CEE Log Integrity and the "Counterpane Paper" [1]

Rainer Gerhards
rgerhards@hq.adiscon.com

December 2, 2010

Abstract

While "the Counterpane paper" provides an excellent method to secure logs against modification, I think it alone is not sufficient to protect real-world CEE logs. This short document provides my reasoning.

1 Quick Summary of the Paper

This is by no means a complete summary, just an extraction of the most important facts as far as CEE is related (obviously based on my personal interpretation of this relationship).

In [1], Schneier and Kelsey describe a system that can be used to secure audit logs in a way the enables tamper-detection.

It is important to note some basic assumption these authors make about log files. In their terms, a log file is a sequence of log Records $L_0, L_1, \ldots$ where each log record L_{i+1} has a strict predecessor L_i and no log records between L_i and L_{i+1}. Let $\mathcal{L}$ be the ordered set of all L_i. In [1, Sect. 3.5, Item 4] Schneier and Kelsey show that it is possible to verify a subset of $\mathcal{L}$, but this subset is restricted to all records from L_0 to L_n for some $n \leq |\mathcal{L}|$. So the subset must always include the first record and all records until the final one requested. As such, the strict sequence of log records within the log file is still preserved[1].

The overall idea of the scheme is similar to cipher block chaining: To sign L_{i+1}, a hash obtained from L_i is used. As such, a "chain of trust" is created from L_0 to any L_n. This chain can be verified by obtaining L_0, running the algorithm in sequence on any L_i with $0 < i < n$ and finally verifying L_n based on L_{n-1}. By utilizing this chain, valid signatures can be obtained. Most importantly, if any L_i with $0 \leq i < n$ is missing, it's non-presence can be detected by the fact that L_{i+1} fails verification.

The paper also provides, in section 4.4 and 4.5, methods which with logs can be secured on multiple systems. But (as far as I understand the paper), even in these cases it is required that all log records are available on all systems.

It is important to note that the scheme in theory can be used to re-start log processing for each individual message. However, in that case it would not

[1] The method is more powerful in that a reviewer is not necessarily enabled to actually read the contents of all obtained log records, but this is not important for my reasoning and so I ignore that

be possible to detect mangled-with logs, as the extra protection provided over other schemes is rooted in the chaining algorithm. Signing single events - or related subsets - removes that capability.

2 Real-World Logging Scenarios

I concentrate on syslog scenarios, as this is the area I am most familiar with. I think it can easily be extended to other protocols, provided that they offer relaying and filtering capabilities.

Let U_0, U_1 be original senders, Let T_0, T_1 be final receivers and let R be a relay. A typical real-world deployment is that at the original sender level, some (noise) events are filtered out, and only the more important ones are sent to the relays. Inside the relay, events are often directed to different destinations based on their content.

In some other setups, original senders may not discard any events and thus will forward everything. However, relays will probably still direct different messages to different final destinations.

In my experience, it is very uncommon that all of an original sender's messages will be forwarded inside the relay chain to the same final receiver and be stored in the final receivers log file and no partial log file is created on any interim system.

3 Why the Method from the Counterpane Paper is not a solution to CEE needs

3.1 Technical Issues

3.1.1 Logical Channels

Now let $\mathcal{L}_0 = (L_{0_0}, L_{0_1}, \ldots)$ be U_0's logfile and $\mathcal{L}_1 = (L_{1_0}, L_{1_1}, \ldots)$ be U_1's log file.

Now I assume that L_{0_1} is a noise event, so U_0 will actually transmit $L_{0_0}, L_{0_2}, \ldots$ to R. At this point, the paper's verification method will correctly detect that messages ar missing at R, but nothing else can be obtained. Most importantly, the messages are deliberately missing, so this is not a failure in itself. One may now argue that to solve this problem, the method could be used on the logical transmission channel. This is true, but this also means the method does no longer guard the authenticity of the complete $\mathcal{L}_0$ set of original log messages but rather the authenticity of the subset that was transmitted over the channel. Furthermore, this becomes very complex if ther are many logical channels, as is the case in multi-level relay chains. Note that RFC5848 provides a solution to many of the questions that relate to logical channels, but leaves open for extension a number of very important cases in real-world deployments.

The same problem occurs if R submits different type messages (e.g. authentication related and kernel-related) to different destinations. Let us assume that R received the following sequence of messages: $L_{0_0}, L_{0_1}, L_{1_0}, L_{0_2}, L_{1_1}$. The relay is configured to forward messages of a certain type, which happen to be L_{0_1} and L_{1_0} to destination T_0 and all others to T_1. While the relay is now capable to

verify each message, T_0 and T_1 are not, simply because each of them misses some messages inside the chain.

Please keep in mind that I am describing typical setups found in practice. It is worth noting how RFC5848 solves this problem. In its section 4.2.3, it defines "Signature Groups", which are actually logical channels. All messages inside a signature group must go to the same set of machines and each machine must have the whole set. RFC5848 then tries to define several potential criteria for building signature groups. The primary mechanism is the syslog PRI field, which has severe limits and thus can often *not* be used inside larger deployments. RFC5848 acknowledges this by providing an extension option (type "3"), which is not further specified. RFC5848 still permits to check the validity of a signature in case of missing messages[2].

3.1.2 Integrity of the Log as Whole

Even if I put aside the complexities of logical channels, the log file $\mathcal{L}_T$ on a destionation system T can not be verified as whole iff we assume that $\mathcal{L}_T$ contains messages from different sources. For example, let $\mathcal{L}_T$ contain $L_{0_0}, L_{0_1}, L_{1_0}, L_{0_2}, L_{1_1}$. Then an attacker could remove all records L_{0_i} from $\mathcal{L}_T$ without this being noticed. The methods described in section 4.4 and 4.5 may be used to guard against this issue, but this comes of the price of tying all systems very closely together.

3.2 Existing Software

One of our goals in CEE is to provide a practical solution that can be quickly adopted by real-world products and thus be used to solve practical problems currently being faced.

Even if all technical issues I see were solved (or my analysis be wrong), we still had the problem that the method described in the Counterpane paper requires that

- only a single protocol is used for logging
- all existing software is adopted to this protocol
- all other protocols are no longer used
- all users always and immediately update all of their systems

This seems absolutely impractical to me.

4 Quick Suggestion for a Practical Solution

I do not intend to specify the technical details in this short paper, but would like to convey a core idea. In my view, a log file $\mathcal{L}$ is a set of log records L_{u_i} where u denotes the original sender and i the i-th log record it sent. In my opinion,

[2]But it is important to know that this stems back to the fact that signature are not part of the message itself. I will not go into more detail here, my main point was that "logical channels" are rather complex.

we should seperate the integrity of the individual message from the integrity of the log as whole.

Signing individual messages reliably is hard, but there already exist methods to do that (e.g. RFC5848 and many proprietary solutions). We should simply permit that L_{u_i}, for all i, uses the signature method that u supports (which may be "none").

Then, we can sign $\mathcal{L}$ itself using some other method. This can be the one from the Counterpane paper, but also any other method that is considered sufficiently secure for the required task (there may even be different legal requirements in different countries).

So we effectively have two layers, where different methods could be used for each layer. Of course, using different methods is not really desirable. Howerver, it may be the only solution in order to get something adopted and used in practice fast (I think the support for multiple CLS encodings and multiple CLT has the very same reasons).

To make the two-layer approach work, we just need to define a proper container for original log records, and some rules to ensure that the original content is not modified. Then, any message signature method could be used with any file-signature method.

References

[1] B. Schneier and J. Kelsey. Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security (TISSEC)*, 2(2):159–176, 1999.