp_stats *Arp_Stats,
arp_stats *Arp_Stats2);
```
Query matching arp entries from the database.

```
00101 int compare_two_arp_entries(arp_stats *Arp_Stats, arp_stats
*Arp_Stats2);
```
Compare two ARP entries in the database.

```
00102 void generate_new_station_alert(arp_stats *Arp_Stats);
```
Generate an Arpmonitor “new station” IDMEF alert.

```
00103 void generate_changed_mac_address_alert(arp_stats *Arp_Stats,
arp_stats *Arp_Stats2);
```
Generate an Arpmonitor “changed MAC address” IDMEF alert.

```
00104 void bailout(int signo);
```
Stop the network sniffing, since a signal was caught.

```
00105 unsigned int alarm (unsigned int seconds);
```
Generate a SIGALM after the specified number of seconds.

IDMEF++

idmef_xml.h

```
00029   void set_my_xml_globals();
```
Set global variables that libidmef requires.

```
00038   void generate_my_xml_classification(idmef_classification
*Idmef_Classification);
```
Generate the IDMEF classification XML tag.

```
00047   void generate_my_xml_address(idmef_address *Idmef_Address);
```
Generate the IDMEF address XML tag.

```
00056   void generate_my_xml_node(idmef_node *Idmef_Node);
```
Generate the IDMEF node XML tag.

```
00065   void generate_my_xml_additionaldata(idmef_additionaldata
*Idmef_Additionaldata);
```
Generate the IDMEF additionaldata XML tag.

```
00074   void generate_my_xml_process(idmef_process *Idmef_Process);
```
Generate the IDMEF process XML tag.

```
00083   void generate_my_xml_analyzer(idmef_analyzer *Idmef_Analyzer);
```
Generate the IDMEF analyzer XML tag.

```
00092   void generate_my_xml_target(idmef_target *Idmef_Target);
```
Generate the IDMEF target XML tag.

```
00101   void generate_my_xml_source(idmef_source *Idmef_Source);
```
Generate the IDMEF source XML tag.

```
00110   void generate_my_xml_createtime(idmef_time *Idmef_Createtime);
```
Generate the IDMEF createtime XML tag.

```
00119   void generate_my_xml_analyzertime(idmef_time
*Idmef_Analyzertime);
```
Generate the IDMEF analyzertime XML tag.

```
00128   void generate_my_xml_detecttime(idmef_time *Idmef_Detecttime);
```
Generate the IDMEF detecttime XML tag.

```
00137   void generate_my_xml_impact(idmef_impact *Idmef_Impact);
```

Generate the IDMEF impact XML tag.

```
00146   void generate_my_xml_assessment(idmef_assessment
*Idmef_Assessment);
```

Generate the IDMEF assessment XML tag.

```
00155   void generate_my_xml_action(idmef_action *Idmef_Action);
```

Generate the IDMEF action XML tag.

```
00164   void generate_my_xml_confidence(idmef_confidence
*Idmef_Confidence);
```

Generate the IDMEF confidence XML tag.

```
00174   void generate_my_xml_correlationalert(idmef_correlationalert
*Idmef_Correlationalert);
```

Generate the IDMEF correlationalert XML tag.

```
00183   void generate_my_xml_toolalert(idmef_toolalert
*Idmef_Toolalert);
```

Generate the IDMEF toolalert XML tag.

```
00192   void generate_my_xml_overflowalert(idmef_overflowalert
*Idmef_Overflowalert);
```

Generate the IDMEF overflowalert XML tag.

```
00201   void generate_my_xml_alertident(idmef_alertident
*Idmef_Alertident);
```

Generate the IDMEF alertident XML tag.

```
00210   void generate_my_xml_alert(idmef_alert *Idmef_Alert);
```

Generate the IDMEF alert XML tag.

```
00219   void generate_my_xml_heartbeat(idmef_heartbeat
*Idmef_Heartbeat);
```

Generate the IDMEF heartbeat XML tag.

```
00228   void generate_my_xml_service(idmef_service *Idmef_Service);
```

Generate the IDMEF service XML tag.

```
00237   void generate_my_xml_webservice(idmef_webservice
*Idmef_Webservice);
```

Generate the IDMEF webservice XML tag.

```
00246   void generate_my_xml_snmp_service(idmef_snmp_service
*Idmef_Snmp_Service);
```

Generate the IDMEF snmp service XML tag.

```
00255   void generate_my_xml_filelist(idmef_filelist *Idmef_Filelist);
```

Generate the IDMEF filelist XML tag.

```
00264   void generate_my_xml_inode(idmef_inode *Idmef_Inode);
```

Generate the IDMEF inode XML tag.

```
00273   void generate_my_xml_file(idmef_file *Idmef_File);
```
Generate the IDMEF file XML tag.

```
00282   void generate_my_xml_fileaccess(idmef_fileaccess
*Idmef_Fileaccess);
```
Generate the IDMEF fileaccess XML tag.

```
00291   void generate_my_xml_linkage(idmef_linkage *Idmef_Linkage);
```
Generate the IDMEF linkage XML tag.

```
00300   void generate_my_xml_user(idmef_user *Idmef_User);
```
Generate the IDMEF user XML tag.

```
00309   void generate_my_xml_userid(idmef_userid *Idmef_Userid);
```
Generate the IDMEF userid XML tag.

```
00318   void generate_my_xml_message(idmef_message *Idmef_Message);
```
Generate the IDMEF message XML tag.

```
00327   void return_my_xml_string(idmef_object *Idmef_Object);
```
Generate the IDMEF XML string, given a passed IDMEF Object instance.

```
00336   void print_my_xml_object(idmef_object *Idmef_Object);
```
Print the XML string for the passed IDMEF object to the screen.

```
00349 void parse_my_xml_string(char *xml_string, idmef_object
*Idmef_Object);
```
Parse the provided XML string, filling in the appropriate information in the IDMEF Object instance.

```
00364 void print_xml_object(idmef_object *Idmef_Object);
```
C++ function to call the C-compatible print_my_xml_object function.

```
00379 char *return_xml_string(idmef_object *Idmef_Object);
```
C++ function to call the C-compatible return_my_xml_string function.

```
00393 void parse_xml_string(char *xml_string, idmef_object
*Idmef_Object);
```
C++ function to call the C-compatible parse_my_xml_string function.

```
00395 void set_xml_globals();
```
Set the global variables that libidmef needs to generate IDMEF XML.

Libmace

aux_stuff.h

```
00016 int query_hex_payload_from_db(char *, char *);
```
Query a hex data payload from the database.

```
00017 int convert_from_hex(char *, char *);
```
Convert the passed strings from hex into ascii.

```
00018 int get_ascii_packet_payload(char *, char *);
```
Query data payload from database, and return as ascii.

error.h

```
00011 void err_ret(const char *fmt, ...);
```
Non-fatal error related to a system call. Print a message and return.

```
00012 void err_sys(const char *fmt, ...);
```
Fatal error related to a system call. Print a message and terminate.

```
00013 void err_dump(const char *fmt, ...);
```
Fatal error related to a system call. Print a message, dump core, and terminate.

```
00014 void err_msg(const char *fmt, ...);
```
Non-fatal error unrelated to a system call. Print a message and return.

```
00015 void err_quit(const char *fmt, ...);
```
Fatal error unrelated to a system call. Print a message and terminate.

```
00016 static void err_doit(int errnoflag, int level, const char *fmt,
va_list ap);
```
Print a message and return to caller. Caller specifies “errnoflag” and “level”.

(Note: These error functions are being used to handle the socket-related errors. These functions were written by Richard Stevens, for demonstration of error handling throughout his many excellent books. The source code can be downloaded from his website - http://www.kohala.com/start/unpv12e/unpv12e.tar.gz)

misc.h

```
char *itoa(int);
```
Convert an integer into a character string.

socket.h

```
00007 int Socket(int family, int type, int protocol);
```
Create a communications socket – wrapper function w/ error handling.

```
00008 void Connect(int fd, const struct sockaddr *sa, socklen_t salen);
```
Connect the socket to a specific address / port – wrapper function w/ error handling.

```
00009 void Inet_pton(int family, const char *strptr, void *addrptr);
```
Convert address from presentation to network format – wrapper function w/ error handling.

```
00007 ssize_t    readline(int, void *, size_t);
```
Read a line of data from the socket.

```
00008 ssize_t    readn(int, void *, size_t);
```
Read specified number of characters from the socket.

```
00009 ssize_t    writen(int, const void *, size_t);
```
Write a specified number of characters to the socket.

```
00010 ssize_t    Read(int, void *, size_t);
```
Read specified number of characters from the socket – wrapper function w/ error handling.

```
00011 ssize_t    Readline(int, void *, size_t);
```
Read a line of data from the socket – wrapper function w/ error handling.

```
00012 void       Writen(int, void *, size_t);
```
Write a specified number of characters to the socket – wrapper function w/ error handling.

(Note: These socket wrapper functions were written by Richard Stevens, for demonstration purposes throughout his many excellent books. The source code can be downloaded from his website - http://www.kohala.com/start/unpv12e/unpv12e.tar.gz)

MACE Expert System

clips_heartbeat_processing.h

```
00016 void construct_heartbeat_input(string *, idmef_object *);
```
Construct an IDMEF heartbeat, given a passed idmef_object.

idmefdb_api.h

```
00016 void send_idmef_object_to_db(idmef_object *, database_query *);
```
Send the entire IDMEF object to the metaalert database.

```
00017 void send_idmef_address_to_db(idmef_address *, string *, string
*, database_query *);
```
Add the IDMEF address to the correct entry in the metaalert database.

```
00018 void send_idmef_createtime_to_db(idmef_time *, string *, string
*, database_query *);
```
Add the IDMEF createtime to the correct entry in the metaalert database.

```
00019 void send_idmef_detecttime_to_db(idmef_time *, string *, string
*, database_query *);
```
Add the IDMEF detecttime to the correct entry in the metaalert database.

```
00020 void send_idmef_analyzertime_to_db(idmef_time *, string *, string
*, database_query *);
```
Add the IDMEF analyzertime to the correct entry in the metaalert database.

```
00021 void send_idmef_classification_to_db(idmef_classification *,
string *, string *, database_query *);
```
Add the IDMEF classification to the correct entry in the metaalert database.

```
00022 void send_idmef_userid_to_db(idmef_userid *, string *, string *,
database_query *);
```
Add the IDMEF userid to the correct entry in the metaalert database.

```
00023 void send_idmef_user_to_db(idmef_user *, string *, string *,
database_query *);
```
Add the IDMEF user to the correct entry in the metaalert database.

```
00024 void send_idmef_snmp_service_to_db(idmef_snmp_service *, string
*, string *, database_query *);
```
Add the IDMEF SNMP service to the correct entry in the metaalert database.

```
00025 void send_idmef_webservice_to_db(idmef_webservice *, string *,
string *, database_query *);
```
Add the IDMEF webservice to the correct entry in the metaalert database.

```
00026 void send_idmef_service_to_db(idmef_service *, string *, string
*, database_query *);
```
Add the IDMEF service to the correct entry in the metaalert database.

```
00027 void send_idmef_process_to_db(idmef_process *, string *, string
*, database_query *);
```
Add the IDMEF process to the correct entry in the metaalert database.

```
00028 void send_idmef_node_to_db(idmef_node *, string *, string *,
database_query *);
```
Add the IDMEF node to the correct entry in the metaalert database.

```
00029 void send_idmef_fileaccess_to_db(idmef_fileaccess *, string *,
string *, database_query *);
```
Add the IDMEF fileaccess to the correct entry in the metaalert database.

```
00030 void send_idmef_linkage_to_db(idmef_linkage *, string *, string
*, database_query *);
```
Add the IDMEF linkage to the correct entry in the metaalert database.

```
00031 void send_idmef_inode_to_db(idmef_inode *, string *, string *,
database_query *);
```
Add the IDMEF inode to the correct entry in the metaalert database.

```
00032 void send_idmef_file_to_db(idmef_file *, string *, string *,
database_query *);
```
Add the IDMEF file to the correct entry in the metaalert database.

```
00033 void send_idmef_filelist_to_db(idmef_filelist *, string *, string
*, database_query *);
```
Add the IDMEF filelist to the correct entry in the metaalert database.

```
00034 void send_idmef_source_to_db(idmef_source *, string *, string *,
database_query *);
```
Add the IDMEF source to the correct entry in the metaalert database.

```
00035 void send_idmef_target_to_db(idmef_target *, string *, string *,
database_query *);
```
Add the IDMEF target to the correct entry in the metaalert database.

```
00036 void send_idmef_impact_to_db(idmef_impact *, string *, string *,
database_query *);
```
Add the IDMEF impact to the correct entry in the metaalert database.

```
00037 void send_idmef_action_to_db(idmef_action *, string *, string *,
database_query *);
```
Add the IDMEF action to the correct entry in the metaalert database.

```
00038 void send_idmef_confidence_to_db(idmef_confidence *, string *,
string *, database_query *);
```
Add the IDMEF confidence to the correct entry in the metaalert database.

```
00039 void send_idmef_assessment_to_db(idmef_assessment *, string *,
string *, database_query *);
```
Add the IDMEF assessment to the correct entry in the metaalert database..

```
00040 void send_idmef_alertident_to_db(idmef_alertident *, string *,
string *, database_query *);
```
Add the IDMEF alertident to the correct entry in the metaalert database.

```
00041 void send_idmef_additionaldata_to_db(idmef_additionaldata *,
string *, string *, database_query *);
```
Add the IDMEF additionaldata to the correct entry in the metaalert database.

```
00042 void send_idmef_analyzer_to_db(idmef_analyzer *, string *, string
*, database_query *);
```
Add the IDMEF analyzer to the correct entry in the metaalert database.

```
00043 void send_idmef_toolalert_to_db(idmef_toolalert *, string *,
string *, database_query *);
```
Add the IDMEF toolalert to the correct entry in the metaalert database.

```
00044 void send_idmef_overflowalert_to_db(idmef_overflowalert *, string
*, string *, database_query *);
```

Add the IDMEF overflowalert to the correct entry in the metaalert database.

```
00045 void send_idmef_correlationalert_to_db(idmef_correlationalert *,
string *, string *, database_query *);
```
Add the IDMEF correlationalert to the correct entry in the metaalert database.

```
00046 void send_idmef_alert_to_db(idmef_alert *, string *, string *,
database_query *);
```
Add the IDMEF alert to the correct entry in the metaalert database.

```
00047 void send_idmef_heartbeat_to_db(idmef_heartbeat *, string *,
string *, database_query *);
```
Add the IDMEF heartbeat to the correct entry in the metaalert database.

```
00048 void send_idmef_message_to_db(idmef_message *, database_query *);
```
Add the IDMEF message to the correct entry in the metaalert database.

```
00049 void query_max_entity_index(string *, string *, database_query
*);
```
Query the maximum entity index of a specific kind of IDMEF element.

```
00050 void send_arg_to_db(char *, string *, string *, database_query
*);.
```
Add the arg to the correct entry in the metaalert database.

```
00051 void send_env_to_db(char *, string *, string *, database_query
*);
```
Add the env to the correct entry in the metaalert database.

```
00052 void send_permission_to_db(char *, string *, string *,
database_query *);
```
Add the IDMEF permission to the correct entry in the metaalert database.

generate_clips_object.h

```
00011 void generate_clips_address(idmef_address *, string *, string *,
int *);
```
Generate the CLIPS representation of a passed IDMEF address.

```
00012 void generate_clips_createtime(idmef_time *, string *, string *,
int *);
```
Generate the CLIPS representation of a passed IDMEF createtime.

```
00013 void generate_clips_analyzertime(idmef_time *, string *, string
*, int *);
```
Generate the CLIPS representation of a passed IDMEF analyzertime.

```
00014 void generate_clips_detecttime(idmef_time *, string *, string *,
int *);
```
Generate the CLIPS representation of a passed IDMEF detecttime.

```
00015 void generate_clips_classification(idmef_classification *, string
*, string *, int *);
```
Generate the CLIPS representation of a passed IDMEF classification.

```
00016 void generate_clips_userid(idmef_userid *, string *, string *,
int *);
```
Generate the CLIPS representation of a passed IDMEF userid.

```
00017 void generate_clips_user(idmef_user *, string *, string *, int
*);
```
Generate the CLIPS representation of a passed IDMEF user.

```
00018 void generate_clips_snmp_service(idmef_snmp_service *, string *,
string *, int *);
```
Generate the CLIPS representation of a passed IDMEF SNMP service.

```
00019 void generate_clips_webservice(idmef_webservice *, string *,
string *, int *);
```
Generate the CLIPS representation of a passed IDMEF webservice.

```
00020 void generate_clips_service(idmef_service *, string *, string *,
int *);
```
Generate the CLIPS representation of a passed IDMEF service.

```
00021 void generate_clips_process(idmef_process *, string *, string *,
int *);
```
Generate the CLIPS representation of a passed IDMEF process.

```
00022 void generate_clips_node(idmef_node *, string *, string *, int
*);
```
Generate the CLIPS representation of a passed IDMEF node.

```
00023 void generate_clips_fileaccess(idmef_fileaccess *, string *,
string *, int *);
```
Generate the CLIPS representation of a passed IDMEF fileaccess.

```
00024 void generate_clips_linkage(idmef_linkage *, string *, string *,
int *);
```
Generate the CLIPS representation of a passed IDMEF linkage.

```
00025 void generate_clips_inode(idmef_inode *, string *, string *, int
*);
```
Generate the CLIPS representation of a passed IDMEF inode.

```
00026 void generate_clips_file(idmef_file *, string *, string *, int
*);
```
Generate the CLIPS representation of a passed IDMEF file..

```
00027 void generate_clips_filelist(idmef_filelist *, string *, string
*, int *);
```
Generate the CLIPS representation of a passed IDMEF filelist.

```
00028 void generate_clips_source(idmef_source *, string *, string *,
int *);
```
Generate the CLIPS representation of a passed IDMEF source.

```
00029 void generate_clips_target(idmef_target *, string *, string *,
int *);
```
Generate the CLIPS representation of a passed IDMEF target.

```
00030 void generate_clips_impact(idmef_impact *, string *, string *,
int *);
```
Generate the CLIPS representation of a passed IDMEF impact.

```
00031 void generate_clips_action(idmef_action *, string *, string *,
int *);
```
Generate the CLIPS representation of a passed IDMEF action.

```
00032 void generate_clips_confidence(idmef_confidence *, string *,
string *, int *);
```
Generate the CLIPS representation of a passed IDMEF confidence.

```
00033 void generate_clips_assessment(idmef_assessment *, string *,
string *, int *);
```
Generate the CLIPS representation of a passed IDMEF assessment.

```
00034 void generate_clips_alertident(idmef_alertident *, string *,
string *, int *);
```
Generate the CLIPS representation of a passed IDMEF alertident.

```
00035 void generate_clips_additionaldata(idmef_additionaldata *, string
*, string *, int *);
```
Generate the CLIPS representation of a passed IDMEF additionaldata.

```
00036 void generate_clips_analyzer(idmef_analyzer *, string *, string
*, int *);
```
Generate the CLIPS representation of a passed IDMEF analyzer.

```
00037 void generate_clips_toolalert(idmef_toolalert *, string *, string
*, int *);
```
Generate the CLIPS representation of a passed IDMEF toolalert.

```
00038 void generate_clips_overflowalert(idmef_overflowalert *, string
*, string *, int *);
```
Generate the CLIPS representation of a passed IDMEF overflowalert.

```
00039 void generate_clips_correlationalert(idmef_correlationalert *,
string *, string *, int *);
```
Generate the CLIPS representation of a passed IDMEF correlationalert.

```
00040 void generate_clips_alert(idmef_alert *, string *, string *, int
*);
```

Generate the CLIPS representation of a passed IDMEF alert.

```
00041 void generate_clips_heartbeat(idmef_heartbeat *, string *, string
*, int *);
```
Generate the CLIPS representation of a passed IDMEF heartbeat.

```
00042 void generate_clips_message(idmef_message *, string *, string *,
int *);
```
Generate the CLIPS representation of a passed IDMEF message.

```
00043 void generate_clips_object(string *, idmef_object *);
```
Generate the CLIPS representation of a passed IDMEF object.

putalert.h

```
00020 void process_metaalert(string, database_query *, database_query
*, protocol *);
```
Process each metaalert that appears as output from CLIPS.

```
00021 void send_metaalert_to_db(idmef_object *, database_query *);
```
Send the metaalert to the IDMEF metaalert database.

snort_viewer.h

```
00020 void add_snort_schema_info(database_query *);
```
Add the Snort schema version number to the viewer database.

```
00021 int check_existance_snort_schema_info(database_query *);
```
Check to see if the Snort schema version number already exists in the viewer database.

```
00022 void send_metaalert_to_viewer_db(idmef_object *, database_query
*);
```
Send the metaalert to the Snort-format viewer database.

```
00023 void populate_event_table_from_metaalert(idmef_object *,
database_query *);
```
Populate the Snort event table with data taken from the IDMEF metaalerts.

```
00024 void populate_iphdr_table_from_metaalert(idmef_object *,
database_query *);
```
Populate the Snort iphdr table with data taken from the IDMEF metaalerts.

```
00025 void populate_signature_table_from_metaalert(idmef_object *,
database_query *);
```
Populate the Snort signature table with data taken from the IDMEF metaalerts.

```
00026 int get_internal_sigid(string *, string *, database_query *);
```
Get the internal sig_id representation of the desired signature, given the sig_name.

```
00027 void populate_tcphdr_table_from_metaalert(idmef_object *,
database_query *);
```
Populate the Snort tcphdr table with data taken from the IDMEF metaalerts.

```
00028 void populate_udphdr_table_from_metaalert(idmef_object *,
database_query *);
```
Populate the Snort udphdr table with data taken from the IDMEF metaalerts.

```
00029 void populate_protocol_tables_from_metaalert(idmef_object *,
database_query *);
```
Populate the Snort protocol table with data taken from the IDMEF metaalerts.

```
00030 void populate_sensor_table_from_metaalert(idmef_object *,
database_query *);
```
Populate the Snort sensor table with data taken from the IDMEF metaalerts.

```
00031 int get_internal_sensorid(string *, string *, string *,
database_query *);
```
Get the internal sensor_id representation of the desired sensor, given the sig_name.

```
00032 void populate_sig_class_table_from_metaalert(idmef_object *,
database_query *);
```
Populate the Snort sig_class table with data taken from the IDMEF metaalerts.

```
00033 int get_internal_sigclassid(string *, string *, database_query
*);
```
Get the internal sig_class id representation of the desired signature class, given the sig_class name.

```
00034 void populate_reference_system_table_from_metaalert(idmef_object
*, database_query *);
```
Populate the Snort reference_system table with data taken from the IDMEF metaalerts.

```
00035 void populate_reference_table_from_metaalert(idmef_object *,
database_query *);
```
Populate the Snort reference table with data taken from the IDMEF metaalerts.

```
00036 int get_internal_refsysid(string *, string *, database_query *);
```
Get the internal refsys_id representation of the desired reference system, given the refsys_ name.

```
00037 int get_internal_referenceid(string *, string *, string *,
database_query *);
```
Get the internal ref_id representation of the desired reference, given the ref_ name.

```
00038 void populate_sig_reference_table_from_metaalert(idmef_object *,
database_query *);
```
Populate the Snort sig_reference table with data taken from the IDMEF metaalerts.

```
00039 int check_existance_sig_reference(string *, string *, string *,
database_query *);
```

Check to see if the Snort signature reference entry already exists in the viewer database.

```
00040 void populate_data_table_from_metaalert(idmef_object *,
database_query *);
```
Populate the Snort data table with data taken from the IDMEF metaalerts.

MACE Preprocessing

clips_client.h

```
00017 int connect_to_clips_server(tcp_socket *);
```
Initiate a connection with the CLIPS server.

```
00018 int send_alert_to_clips(idmef_object *, tcp_socket *);
```
Send an IDMEF object to the CLIPS server.

```
00019 int send_run_to_clips(tcp_socket *);
```
Send a 'run' command heartbeat to the CLIPS engine.

```
00020 int disconnect_from_clips_server(tcp_socket *);
```
Break the connection with the CLIPS server.

```
00021 void construct_command_heartbeat(string *, string *);
```
Construct an IDMEF heartbeat the contains a command for the CLIPS server.

```
00022 void send_line_to_clips(string *, tcp_socket *);
```
Send a line across the socket to the CLIPS engine.

PAM Snort

getalert.h

```
00047 void initialize_xml_globals();
```
Initialize the global variables required to use libidmef.

```
00048 int get_current_sensor_list(sensor_list *);
```
Get a list of the sensor id's contained in the primary Snort database.

```
00049 int get_sensor_max_cids(sensor_list *);
```
Get the maximum CID value for each of the sensors.

```
00050 int allocate_alert_structure(idmef_object *);
```
Allocate memory necessary to store alert data in an IDMEF Object.

```
00051 int query_next_alert(char *, char *, idmef_object *, protocol *);
```
Query the next alert from the primary Snort database.

```
00052 int query_next_cid(char *, char *, string *);
```
Query the next alert CID from the primary Snort database.

```
00053 int query_alert_from_db(char *, char *, idmef_object *, protocol
*);
```
Query the specified alert from the primary Snort database.

```
00054 int query_attack_type_from_db(char *, char *, string *);
```
Query the specified attack type from the primary Snort database.

```
00055 int query_attack_name_from_db(char *, char *, string *);
```
Query the specified attack name from the primary Snort database.

```
00056 int query_ip_protocol_from_db(char *, char *, string *, protocol
*);
```
Query the specified IP protocol from the primary Snort database.

```
00057 int query_timestamp_from_db(char *, char *, string *);
```
Query the specified timestamp from the primary Snort database.

```
00058 int query_src_ip_address_from_db(char *, char *, string *);
```
Query the specified source IP address from the primary Snort database.

```
00059 int query_dest_ip_address_from_db(char *, char *, string *);
```
Query the specified destination IP address from the primary Snort database.

```
00060 int query_src_port_from_db(char *, char *, string *);
```
Query the specified source port from the primary Snort database.

```
00061 int query_dest_port_from_db(char *, char *, string *);
```
Query the specified destination port from the primary Snort database.

```
00062 int query_sensor_name_from_db(char *, string *);
```
Query the specified sensor name from the primary Snort database.

```
00063 int query_attack_url_from_db(char *, char *, string *);
```
Query the specified attack URL from the primary Snort database.

```
00064 int query_packet_payload_from_db(char *, char *, string *);
```
Query the specified packet payload from the primary Snort database.

```
00065 void filter_special_characters(string *);
```
Filter special characters out of the (ASCII) packet payload before it is sent to the database.

```
00066 int query_interface_from_db(char *, string *);
```
Query the specified interface from the primary Snort database.

mace_client.h

```
00016 int connect_to_mace_server(tcp_socket *);
```
Initiate a connection with the MACE server.

```
00017 int send_alert_to_mace_server(string, tcp_socket *);
```
Send an IDMEF alert to the MACE server.

```
00018 int disconnect_from_mace_server(tcp_socket *);
```
Break the connection with the MACE server.