AN APPROACH TO BUILDING A FORTIFIED NETWORK LOGGER THAT IS RESILIENT TO CRACKING ATTEMPTS

Jaco Kroon **Martin S Olivier**

Information and Security Architectures Research Group
Department of Computer Science, University of Pretoria, Pretoria

ABSTRACT

Keeping logs secure and untampered with has always been a problem. The first things a cracker goes after once a system has been compromised are the logs[9]. They do this in order to cover their tracks and to hide their presence.

The preferred current solution the use of remote logging [14, p 372]. Even though this increases security of your log files, there is still the chance that a skilled cracker will be able to break into your log server, and tamper with your logs.

A simple way to effectively reduce the chances that a computer can be cracked is by cutting the transmit wires on the communication medium. This effectively prohibits TCP connections and reduces the device to a receiving only station. This further reduces the chances of this computer being hacked simply because there is no way to get feedback from it, or even detect it.

Private networks, with no routing into or out of the network, are an effective way of establishing secure and reliable communication between a set of stations. In order to gain access to the network a station on the perimeter has to be compromised before access to the private network can be gained.

This paper proposes an alternative manner in which to perform logging. It makes use of a silent log server, preferably inside a private network. A dummy log server can also be used inside this private network to heighten security by hiding the fact that you are using a silent logger. [1]

[1] This material is based upon work supported by the National Research Foundation under Grant number 2054024 as well as by Telkom and IST through THRIP. Any opinion, findings and conclusions or recommendations expressed in this material are those of the author(s) and therefore the NRF, Telkom and IST do not accept any liability thereto.

1 INTRODUCTION

Logs are used to record actions that take place in a computer environment. These then serve as an audit trail that can be used to hold users accountable for their actions or to help in criminal investigations.

Traditionally logs were stored on the local system, relying on local system integrity for security. It is well known that crackers attempt to alter the log files as soon as a host has been compromised, therefore logs should ideally not be stored on a local system.

The obvious solution to the problem caused by storing logs on the local machine is to store logs on another computer (dedicated to the task of storing logs) connected to the network. This enhances log security by keeping the logs out of reach of an attacker. Unfortunately such a dedicated host can still be compromised.

There are further enhancements to this scheme that attempt to ensure log integrity[13]. Message authentication codes, for example, are used to ensure that no other host can spoof log messages, and firewalls prevent unwanted network communication and make compromise harder. Reducing the number of services running on a server also improves security, therefore a log server should only be running a log service, if at all possible. A network dedicated to the task of logging can prevent attackers from directly attacking the log server. However, none of these schemes rule out the possibility of a compromise of the log server. If your log server gets cracked and the logs deleted, log integrity does not help at all.

While these schemes do improve the overall reliability of logging it also introduces unwanted complexity, thereby reducing trust in the system.

This paper will propose an alternative solution that is reliable, secure and easy to implement. We will show how to create a logging facility that is highly resilient to tampering. We rely on existing technologies such as network logging (for saving log entries from the server to a dedicated machine) and one way communication (at a physical level).

Section 2 will provide some background information. Our proposal will be stated in section 3. Section 4 will take a look at how to create a silent host. In section 5 we will explain how to log to such a host whilst in section 6 we will explain how to create a dummy host. A few final words are made in section 7, the conclusion.

2 BACKGROUND

Logs have been generated for various purposes over time, from replaying events [18] (for debugging purposes) right through to security [15]. Others include statistics - usage patterns in particular.

What gets logged depends on the reason for logging. For example, if one desires statistical information it might be more important to know what services people are accessing, whilst if one are looking at logging from a security point of view, it might be more important to log who is accessing what data, or to look for suspicious activity. Also, if one is merely logging statistical data it might not be as important to obtain every single bit of data. Thus, it might be acceptable to cache log data for a short while to improve disk access patterns. When logging for security

reasons, we do not want to miss any entries, and we definitely do not want dropped entries due to the fact that they have been stored in a temporary cache when a crash occurred.

Analysing logs is also a cumbersome process. Various tools such as analog [1] and logwatch [3] have been developed in order to monitor logs. Other standard tools, such as text editors and other text manipulation tools could be used to perform manual analysis of log files.

In the case where this information is used for security auditing, it is particularly important to keep this information as secure as possible. In many cases simply making it only accessible to the root user, or another restricted user, is good enough (for statistical purposes). In other cases a dedicated host for storing logs can be used effectively, providing that such a host is hard to compromise.

An interesting, and viable, alternative is to log to a line printer. This has perfect integrity as well as confidentiality, permitting physical security is properly implemented. It cannot be altered or tampered with remotely. The printer can however run out of paper or ink. The fact that a computer cannot help with performing queries is problematic. Normally computers can be used to quickly and effectively filter through huge amounts of text for specific phrases or patterns. This especially becomes an issue when one is conducting an investigation. It has the further disadvantage of wasting huge amounts of paper and ink. The possibility that a line printer cannot keep up with the task of printing log lines also exists, and for this reason logging to this media should be restricted as much as possible.

A serial line could also be used to log to another machine, which is a very reliable approach. We do not like this approach since it requires us to set up a dedicated cable between the log server and each machine from which we would like to log. Furthermore, it is rather difficult to log to more than one log server – which one would like to do for the purposes of redundancy. There is also a lot of overhead involved when adding another log client. Since serial lines are slower than parallel lines there should be a cache on the client side, resulting in the possibility of data being lost due to caching.

Various packages to perform logging for the Linux operating system exist. Arguably the best known logger is syslog [7], which works along with sysklogd [6]. Not only can these packages only filter based on the log facility and log level, they are also hard to configure. Other packages include metalog [4], which is quite good when it comes to logging on the local system. In our opinion metalog would be one of the best loggers around if it were not for the fact that it does not support network logging. For this reason, we have decided to use syslog-ng [8] instead. It supports network logging, using either UDP or TCP, as required. It also has very powerful filtering features based on regular expressions, which fall outside the scope of this paper. The remainder of this article will assume the use of the Linux operating system and the syslog-ng package as the logging system. The choice of Linux is merely due to familiarity and concepts apply equally to other operating systems.

3 OUR SOLUTION

We would like to propose that logging to a silent host might be a viable alternative, solving most of the problems with existing methods mentioned above. Such a host would be a normal

computer connected to the same network as the log clients. It has all the benefits remote logging has. In addition it has a very high degree of security, even though availability can never be guaranteed (there is always the possibility of a system crash). Such a host would not be able to generate network traffic. This should make it impossible to detect such a host on the network. Even should such a host be detectable, it is near impossible to attack – it might be possible to launch a denial of service attack on such a host, but there is no way of verifying the success of such an attack. Network architecture also plays an important role in preventing denial of service attacks on our logging system, as will be seen in section 5.1.

The strength of such a host is also its weakness. It is useful for administrators to be able to log in on hosts remotely. The reasons for this are manyfold, mainly the ability to perform administration remotely and the ability to have hosts automatically notify the administrator of certain events. It is also common practice to configure hosts in such a manner that they monitor each other to detect when the other goes down and to notify the administrator. For these reasons, and many others, we recommend adding a dummy log server to the configuration.

A dummy server doubles as a visible log server. An administrator can use this host to check out logs remotely, and in real time. It also adds to the overall security by obscuring the fact that one is using a silent log server. It should be noted that this does not solve the problem of remote administration of the log server. Such a dummy log server can very easily double as one's intrusion detection system – it still has the ability to send a notification off the host to the administrator. As can be seen here we already obtain the ability to log to more than one host without much additional overhead.

The advantages of logging to multiple hosts are not immediately obvious. This is one approach to improve availability – the chances of all hosts crashing at exactly the same moment are minuscule. It does not rule out the possibility of an attacker sending a specially crafted packet to disable the log servers all at once. Careful code review would be required to verify that no vulnerabilities exist in the log software or in the parts of the kernel that is involved in retrieving and passing those packets to the log software. The same argument can be applied to hardware failures.

The scalability of network logging is much better than that of line printers or serial lines. The reason for this is purely the amount of available bandwidth. Our solution still maintains the advantages of these solutions, namely that of integrity and confidentiality. Chances are, however, that line printers will run out of ink or paper at some point or another; the chances of a dedicated log server running out of hard drive space is much smaller, and even should it do so we could simply start rotating logs and discarding old logs. The main advantage over serial lines is increased bandwidth.

Thus the solution of logging to a silent host has all of the advantages of existing logging methods, but without many of the disadvantages.

4 CREATING A SILENT HOST

A silent host is a host that can receive network traffic, but cannot generate any network traffic on the network device connected to the network. This network device will be referred to as the

external network interface for the remainder of this discussion. One way to create such a host would be to install a firewall that blocks all outgoing traffic. Now the question arises of whether that is truly silent? A protocol that can be easily overlooked is the address resolution protocol (ARP) which maps logical network addresses (such as IP addresses) to physical addresses (such as MAC addresses).

A very simple silent host can be created by making sure that the kernel only supports IP, has support for iptables (part of the netfilter [5] project) and arptables (part of the ebtables [2] project). Then it is a matter of dropping all outgoing ARP packets, both requests and responses, as well as all outgoing IP traffic on external network interfaces. (We still want full communication on the local loopback device.) The following three commands will achieve our goal of denying transmitting of data to a relatively high degree of certainty:

```
# arptables -P OUTPUT DROP
# iptables -P OUTPUT DROP
# iptables -A OUTPUT -o lo -j ACCEPT
```

This prevents the host from transmitting any packets. As is the case with any complex software, there exists the possibility of a bug. In this case it might allow an attacker to bypass the netfilter rules, permitting bidirectional communication. A simpler, and more reliable solution is called for.

We would prefer to go below the network layer on the ISO OSI model [10, p 19]. This leaves us with the data link layer and the physical layer. The data link layer potentially has the same problems as described for the network layer, if one wants to guarantee silence. There is no obvious way of restricting network communication on the data link layer, as such the only truly viable option is to take a look at the physical layer.

If the network cabling only allows communication in one direction, it is impossible for a remote attacker to force communication in the other direction. A UTP CAT5 cable is wired with 4 pairs of wires [17, p 232]. We assume the use of the T568A wiring scheme. Ethernet 100Tx only makes use of two of these pairs. On the host side, pins 1 and 2 form the transmit pair (the orange wires) and pins 3 and 6 the receive pair (the green wires).

A first attempt at creating a one way communications channel involved simply cutting the transmit wires. This had the unfortunate side effect of letting the link status drop on the hub side, causing the hub to not transmit data to the host. This is due to the hub's inability to detect the network device (there is no signal on the Tx wires), which then causes it to not transmit on that port. Simply cutting the transmit wires does not work.

A further idea (as illustrated in figure 1) makes use of a resistor and a capacitor to scramble any transmitted data [12]. The idea is to cause CRC errors [11] in the transmission stream. We have not tested this; our feeling is that this might be detectable by other hosts on the network segment.

Another idea is based on grounding the transmit wires on the hub side of the laceration to the receive wires [11] (as illustrated in figure 2). Our attempt at doing this did not work. It caused every single packet transmitted on the network to be dropped due to carrier errors. The reason for this might be "extremely poor electromagnetic performance" [11, p 1]. The reason for this could well be the fact that our cables were not of a high quality, nor were our connection points

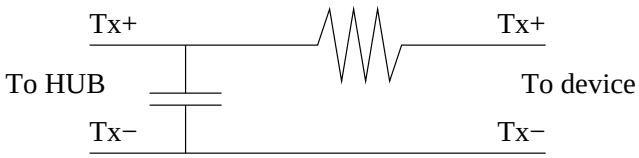

Figure 1: Scrambling data on the transmission lines

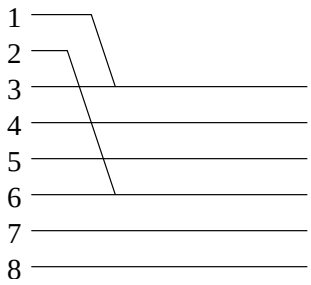

Figure 2: Wiring diagram for grounding Tx on Rx

very clean. It could also be due to duplex issues (The HUB may think that the host is transmitting whilst it is in fact receiving it's own transmission).

After noticing that the link status only dropped on the hub side, another idea came to the fore. By simply splitting the transmit wires away from the receive wires, and taking them to different destinations one ends up with two network devices, one being capable of sending and the other of receiving (as illustrated in figure 3). The receive wires go to the silent host while the transmit wires are connected to another network card on another machine, or to another hub (unused except for the purpose of providing the first hub with a carrier wave). This causes the original hub to see a carrier wave, link, and transmit the data to our silent host as intended. We tested this using a second hub. This has the unfortunate disadvantage of requiring an extra network device. If one is going to use a dummy host, it is possible to place an extra network card into that machine and use it for this purpose. Such a second network card should not be activated by the kernel.

Preventing communication on a physical level is thus relatively easy, very reliable and trustworthy – if there is no media to transmit across, there is simply no way of transmitting. A perfectly silent host.

5 LOGGING TO A SILENT HOST

5.1 Network architecture

It should be noted that one's network architecture plays a big role in protecting one's assets. Since one would like to make one's log server highly resilient and well protected it might be worth it to consider a network dedicated to the task of logging. This idea is illustrated in figure 4. As can be seen, each of the log clients has two network interfaces, one connected to the log network and one connected to the external network, which is assumed to be connected to the

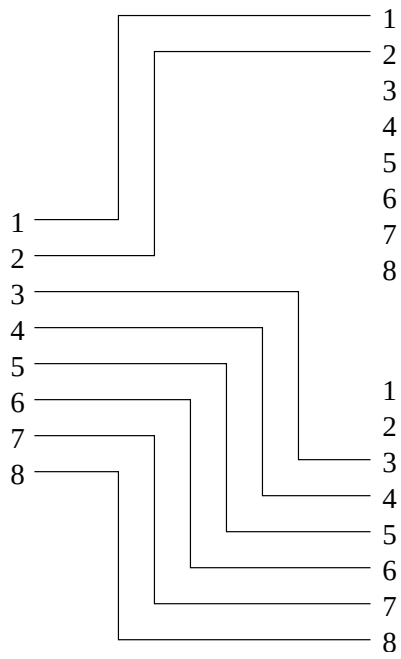

Figure 3: Splitting Tx and Rx to different devices

Internet somehow. Both log servers have one network card each. In addition the silent log server can only receive data, while the dummy log server has two-way communication. This serves to mislead potential attackers, as well as allow an administrator to perform remote reading of log files.

One's network architecture, however, does not affect how the silent log server functions. The only really important thing concerning the log server is that it actually receives all packets intended for it. If possible it should also be protected from spoofed packets coming from other hosts. This is easier to guarantee with a dedicated network but can also be achieved by using iptables on the log server. One would prefer to use private IP ranges [16] for one's log servers, which makes spoof attacks from outside one's organization harder.

5.2 Establishing communication with one's silent host

In order to send data to one's host one need to be able to do address resolution. This can be achieved by adding an entry to `/etc/ethers` on the client, for example:

```
00:02:B3:2B:20:64       192.168.0.3
```

This will tell the host that the physical address for 192.168.0.3 is 00:02:B3:2B:20:64, and will cause the host to never perform ARP lookups for that host. This will prevent the client host from thinking that the destination host is unreachable. The MAC address specified here needs to match up with the MAC address on your server host's network card. This file can be loaded by executing `arp -f /etc/ethers`. This file gets loaded automatically as part of the startup process during boot.

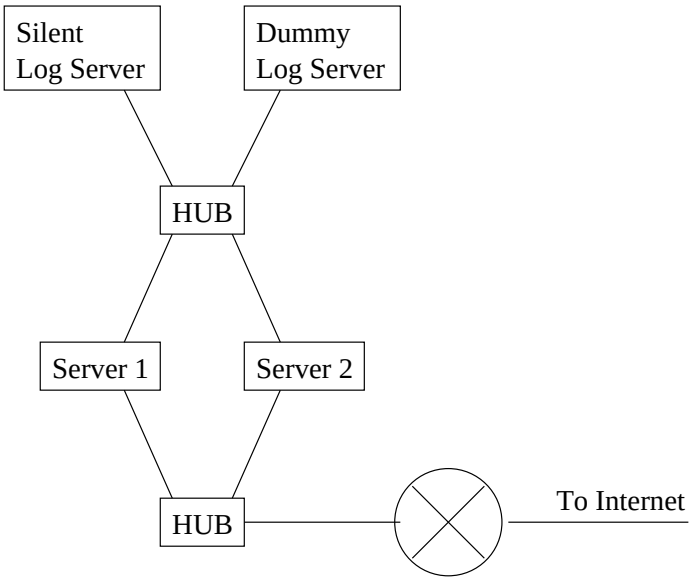

Figure 4: Using a dedicated network for logging

5.3 Configuring the network devices

There are two ways in which this can be approached. The external interface can be configured with an IP address, or it can be brought up into promiscuous mode without an IP address, with an IP alias assigned to the local loopback device. The first of these methods is simpler and does not require any additional software.

By configuring your external interface to have an IP address you will be able to configure syslog-ng in the normal manner (with the restriction of only being able to use UDP packets - as discussed in section 5.4). This has the disadvantage of picking up all packets addressed to this host. This configuration also does not make provision for a dummy log server. It has the further disadvantage of requiring explicit firewalling.

An alternative method is to only activate the external interface without assigning it an IP address and then sniffing for the required data. The software we wrote for this does, however, require that the destination IP address be reachable and that we can send packets to it. In this case, it is also not required that the MAC address in `/etc/ethers` on the clients match the actual MAC address of the log server. The following sequence of commands can be used to achieve this (assuming that your external interface is eth0):

```
# ifconfig eth0 up
# ifconfig eth0 promisc
# ifconfig lo:0 inet 192.168.0.3 netmask 255.255.255.255
```

5.4 Configuring syslog-ng

In this section we will provide a rather minimalistic syslog-ng configuration. The configuration files for syslog-ng is well documented in the manual pages and we do not want to cloud the issue

by introducing unnecessary complexity.

5.4.1 The client

The client can already implement some filters as required. The important configuration lines are however those creating the destination and connecting the source filters and destinations together. As an example, the following three lines will create a source that catches everything and logs it to the remote logger:

```
source src { unix-stream("/dev/log"); internal();
    pipe("/proc/kmsg"); };
destination remote_log { udp("192.168.0.3" port(514)); };
log { source(src); destination(remote_log); }
```

This declares a *src* source that collects all log messages generated by the system. It also declares a destination *remote_log* that represents a UDP socket that transmits all generated messages to the destination IP address as specified. It then proceeds to log all messages collected by *src* to *remote_log*.

We considered using a broadcast address such as `192.168.0.255`, but it seems that syslog-ng does not properly handle this on the server side. This would get rid of the `/etc/ethers` entry on the clients as packets would be broadcast anyway. It will also make it much harder to locate the log server, although arp scans will still detect dummy hosts.

5.4.2 The server

On the server it is simply required to add a UDP source, and a destination (or one of the existing ones can be used as below), and then map the source to the destination. Again, some filters can be applied; here it might be of interest to split different hosts and facilities and so forth to different files. The basic configuration, which will simply log all incoming messages to the existing /var/log/messages file looks as follows:

```
source src { unix-stream("/dev/log"); internal();
    pipe("/proc/kmsg"); };
source rem { udp(); };
destination messages { file("/var/log/messages"); };
destination console_all { file("/dev/tty12"); };
log { source(src); source(rem); destination(messages); };
log { source(src); source(rem); destination(console_all); };
```

This is similar to the client, except that instead of logging to a UDP socket, we instead read from one. The same *src* source is again used to log events on the local system. The *rem* source receives events. Everything gets logged to both `/var/log/messages` and to `/dev/tty12`. This allows us to switch to terminal 12 and view the logs there in real time.

5.5 Using snifflog

If the sniffing method is used to capture log data, then logging will not work yet. This will be the case if one has configured one's external interface to be in promiscuous mode with-

out an IP address. We wrote a sniffer for this purpose, the source code of which is available at `http://www.kroon.co.za/snifflog.php`. Instructions on compiling and running it are provided in the README file. The program will place the external interface device in promiscuous mode - if it is not already in promiscuous mode - and start sniffing for packets destined to the IP address as defined in the source, to port 514 (the syslog-ng UDP port). It will then proceed to simply re-inject a verbatim copy of every packet into the local IP stack. This requires that the destination IP address specified in the packet be reachable, thus the requirement of the IP alias on the loopback device.

6 ADDING A DUMMY LOG SERVER

A dummy log server can be added by configuring a silent logger with no IP address on the external interface but with the IP address assigned as an alias to the local loopback device. For the dummy host the same IP address is assigned to the external interface. The syslog-ng configuration is then duplicated so that both the silent and the dummy log server have the same configuration. The media used by the dummy log server allows for two-way network communication. This concludes the configuration on the server side. It is important that no arptable rules are duplicated. This does not hold for iptables rules, which may well improve the security of the dummy log server. A minor change needs to be made on the client side, the MAC address in `/etc/ethers` is no longer required. We strongly recommend leaving it there, but with the MAC address of the dummy host replaced into it instead. This prevents ICMP (Internet Control Message Protocol) destination host unreachable messages from breaking your logging system should the dummy host go down.

7 CONCLUSION

We have set out to build a logging system that is highly resilient to network cracking at a physical level. Our proposed solution to this problem solves the problems of many other secure logging systems by using various known techniques such as one-way communication and remote logging. The possibilities of failure has not been covered.

Failure could potentially be caused by hardware or software failure, resulting in system crashes. This can probably be overcome by using redundant log servers. Some system still needs to be implemented to make sure that system administrators become aware of failure as soon as possible. Other problems, such as spoofed log messages can, potentially, become an issue; specialized firewall rules can control this to a certain degree.

Another problem includes log overruns. Different logging systems handle this in different ways, some just stop logging with an “out of disk space” error, others start overwriting log entries. Crackers may exploit this to ensure that their actions do not get logged. A possible solution to this might be to have multiple silent hosts implementing different policies, in which case both conditions will be handled correctly.

The biggest risk involved in our solution is that of system crashes. It has most, if not all, of the advantages of serial lines, line printers and remote logging.

In the worst case, our solution will perform as least as well as any other solution.

References

[1] analog. Web site. http://www.analog.cx/.

[2] ebtables. Web site. http://ebtables.sourceforge.net/.

[3] logwatch. Web site. http://www.logwatch.org.

[4] metalog. Web site. http://metalog.sourceforge.net/.

[5] netfilter. Web site. http://www.netfilter.org/.

[6] sysklogd. Web site. http://www.infodrom.org/projects/sysklogd/.

[7] syslog. Web site. http://www.syslog.org/.

[8] syslog-ng. Web site. http://www.balabit.com/products/syslog_ng/.

[9] J. Aslam, S. Bratus, D. Kotz, R. Peterson, D. Rus, and B. Tofel. The kerf toolkit for intrusion analysis. In *Proceedings of the 2003 IEEE*, page 302, West Point, NY USA, 2003.

[10] B. A. Forouzan. *TCP/IP Protocol Suite, Second Edition*. McGraw-Hill, 2003.

[11] P. Gray. Making a one-way cat5 cable for ids deployments. Whitepaper, April 2002. http://sentinelsecurity.net/whitepapers/OneWayCable.pdf.

[12] P. Greeff. Personal Communication, April 2004.

[13] L. Jiqiang, H. Zhen, and L. Zengwei. Secure audit logs server to support computer forensics in criminal investigations. In *2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering*, pages 180–183, Beijing, CHINA, 2002.

[14] K. Mandia and C. Prosise. *Incident Response: Investigating Computer Crime*. Osborne/McGraw-Hill, 2001.

[15] J. McCray. A roadmap to becoming security conscious. In *Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society*, pages 1–9, 2003.

[16] Y. Rekhter, C. Systems, B. Moskowitz, C. Corp., D. Karrenberg, R. NCC, G. De Groot, E. Lear, and I. Silicon Graphics. Address allocation for private internets. RFC 1918, IETF, February 1996.

[17] C. E. Spurgeon. *Ethernet, The Difinitive Guide*. O'Reilly, 2000.

[18] M. Xu, R. Bodik, and M. Hull. A ”flight data recorder” for enabling full-system multiprocessor deterministic replay. In *Computer Architecture, 2003. Proceedings. 30th Annual Internation Symposium on*, pages 122–133, Madison, USA, 2003.

J. Kroon and M. S. Olivier, “An approach to build a fortified network logger that is resilient to cracking attempts,” in *Proceedings of the Fourth Annual Information Security South Africa Conference (ISSA2004)*, Midrand, South Africa, June/July 2004. Work in progress paper, published electronically.