finbarr / **yolobox** Public

Code | Issues 1 | Pull requests | Actions | Projects | Security | Insights

1 Branch 1 Tag | Go to file | Go to file

finbarr and claude — Expand security model documenta…

ad77601 · 1 hour ago

.github/wor…	Fix release workfl…	yesterday
cmd/yolobox	Add low memory …	yesterday
.gitignore	Initial release of y…	4 days ago
CLAUDE.md	Add low memory …	yesterday
CONTRIB…	Initial release of y…	4 days ago
Dockerfile	Simplify PS1 pro…	yesterday
LICENSE	Initial release of y…	4 days ago
Makefile	Use buildx if avail…	4 days ago
README.…	Expand security …	1 hour ago
go.mod	Initial release of y…	4 days ago
go.sum	Initial release of y…	4 days ago
install.sh	Initial release of y…	4 days ago

About

Let your AI go full send. Your home directory stays home.

Readme
MIT license
Contributing
Activity
46 stars
0 watching
1 fork
Report repository

Releases 1

v0.1.0 - Initial Release Latest
yesterday

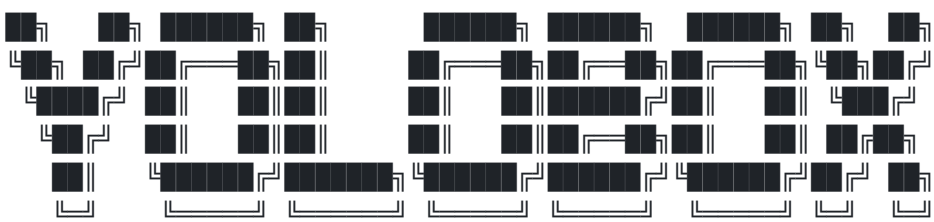
Packages 1

yolobox

Contributors 2

finbarr Finbarr Taylor

README | Contributing | MIT license



Let your AI go full send. Your home directory stays home.

Run Claude Code, Codex, or any AI coding agent in "yolo mode" without nuking your home directory.

The Problem

AI coding agents are incredibly powerful when you let them run commands without asking permission. But one misinterpreted prompt and `rm -rf ~` later, you're restoring from backup (yea right, as if you have backups lol).

The Solution

`yolobox` runs your AI agent inside a container where:

- ✅ Your **project directory** is mounted at `/workspace`
- ✅ The agent has **full permissions** and **sudo** inside the container
- ✅ Your **home directory is NOT mounted** (unless you explicitly opt in)
- ✅ Persistent volumes keep tools and configs across sessions

The AI can go absolutely wild inside the sandbox. Your actual home directory? Untouchable.

Quick Start

```
# Install (requires Go)
curl -fsSL https://raw.githubusercontent.com/finbarr/yolobox/master/install.sh | bash

# Or clone and build
git clone https://github.com/finbarr/yolobox.git
cd yolobox
make install
```

Then from any project:

```
cd /path/to/your/project
yolobox
```

You're now in a sandboxed shell. Run `claude` and let it rip.

What's in the Box?

The base image comes batteries-included:

- **AI CLIs**: Claude Code, Gemini CLI, OpenAI Codex (all aliased to run in full-auto mode!)
- **Node.js 22** + npm/yarn/pnpm
- **Python 3** + pip + venv
- **Build tools**: make, cmake, gcc
- **Git** + **GitHub CLI**
- **Common utilities**: ripgrep, fd, fzf, jq, vim

Need something else? You have sudo.

AI CLIs Run in YOLO Mode

Inside yolobox, the AI CLIs are aliased to skip all permission prompts:

Command	Expands to
`claude`	`claude --dangerously-skip-permissions`
`codex`	`codex --dangerously-bypass-approvals-and-sandbox`
`gemini`	`gemini --yolo`

No confirmations, no guardrails—just pure unfiltered AI, the way nature intended.

Commands

```
yolobox                    # Drop into interactive shell
yolobox run <cmd...>       # Run a single command
yolobox run claude         # Run Claude Code in sandbox
yolobox upgrade            # Update binary and pull latest image
yolobox config             # Show resolved configuration
yolobox reset --force      # Delete volumes (fresh start)
yolobox version            # Show version
yolobox help               # Show help
```

Flags

Flag	Description
`--runtime <name>`	Use `docker` or `podman`
`--image <name>`	Custom base image
`--mount <src:dst>`	Extra mount (repeatable)
`--env <KEY=val>`	Set environment variable (repeatable)
`--ssh-agent`	Forward SSH agent socket
`--no-network`	Disable network access
`--readonly-project`	Mount project read-only (outputs go to `/output`)
`--claude-config`	Copy host `~/.claude` config into container

Auto-Forwarded Environment Variables

These are automatically passed into the container if set:

- `ANTHROPIC_API_KEY`
- `OPENAI_API_KEY`
- `GITHUB_TOKEN` / `GH_TOKEN`
- `OPENROUTER_API_KEY`
- `GEMINI_API_KEY`

Configuration

Create `~/.config/yolobox/config.toml` for global defaults:

```
runtime = "docker"
image = "ghcr.io/finbarr/yolobox:latest"
ssh_agent = true
```

Or `.yolobox.toml` in your project for project-specific settings:

```
mounts = ["../shared-libs:/libs:ro"]
env = ["DEBUG=1"]
no_network = true
```

Priority: CLI flags > project config > global config > defaults.

> **Note:** Setting `claude_config = true` in your config will copy your host's Claude config on **every** container start, overwriting any changes made inside the container. Use the CLI flag `--claude-config` for one-time syncs.

Runtime Support

- **macOS**: Docker Desktop, OrbStack, or Colima
- **Linux**: Docker or Podman

> **Memory:** Claude Code needs **4GB+ RAM** allocated to Docker. Colima defaults to 2GB which will cause OOM kills. Increase with: `colima stop && colima start --memory 8`

Security Model

How It Works

yolobox uses **container isolation** (Docker or Podman) as its security boundary. When you run `yolobox`, it:

1. Starts a container with your project mounted at `/workspace`

2. Runs as user `yolo` with sudo access *inside* the container
3. Does NOT mount your home directory (unless explicitly requested)
4. Uses Linux namespaces to isolate the container's filesystem, process tree, and network

The AI agent has full root access *inside the container*, but the container's view of the filesystem is restricted to what yolobox explicitly mounts.

Trust Boundary

The trust boundary is the container runtime (Docker/Podman). This means:

- ✅ Protection against accidental `rm -rf ~` or credential theft
- ✅ Protection against most filesystem-based attacks
- ⚠️ **NOT protection against container escapes** — a sufficiently advanced exploit targeting kernel vulnerabilities could break out
- ⚠️ **NOT protection against a malicious AI** deliberately trying to escape — this is defense against accidents, not adversarial attacks

If you're worried about an AI actively trying to escape containment, you need VM-level isolation (see "Hardening Options" below).

Threat Model

What yolobox protects:

- Your home directory from accidental deletion
- Your SSH keys, credentials, and dotfiles
- Other projects on your machine
- Host system files and configurations

What yolobox does NOT protect:

- Your project directory (it's mounted read-write by default)
- Network access (use `--no-network` to disable)
- The container itself (the AI has root via sudo)
- Against kernel exploits or container escape vulnerabilities

Hardening Options

Level 1: Basic (default)

```
yolobox  # Standard container isolation
```

Level 2: Reduced attack surface

```
yolobox run --no-network --readonly-project claude
```

Level 3: Rootless Podman (recommended for security-conscious users)

```
# Install podman and run rootless
yolobox --runtime podman
```

Rootless Podman runs the container without root privileges on the host, using user namespaces. This significantly reduces the impact of container escapes since the container's "root" maps to your unprivileged user on the host.

Level 4: VM isolation (maximum security)

For true isolation with no shared kernel, consider running yolobox inside a VM:

- **macOS**: Use a Linux VM via UTM, Parallels, or Lima
- **Linux**: Use a Podman machine or dedicated VM

This adds significant overhead but eliminates kernel-level attack surface.

Network Isolation with Podman

For users who want to prevent container access to the local network while preserving internet access:

```
# Rootless podman uses slirp4netns by default, which provides
# network isolation from the host network
podman run --network=slirp4netns:allow_host_loopback=false ...
```

yolobox doesn't currently expose this as a flag, but you can achieve it by running rootless Podman (the default network mode for rootless is slirp4netns).

Building the Base Image

```
make image
```

This builds `yolobox/base:latest` locally.

Why "yolobox"?

Because you want to tell your AI agent "just do it" without consequences. YOLO, but in a box.

License