SERPEND, a forensic utility for analysis of systemd logs

CCF project spring 2017

Arthur de Fluiter
afr480, 2536812
11432659

Joost Heitbrink
jhk810, 2517669
11432640

April 2, 2017

Abstract

Systemd is an init process used on a large amount of Linux machines, besides it's many roles in Linux, it stores all sorts of interesting information in log files. We present an extensible, cross platform tool/library for automated analysis of such files, called SERPEND. SERPEND can be used as standalone rule-based tool, or extended by importing in Python.

Introduction

With an increasing number of services and companies becoming dependent on computers, the need for proper forensic tools has increased exponentially over the last decade. Where previously hackers and malware were often just harmless jokes by curious teenagers, nowadays millions of dollars can be earned by engaging in computer fraud, breach and hacking. Being able to quickly ascertain a breach of a system can help in catching the perpetrator and bringing them to justice. Chief amongst the evidence that a computer can provide to prove an intrusion has taken place are system logs.

With the rise of more sophisticated server management software, a new system has enjoyed widespread success: systemd. Its approach to logging is very different than its predecessors, and modern forensic tools do not yet have a proper way of handling these logs for correct forensic analysis.

Thus, new tools are needed, not only for handling new approaches to log formats, but also in utilizing the new security primitives that these modern system management components provide, and transforming this data into presentable evidence that is able to hold both in the court of law and in a cross-examination.

Research question

When a Linux machine needs to be investigated, the logs can contain vital information. However due to the nature of what applications log, most of the information in system logs . An investigator might need to look at certain aspects like kernel warnings, information on devices that were plugged in, kernel module crashes, etc.

Another complicating factor is that installing Linux or systemd is not possible for a multitude of reasons, chief amongst them that several forensic frameworks are Microsoft Windows based. Having a tool available that works regardless of platform can significantly aid investigations.

Thus, our research question will be as follows:

Can we transform systemd's logs and controls into presentable evidence by automated analysis?

Init systems

Our research project focuses on systemd specifically, which is a so called init system. This system has a special task on Linux, since it is appropriated process identifier 1. Thus, the init system is

the first process the kernel spawns, and the last process to terminate before a system is shut down.

Being the first process the kernel spawns brings with it a slew of responsibilities. The most important of which is the responsibility of the init system to automatically become the parent of any orphaned processes in the system. The init system needs to properly take care of these orphans, because their original parents can no longer complete the *waitpid* system call, to allow a finished child process to be removed from the process table.

The init system thus becomes the parent of all orphan processes to ensure the processes shut down successfully and return their allocated resources back to the operating system.

While managing orphan processes is the chief responsibility of the init system, and while some lightweight init systems choose stop there, modern init systems such as SysVinit and Upstart have become more than just a process manager, taking on some system management tasks as well.

This is where systemd comes into play, which is a modern init system developed by Redhat Inc. Systemd took the Linux desktop world by storm, with almost every distribution shipping systemd by default.

Besides being a traditional init process, it is capable of much more, making it much more of a "system manager". It also poses a much stricter control than even SysVinit on services and system daemons.

This means that systemd has immense control over what happens on a system, and also logs almost any activity on a given system. Thus, being able parse and analyze this information is of enormous value in forensic investigations, where detailed logs can aid significantly in timeline construction.

Linux control groups

An important feature systemd utilizes is control groups. Linux control groups were a relatively recent addition[1] to the Linux kernel, and allows the system to create process groups much in the same way as virtual memory is viewed: a process inside a specific control group is unaware of anything happening outside of that group. This includes process identifiers, meaning that in two different groups two processes might exist which have the same process identifier. These processes are separated from each other and from the process perspective, the other process simply does not exist.

This prevents processes from taking up too much resources in the system, obfuscate or escape its parent by double-forking, or interfere with other processes on the system.

Since such fine grained and strict resource management is very desirable for both desktop and server systems, systemd makes heavy use of control groups to manage daemons and programs.

The added benefit of using control groups to monitor processes instead of just process identifiers is that a process which double-forks can no longer "escape" a process group or obfuscate its original parent, since the process is still contained within the control group it was created in.

The implication of this is that process hiding and tampering has become much harder, because while a process is aware of its own PID (through *getpid(2)*), a process cannot do the same for cgroups. Thus, a process cannot, through regular system interfaces, discover if it is in a cgroup or not, and even if it could, it cannot escape the control group.

systemd

In order to understand the power of systemd, we first show how systemd models daemons and processes on a system, after which we examine its logging facilities.

Systemd models the configuration and restrictions of daemons and programs on a systemd through so called *unit files*. A unit file describes a resource systemd can monitor, be it a file path,

[1] Feb 16 2011, see commit 023695d96ee06f36cf5014e286edcd623e9fb847, Linux kernel

timer, disk mount point, or daemon. The most interesting type of unit is the *service* subtype, which represents a family of processes providing a single service.

The distinction between services and processes is important, since a database might spawn several worker processes to better distribute workloads. Since these processes are still considered part of the same service, systemd groups all of these processes into a single cgroup.

For example, a MySQL database configuration could be described through a file called *mysql.service*. In the service file, the administrator of the system can set a multitude of options, not only restricting its access to system resources like memory and CPU time, but also its ability to connect to the network, certain ranges of IP addresses, and even which protocol the service can use for connecting to networks.

This allows a system administrator to really fine-tune the limits of an application to ensure that a misbehaving daemon or service, be it through error or malicious intent, is not capable of compromising the entire system.

Log file format

In contrast to SysVinit and Upstart, systemd elects not to write log messages in plain text to log files. Instead, systemd creates a key-value store of messages and hashes these log messages using SipHash. Afterwards, a log file may be compressed using a compression algorithm of the users choosing.

While this way of logging is more efficient in live log tracing and log analysis on a running machine, it complicates matters when systemd, and by extension, journald, are not available to decode the encoded logs.

However, an important point to notice with the new binary log format is that it is harder to manipulate. Regular syslog files are simple plain text files. Thus, to escape detection a hacker only needs to find the relevant offending log file or entry and remove the incriminating lines from the log. This can be done with very little effort and, unless the system administrator has added extra safety measures, is virtually undetectable.

Systemd's logs however, are as stated before, saved as a binary file. Thus, it already becomes significantly harder to modify the files, since journald will detect a change to the log and view the change as a corruption to the log. This will prompt journald to seal the log with a cryptographic hash, and rotate the log to storage. Thus, an attacker can no longer 'just' quickly edit a log file. Furthermore, journald will add extra data to log files that is not easily forged, like specific machine and kernel id's. The log format also explicitly prepends trusted fields with an underscore(_), which are considered not forgeable by an attacker. Furthermore, editing these fields will invalidate the hash value of that entry, and trying to add bogus entries to the log will also fail since the hash is computed using information that the attacker cannot glean from previous log entries or the system itself. Thus, an administrator can compare trusted fields with untrusted data and check for any discrepancies.

Another interesting feature for forensic experts is the possibility of BLOB (Binary Large OBject) embedding into log files. This allows for a mulitude of useful applications. For example, if an application recieves suspicious input or detects unauthorized accesses, it can generate a compressed file containing all the relevant information of the intrusion and simply 'log' that to journald. Journald will ensure the file will be properly appended to the log.

These properties make the systemd log format a valuable source of information in incident timeline reconstruction.

Log file internals

For both speed and convenience, log files are essentially loaded into memory as memory mapped object. That is to say that systemd asks the operating system through the `mmap` call, to map the entire file in to virtual memory. From there it can read/write to it as a normal memory area with

the changes reflected into the actual file. Not only allows this the operating system to fully optimize which parts of the files have to be in the memory, but it allows for easy embedding of structures into the file. (Offsets within the file can easily be used as pointers)

When it comes down to it, the log files are most of all a complicated store for entry-objects, which represent a log entry. Every entry in turn is a collection of key-value pairs, some of which are guaranteed by systemd to contain accurate data (for instance the timestamp at the time of logging). In figure we see this scheme at work. Besides the standard fields, and the common `MESSAGE` field, we have also the ability to add our own key value pairs to these files.

The file provides in 3 ways find entries for performance reasons. First of all there's EntryArray objects, which are a linked list of arrays pointing to all Entry objects in the file. As stated before, these Entry objects represent an actual log entry. Besides linked list pointers and offsets to the Data objects that represent the key-value pairs, they also store time stamps of the log entry.

Besides there are also 2 hash tables, the FieldHashTable and DataHashTable both of which are a simple arrays of fields and data objects respectively.

The FieldHashTable points to a number of FieldObjects (representing the keys of key-value pairs). The FieldObjects themselves point to a linked list of DataObjects having the field and next FieldObject in the linked list of the field hash table.

DataHashTable, similar to the FieldHashTable points to an array of DataObjects, which in turn are again parts of linked lists of nodes with the same key value pairs (or hash collisions). Every DataObject also contains the actual key-value pair of the entries.

Because of this, one can simply iterate over all the entries, look up all entries with a specific key, or look up all entries with a specific key-value pair.

Note: It is worth noting that in this structure, everything is hashed constantly, nearly all pointers to data structures are accompanied by hashes of the values of this structures. On top of this, a cryptographic hash (SHA-256) is generated over all content, which is stored in the TagObject (linked by the header). All of which makes manual editing of this file extremely complicated and easily detectable unless done without mistakes. (write privileges are by default only for root users).

systemd facilities

Systemd itself is a very powerful system management facility, but it is not designed as being a single entity. Systemd is tightly integrated with a number of useful system management components. A key takeaway is that each integrated component carefully logs events to journald on the system level. Thus, these events can show on a very low level what happened on a system. We will now describe the integrations that systemd has with these components, and what information these components handle and log.

coredumpd

Coredumpd is a daemon responsible for handling any core dumps generated by any given program. Coredumpd will ensure that rlimits of the system are enforced correctly and ensures that all core dumps are saved correctly.

Coredumpd can be configured to either save a core dump into a set directory, aggregating all core dumps of all users in that directory, or it can be configured to directly append the core dump to the log file of the event that generated that core dump.

nspawn

systemd-nspawn is a low-level facility for launching and running containers. Containerized applications still want to log or provide information about their status, and thus systemd-nspawn will make sure that any logging done in the container is correctly and transparently transferred to journald, allowing a system administrator to quickly inspect all containers running on a system, and identify and isolate misbehaving containers.

Container log messages have additional information appended to them, chiefly which container it originated from, which control group it is in, and its resource usage and permissions. This information is appended by journald at log time, the application is not notified in any way that it is running in a containerized environment.

networkd

Networkd is the networking daemon shipped with systemd, and responsible for network configuration management and related settings. Any connection settings and changes to them must be communicated to the networkd daemon, meaning that if an attacker wants to install a stealthy backdoor, the attacker will need to circumvent or otherwise trick networkd to prevent it from logging a change.

Note that networkd will not act as a gatekeeper or firewall, but it can detect and log any changes in the network configuration, again complicating an attacker's stealth capabilities.

resolved

Resolved is the network resolver shipped alongside systemd. It provides DNS/DNSSEC name resolution to local applications. Resolved provides hostname resolution to the system in three possible ways:

1. Directly through the API the resolved exposes on dbus. This allows daemons to asynchronously resolve hostnames.
2. through *getaddrinfo(3)* glibc library function. All DNS requests that use the *getaddrinfo(3)* function and its derivatives get routed to resolved.
3. through the loopback address *127.0.0.53*, which allows programs which cannot or will not rely on glibc or equvalent to still utilize resolved's capabilities.

resolved can be configured to blacklist or whitelist DNS servers through the regular *etc/resolv.conf* file.

logind

Logind manages, as its name suggests, all user sessions and resource limits. logind will ensure that multiple users on a system cannot interfere with other user sessions in any way, and that resource limits for each user are enforced.

Logind ensures that users and multiseat sessions cannot take the entire system down by creating a separate cgroup for each user that logs in. Thus, it is also possible to set per-user quotas and restrictions on user programs.

Logind's configuration can be edited while the system is operational using the loginctl tool. Utilizing this tool is restricted to the system admin, and can be used to quickly disable user programs or modify user restrictions.

serpend

We now present SERPEND, a tool that analyzes raw systemd log files and can filter entries depending on a set of user-defined rules, written in a snort-like syntax. Thus, the tool and format should feel familiar to snort users.

SERPEND comes with a set of predefined rules that detect the most common log messages that indicate possible suspicious behaviour. SERPEND can read additional rule files and parse log files according to the rules.

serpend rules

Serpend's rules are based on the snort network sniffer rule syntax, however IP addresses, ports and network abstractions don't really make sense for a system log analyser. Our current design looks as follows:

```
'alert' <pid> <uid> <gid> [ <msg> ] '(' <other filters> ')'
```

All filters (like `<pid>`) use a variety of patterns, from simple numeric/string matching, to numeric range based and regex based searches. All of which are provided as a means to give the developer of the rule files enough expressivity.

The `<msg>` allows for string interpolation in a perl-like way, with the fields as specified in the official documentation.

For example:

```
alert 0 * * "[$__REALTIME_STAMP] HIGH PRIORITY KERNEL MESSAGE: $MESSAGE" (PRIORITY:6)
```

Using these rule files one can easily construct different analysis tools for different purposes. For instance a rule-file can be custom made on the spot, or like with snort-rules, these rules can be spread around in the communities to detect a variety of intrusions / weird behaviour.

journald log file scalpel file

As an added bonus we also added a small scalpel[2] configuration, which is able to extract systemd log files directly from a disk image. Since investigators at times only have a binary blob (no folder structure), this allows them to still recover full logs. This is the first scalpel configuration to our knowledge able to extract systemd logs purely from a disk image. The filter is somewhat however limited however, due to the logs having variable size (by default depending on size of the harddrive) and lacking a standard final pattern (which scalpel usually relies on).

Related Work

Rule based analysis of log files has been done before by [1], but this research focusses more on generating profiles from misbehaving internal actors and users specifically. Our tool does not discriminate or weigh users any different, and is mainly concerned with finding possible anomalies. The interpretation of these anomalies is left to a third party, which can be another tool or a forensic investigator.

Discussion

SERPEND's current functionalities set itself firmly to be a helpful tool for forensic investigations on systems which have a lot of services running and where systemd is used to manage the entire system. However, while systemd can protect a great deal of the system, it can only do that on the assumption that itself or the kernel has not been compromised. Thus, any attacker with sufficient knowledge or a zero-day vulnerability can still subvert all security measures.

The key takeaway is that a sufficiently advanced attacker (primarily state sponsored actors) will very likely be able to defeat the security measures. However, such an effort would take an effort that only a nation state can sponsor, meaning that regular criminals will have a much harder time quickly compromising a system and exfiltrate sensitive data. And even in the event that such an exfiltration is successful, the log format and system still allows for efficient and quick analysis of the compromised accounts or services, meaning that incident response times can benefit significantly from this tool.

Another problem that might present itself is that systemd, though it has far reaching control, will only log so much by default. Systemd's true power of restriction and detection comes from an administrator properly configuring roles for the services running on a machine, and defining its realm of acceptable behavior. Thus, if the administrator configures the system to be lenient by default for services, systemd's effectiveness in detecting and containing violations will be greatly diminished, which in turn also greatly diminishes the log file analysis.

A waterproof logging system will border on impossibility with a root user having access to these files. Another problem with SERPEND is the interpretation of the evidence. SERPEND can only detect anomalies given a rule set, but it cannot check in any way if those rules actually make sense beyond the syntax. Thus, it still falls on the investigator to draw conclusions from the evidence SERPEND provides, as no tool can ever have or acquire the necessary context that is required in the court of law.

Future work

Network based analysis server

A possible extension of this system is to send logs over the network over a secure connection with on the other end an analysis server, which simply attempts to store the logs it receives and alerts the administrator of any suspicious behaviour [3].

This would provide in an extra target which would have to be taken over, but can be made robust by only accepting these log-packets on designated ports. Furthermore, the need to take down an extra target makes it increasingly likely that the attacker will at some point reveal itself, allowing an administrator to take action.

Advanced Integration and extensions

A big extension to this tool would be the usage of the Python-systemd API, which would allow the tool to not only be used for post-incident analysis, but also as a live monitoring tool and anomaly detector. Based on events in the system, SERPEND can feed back analyzed events to the systemd-API, allowing systemd to take appropriate action in limiting the intrusion.

This detection can be achieved by crafting a set of rules as invariants for a system [4], which could then be set up to allow live monitoring and invariant checking, allowing for quick and efficient incident response.

The existing grammar of SERPEND can be easily extended to support a wide variety of custom scenarios, since the original grammar is based on the widely used PYPARSING module. The program could even be extended to support arbitrary grammar files, allowing for powerful, custom-made grammars to be utilized.

A final enhancement that would benefit SERPEND is the addition of configuration file lookup. In the event that SERPEND finds a service which triggers one of its rules, serpend can look on the filesystem for the specific configuration file of that service, parse it and present a compressed summary to the forensic investigator, showing the limits of the service as set by the system administrator, and the violation detected in the log. This form of correlation can provide a lot of useful context when the configuration file provides more context as to why a certain service was restricted from certain resources.

Conclusion

In this paper, we have shown the extensive system monitoring capabilities systemd provides, and presented the forensic log analysis tool SERPEND. SERPEND is able to extract log messages based on rules provided either by the standard rule set or user-defined rules.

The capabilities of SERPEND are however tied to the limitations imposed by systemd, and by default, systemd only provides a minimal set of protection primitives. Thus, the effectiveness of SERPEND finding suspicious log entries and generating presentable evidence from that data hinges much more on the correct configuration of the system in question and the trust of the kernel not being compromised than SERPEND itself.

Thus, we answer our question to our research question, *Can we transform systemd's logs and controls into presentable evidence by automated analysis?*: No, we cannot.

We have seen that in order to present the raw data that log files provide as evidence, even when filtered by rules, still needs a qualified forensic investigator to provide the correct context to the discovered discrepancies in the analyzed log files. Thus, our tool can aid in the correct acquisition of the evidence and improving the data acquisition phase of a forensic investigation, but the automated translation to presentable evidence is beyond the scope of this program.

References

[1] T. Abraham and O. de Vel. Investigative profiling with computer forensic log data and association rules. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 11–18. IEEE, 2002.

[2] S. H. S. W. Brian Carrier, E. Saunders. Scalpel, sleuthkit. `https://github.com/sleuthkit/scalpel`, 2013.

[3] 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.

[4] T. Stallard and K. Levitt. Automated analysis for digital forensic science: Semantic integrity checking. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 160–167. IEEE, 2003.

Figure 1: journald log files.
Every arrow represents one or more pointers from an object to an object / multiple objects