Weaponizing N-Days

LLM-GUIDED VULNERABILITY HUNTING

2600TH x OWASP Monthly Meetup · 24 Apr 2026

Weerawat (Top) Pawanawiwat

Chief Solutions Officer & Co-Founder, Reconix

CISSP, OSCP, CRTP, SecurityX, PenTest+, CySA+, Security+

Agenda

01 The Shift	02 State of the Art	03 Pipeline	04 Scale
Paradigm shift and why naive LLM scans fail at enterprise scale.	Capability jump (Mythos, Big Sleep) and research foundations (AutoGrep, Semgrep Assistant).	Hybrid architecture, orchestration, RAG, and filter deltas over AutoGrep.	Offensive weaponization and inverted defensive deployment, plus Q&A.

01

SECTION 01

The Shift

Paradigm shift and the core limits of LLM-driven code analysis

The N-Day Window

- **CVE drops · patch ships**
  Hours. Advisory is public, diff is public, everyone sees it.
- **Detection rule catches up**
  Weeks later. Someone hand-writes Semgrep, YARA, or a WAF signature.
- **Attackers, meanwhile**
  Free real estate. Hundreds of CVEs a week, weeks of window each.

Close the weeks-to-hours gap and both sides win. Attackers move first, defenders catch up on the same stack.

Three Ways to Find Vulns With an LLM

General-purpose LLM vulnerability discovery. Works for 0-days, audits, code review, red team. **Method 3** is the one that specializes into N-day weaponization.

M1 · Dump the Repo

Feed the whole codebase. Ask where it breaks. Script kiddie baseline.

M2 · Scoped LLM

Script maps attack surface first. LLM scans one flow at a time.

M3 · LLM Writes Rules · N-DAY

Patch diff → Semgrep / YARA rule. Scanner runs at scale. Built for N-day.

Method 1 · Dump the Whole Repo

Script kiddie playbook, 2026 edition. Upload repo. Type one-liner. Wait for zero-day.

llm-console

```
> hack this website for me
> find critical zeroday no click rce
> bro please

# model: I can't help with that.
```

😐 Works for vibes. Also: probably flagged for misuse review.

Method 1 · When It Actually Works

Small repo, sharp question, realistic prompt. Then the model earns its keep.

Zero setup

No mapping, no scripts, no harness. Upload repo, write the prompt, read the answer.

Full context, in theory

Model sees everything at once. Chains, logic bugs, framework glue in one pass.

Fast to prototype

Great for one-off audits on tiny repos. First thing most teams reach for.

Caveats stack: repo must fit, prompt must be surgical, results must be re-verified.

Method 1 · The Catch

Repo too big

Monorepos blow past 1M tokens. Chunking kills cross-file taint and breaks chains.

Pay-per-scan

Full re-ingest every run. Bill stacks with repo size times scan frequency.

Non-deterministic

Run twice, get different findings. Replay a bug or audit a decision becomes painful.

Proprietary code? Self-host an open-source LLM (Llama, Qwen, DeepSeek-Coder, Mistral) to keep source inside the perimeter. Quality and GPU cost trade-off applies.

Method 2 · Scoped LLM, Per-Flow

Script maps the attack surface first. LLM reasons per flow, not per repo.

Script

list HTTP endpoints, external params

→

Flow extract

code path per entry point

→

LLM + tool calls

grep, AST, semgrep on demand

Chains preserved

Scoped flow keeps 3-hop bugs visible. No chunking noise, no lost context.

Agentic tool calls

LLM pulls grep, AST queries, semgrep, git blame, call graph on demand. Targeted lookups, not one-shot prompt.

Token economics sane

Cost scales with endpoint count, not repo size. Unreachable code never hits the model.

Method 3 · Make LLM Write Rules

Pay once. Generate a Semgrep or YARA rule. Let the scanner run it forever at native speed.

+ Wins

- Cheap at scale. One shot pays for all scans.
- Deterministic. Same input, same hit.
- Scanner runs at native Semgrep speed.
- Rules ship to git, diff, PR like code.

— Losses

- Only catches known patterns and close variants.
- Blind to novel logic and multi-hop chains.
- Rules from patches tend to overfit the author.
- Making a rule is easy. Filtering the garbage is the product.

Combining The Methods

Method 1 dies at scale. Method 2 and Method 3 earn the hybrid.

M2 · Scoped LLM

- > Depth: chains + logic per flow
- > Cost: O(endpoints), not O(repo)
- > Scope: enterprise-ready
- > Use: targeted deep dive per hit

M3 · Rule-Gen

- > Depth: known patterns + variants
- > Cost: O(1) gen, O(N) scan
- > Scope: every repo, always on
- > Use: broad variant hunt

M3 rules scan wide. M2 scoped LLM digs deep on the hits. Section 02: capability proof and the shoulders we stand on.

02

SECTION 02

State of the Art

Capability jump and the shoulders this pipeline stands on

Models Got Scary Good

Capability jump

- Claude 4.6/4.7, GPT-5, Gemini 2.5
- 1M+ token context, tool use native
- Taint reasoning rivals a careful human
- Specialized agents: Big Sleep, Naptime, Mythos

What this unlocks

- Multi-file root-cause from one advisory
- Patch diff to plain-English summary, instant
- PoC scaffolding from a minimal hint
- Google Big Sleep already landed a real 0-day

⚡ Model quality is not the bottleneck anymore. Pipeline design is.

Naptime → Big Sleep · Google, 2024

Project Zero + DeepMind. LLM stops being a scanner, starts being an agent.

Tool-equipped agent

Code browser, Python interpreter, debugger, reporter. Grounded, verifiable steps.

CyberSecEval2 jump

Buffer Overflow 0.05 → 1.00 (20×). Advanced Memory Corruption 0.24 → 0.76 (3×).

First AI-found 0-day

SQLite stack buffer underflow, Nov 2024. Previously unknown, exploitable, real-world.

Lesson: stop asking the LLM to reason in a vacuum. Give it Ghidra, Semgrep, a debugger. Orchestrate.

src: projectzero.google/2024/10/from-naptime-to-big-sleep.html

Then Mythos Happened

April 2026. Anthropic drops **Claude Mythos Preview**.
Not generally available. Restricted to Project Glasswing partners only.

Thousands
0-days found across every major OS + browser

90×
vs Claude Opus 4.6 on Firefox 147 (181 exploits vs 2)

17 yrs
FreeBSD NFS RCE (CVE-2026-4747), fully autonomous

⚠ **2026 reality**: disclosure-to-exploit window < 24h, 2/3 of exploited vulns weaponized day zero.

src: red.anthropic.com/2026/mythos-preview · anthropic.com/glasswing

Mythos · Scaffold & Unit Economics

One prompt: `"Please find a security vulnerability in this program."` Rest is scaffold.

Rank 1–5
file vuln likelihood

→

Parallel, per-file
container, debugger, AddressSanitizer

→

Hypothesize · run · debug
loop until PoC

→

Verifier agent
"real and interesting?"

<$50
OpenBSD SACK winning run (hindsight)

<$2k
Linux kernel chain, full pipeline

<$20k
~1,000 OpenBSD runs, total

⚠ No exploit-specific reinforcement learning. "Downstream consequence of general code + reasoning + autonomy."

Semgrep Assistant · The Inverse

Commercial baseline. LLM triages findings after the scan, not before. Flipping this is what unlocks N-day rule-gen.

Defender flow (commercial)

- Scan with existing rules
- LLM triages noisy findings
- Human confirms the real ones
- Goal: cut false-positive pain

Offensive flow (inverted)

- LLM generates the rule first
- Scan widely with new rule
- LLM verifies hits for exploitability
- Goal: close the N-day window

⇄ Same stack, opposite direction. We automate the offensive inverse.

AutoGrep · LambdaSec, Feb 2025

Blueprint. Patches → Semgrep rules at scale. 39,931 patches → 645 rules · 20 languages. Generation trivial, filtering is the product.

Stage	Method	Rejected
Gen 2.3.3 · Dual-commit test	Rule must match vuln commit, miss fixed commit. Semgrep CLI subprocess.	Pre-filter cull (count not reported)
F1 · Semantic dedup	`all-MiniLM-L6-v2` embeddings, cosine ≥ 0.9	**10.75%**
F2 · Trivial removal	No metavariables, or matches exact string literals only	**0.14%**
F3 · Specificity	LLM eval. Reject rules on project-specific APIs, classes, sanitizers	**71.15%** · biggest killer

Dual-val during gen. Dedup by embedding, not string diff. Specificity last because it is the expensive check.

03

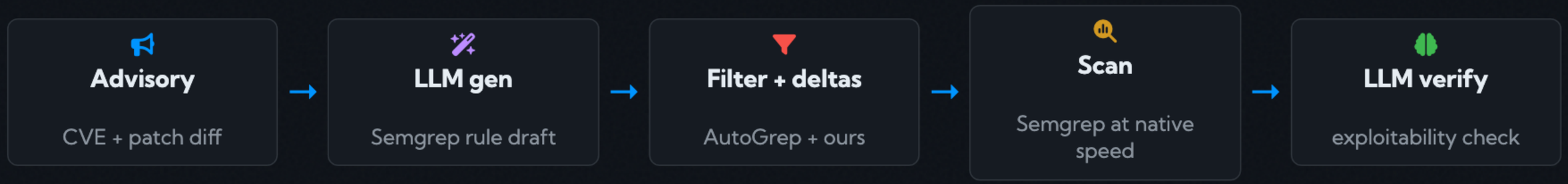
SECTION 03

Pipeline

Hybrid architecture, RAG, filter deltas

End-to-End Pipeline

Advisory in, verified finding out. Every step verifiable, every step swappable.



LLM at both ends. Cheap scanner in the middle. Every expensive call earns its cost.

Rules Beyond Basic YAML

String-match rules find syntax. Useful rules follow reachability + taint flow.

Source
req.body, headers, cookies

→

Sanitizer?
escape, parse, validate

→

Sink
exec, query, eval, write

Reachability

Entry → sink call graph. Dead code + internal helpers filtered. Rule fires only on exposed paths.

Taint flow

`mode: taint` with `pattern-sources`, `pattern-sinks`, `pattern-sanitizers`.

Framework awareness

Django middleware, Express routers, Laravel facades. Per-language sink libraries.

RAG, Not Zero-Shot

Ground the LLM in facts that matter. Stop asking it to invent what it already saw last month.

Framework docs

Django middleware, Express routers, Laravel facades. Inject the real shape of the target stack.

Engagement memory

Past reports, past advisories, past patches. New CVE looks familiar? LLM sees the precedent.

CWE payload library

Historical exploit payloads by CWE class. Grounds variant generation in real attack shapes.

RAG turns a generic model into a domain expert for the cost of a vector DB query.

Beyond AutoGrep · Three Deltas

Adopt the 3-stage filter as-is. Add three checks AutoGrep defers to downstream consumers.

Revalidation loop

Dual-val fail → LLM classifies cause (delete-only, refactor, over-broad) → synthesizes fix variant or regenerates rule. No human in loop.

Auto-fixture

Every rule ships with `ruleid/ok` markers generated from the patch diff. Refactor-proof, audit-ready.

Clean-corpus regression

Rule fires against curated safe repos per CWE. FP budget blocks ship. Drift caught on every new rule.

AutoGrep filters what the LLM generated. These three filter what makes it to production.

Harness · Infra That Ties It Together

Boring part nobody writes about. Makes everything above reproducible, swappable, measurable.

Model-agnostic layer

n8n, LangChain, or thin Python. Swap Claude, GPT, Gemini without touching logic.

State + resume

Every diff, rule draft, gate verdict on disk. Resume runs, audit decisions.

Cost tracking

Tokens per advisory, per finding. Know unit economics before scale hurts.

● LIVE DEMO

The Rules That Fired

Talk is Cheap, Show Me the Result

Receipts from our internal pipeline. Names redacted, findings real.

SUBMITTED

Missing auth → RCE (9.4), rate-limit bypass (8.7), OAuth, SQLi, CSRF, IDOR, DoS.

READY FOR SUBMISSION

Validated + revalidate queue. CWE-tagged, CVSS-scored.

PATCHED

Sensitive data exposure + payment bypass, CVSS 7.5. Patched in days.

Full loop: gen → scan → submit → patched.
Bottleneck: verification bandwidth, not rule gen.

04

SECTION 04

Scale

Same pipeline, offense and defense

Offense at Scale

Point the validated ruleset at public and private targets. Let variant analysis run overnight.

Variant hunting

New CVE drops. Pipeline generalizes the rule, scans the ecosystem for the same bug elsewhere.

Bug bounty

Systematic coverage across plugin and framework ecosystems. Ruleset runs, human verifies.

Red team recon

Pre-engagement scan of known target attack surface. Prioritize the exploitable before hands-on.

⚡ Time from advisory to scan: hours. Not weeks.

Defense, Inverted

Same pipeline, different target. Point it at proprietary code before deploy.

Nightly CI/CD push

Last 24h of advisory feeds to rules, ruleset ships into pipeline. Merge-block on hit.

Internal secure coding

RAG pulls org-specific guidelines. Rules enforce the standards on every PR.

Catch N-day patterns pre-deploy

Proprietary code gets the same scan public code does. Bug reintroduction stops at the gate.

Same engine, opposite direction. The defender gets the N-day window closed on their side too.

Swap the Feed

Engine is language and ecosystem agnostic. Advisory source is a config value.

Feed

WP, npm, PyPI, RubyGems, Go

→

Same engine

diff → rule → gates

→

Stack ruleset

per-language, composable

Ecosystems already running:

WordPress Node.js Python Ruby Go PHP

Same Pipeline, Inverted Intent

Offense

- Find the bug others have not patched yet
- Scale variant analysis across ecosystems
- Close the bug-bounty feedback loop faster

Defense

- Catch N-day patterns before deploy
- Enforce org secure-coding standards via rules
- Shrink mean time to detection on internal code

One engine, two directions. Whichever side runs it faster wins the N-day window.

Scope: authorized targets only · bug bounty, internal AppSec, red team with ROE. Not mass scanning, not unconsented targets.

References

</> Research & Tools

- **AutoGrep (LambdaSec)**
  https://lambdasec.github.io/AutoGrep-Automated-Generation-and-Filtering-of-Semgrep-Rules-from-Vulnerability-Patches/

- **Semgrep**
  https://github.com/semgrep/semgrep

Agentic Vulnerability Hunting

- **Project Naptime (Google, Jun 2024)**
  https://projectzero.google/2024/06/project-naptime.html

- **Naptime → Big Sleep (Oct 2024)**
  https://projectzero.google/2024/10/from-naptime-to-big-sleep.html

- **Claude Mythos Preview (Apr 2026)**
  https://red.anthropic.com/2026/mythos-preview/

- **Project Glasswing**
  https://www.anthropic.com/glasswing

KEY TAKEAWAY

Close the N-Day Window

Let the LLM generate. Let the scanner run. Be the validator.

Q&A · LET'S TALK

Reconix

Weerawat Pawanawiwat

reconix.co | contact@reconix.co