Pixee

OPERATIONAL PLAYBOOK · MYTHOS-READY SECURITY

VulnOps

The Operational Playbook for Mythos-Ready Security Programs

A Pixee field guide to machine-speed defense in the resolution layer — where triage that drops the noise and remediation that merges are co-equal.

SURAG PATEL · CEO, PIXEE

<24 hr

DISCLOSURE-TO-EXPLOIT WINDOW IN 2026 — FROM 2.3 YEARS IN 2018

181

WORKING EXPLOITS FROM ONE MYTHOS PREVIEW — ITS PRIOR MODEL FOUND 2

252 days

AVG. TIME TO REMEDIATE A CRITICAL FLAW TODAY (VERACODE SoSS 2025)

MAY 2026

PIXEE · AGENTIC SECURITY ENGINEERING

PIXEE.AI/VULNOPS

TL;DR

In early May 2026 the Cloud Security Alliance, SANS Institute, [un]prompted, and the OWASP GenAI Security Project published *The AI Vulnerability Storm: Building a Mythos-Ready Security Program*. The paper is the industry's coordinated response to what effective cybersecurity programs will look like given a permanent structural acceleration in offensive AI capability.

The signatories include Google, Sysdig, Cloudflare, Sophos, Rivian, NFL, Atlassian, GitLab, Salesloft, TransUnion, Justworks, lululemon, FDIC, and dozens more.

> *Sixty CISOs do not co-author papers casually.*

The paper is an industry wake-up call, arguing that defense has to be re-framed for a world of machine-speed exploitation. To some extent that has already been true. The window between vulnerability disclosure and confirmed exploitation has compressed from 2.3 years (in 2018) to under 24 hours (in 2026). But the pressure is only compounding as Anthropic's Mythos and successor frontier models continue to ship over the coming weeks.

To respond to this moment, the paper argues that we need to implement eleven priority actions (PAs):

DETECTION RESOLUTION FOUNDATION

PA 1 Point agents at your code & pipelines	PA 2 Require AI agent adoption
PA 3 Defend your agents	PA 4 Establish innovation governance
PA 5 Prepare for continuous patching	PA 6 Update risk models & reporting
PA 7 Inventory & reduce attack surface	PA 8 Harden your environment
PA 9 Build a deception capability	PA 10 Build an automated response capability

PA 11 Stand up VulnOps ← THIS PLAYBOOK

The paper groups these PAs into governance, risk control, and operational enabler categories. We like to think of them as mapping to detecting vulnerabilities and attack surfaces, fixing vulnerabilities with a compliance and audit trail, and general foundational work across governance, risk control, and general operational enablement.

That looks like this:

Eleven priority actions, three layers — Pixee runs the Resolution layer.

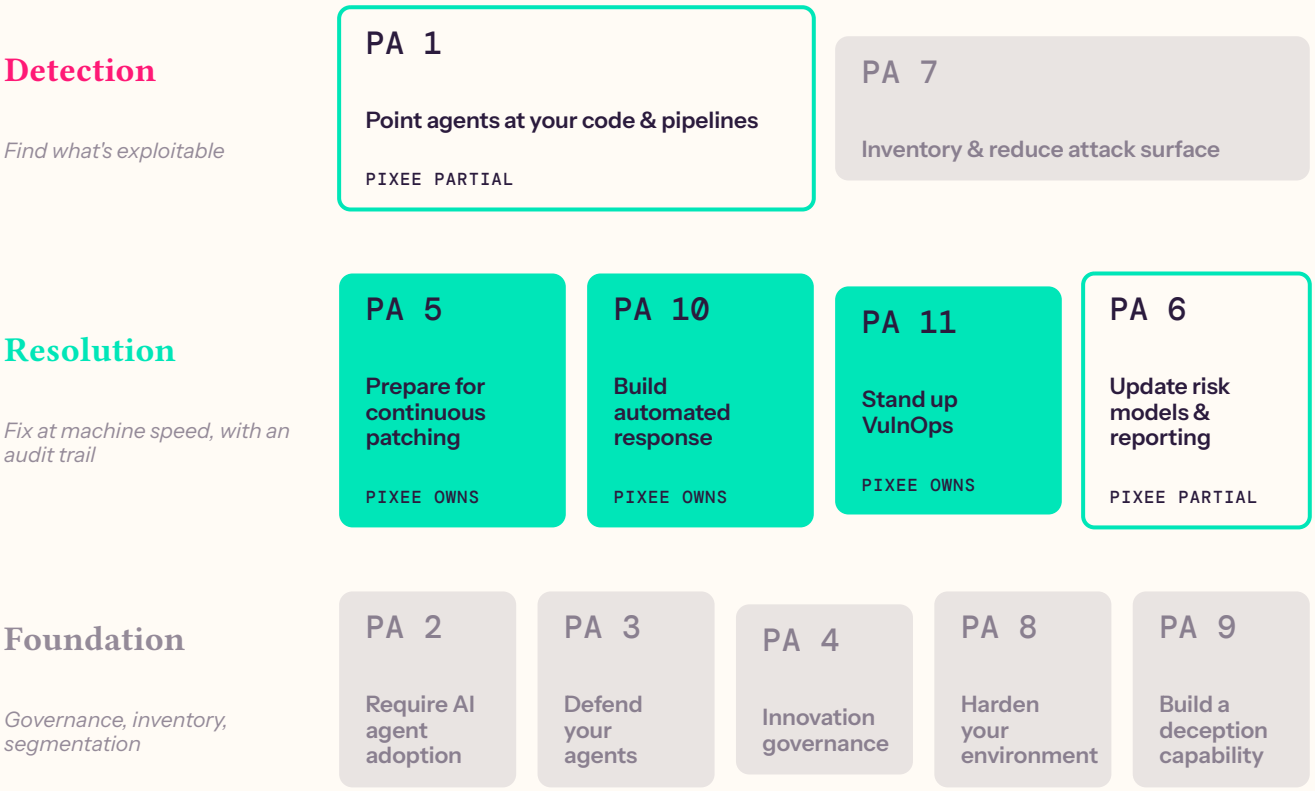
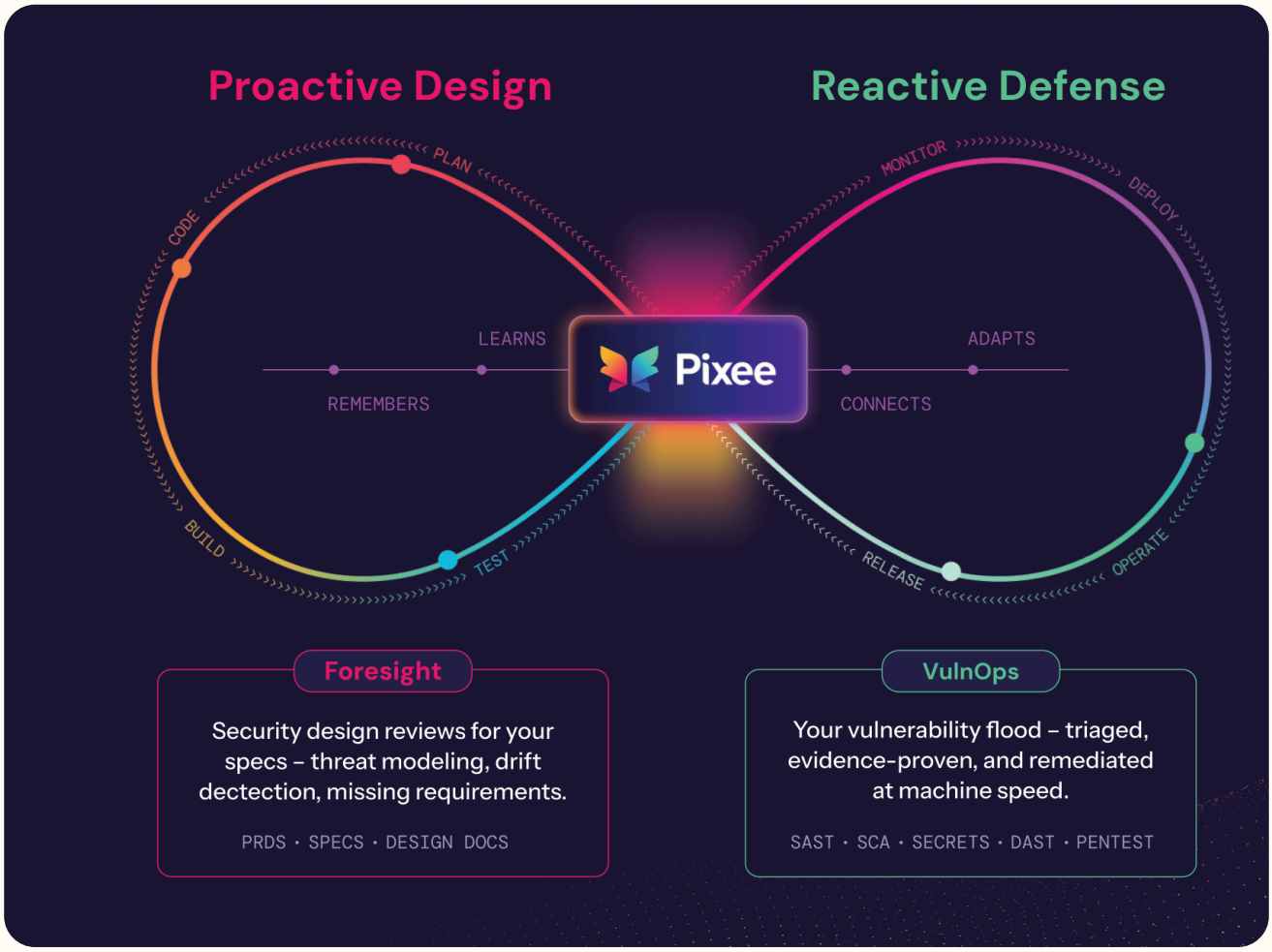


FIG. 2 *The CSA paper's 11 Priority Actions, grouped into the Detection, Resolution, and Foundation layers as Pixee reads them. Pixee directly owns PA 5, 10, and 11; answers PA 6 inside every auditable disposition; partially covers PA 1 via pre-merge LLM review. PA 7 and the Foundation layer are out of scope — upstream and parallel.*

SOURCE: Pixee's layer reading of CSA × SANS × [un]prompted × OWASP GenAI Security Project, The AI Vulnerability Storm v1.0 (2026).

Taken together the recommendations flow into a new category of defense they name **Vulnerability Operations**, or **VulnOps**: the operating layer that ingests findings from every detection source, dispositions them under a single exploitability and severity model, generates validated remediations against the codebase, and writes an auditable record of every machine-judged step. The program then runs as fast as findings arrive, without losing the ability to show its work.

To us, a well-functioning VulnOps practice requires taking both a proactive approach to hardening code before it's written and a reactive approach to patching vulnerabilities as soon as they are created and discovered.

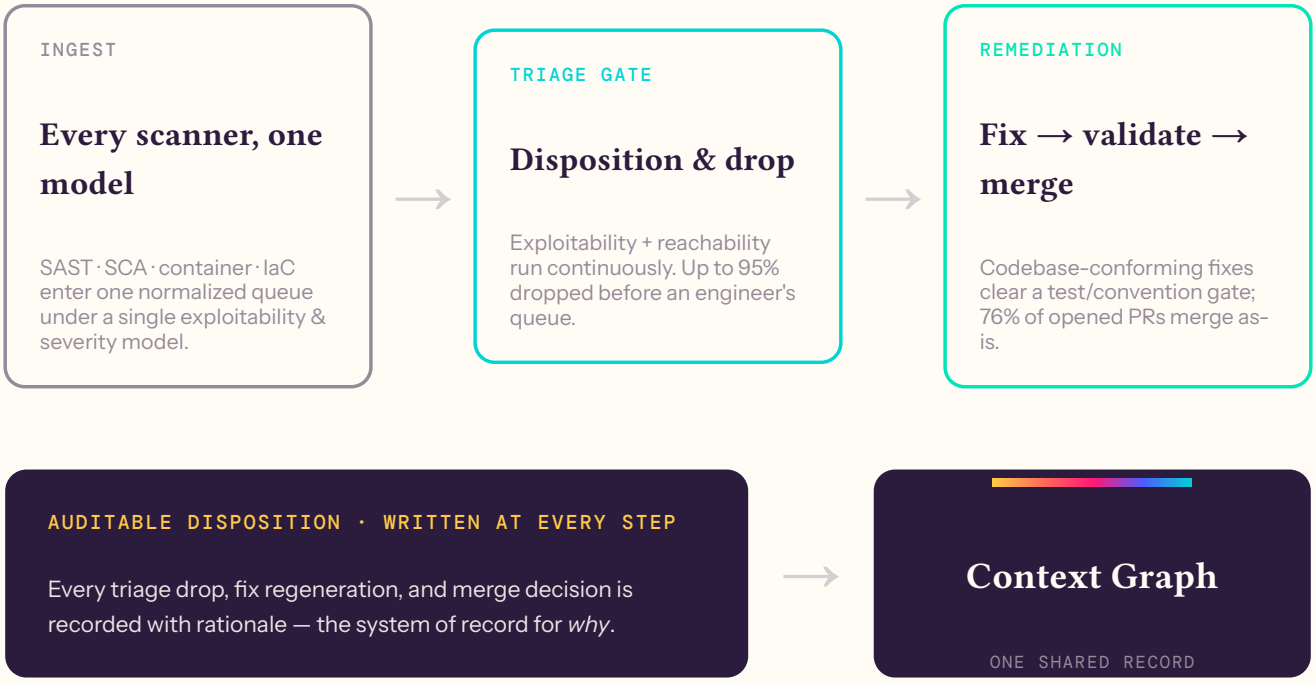


AGENTIC SECURITY ENGINEERING — PROACTIVE (FORESIGHT) ‖ REACTIVE (VULNOPS), ONE CONTEXT GRAPH

On the reactive, fix side of the house — which is where this paper concentrates — it looks like this:

The VulnOps pipeline — one operating layer

Ingest every scanner under one model, drop the noise at the triage gate, merge validated fixes — and write down why at every step.



SOURCE: Pixee VulnOps operating-layer architecture.

In the rest of this paper we focus on what it takes to build machine speed defense in the resolution layer of vulnerability operations. This is the space Pixee has been operating for over two years with automated triage and fix agents spanning SAST and SCA handling everything after vulnerability detection. Increasingly it also relates to our *Foresight product* which focuses on hardening architecture choices at design and PRD stages in order to catch gaps before a single line of code is written. The beauty of this dual approach is that we create a closed loop that enhances the contextual layer that any machine speed defense program must prioritize building (a separate topic we've written about here).

Pixee

SECTION ONE

01

The Mythos Shift

The window between disclosure and weaponization collapsed from 2.3 years to under 24 hours — and the collapse is permanent.

The state of the union: a crisis of find-but-never-fix

Before Mythos, the existing AppSec stack was already failing to keep pace with human-driven offense.

- The Ponemon Institute reports **66%** of organizations carry **100,000+ open vulnerabilities**.[1]
- Veracode's longitudinal *State of Software Security 2025* reports a **252-day average time to remediate** critical flaws.[2]
- SAST industry baseline false-positive rates run **71–88%** per scanner.[3]
- **81% of teams ship vulnerable code** because of capacity, not negligence.

Aside from a weakened security posture, this has a real cost. Aikido reports developers spend 6.1 hours per week on triage at a productivity cost of roughly $20,000 per developer per year.[4]

This is the **find-but-never-fix gap**, and it predates the AI shift entirely.

This is the baseline. Now enter Mythos and AI-enabled attack velocity.

The AppSec stack was already losing to human-speed offense — before Mythos.

66%
of organizations carry 100,000+ open vulnerabilities

252 days
average time to remediate a critical flaw

71–88%
SAST false-positive rate, per scanner

81%
of teams ship vulnerable code for capacity, not negligence

1. 6.1 hrs/week per developer on triage ≈ $20,000 per developer per year

FIG. 1 *The find-but-never-fix gap predates the AI shift entirely: organizations accumulate findings far faster than they remediate them, at a real and measurable productivity cost.*

SOURCE: Ponemon Institute; Veracode State of Software Security 2025; SAST vendor-benchmark baselines; Aikido developer-productivity benchmark.

Mythos takes this to a whole new level

The window between vulnerability disclosure and confirmed exploitation has compressed from 2.3 years (2018) to 56 days (2024) to under 24 hours (2026). Sergej Epp's Zero Day Clock at Sysdig has the data. It's still moving the wrong direction. For example:

- Anthropic Mythos preview (April 2026) generated **181 working Firefox exploits** where Anthropic's prior Opus 4.6 model produced 2 under the same internal lab conditions. The paper reports a **72% exploit success rate** on full chains.[5]
- DARPA AICC finalist systems found **54 vulnerabilities** and produced exploits across **54 million lines of code in 4 hours** of compute.[6]
- Anthropic Claude Opus 4.6 found and reported **500+ high-severity OSS vulnerabilities** in February 2026. The same evaluation surfaced **12 OpenSSL zero-days**, including a **CVSS 9.8** dating from 1998 that survived **28 years** of human code review.

As the CSA authors state, "The window between discovery and weaponization has collapsed into hours. This represents a permanent acceleration, not a temporary spike." (Paper p.8.) The time collapse is significant. The permanence of that collapse is what necessitates shifts in our approach.

Disclosure-to-exploit has collapsed from 2.3 years to under 24 hours — and the collapse is permanent.

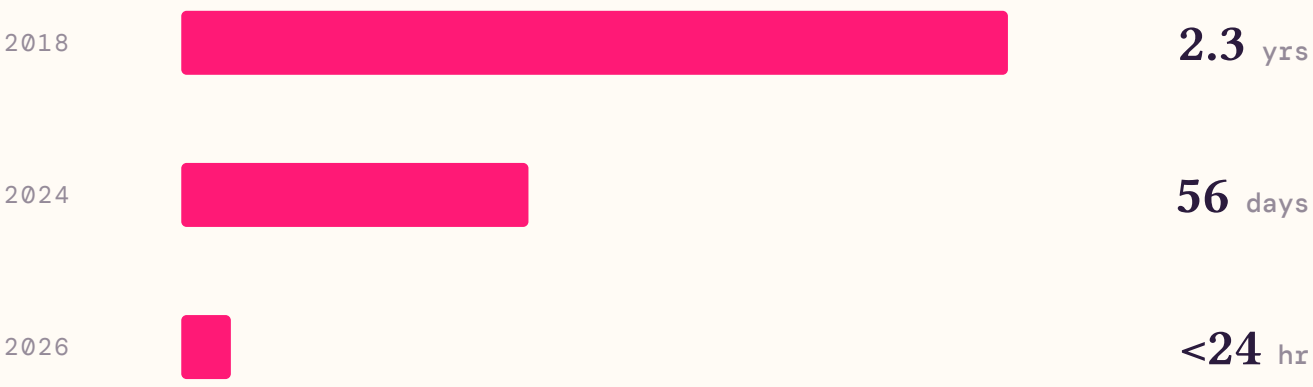


FIG. 3 *The window between a vulnerability becoming public and its first confirmed exploitation has compressed by three orders of magnitude in eight years. Bars are scaled to show the collapse, not to linear time.*

SOURCE: Sysdig Zero Day Clock (S. Epp); compression figures as cited in CSA The AI Vulnerability Storm v1.0.

> *The window between discovery and weaponization has collapsed into hours. This represents a permanent acceleration, not a temporary spike.*
>
> — CSA, AI VULNERABILITY STORM (P. 8)

This is not human researchers working faster. Offensive AI now chains exploit primitives that used to take weeks of human work, against vulnerability classes it's been trained to recognize. There's no version of that capability that gets slower.

This is not human researchers working faster — it is offense that no longer slows down.

181 working exploits

from one Anthropic Mythos preview, where the prior Opus 4.6 model produced 2 under identical lab conditions — a 72% success rate on full chains.

54M lines of code, 4 hours

DARPA AICC finalist systems found 54 vulnerabilities and produced exploits across 54 million lines of code in four hours of compute.

A 28-year-old zero-day

Opus 4.6 reported 500+ high-severity OSS vulnerabilities in one month, surfacing 12 OpenSSL zero-days — including a CVSS 9.8 dating from 1998 that survived 28 years of human review.

FIG. 4 *Three independent results from a single quarter show offensive AI chaining exploit primitives at a scale and speed human research cannot match.*

SOURCE: CSA The AI Vulnerability Storm v1.0 (Anthropic Mythos preview, internal lab eval); DARPA AI Cyber Challenge final results (Aug 2025).

Where do current approaches breakdown?

On the resolution side of vulnerability operations we see three typical bottlenecks across clients we start working with.

On the resolution side, the same three bottlenecks break — in the same order.

BREAKS FIRST

Scanner & tool-stack sprawl

Five or more scanners, each with its own UI, severity model, and false-positive rate. Cross-scanner triage is not a feature of any single-scanner product.

BREAKS SECOND

Human-speed fixing

Hand-coded fixes scaled when input volume was humane. Against machine-speed offense, the backlog gap only widens — you cannot outwork it.

BREAKS THIRD

Poor fix quality from general LLMs

Without context engineering, generic AI patches miss conventions, import unfamiliar libraries, or fail tests — and get rejected at high rates.

FIG. 5 *Across clients Pixee starts working with, the resolution layer fails in a predictable sequence: scanner sprawl first, manual remediation second, and generic-LLM fix quality last.*

SOURCE: Pixee field observation across resolution-layer engagements.

Scanner and Tool Stack Sprawl

Scanner sprawl breaks first. Most enterprise security teams run five or more scanners across SAST, SCA, container, and IaC. Each ships its own UI, severity model, finding metadata, and false-positive rate. At human-speed offense, the noise was painful but somewhat palatable. At machine-speed offense, every minute your team spends triaging a false finding from one scanner is a minute the exploitable finding from another stays open. Cross-scanner triage is not a feature of any single-scanner product. By definition, it cannot be.

Human-speed Fixing

Manual remediation breaks second. Human triage and hand-coded fixes scaled because the input volume was somewhat humane. Throw in more AI-driven exploits and higher exploitable AI generated code in general[7] and the backlog gap continues to widen. As the CSA authors bluntly put it: “we cannot outwork machine-speed threats.”

> *We cannot outwork machine-speed threats.*
>
> — CSA, AI VULNERABILITY STORM

Poor Fix Quality with General LLMs

The last breaking point is more nuanced. It is the fact that generic AI-generated fixes get rejected by reviewers at high rates. We all know frontier models will gladly propose patches. But without elaborate context engineering and an understanding of your organization's specific security architecture these fixes usually do not match codebase conventions, often import unfamiliar libraries, or fail existing tests.

As tempting as it is to throw tokens at the problem, that doesn't scale, in terms of either quality or cost (we break down the buy vs. build question at length here).

What "Mythos-Ready" means, defensively

If we can't outwork machine-speed offense and machine-speed code generation the only option is to build a machine-speed defense program.

"Mythos-Ready" is the operational state of running a security program that produces validated, merge-ready remediation artifacts at the same rate findings arrive. In our mind, this means discovery, triage, fix generation, validation, and PR creation operating as a continuous pipeline rather than a queue.

Concretely, a Mythos-Ready program has four defensive characteristics:

- **Cross-scanner unification at ingestion, not at the dashboard.** Findings from every scanner enter one normalized queue with one exploitability model, one severity convention, and one disposition tracker. Manual scanner-by-scanner reconciliation does not survive machine-speed input volume.
- **Triage discipline as a gate, not as the bottleneck.** Exploitability and reachability run automatically and continuously, not as a periodic human review pass. Findings that fail the triage gate never enter the remediation pipeline. Findings that pass are eligible for fix generation.
- **Fix generation that produces merges, not patches.** Codebase-conforming fixes, validated against existing test suites and conventions, opened as PRs against the right branch. The metric is merge rate, not PR rate.
- **Engineering merge-decision authority preserved.** VulnOps generates merge-ready candidates and tracks dispositions; engineering reviews and merges. The per-PR triage

burden disappears; the merge-decision authority does not. Machine speed does not mean no human-in-the-loop anywhere, it means carefully considered checkpoints.

A Mythos-Ready program runs triage as a gate and fix-generation as merges — co-equal disciplines.

TRIAGE

Cross-scanner unification at ingestion

Every scanner's findings enter one normalized queue — one exploitability model, one severity convention, one disposition tracker — not reconciled at the dashboard.

TRIAGE

Triage discipline as a gate

Exploitability and reachability run automatically and continuously. Findings that fail the gate never enter the remediation pipeline.

REMEDIATION

Fix generation that produces merges

Codebase-conforming fixes, validated against existing tests and conventions, opened as PRs against the right branch. The metric is merge rate, not PR rate.

REMEDIATION

Engineering merge-decision authority preserved

VulnOps generates merge-ready candidates and tracks dispositions; engineering reviews and merges. The per-PR triage burden disappears; the decision authority does not.

FIG. 6 *Four defensive characteristics define the operational state. Two govern triage (the gate that drops the noise); two govern remediation (the merges that close findings). Neither half is optional.*

SOURCE: Pixee operational definition of a Mythos-Ready program.

SCOPE NOTE

This playbook is not arguing that detection and discovery are obsolete. The paper's PA 1 — pointing agents at your code and pipelines — names specific commercial discovery vendors

(Anthropic Claude Code Security, OpenAI Codex Security, Knostic, raptor, Trail of Bits agentic skills) that form a new AI detection layer on top of existing scanners. Those tools are upstream of the operating layer this playbook describes; they feed it findings. Our argument, and what we've built at Pixee, is that the primary bottleneck in AppSec is not finding issues, it's understanding which issues matter and fixing them at speed with quality.

How to know if you have a problem?

Before any vendor evaluation makes sense, three diagnostics tell you whether your current stack is at the breaking point or still ahead of it. None of these require a PO or a procurement cycle.

- **Count your scanner severity models.** List every scanner producing findings against your stack today — SAST, SCA, container, IaC. For each, write down how that scanner ranks severity and how its top-severity tier compares to the others. If you can't reconcile three or more severity scales into one prioritization ranking on a whiteboard, the manual triage layer is already operating on guesswork. That's the cross-scanner unification gap we're talking about
- **Measure your 100-finding triage throughput.** Pick the next 100 findings any scanner produces into your queue. Time how long it takes a triage analyst to disposition all 100 (reachable, exposed, compensated, false positive) to a level you'd defend in an audit. Divide 100 by hours-of-analyst-time. That number is your current triage rate. Compare it to the rate findings arrive: if findings arrive faster than your analysts can disposition them, you're going backwards.
- **Audit your vulnerability fixes.** Walk your last 90 days of security-driven pull requests by bug class. Which classes merged at >80% without engineering edits — dependency bumps, hardcoded-secret removals, library-version pins? Those are the candidates a written auto-merge policy could move out of the per-PR review queue tomorrow, before any new tooling is in scope. The point is to confirm engineering review queue throughput is the binding constraint, not lack of clean candidates upstream.

These three diagnostics are the prerequisites to evaluating any VulnOps function, whether vendor-built, in-house, or hybrid. If the diagnostics return "we're fine," the rest of this playbook is reading material, not action. If they return what we typically see when we walk a customer through them — irreconcilable severity scales, inverting triage rate, idle auto-merge candidates — the rest of this playbook is the operational map.

Pixee

SECTION ONE · AND · A · HALF

1.5

Can We Prevent Vulnerabilities in the First Place?

Shifting left into the PRD: catching security gaps before a single line of code is written.

Everything above this section is the reactive side: findings arrive, the pipeline triages them, the pipeline fixes the ones that survive, the audit trail records what happened. That is the core work of Pixee's triage and fix agents.

A reactive program ingests vulnerabilities that already exist in code that already ships. This work is foundational, and machine-speed offense makes it matter more, not less. But a program that only operates after detection is permanently downstream of the rate at which new exposures get authored.

As AI-assisted coding increases code velocity more of the security-relevant decisions need to happen earlier, in places the scanner stack will never see. To us that means shifting even further left than ever into the PRD, in the ticket, in the spec before code is ever written. That's the proactive side of machine speed defense.

Building Proactive defense before code is written

AI is or will write the vast majority of code in your organization. Which means we can actively improve the security posture of that code by engineering the context and security awareness in the design phase. But no security organization has the headcount to deep-review every design across their product functions, which is why most designs get a cursory pass and the gaps get caught (or missed) in production when there are already vulnerabilities. The problem extends past PRD review. Even a good PRD review needs to translate its promises to action, e.g. to tickets, and then to trace the implementation of those promises in the actual code.

Our newest product, Foresight — now live in production — is designed to do just that. It starts with a product requirement document or PRD. The system reads it the way a senior security reviewer would. It seeks to identify the components involved, surface the gaps where the PRD assumes tribal knowledge, and research those gaps across the customer's actual systems (GitHub, Notion, ticketing, internal docs, repo structure). From there it creates a structured set of **security promises** the PRD makes. Some are explicit ("all exported PII must be encrypted with SHA-256"). Some are implicit ("a new internal service is being created, which under this organization's existing pattern must register with AppSec"). It then ensures those promises make its way into tickets, flagging gaps and proposing additions proactively. In other words, security promises in the design stage serve as an ongoing unit of accountability in the entire SDLC that hardens your code before it's ever written.

Why the proactive side cannot stand alone, and why the reactive side cannot either

A program that runs only Foresight would be incomplete in the same direction a reactive VulnOps-only program is incomplete. Foresight catches what gets designed in; it does not retroactively fix the backlog of code that already shipped. VulnOps catches what scanners surface from code that already shipped; it does not prevent the next batch from being authored. Both modes are necessary because they operate on different inputs at different points in the SDLC, and the same organization has both kinds of risk every day.

The integrating concept is that both feed one Context Graph. The auditable dispositions VulnOps produces and the security promises Foresight produces are different shapes of the same record — structured evidence of a security-relevant decision, with rationale, traced to the artifact (PR, ticket, PRD, finding) that triggered it. When the Graph is shared, the proactive side learns from what slipped through to the reactive side, and the reactive side gets earlier signal on the classes of work the proactive side is catching. The closed loop is an enduring structural advantage for your organization that scales regardless of any other build vs. buy or model considerations you pursue.

What this looks like in practice

Two modes, one shared record: prevention and resolution feed the same Context Graph.

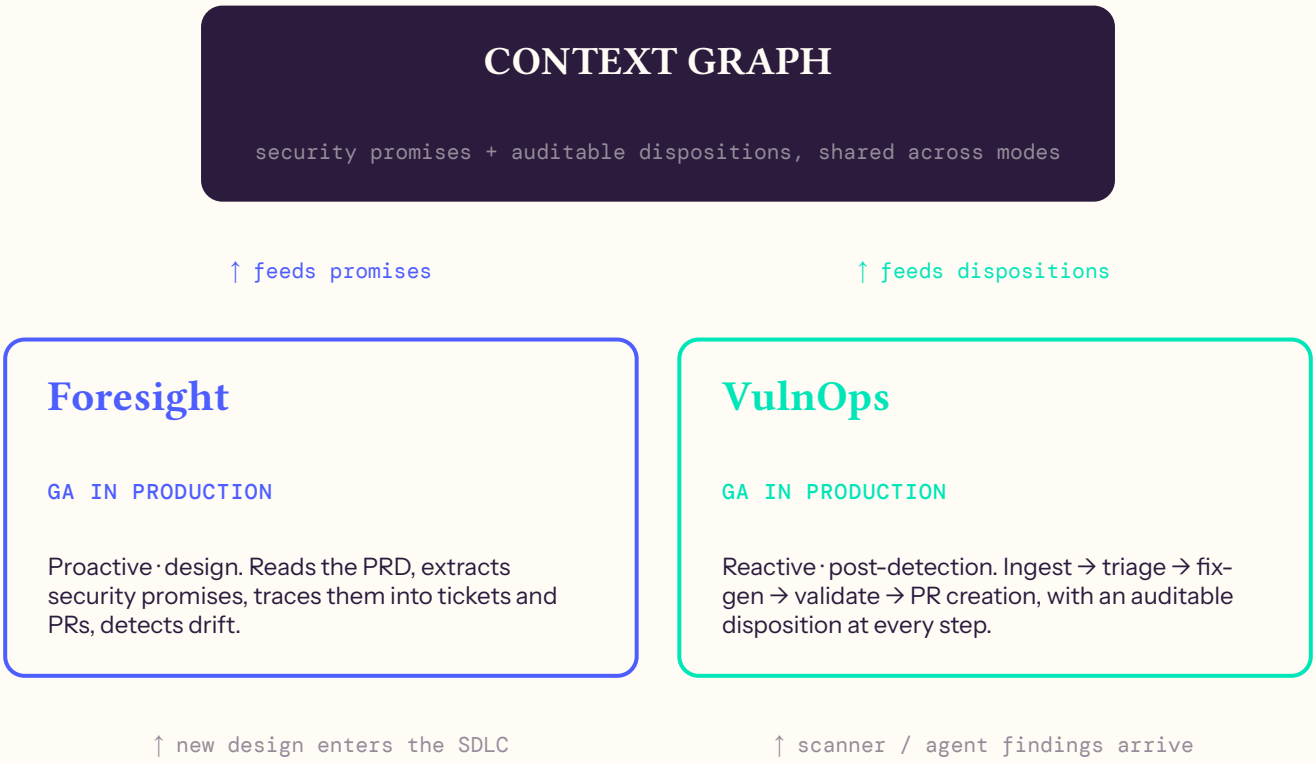


FIG. 8 *Foresight produces security promises at design time; VulnOps produces auditable dispositions after detection. Both are the same-shaped record — evidence of a security-relevant decision, traced to its artifact — and both flow into one shared Context Graph, so each mode learns from the other.*

SOURCE: Pixee platform architecture. Foresight (proactive) and VulnOps (triage + fix) are both GA in production.

The shared record has a name on each side. On the reactive side it is the **auditable disposition** — the structured record Pixee writes for every triage drop, fix regeneration, fix exit, and PR merge, with rationale. That disposition stream is the substrate underneath the merge-rate methodology, the substrate underneath the false-positive-reduction claim, and the evidence a customer's risk model points to when it needs to show its work (PA 6). Every machine-judged step writes one; nothing exits the pipeline silently. On the proactive side the same-shaped record is a **security promise**, extracted from a PRD rather than emitted from a triage decision. Both flow into the one Context Graph above.

Core Architecture Principles for Agentic Security Design

Regardless of whether you consider Pixee's product or are pursuing building your own solutions our 3+ years of building agentic security pipelines involve a lot of hard-won lessons that should be valuable. Here are four of the most important.

Clearly define gates: Clear gates that fail closed. Every stage has a deterministic gate. A finding that fails any gate stops there. The pipeline does not produce PRs that introduce new bugs or violate branch policy, full stop.

Create Accountability via per-finding disposition tracking. Every finding has a logged disposition (merged, closed, deferred, escalated) flowing back to the risk model and the methodology disclosure.

Create constraints to your LLMs. Agentic guardrails on every machine-judged step. Where the pipeline reasons, it reasons under named constraints: triage bounded to reachability, exposure, and compensating controls; fix generation bounded to candidates that pass test-suite, convention, and regression-risk checks before any PR opens.

Prioritize context engineering at every layer. The pipeline never generates a triage decision or a fix from finding metadata alone. Codebase conventions, framework patterns, deployment context, and dependency-chain context are vital inputs at every stage.

Build with cost optimization in mind too. The CSA Mythos paper highlights one additional complexity in the machine-speed era: a cost asymmetry between offense and defense that the per-finding numbers obscure unless you read them carefully.

Offensive AI produces a complete exploit chain for roughly **$2,000**.[8] On the defensive side, a single finding-touch — a triage pass, a fix attempt, a regression check — runs roughly **$790**,[9] and **findings take many touches** before they exit the pipeline. Full remediation of a single finding sits in the **$4,000–7,000** band.[10] Multiply by the volume of findings a modern enterprise stack produces and the **cumulative defensive cost dwarfs** the offensive chain.

One exploit chain costs the attacker ~$2,000. Defending one finding costs many times that — and you pay it per finding.

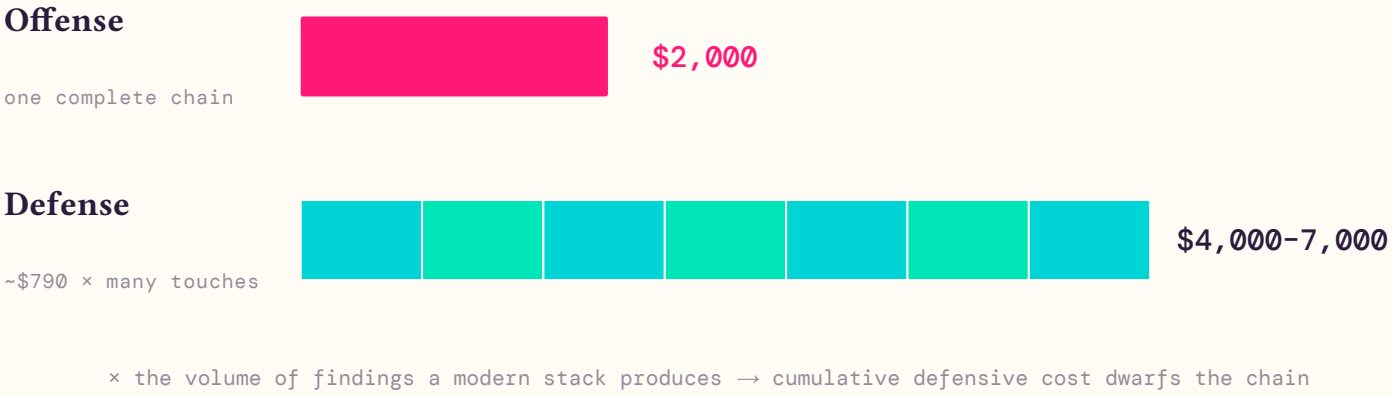


FIG. 7 *The per-finding numbers obscure a structural cost asymmetry. Offense produces a complete exploit chain for roughly $2,000. On defense, a single finding-touch — a triage pass, a fix attempt, a regression check — runs $790, and findings take many touches before they exit the pipeline, putting full remediation in the $4,000–7,000 band. Multiply by enterprise finding volume and cumulative defensive cost dwarfs the offensive chain. The answer is not more spend; it is a smarter context harness that bends the cost curve down.*

SOURCE: Offense $2,000: Anthropic Mythos preview. Defense $790/touch: Phoenix Security. $4,000–7,000 full-remediation band: Pixee internal estimate (weakest of the three; labeled as such).

At scale this is akin to launching million dollar interceptor missiles at $20,000 attack drones. The solution isn't more missiles. It's a smarter harness that lets you deploy context strategically to ensure efficient and effective fixes that drop the cost curve dramatically.

These principles are what produces the numbers. Because triage runs as a gate rather than a periodic review pass, up to 95% of findings are dispositioned and dropped before they ever reach an engineer's queue. Because every fix candidate must clear a validation gate (existing tests still pass, codebase conventions are honored, and the change stays scoped to the fix) before a pull request opens, the PRs that do open merge as-is at a 76% rate (per Pixee's published methodology), against an industry baseline of 30–40% for unsupervised AI fixes.[11] Triage that drops the noise and remediation that merges are co-equal outcomes of the same disciplined pipeline, not a fix bolted onto a scanner.

Triage that drops the noise and remediation that merges are co-equal outcomes of one pipeline — not a fix bolted onto a scanner.

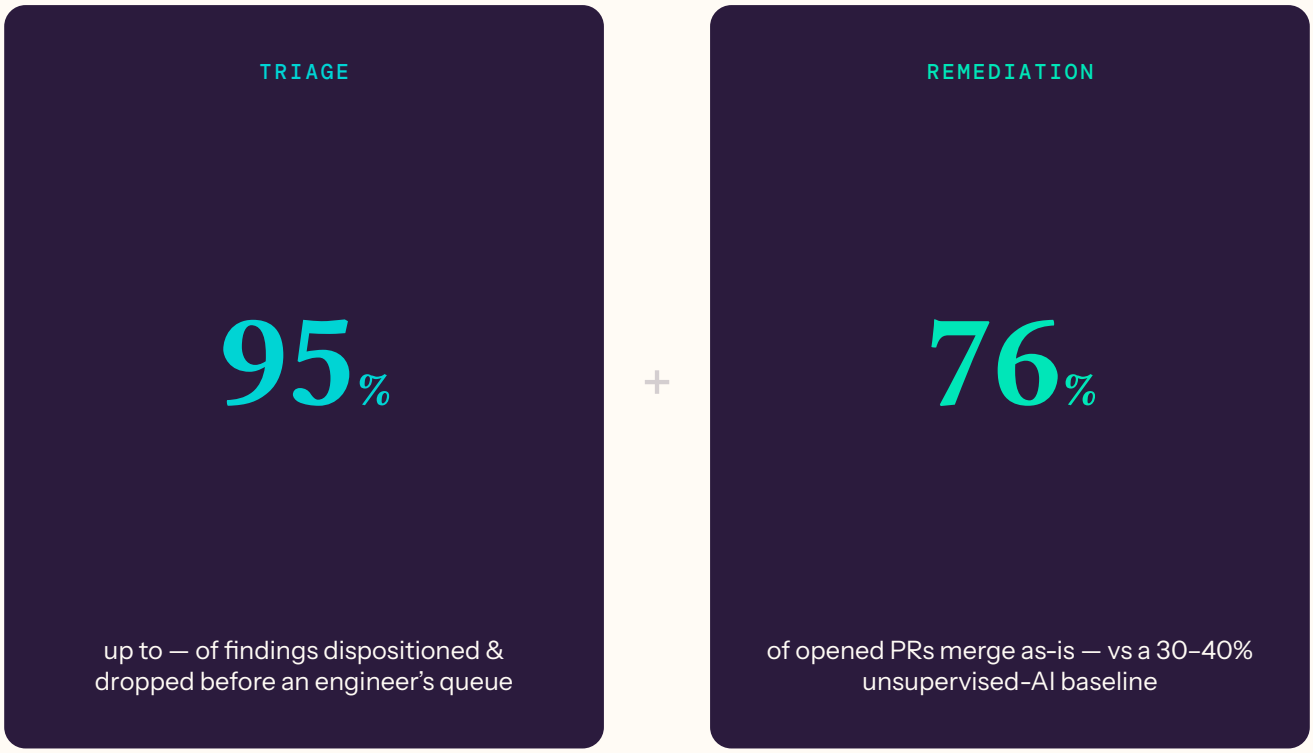


FIG. 9 *Because triage runs as a gate, up to 95% of findings are dispositioned and dropped before an engineer sees them. Because every fix candidate clears a validation gate before a PR opens, the PRs that open merge as-is at 76% — against a 30–40% baseline for unsupervised AI fixes.*

SOURCE: Pixee published merge-rate methodology; 30-40% baseline from Pixee internal benchmarking (directional, not a third-party study).

The last hard-won lesson is operational, not architectural. Don't roll this out everywhere all at once. Seems obvious but the most common failure mode we see in a rollout is the engineering review queue. Cap PRs per repository at the start, prioritize the highest-exploitability findings, and raise the cap only as the team builds trust in what comes through. Again — more merged fixes is the goal, not more PRs. As the disposition stream accumulates merge-decision data per bug class, the classes that clear review with near-zero edits (dependency bumps, hardcoded-secret removals) can (if desired) graduate to pre-authorized auto-merge under written policy for less important repos.

Next step

THE COMING WEEKS

Mythos and successor frontier models ship in the coming weeks. Teams who stand this function up before then will spend that window **ahead of the curve**. Everyone else will spend it **catching up**.

Get in touch → pixee.ai/vulnops

Sources & references

THE ANCHOR PAPER

The AI Vulnerability Storm: Building a Mythos-Ready Security Program

v1.0 · originally 12 April 2026 · last revised 1 May 2026 · CC BY-NC 4.0

CO-PUBLISHED BY

Cloud Security Alliance · SANS Institute · [un]prompted · OWASP GenAI Security Project

60+

named CISO co-authors & reviewers

SIGNATORIES INCLUDE

Google · Sysdig · Cloudflare · Sophos · Rivian · NFL · Atlassian · GitLab · Salesloft · TransUnion · Justworks · lululemon · FDIC

1. dozens more

→ cloudsecurityalliance.org/artifacts/the-ai-vulnerability-storm

PIXEE OPERATIONAL REFERENCES

VulnOps landing · scanner integration matrix · public PR footprint · context engineering · merge-rate methodology.

CONTACT

Surag Patel, CEO, Pixee — surag@pixee.ai. The 30-minute walk-through is bookable at pixee.ai/vulnops.

NOTES

1. Ponemon Institute — organizations carrying 100,000+ open vulnerabilities.
2. Veracode, *State of Software Security 2025* — longitudinal mean time to remediate critical flaws.

3. SAST industry baseline false-positive range, 71–88% per scanner — cross-referenced in vendor-benchmark literature; see Pixee's false-positive analysis.
4. Aikido developer-productivity benchmark — 6.1 hours/week on triage, $20,000 per developer per year.
5. Anthropic Mythos preview results (181 Firefox exploits vs. Opus 4.6's 2; 72% full-chain success) — Anthropic internal lab evaluation, cited in CSA × SANS × [un]prompted × OWASP GenAI, *The AI Vulnerability Storm*, v1.0 (2026).
6. DARPA AI Cyber Challenge (AICC) finalist results (54 vulnerabilities + exploits across 54M LOC in 4 hours of compute) — DARPA AICC final results, August 2025, cited in CSA *The AI Vulnerability Storm*, v1.0.
7. CodeRabbit's analysis of 470 real pull requests found AI-generated code produced 2.74× more cross-site scripting (XSS) errors than human-written code. Similar increases appear across improper password handling (1.88×), insecure object reference (1.91×), and insecure deserialization (1.82×).
8. $2,000 per offensive exploit chain — Anthropic Mythos preview.
9. $790 per defensive finding-touch — Phoenix Security.
10. $4,000–7,000 full-remediation band — Pixee context-engineering estimate; cited as a Pixee-internal estimate rather than a third-party study (the weakest of the three cost sources).
11. 30–40% industry baseline merge rate for unsupervised AI-generated code fixes — Pixee internal benchmarking against off-the-shelf LLM patch-suggestion output; directional, not a third-party study.

Pixee

ABOUT PIXEE · AGENTIC SECURITY ENGINEERING

Triage that drops the noise. Remediation that merges.

Pixee is the operating layer for vulnerability operations — the resolution layer that runs after detection. **VulnOps** ingests findings from every scanner under one exploitability model, dispositions and drops the noise before it reaches an engineer, then opens codebase-conforming, test-validated fixes that merge as-is. Triage and remediation are co-equal outcomes of one disciplined pipeline — never a fix bolted onto a scanner.

Foresight, our **proactive prong**, hardens the PRD before code is written. Both feed one Context Graph — the system of record for *why* every security-relevant decision was made, with the rationale traced to the artifact that triggered it.

TRIAGE

up to 95%

of findings dropped before

REMEDIATION

76%

of opened PRs merge as-is, vs

AUDIT TRAIL

every

machine-judged step writes an

Book a 30-minute walk-through → pixee.ai/vulnops