Automating UNIX Log Monitoring

Miguel Rodriguez: miguelr@cs.ucr.edu: University of California, Riverside

Abstract

We present a methodology for the maintenance and use of UNIX log files with the Perl scripting language. We begin with defining UNIX logs then continue with our motivations and goals. Other applications exist for these purposes. We compare these with our own and discuss any differences. The technical details, along with the motivations, of our approach are then explained. Our results show that a systematic approach to the handling of UNIX files is essential for successful system administration.
Keywords: UNIX, logs, Perl, scripting, SWATCH.

Introduction

UNIX logs are an integral part of a UNIX system. They record events that occur, each with a different priority. Some may be errors, warnings, emergencies, or they may simply be a notice or message of what has occurred. It is important to note that these logs record what *has* happened and not what is or what will be happening. However, it is possible to notice trends. An experienced System Administrator should be able to anticipate what will happen based on the current entries in the log files.

These logs can be useful for troubleshooting and diagnosing problems, such as network failures, SPAM, unauthorized access, exceeded disk space, etc. They can also be used to monitor the activity of authorized users and enforce any policies or rules that are in place. The benefits are numerous. To further exemplify the importance of UNIX log files, consider the nonexistence of these files. How would a System Administrator monitor their system? Assume that the administrator does indeed care to see their system in good "health." To monitor activity, he or she would have to perform the logging manually...for everything! Every service, every daemon, every request, etc. This will get very tiresome, very quickly. Perhaps to decrease the workload, say that a few more administrators were hired. This would make things slightly better for the first administrator, however this would increase the cost for their employer. As a result, more man power is clearly not the solution to the tediousness of manual logging. So we return to having a single administrator. Now consider the amount of time required to perform the logging. Assuming the system is always online, our poor administrator will never be done with the logging! There will always be an event to log. Additionally, note that during the time required to create a log entry, there may have occurred many other events that require logging, which in effect, create a log queue. Therefore, we can see that the amount of time required for manual logging increases indefinitely (as does the log queue). Thus, we conclude that manual logging is not only impractical, it is also infeasible.

Of course, this is not situation that exists in reality. UNIX logs implement automatic logging, which relieves the System Administrator of this tedious task and allows them to focus on other important duties. Problem solved? Not quite. We still have the next task after the logging is done, which is to look at them and identify anything that might be of concern. First we must understand where these log files are kept. Without going into detail describing the UNIX System Log Facility (1), we will simply point out the most logs are stored in the directory /var/log. There are many log files that serve different purposes and services. Furthermore, they do not all conform to the same format. This and their sheer size contribute to the dullness of having to sift through them by hand. The amount of time required for this is also significant. Again, this leaves less time for other, possibly more important, tasks that must be performed. As we can see, the monitoring of log files is also tedious. The solution is to somehow automate this log monitoring process, which was the solution for the manual logging problem.

Our project aims to solve the problem of manual UNIX log monitoring. Such a solution would greatly increase a System Administrator's ability to serve other tasks and personnel. It also cuts the cost for their employer because it eliminates the need for additional administrators. As an overview, their are two steps to our project. The first and most important is to research and identify the types of situations and events that an administrator would like to monitor. The first step would be to identify an even of interest, view the appropriate log file and take note of the types of entries and their meanings. If their meaning is not initially clear, take steps to learn what information the entry is recording. The second and easier step is to write a script or scripts that extract(s) the desired information and reports some statistics from this information. Step one is

a prerequisite for step two. In truth, there is actually a third step to this process, but it is not strictly a part of our project. The third step is to take any action if necessary. This is left as a responsibility for the administrator and our goal is to make this responsibility easy to fulfill.

Related Work

As mentioned earlier, their are similar projects that address the problem of manual log monitoring. Simple WATCHer, or SWATCH (2), is one of the more notable projects. It is a Perl script that monitors a specified log file with a configuration file that defines what patterns to look for and what actions to execute when a pattern is found. This is the first difference between SWATCH and our script. Our script does not execute any actions like SWATCH does. We simply report information and statistics. The actions are assumed to be taken by the administrator. Another difference is SWATCH's configuration file and our lack of such a file. It's configuration file is highly flexible. The patterns are defined by regular expressions, which are extremely powerful in describing sets of strings or patterns. The pattern is then followed by the action that is to be performed when pattern is matched. The following is an example entry: watchfor /failed/ echo bold mail addressess=root,subject=Failed Authentication Given a log file, say sulog, SWATCH will try to match the string "failed" in each line in the file. When it finds a match, it will send an email to root. Observe that, given the pattern and input file (sulog), this implies that we are interested in all the failed authentication attempts (also indicated by the subject field). This demonstrates the first step in our process, which is to identify and understand the event of interest. Step two is to use a script, in this case, SWATCH to automate the log monitoring. The more impressive feature about SWATCH is that it also does the third step - take action. This makes SWATCH a good tool for the administrator. It automates log monitoring and any necessary actions that must be performed. We are still left with step one, though, which is to identify what we are interested in. This is, inherently, a problem suited for humans to solve...at least until we see further advancements in artificial intelligence. Nevertheless, SWATCH does a good job of automating both log monitoring and the actions required, which leaves the administrator with only having to decide what to look for. The bulk of our efforts focus on this aspect.

Technical Details

To solve the problem of manual UNIX log monitoring, we first broke it up into subproblems. The first being to obtain useful and realistic data in the form of log files. We first considered creating dummy logs but found it to take a significant amount of time. We looked at the logs on our Linux box, but discovered them to be of little use. First of all, there is only one user on this box and second of all, the file sizes are too small to be useful. Another reason why dummy logs would be impractical is because we would have to learn all the formats that log files take and we would have to learn the meaning of the content of each entry in each file. Additionally, suppose we introduced some invalid entries, that is, log entries that would otherwise never occur. This would clearly invalidate any of our results and conclusions. As a result, we endeavored to obtain large amounts of real data.

We were able to obtain several log files from the Graduate Sun Lab in the Department of Statistics. The two that we will consider are sulog and authlog with sizes of 32K and 4.6M, respectively. The file sulog logs all successful and unsuccessful attempts of the su command and the file authlog logs all authentication attempts. Clearly, we could very well use any or all the logs for our project, but for the sake of brevity, we will only consider these two log files. The methods we describe can similarly be applied to any log file of interest.

The next subproblem is to identify the patterns or events that we wish to look for in these log files. This quickly becomes the most time consuming part. To learn what to look for, we first must view the files. Immediately, one would be overwhelmed by the size of the file. Where should we start? Starting and the beginning of the file is essentially pointless, depending on the life span of the file. If it has many months worth of data, the entries at the beginning would be of little use for us at the present. At the very least, this would tell us when a problem began to occur. A better place to start perusing the file would be at the end. This is the closest point to the present. It tells us what recently happened (good or bad). From here, we can work backwards and if we were to find a problem, we could trace backwards to find where the problem began. This is the process we will adhere to.

The entries in sulog contain the following information: the command su, date, time, whether or not the attempt was successful, denoted by a "+" or "-", port, the user, and the new user the user attempted to su to. An example entry looks like: SU 04/28 15:09 + pts/3 mrodr008-root. For simplicity, we only consider the entries in which a user attempts to su to root since this is the most important user id and the one in which requires the most security. This is a rather simple log to understand, which is why we will begin with it. We

have identified what we wish to look for and our next step is to write a Perl script that will do this task for us. For the System Administrator, this should be no problem. We omit the details of the Perl script and refer the interested reader to the references for resources on Perl (3). In our script, we take the sulog as input and output the users who successfully and unsuccessfully su'd to root, along with their count. To do this, we test for a match for the string "root" at the end of each line in the file. We also output a time distribution which tells us the hour when attempts to su root occurred. This distribution spans all the users who made such attempts. If desired, one could instead output a time distribution for each user.

In contrast to sulog, the file authlog contains a larger variety of log entries. As a result, a lot of time was spent researching their meanings. Again, for simplicity, we continue our quest to detect any unauthorized root logins. In this case, we are concerned with ssh attempts to root. In our script, we check for the following log entries: May 27 14:24:05 s_local@statserv.ucr.edu sshd[23991]: [ID 800047 auth.info] Failed password for root from 138.23.5.156 port 53250 ssh2. To achieve this, we use Perl's regular expression capability and try to match the string "Failed password for root". For each match, we then extract the IP address and run the program "traceroute" on it to find out where the attempt originated from. Alternatively, you could instead use the program "whois", which would give more detailed information. Then one would parse out the location. At this point, we are at step three now - take action. It is now the administrator's responsibility to decide whether these ssh to root attempts are malicious or not and take the appropriate action.

Results

We now present the results of our methods, which we will use to support our claims. After obtaining the log data and identifying our interests (looking for root access attempts), we wrote the appropriate Perl scripts and then put them to work. Each file was not modified in any way. From sulog, we obtained the following information:

Successful su root attempts

billdbrk	77
gaston	73
ggonzale	286
mikek	1
mrodr008	8
njames	123
root	2
russ	1
sgeadmin	1

Unsuccessful su root attempts

atristan	4
billdbrk	143
gaston	15
ggonzale	84
mikek	1
mrodr008	2
njames	35
russ	3

Time Distribution

00	5
01	6
02	3
03	1
04	0
05	0
06	1
07	11
08	33
09	69
10	74
11	75
12	108
13	106
14	99
15	111
16	60
17	39
18	21
19	9
20	10
21	7
22	1
23	10

Upon inspection, we find that all of these user id's are known to us except for "njames". The others we know to be either administrators of Statistics or other employees of UCR C&C. It may be that "njames" is also a part of C&C. We would have to look into it.

From the tables, we see that there have been no unauthorized root access attempts. This is good to see. There also appears to be nothing too unusual in the access times and most of them took place after lunch, with some Night Owls as well.

From authlog, we obtained the following:

IPs that failed to ssh to root

87.194.33.69	London city residential static 2 service
203.24.211.3	WebCafe-AU
168.167.21.90	Botswana Telecommunications Corporation
138.23.5.156	UCR Computing & Communications

Based on this information, three of the four IPs are, in all likelihood, unauthorized ssh to root attempts. The fourth one is nothing to worry about...unless we have corrupt employees. Next we have to decide what action to take. We could do nothing, since the bad IPs were unsuccessful in their attempts and assume that this will continue to be the case. Or we could contact these organizations from the information provided in the output of the "whois" command. However, we expect this to do little good. If they indeed have malicious intent, would they really reply back? Not likely. A better action would be to further strengthen the root password because we anticipate these types of events to re-occur.

Conclusion

In this paper, we have presented and discussed the motivation for a systematic procedure for monitoring and analyzing UNIX log files. We first showed how the UNIX System Log Facility eliminates the need to perform manual logging. Then we discussed how manual log monitoring is very time consuming for a System Administrator to perform, thus demonstrating the need for an automatic alternative. Our method consists of two steps, followed by an assumed third step to be performed by the administrator. First, after gathering the desired data, identify what events or trends to look for. Second, write scripts that look for the patterns of interest and report the results and statistics. The third step is to perform the appropriate action in response to the results of the analysis. System administrators shall benefit from these procedures in many ways. First, they will obtain more time to perform other important system administration tasks, such as performing hardware or software upgrades, updating policies, or dealing with his or her users in person, etc. Another benefit is the faster way in which they will learn of any problems occurring on their system. This would help to minimize any down time that would occur. If we had to sum up our project in one sentence, it would be: "Look at your logs, learn what to look for, respond accordingly."

To further improve on our results and methods, we will implement more advanced functionality in our scripts. As noted before, it is rather simple to tailor our scripts to work with the many other UNIX logs that are available. Ideally, we would want them to work with all UNIX logs. Furthermore, setting up our scripts as a cron job would be even more beneficial as it would let the administrator not have to initialize the log monitoring processes manually. We refer to this as "true" automatic log monitoring. Thus, we can see that the process of automating UNIX log monitoring is multifaceted. Current methods, together with ours, allows for any administrator to efficiently perform their job.

References

[1] Garfinkel, S., and Spafford, G. Practical UNIX & Internet Security. *2nd Edition* (1996), Ch. 10.5.

[2] Atkins, T. SWATCH homepage http://swatch.sourceforge.net SWATCH man page http://linux.die.net/man/1/swatch *Version 3.1* (2004)

[3] Wall, L., Christiansen, T., and Orwant, J. Programming Perl. *3rd Edition* (2000)