

Implementing Snort IDS Using FreeBSD

Daniel Owen

Middle Tennessee State University

Abstract

This paper explores the elements involved in implementing a Snort IDS and associated software. The considerations and steps taken in building the IDS are discussed, as are the pitfalls and compromises inherent in the implementation discussed in this paper. This paper should be helpful to anyone considering setting up an IDS for the first time regardless of the final software solution that is chosen.

Overview

This paper will outline the steps taken and the lessons learned through the implementation and use of a Snort Intrusion Detection System (IDS). I will discuss the items that worked well, items that were troublesome as well as how and why certain decisions and tradeoffs were made. This should be helpful to others implementing an IDS solution even if they vary significantly from my methodology. At a minimum I hope this document can serve as a starting point for new IDS implementers and maybe help others avoid some of my mistakes.

In 2004, after much lobbying, it was decided that a network based intrusion detection system (NIDS) should be implemented as an additional level of security in depth at the company I work for. At the time, I hoped to be able to use the IDS for a few purposes. First I wanted a way of seeing just what type of dangerous traffic was coming through the firewall and into the protected LAN. Secondly I wanted to see what traffic was leaving the network. The second goal had two sub aims. First by looking at traffic leaving the network I hoped to be able to review policy compliance. More importantly I hoped to be able to proactively detect attacks such as virus infected machines sending traffic outside of the LAN. I hoped a NIDS on the internal network would allow me to detect any intentional attacks carried out by internal staff. Finally I hoped that being able to watch the network for anomalies might help in identifying network problems that are not necessarily malicious. Some of these goals have been met and some have been abandoned to be implemented later or with another product.

A little background about the company that I implemented this IDS for is important to allow understanding of why certain decisions were made. The company has

between forty-five and fifty full time employees at any given time. Turnover is relatively stable with around ten percent turnover on a year to year basis. Turnover is primarily in the lower levels of the organization with manager level and above turnover being low. We typically have approximately ten college interns working at any given time. Most interns work for one semester, but it is not uncommon for interns to come back for a second semester or to be hired as part time temporary employees while they are still in school. The number of temporary workers varies by the time of year but is typically less than five. We also have a few contract workers that work from the corporate office. Contractors working in the office typically number less than five, but at times there will be as many as twenty contractors working in the office. Contractors are typically segregated to their own network, but there are times when contractors must be allowed to use the protected corporate network to access certain resources. As a general rule, employees of all categories are treated with a fair amount of trust when using company owned computing resources.

The company is small enough that it should not typically be a major target of attack, but there are a number of factors that increase the risk factors for the company. The company has a fairly high level of visibility so that makes it something of a target. The company is part of the music industry, which has been a popular target of attacks over the last couple of years. Equally importantly, many attackers are not concerned as much about the target as the exploit. These attackers, often referred to as script kiddies, are not particularly skilled attackers but attackers of opportunity. Script kiddies will use tools developed by more knowledgeable individuals and look for any target that is vulnerable. This does not mean that the exploited companies will not be targeted for

additional harm once these script kiddies have a hold on a company. The attacker may use the exploited company as a jumping off point for other attacks. It is also possible that once an attacker of opportunity gets in they may discover valuable resources inside the company. Depending on the company, these resources of opportunity could include intellectual property, credit card data or any number of other usable or sellable commodities. For these and other reasons I do not feel that any company is too small or too unimportant for security due diligence to be important.

It was decided that adding a level of security at this time was prudent. It is hoped that we will see very little activity on the IDS meaning that our other levels of protection are blocking attacks before they enter the protected network. Without the use of an IDS we can only guess as to the effectiveness of our other forms of computer security.

The initial challenge, as is often the case for security initiatives, was the cost of implementing a more secure environment when everything appeared to be working just fine. Security is very often an invisible component that no one wants to spend money on until they have experienced a breach. At that point the cost of recovery is much higher than the cost of prevention. An IDS is not, strictly speaking, designed to stop intrusions but to alert when an intrusion is attempted so that counter measures can be taken to thwart the attempt or to improve clean up efforts if an attack has been successful. Commercial IDS packages were ruled out early in the IDS evaluation process. Cost was a major factor in this decision, but equally important was the belief that an open source solution that allowed best of breed combinations of software would provide a better, more customizable solution at a lower cost.

Software and hardware

I chose simple PC hardware. Most of the hardware is not what I would classify as “server class” hardware. The decision not to use true server class hardware was made primarily as a cost savings measure. We spent approximately \$1200 on all of the hardware used. An inexpensive commercial server similarly configured would have cost more than double this amount. The tradeoffs that I made were primarily in support. I have used essentially the same configuration on another server with good results so I did have some experience with using this configuration successfully.

Most companies that sell server hardware assume that the server will have either Windows or Linux loaded on it, therefore it can be difficult to get hardware support when expected tools are not available on the system since technical support representatives often want to go through a diagnostic process to assure that hardware has failed and the end user is not experiencing a software problem. This has also been an issue for me in the past with Windows system that I have stripped of all extra software to improve the security of the server. There are certainly compromises in the architecture I designed. With the exception of hard drives, there are not any redundant parts and I did not choose to use ECC memory. These same compromises are often found in entry-level servers as well. I do not see these compromises as a major problem for this implementation. It should be mentioned that when looking for complete systems that include both hardware and software a review of the underlying hardware is important. I have seen vendors offer solutions, especially in the SMB market, that use hardware that is not as resilient as this machine. The final decision came down to a belief that if we have a hardware failure even an extended downtime would be acceptable since this is essentially an invisible

system as far as the end user is concerned. An outage is undesirable but not a business critical issue. In looking at budgeting issues I felt that it would be difficult enough to receive approval for the small amount of money that I was asking for so asking for funding for a more resilient server was not an option.

I started with a PC architecture that I have used for other server solutions. I chose an AMD Athlon 64 to allow us to take advantage of 64 bit processing improvements while still having backward compatibility with 32 bit applications. I installed two gigabytes of RAM to limit the likelihood of exhausting the available physical memory. I used a pair of large SATA drive in a hardware based RAID 1 configuration to allow a large number of logs to be stored before we need to archive or delete old data. I also used Intel dual port gigabit server NICs to help offload the processing of network traffic to the NIC. A dual port network card was chosen to allow room for future expansion, but we are currently only using one of the two available ports.

I chose a combination of FreeBSD, Snort, MySQL and Analysis Console for Intrusion Databases (ACID) for this NIDS implementation. FreeBSD was chosen for a number of reasons. It is believed that, at least in the initial out of the box configuration, FreeBSD is more secure than Windows and most Linux variants. This is due to the fact that both Windows and Linux come with a great deal of software that is loaded and started by default. A large number of services are started on these systems in an effort to make the system easier to use. Much of this unneeded software is network aware and must therefore be striped out to create a more secure environment which is obviously important when implementing a security tool. Stripping an operating system to its bare essentials is a time consuming and tedious process on Windows and the expertise to strip

a Linux implementation to a bare install does not exist within the company. It should be mentioned that had there not already been experience with FreeBSD in the company Windows would have been a viable, although more difficult to secure option. Linux also has a good following amongst Snort users since it is the native operating system for most of the developers who work on Snort and associated add on software. OpenBSD was also briefly considered. OpenBSD has a reputation for being the most secure operating system available due to the emphasis put on security and code auditing by the OpenBSD team. Ultimately it was decided that due to available knowledge FreeBSD was the best solution.

Snort was chosen because it has similar performance and functionality when compared to commercial IDS solutions, and no other open source IDS comes close to Snort in functionality. Because of Snort's popularity there is a great deal of support available for Snort. This support comes primarily in the form of message boards, but there are also a number of paid support options available as well as a commercial derivative of Snort available from Sourcefire. We have the option of purchasing commercial support at a later date if we decide that we need the additional level of support. Another equally important advantage of using Snort is that there are a large number add on products that can be used to expand its functionality.

It should also be mentioned that there is not only a great deal of support for Snort in the open source world. A number of commercial packages have implemented interfaces that allow users to integrate signature files from Snort into these commercial products. This makes a strong statement about the reliability and quality of the signature files available for the Snort package.

Snort is a signature based IDS. This means that Snort relies on a continually updated set of rules to detect new attacks. These rules come from a number of sources but the basic groups I am using are Sourcefire and Bleeding Edge Snort. The administrator of a Snort IDS also has the option to write their own rules if they see an attack that is not yet identified by available Snort rules. Sourcefire is the developing organization for Snort so they are the acknowledged holder of the stable rule sets. There are two possible options to acquire rules from Sourcefire. Sourcefire offers a subscription at a price of \$1,795 per year for an unlimited number of servers or the same rules are available for free five days after they are made available for paying subscribers. This is a balance that Sourcefire has made between their free roots and the desire to make a profit off of their product. In a fast paced world where new attacks can traverse the Internet in a matter of hours, five days can be a long time. By the same token for a small company trying to make the most of their resources we have for the moment decided to wait and see if it is worth the cost of moving to the paid service. Bleeding Edge Snort is a project made up of security professionals and hobbyists who write signatures for the Snort IDS. These signatures are not as rigorously tested as the rules provided by Sourcefire so there is a higher likelihood of experiencing false positives. At the same time Bleeding Edge Snort releases signatures much more often than Sourcefire so they as a general rule should be able to catch problems faster. It is not typically recommended to automatically install new IDS signatures without reviewing the changes between the old version and the updated version. This is unusual in comparison to other signature based environments, such as anti-virus or anti-spam solutions, where downloading signatures is typically scripted to happen automatically. This difference in best practices for IDS is due to the fact that an

errant IDS rule can be more devastating to network traffic than a bad anti-virus rule especially if the IDS is configured to automatically block traffic. There is also a difference in that a virus is a virus for any company, but traffic that is a definite attack for one company may be completely normal for another company.

I chose to use MySQL for the SQL database. MySQL and PostgreSQL are both open source SQL implementations that work with the mix of software I chose. MySQL seems to have a larger market share and associated support at this time. Snort does not need SQL to run, but our chosen alert management product, ACID, needs alerts to be stored in a SQL database. Along with storing data in a SQL database Snort also drops copies of all alerts to a human readable text file. This can be useful for debugging and investigation alerts, but the log file must be rotated on a regular basis or it will become too large to be useful.

The final major item that I needed to have a complete solution was an alert manager. Snort can handle alerts in a number of ways including alerting to the console, sending e-mail, sending a pager message, sending SMS messages, writing to a log file and logging to a database. Logging to a database allows a third party tool, such as ACID, to read the logs and generate reports. This is a less proactive approach than some of the others but it is what I needed for this project. There are a number of analysis consoles available for Snort and nothing keeps someone from using more than one. I chose ACID because it has the features that are needed and has proven itself to be stable. The only disadvantage of ACID is that it is not currently being actively developed. If at a future time a new analysis package that has needed features not available in ACID becomes available we can move to a new product at that time.

Installation Process

Network Architecture

Our Snort NIDS is currently attached to a gigabit span port on a 24-port gigabit switch used by all servers in the office. The LAN interface for the firewall also attaches to this switch so all traffic leaving the LAN going to the Internet and traffic coming in from the Internet can be seen. We are not currently looking at traffic in the DMZ or on the network used by contractors. (See Figure 1) The contractor network is essentially an unmanaged network with no egress filtering so it will continue to be ignored by the IDS.

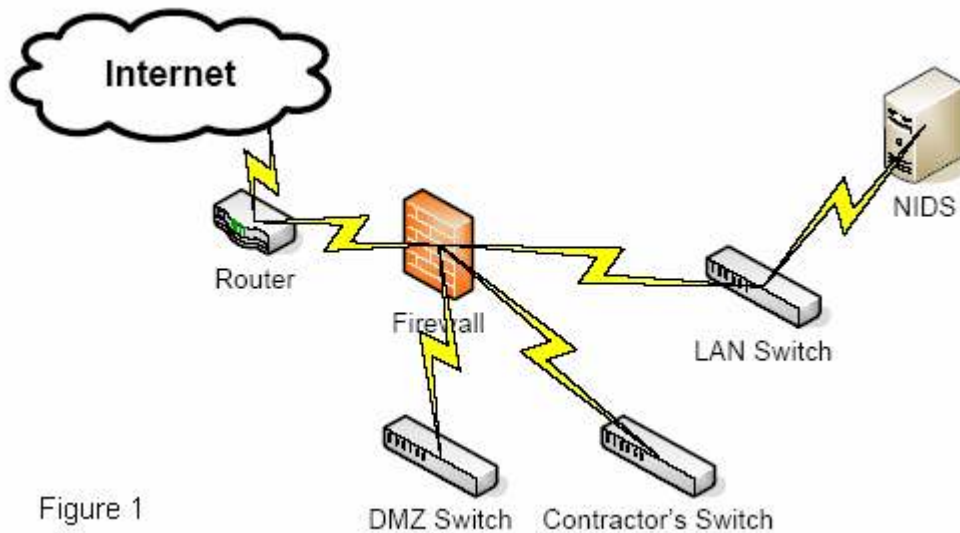


Figure 1

I may at a later date add a sensor to the DMZ. I will need to use a tap for the DMZ since it does not have an available span port. Much of the decision about whether to implement a sensor in the DMZ will depend on whether the IDS proves itself to be valuable in its initial implementation in the LAN. I believe that I may be able to run the same physical IDS sensor for both the LAN and the DMZ. Before I commit myself to this architecture I will need to investigate the possibilities of the IDS being used as a jumping off point to go from the DMZ to the LAN. Due to both networks being attached to the

IDS this is possible but not necessarily any easier than compromising the firewall which is also attached to both networks. (See figure 2)

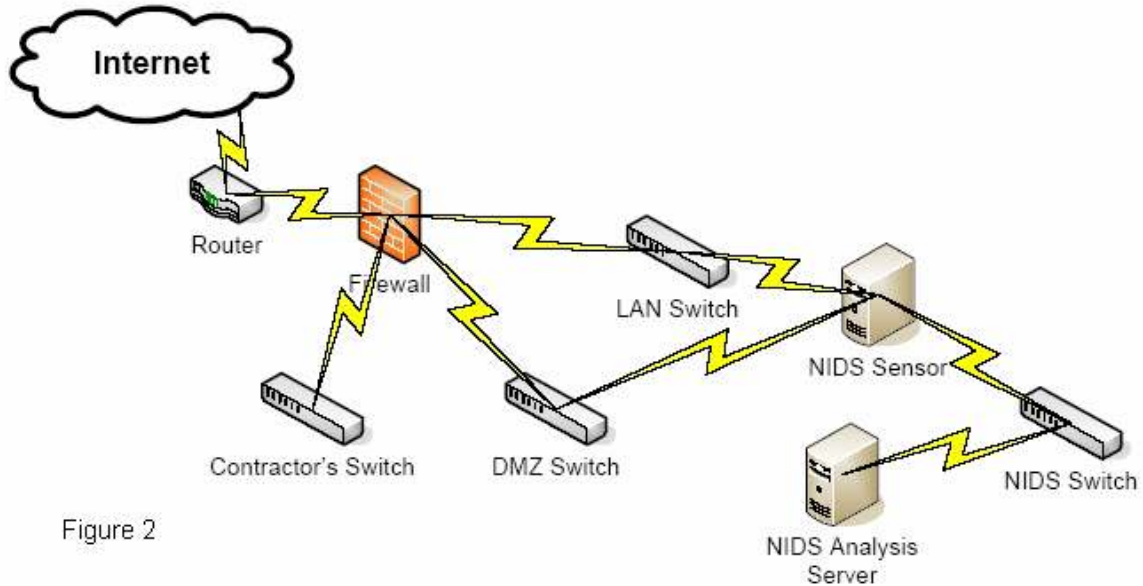


Figure 2

IDS Server Configuration

As a cost savings measure I decided to implement all IDS functions on a single server. There are three major advantages to this architecture. There is the obvious cost savings of running one server. There is also a level of simplicity inherent in only having one system to manage. Finally since all traffic between the sensor and logging server the information on Snort is local to one machine there is no need to protect the traffic between the sensors and logging server. This final advantage can be made moot by creating a dedicated network for traffic between the IDS sensor and the analysis server. Considering the low cost of networking equipment a dedicated network for IDS traffic should not be a strain on the budget of even the smallest companies. In a system with only one sensor a crossover cable could even be used to connect the sensor to the analysis server. Network traffic between the sensor and the analysis server can also be encrypted

as an additional level of protection. On highly loaded servers the processing power needed for encryption and decryption must be weighed against the risk of sending unencrypted traffic between the servers. On a dedicated network between IDS elements this should be a fairly low risk.

There are also a number of disadvantages to a configuration that holds all services on one machine. First there is the issue to resource utilization. A major issue in the development of IDS systems is making sure that no packets are lost. On a loaded fast network this can be a challenge even without additional processes vying for limited memory and processing power. In this particular environment it was decided that the utilization is typically low enough that we would try running all processes on one machine and separate the analysis and sensor processes if needed. Initially I saw drop rates near one percent during SQL queries while using ACID. This was primarily due to a poor choice in my configuration of the IDS. I had initially configured the IDS to treat the internal network the same as the external network. I will discuss this more shortly but for the moment suffice it to say that I received thousands of false positives per day in this configuration leading to a very large amount of data being stored in the SQL database. During SQL queries from the ACID interface CPU utilization reaches in excess of ninety-nine percent for extended periods of time. During normal usage without database queries CPU utilization remained below one percent utilization even with the high level of false positives. I can certainly say from this experience that in an environment that will generate a large number of alerts running on a single server is not a good option. I can also say that for all but the highest utilization networks the sensor should not require an extraordinary amount of processing power considering the low processor utilization I

have observed. After I reconfigured the IDS to treat the LAN and WAN differently the number of alerts has gone down significantly and there is no longer an issue with dropped packets when using ACID. Memory is heavily used but so far there has not been any use of the swap file and I do not expect this to change.

In the future if I decide to add additional sensors I will also add a dedicated network segment for traffic between IDS servers. This will help to alleviate some of the risk of an attacker viewing traffic going from the sensor to the logging server without needing to encrypt the traffic.

An important disadvantage of the single server implementation that I chose is security in the event of a compromise of the IDS server. In a distributed IDS implementation a sensor should be the only element of the IDS architecture that is exposed to the outside world. The logging and analysis server should be on a dedicated network segment. This means that the analysis and logging server cannot be directly attacked since it is not attached to a shared network. An attacker will need to first compromise a sensor and then from there they will be able to start to attack the analysis server. In a combined solution like the one I have implemented the compromise of the IDS will mean that IDS logs can no longer be relied on since a competent attacker will either delete all logs or edit out their own attacks. This is a risk that we felt we can afford at this time. In the future I may move to a distributed system to help eliminate this potential vulnerability as well as to help reduce resource utilization on the sensors.

Initial Implementation Difficulties

The first major difficulty that I encountered was in my choice to take advantage of AMD's 64-bit processor extensions. At the time of installing this system I felt that

moving to the AMD 64 port of FreeBSD was a fairly safe move. FreeBSD has supported the AMD 64 architecture since version 5.2, which was released in January of 2004.

Unfortunately some of the tools that I needed did not work out of the box with the AMD 64 optimized version of the operating system. After many fruitless hours trying to make everything work together I decided to step back and see if the i386 (32 bit) version would work better. Within a few hours I had an incomplete but functional system. I decided at that time to stick with i386 for the time being. I do not see a major problem with processing power so this seems to be an acceptable choice for the moment.

Once I got past the issues with trying to get the AMD 64 version of FreeBSD to work with Snort and its associated software and fell back to the i386 version of FreeBSD most of the implementation was fairly straightforward. I will make an observation that there is a tendency for a great deal of online and printed documentation to assume that the user is running on Linux and often a specific flavor of Linux. Because of this some issues can be a bit more difficult to correct since I need to convert solutions from Linux oriented to FreeBSD oriented. That is typically not a big problem. Most of these issues come down to differences in assumptions between System V and BSD derived UNIX like operating systems. Having used FreeBSD for some time I have become accustomed to this shortcoming. I would assume that some of the same issues would exist for users of the Windows port of Snort, but Snort does seem to have a more dedicated Windows based following.

Time Needed to Implement

I would estimate that once I got past the initial road blocks I spent twenty to twenty five hours getting a fully functional system working. I would estimate that I could

reproduce the same system in less than a day using the knowledge I gained from this first implementation.

Once I got the initial system up and running I spent a number of follow up hours in continual improvements to the system. These improvements include reductions in false positives as well as improved functionality through additions of new software.

I have seen the statement that it take approximately one week for implement an IDS, get it fully configured and get the false positive problem under control. In my experience with this project, that seems like a fair estimate for a reasonably competent administrator. For someone new to IDS software an extra few days will be needed to allow time for mistakes and setbacks. I'm not sure that an IDS can ever truly be called finished since there will always be alerts to investigate, but it can certainly be moved to a maintenance phase.

Refinements

The first major improvement that I implemented after the initial Snort IDS was functional was to reduce the number of false positives I was seeing. Some false positives will not be avoidable but I saw well over one million alerts in the first three weeks the system was running. Approximately half of those came from one single rule. At this level the alerts were not reviewable on a meaningful basis. When I started to review the alerts the problem quickly became obvious. By default Snort defines the external interface as all traffic including traffic seen on the local network. I left this configuration in place so that I would see any attacks against servers originating from the LAN. As I quickly discovered, due to the way that signatures are written this configuration causes a large number of false positives. Much normal traffic should be worried about if it is coming

from outside the trusted network. For example server message block (SMB) messages are a normal occurrence on a Windows network but not something an administrator typically wants to see coming in from outside the LAN. The Snort Manual suggests that the internal interface be defined as the IP range of the protected network and the external interface be defined as any traffic not originating from the internal interface. In retrospect this makes sense. The reason that so many attacks work is because traffic from untrusted sources is allowed to operate as if it was trusted. The IDS signatures have no way to tell that traffic is safe if the internal interfaces are treated the same as external interfaces. This does have the unfortunate disadvantage of meaning that some internal attacks will not be detectable. Most problems such as Trojan horse software and viruses should be detected regardless of where the traffic originates since the rules are written to look for this type of traffic regardless of its origin. There is also the option of writing custom rules to watch for known problems.

According to all of the IDS literature filtering out false positives is an ongoing problem for any IDS implementation. The level of false positives in a default IDS implementation is also a big reason that people give for abandoning IDS implementations. I expected this on the front end so that is helpful in setting my expectations. In all honesty I did not anticipate how high the false positive rate would be, but once I reconfigured Snort to treat the LAN properly I am down to tens of false positive alerts per day, which is a very manageable level. If I can filter out these false positives without decreasing the effectiveness of the IDS I will do so, but since the false positive level is very manageable these false positives are not a particularly troublesome issue.

I was seeing some dropped packets during heavy database usage before reconfigured Snort to treat the LAN differently than other traffic. This was primarily true when I did log analysis using ACID. At the highest points of dropped packets I was seeing nearly one percent of all packets being dropped for short periods of time. This was significant because any lost packets are potentially lost attack signatures so dropping packets is not acceptable if it can be avoided. This was a good lesson on why having both the analysis server and the sensor on the same server is not a good idea in a high traffic network. Sensors can be implemented using fairly inexpensive hardware so adding sensors is not an excessively expensive choice.

The IDS is running quite well. I have been pleasantly surprised throughout this process by how easy the whole process has been. IDS implementations have a reputation for being almost a black art, but I have not seen this to be any more difficult than any other new system implementation and easier than some.

Conclusions

This has without a doubt been a learning experience. There were a number of assumptions that were made a year ago when we started looking at implementing an IDS and laying out a budget. Some of these assumptions have turned out to be simply wrong.

I am very happy with Snort and the other software that I have used to build this IDS. Snort has proven itself to be rock solid when installed on top of FreeBSD. Snort is not particularly resource intensive, but some of the associated software that I am using to make it a complete solution can be resource intensive at higher loads.

I would certainly encourage anyone who is considering implementing an IDS to consider Snort as a possible solution. As for my choice of operating systems, FreeBSD

has been a good choice for me. For administrators familiar with FreeBSD it is an excellent choice. I would encourage anyone planning to follow my footsteps to choose an operating system that they have experience with. I believe far too often people choose an operating system because they allow someone else to convince them that one operating system is better or more secure than another. There is sometimes validity to this argument, but in most circumstances a good security aware administrator should be able to make any operating system they have expertise in reasonably secure. That same administrator placed in front of an unfamiliar system will probably be able to make the most secure system an attacker's dream. If a person has Linux experience that is the system that will probably be easiest and most well documented due to the number of other people using Linux and the fact that Snort is primarily developed on Linux. Unfortunately, most Linux distributions will also be more time consuming to strip down to a secure configuration due to the kitchen sink approach that many commercial Linux distributions have taken. For Windows administrators without UNIX experience Windows is a viable solution, but servers may need to be sized a bit larger to compensate for the resources taken by the operating system.

As for ACID and MySQL, I believe both are good choices. ACID is still the de facto standard for log analysis with Snort implementations. MySQL is a very capable SQL implementation, but if an implementer has more experience with PostgreSQL or Microsoft SQL that is probably going to be the SQL implementation that should be used.

Once again, for anyone considering installing an IDS I encourage you to look at Snort. If you want to buy a commercial product with associated commercial support I would encourage you to investigate Sourcefire, which is the commercial arm of Snort.

Snort has proven itself to be a powerful IDS solution. Even if an administrator decides to use another product after trying Snort the lessons learned using Snort and it's associated tools will assure that the decision to use another product is an educated decision based on experience.

Review of Resources Used

I used a number of documents in the implementation of the Snort IDS outlined in this paper. Below, I have included a short description and my opinions of the resources I made the heaviest use of.

Snort 2.1 Intrusion Detection (2nd Edition)

Snort 2.1 Intrusion Detection (2nd Edition) is useful as a general introduction to intrusion detection and Snort. Having already versed myself in more in depth intrusion detection texts I did not find the coverage of IDS principals to be particularly useful. For someone who only wants to review the basic IDS principals quickly and without a great deal of extra detail the IDS coverage in this book is sufficient. Much of the information on Snort felt like a retelling of Snort Users Manual that I had already become familiar with. Part of this feeling may be due to the fact that members of the Snort development team who undoubtedly had a hand in the user's manual wrote the book. This book does go into more detail on some subjects than the Snort Users Manual. There is a good step by step set of instructions for installing Snort and associated software on either a Windows or a Linux system. Overall this book seems to be a pretty good overview of Snort for someone looking to use only one resource, but I do not see anything that is not also available in other documentation available.

Implementing Intrusion Detection Systems

Implementing Intrusion Detection Systems by Tim Crothers is an excellent introduction to the topics important to implementing any IDS. Crothers uses Snort as a

reference IDS system, but the coverage of Snort is not intended to be comprehensive. Crothers does an excellent job of giving a very basic overview of underlying protocol elements that need to be understood to be a competent IDS manager without going into excessive detail for the generalist.

Network Intrusion Detection

Network Intrusion Detection by Nortcut and Novak picks up where Crothers leaves off. In this book the in depth elements of networking technology are discussed with an emphasis on NIDS. This is not, in my opinion, an ideal book for someone starting to learn about IDSeS but is more appropriate for someone who has some experience with an IDS implementation and wants to learn to be a better IDS manager. For anyone who wants to understand attacks at a low level this is a mandatory book.

Snort Cookbook

The *Snort Cookbook* follows a well defined pattern set by O'Reilly and Associates in the cookbook series. The book follows a pattern similar to that of a frequently asked question (FAQ) file in that the authors take a given problem and give a concise answer without an expectation that the reader has already read previous portions of the book. This is an excellent format for the practitioner that wants quick answers to immediate questions. This is also a good book for someone trying to familiarize themselves with all of the available options in Snort. Overall the book is well written and easy to read. None of the content is particularly unique to the Snort cookbook, but having all of the answers in one place is probably worth the cost of purchasing the book. It is

also helpful that each section ends with a list of additional resources to consult for more in depth discussion of the topic.

Snort FAQ

The Snort FAQ is what it sounds like. It is a list of frequent questions and answers. It is not particularly interesting reading, but it is worth at least skimming the index before starting a Snort implementation just as added knowledge of what to look out for during the implementation process.

Snort Users Manual

The Snort Users Manual is the document that all discussion about Snort eventually refers back to. The manual is an excellent place to start when considering implementing a Snort IDS. All of the common implementation questions are addressed so it is a must to check the manual before asking questions in Snort community support areas. If a person is planning to read nothing else before implementing Snort the Snort Users Manual is probably the place to start. My only complaint about the user's manual is that it is a bit dry for reading straight through.

How to Setup and Secure Snort, MySQL and ACID on FreeBSD

4.7

I used How to Setup and Secure Snort, MySQL and ACID on FreeBSD 4.7 Release as a starting point for my Snort implementation. The document is well written and gives just enough detail to explain why given steps are being taken while still putting the emphasis on a quick install guide feel. Unfortunately the documentation is somewhat

out of date so many of the steps needed modification to allow them to work with newer version of the software. Overall this is an excellent document that saved me many hours of discovering how to make different components work together.

Resources

- Alder, Raven, Babbin, Jacob, Doxtater, Adam, Foster, James C., Kohlenberg, Toby and Rash, Michael. (2004). *Snort 2.1 intrusion detection (2nd Edition)*. Rockland, MA: Syngress.
- Crothers, Tim. (2003). *Implementing intrusion detection systems: A hands-on guide for securing the network*. Indianapolis, IN: Wiley Publishing, Inc.
- Northcut, Stephen & Novak, Judy. (2002). *Network intrusion detection (2nd ed.)*. Indianapolis, IN: Sams.
- Orebaugh, Angela, Biles, Simon and Babbin, Jacob. (2005). *Snort cookbook*. Sebastopol, CA: O'Reilly.
- Snort Core Team. (n.d.) The Snort FAQ. Retrieved May 22, 2005 from <http://www.snort.org/docs/faq/1Q05/faq.pdf>.
- Snort Project. (April 22, 2005) Snort users manual 2.3.3. Retrieved May 22, 2005 from http://www.snort.org/docs/snort_manual.pdf
- Tokash, Keith. (January 28, 2003) How to setup and secure Snort, MySQL and Acid on FreeBSD 4.7 Release. Retrieved May 22, 2005 from <http://www.snort.org/docs/FreeBSD47RELEASE-Snort-MySQLVer1-3.pdf>