SnortMart – a Network Intrusion Detection System DataMart

John Doe
JDoe@csudh.edu
Computer Science Department
California State University, Dominguez Hills

ABSTRACT

Network intrusion detection is commonly thought of as the process of determining when unauthorized people are making an attempt to break into your network. However, this is not a complete picture of network intrusion detection. Though unauthorized login attempts is an easy to understand example of an intrusion, there are other types of activity that are not as clear cut, such as probing your network with port scans or pings. Though not a direct attempt to break into your network, these types of activities are a typical precursor to more hostile activity, and thus are considered an intrusion and should be identified as such. Network Intrusion Detection Systems (NIDS) capture large amounts of data that is difficult or impractical to report and analyze directly from the capture device. It is also common to have more than one NIDS device and reporting from a consolidated (multi-NIDS device) perspective can also be difficult or not practical, depending on the number of NIDS devices. To provide a platform for multi-NIDS device reporting and analysis, this paper describes a consolidated database, or DataMart design and implementation to store data from multiple Snort NIDS devices. This consolidated DataMart, called "SnortMart" is optimized for reporting and analysis and can provide a platform for better understanding of NIDS device information.

KEYWORDS

Network, Intrusion Detection, DataMart, SnortMart

1. INTRODUCTION

Network intrusion detection is commonly thought of as the process of determining when unauthorized people are making an attempt to break into your network. However, this is not a complete picture of network intrusion detection. Though unauthorized login attempts is an easy to understand example of an intrusion, there are other types of activity that are not as clear cut, such as probing your network with port scans or pings. Though not a direct attempt to break into your network, these types of activities are a typical precursor to more hostile activity, and thus are considered an intrusion and should be identified as such.

Many organizations will have more than a single system and/or networks. Intrusion detection monitoring of these multiple systems and networks requires the existence of multiple intrusion detection systems. This is because each intrusion detection system must be connected to the networks being monitored. Also, bandwidth limitations more than one intrusion detection system may be required on the same network.

One of the most troublesome issues that an intrusion detection system administrator faces is what is called the "false positive." This is basically an event identified as an intrusion attempt, but in reality is not one. The typical response to this by the administrator is to reconfigure the intrusion detection system to not identify that particular event as an intrusion attempt. However, this means that a real intrusion attempt may be missed if the intrusion detection system is configured not to identify the event as such. On the other hand, being constantly notified by "false positives" may also result in a false sense of security, as the administrator can adopt an attitude that typically intrusion events reported by the intrusion detection system are false positives and may not properly respond to a real intrusion attempt.

So to be able to have the opportunity to be able to respond to more intrusion events, the intrusion detection system must be configured in a way that will probably report many of these "false positives" and thus, will process and store more information that is typically expected. Along with this is a greater effort required to review this information – though the intrusion detection system can identify store and event notify an administrator of an intrusion event, a person must still access and review the data to analyze the event, and typically will need to be able to review this event with others as well. This is because an intrusion attempt may be targeted at multiple systems and or networks in parallel or in series within some type of measurable timeframe.

Additional tools are required to better understand all of the data generated by the intrusion detection systems. This need is not new, there are many existing solutions that will read from and produce reports from intrusion detection system logs.

However, most of these tools are tools are designed to directly read from the intrusion detection system log and are not optimized for reporting. Because of this, creating custom reports for analysis is more difficult and will often take a lot of time to produce information for analysis. This can result in these types of tools not being used, as they are too difficult and frustrating to use on a regular basis.

2. BACKGROUND

As indicated by Cox and Gerg [2], Snort is an open source network packet monitoring and Intrusion Detection System. Snort looks for attack signatures, which are specific patterns of activity that has been defined to be of a suspicious or malicious intent. Snort analyzes network packets, and thus is classified as a Network Intrusion Detection System, or NIDS. These types of systems must be connected to the networks that they monitor and unless the network topology is very simple, multiple Snort systems, called Snort sensors, must be setup and configured to monitor these networks. Snort relies on the ability to recognize attack signatures in order to identify an attack. These pattern recognition definitions are called rules. Attacks are not static, as they are continuously evolving as systems are protected to withstand existing attack methodologies. Thus, it is critical to perform analysis of prior activity to look for trends or changes in activity that are not typically classified as an attack which are often the precursor to an attack.

Though it is possible to analyze information on multiple Snort sensors one at a time, it is difficult to summarize or perform analysis from a multi-Snort sensor perspective. For example, if an organization has multiple office locations in widely different geographical locations, it would be expected that separate Snort sensors are configured and operating. If an attacker targets the organization, it is possible that these different geographical locations are probed and attacked in series or simultaneously. Being able to recognize probing or attacking at a multi-geographic perspective can provide value in understanding individual sensor alerts.

Snort evaluates data at the packet level and thus must process a large amount of data in real-time. Because of this, logging performance is important and to achieve this, the data storage implementation for Snort is optimized for fast writing. This results in a highly normalized database design where there are many one to one entity relationships. In fact, information representing the primary type of information in Snort, called an event (which is the packet information that matched one or more Snort rules), is represented in no less than six tables. One of the tables contains information that must always be provided for every event, and the other five tables contain information that is optional depending on the type of event that has occurred. This implementation allows for writing the smallest amount of information at a time, which allows for high performance when logging information.

However, this design's drawback is when there is a need to read the information for reporting and other analysis. Displaying information for a single event may require joining six or more tables using outer joins, which impacts reporting performance. Even with this highly normalized database design, the log data cannot be kept indefinitely, requiring that the data is removed from the sensor system by deleting older data. This results in the loss of data that could be used to develop better rules or provide evidence of an attack. Though it can be archived before deleting, the data is then offline and harder to analyze.

Existing multi-Snort log reporting applications do exist. ACID, a popular web-database application has been available since. However, ACID is designed so that it can be configured as the primary store for Snort log data and thus is subject to the same performance and historical data issues that the Snort sensors face – in the section titled "The Ongoing Use of the ACID Console," Cox and Gerg [2] discusses deleting Snort log data on a periodic basis, though recommending backing up the data to some type of offline storage before deleting the data.

3. SNORTMART DESIGN

While Snort is focused on high performance logging, SnortMart is focused on easy navigation and high performance reporting. To accomplish this goal, the database design for SnortMart will emphasize minimizing

the number of entities and joins. One entity will contain the core information from the Snort sensors, while all of the other entities will contain classification information that will help guide queries to the core information.

This type of database is called a dimensional data warehouse and this type of design is called the star schema. The design is called the star schema because the center of the star contains the core entity, or fact entity for the database, while connected to this fact entity are various entities that contain the classifications, or dimensions that allow us to select subsets of information from the fact entity (when represented on a diagram, the fact entity with the dimension entities connected to it represents a star shape).

Defining the fact entity is of key importance, as each fact defines the lowest level of detail available to users of SnortMart. Determining this level of detail is called defining the granularity, or what Kimball [3] calls "the grain." The Snort database design defines the lowest level of detail as an event, which is the combination of a collection of packet data, called the data payload to an active Snort rule, called a signature. We will use this level of grain for the SnortMart fact entity – a single row in the fact entity will be defined as a Snort event which consists of the Snort Sensor Event ID, Signature, Packet Data, IP, TCP, UDP, ICMP, and the date and time of the event.

As mentioned previously, the Snort database design emphasized write speed and thus was implemented as a normalized relational database, using no less than six entities to store the event, data, IP, TCP, UDP and ICMP information for each event (see Figure 1, Snort Event, Network and Data Database Design). In addition, data types were stored in the most efficient data types rather than the more human readable formats. For example, the IP Address was stored as an integer (3232235530) rather than the more human readable IP4 format (192.168.0.10). The six Snort entities were merged into a single fact entity in SnortMart (see Figure 2, SnortMart Event, Network and Data Database Design) that utilizes more easily searched and human readable data formats. A negative consequence of this merging and data conversion resulted in that each fact row is quite large, which is not typical for dimensional data warehouses. The best fact attributes are typically numeric and additive and in SnortMart we have the unique experience that none of the attributes fall into that category. Because of the team's lack of experience with intrusion detection, we could not further reduce fact attributes. We believe that after gaining more experience in working with real Snort data, the fact table design could evolve in the future to a more efficient design.

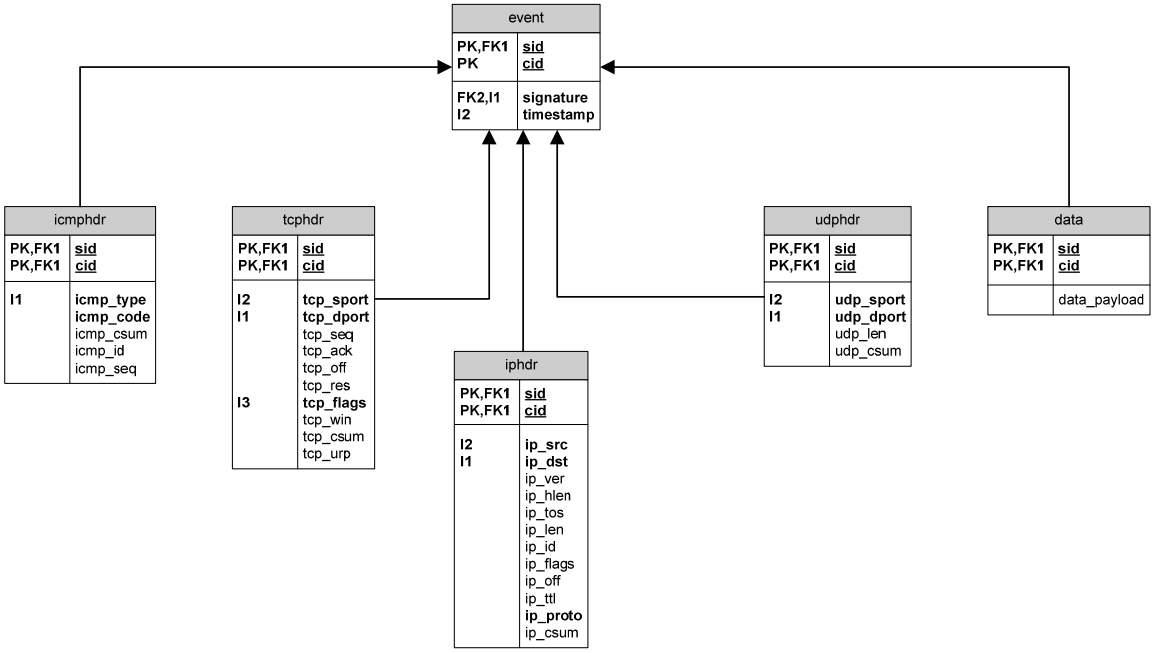

Figure 1. Snort event, network and database design

Four core dimensions are connected to the fact entity, allowing for navigation into the fact information based on signatures, dates, times and Snort sensor logs. The number of dimensions are actually quite small (2-3 is considered the minimum), but due to our lack of experience with Snort data analysis, we decided that we would select a simple set of dimensions and recommend revisiting them in the future (see Figure 3, SnortMart Entity Relationship Diagram).

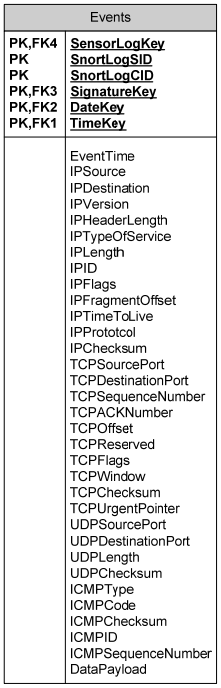

Figure 2. SnortMart event, network and database design

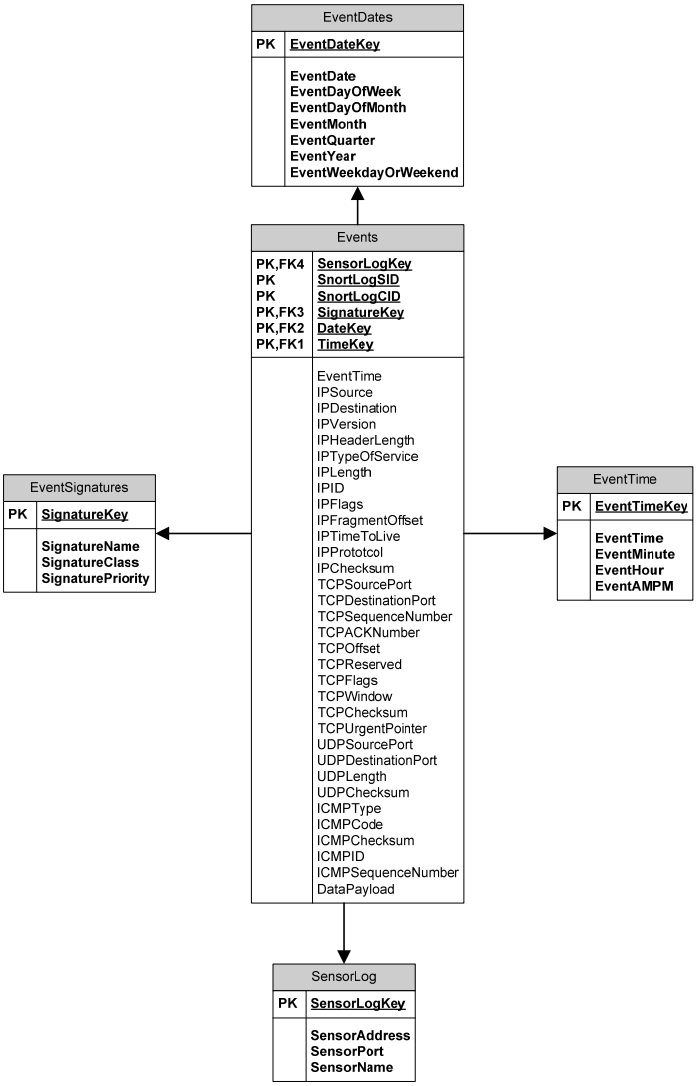

Figure 3. SnortMart entity relationship diagram

3.1 SnortMart Implementation

CSUDH was unable to provide any of the infrastructure for this project, which required the design and implementation of a Snort environment before work could begin on SnortMart. This resulted in the design and implementation of Snort Sensors, an Internet accessible web server behind a firewall to be monitored, an attack system and multiple networks containing the equipment to be monitored and a secure private network for the Snort Sensors and SnortMart. This had a considerable impact on the project, as the project team had to allocate approximately 3 weeks to design and setup the infrastructure just to generate Snort events (see Figure 4, Snort/SnortMart System Diagram).

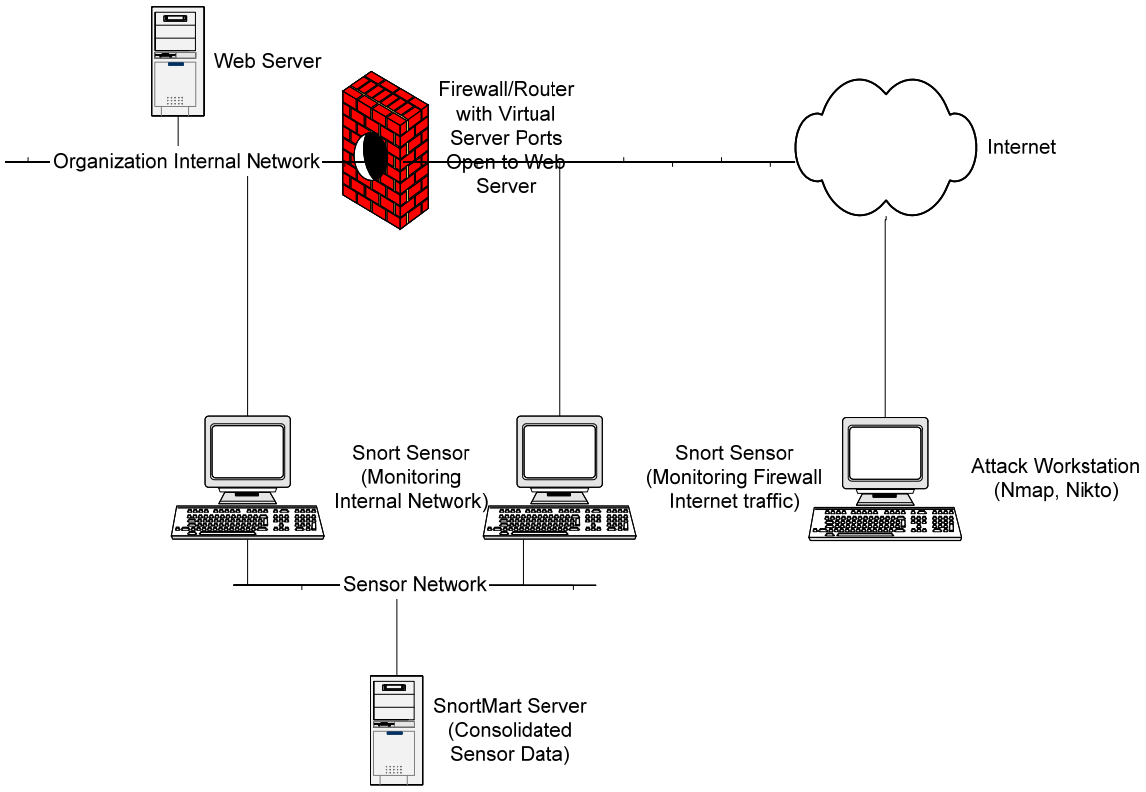

Figure 4. Snort/SnortMart system diagram

The Snort sensors monitored two network devices, a NAT Router/Firewall and a web server. The NAT Router/Firewall was configured to allow Internet access to the web server, by mapping ports 22, 80, 443, 554, and 3389 to the web server behind the NAT Router/Firewall on the internal network. This configuration was selected to allow a single attack to simultaneously attack the NAT Router/Firewall and the web server so we could generate Snort events that had identical timestamps to ensure that we could successfully merge data from multiple snort sensors with identical timestamps.

The web server was an Intel-based PC running Microsoft Windows 2000 Server, Microsoft IIS 5.0 with several active websites and Windows Media Services. Open SSH was also installed as well as Microsoft Terminal Services to allow remote management of the server.

The attack system was an Intel-based PC running Fedora Core 3 (FC3) GNU/Linux, NMap (included with FC3) and Nikto 1.34. NMap was selected as it was considered by Cox and Gerg [2], as one of the most widely used port scanners for network analysis. Nikto was selected as it was indcated by Scambray and Shema [4] to be easy to use and freely available. NMap and Nikto were used to supplement the real live probes and attacks to the NAT Router/Firewall and web server so that we could have enough Snort event data for testing.

Each Snort sensor was an Intel-based PC running Microsoft Windows 2000 Professional, WinPCap 3.0, Snort 2.3.0 and mySQL 4.3.10. Each Snort sensor system had two Network Interface Cards, one connected to the network to monitor and the other connected to a private network consisting of the Snort Sensors and the SnortMart system. Each Snort sensor was configured with identical rule sets (the set of rules included with Snort 2.3.0), to run in Intrusion Detection System (IDS) mode, and to log to the MySQL database engine installed on each Snort sensor. As indicated by Beale et al [1], logging to a relational database was selected as it is considered to be more efficient than logging to flat files. MySQL specifically was selected as the relational database engine for logging as it was freely available and the plentiful amount of documentation providing information on configuring Snort specifically with MySQL [1,4,5,6].

Finally, the SnortMart system was an Intel-based PC running Microsoft Windows 2000 Server, Microsoft SQL Server 2000 Standard Edition and the MySQL 4.1 client. The SnortMart system only had a single Network Interface card and was connected only to the private network that the Snort sensors are connected to.

As described previously, the SnortMart database model is different than that of Snort. Both entity and attribute definitions are different, and thus must be processed before it can be added to SnortMart. This process of obtaining data from Snort, modifying it and loading it into SnortMart is called the Extraction, Translation and Load (ETL) process.

Though it is possible to attach to each Snort system, extract, translate and load the data directly into the SnortMart database, this is not a desirable approach for ETL processing. This is because to perform to perform the whole ETL process all at once per Snort system will require a longer time period in which the database connection between the SnortMart and Snort system is open, which negatively affects the performance of the Snort system, as it has to maintain two database connections that will be accessing the same database tables. The MySQL database 4.1 engine for each Snort sensor is configured to use MyISAM table type, which as noted by Willams and Lane [7] is designed for the minimum amount of overhead for writing data. To achieve this, the MyISAM table type implements table locking, which provides the maximum level of performance when database transactions are short and are accessed by as few concurrent connections as possible, and thus it is critical to minimize the amount of time that the Snort database tables are accessed by the ETL process for SnortMart.

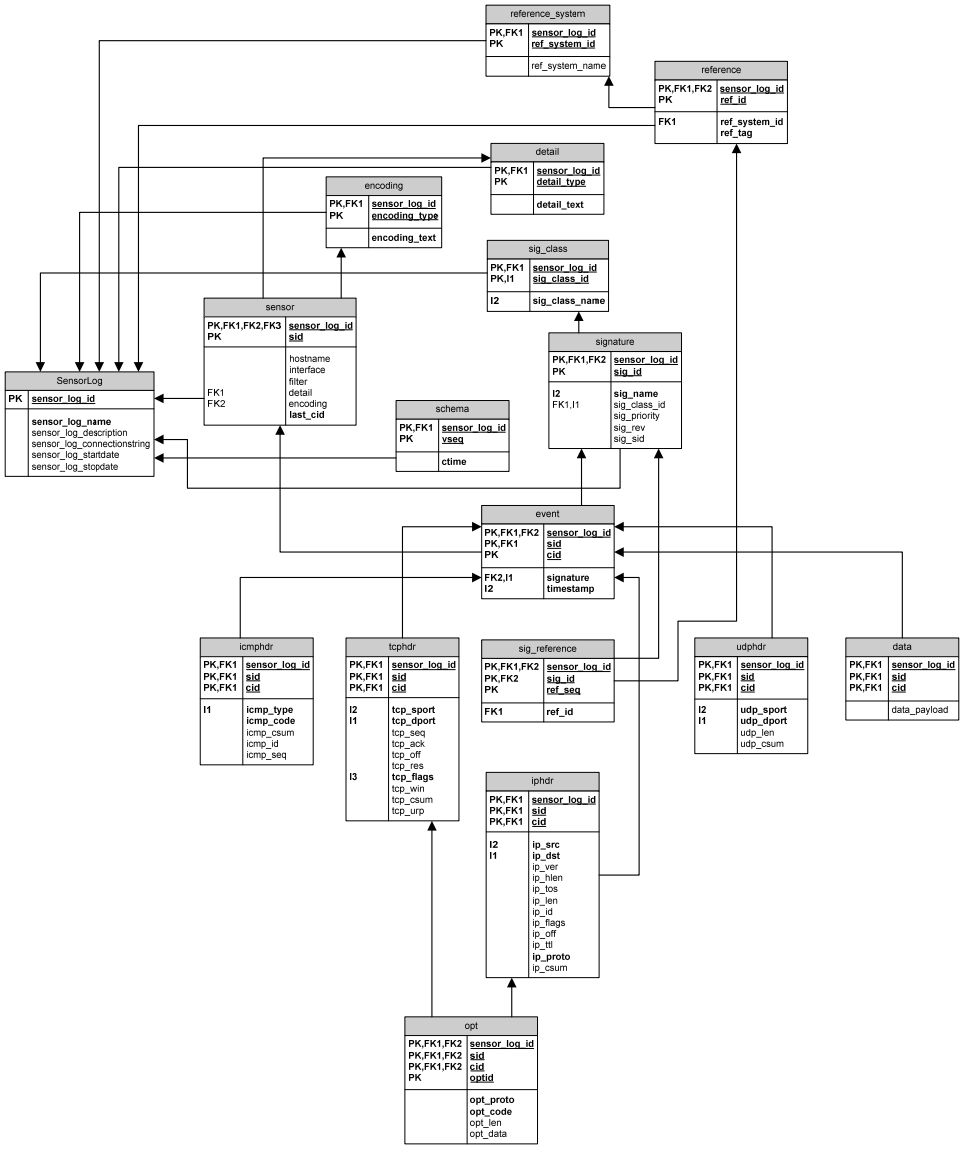

Figure 5. SnortETL entity relationship diagram

To address this issue, a separate database was created to store data in the identical format as the Snort database, with the exception of an additional attribute in each entity to ensure that data extracted from multiple Snort systems will be unique (see Figure 5, SnortETL Entity-Relationship Diagram).

The implemented ETL process performs the extraction process, extracting each table to memory, disconnecting from the Snort database, and finally connecting to and saving the data into the SnortMart ETL database. This method is used to minimize the load on the MySQL database engines used for Snort logs. The extraction implementation uses a custom built Windows Scripting Host solution using Microsoft VBScript.

After the completion of the extraction process, the Translation/Load process then takes the consolidated data in the SnortETL database and then translates the data into the format for SnortMart then saves the data in the SnortMart database.

The translation/load process was implemented similarly to the extraction implementation, using a custom built Windows Scripting Host solution using Microsoft VBScript. Two additional objects were added to the SnortETL database to aid in the translate and load process: a Microsoft SQL Server User Defined Function to convert IP Address information from an integer value to the more human readable IP4 format and a view to map event signature information from multiple Snort logs into a single universal list.

Since both the extraction and translation/load processes shared some functionality, a shared script object for the Extraction and Translate/Load process was created – the object is instantiated in both the Extraction and Translate/Load scripts.

3.2 Findings

We found the SnortMart database design allows for more simplistic use of the SQL language when constructing queries. This is due to the reduction of entities, use of human readable data types, and addition of entity attributes.

The reduction of entities allows for the elimination of complex SQL joins. The greatest example of this is found when retrieving all of the fact information associated with an event, where six Snort entities were merged into a single SnortMart entity (see Table 1, SnortMart Join Reduction).

Table 1: Snort vs. SnortMart join reduction

Snort Query	SnortMart Query
Select * From event Left Outer Join iphdr On event.sid = iphdr.sid And event.cid = iphdr.cid Left Outer Join tcphdr On event.sid = tcphdr.sid And event.cid = tcphdr.cid Left Outer Join icmphdr On event.sid = icmphdr.sid And event.cid = icmphdr.cid Left Outer Join udphdr On event.sid = udphdr.sid And event.cid = udphdr.cid Left Outer Join data On event.sid = data.sid And event.sid = data.cid	Select * From events

The use of human readable data types allows for more meaningful data to be used when constructing queries and reviewing the results returned by them. A good example of this is when accessing the IP address associated with an event. IP addresses are stored as integer values in Snort, rather than the IP4 (or IP6) format that most users are accustomed with (see Table 2, Snort vs. SnortMart Data Types).

Table 2. Snort vs. SnortMart data types

Snort Query	SnortMart Query
Select * From event Left Outer Join iphdr On event.sid = iphdr.sid And event.cid = iphdr.cid Where ip_dst = 3232235530	Select * From events Where ipdestination = '192.168.0.10'

As you can see in the table, to retrieve data by IP address in Snort requires entering the IP address as a single integer value, which is more difficult than in SnortMart, where we can use the IP4 format. Another example can be shown when viewing query results (see Table 3, Snort vs. Snort Mart Data Types in Result Sets).

Table 3. Snort vs. SnortMart data types in result sets

Snort Query and Result Set	SnortMart Query and Result Set
Select ip_src, ip_dst From iphdr	Select ipsource, ipdestination From events
3232235530 3475932798 70151371 3232235530 1070505800 3232235530 . . .	192.168.0.10 207.46.134.126 4.46.108.203 192.168.0.10 63.206.159.72 192.168.0.10 . . .

In the Snort database, it is possible to create a user-defined function to translate the integer IP values to IP 4 and modify the query appropriately to provide the same level of human readability as in the SnortMart query (see Table 4: Modified Snort Query for Human Readability).

Table 4. Modified snort query for human readability

Modified Snort Query for Human Readability
Select dbo.ipint2ip4(ip_src), dbo.ipint2ip4(ip_dst) From iphdr
192.168.0.10 207.46.134.126 4.46.108.203 192.168.0.10 63.206.159.72 192.168.0.10 . . .

However, evaluating the execution plans of queries using the user defined function in Snort (see Figure 6: Snort Performance Analysis 1) versus the already translated IP addresses in SnortMart (see Figure 7: SnortMart Performance Analysis 1) demonstrated that queries using the SnortMart database are more efficient. This was evident in the larger and more complex execution plans which also indicated that the database engine had to evaluate rows by performing the IP address conversion on every integer IP address, then comparing it to the IP address that is being searched for (192.168.0.10). This is in sharp contrast to the SnortMart version of the same query, which an index seek was used to select the specific rows to return.

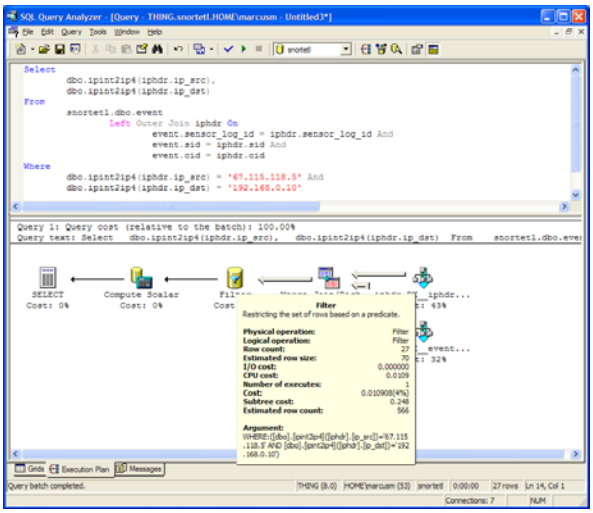

Figure 6. Snort performance analysis 1

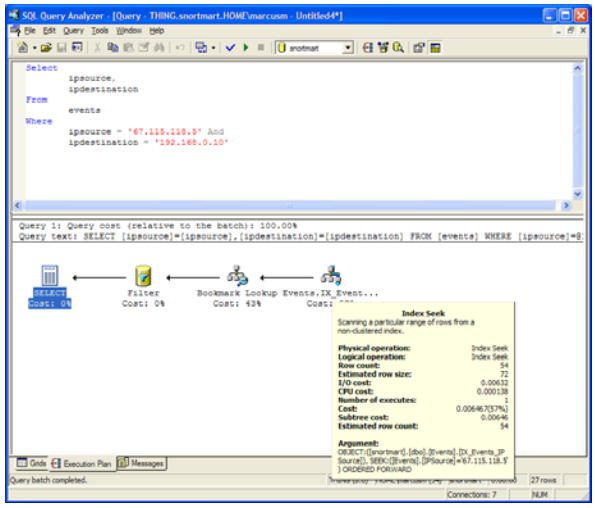

Figure 7. SnortMart performance analysis 1

Similar to the human readable translation features of SnortMart, the addition of extra entity attributes allows SnortMart queries to avoid the use functions or other complex conversions of data that are more difficult to write and pose performance penalties. An example of this is when working with date or time values (see Table 4, Snort vs. Snort Mart Extra Entities).

Table 5.Snort vs. SnortMart extra entity attributes

Snort Query	SnortMart Query

Select
  timestamp,
  signature
From
  event
Where
  datepart(dayofweek, timestamp) In
(1,7)
	Select
	  eventtime,
	  signaturekey
	From
	  events e
	  Inner Join eventdates ed On
	  e.datekey = ed.eventdatekey
	Where
	  ed.weekdayorweekend = 'weekend'

Though the Snort query is actually shorter, there is greater complexity in knowing the datepart() function and that Saturday = 7 and Sunday = 1.

Also, an evaluation of the execution plans for these queries demonstrated that the SnortMart queries were more efficient (see Figure 8: Snort Performance Analysis 2 and Figure 9: SnortMart Performance Analysis 2). This was evident in use of a clustered index scan in the Snort query (which means that all rows in the entity were evaluated) versus the use of an index seek in SnortMart to quickly find the rows to return.

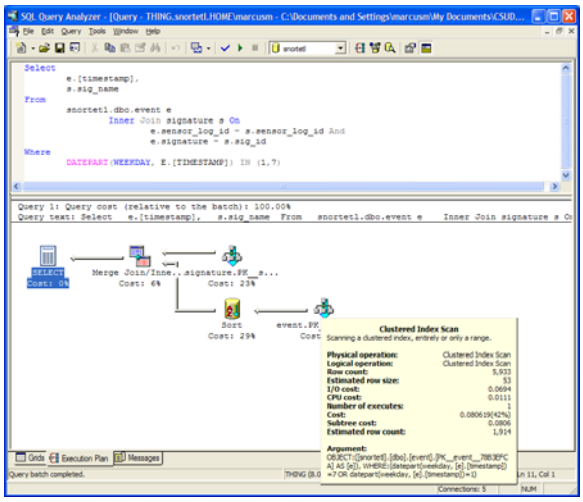

Figure 8. Snort performance analysis 2

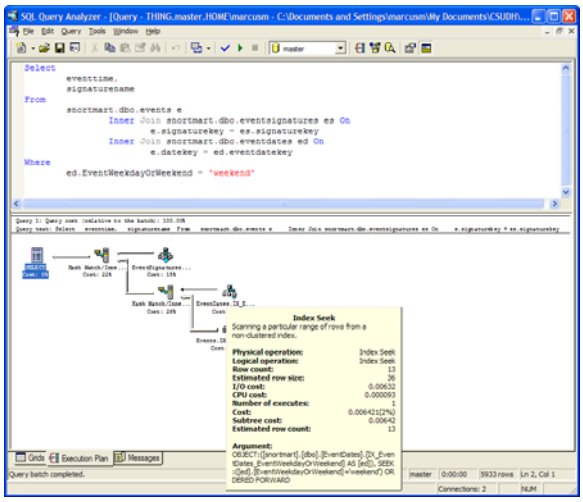

Figure 9. SnortMart performance analysis 2

In addition to easier query construction, the use of a database separate from the operational databases used by Snort sensors provides an environment that encourages greater use, as a SnortMart user does not have to be concerned about affecting logging performance of Snort systems. We feel that these benefits will help encourage reporting and analysis of intrusion detection data.

4. CONCLUSION AND FUTURE WORK

Not having an established Snort environment already in place really limited our work, as a great deal of time was spent setting up an environment to create intrusion detection data and to develop a basic understanding of Snort. This meant that the implemented dimensional model is really a very early prototype and needs to evolve into a more efficient model. Since the team had little or no experience with Snort, this would occur after a longer cycle of working with the Snort and SnortMart data.

Future work should emphasize a revisit of both the fact and dimensions; in general there probably should be a reduction of fact attributes and an increase of dimension entities and attributes.

It is believed that a good approach for achieving this would be an expansion of the solution: including a consolidated version of the operational Snort database that is to be used in conjunction with SnortMart for reporting and analysis.

One of the key reasons that the fact entity has so many attributes was the concern of excluding a fact attribute and thus not having that data available at all. This resulted in the inclusion of practically all of the Snort event data, which is a poor way to create dimensional databases – an attribute typically should only be included in the fact entity if there is a defined need for it.

An example of this is the data payload – it is valuable information when looking at a detail level, as in some cases the data payload can contain the information that produced a Snort alert. However, as the data payload from a single Snort event can be up to 8K and is stored in hex format, there is a question whether or not analysis from a perspective of looking for trends and/or patterns in a large volume of data of this type is reasonable. It was decided that we would include it, because if we didn't, we would not be guaranteed to be add it in the future with a complete set of historical data.

Adding a consolidated version of the operational Snort database to the solution will provide us the best of both worlds – the consolidated Snort database will contain all data from each Snort sensor preserving the data in its original format. The SnortMart dimensional database will only contain fact attributes and dimensions that have been determined to provide a real value from a large database analysis point of view (facts that really fits in "the grain").

A user of this solution would be able to perform analysis by looking at the dimensional database (SnortMart), then connect to the consolidated operational Snort database and retrieve a greater level of detail

when needed. Providing a seamless connection between the two database could be achieved within the logic of a custom reporting application, through the use of special features of the database engine or by simply replicating all of the consolidated operational data into SnortMart.

If it is determined that an additional Snort attribute needs to be added to SnortMart as a fact or dimension, this can be added when needed, and most importantly, all of the historical data associated with it can be populated into SnortMart from the consolidated operational Snort database.

In fact, the foundation for the consolidated operational Snort database already exists – the SnortETL database. It does contain all of the Snort data in slightly modified versions of the original Snort database – the only modification was to add additional primary key information for each entity to allow data from multiple Snort logs to coexist. The key concern is to ensure that reporting from SnortETL will not negatively impact the ETL process, which is a key requirement of the SnortMart database. Limiting connections during the ETL processing window may provide an acceptable solution to this, and could be the topic of further discussion.

REFERENCES

[1] Beale, Jay, 2004. *Snort 2.1 Intrusion Detection*, Second Edition, Syngress.

[2] Cox, Kerry and et. at., 2004. *Snort and IDS Tools*, O'Reilly Media, Inc.

[3] Kimball, Ralph and et. at., 2002, *The Data Warehouse Toolkit*, Second Edition, John Wiley and Sons.

[4] Scambray, Joel and et. al., 2002, *Hacking Web Applications Exposed*, McGraw Hill/Osborne.

[5] Jeremy Brian and Hewlett, *Snort Users Manual*, Caswell, , 2003, Snort.org. <http://www.snort.org/docs/snort_htmanuals/htmanual_232>

[6] Roman, Danyliw, *Snort Database Plugin Documentation*, 2002, Snort.org. August 2002. <http://www.snort.org/docs/snortdb/snortdb.html>

[7] Williams, Hugh E., and et. al., 2004, *Web Database Applications with PHP and MySQL*, O'Reilly Media, Inc.