Proceedings of the 4th WSEAS Int. Conf. on Information Security, Communications and Computers, Tenerife, Spain, December 16-18, 2005 (pp201-206)

A Novel Approach to Secured and Central Logging Data

NGUYEN ANH QUYNH, YOSHIYASU TAKEFUJI
Graduate School of Media and Governance
Keio University
5322 Endoh, Fujisawa, 252-8520
JAPAN

Abstract: Logging data is valuable and important information to reveal the attacker's activities and recover broken system. Unfortunately, once the attacker successfully penetrates a protected system, he never fails to either modify the logging data, or even worse, delete them to cover his traces. To avoid such a tragedy, it is best to keep logging data in another machine by forwarding them to a central logging server. However, this approach has a flaw: while transmitting on network, data could be illegally sniffed or the traffic might be secretly redirected to a malicious machine. This paper proposes a novel method named *Xenlog* to secure logging data for systems run on Xen virtual machine: the solution does not use network stack to send data. Experimental and resulted tool proves that this approach is more secure than the traditional solution, while logging process is far more effective (nearly 24 times faster) and more reliable.

Key-Words: Xen, Linux, secure logging, central logging, security attack, forensic analysis

1 Introduction

Nowadays intrusion detection systems are widely used to detect security attacks, but still the most hardened systems are penetrated. Once the attacker breaks in, he will try to steal information, corrupt valuable data and install Trojans, rootkits. Last but not least, the wise attacker will never fail to cover his traces, and logging data is the first priority then. The reason is obvious: system logging data might has important information to disclose the attacker's activities, so he will attempt to either modify the logging data, or even delete all the information which might be used to catch him.

Without correct, accurate and complete logging information, forensic process will be much harder, or even impossible. That is why keeping and secure logging data is one of the most important things the administrator must do.

Basically, there are two approaches to secure the logging data, declared below. Unfortunately they all have unpleasant flaws:
- The logging data is either encrypted or signed before storing locally, like what Shamhain [1] does to its log file, so the attacker will neither see the ciphered data, nor modify it without the administrator's aware. This solution sounds nice, but there is still a problem: what if the attacker deletes all the logging data, without needing to care the administrator will detect the trouble or not? In this situation, certainly the forensic investigators will know somebody has already broken in, but that is probably everything they learn, as the logging data has gone. Especially if the attacker deletes data using special tool such as Eraser [2], logging data become more difficult, or even impossible, to recover.
- The logging data is sent out to a central logging system on another machine, via network. This method of keeping precious data is actually widely used with tool like *syslogd* [3]. The logging data will be sent to the central system, to be gathered and kept there. Unfortunately while this is a very popular scheme to prevent the attacker from modifying or deleting logging data locally, we again face some major troubles: Since the data is transmitted on the network, they might be the targets of the attacker. Various types of assault can happen: the precious (and usually sensitive) data can be secretly sniffed when they come across the network with well known packet captured tool like tcpdump [4]. Even worse, the data traffic can be redirected to a malicious host with infamous methods like Man-In-The-Middle or ARP Poison with available tools such as dsniff [5].

Last but not least, as the central logging server is exposed to the network, the attacker can directly break the central server, especially if the server runs a buggy syslogd daemon [6]. Moreover, since syslogd does not provide any authentication form, it must be strictly managed by low level access control like TCP wrapper [7] or local firewall rules (such as ipchains or iptables [8] on Linux platform). Life is so hard.

A complicated and weird solution [9] is even suggested to secure the central server: runs the central server without IP address with the hope that nobody can see it, and use a network sniffer (snort* in this particular solution) to capture data sent out from the

* snort is an open source Intrusion Detection System, and can function as a network sniffer

logging machine.

To overcome the above headaches, this paper offers a solution, including the design and implementation. A system named *Xenlog* will be presented: Xenlog is based on Xen Virtual Machine architecture, and Xenlog can be effectively used as a central logging system. With Xenlog, logging daemon and even central server is not necessarily exposed to the network, and data is securely forwarded to the central server without getting through network stack. This approach leaves the attacker no chance to sniff, steal or redirect our valuable logging data.

The paper consists of six parts: the first part briefly overviews the Xen Virtual Machine architecture. The next part presents Xenlog architecture, while the third part discusses in detail the design and implementation of Xenlog. The fourth part shows us how easy it is to deploy Xenlog software package, and demonstrates Xenlog's effectiveness with several measures. Related works are summarized in fifth part. Finally conclusions will close this paper in sixth part.

2 Xen Virtual Machine

Xen [11, 12] is a virtual machine monitor initially developed by the University of Cambridge Computer Laboratory and now promoted by various industrial monsters like Intel, AMD, IBM, HP, RedHat, Novel and by the whole open source community. Being released under the open source GNU GPL license, Xen can be used to partition a machine to support the concurrent execution of multiple operating systems (OS). Commodity OS (now officially Linux, FreeBSD, NetBSD are supported) can run on Xen with small changes to the kernel. Xen is outstanding because the performance overhead introduced by virtualization is negligible: the slowdown is around only 3% [13]. Various practices take the advantage offerd by Xen, such as server consolidation, co-located hosting facilities, distributed services and application mobility.

Xen community is working hard to gradually push Xen into Linux kernel, so it will be available for every Linux users. The process is expected to start from kernel 2.6.15.

3 Xenlog Solution

Xenlog is an architecture that allows DomU to forward logging data to Dom0 via shared memory (but not vice versa). Whenever DomU wants to send out the data, DomU just needs to write the data to a special software device in its kernel called *xenlogU*, and *xenlogU* device will automatically transmit those data through shared memory to another special device in Dom0, named *xenlog0*. (We name this device *xenlog0* to distinguish it with the *xenlogU* device in DomU). In Dom0, there is a daemon process named *xenlogd*: its sole job is to continuously query for the new data from the *xenlog0* device, and write down the gathered logging data to Dom0's file system.

Obviously with this architecture, applications in DomU can send their logging data to Dom0 via *xenlogU* device, and since *xenlog0* and *xenlogU* devices exchange information with each other via shared memory between DomU and Dom0, no whatsoever data will need to get through the network stack. That is a major improvement to the traditional methods, and it can help to overcome all the mentioned security problems with networking attacks. Another advantage of Xenlog is: as all the data is exchanged through memory, the logging process would be extremely fast and much more reliable than in the legacy way.

3.1 Xenlog Design

Goals and Approach: Xenlog is designed for simplicity and compliance. Simplicity is necessary to allow all kinds of applications use it without any difficulty. Compliance is to make sure all the applications are using *syslogd*[*] can move on to use Xenlog transparently. Moreover, Xenlog should provides solution for software programs which don't send log data to syslogd, by allowing them to forward data to Xenlog device without modifying anything in the code, or not even need to recompile.

Design: With those objections in mind, Xenlog architecture is built like below (see Figure 1):

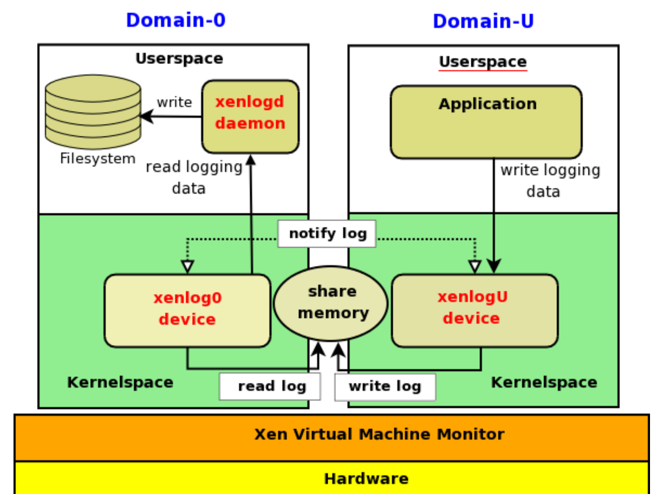

Figure 1: *Xenlog architecture*

Xenlog architecture consists of three main components: *xenlogU* device in kernel space of DomU, *xenlog0* device in kernel space of Dom0 and

* syslogd is a de-factor standard logging system of Unix and Unix-derived world

Proceedings of the 4th WSEAS Int. Conf. on Information Security, Communications and Computers, Tenerife, Spain, December 16-18, 2005 (pp201-206)

xenlogd, a logging daemon process, runs in Dom0's user space.

In this architecture, applications run in userspace of DomU sends logging data to *xenlogU* device located at */dev/xenlog* (for DomU's user perspective, there is no *xenlog0* device, so we will not name the device */dev/xenlogU*). When *xenlogU* obtains the data, it will put them in the shared buffer, then inform *xenlog0* device in Dom0. *xenlog0* will read the data from the shared buffer, and forward them to *xenlogd* daemon in Dom0.

On the other hand, *xenlog0* device exposes a device file at */dev/xenlog* in Dom0 (yes, we use same name as the device file exposed by *xenlogU* in DomU for the explained reason). In Dom0's user space, *xenlogd* daemon repeatedly probes *xenlog0* device for the new logging data. As the new logging data arrives, *xenlogd* will attempt to find those data belong to which domain. After that, logging data will be written down into the Dom0's file system, separately for each domain.

To make it easy for third party, Xenlog provides a file header *xenlog.h*, which provides declaration of necessary data structure, so people wish to access to Xenlog device for logging data can use it to write their applications, for instance should they wish to write their own logging daemon in Dom0 to take the advantage of Xenlog architecture.

Currently Xenlog is implemented in Linux OS: the reason is the port of all other Oases but Linux (such as FreeBSD, NetBSD) are not quite mature yet. We plan to provide support for those platforms once they are ready.

The following part will present our achievement in Linux environment, but let's keep in mind that implementation is basically same to all the others.

3.2 Xenlog Implementation

xenlog0 and *xenlogU* are both software devices developed to run in the kernel space of Dom0 and DomU, respectively. Written in C language, those device drivers can be employed as module inserted into kernel, or patched to built-in into the kernel: We provide both kernel module and kernel patches for those devices, and users can choose either way to apply Xenlog to their system.

As DomU and Dom0 run on the same machine, they can share memory with each other. In Xenlog architecture, when *xenlogU* is initialized, it will allocate some memory for sharing (the amount of shared memory is configurable at runtime - by default is 1 page, which is equivalent to 4KB on x86 systems), and grant those memory to Dom0 by using Xen grant reference API ([13]). This shared memory will be used to store the logging messages sent in by applications from DomU, and *xenlog0* will also read the messages from there. That is how *xenlog0* device gets logging data from *xenlogU* device.

To communicate with *xenlog0*, *xenlogU* assigns an event-channel port to send and receive notifications with *xenlog0*. This event-channel will be bound to an interrupt request (irq) handler, so it can get notifications dispatched from *xenlog0* at runtime. After that, *xenlogU* informs the value of grant reference got in the above step, together with the event-channel port to *xenlog0*. At this moment, the event-channel is not established yet, so *xenlogU* will write this information to xenstore via xenbus interface.

In Dom0, when initializing *xenlog0* registers a xenbus watch to listen for change to xenstore. When it detects the new notifications written to xenstore by *xenlogU*, *xenlog0* will try to map the shared memory reference granted by *xenlogU*. After that, it allocates an event-channel port corresponding to the event-channel port of *xenlogU*. Finally, *xenlog0* binds its event-channel to the irq handler, so it can handle the notification dispatched from *xenlogU*. From now on, *xenlogU* and *xenlog0* can contact through event-channel.

Since Xenlog device driver is designed to gather logging messages from all the applications on the system, it must be able not to be overwhelmed by too much data. No matter how big the internal buffer is, we must take into account a problem when the logging data is so voluminous that the shared buffer cannot handle them.

Another difficulty is: the shared buffer must be read and written at the same time. Let's imagine this scenario: an application in DomU tries to write data to *xenlogU* device's buffer, and at the same time *xenlog0* tries to read data from the same buffer. These activities can lead to conflict or causes the race issues.

Those troubles direct us to the decision: the shared buffer should be designed as a ring buffer. Ring buffer is special data structure which has 2 heads: one for reading and one for writing, and these heads can wrap-around when they reach the end of the buffer. Writing data to buffer will take away some spaces, but reading from the buffer will release some spaces, and the free space then can be used for another writing request later. See figure 2 below for the

declaration of the ring buffer structure.

```
struct ringbuf
{
 u32 write; /* next place to write */
 u32 read;  /* next place to read */
 u32 size;  /* buffer size */
 char buf[0]; /* data buffer */
} __attribute__((packed));
```

Figure 2: *Shared memory and internal buffer of xenlog0 use ringbuf data structure*

The ring buffer helps to address the conflict problem in shared memory: while data from DomU is written to buffer (from application request) at write-head, buffer can be read from read-head by *xenlog0*. With this solution, we do not need to use any locking primitive, which will unavoidably slow down the whole process.

Another job *xenlogU* must do when starting is to register a misc device located at */dev/xenlog*. Whenever this device gets the write request, the data from the request (this is the logging data we must handle) will be written into the shared memory.

After request data is saved into the buffer, *xenlogU* notifies *xenlog0* about the new data in the shared memory, so *xenlog0* can extract data out (after getting logging messages, *xenlog0* will release the corresponding space, thus *xenlogU* can recycle it to save new data later). Then *xenlog0* will notify *xenlogU* that it has finished the reading. All these notifications are taken place through event-channel.

After getting logging data from DomU, *xenlog0* puts this data into its internal ring buffer. The size of this buffer is also configurable at boot time, is 2 pages by default (equivalent to 8KB on x86). To distinguish data from different domains, *xenlog0* puts the logging data into a C structure named *xenlog_record*. (See figure 3) This structure will save domain id, so *xenlogd* can know which DomU sent this message. Together with *domid*, the length of message is also stored. After that, logging data is appended at the end of the structure, and everything is put into the ring buffer.

```
struct xenlog_record {
 domid_t domid; /* domain id */
 u16 length;   /* message's length */
 char buf[0]; /* message content */
} __attribute__((packed));
```

Figure 3: *logging message sent from DomU will be saved in xenlog_record structure*

On Dom0 side, *xenlog0* also registers a misc device at */dev/xenlog* and allow applications (*xenlogd* daemon in this particular case) read them. The read request to */dev/xenlog* will get logging data from the read-head of the internal ring buffer of *xenlog0*.

xenlogd is the final piece of the picture: it is a special daemon process that continuously reads data from Xenlog device put at */dev/xenlog*. While Xenlog devices are written in C, *xenlogd* is written in Python language. The reason we choose Python is *xenlogd* must report the domain name, while *xenlog_record* can only offer domain id. Unfortunately domain name is the concept at tool set level, but not at lower level, so we cannot get it via Xen API or *libxenctrl*[*]. The right way to convert domain id to domain name is to get it through xend daemon, and because *xend* exposes the domain properties via Python module, Python language is the best choice to implement *xenlogd*.

At runtime, *xenlogd* repeatedly queries */dev/xenlog* for new logging data. As new data arrives, *xenlogd* extracts information from *xenlog_record*, figures out how much logging data appended after this record thanks to the *length* field. Then *xenlogd* attempts to read exactly that amount of data. After all, *xenlogd* will convert the domain id in *xenlog_record* structure to domain name (*xenlogd* inquires this information from *xend*), then open a log file corresponding to that domain, and finally adds the logging data to the tail of that file.

As the logging data usually consist of lines, before writing to the log file, *xenlogd* will split the data into lines, and put time record before each line to declare the local time it receives this log.

4 Deploy and Evaluate Xenlog

The following parts show us how to deploy Xenlog and its add-in tool. After that the performance of Xenlog is verified by some experimental.

4.1 Deploy Xenlog

As one of the designed objects of Xenlog is to be used to transparently send the logging data to central server put in Dom0, the recommend way to deploy it is to get syslogd (or similar logging daemon) to write out the logging data to Xenlog device. We can simply put this line into */etc/syslog.conf* [14] of DomU (see figure 4)

```
*.*   /dev/xenlog
```

Figure 4: *Configure syslog to send data to xenlog device*

The remaining job is to restart syslogd (usually with the command "*/etc/init.d/syslog restart*")

* libxenctrl is a Xen library provided for third party applications

To capture logging data, we run *xenlogd* in Dom0 with command "*xenlogd start*". For those who want to see how *xenlogd* works, they can run it in debug mode (with command "*xenlogd start --debug*") in Dom0: The entire logging data received by *xenlogd* will be output on the console.

To make it easier for people to deploy Xenlog package, we provide with *xenlogd* an init-script named *xenlogd.sh* to put it into */etc/init.d* directory of Dom0. The administrator just needs to link this script to default run-level directory (for example */etc/rc3.d*), and *xenlogd* will be started automatically at boot up.

4.2 Add-in tool

There is another class of software programs need to be taken care: those have logging data, but do not use syslogd for logging. We support these kind of applications with another tool named *xentaild*. Running in background, this tool continuously checks to see if the configurable logging files are updated. If it detects the new data has been appended to a specific log file, it will extract the new logging data out, and forwards them to */dev/xenlog*.

This handy tool has derived from xtail [15] with some newly advanced features; among of them are: runs as daemon process in the background, supports configuration file with unlimited logging files at the same time.

xentaild is provided with our Xenlog package, and should be run as a daemon in Dom0 together with *xenlogd*.

4.3 Evaluation

As we see, Xenlog architecture forwards data to the central server without going through the network stack, therefore we anticipate that Xenlog gives us a better performance than the traditional way. Our measures confirm that expectation.

We will evaluate 3 schemes as follows:

- *syslogd method*: This test sends logging data using the traditional method: syslogd daemon in DomU is setup to forward logging data to remote syslogd daemon in Dom0. We configure syslogd in Dom0 to accept remote logging data (with *-r* option), and the obtained data will be forwarded to a named pipe for our awaiting script.
- *syslog_xenlog method*: In DomU, logging data are sent to syslogd, and those data will be forwarded to Dom0 via *xenlogU* device. In Dom0, a small Python script is used to get data directly from *xenlog0* device.
- *xenlog method*: In DomU, data is directly sent to *xenlogU* without going through syslogd, and this test does not require to reconfigure syslogd. In Dom0, a small Python script is used to get data directly from *xenlog0* device.

Of the above methods, only the *syslogd method* routes messages to Dom0 through network stack, while the other two uses shared memory between DomU and Dom0 to send data. The 2nd method still uses syslog to send messages, while the 3rd method directly connects with *xenlogU* device.

With the first 2 methods, a small (11 lines) Python script will send 1000 syslog messages, each message contains 150 characters, of *local7.debug* type to Dom0. Likewise, with the last method a similar script will send the same data to Dom0 via *xenlogU* device. Each test will be repeated 10 times for corresponding method, and will be named Test1, Test 2, ..., Test 10.

At the time of test, syslogd is restricted not to receive any logging data, but only messages of the special type *local7.debug*, so we ensure that nothing will interfere our experiments.

On receiving side in Dom0, in the first method we use a small Python script (of 39 lines) to get data from syslog named pipe, and a similar script to pick up data from *xenlog0* device in the last 2 cases. These scripts will record the time they obtain the 1st and the 1000th message, then calculate the difference to give us the total time needed to receive all the messages.

The evaluation is carried out on a machine with Athlon XP 2500 processor, 512MB RAM and IDE HDD of 40GB. We uses Xen 3.0 unstable version, and Dom0 is setup with 384MB, while DomU uses 100MB memory with a file-backed files system of 2GB and a file-backed swap file system of 256MB. Both Dom0 and DomU use Linux Ubuntu BreezyBadger distribution.

Our evaluation returns the result as followings: on average *syslogd method* takes 0.430896 sec, *syslog_xenlog method* takes 0.312821 second and *xenlog method* takes 0.018188 second to complete the test. Of 3 methods, *syslogd method* takes most time to complete: 1.377 times more than *syslog_xenlog method*, and 23.691 times more than *xenlog method*. We can also conclude that users can benefit from Xenlog by redirect the logging data to *xenlogU* device, as demonstrated with the result of *syslog_xenlog method*. And above all, *xenlog method* is unsurprisingly the fastest method, far faster than the traditional method *syslogd*.

In general, we also believe that *xenlog method* is

more reliable than *syslogd method*. The reason is with Xenlog, data is exchanged in the memory, there is very little chance that something will corrupt our data. Another important point we should not ignore: syslogd uses UDP, a connectionless protocol, to send data, and that makes syslogd even less trusted.

5 Related works

Because central logging is very important for distributed system, many efforts have been made to send the logging data to the central server, in many ways. The most popular mechanism is syslogd. The syslogd and syslogd-derived solutions all have capability of forwarding logging data to remote central machine.

In particularly, there are some alternatives to syslogd with enhanced functions. One is syslog-ng [16], which is declared as syslogd replacement. syslog-ng provides some nice features; among them are filter based on message contents using regular expression, more flexible configuration and allow to use TCP as transmission protocol when sending logging data to remote machine.

Another option is rsyslog [17]: the outstanding feature of rsyslog is this tool permits to send logging data to a database server (like MySQL) on the network.

The common requirements to all of the above solutions are: the client and server need to expose to the network, and all the transmission is through network. Thus they are all vulnerable to security issues we mentioned.

Since the above projects are compatible with syslogd, Xenlog could be comfortably used with all of them.

As we are aware, Xenlog is the first project exploiting the advantage of Xen technology to secure the logging data.

6 Conclusion

Traditionally, logging data could be secured by sending out to the central system on the network. This method poses a lot of issues: sensitive data when going through the network might be captured, stolen, and there are some potential troubles for the central server which must expose to the network. This paper has proposed a full architecture based on Xen virtual machine named Xenlog. With Xenlog, data is transferred from logging system runs on DomU to a daemon on Dom0. Xenlog exploits the fact that both DomU and Dom0 run on the same machine, so they can exchange information straightly via shared memory. Consequently we no longer need to use the network stack for transmitting data, and we can happily avoid the problems of the legacy approach. The resulted software proves to be flexible, reliable, far more effective, and can be used by all kinds of applications wish to forward logging data to the central server. Especially, if applications are rewritten to take the advantage of the new interface */dev/xenlog*, they can send messages to central server in Dom0 in a very high speed, as validated in the evaluation.

The method and solution applied for Xenlog can be reused for many other problems which need to exchange data between DomU and Dom0 in Xen.

References:

[1] *Shamhain tool*. http://samhain.sourceforge.net. September 2005

[2] *Eraser tool*. http://www.heidi.ie/eraser/. August 2003

[3] RFC 3164 - *The BSD Syslog Protocol*. http://www.faqs.org/rfcs/rfc3164.html. August 2001

[4] tcpdump tool. http://www.tcpdump.org. October 2005

[5] Dug Song, *dsniff tool*. http://naughty.monkey.org/~dugsong/dsniff/. December 2000

[6] Bugtraq. *Linux syslogd Denial of Service Vulnerability*. http://www.securityfocus.com/bid/809. November 1999

[7] *TcpWrapper*. ftp://ftp.porcupine.org/pub/security/. April 2004

[8] *Netfilter/Iptables project*. http://www.netfilter.org. July 2005

[9] Mick Bauer: Stealthful Sniffing. *Intrusion Detection and Logging*, http://www.linuxjournal.com/article/6222. October 2002

[10] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebar, Ian Pratt, Andrew Warfield, *Xen and the art of virtualization.* ACM Symposium on Operating Systems Principles, October 2003

[11] Ian Pratt, Keir Fraser, Steven Hand, Christian Limpach and Andrew Warfield, Dan Magenheimer, Jun Nakajima and Asit Mallick, *Xen 3.0 and the art of virtualization*. Proceedings of Linux symposium, July 2005

[12] Bryan Clark, Todd Deshane, Eli Dow, Stephen Evanchik, Matthew Finlayson, Jason Herne and Jeanna Neefe Matthews, *Xen and the art of repeated research,* Freenix 2004

[13] Christopher Clark, *A Rough Introduction to Using Grant Tables,* Xen tree code: docs/misc/grant-tables.txt, March 2005

[14] *syslog configuration*, http://www.die.net/doc/linux/man/man5/syslog.conf.5.html

[15] Chip Rosenthal. *xtail tool*, http://www.unicom.com/sw/xtail/. June 2000

[16] syslog-ng project. http://www.balabit.com/products/syslog-ng/. April 2003

[17] rsyslog project. http://www.rsyslog.com. October 2005