Version 1.0

Rainer Gerhards (rgerhards@adiscon.com),
Andre Lorbach (alorbach@adiscon.com)

The systemd journal announcement in 2011 lead to a big heated discussion whether or not the end of line for syslog has been reached. The original journal announcement posted a number of claims that the syslog community considered as simply wrong. However, there were also some good arguments present. Since then, roughly one and a half year has passed. In this paper, we will describe the situation as of May 2013 as the rsyslog project sees it. We describe which integration efforts have been made, what can be learnt from the past and how the rsyslog sees systemd journal and rsyslog coexist in tomorrow's environments.

Some journal / rsyslog history

When systemd journal was announced in late 2011, it described a couple of so-called „serious problems“ with syslog. In the last section of this document, the original rebuttal to these arguments can be seen – they are still valid, except that even more functionality has been added for things that were considered missing.

The journal announcement caused some disruption to rsyslog development, as Adiscon, the prime sponsor, reconsidered if funding something that is „dead“ would be a smart thing. Thankfully, the decision makers could be convinced to continuously support the project, which resulted in even higher activity, as some points from the journal announcement were indeed valid, or at least good suggestions, and syslog technology was advanced.

The big question is still „does the journal replace syslog“. Today, the answer from the rsyslog team is probably partly, but not as a whole. To understand that answer, we first need to dig into some history and a related operating system, then look at integration scenarios and the technology that enables them. Then, we will look into rsyslog architecture and finally describe some new important features (for example log signing and encryption) as well as provide the still-valid answer to the original journal announcement.

Journal-Like Systems in the Past

In our opinion, the core ideas of systemd journal and the Windows Event Log are very similar (while the technology behind them probably is very different).

Both utilize a binary database which provides roll-over capability (in a ringbuffer-like way), searchability and relatively fast seek times. Both use simple structured data inside the logs, but provide the ability to use free-form text. They also collect meta data information, like timestamps, user ids and so on. In both cases, the actual files are secured, in the case of Windows even by the core OS (the Windows Event Log is a core OS service). And finally both try to make messages identifiable by utilizing unique Ids for event classes. In the case of the Windows Event Log, this is a hierarchical key, with an integer id at the lowest level, while in systemd journal a GUID is used.

Give these similarities, the use and evolution of the Windows Event Log is probably a good indicator of how the journal will be used.

Windows Event Log History

The Windows Event Log was introduced in 1993 as part of Windows NT 3.1. It was originally a single-computer system without any network capabilities, except for the ability to remotely connect to a system and read its event log. The core features mentioned above were already present in 1993.

Besides minor enhancements, the system was mostly unchanged until the release of Windows Vista in 2007. Here, it was remodelled to permit custom event logs, easier filtering and programmatic access and – most importantly – network access. Starting with the new Event Log system, it was possible to create push and pull subscriptions, so that central event logs could be managed. In any case, the event log database system is the same for both local-only and central event repositories.

Windows Event Log and Syslog

In theory, the system provides a solution for all logging needs. In practice, there were many gaps to fill, which lead to a rich set of syslog tools on Windows.

A core problem was that the Event Log was a closed system. However, in practice we have a heterogeneous world and there are many more systems than Windows: routers, switches, other operating systems and many more network-enabled devices.

Out of this problem, Rainer Gerhards wrote the first syslog server for Windows, named „NTSLog“ when it was first released in 1996. This project was the result of network consulting work and the frustration that arose out of the inability to gather that all-important syslog debugging information (back in 1990's security issues were not so much a driver behind logging technology, it was very debugging-focussed). Over the years this tool has evolved in a full-blown syslog server for Windows, which is now named „WinSyslog“. Many other vendors also followed on that route, with „Kiwi Syslog“ probably being the best known on Windows. These tools usually provide the ability to store messages to local files and databases, as well as the Windows Event Log, if so desired (this is a very low-end use case, due to the limits of the ring-buffer database and its performance). These tools are used to integrate Windows as a receiver into a heterogeneous infrastructure.

The other way around, integrate Windows as a client into an enterprise, was also a hot topic from very early days on. Especially as the original Event Log did not have any decent network functionality. Out of that need, Rainer Gerhards in 1997 created the tool „EvntSLog“, which was the first Event Log to Syslog forwarder. It was later renamed to „EventReporter“ as which it is currently being distributed. This tool became very popular by large enterprises, which used it for centralized troubleshooting, a very hard problem if someone ran thousands of Windows machines. Note that many large companies already had Unix-based management systems in place at that time, and enabling Windows to talk to the enterprise environment was vital for many of them.

The class of EventLog-To-Syslog tools became very popular, and many small and large vendors wrote their own tools providing similar services. A well-known example is the Snare for Windows (the rsyslog Agent for Windows is another example).

What was actually solved?

These tools became so successful because the solved the „language translation problem“ of network event logging. While Unix and Linux did not understand Windows natively, and Windows did not understand them, with the help of syslog they could be integrated into a single system. Even more so for the myriad of other network devices.

Of help here is that a basic UDP syslog client is extremely easy to write, and almost every vendor has done so. While such a basic client is missing a lot, the base functionality of conveying information is still available, and better limited than not at all. So syslog became the universal language of network event logging.

In order to build a centralized system, one needs either a common protocol – syslog in this case – or the ability to understand multiple protocols. The latter is also possible with modern syslogd's, so that they can tackle the problem from different angles.

Lesson Learned from Windows Event Log

What the Windows Event Log story tells us is that no single system is sufficient for all logging needs. There need to be common standards and tool-sets with translation functionality.

Even though the Windows Event Log provided quite good logging features, syslog grew up, matured and is very well alive in this operating system space. Quite interestingly, not even the much-enhanced network support in post-2007 Windows Event Log lead to a decline of syslog technology on Windows.

Relation to journal

With that historical background, we conclude that the systemd journal will also not replace the syslog infrastructure on Linux. However, it will affect it and already has done so.

Changes caused by the journal

As in the Windows case, there are many environments where no enterprise integration is needed, and the journal probably provides a very valid and attractive solution to their logging needs. This is especially true for personal desktops and notebooks, where the prime logging need still is troubleshooting. We envision that journal will become dominant in that environment.

Things are different, though, in the enterprise space. Integration into heterogeneous environments is very important here, and syslog provides the necessary integration services. Our vision is that in this space rsyslog and the journal will work together for the foreseeable future.

These changes obviously need to affect the rsyslog project. The original mission for rsyslog was:

> *Rsyslog is an enhanced syslogd supporting many modern features. It is quite compatible to stock sysklogd and can be used as a drop-in replacement. Its advanced features make it suitable for enterprise-class, encryption protected syslog relay chains while at the same time being very easy to setup for the novice user.*

This was the philosophy when we originally started the project in 2003. Actually it became a bit dated in respect to the more advanced use cases even before the journal arrived. Finally, when the journal came up and affected the logging space, we had long discussions resulting in a new mission statement:

> *Rsyslog is a fast and feature-rich enterprise IT event processing system. It provides rich out-of-the box functionality for many applications and provides a great framework to support custom uses. It supports trusted logs, secure-long-term storage and enables users to unify events, augment, and transform them. Thus, it bridges the gap between different logging systems and lays the foundation for understanding the ultimate meaning of captured events.*

Rainer Gerhards, Feb. 2012

Note the not-so subtle differences between them. A core point, not induced by the journal, is that rsyslog goes away from pure syslog logging to be a network event processor. This transition already happened in 2012, and so the mission statement change just reflected how things had evolved. The other big change is that low-end use cases and novice users have been removed from the core mission. This does not mean that rsyslog will not work well on low-end systems or provide the services needed to novice users. However, it means that whenever there is a conflict where a feature can become better for enterprise computer or non-enterprise computing, the enterprise use case will win that battle. This policy change is a direct result of the changed logging world due to the journal, and our anticipation that the journal will be dominant in the non-enterprise space.

Rsyslog / Journal integration

The actual integration is done via rsyslog input and output modules and of course a bit of configuration language tying things together. Integration scenarios follow the use cases described above.

New rsyslog Modules

omjournal

This is rsyslog's output module to write data to the journal. It permits to store data rsyslog received inside the journal, so that it can be processed by journal tools. The core idea here is that we need a way to integrate some syslog sources, with probably low message volume, right into the journal, which serves as the users primary troubleshooting tool. We assume that the user may not actually be aware that he is running rsyslog (much as it is today in those scenarios) and does not really care about it's configuration.

The initial stable version is available with rsyslog 7.4.0 and above.

There are currently some subtle issues, as journal does not trust rsyslog, so all messages will show as if they originated from rsyslog (and not the original system). However, we do not consider this to be problematic for the given use case as the user will most probably be just interested in the message text and much less in the meta data. We will of course evaluate user needs and potential enhanced journal APIs to provide an even tighter integration.

Omjournal is a regular action, and so enhanced filters can be used to define what will actually be written to the journal. Most importantly this permits to remove known noise events before they hit the journal database. However, we do not anticipate that many users will make use of that facility.

Imjournal

This is rsyslog's input module to natively read the journal, a Red Hat contribution. It is currently not project-supported, but chances are high it will become so.

Journal provides two ways to obtain data:

- via the traditional system log socket
- via the journal API, directly from the journal database

For many use cases, using the traditional log socket is sufficient. It provides the message as well as some meta date (like pid, process user ID and so on – basically SCM_CREDENTIALS information). However, this interface does not provide structured logging elements present inside the journal. This is not a big problem currently, as there is only very limited structured data present in the journal.

The journal API obviously provides every information content the journal has stored. So it can be used for deeper integration and will become more important as -hopefully- more structured data gets written to the journal.

Note that it is possible to run both imuxsock (the traditional system log input) and imjournal concurrently on the same system. If so, care must be taken to avoid message duplication. Rsyslog currently does not offer any special methods to do so other than the regular filter capabilities.

The initial stable version is available with rsyslog 7.4.0 and above.

We will evaluate the use of this module and user request very carefully and plan to extend it's capabilities as need arises. We do not envision any strong demand in the near future. In our opinion, the need is closely coupled to the success of structured logging via the journal.

Deployment Scenarios

The deployment scenarios are grouped on use cases.

Non-Enterprise Environment

As we assume the user is not really interested in rsyslog here, but rather in integrating non-journal data into the journal, we think a very simple configuration will probably be able to handle the majority of use cases. As a hint to distro packagers, they may consider to provide a package that contains such a canned config.

A typical scenario we envision here is that a home user wants to integrate his router's logs into the journal.

The proposed configuration file for this use case is as follows:

```
rsyslog.conf:
      /* first, we make sure all necessary modules are present: */
      module(load="imudp") # input module for UDP syslog
      module(load="omjournal") # output module for journal
```

```
/* then, define the actual server that listens to the
* router. Note that 514 is the default port for UDP syslog.
*/
input(type="imudp" port="514" ruleset="writeToJournal")

/* inside that ruleset, we just write data to the journal: */
ruleset(name="writeToJournal") {
      action(type="omjournal")
}
```

Note that this can be used as sole config. It does not even write to local log files. We proposed UDP syslog, as this is the most common option for low-end devices, or at least their default. The security issues associated with plain UDP syslog do not strongly matter in the envisioned environment.

Enterprise Environment

It is not possible to provide a similar blueprint configuration for enterprise needs – they are too different. Basically an enterprise needs to evaluate it's logging needs and craft rsyslog configurations accordingly.

Often, the traditional system log socket may be sufficient, but in other cases imjournal may be needed. As an example, we provide the following configuration snippet that pulls data from the journal and stores it as structured log in Project Lumberjack JSON/CEE structured log format:

`rsyslog.conf` **snippet**:

```
module(load="imjournal" PersistStateInterval="100"
       StateFile="/path/to/file")
module(load="mmjsonparse") #load mmjsonparse module for structured logs

# the template below must be on ONE line
$template CEETemplate,"%TIMESTAMP% %HOSTNAME% %syslogtag%
            @cee: %$!all-json%\n" #template for messages

*.* :mmjsonparse:
*.* /var/log/ceelog;CEETemplate
```

This is deliberately just a config snippet – for this use case it is assumed that data is further processed. The key points for above examples are that

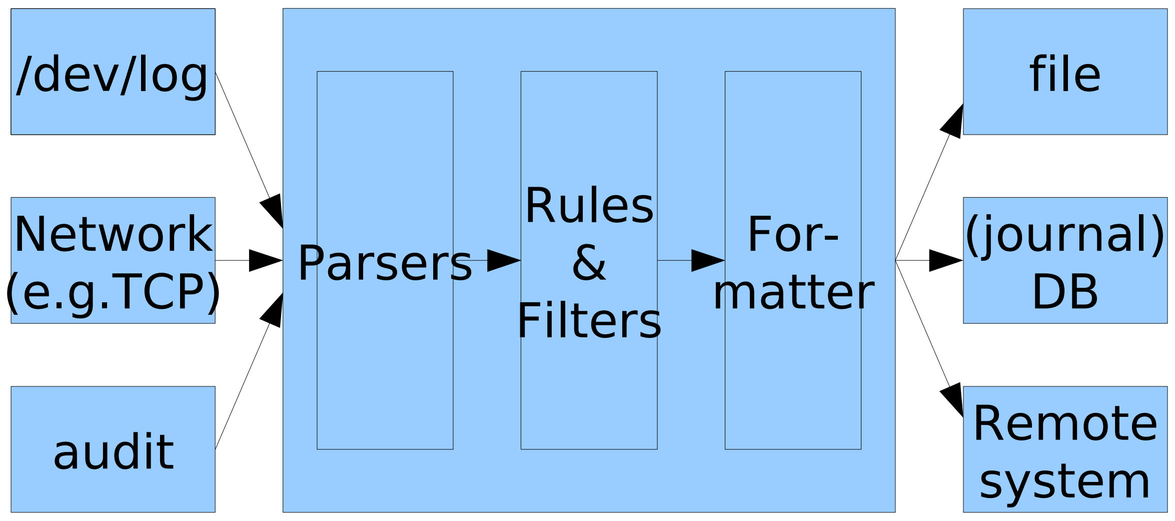
- journal data is obtained in native format (via imjournal)
- structure is preserved by using Lumberjack CEE format

All actual processing is user-defined and beyond the scope of this paper.

Rsyslog Design

This section is meant to be a very quick overview over the core rsyslog design, as much as we need to understand for rsyslog / journal integration.

A key point is that rsylsog is highly modular and extensible. In a rough sketch, its key elements are:



On the left-hand side are input modules. These gather data from various sources, like files, the log socket or network protocols like syslog. Each of these modules can be thought of as an individual project (and actually is) and each can by dynamically loaded.

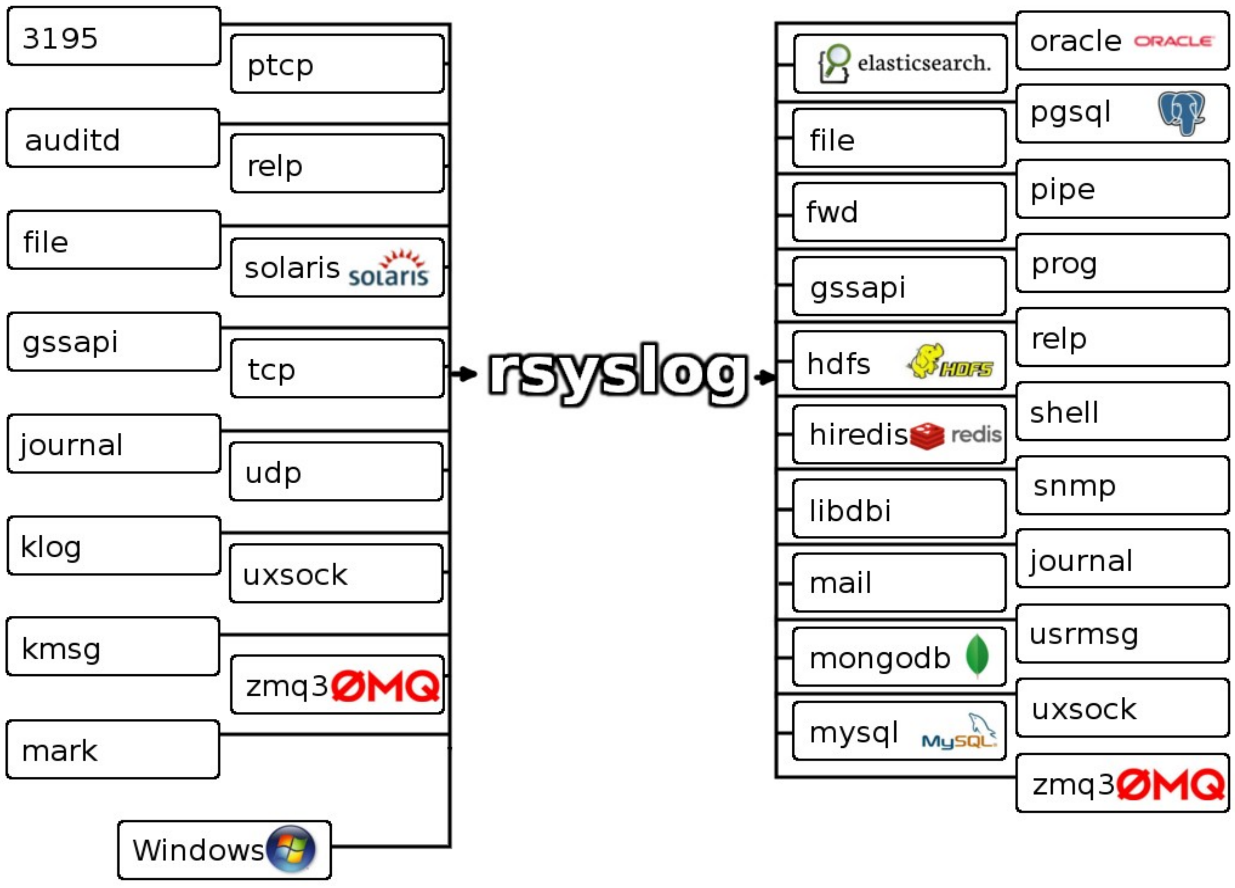
At the right-hand side are the output modules, similar in structure to the inputs except that they obviously are message sinks. Most importantly, they are also dynamically loadable.

Right in the middle is the rsyslog core, here drawn in a very simplified sketch. It contains all the plumbing that is needed to bind inputs to outputs, and do the filtering and transformation that is necessary inside a network event processor. Drawn are the parser modules, which take the input data and parse it into usable structure, the rule and filter engine, which controls processing flow, and the formatter, which provides customer formats to the output modules. Many of them are also loadable modules and there are more interfaces and module types present, which we left out for brevity.

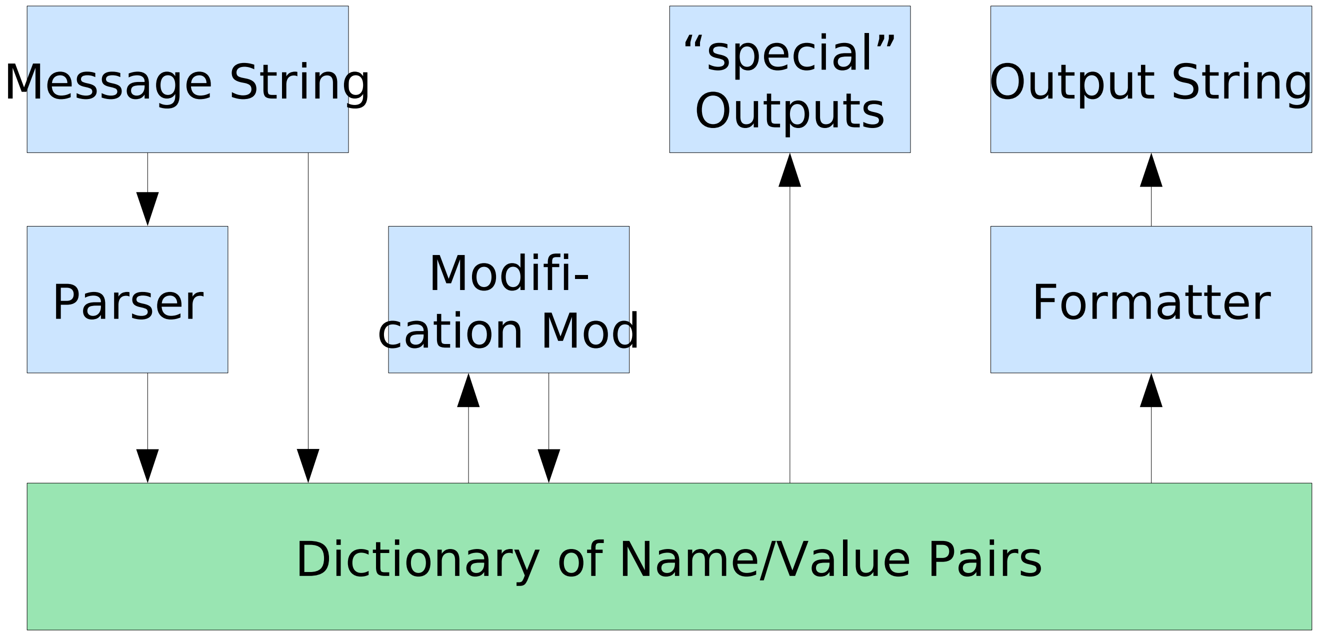
The core idea is that all the „hard plumbing“ is done inside the rsyslog core, which makes it fairly easy to write input and output plugins.

This explains why it was easy to add imjournal and omjournal for journal integration: all the required base technology was already present and only some tiny code areas needed to be written to actually connect to that subsystem.

An overview over currently existing inputs and outputs is provided in the next drawing:



To understand the rsyslog core message handling a little bit better, let's look at the internal message flow:



Arriving messages are run either through parsers or stored directly by the input module in a dictionary of name/value pairs, which is the actual internal rsyslog message representation. So at the base, a message is already treated as a structured element (this is a core design idea since many years, with the base implementation dating back to around 2006).

This dictionary is being worked on by so-called „modification modules“, which enable to change the dictionary context. A good example is „mmjsonparse“, which takes a traditional syslog message and checks if it is JSON-enhanced (according to Project Lumberjack specification). If so, the structure is extracted and the dictionary populated with it.

Finally, output modules either directly access the dictionary or use formatters to have a user-selected format provided. The latter is the usual case, whereas direct dictionary access is available primarily for a select set of advanced use cases (direct access obviously removes the user ability for format conversions).

There are also facilities inside the rsyslog.conf language to modify the dictionary, most importantly to create new sub-structures.

Important New Features

Rsylsog v7 provides many important new features, for example a much easier to grasp configuration language.

In our context, however, mainly those affecting the journal and it's use cases are of interest. The most important ones are

- input and output modules for the journal, as already described
- and rsyslog's new security pack

In this section, we will focus on the latter. The security pack was developed in the 7.3 branch and is available as stable version in 7.4. It offers three core features

1. log anonymization
2. log file signatures
3. log file encryption

The anonymization part currently focusses on IPv4 addresses. While it provides very solid functionality in that space, it is also a tester to see how well-received such a feature is. If demand will be voiced, the module will be expanded, probably starting with IPv6 support – and whatever else may be brought up.

The log file signature system provides integrity proof for log files. This is surprisingly hard to get right, and as such a full section has been provided on this feature later in the paper.

Encryption support is also vital in some environments. Rsyslog now natively supports symmetric cryptography via libgcrypt to realtime-encrypt log messages. This works in combination with signing the logs, if so required. Special care has been taken in regard to key management. Rather than hard-coding key management into rsyslog, we followed its modular design and have the ability to call a program which actually obtains the key for rsyslog's use. That way, the user is able to provide as complex and secure key exchange protocols, including secure stores, as necessary. As a working basic mechanism, we have also included the ability to read the key from a local file. That, however, is not a very secure choice against a well-done attack, but may be useful for less critical use cases.

In the 7.4 branch encryption applies to log files, only. In 7.5 experimental, we extend this feature to on-disk queue (spool) files.

Note that network encryption is already available for some years. It uses RFC 5426 TLS protected syslog with mutual authentication. So this is not a new albeit very useful feature (and one that the initial journal announcement totally ignored).

The history of rsyslog's log signing solution

In this chapter we have an in-depth look at the rsyslog log signing solution, and how it evolved. This is a very important topic as the journal team claimed that tamper-detection was a key benefit that the journal offered. In this section, we try to describe how the rsyslog's team thinking on signatures evolved, where we saw the weak spots in the past and why we finally moved to the solution that now is in place – and why we consider it to be secure.

Note that this section and a related blog posting on http://blog.gerhards.blog (the rsyslog main author's blog) were written in parallel and contain mostly the same information.

Rsyslog 7.3 is the first version that natively supports log signatures, and does so via a newly introduced open signature provider interface.

We need to dig a little back in history to understand the decision for the current signature system. We looked at log signature a long time ago, the interest started around 2001 or 2002, as part of the IETF syslog standardization efforts. We have to admit that at this time there was very limited public interest in signed logs, even though without signatures it is somewhat hard to prove the correctness of log records. Still, many folks successfully argued that they could proof their process was secure, and that proof seemed to be sufficient, at least at that time.

A core problem with log record signatures is that a single log record is relatively small, typically between 60 bytes and 2k, with typical Linux logs being between 60 and 120 bytes. This makes it almost impossible to sign a single log record, as the signature itself is much larger. Also, while a per-record signature can proof the validity of the log record, it cannot proof that the log file as such is complete. This second problem can be solved by a method that is called "log chaining", where each of the log records is hashed and the previous hash is also used as input for hashing the current record. That way, removing or inserting a log record will break the hash chain, and so tampering can be detected (of course, tampering within a log record can also be easily detected, as it obviously will invalidate the chain as well).

This method is actually not brand new technology but ages old. While it sounds perfect, there are some subtle issues when it comes to logging. First and foremost, we have a problem when the machine that stores the log data itself is being compromised. For the method to work, that machine must both create the hashes and sign them. To do so, it must have knowledge of the secrets being used in that process. Now remember from your crypto education that any secrets-based crypto system (even PKI) is only secure as long as the secrets have not been compromised. Unfortunately, if the loghost itself has been compromised, that precondition does not hold any longer. If someone got root-level access, he or she also has access to the secrets (otherwise the signature process could also not access them).

You may now try as hard as you like, but if everything is kept on the local machine, a successful attacker can always tamper the logs and re-hash and re-sign them. You can only win some advantage if you ship part of the integrity proof off the local system - as long as you assume that not all of the final destinations are compromised (usually a fair assumption, but sometimes questionable if everything is within the same administrative domain).

The traditional approach in logging is that log records are being shipped off the machine. In IETF efforts we concentrated on that process and on the ability to sign records right at the originator. This ultimately resulted in RFC 5848, "signed syslog messages", which provides the necessary plumbing to do that. A bit later, at Mitre's CEE effort, we faced the same problem and discussed a similar solution. Unfortunately, there is a problem with that approach: in the real-world, in larger enterprises, we usually do not have a single log stream, where each record is forwarded to some final destinations. Almost always, interim relays filter messages, discard some (e.g. noise events) and transform others. The end result is that a verification of the log stream residing at the central loghost will always fail. Note that this is not caused by failure in the crypto method used - it's a result of operational needs and typical deployments. Those interested in the fine details may have a look at the log hash chaining paper Rainer Gerhards wrote as an aid during CEE discussions. In that paper, he proposed as an alternative method that just the central log host signs the records. Of course, this still had the problem of central host compromise.

Let's sum up the concerns on log signatures:

- signing *single* log records is practically impossible
- relay chains make it exceptionally hard to sign at the log originator and verify at the central log store destination
- any signature mechanism based on locally-stored secrets can be broken by a sufficiently well-done attack.

These were essentially the points that made us stay away from doing log signatures at all. As we had explained multiple times in the past, that would just create a false sense of security.

The state of all the changed a bit after the systemd journal was pushed into existence with the promise that it would provide strong log integrity features and even prevent attacks. There were limited technical facts associated with that announcement, but it was very interesting to see how many people really got excited about it. While one of us clearly described at that time how easy the system was to break, people begun to praise it so much that we quickly developed LogTools, which provided exactly the same thing. The core problem was that both of them were just basic log hash chains, and offered a serious amount of false sense of security (but people seemed to feel pampered by that, so the social engineering worked...).

Our initial plan was to tightly integrate the LogTools technology into rsyslog, but our seriousness and concerns about such a bad security took over and we (thankfully) hesitated to actually do that.

At this point of the writing credits are due to the systemd journal folks: they have upgraded their system to a much more secure method, which they call forward secure sealing. We haven't analysed it in depth yet, but it sounds like it provides good features. It's a very new method, though, and security folks for a good reason tend to stick to proven methods if there is no very strong argument to avoid them (after all, cryptography is extremely hard, and new methods require a lot of peer review, as do new implementations).

That was the state of affairs at the end of last year. Thankfully we were approached by an engineer from Guardtime, a company that we learned is deeply involved in the OpenKSI approach. He told us about the technology and asked if we would be interested in providing signatures directly in rsyslog. They seemed to have some experience with signing log files and had some instructions and tools available on their web site as well as obviously some folks who actually used that.

We have to admit that we were not too excited at that time and very busy with other projects. Anyhow, after the Fedora Developer Conference in February 2013 we took the time to have a somewhat deeper look at the technology - and it looked very interesting. It uses log chains, but in a smart way. Instead of using simple chains, it uses so-called Merkle trees, a data structure that was originally designed for the purpose of signing many documents very efficiently. They were invented back in the 1970's and are obviously a quite proven technology. An interesting fact about the way Merkle trees are used in the OpenKSI approach is that they permit to extract a subset of log information and still prove that this information is valid. This is a very interesting property when you need to present logs as evidence to the court but do not want to disclose unrelated log entries.

While all of this is interesting, the key feature that attract our interest is the "key-less" inside the KSI name. If there is no key, an attacker can obviously not compromise it. But how will signing without a key work? We admit that at first we had a hard time understanding that, but the folks at Guardtime were very helpful in explaining how it works. Let us try to explain in a nutshell (and with a lot of inaccuracies):

The base idea goes back to the Merkle tree, but we can go even more basic and think of a simple hash chain. Remember that each hash is depending on its predecessor and the actual data to be hashed. If you now create a kind of global hash chain where you submit everything you ever need to "sign", this can form a chain in itself. Now say you have a document (or log) x that you want to sign. You submit x to the chaining process and receive back a hash value h. Now you store that h, and use it as a signature, a proof of integrity of x. Now assume that there is a way that you give out x and h and someone can verify that x participated in the computation of the "global" log chain. If so, and if x's hash still matches the value that was used to generate h, than x is obviously untampered. If we further assume that there is a fixed schedule on which some "anchor hashes" are being produced, and assume that we can track to which such anchor hash h belongs to, we can even deduce at what time x was signed. In essence, this is what Guardtime does. They operate the server infrastructure that does this hashing and timestamping. The key hashes are generated once a second, so each signature can be tracked very precisely to the time it was generated. This is called "linked timestamping" and for a much better description than I have given just follow that link.

The key property from our point of view is that with this method, no secrets are required to sign log records. And if there is no secret, an attacker can obviously not take advantage of the secret to hide his tracks. So this method actually provides a very good integrity proof and does not create a false sense of security. This removed a major obstacle that always made me not like to implement log signatures.

The alert reader may now ask: "Doesn't that sound too good? Isn't there a loophole?". Of course, and as always in security, one can try to circumvent the mechanism. A very obvious attack is that an attacker may still go ahead and modify the log chain, re-hash it, and re-submit it to the signature process. Our implementation inside rsyslog already makes this "a bit" hard to do because we keep log chains across multiple log files and the tooling notifies users if a chain is re-started for some reason (there are some valid reasons). But the main answer against these types of attacks is that when a re-hashing happens, the new signature will bear a different timestamp. As such, if comparing the signature's timestamp and the log record timestamps, tampering can be indicated. In any case, we are sure we will see good questions and suggestions on how to improve my code. What makes us feel about the method itself is that it

- is based on well-known and proven technology (with the Merkle tree being one foundation)
- is in practical use for quite some while
- ample of academic papers exist on it
- there is a vital community driving it forward and enhancing the technology

So everything looks very solid and this is what triggered our decision to finally implement log signature directly into the rsyslog core. And to do it right, we did not create a KSI-specific interface but rather modelled a generic signature provider interface inside rsyslog. So if for whatever reason you don't like the KSI approach, there is a facility inside rsyslog that can be used to easily implement other signature providers. Due to the reasoning given above, however, we have "just" implemented the KSI one for now.

The original reaction to systemd journal claims

In this section, we reproduce Rainer Gerhards' initial response to the claims from the initial journal announcement. This is a copy from he blog. The posting contains many in-depth information and still is valid, so we thought it is a good addition to this paper, enabling to have all information in one place.

Note however that the posting dates back to end of 2011, so the new developments we describe in the paper above were obviously not know at that time.

In the paper introducing journald/Linux Journal a number of shortcomings in current syslog practice are mentioned. The authors say:

> *Syslog has been around for ~30 years, due to its simplicity and ubiquitousness it is an invaluable tool for administrators. However, the number of limitations are substantial, and over time they have started to be serious problems:*

I have now taken some time to look at each of these claims in depth. But before I start, I need to tell that I am working in the IT logging field for nearly 15 years, have participated in a number of standards efforts and written a lot of syslog-related software with rsyslog being a prime example (some commercial tools I have been involved with can be found here). So probably I have a bias and my words need to be taken with a grain of salt. On the other hand, the journald authors also have a bias, so I guess that's a fair exchange of arguments ;).

In my analysis, I compare the journald effort with what rsyslog currently provides and leave closed source software out. **It is also important to note that there is a difference between syslog, the protocol, a specific syslog application (like rsyslog) and a system log message store.** Due to tradition, these terms are often used for different things and one must deduce from context, what is meant. The paper applies the same sloppiness in regard to terms. I use best effort to extract the proper meaning. I quote the arguments as they originally appeared inside the paper. However, I rearrange them a bit in order to put related things closer together. I retain the original numbering so that you can compare to the original paper. I also tried to be similar brief with my arguments. Now proof-reading the post, I see that I failed with that. Sorry, but that's as brief as I can provide serious counterargument. I broadly try to classify arguments in various levels of "True" vs "Wrong", so you may take this as an ultra-short reply.

So let's start with Arguments related to the log storage system. In general, the paper is right that there is no real log storage system (like, for example, the Windows Event Log). Keeping logs only in sequential text files definitely has disadvantages. Syslog implementations like rsyslog or syslog-ng have somewhat addressed this by providing the ability to use databases as storage backends (the commercial syslog-ng fork also has a proprietary log store). This has some drawbacks as well. The paper proposes a new proprietary indexed syslog message store. I kind of like this idea, have even considered to write something like this as an optional component for rsyslog (but had no time yet to actually work on it). I am not convinced, though, that all systems necessarily need such a syslog storage subsystem.

With that said, now let's look at the individual arguments:

> *5. Reading log files is simple but very inefficient. Many key log operations have a complexity of O(n). Indexing is generally not available.*

True. It just needs to be said that many tools inside the tool chain only need sequential access. But those that need random access have to pay a big price. **Please note, however, that it is often only necessary to "tail" log files, that is act on the latest log entries. This can be done rather quickly even with text files.** I know both the problems and the capabilities, because Adiscon LogAnalyzer, in which I am involved, is a web-based analysis and reporting tool capable of working on log files. Paging is simple, but searching is slow with large files (we recommend databases if that is often required). Now that I write that, a funny fact is that one of the more important reasons for creating rsyslog was that we were unhappy with flat text files (see rsyslog history doc). And so I created a syslogd capable of writing to databases. Things seem to be a bit cyclic, though with a different spin ;)

> *8. Access control is non-existent. Unless manually scripted by the administrator a user either gets full access to the log files, or no access at all.*

Mostly True and hard to make any argument against this (except, of course, if you consider database back ends as log stores, but that's not the typical case).

> *10. Automatic rotation of log files is available, but less than ideal in most implementations: instead of watching disk usage continuously to enforce disk usage limits rotation is only attempted in fixed time intervals, thus leaving the door open to many DoS attacks.*

Partly True, at least in regard to current practice. Rsyslog, for example, can limit file sizes as they are written ("outchannel action"), but this feature is seldomly used and due to be replaced by a better one. The better one is partly implemented but received no priority because nobody in the community flagged this as an urgent requirement. As a side-note: Envision that journald intends to shrink the log and/or place stricter restrictions on rate-limiting when disk space begins to run low. If I were an attacker, I would simply begin to fill the disk then, and make journald swipe out the log store for me.

> *11. Rate limiting is available in some implementations, however,*

> *generally does not take the disk usage or service assignment into account, which is highly advisable.*

It needs to be said what "rate limiting" means. I guess it means preventing an application from spamming the logs with frequently repeated messages. This feature is available in rsyslog. It is right that disk usage is not taken into account (see comment above on implications). I don't know what "service assignment" means in this context, so I don't comment on that one. Rate limiting is more than run-away or spamming processes. It is a very complex issue. Rsyslog has output rate limiting as well, and much more is thinkable. But correct, current rate limiting looks at a number of factors but not the disk assignment. On the other hand, does that make sense, if e.g. a message is not even destined to go to the disk?

> *12. Compression in the log structure on disk is generally available but usually only as effect of rotation and has a negative effect on the already bad complexity behaviour of many key log operations.*

Partly True. Rsyslog supports writing in zip format for at least one and a half year (I am too lazy to check the ChangeLog). This provides huge savings for those that turn on the feature. Without doubt, logs compressed in this way are much harder to process in real-time.

> *7. Log files are easily manipulable by attackers, providing easy ways to hide attack information from the administrator*

Misleadingly True. If thinking of a local machine, only, this is true. However, all security best practices tell that it is far from a good idea to save logs on a machine that is publicly accessible. This is the reason that log messages are usually immediately sent do some back end system. It is right that this can not happen in some setup, especially very small ones.

My conclusion on the log store: there definitely is room for improvement. But why not improve it within the existing frameworks? Among others, this would have the advantage that existing methods could be used to decide what needs to be stored inside the log store. Usually, log contain noise events that administrators do not want to log at all, because of the overhead associated with them. The exists best practices for the existing tool chain on how to handle that.

Now on to the other detail topics:

> *1. The message data is generally not authenticated, every local process can claim to be Apache under PID 4711, and syslog will believe that and store it on disk.*
>
> *9. The meta data stored for log entries is limited, and lacking key bits of information, such as service name, audit session or monotonic timestamps.*

Mostly wrong. IMHO, both make up a single argument. At the suggestion of Lennart Poettering, rsyslog can force the pid inside the TAG to match the pid of the log message emitter - for quite a while now. It is also easy to add additional "trusted properties". I made an experimental implementation in rsyslog yesterday. It took a couple of hours and the code is available as part of rsyslog 5.9.4. As a side-note, the level of "trust" one wants to have in such properties needs to be defined - **for truly trusted trusted properties some serious cryptography is needed** (this is **not specified in the journald proposal nor currently implemented in rsyslog)**.

> *2. The data logged is very free-form. Automated log-analyzers need to parse human language strings to a) identify message types, and b) parse parameters from them. This results in regex horrors, and a steady need to play catch-up with upstream developers who might tweak the human language log strings in new versions of their software. Effectively, in a away, in order not to break user-applied regular expressions all log messages become ABI of the software generating them, which is usually not intended by the developer.*

Trivial (I can't commit myself to a "True" or "Wrong" on such a trivial finding). Finally, **the authors have managed to describe the log analysis problem as we currently face it.** This is not at all a syslog problem, it is problem of development discipline. For one, syslog has "solved" this issue with RFC5424 structured data. Use it and be happy (but, granted, the syslog() API currently is a bit problematic). The real problem is the missing discipline. Take, for example, the Windows Event Log. The journald proposal borrows heavily on its concepts. In Windows Event Log, there is a developer-assigned unique ID within the application's reserved namespace available. The combination of both app namespace (also automatically created) and ID together does exactly the same thing as the proposed UUID. In Windows Event Log, there are also "structured fields" available, but in the form of an array (this is a bit different from name-value pairs but far from totally different). This system has been in place since the earliest versions of Windows NT, more than 15 years ago. **So it would be a decent assumption that the problem described as a syslog problem does not exist in the Windows world, right (especially given the fact that Windows purposefully does not support syslog)? Well, have a look at the problems related to Windows log analysis: these are exactly the same!** I could also offer a myriad of other samples, like WELF, Apache Log Format, ... The bottom line is that developer discipline is not easy to achieve. And, among others, a taxonomy is actually needed to extract semantic meaning from the logged event. It probably is educating to read the FAQ for CEE, a standard currently in development that tries to somewhat solve the logging mess (wait a moment: before saying that CEE is a bunch of clueless morons, please have a look at the CEE Board Members first).

> *3. The timestamps generally do not carry timezone information, even though some newer specifications define support for it.*

Partly Wrong. High-Precision timestamps are **available for many years** and default in rsyslog. Unfortunately, many **distros have turned them off**, because they break existing tools. So in current practice this is a problem, but it could be solved by deleting **one line** in rsyslog.conf. And remember that if that causes trouble to some "vital" tool, journald will break that tool even more. Note that some distros, like Gentoo, already have enabled high precision timestamps.

> *4. Syslog is only one of many log systems on local machines. Separate logs are kept for utmp/wtmp, lastlog, audit, kernel logs, firmware logs, and a multitude of application-specific log formats. This is not only unnecessarily complex, but also hides the relation between the log entries in the various subsystems.*

Rhetorically True - but what why is that the failure of syslog? **In fact, this problem would not exist if developers had consistently used syslog.** So the problem is not rooted in syslog but rather in the fact that syslog is not being used. Lesson learned: even if standards exist, many developers simply ignore them (this is also an interesting argument in regard to problem number #2, think about it...).

> *13. Classic Syslog traditionally is not useful to handle early boot or late shutdown logging, even though recent improvements (for example in systemd) made this work.*

True - including that fact that **systemd already solved that problem**.

> *14. Binary data cannot be logged, which in some cases is essential (Examples: ATA SMART blobs or SCSI sense data, firmware dumps).*

Wrong, the short answer is: it can be logged, but must be properly encoded. In the IETF syslog working group we even increased the max message sizes for this reason (actually, there is no hard limit anymore).

The longer, and more correct, answer is that this is a long-standing discussion inside the logging world. Using that view, it is hard to say if the claim is true or false; it often even is argued like being a religion. Fact is that the current logging tool-set does not work well for binary data (even encoded). This is even the case for the Windows Event Log, which supports binary data. In my view, I think most logging experts lean towards the side that binary data should be avoided and, if unavoidable, must be encoded in a text-friendly way. A core problem with the usefulness of binary data is that it often is hard to decode, and even more to understand, on the non-native platform (remember that the system used during analysis is often not the system where the event was initially recorded).

> *6. The syslog network protocol is very simple, but also very limited. Since it generally supports only a push transfer model, and does not employ store-and-forward, problems such as Thundering Herd or packet loss severely hamper its use.*

Wrong, missing all improvement made in the past ten years. There is a new RFC series which supports TLS-secured reliable transmission of syslog messages and which permits to place fine-grain access control on who can talk with whom inside a relay chain. UDP syslog is still available and is so for good reason. I cannot dig into the details here, part of that reasoning is on the same grounds why we use audio more often over UDP than TCP. Using UDP syslog is strictly optional and there are few scenarios where it is actually needed. And, if so, the "problem" mentioned is actually a "solution" to a much more serious problem not even mentioned in the journald paper. For a glimpse at these problems, have a lock at my blog post on the "reliability problem". Also, store-and-foward is generally available in rsyslog via action queues, failover handling and a lot of other things. I admit that setting up a complex logging system, sometimes requires an expert. On the "loss issue", one may claim that I myself say that plain TCP syslog is not totally lossless. That claim is right, but the potential loss Window is relatively small. Also, one can use different protocols that solve the issue. In rsyslog, I introduced proprietary RELP for that very reason. There is also completely lossless RFC3195, which is a great protocol (but without future). It is supported by rsyslog, but only extremely few other projects implement it. The IETF (including me) assumes that RFC3195 is a failure - not from technical fact but in the sense that it was too far away from the usual logging practice to be picked up by enough folks. [Just to avoid mis-intepretation: contrary to RFC3195, RELP is well alive, well-accepted and in widespread use. It is only RFC3195 that is a failure.]

Concluding my remarks, I do not see anything so broken in syslog that it can only be fixed by a total replacement of technology. Right contrary, there is a rich tool set and expertise available. Existing solutions, especially in the open source world, are quite modular and can easily be extended. It is not even necessary to extend existing projects. A new log store, for example, could also be implemented by a new tool that imports a decent log format from stdin to a back end data store. That would be easily usable not only from rsyslog but from any other tool that is part of the current log tool chain. For example, it may immediately consume Apache or other application logs (of course, such a tool would require proper cryptography to be used for cryptographic tasks...). There is also need for a new logging API - the catch-all syslog() call is clearly insufficient (an interesting detail fact is that journald promises to retain syslog() as a first-class logging interface -- that means journald can solve none of the issues associated with that API, especially in regard to claim #2).

So extending existing applications, or writing new ones that tightly integrate into the existing tool-set is the right thing to do. One can view journald as such an extension. However, this extension is somewhat problematic as its design document tells that it intends to **replace** the whole logging system. Especially disturbing is that the reasoning, as outlined above, essentially boils down to a new log store and various well-known mostly political problems (with development discipline for structured formats right at the top of them). Finally, the proposal claims to provide more security, but fails to achieve at least the level that RFC5848 syslog is able to provide. Granted, rsyslog, for example, does not (yet) implement RFC5848. But why intends journald to implement some home-grown pseudo security system when a standard-based method designed by real crypto experts is available? I guess the same question can be applied to the reasoning for the journald project at large.

Let me conclude this posting with the same quote I started with:

> *Syslog has been around for ~30 years, due to its simplicity and ubiquitousness it is an invaluable tool for administrators. However, the number of limitations are substantial, and over time they have started to be serious problems:*

Mostly Wrong. But it is true that syslog is an invaluable tool,especially in heterogeneous environments.