Vibe Coding

Scoping to Reduce Risks

Andras Iklody · CIRCL · Hackathon 2026

Tuesday, April 14 · Salle Europe

Previously on

"Vibe Coding Gone Wrong"

hack.lu 2025 · Call for Failures

The Draugnet Experiment

Building an app exclusively via ChatGPT prompting

The Draugnet Experiment

Building an app exclusively via ChatGPT prompting

→ Tiny context windows — AI forgets what it built 500 messages ago

The Draugnet Experiment

Building an app exclusively via ChatGPT prompting

→ Tiny context windows — AI forgets what it built 500 messages ago

→ No tool use — can't test, can't run, can't verify anything

The Draugnet Experiment

Building an app exclusively via ChatGPT prompting

→ Tiny context windows — AI forgets what it built 500 messages ago

→ No tool use — can't test, can't run, can't verify anything

→ The **80/20 trap** — first 80% felt like magic

The Draugnet Experiment

Building an app exclusively via ChatGPT prompting

→ Tiny context windows — AI forgets what it built 500 messages ago

→ No tool use — can't test, can't run, can't verify anything

→ The **80/20 trap** — first 80% felt like magic

→ Every view looked different, copy-paste everywhere, architecture decaying with each prompt

The Draugnet Experiment

Building an app exclusively via ChatGPT prompting

→ Tiny context windows — AI forgets what it built 500 messages ago

→ No tool use — can't test, can't run, can't verify anything

→ The **80/20 trap** — first 80% felt like magic

→ Every view looked different, copy-paste everywhere, architecture decaying with each prompt

→ End result: **took over and finished everything by hand**

"Vibe coding will replace us all

—

I was already looking into goat farming"

Fast Forward to 2026

→ Agentic coding arrived — tool use, persistent context, autonomous loops

→ Since then: **~30k LOC** of MISP tooling across Rust, PHP, Python — in weeks

→ Even **Draugnet itself** got a complete rework — backend rewritten, frontend overhauled, security hardened, CI added. A completely different ballgame with agentic AI

→ But we're building **security tooling** — "move fast and break things" is not always an option

Fast Forward to 2026

→ Agentic coding arrived — tool use, persistent context, autonomous loops

→ Since then: **~30k LOC** of MISP tooling across Rust, PHP, Python — in weeks

→ Even **Draugnet itself** got a complete rework — backend rewritten, frontend overhauled, security hardened, CI added. A completely different ballgame with agentic AI

→ But we're building **security tooling** — "move fast and break things" is not always an option

Threat Model Your Vibe Coding

The question isn't "should I vibe code?"

It's: **what's the blast radius if this goes wrong?**

The question isn't "should I vibe code?"

It's: **what's the blast radius if this goes wrong?**

That answer determines your workflow — not the other way around.

The Risk Spectrum

What you're touching	Blast radius	If a bad merge ships
Core platform HIGH	Thousands of orgs, globally	Data loss, security bypass
Library / API client MED	Downstream consumers	Silent incorrect behaviour
Internal tooling LOW	Your team	Fixable same day

The Risk Spectrum

What you're touching	Blast radius	If a bad merge ships
Core platform HIGH	Thousands of orgs, globally	Data loss, security bypass
Library / API client MED	Downstream consumers	Silent incorrect behaviour
Internal tooling LOW	Your team	Fixable same day

The mistake: using the **same workflow everywhere** — too loose on critical code, or too rigid on throwaway tooling

Compartmentalise

→ Don't build one monolith with vibe coding — **split into compartments**

→ Each compartment gets its own threat model, workflow, level of autonomy

→ The AI doesn't decide the risk level — **you do**, before writing the first prompt

1 Tight Scope, Full Control

For changes to critical production software

The Rules

- → **One focused task** at a time — no "while you're at it"
- → **Manual git commits** after reviewing every diff
- → Lean on **existing CI/test pipelines** — they're battle-tested
- → Human is the bottleneck **by design**

Example: Query Performance Tuning

Partner CSIRT reported slow tag-filtered attribute searches

Example: Query Performance Tuning

Partner CSIRT reported slow tag-filtered attribute searches

→ Built a **SearchPerformanceShell** diagnostic to harvest DB stats and evaluate query plans

Example: Query Performance Tuning

Partner CSIRT reported slow tag-filtered attribute searches

→ Built a **SearchPerformanceShell** diagnostic to harvest DB stats and evaluate query plans

→ Optimised query planner — switching between **correlated EXISTS** and **uncorrelated IN** subqueries based on filter selectivity

Example: Query Performance Tuning

Partner CSIRT reported slow tag-filtered attribute searches

→ Built a **SearchPerformanceShell** diagnostic to harvest DB stats and evaluate query plans

→ Optimised query planner — switching between **correlated EXISTS** and **uncorrelated IN** subqueries based on filter selectivity

→ Also caught a tag search regression — **silently broken for 5 months**

Example: Query Performance Tuning

Partner CSIRT reported slow tag-filtered attribute searches

→ Built a **SearchPerformanceShell** diagnostic to harvest DB stats and evaluate query plans

→ Optimised query planner — switching between **correlated EXISTS** and **uncorrelated IN** subqueries based on filter selectivity

→ Also caught a tag search regression — **silently broken for 5 months**

Small commits · existing CI · manual review · Claude wrote the code, I decided what shipped

Example: Geolocation & GPX Visualisation

Interactive maps for geolocation objects, GPX tracks, drone flight paths

Example: Geolocation & GPX Visualisation

Interactive maps for geolocation objects, GPX tracks, drone flight paths

→ Leaflet.js maps for 4 object types — including **UAV flight path rendering**

Example: Geolocation & GPX Visualisation

Interactive maps for geolocation objects, GPX tracks, drone flight paths

→ Leaflet.js maps for 4 object types — including **UAV flight path rendering**

→ ~700 lines across 7 commits — client-side parsing, CSP headers, bundled assets

Example: Geolocation & GPX Visualisation

Interactive maps for geolocation objects, GPX tracks, drone flight paths

→ Leaflet.js maps for 4 object types — including **UAV flight path rendering**

→ ~700 lines across 7 commits — client-side parsing, CSP headers, bundled assets

→ Shipped as a **plugin**: disabled by default, opt-in — not a surprise

Example: Geolocation & GPX Visualisation

Interactive maps for geolocation objects, GPX tracks, drone flight paths

→ Leaflet.js maps for 4 object types — including **UAV flight path rendering**

→ ~700 lines across 7 commits — client-side parsing, CSP headers, bundled assets

→ Shipped as a **plugin**: disabled by default, opt-in — not a surprise

Same rules: one feature at a time, manual review, existing CI catches the rest

Why This Works for Critical Code

→ Codebase has **15+ years** of conventions, ACL, audit logging

→ Claude can reason about the code — but has **no intuition for production consequences**

→ You're trading speed for safety — on critical software, **that's the right trade**

2 PRD-Driven Autonomous Loops

For greenfield tools where you control the full lifecycle

The Insight

→ New tools have no existing users, tests, or conventions to break

→ The blast radius is contained — you can afford **more autonomy**

→ But autonomy without structure = **Draugnet all over again**

The Insight

→ New tools have no existing users, tests, or conventions to break

→ The blast radius is contained — you can afford **more autonomy**

→ But autonomy without structure = **Draugnet all over again**

The **PRD** is what changed everything

The Insight

→ New tools have no existing users, tests, or conventions to break

→ The blast radius is contained — you can afford **more autonomy**

→ But autonomy without structure = **Draugnet all over again**

The **PRD** is what changed everything

Not a wish list — an architecture spec the AI re-reads every iteration

The Workflow

Plan → PRD → Loop → Review → Test

The Workflow

Plan → PRD → Loop → Review → Test

→ **Plan:** conversation about goals, constraints, architecture

The Workflow

Plan → PRD → Loop → Review → Test

→ **Plan:** conversation about goals, constraints, architecture

→ **PRD:** detailed spec — the contract that prevents architectural drift

The Workflow

Plan → PRD → Loop → Review → Test

→ **Plan:** conversation about goals, constraints, architecture

→ **PRD:** detailed spec — the contract that prevents architectural drift

→ **Loop:** autonomous iterations, one task per cycle, atomic commits

The Workflow

Plan → PRD → Loop → Review → Test

→ **Plan:** conversation about goals, constraints, architecture

→ **PRD:** detailed spec — the contract that prevents architectural drift

→ **Loop:** autonomous iterations, one task per cycle, atomic commits

→ **Review:** human-in-the-middle — you steer, the AI builds

The Workflow

Plan → PRD → Loop → Review → Test

→ **Plan:** conversation about goals, constraints, architecture

→ **PRD:** detailed spec — the contract that prevents architectural drift

→ **Loop:** autonomous iterations, one task per cycle, atomic commits

→ **Review:** human-in-the-middle — you steer, the AI builds

→ **Test:** baked into every iteration, not a phase at the end

Autonomous Looping in Practice

```
#!/bin/bash
# ralph-once.sh — one task, supervised
claude --permission-mode acceptEdits \
  "@PRD.md @progress.txt \
  1. Read the PRD and progress file. \
  2. Find the next incomplete task and implement it. \
  3. Commit your changes. \
  4. Update progress.txt with what you did. \
  ONLY DO ONE TASK AT A TIME."
```

Autonomous Looping in Practice

```
#!/bin/bash
# ralph-once.sh — one task, supervised
claude --permission-mode acceptEdits \
  "@PRD.md @progress.txt \
  1. Read the PRD and progress file. \
  2. Find the next incomplete task and implement it. \
  3. Commit your changes. \
  4. Update progress.txt with what you did. \
  ONLY DO ONE TASK AT A TIME."
```

Also: `afk-ralph.sh N` — runs N iterations autonomously with early exit on PRD completion

Autonomous Looping in Practice

```
#!/bin/bash
# ralph-once.sh — one task, supervised
claude --permission-mode acceptEdits \
  "@PRD.md @progress.txt \
  1. Read the PRD and progress file. \
  2. Find the next incomplete task and implement it. \
  3. Commit your changes. \
  4. Update progress.txt with what you did. \
  ONLY DO ONE TASK AT A TIME."
```

Also: `afk-ralph.sh N` — runs N iterations autonomously with early exit on PRD completion

Each iteration: implement → test → commit → update progress → next

Human-in-the-Middle (HITM)

→ After autonomous loops — review **accumulated commits**, not every keystroke

→ Interactive sessions to fix issues, adjust direction, handle edge cases

→ The AI builds — **you decide what survives**

Human-in-the-Middle (HITM)

→ After autonomous loops — review **accumulated commits**, not every keystroke

→ Interactive sessions to fix issues, adjust direction, handle edge cases

→ The AI builds — **you decide what survives**

This is the key difference from Draugnet: the human stays in the loop,
but at the *right* level of abstraction

Testing: Continuous, Not Eventually

→ Tests are **milestones in the PRD**, not a phase at the end

→ Each loop iteration must **pass tests before committing**

→ Reuse existing infrastructure where possible — e.g. validating a Rust client against the Python test suite it's replacing

→ Some projects ended up with **more test code than app code** — that's a feature

3 Sandbox Your Autonomous AI

Least privilege — for AI agents too

Two Layers of Containment

Container Isolation

```
docker sandbox run claude \
  . ~/.claude:ro -- \
  --permission-mode acceptEdits \
  -p "@PRD.md ..."
```

→ Only the repo is mounted

→ No host secrets or network

→ Config is read-only

Permission Allowlists

```
{
  "allowedCommands": [
    "cargo build",
    "cargo test",
    "cargo clippy"
  ]
}
```

→ Per-project curated commands

→ Everything else blocked

→ No escalation possible

The Principle

You wouldn't give someone **write access to your repo** after one pull request

The Principle

You wouldn't give someone **write access to your repo** after one pull request

Don't give it to your **AI agent** either

The Principle

You wouldn't give someone **write access to your repo** after one pull request

Don't give it to your **AI agent** either

The sandbox cost is near zero.
The cost of an autonomous agent running `curl` against production
with an API key it found in the environment... is not.

4 Make the AI Audit Itself

Then audit it yourself

Targeted Security Prompts

→ Don't say "find bugs" — say **"search for XSS in all template rendering"**

→ Specific vulnerability classes: XSS, path traversal, SQLi, supply chain

Targeted Security Prompts

→ Don't say "find bugs" — say **"search for XSS in all template rendering"**

→ Specific vulnerability classes: XSS, path traversal, SQLi, supply chain

This caught real issues:

→ **XSS** in template rendering — found and fixed

→ Led to **25 path traversal tests** in file I/O

→ CDN dependencies flagged as supply chain risk — **vendored all JS locally**

Targeted Security Prompts

→ Don't say "find bugs" — say **"search for XSS in all template rendering"**

→ Specific vulnerability classes: XSS, path traversal, SQLi, supply chain

This caught real issues:

→ **XSS** in template rendering — found and fixed

→ Led to **25 path traversal tests** in file I/O

→ CDN dependencies flagged as supply chain risk — **vendored all JS locally**

Scale Reviews to the Threat Model

LOW Internal tooling

→ AI self-review + quick manual check

HIGH Widely-used platform

→ AI self-review + thorough manual review + CI + peer review

Scale Reviews to the Threat Model

LOW Internal tooling

→ AI self-review + quick manual check

HIGH Widely-used platform

→ AI self-review + thorough manual review + CI + peer review

The review process scales with the threat model — just like everything else

What Changed Since Draugnet

Not Just Better AI — Better Process

Not Just Better AI — Better Process

→ **Persistent context + tool use** solved the amnesia and the 80/20 trap

Not Just Better AI — Better Process

→ **Persistent context + tool use** solved the amnesia and the 80/20 trap

→ **PRDs** solved architectural drift

Not Just Better AI — Better Process

→ **Persistent context + tool use** solved the amnesia and the 80/20 trap

→ **PRDs** solved architectural drift

→ **Sandboxing** solved the trust problem

Not Just Better AI — Better Process

→ **Persistent context + tool use** solved the amnesia and the 80/20 trap

→ **PRDs** solved architectural drift

→ **Sandboxing** solved the trust problem

→ **HITM** solved the "who's actually in charge" problem

Takeaways

Threat model your vibe coding — blast radius determines workflow

Compartmentalise — different projects get different autonomy

PRDs are guardrails — they make autonomy productive, not chaotic

Test continuously — bake it into every iteration

Sandbox & allowlist — least privilege applies to AI too

AI audits itself — but a human always reviews last

Vibe coding won't replace software engineers

Vibe coding won't replace software engineers

But engineers who know how to **scope, plan, and compartmentalise** will build things that were previously out of reach

Vibe coding won't replace software engineers

But engineers who know how to **scope, plan, and compartmentalise** will build things that were previously out of reach

Vibe coding won't replace software engineers

But engineers who know how to **scope, plan, and compartmentalise** will build things that were previously out of reach

The goats will have to wait.
Perhaps indefinitely.

Thanks!

Andras Iklody · **@iglocska**
CIRCL · MISP Project