;login:
THE MAGAZINE OF USENIX & SAGE
November 2001 • Volume 26 • Number 7

Special Focus Issue: Security
Guest Editor: Rik Farrow

inside:

INTRUSION DETECTION
Implementing Snort in a Production Environment
by Jon Lasser

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

implementing snort in a production environment

by Jon Lasser

Jon Lasser is a UNIX consultant in Baltimore, Maryland, and is the author of *Think UNIX*. He is lead coordinator for the Bastille Linux Project and thinks the four food groups are coffee, cheese, chocolate, and beer.

jon@lasser.org

Last year, I implemented Snort in a production environment at a medium-sized Web-hosting firm specializing in dedicated servers. We implemented Snort version 1.6 (later upgrading to version 1.7) on a system running Red-Hat Linux version 6.2. After we performed extensive tuning of the Intrusion Detection System (IDS) rule set and wrote a number of scripts to parse its output, the system proved quite useful for identifying denial-of-service (DoS) attacks. It was somewhat less useful for detecting actual intrusions, but even in this regard it was superior to running nothing at all.

Project goals were very vague, but the general idea was to increase the security and manageability of our network. Because we had no direct control over customer servers, we could identify misbehavior only by locating suspicious network traffic. A number of customers also ran IRC servers, making them prime targets for DoS attacks. Our hope was that by implementing an IDS we would be able to quickly identify the targets of DoS attacks. While we hoped to use the system to identify the attackers, addresses were almost always spoofed and the degree of cooperation with our upstream network feed did not lend itself to tracking down the attackers. (Many larger network providers will not even attempt to trace back spoofed packets unless you appear to be serious about prosecuting the attacker. They have limited resources to deploy, so this strategy makes sense from their perspective.)

Our system was a generic Intel-based Pentium III system running at approximately 700 megahertz with a 100-megabit Ethernet card. Since customers were permitted to perform virtually any activity on their dedicated servers, there was no site-wide firewall. The IDS sat on the VLAN used by the routers to talk to each other and sat on the "span" port so as to see all of the relevant traffic. However, the routers talked to each other over gigabit fiber, and traffic averaged approximately 120 megabits per second, so the IDS saw what amounted to a (hopefully) good sample of the traffic. Keeping up with the throughput given by the 100-megabit card kept the CPU at full utilization all of the time. Planned updates to the system included splitting the sensor from the processor so as to allow for expansion and to reduce CPU usage, but this was never implemented.

Linux was chosen as a platform not because of its strong packet-capture abilities but to remain consistent with the other servers on site. Simplicity of management was one goal of the implementation; there was no full-time security person on-site, so managing the IDS was one of a number of duties I had. Because the pcap library does not keep track of dropped packets on Linux, it is impossible to estimate with any accuracy the actual percentage of traffic processed by the server. It was certainly adequate for much of the task at hand. My rough guess is that the IDS saw three-fifths of the traffic. This guess is based primarily on what appear to have been complete sweeps of our network space: approximately three out of five target IP addresses in our range tended to show up in these sweeps. With even faster hardware now available inexpensively, it might be possible to capture all hundred megabits of traffic even without a more complex architecture; as some commercial vendors claim to be doing 200-300 megabits per second of actual traffic, it should be possible to push the systems harder. This may require more extensive engineering and more careful selection of rule sets.

Snort 1.6 and 1.7 built out-of-the-box on RedHat 6.2; newer versions of RedHat Linux have prebuilt Snort 1.7 packages available as part of their "PowerTools" collection of applications, though Snort 1.8 is now available. Snort was chosen due in part to my familiarity with parsing its output and the availability of other administrators in the area who understood and were willing to help with Snort. (Thanks, Andy!) Low cost of implementation was another consideration: the project goals were very vague, so justifying expensive software or hardware was out of the question.

The most sensitive part of the Snort configuration was the rule set used. A Snort rule set is a list of filters that apply to packets and determine which ones set off alarms. Rules can match packets based on source and target addresses and ports, particular protocols or flags, and the actual contents of the packet. The more rules implemented, the slower the system will be, and the more alarms will go off. It is difficult to strike a balance between a "quiet" IDS that misses important incidents and a "loud" IDS that finds such a quantity of suspicious traffic that actual attacks are lost in the noise. Our experience was one of trial-and-error, starting with an extensive rule set and whittling out those rules that became more trouble than they were worth.

A Snort rule set is a list of filters that apply to packets and determine which ones set off alarms. . . many DoS attacks will set off a large number of these rules simultaneously, lighting up the target system like a Christmas tree

We based our rule set on the one made available at *http://www.whitehats.com* with a number of substantial changes. First, because we were as worried about mischief originating from our network as that aimed at our network, we ignored the "internal" and "external" network definitions and modified all rules to consider any source and destination addresses as matching the given rule. We disabled all rules that matched solely on source and target ports; these rules have a very high false-positive rate. On the other hand, many DoS attacks will set off a large number of these rules simultaneously, lighting up the target system like a Christmas tree. Even with these changes, a large number of rules generated many false positives and were disabled. After that, we still had 10,000–20,000 individual alerts generated by the system on a typical day, many of which were false alarms that were too difficult to programmatically eliminate.

I wrote a group of scripts that ran hourly, examining the last full hour's incident logs. One script summarized all of the activity for the hour. It listed all of the rules that were triggered, sorted by the number of times each rule was triggered; it listed the top 10 source addresses of packets that triggered rules; and it listed the top 10 destination addresses of packets that triggered rules. This hourly summary gave me a quick read on the status of my systems over time and allowed me to find serious incidents not detected by the script described below. I also ran a daily summary script that provided the same information on a daily basis, allowing me to keep (essentially useless) statistics of how many detects a day we experienced.

Another script looked for particular alerts that we considered of high importance, such as those matching a particular exploit. The script then collated all attacks from that source IP over that hour – not simply the attacks serious enough to flag an email, but all detects from that source IP address – and mailed me the resulting log excerpt, with some boilerplate text up-front. These log excerpts often read like detailed summaries of scan-attack-scan scripts.

I used procmail to sort all of these messages into a single folder and set my mail reader to sort that folder by subject. Since each subject began with the IP address of the attacking system, it was easy to see attacks that spanned multiple hours. Using whois, it was quite simple to track down a responsible party for any particular server. By using Mutt's ability to act upon groups of messages simultaneously, and some clever vi

Frankly, whois information provided by most users is completely insufficient for accurate security-specific response

scripts to simplify stripping out extraneous information, I was able to quickly respond to a particular incident or group of incidents. However, the volume of alerts was such that I typically spent several hours a day sending "nastygrams" to sites that had scanned our network or attacked servers on it. The only solutions were to be more picky about what attacks and probes required a response or to automate further. Further automation was scheduled for the never-implemented second iteration of the IDS.

Frankly, whois information provided by most users is completely insufficient for accurate security-specific response. The one exception was Brazil's NIC, which included provisions for tagging an individual as the person responsible for security at a given site. I strongly urge the more widespread adoption of this technique, as it would both improve the usefulness of whois as relevant to security and to allow for further automation of security response. In general, the quality of information provided by the various registrars was abysmal: not a day went by without several security messages bounced due to invalid email information, and many more messages were misdirected due to changes in IP space management not reflected in the database. I also urge the use of a standardized "security" email address to simplify site contact for security-critical information.

The rate of response to security nastygrams varied wildly; some days as many as one-third of the sites to whom I sent complaints responded, while other days nobody at all did so. Responses were no more likely to come from places where English is the primary language than from other locations, and, in general, native English speakers were the least polite. However, they were also the most likely to request help in interpreting the logs or to offer to track down rogue server activity.

The most frightening activity picked up by the IDS was "low and slow" probes of our network: at any one time, there were probably five or six different sites involved in such probes. I have no way of knowing if these were coordinated attacks; because we never implemented a database back end for the IDS, I do not know if the regions scanned overlapped. The most obvious (!) of these attacks sent no more than two packets a day, inquiring of the version number of BIND running on a particular system. Even in the best case, it was hard obtaining sufficient evidence of malicious behavior to send to administrators at the appropriate sites: only over a four- or five-day period would enough packets accumulate that I would not feel foolish sending them to the administrator of the machine scanning our network. As I tended to go through the mail on a daily basis, I usually became aware of the low-and-slow probes when coming back from a long weekend. Had I tried to respond to all such attacks more frequently, I could have spent a full day every day doing nothing but processing logs.

As stated above, the most effective use of the IDS was tracking down the targets of DoS attacks. In fact, any packet-capturing system that can provide the source and destination IP addresses of individual packets can be used to do what we did. Simply, we had a script that started a Snort process which was configured to capture a particular number of packets (we used 10,000 packets in most cases) and print out the packet header information. We then extracted the source IP address of the packets, sorted them, and then used uniq to count the instances of any particular address. Finally, we did a reverse sort based on the packet count and printed the top 10 senders of packets.

Some high-volume customers continually ranked in the top 10, but there was usually an order-of-magnitude difference in packet counts when a DoS attack was underway.

Under normal circumstances, the top user might have 500 packets out of that 10,000; during an attack, the top user would likely have 3000–5000 packets. As a result of this, our "top 10" script was our most effective anti-DoS tool, followed closely by MRTG, which we could use to examine the traffic patterns of our top packet senders to determine whether or not the activity we were seeing was typical of that user.

As a means of detecting the actual compromise of servers, the IDS was rather ineffective. As most current exploits do not check for vulnerability before attempting to gain access to a system, we would frequently see exploits attempted against servers running different operating systems or even CPU architectures. Verifying the possible vulnerability of particular system for particular exploits was a Herculean task and, to be frank, boring. Worse, as we did not have root on most of our customers' systems, it was often impossible for us to verify whether or not a particular service was vulnerable on a given system.

This points to two areas that I believe need improvement in intrusion detection. First, I would like to see stateful systems that are not only able to detect a probable attack against a particular system but, following that, would be able to detect the success or failure of the exploit. This would require much more state than most intrusion detection systems maintain, and would likely be quite expensive in terms of memory and CPU usage. Nevertheless, it would allow much better prioritization of incidents, allowing limited resources to be deployed more effectively.

Second, it seems clear that intrusion detection is one small piece of the larger problem of incident response. This is the "what now?" problem: one of the rules has been triggered; what do you do now? This question goes unanswered by Snort and most other IDSes. After detection, the incident needs to be tracked, the system needs to be examined, and the offending system's administrator must be contacted. If the targeted system is not under your direct control, its administrator must be contacted, and so forth. I would like to see intrusion detection and response suites, essentially specialized trouble-ticketing systems, that would allow a group of detects to be classified as an incident and, once ticketed, allow that incident to be tracked. This would probably also include tracking the "owners" of blocks of IP addresses, including accurate contact information. This database could initially be populated via whois but could be refined by the system's users over time.

Overall, implementing a Snort-based IDS could not have been simpler. Were I to do it again, I would definitely use a separate sensor and a database back end: although the text-based logs were easy to process using Perl and shell scripts, the additional power of database queries might have turned up more useful information and saved a lot of time.

What was difficult about implementing an IDS with Snort was sorting the vast quantity of data produced by this system. Judicious selection of alert rules is crucial, as is some sort of automated post-incident processing. For my site, our hourly detect script was the most useful. It found all "serious" alerts and sent an email with all alerts, serious and otherwise, from that source IP. Even this was far overshadowed by the ability of the system to detect DoS attacks. That, however, could have been done on a much simpler system connected to the span port on the router; we simply did not think to use packet captures to detect DoS attacks until we had already implemented the IDS. (Before the IDS, we simply used MRTG to find spikes in network load on particular ports.)

This is the "what now?" problem: one of the rules has been triggered; what do you do now?

Unless a site has a specific need or a full-time security person on staff, I cannot yet recommend implementing an intrusion detection system. Although simple to implement, the data produced by such a system tends to be useful only to the extremely paranoid or those without adequate control of systems on the network in question. Over time, however, I expect that intrusion detection systems will become more powerful and more useful for the typical system administrator. Right now, however, these systems are too good at producing high volumes of data and not good enough at providing tools for turning that data into information.