Lambda Twelve

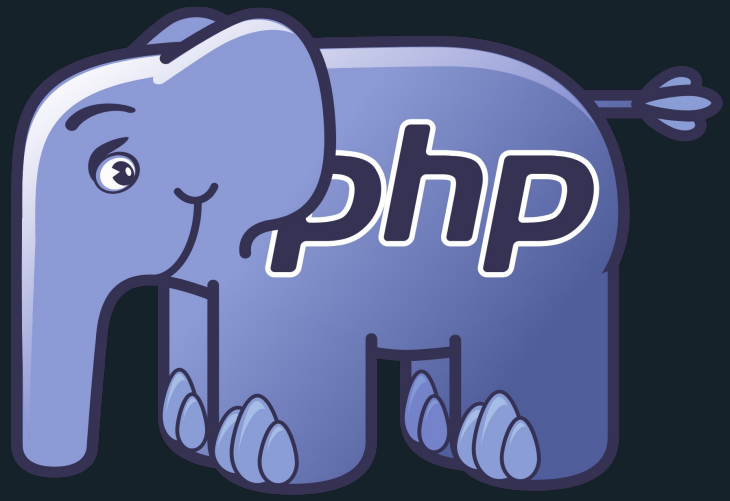
I asked AI to build an open-source security tool. Then I asked AI to break it.

Nick Andriopoulos, Lambda Twelve

FOSSCOMM 2026, University of West Attica, Athens

Before we start...

Let’s address the elePHPant in the room.



PHP2Shell

`in plain words` Two bugs in PHP, the language behind most of the web. Anyone could run their own code on your server, no password needed.

Hands up: who writes software for a living?

Hands up: who spent quality time with PHP servers these past two weeks?

Who am I?

1987

Writing software for fun.

2001

Writing software for money.

today

Running IT.
CISO, so that things don't catch fire.
Lambda Twelve, because apparently I needed another job.

I have spent enough time creating bugs, and being responsible for other people's bugs, to be suspicious of magic.

OWASP





owasp.org

Open Worldwide Application Security Project. A non-profit of volunteers, since 2001. Everything it makes is free and open: the Top 10, guides, tools.

OWASP Athens

The Athens chapter is coming back.



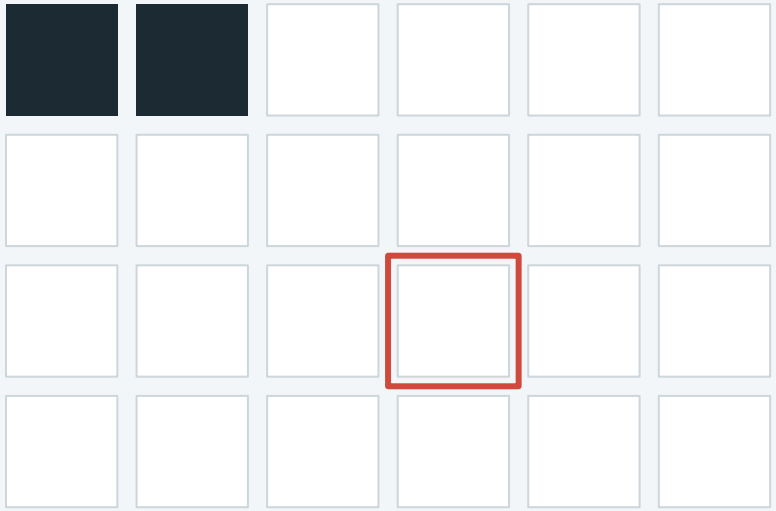
l12.gr/owasp

We are relaunching it, and OWASP is processing the reactivation. If you want to be in the room for the first meeting, or help run the chapter, leave your name.

`in plain words` A volunteer community that gives away security know-how. Athens had a chapter; it went quiet; a few of us are bringing it back.

The problem it was solving

your application



CVE Lite CLI

An OWASP project for the JavaScript world. It scans the libraries your project uses, names the ones with known holes, and hands you the exact command that fixes each.

github.com/OWASP/cve-lite-cli

run it on your project → it lists the libraries with holes → it gives you the command that fixes each

your code other people's libraries known hole

in plain words Your app is a little of your code and a lot of other people's libraries. A hole in one of them is your hole. The hard part is closing it without breaking the rest.

The itch

“Why don’t Composer users have this?”

`in plain words` Composer is how PHP applications get their libraries. It can tell you which library has a hole. It does not tell you which upgrade closes it without breaking the rest.

`one extra requirement`

Never recommend a command until Composer’s own solver has proved it works.

Composer Remediate

```
/home/hexblot/src/existing-php-project $ composer remediate

CVE-2026-XXXXX  symfony/http-foundation 6.4.21
  Introduced by  root → drupal/core-recommended 11.4.2 → symfony/http-foundation
  Recommended remediation
    drupal/core-recommended 11.4.2 -> 11.4.3   (3 packages updated, 0 added, 0 removed)
  Composer validation
    PASS
  Recommended command
    composer update drupal/core-recommended -W -m
```

in plain words — You tell it nothing. It finds the hole, works out the smallest upgrade that closes it, proves the upgrade with Composer itself, and prints the one command to run.

01

Building it

Specification first. Then an AI wrote most of the code.

I did not ask AI to “build me an app”

Slop

“Generate a Composer security tool.”

in plain words

The brief you would never give a builder: “make me a house”. No plan, no materials list, no inspection.

Engineering

- Behaviour: what it does, and what it refuses to do
- Security invariants: what must never happen
- Failure modes: how it fails, and that it fails closed
- Documentation, written before the code
- Real-world fixtures to test against
- Release criteria: what “done” means

The gates the code had to pass

PHPStan level 8

the strictest static analysis the language has

Deptrac

architecture layers as rules; a wrong import fails CI

507 tests

3,937 assertions, run against real project fixtures

94% coverage

measured the same way locally and in CI

Every PR reviewed

by Gitar, an independent reviewer that found real defects

Supply chain

SHA-pinned actions, scoped permissions, Aikido scans

`in plain words` The same inspections a human team would have to pass. The AI did not get to skip any of them, and neither did I.

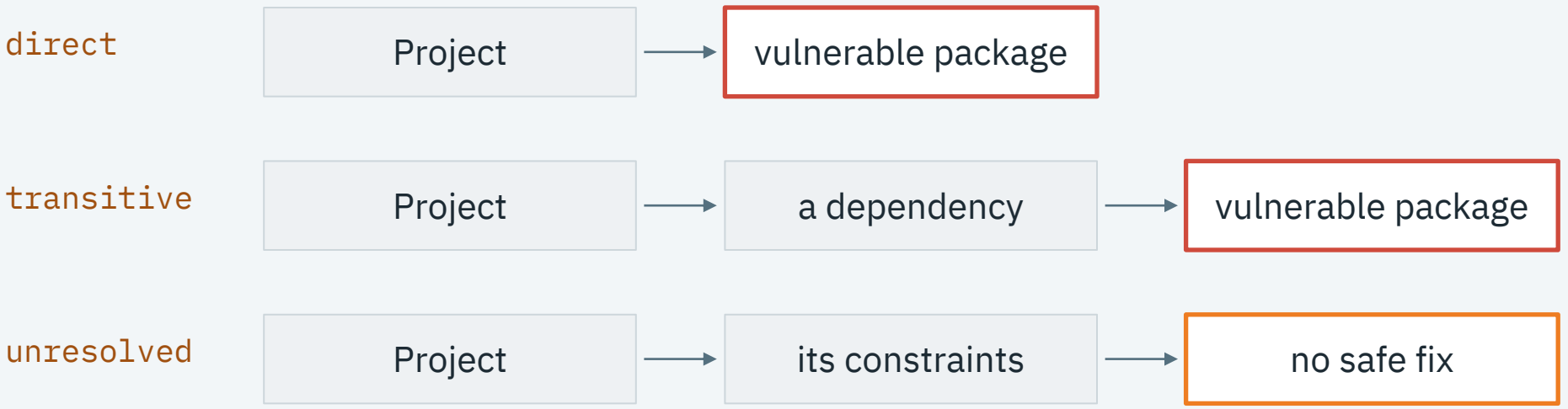
The build loop

Talk it through with ChatGPT › Refine the plan in Claude Code › Validate the plan again › Claude Code builds v1 › Human skim

Different roles. Different context. Deliberate hand-offs.

`in plain words` A design meeting, an architect's drawing, a second opinion on the drawing, a builder, and a walk-through before anyone moves in.

Tests that collide with reality



Seventeen real FOSS projects, at the moments in their history when they were vulnerable.

`in plain words` Tested against history, where the right answer is already known. “No safe fix” has to be a valid answer.

Seventeen moments from thirteen real projects

BookStack
Laravel, four moments

koel
Laravel, 2024

Pixelfed
Laravel 11, 2024

Invoice Ninja
Laravel, 2022

Shopware
Symfony, two moments

Kimai
Symfony 4.4 on PHP 7.3, 2023

wallabag
Symfony 5.4 on PHP 7.4, 2024

Mautic
Symfony 5.4, 2024

Islandora
Drupal, 2024

USAGov
Drupal 10.2, 2024

Mass.gov
Drupal 10.3, 2024

Open Social
Drupal distribution, 2024

Acquia CMS
Drupal 10.3 monorepo, 2024

Each one captured as of its date, so the tool sees what the developer saw, and the right answer is on record.

`in plain words` Public projects, real history, known outcomes.

By the numbers

< 24h

from the idea to the first working release

12

working releases in the first week

4

redesigns: decisions reversed after they were built

AI did not make quality cheap. It made iteration cheap.

02

Breaking it

“Build this” became “break this”. Fresh context, different models.

Nine rounds of attack

01 Could a stranger adopt it? A fresh model, the published repository, three rechecks.

04 Filesystem safety, advisory completeness, CI decisions: ten findings, a regression test each.

05 The GitHub Action, apply mode, the security boundary: seven findings, two more on recheck.

06 The advisory database’s own supply chain. One finding was a false positive; measured, not assumed.

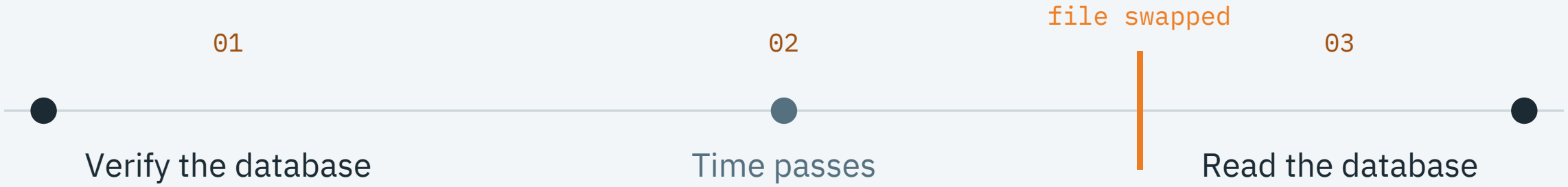
07 The first review to arrive with a runnable reproduction of every finding.

08 Ten findings. Each recheck found the repairs incomplete: four, then three, then one.

09 The audit before the first beta: four defects, all fixed with tests.

in plain words Nine independent inspections, each told to find what the last one missed.

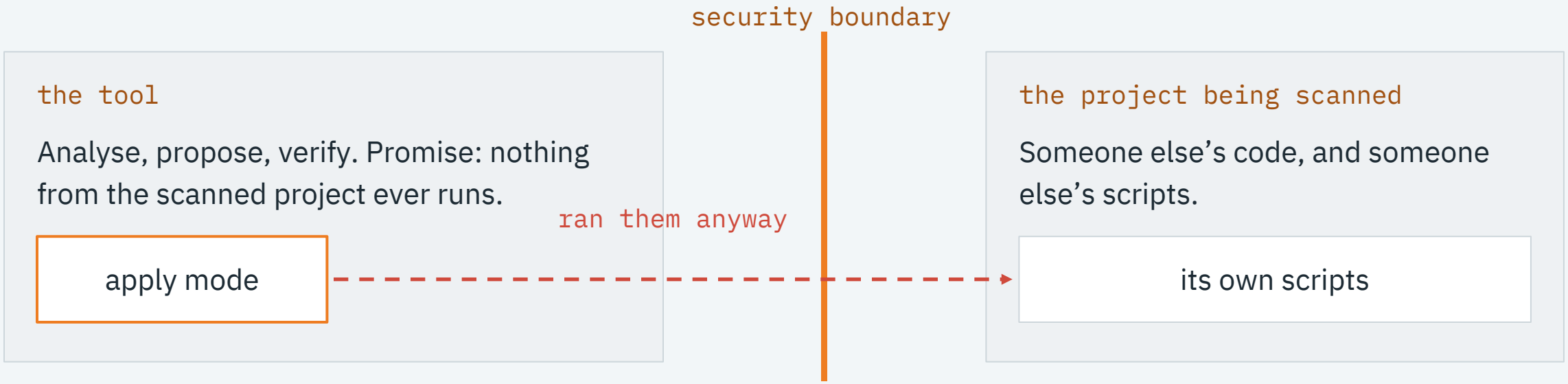
Failure 1: verified, then changed



A check can be correct and still protect the wrong moment.

`in plain words` Checking IDs at the door, then letting people swap places in the corridor.

Failure 2: the boundary looked real



The implementation crossed a boundary it appeared to respect.

in plain words A contractor who promises not to touch the wiring, then hires a subcontractor who does.

Failure 3: everything worked. Except the result.

```
$ run-tests | tee results.log
TEST FAILURE
X security invariant violated
$ echo $?
0
```

The tests screamed. The shell cheerfully reported success.

in plain words The smoke alarm went off. The status board in the lobby said “all clear”.

Failure 4: no attacker required

A feed answers: well-formed, and empty › The database is built without that source › Checksummed, attested, minutes old › A vulnerable lock reports clean

Every signature was valid. Nobody attacked anything.

An empty answer is now a failed build, and a feed that says nothing is a coverage gap, never silence.

`in plain words` Signed, sealed, delivered on time. The envelope was empty.

I fixed one security invariant four times.

check 1: at the right moment	check 2: at the right moment	check 3: at the right moment	a private copy of the verified file
`came back`	`came back`	`came back`	`held`

The fourth fix was not another check.

I changed the architecture so the failure could no longer occur.

Fail closed

an answer		“I could not find out”	
0	No known vulnerabilities.	3	Tool error. A solve left an outcome unknown.
1	Vulnerable, and a verified fix exists: one command.	4	Advisory data unavailable, unverified or too old.
2	Vulnerable, and no verified fix. Said out loud.	5	Package metadata could not be fetched.

A broken tool never reads as “no fix exists”, and never as “all clear”.

in plain words — It is allowed to say “I don’t know”. It is not allowed to say “fine” when it means that.

Then the outside world joined the test suite

`live data`	A misnamed package in GitHub’s advisory database. Fixed upstream.
`missing data`	A whole ecosystem’s advisories were absent from the sources.
`our environment`	A test database leaked through a cache shared between projects.

Every test passed. The defects were outside what the tests thought they were testing.

Reproduce before believing

`the false positive`

A reviewer reported that the database watchdog could never file its first issue.

Measured: the GitHub CLI returns an empty string for an empty list where plain jq prints “null”. The code was right; the reviewer’s mental model was not.

`the one left open`

Path search is bounded in what it returns, not in the work of finding it: six extra packages took 0.007 s to 6 s and 144 MiB.

Two attempts to fix it changed which command the planner recommends on a real fixture. A narrower recommendation is worth more than the saved seconds, so it stays open and documented.

Reproduce every finding before believing it. Then decide; not every finding is a fix.

`in plain words` AI reviewers are confident in both directions. The measurement is the referee.

Different tools were good at different jobs

Claude Code

- Strong, coherent implementation
- Carries a large plan
- Weak at attacking its own assumptions

Codex

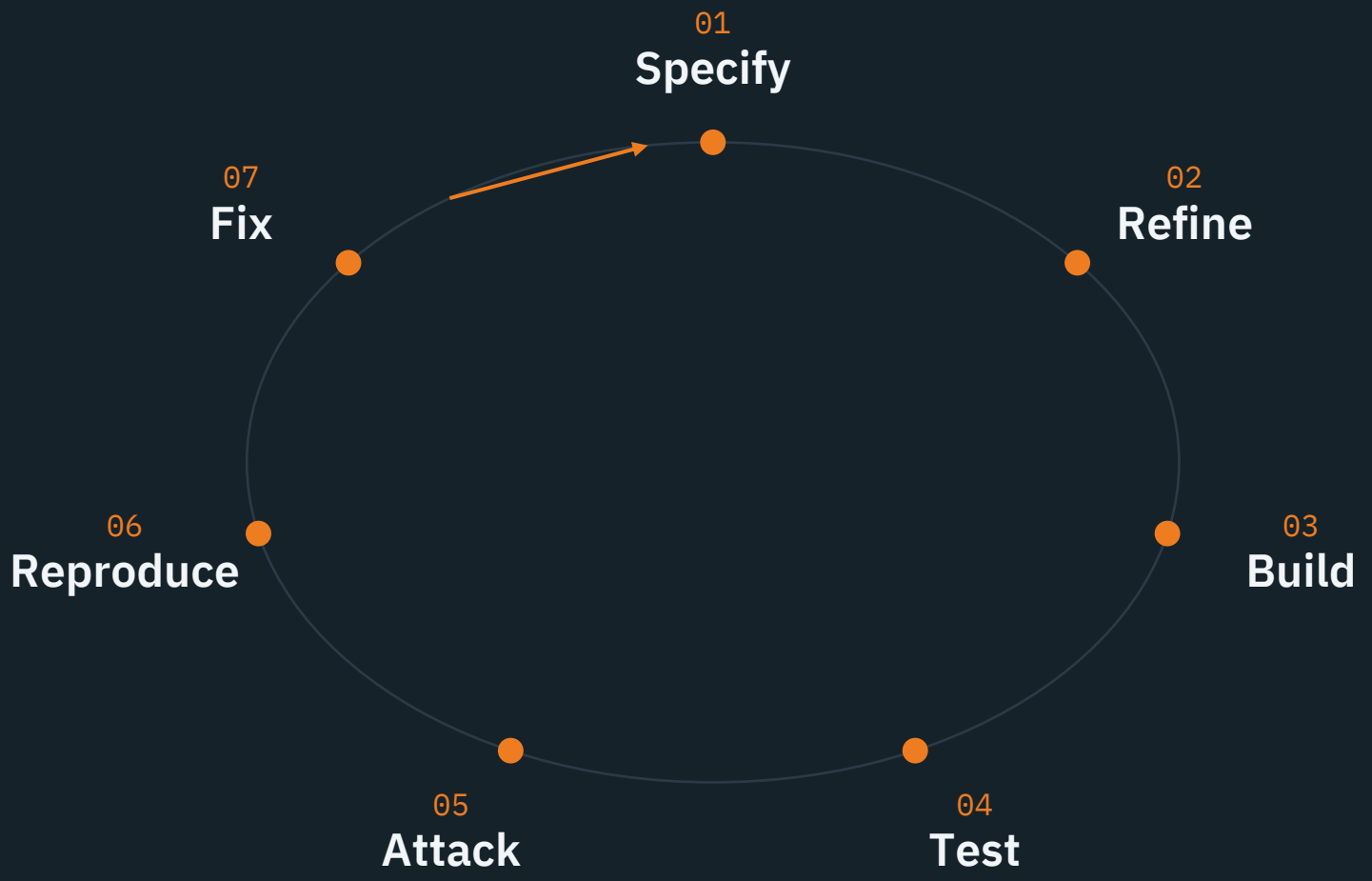
- Strong adversarial review
- Found the boundary and invariant failures
- Different hypotheses

Gemini

- A different review angle
- Useful disagreement
- Found different classes of issue

The winner was not a model. It was role separation and fresh context.

What makes it not slop



Specify what must hold. Build. Test against reality. Attack with fresh eyes. Reproduce every finding before believing it. Fix. Then round again.

And when a rule keeps failing: make it structural.

None of this is new. It is what engineering looked like before generating code got cheap.

The paperwork is part of the product

Use it

- Getting started
- Reading the report
- CLI reference

Understand it

- How it works
- Privacy and network
- Advisory database

Integrate it

- CI integration
- Exit codes to gate on
- Report formats

Evidence

- Case studies
- Comparison with other tools
- Test fixtures

Project

- Design decisions
- Roadmap and delivered
- Changelog with Upgrading notes

It all lives on a public documentation site, written before the code and kept with it: a stranger can see what was promised and check it.

Release policy in writing: patches only fix, minors may change behaviour and say so, CI pins exact versions.

`in plain words` The manual is a promise. The changelog is the record of keeping it.

Slop is not “code written by AI.”

Slop is abandoning engineering because generation became easy.

Better tools amplify the process you give them.

AI is a tool, like your IDE.

AI can write it.

AI can test it.

AI can try to break it.

You still own it. Your name goes on it.

Composer Remediate

From vulnerable dependency to verified Composer fix.

MIT licence. PHP 8.1 and later. Development sponsored by Lambda Twelve.

`code`	github.com/hexblot/composer-remediate
`docs`	hexblot.github.io/composer-remediate
`install`	composer require --dev hexblot/composer-remediate



github.com/hexblot/composer-remediate

`in plain words` Free, open source, and it would love additional pairs of eyes. Issues and pull requests welcome.

Lambda Twelve

Thank you. Questions?

Nick Andriopoulos

nick.andriopoulos@lambda-twelve.com

lambda-twelve.com

download these slides
l12.gr/fc26-pptx