

CHICAGO POINTS TRAVEL FESTIVAL 2026 · MASTER CLASS · LEAVE-BEHIND SLIDES

Build the tools that work for you.

How I used AI Assistants, AI coding and API automation to build my own travel hacking knowledge database and answer engine; the pitfalls to avoid and lessons learned throughout the process.

The one-page cheat sheet is on slide 18.

WHAT'S POSSIBLE

INSPIRE

SHARE

FROM

Too busy, I can't keep up!

TO

Ask a question and trust the answer

PRESENTER



Carlos Ortega

Restock Bot Alerts

LESSONS

What I learned along the way

My mental model, best practices and pitfalls

LESSONS

I learned the hard way

TIPS

Best practices that boost
your productivity

GOTCHAS

Pitfalls to avoid



Trust it. Rely on it. Afford it.

Trust

Can I trust it?

Fails as: wrong or made-up output, bad skills, too much access

I built: skill vetting, hardening, limited access

Reliable

Will it keep working?

Fails as: silent failures, nobody told me

I built: alerts, post-mortems, edge cases

Efficient

Is it worth the cost?

Fails as: surprise bills, repeated work

I built: cost tracking, cheaper defaults, shared context



TRUST

Make it trustworthy



Vet it

Scan before I trust



Harden it

Stop, don't guess



Limit access

Earn it step by step

THE PAYOFF

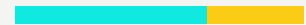
Hand work off at night.
Wake up to it done, safe
and checked.

THE HOW

Vet it, harden it, limit its
access

BY THE END

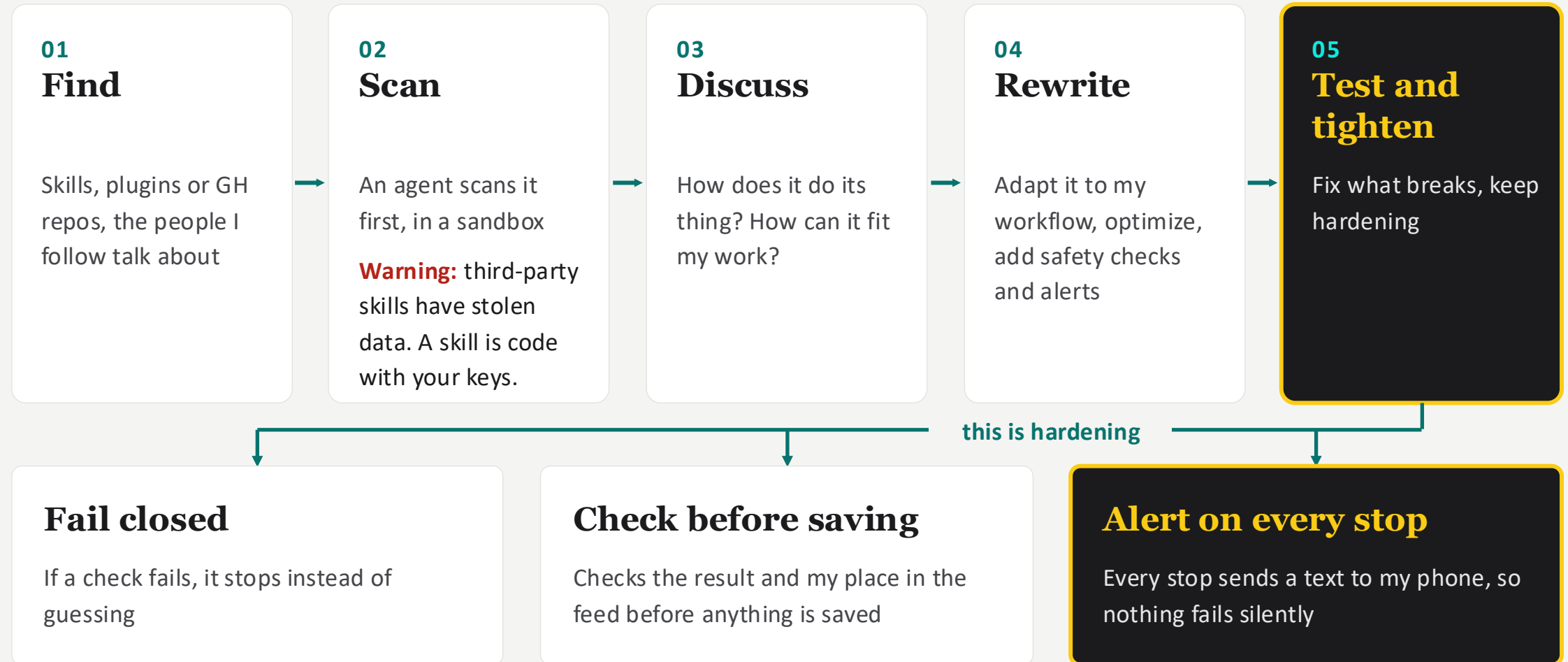
Trust is earned, not granted



The steps I take so I can trust my agents' work



Borrow it, then make it mine



Access is earned, not granted

01

Start narrow, widen only when needed

I gave an agent my email for one job, then took the access away. I don't want unlimited access to my inbox.

02

A sub-vault, not the whole vault

1Password is connected to Claude Code and Hermes through a separate vault. It holds only the logins the agents need.

03

Some things stay off

For now: credit cards, bank accounts and health data.

Before you give a new assistant your data

Meta's Muse and OpenAI Dots launched in fall 2026.

- Who is behind it, and do I trust them?
- Have they been hacked, and how did they respond?
- What have they done with customers' data before?

Trust is built **in small steps.**



What an agent reads can give it orders

Hidden text in web pages and emails can tell an AI to reveal its secrets or passwords.

GOAL: Break the rules

Jailbreak commands, a structured template, a fake system command

GOAL: Steal data

A “free association” game; a flood to expose how it’s built

GOAL: Run up the bill

Tokenade, a flood hidden as an emoji; siege, many floods at once

WHAT I DO

Use a strong model

Stronger models resist these tricks better.

Not immune

Better is not safe. Pair it with narrow access, so a tricked agent has little to steal.

All of them failed in this test. Attackers only need one to work.

Source: Matthew Berman, “I was hacked...”, YouTube. Red-team test of a live OpenClaw agent.



Make it reliable



Reliable means it keeps on working, not just on day one

THE PROBLEM

Worked on day one, then failed

THE HOW

Add a check for every failure

IF IT FAILS

Make it fail loudly

BY THE END

It keeps working as expected



Three things broke and nothing told me

01

Flaky tools

The built-in web tools got blocked or rate-limited. I had picked free over reliable.

02

Silent disconnects

Logins to Portal and Codex dropped, and tasks failed with no message.

03

61 days of silence

Jul 24 to Sep 22. My safety rule refused to push over unsaved changes, 800 articles piled up, and nothing told me.

Same cause every time: **nothing told me.**

DO Assume things will fail. **Make them fail loudly.**



I built something to watch it

FOR HERMES Watchdog

Every 6 hours it checks Firecrawl and every Codex account, then alerts Telegram and iMessage.

FOR THE DIGEST Failure alert

Every 30 minutes it looks for a missed or failed digest, texts me, and sends a note when fixed.

FOR THE WIKI Stall alarm

Added after the 61 days. The weekly check now flags a wiki update that stopped.

FOR THE DIGEST It paid off

Sep 24: Firecrawl and both Codex accounts went down. The browser fallback kept the digest running and the watchdog told me.

Making something reliable often means **building something to watch it.**

⚠️ Hermes provider health alert

- Nous/managed Firecrawl: DOWN (re-login or service repair needed)
- Direct Firecrawl backup: healthy
- OpenAI Codex: healthy
- This alert will repeat every 6 hours until all primary paths are healthy.



Post-mortem: every failure became a rule

01

Unsaved folder

One unsaved change stopped the nightly update for 61 days. Fix: an alarm when ingest stops.

02

Lost its place

If its record of where it stopped goes blank or gets lost, it could re-read all the posts. Fix: it waits instead.

03

Page bloat

14 wiki pages grew past 4,000 words. Fix: a size cap trims them nightly.

04

Full disk

AI-written tests filled 42 GB with temp folders. Fix: keep them only when a test fails.

When something fails: **find the cause, fix it, add a rule.**



Push it to its limits before it breaks

01

Source fails

What if a feed is down,
or returns a blank page?

02

10x the load

What if ten times the
articles arrive in one
night?

03

Keeps growing

What if a page, a folder
or the disk never stops
growing?

04

Login expires

What if a login drops at
3 a.m. and nobody is
watching?

Break it on purpose, **so real use never gets close.**



COST

Keep an eye on the meter



Cost



Tokens



Time

THE PROBLEM

Reliable isn't the same as cheap

THE HOW

Track cost, tokens and time per task -> Optimize

BY THE END

No surprise bills

Reliable doesn't mean cost-effective



One default, five times the cost

01

What I found

Claude Code turned on extended thinking for Haiku by default.

02

What it did

About \$0.027 and 48 seconds per call, against \$0.005 and 4.5 seconds with it off.

03

What I do now

Check defaults before a big run, and track cost, tokens and time for every task. The full run cost \$20.56, not about \$104.

Turn on thinking mode **only when you truly need it.**



Read once, not once per agent

01

What it did

Each agent read the whole codebase, found where to change, then made the change.

02

What it cost

Several agents on the same files repeated all that reading. Very costly and very slow.

03

The fix

One agent reads and researches once, then hands the context to the orchestrator, which sends sub-agents to do exactly their part.

Do the reading once, **then share the context.**



Stop being the messenger between agents

01

What I did

Two sessions in one codebase could conflict. I copied messages between them by hand, so I had to be there.

02

Between sessions

Claude Code agents can message each other. Point one at the other's session name and tell it to coordinate.

03

Between tools

Claude and Hermes both work on the wiki.
A shared log holds their questions and statements. Hermes watches it and replies, without me in the middle.

Let agents coordinate. **Save your time for the decisions.**



YOUR TURN

The repeatable recipe

01

Name the problem

What do I keep redoing?

02

List your tools

What do I already have?

03

Split into jobs

One job per step

04

Give each job a tool

Find the best, most reliable and cost-effective tool

05

Harden the skills

Read, rewrite, test

06

Automate it, fail loudly

A schedule or a text message, and an alert on every failure

07

Measure and fix

Track cost, tokens and time

Name it. Split it. Give each job a tool. Then measure. Rinse and repeat.



Do's, don'ts and warnings

DO

- Stay curious: ask how it does the job
- Judge options on quality, not price
- Add a fallback for your main method
- Make it say back what it understood
- Track consumption to catch runaway prompts
- Set a size cap on day one
- Keep a human on big changes
- Use thinking mode only when needed
- Assume it will fail. Make it fail loudly.
- Use a strong model for web and email

DON'T

- Trust it because it works
- Pick the free option before you know its downsides
- Leave repeatable decisions to an open prompt
- Let a safe stop go silent

WARNING

- Agents can go very wrong on their own
- Open prompts get read differently each run
- Agents leave things behind: tests filled 42 GB of disk
- A skill is code with your keys: third-party skills have stolen data
- Hidden instructions in emails and web pages can tell an AI to reveal secrets



THANK YOU · CHICAGO POINTS TRAVEL FESTIVAL 2026

Thank you.

Questions, ideas, or your own build story? I'd like to hear them.

Free to share with credit.

WEBSITE

restockbotalerts.com

EMAIL

carlos@tremscol.com

PRESENTER



Carlos Ortega

Restock Bot Alerts