« systemd has been a complete, utter, unmitigated success

8 July, 2025 | 1,782 words | 7 minute read time

The year is 2013 and I am *hopping mad*.

The systemd wars of the tenties were harsh and casualties were many

`systemd` is replacing my plaintext logs with a binary format and pumping steroids into `init` and it is *laughing* at me. The unix philosophy cries out: is this the end of Linux (or, as many are calling it, GNU plus Linux)?

The year is 2025 and I'm here to repent. Not only is `systemd` a worthy successor to traditional `init`, but I think that it deserves a defense for what it's done for the landscape – especially given the hostile reception it initially received (and somehow continues to receive? for some reason?). No software is perfect – except for TempleOS – but I think that systemd has largely been a success story and proven many dire forecasts wrong (including my own). I was wrong!

» The `init` Paleolithic

I hope that I don't need to whine about why the old status quo wasn't great – init scripts of varying quality with janky dependencies and wildly varying semantics were frustrating. It's sort of wild to me that I was working as a full-time software engineer during an era in which we were still writing bespoke *shell* scripts to orchestrate process management. "Lost" or unmanaged processes, the weirdness of `S99` -type directories for dependency ordering, and different interfaces into `/etc/init.d` scripts were all real problems.

/etc/init.d, uh, finds a way

During the **LINUX INIT WARS**, you could probably write an upstart, s6, or OpenRC init script that didn't have *too* many problems. But *even then* you're supporting a variety of service management configuration formats with slightly differing behaviors. I wrote services for all of these different init systems! And the experience wasn't super!

Many of the deficiencies of traditional service management are more obvious in hindsight. Whereas bare-bones `init` was mostly about handling and/or reaping orphaned processes, entrusting a systemd-based PID 1 is also big for sandboxing and dependency management. We haven't even talked about timers, sockets, or mounts, either.

» I Deprecated Your Mom

We don't need to re-tread in great detail the history of *how* we arrived here. But the *how* is part of the reason I think systemd worked out in the end.

Consider that the two primary ways that older init systems managed processes – either foregrounded or forked – were (and are!) fully supported modes. Modern systemd provides for more nuanced "I'm ready" signaling apart from "is the process alive" (via `Type=notify`), but this kind of backward compatability really helped bridge the legacy gap. The systemd authors even wrote generator code to help migrate old services.

I don't think the ini-style configuration format is a panacea (I like Dhall), but that's another olive branch from systemd authors to system administrators: it doesn't require a turing-complete configuration format or domain-specific language. You can generally understand what this means when you read it and how to change it:

Systemd

```
[Service]
Type=forking
```

Defaults matter and configuration languages matter, too. I appreciate that systemd chose one that is obvious.

I can cite other examples but the point I want to make is that systemd deliberately chose

- backward compatability,
- simple configuration paradigms,
- and to proactively support and help folks migrate.

Not every open source project chooses to take explicit steps to support old paths on the road to deprecation. Lennart, you sweetheart.

» Trust the Process

I don't just think that systemd is our newer, cooler Dad now that does previously-annoying things better, but that systemd also brought us good, *brand new* things.

» Won't Somebody Think of the Plaintext?

Logged logs logging loggily

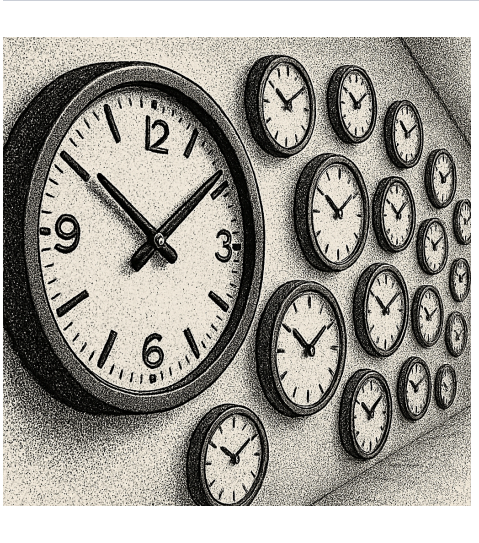
`journald` is here. Past Me hated it, too. The primary complaint with journald is that its journal files aren't in plaintext. Do I miss that? A little, yeah. I'm sort of a Linux boomer at heart and like to use `awk` for everything.

However, I *really* like having one place to send `stdout` and `stderr`! Have you ever leveraged custom fields when writing logs to the journal natively? I attach `NOTIFY_SLACK=1` to some of my services and listen to my lab's log stream for these events and forward them along to a Slack channel to see logs I want more easily, it's great!

Moreover, delegating the responsibility to journald is also convenient from a rotation and disk space perspective. With an awareness of filesystem space, I essentially never have to make rough guesses about rotation frequency any more, either[1]. Are you aware that part of the reason your journal files are in a binary format rather than plaintext because journald is compressing them transparently? That default choice is probably saving exabytes of space in aggregate across the entire computing space.

We can still live-tail logs, we can still forward log streams to different servers, and services can now reliably trust that their output will be captured during runtime. These are all just net Good Things.

» Time-r Out



I can still remember debugging `cron` scripts at my university job: was `$PATH` wrong? Should I `echo $USER` somewhere? Why am I emitting output to the *mail spool* by default???

If there's a candidate for "most legible over its predecessor", it might be the systemd timer system. Every Linux person feels some smug pride knowing what `0 0 * * *` means just by seeing a sequence of asterisks, but we all know `OnCalendar=daily` is easier to understand. Is `OnCalendar=minutely` a word? Not according to the grammar police, but you can probably infer what `minutely` means!

I could fill a blog post with things I love about systemd timers, so here's a list instead:

- `Persistent=true` is a great tool to ensure you don't miss timer executions.
- `systemctl list-timers` is an excellent way to see everything scheduled on a machine.
- The scheduling flexibility of `OnCalendar=` and `OnActiveSec=` are both powerful and easy to understand.

» Socket Activation

This *alone* is a hugely different and powerful way to optimize a system. `nix-daemon` leverages this to great effect by "lazily" running only when you need it: the daemon will stop when you aren't building anything, but as soon as you ask for it, `nix-daemon.socket` will start `nix-daemon.service`. That's a great feature!

True to form, systemd even provides the `systemd-socket-proxyd` executable to bridge the gap for services that may not speak the native protocol yet. I leverate this trick with heavy-handed daemons like Minecraft servers to great effect: I don't need to alter the original daemon at all, but `systemd-socket-proxyd` lets me leverage socket activation to run it on-demand anyway.

» A Fistful of Units

When you glue together the various unit types - `service`, `path`, `timer`, `mount`, `socket`, and so on - you can almost create a state machine out of your system. I've done this on NixOS and it's a powerful way to model interdependent service management.

Expressing system configuration like mounts as `mount` units lets you correctly order a daemon that needs a network mount to function. Triggering a service to restart when a file changes is easy with a `path` unit. The variety of options available to a `service` unit are mind-boggling and address almost every need you can think of. Seriously – did you know that `ConditionVirtualization=` can be used to run a unit depending on whether you're in AWS or Docker, for example? That's crazy.

» Security

If you've written a nontrivial number of `.service` units, then you know the options available for hardening services are vast in number. There are already many great blog posts about what they are; I won't go into that there.

Personally, my problem is **remembering what those options are**. Did you know that systemd built tools to help with that, too? **Each one of these** explains some operational security benefit you can wrap a daemon with and in most cases they're each easy to add and don't break functionality. These are a great way to take advantage of features like capabilities easily.

shell

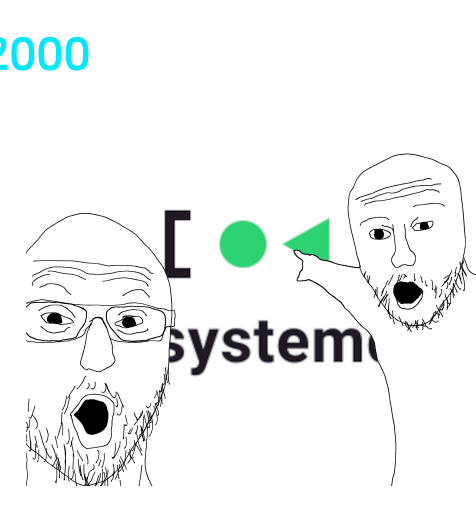
```
$ systemd-analyze security polkit.service
```

```
  NAME                                                        DESCRI
✓ SystemCallFilter=~@swap                                     System
✗ SystemCallFilter=~@resources                                System
✓ SystemCallFilter=~@reboot                                   System
✓ SystemCallFilter=~@raw-io                                   System
✗ SystemCallFilter=~@privileged                               System
✓ SystemCallFilter=~@obsolete                                 System
✓ SystemCallFilter=~@mount                                    System
✓ SystemCallFilter=~@module                                   System
✓ SystemCallFilter=~@debug                                    System
✓ SystemCallFilter=~@cpu-emulation                            System
✓ SystemCallFilter=~@clock                                    System
✓ RemoveIPC=                                                  Servic
✗ RootDirectory=/RootImage=                                   Servic
✓ User=/DynamicUser=                                          Servic
✓ RestrictRealtime=                                           Servic
✓ CapabilityBoundingSet=~CAP_SYS_TIME                         Servic
✓ NoNewPrivileges=                                            Servic
✓ AmbientCapabilities=                                        Servic
✓ CapabilityBoundingSet=~CAP_BPF                              Servic
✓ SystemCallArchitectures=                                    Servic
✗ CapabilityBoundingSet=~CAP_SET(UID|GID|PCAP)                Servic
✗ RestrictAddressFamilies=~AF_UNIX                            Servic
✓ ProtectSystem=                                              Servic
✓ SupplementaryGroups=                                        Servic
✓ CapabilityBoundingSet=~CAP_SYS_RAWIO                        Servic
✓ CapabilityBoundingSet=~CAP_SYS_PTRACE                       Servic
✓ CapabilityBoundingSet=~CAP_SYS_(NICE|RESOURCE)              Servic
✓ CapabilityBoundingSet=~CAP_NET_ADMIN                        Servic
✓ CapabilityBoundingSet=~CAP_NET_(BIND_SERVICE|BROADCAST|RAW) Servic
✓ CapabilityBoundingSet=~CAP_AUDIT_*                          Servic
✓ CapabilityBoundingSet=~CAP_SYS_ADMIN                        Servic
✓ PrivateNetwork=                                             Servic
✓ PrivateTmp=                                                 Servic
✓ CapabilityBoundingSet=~CAP_SYSLOG                           Servic
✓ ProtectHome=                                                Servic
✓ PrivateDevices=                                             Servic
✗ ProtectProc=                                                Servic
✗ ProcSubset=                                                 Servic
✗ PrivateUsers=                                               Servic
✗ DeviceAllow=                                                Servic
✓ KeyringMode=                                                Servic
✓ Delegate=                                                   Servic
✗ IPAddressDeny=                                              Servic
✓ NotifyAccess=                                               Servic
✓ ProtectClock=                                               Servic
✓ CapabilityBoundingSet=~CAP_SYS_PACCT                        Servic
✓ CapabilityBoundingSet=~CAP_KILL                             Servic
✓ ProtectKernelLogs=                                          Servic
✓ CapabilityBoundingSet=~CAP_WAKE_ALARM                       Servic
✓ CapabilityBoundingSet=~CAP_(DAC_*|FOWNER|IPC_OWNER)         Servic
✓ ProtectControlGroups=                                       Servic
✓ CapabilityBoundingSet=~CAP_LINUX_IMMUTABLE                  Servic
✓ CapabilityBoundingSet=~CAP_IPC_LOCK                         Servic
✓ ProtectKernelModules=                                       Servic
✓ CapabilityBoundingSet=~CAP_SYS_MODULE                       Servic
✓ CapabilityBoundingSet=~CAP_SYS_TTY_CONFIG                   Servic
✓ CapabilityBoundingSet=~CAP_SYS_BOOT                         Servic
✓ CapabilityBoundingSet=~CAP_SYS_CHROOT                       Servic
✓ PrivateMounts=                                              Servic
✓ CapabilityBoundingSet=~CAP_BLOCK_SUSPEND                    Servic
✓ MemoryDenyWriteExecute=                                     Servic
✓ RestrictNamespaces=~user                                    Servic
✓ RestrictNamespaces=~pid                                     Servic
✓ RestrictNamespaces=~net                                     Servic
✓ RestrictNamespaces=~uts                                     Servic
✓ RestrictNamespaces=~mnt                                     Servic
✓ CapabilityBoundingSet=~CAP_LEASE                            Servic
✓ CapabilityBoundingSet=~CAP_MKNOD                            Servic
✓ RestrictNamespaces=~cgroup                                  Servic
✓ RestrictNamespaces=~ipc                                     Servic
✓ ProtectHostname=                                            Servic
✓ CapabilityBoundingSet=~CAP_(CHOWN|FSETID|SETFCAP)           Servic
✓ LockPersonality=                                            Servic
✓ ProtectKernelTunables=                                      Servic
✓ RestrictAddressFamilies=~AF_PACKET                          Servic
✓ RestrictAddressFamilies=~AF_NETLINK                         Servic
✓ RestrictAddressFamilies=~…                                  Servic
✓ RestrictAddressFamilies=~AF_(INET|INET6)                    Servic
✓ CapabilityBoundingSet=~CAP_MAC_*                            Servic
✓ RestrictSUIDSGID=                                           SUID/S
✓ UMask=                                                      Files

→ Overall exposure level for polkit.service: 1.2 OK :-)
```

» Hater Sauce and The Terror From The Year 2000

Part of the reason I wrote this piece is that I keep stumbling onto threads like this:



> *i used to think that systemd was made the default and adopted by most distros because of its ease of use and the fact it supplied a whole bunch of things in one suite and i see where the appeal is in that but after switching to artix openrc, im just lost on why they decided to use systemd when openrc is objectively better when it comes to being an init system and for managing services, and all the other components of systemd suite can just be replaced, like why would they do this?*

Oh my god. Look, I respect that `stvpidcvnt111111` has a right to their opinion, but we can't let rhetoric with the intellectual weight of a mediocre fart waft into spaces as critical as computing infrastructure. Get your stench **outta here**.

I'm not going to argue with straw men here, but wait, I am actually:

> *systemd does too much.*

Have you considered that just "reaping old process IDs" wasn't *enough* responsibility for an init daemon on a secure, robust system? That maybe it should be protecting other parts of the system and tracking the liveness of a desired service?

> *systemd does a bad job*

If I see an argument like this then I can only assume the interlocutor doesn't do software engineering. Any sort of consistent experience using `systemctl` or `journalctl` will tell you otherwise. I've never even *heard* of systemd failing at its core responsibilities (starting, stopping, and managing daemons).

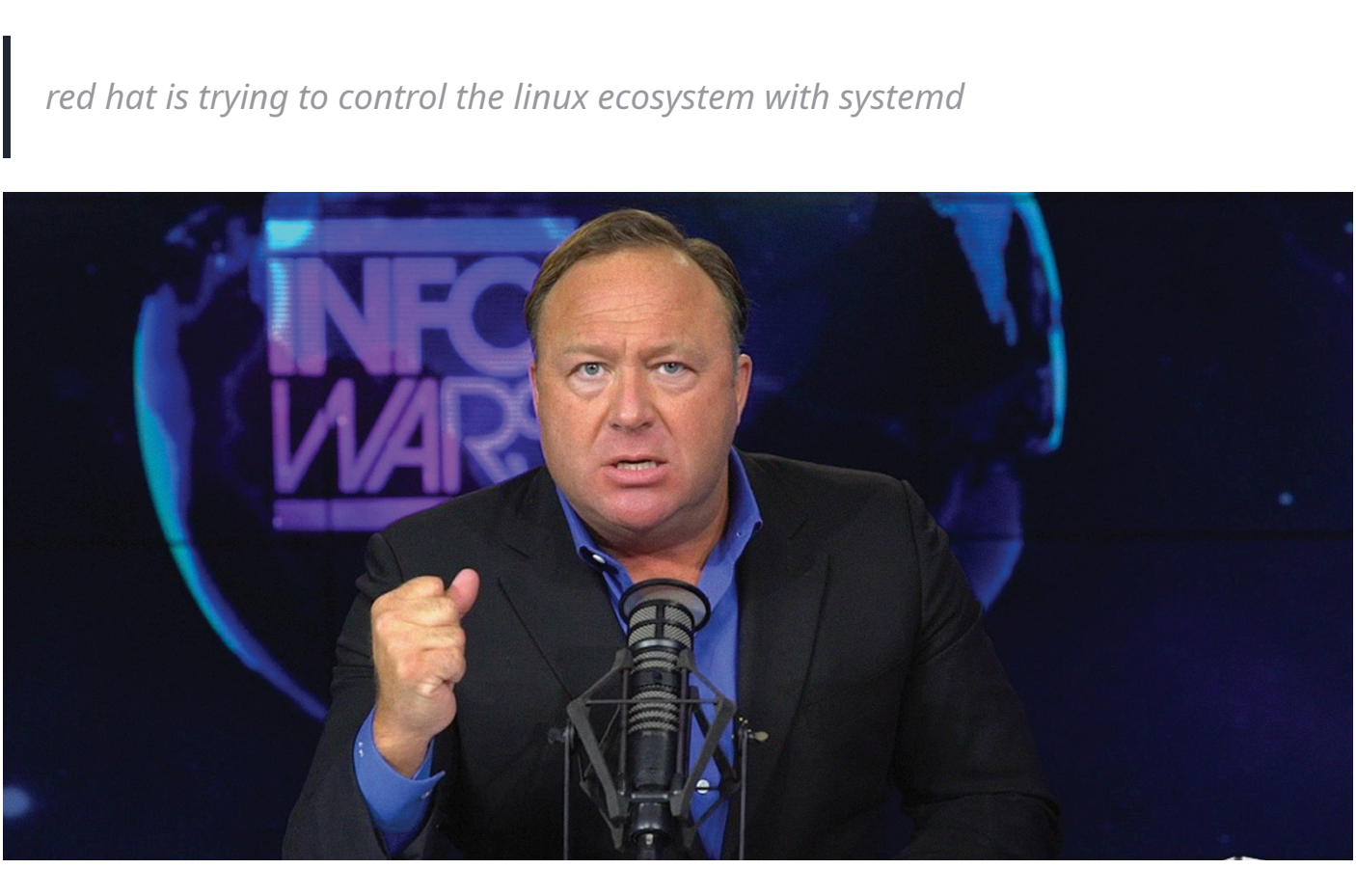
> *systemd is too bloated and tries to do too much*

For everything that modern systemd does, I'm shocked that there aren't more vulnerabilities (and yes, I'm aware of the CVEs that systemd does have). I have no hard numbers supporting this claim, but I do wonder what the delta is between "exploits due to systemd itself" against "exploits blocked by the service sandboxing that systemd provides" is. The ease of dropping an executable in an unprivileged environment is a great feature. The industry as a whole is almost assuredly safer with the accessibility to process sandboxing that systemd brought down to an easier level.

Yeah, `systemd-boot` and `systemd-networkd` do different things. Frankly, my life as an operator has been significantly *better* thanks to the quality of software that comes out of `systemd-*` based projects and they're all configured in similar ways, too. I've integrated at a low level with systemd APIs as well, so it's not as if this scary-sounding sprawl is *closed*, either. The APIs are there! You can use them!

I've consistently found myself *preferring* to use the systemd based alternatives like `systemd-resolved` and `systemd-networkd` when given the option because they end up being easier to configure and use.

> *red hat is trying to control the linux ecosystem with systemd*



This is absolutely true. I can't believe we, the **SYSTEMD GLOBALIST ILLUMINATI**, have been exposed.

Footnotes:

I know logrotate can do very intelligent things. But the configuration steps for journald to "print to stdout, done".

« The Human Resources Alignment Problem

Start Discussion 0 replies
