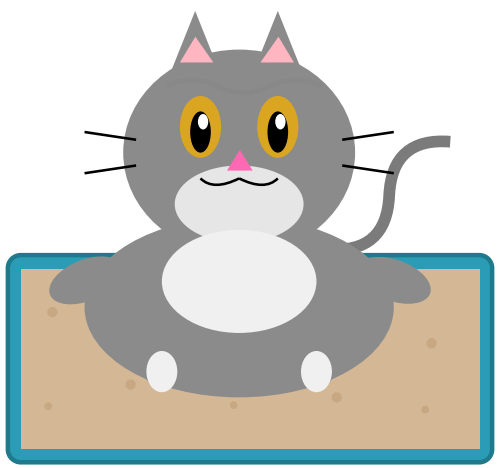
Litterbox

Somewhat Isolated Development Environments

Gerhard de Clercq - https://litterbox.work



Contents

The Problem: Implicit Trust

Supply chain attacks

- **npm install** - Thousands of dependencies, any could be compromised
- **VS Code extensions** - Full access to your home directory
- **cargo build** - Build scripts can do anything
- **Etc.**

The reality

Most development workflows implicitly trust hundreds of thousands of lines of code with full access to your home directory and SSH keys.

AI coding agents

- Agents can often run arbitrary shell commands
- Agents usually have/need internet access
- LLMs can easily be hijacked via prompt injection
- LLMs can easily get derailed all on their own
- Agent harnesses usually provide an unknown level of sandboxing

TL;DR

The scale of the risk is unclear, but there is great potential for disaster!

Where Existing Solutions Fall Short

What's out there?

Full VM:
- Strong isolation
- Heavy overhead
- Poor integration

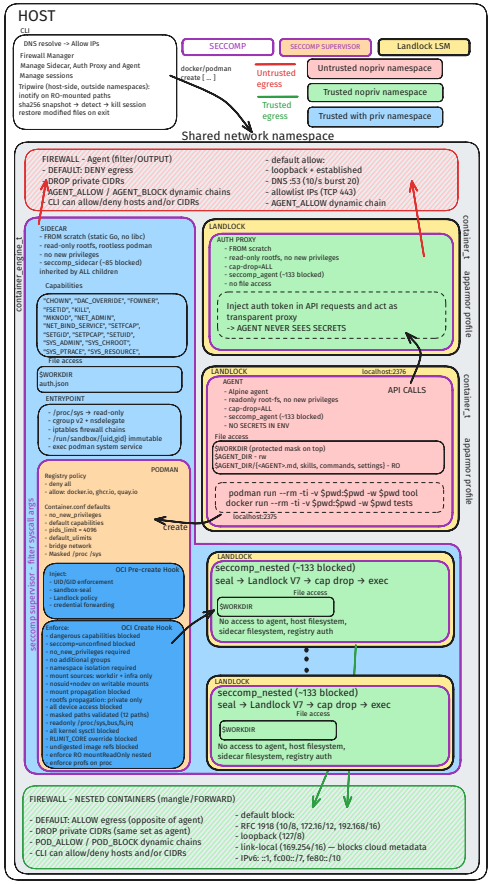
Dev Containers:
- Weak isolation
- Requires IDE Support
- Limited flexibility

Distrobox:
- No isolation
- Maximum convenience
- Seamless integration

Recent (complex looking) additions

- https://github.com/89luca89/clampdown
- https://github.com/NVIDIA/OpenShell

Complex looking?



https://github.com/89luca89/clampdown/blob/main/assets/clampdown-diagram.svg

What was missing?

A tool that provides:

- **Decent isolation** with minimal overhead
- Ability to run your **full IDE inside** the container
- Flexibility to work with **any editor**
- Protection for SSH keys while keeping them accessible
- Customizable access controls per environment
- Smooth integration with your host system

The Solution: Litterbox

One tool, two goals

SOME isolation:

- Restrict access to files
- Restrict access to devices
- Restrict access to the network
- Protect SSH keys

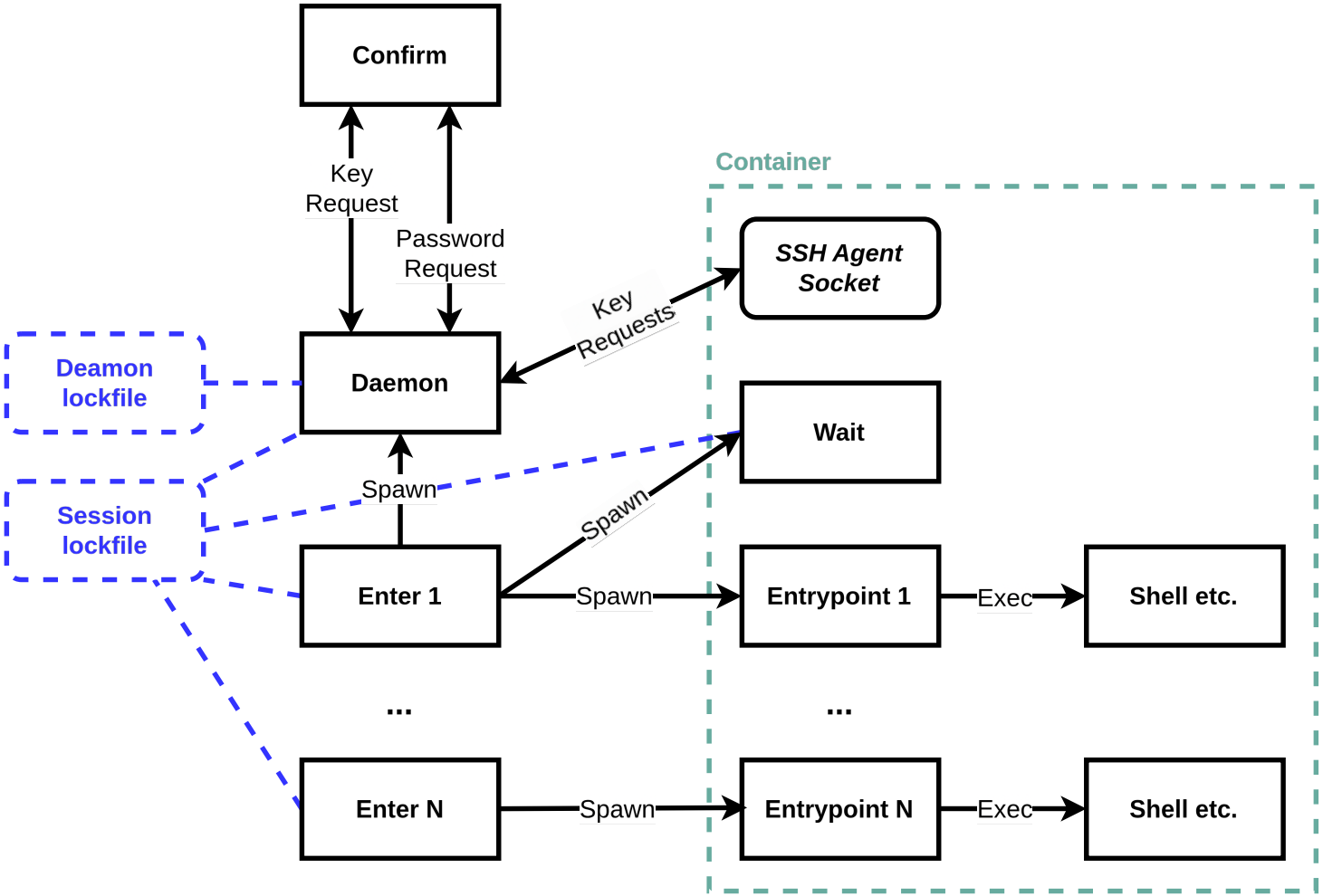
Reproducible environment:

- Documented setup and dependencies
- Documented permissions/settings
- Portable across machines
- Easily used in CI/CD

Demo time!

Internal Workings

Architecture



Landlock LSM integration

Benefits:

- Improved container isolation through proper access control
- Processes sandbox themselves - **no root required**
- Compatible with existing LSMs (AppArmor and SELinux)
- Processes control when the sandbox is applied
- Rich set of sandboxing capabilities that is growing

Challenges:

- Sudo no longer works after sandboxing

Limitations

Known vulnerabilities & holes

- **Kernel exploits** - Container shares host kernel
- **Wayland access** - Full access to compositor, clipboard
- **PipeWire** - Audio I/O, potential exploit vector
- **Podman** - Runtime vulnerabilities possible
- **Russh agent** - Not the most battle testsed SSH codebase
- **Device exposure** - Full access to attached devices
- **Network** - Limited network isolation by default
- **Rust crates** - Too many dependencies
- **And more!**

Litterbox is NOT for running known malware!

How I Use Litterbox Daily

My development enviroment

- Each project gets its own Litterbox with tailored dependencies
- Zed or VSCode run entirely inside the container
- Different base distros are used as needed
- Different hardware (such as serial ports) are attached as needed
- Tun/Tap possible without root for simulating network stacks
- Images built on Github Actions and shared with runners
  (see https://github.com/Low-Noise-Factory/Verilator-Rust-Litterbox)
- Probe-rs currently requires “remoting” via a Unix socket
- Sometimes I SSH into a remote machine

Get Involved

Help wanted!

Adding new features is not a high priority for me, but I'm happy to accept useful PRs at https://github.com/Gerharddc/litterbox!

Most urgent needs

- **Automed testing**
- **More maintainers**
- Better documentation
- Improved security/hardening
- Less dependencies
- MicroVM support would also be nice :)

Questions?