AVERLON

AFTER MYTHOS: BUILDING VULNERABILITY RESILIENCE

Why AI-accelerated discovery and exploitation require security programs to move beyond vulnerability management

Sunil Gottumukkala
Founder & CEO, Averlon

May 2026

EXECUTIVE SUMMARY

AI is changing vulnerability management faster than most enterprise operating models can absorb.

The immediate issue is not that one model, one vendor, or one announcement has made every organization indefensible. The more important shift is that advanced AI systems are compressing the time between software creation, vulnerability discovery, exploit validation, and attacker use. Anthropic's Claude Mythos Preview showed that frontier models can perform advanced vulnerability research and exploit development against real software. The UK AI Security Institute's (AISI) evaluation found that Mythos represented a meaningful step up in cyber performance, including success on expert-level capture-the-flag tasks and completion of a multi-step corporate network attack simulation in controlled conditions. AISI's evaluation of OpenAI's GPT-5.5 reached a similar level, showing that these capabilities are not isolated to a single lab or vendor.

For CISOs and security leaders, the implication is direct: vulnerability management programs built around periodic scanning, CVSS-driven ticket queues, and patch SLAs measured in weeks or months will fall behind.

The pressure comes from both sides.

First, AI-assisted development increases the volume and velocity of software entering the enterprise. More code, more dependencies, more APIs, more infrastructure-as-code, more generated scripts, and more rapid change all expand the defect surface that security teams must understand.

Second, AI-assisted vulnerability discovery increases the speed at which weaknesses can be found, validated, and turned into usable attack paths. Defenders can use these same capabilities, but attackers do not need complete enterprise context to cause harm. They only need one exposed path that is reachable, exploitable, and insufficiently contained.

Security leaders need to move beyond patch-first vulnerability management toward vulnerability resilience: the ability to reduce business impact even when vulnerabilities cannot be patched immediately.

A resilient program still patches aggressively. But it also prioritizes by exposure and exploitability, manages inherited supplier and open-source risk, uses AI to accelerate triage and remediation, deploys runtime and compensating controls for the no-patch window, and measures containment and recovery as part of the vulnerability management mission.

The goal is not to eliminate all vulnerabilities. The goal is to reduce the chance that a newly discovered weakness becomes a business-impacting compromise.

1. What Changed: AI Has Compressed the Vulnerability Lifecycle

The security industry has seen automated vulnerability research before. Static analysis, fuzzing, symbolic execution, exploit generation, and autonomous cyber ranges are not new. What is changing is the quality, accessibility, and persistence of AI systems that can increasingly combine these tasks into working pipelines.

The practical change is compression. Advanced models are compressing the time between four things that used to be separated by weeks or months: code creation, vulnerability discovery, exploit validation, and attacker use. Research by Mandiant shows that the time to exploit has collapsed from 63 days in 2018-2019 to **minus** seven days in 2025 - the negative number indicating exploitation before patches are released. That matters because vulnerability management was never only a discovery problem. It is a lifecycle problem, and the lifecycle is getting shorter at every stage.

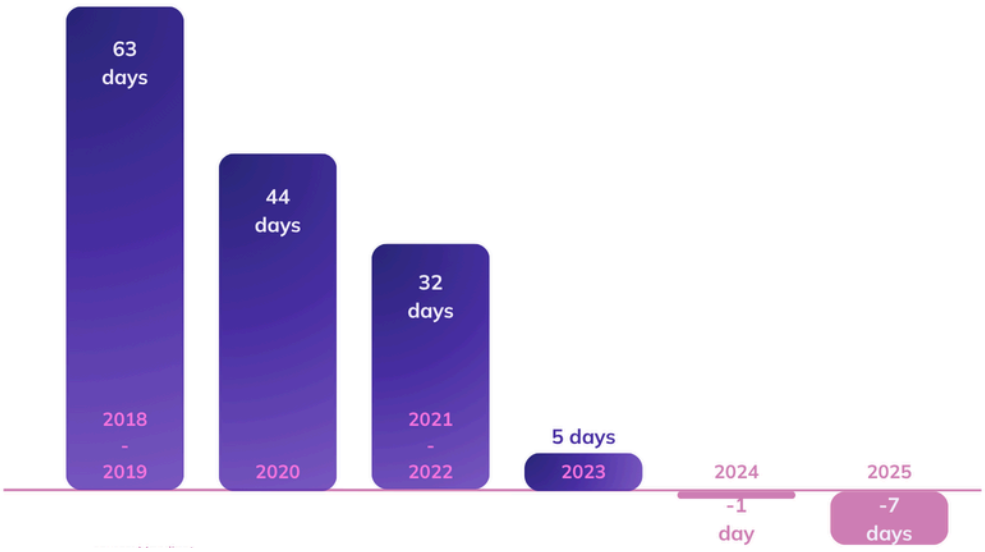

Fig. 1: *The rapidly shrinking, now negative, time to exploit (source: Mandiant).*

Anthropic's April 2026 Mythos Preview announcement put this in the boardroom. Anthropic described Mythos as a highly cyber-capable model and launched Project Glasswing to give selected critical software vendors and maintainers early access for defensive vulnerability discovery and remediation. In its technical writeup, Anthropic said Mythos can find and exploit zero-day vulnerabilities in real open-source codebases and can turn known but unpatched vulnerabilities into working exploits. Most of what it found, Anthropic noted, had not yet been patched, which limited what could be publicly disclosed.

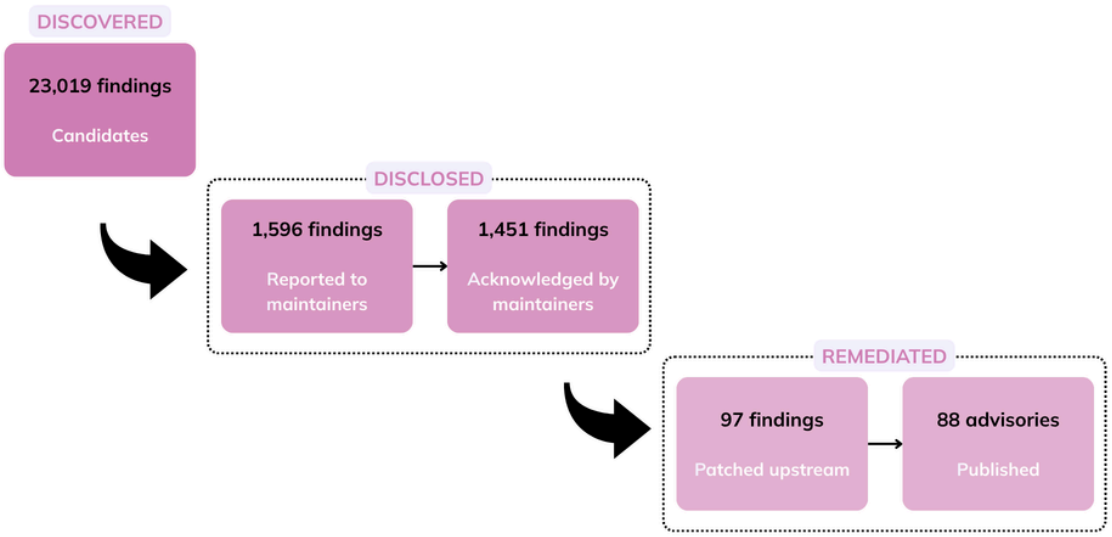

Fig. 2: *Mythos vulnerability disclosure dashboard (as of May 22, 2026)*

Mozilla's Firefox work turned that claim into evidence. Firefox 150 shipped fixes for 271 vulnerabilities identified during an initial evaluation of Mythos, and Mozilla later reported fixing 423 security bugs across April 2026 using a mix of Mythos, other AI models, fuzzing, manual inspection, and its existing security pipeline. The number is striking, but the lesson is not the count. It is the target. Firefox is a hardened codebase with mature fuzzing, deep internal security expertise, and years of defense-in-depth, and it still held a large body of latent issues that became discoverable and fixable once AI was added to the lifecycle. Most enterprise software is far less hardened than Firefox.

This is not the story of one lab. Independent evaluation by the UK AI Security Institute found that Mythos was a meaningful step up over previous frontier models and could execute multi-stage attacks against vulnerable networks in controlled settings. AISI also tested OpenAI's GPT-5.5 and rated it among the strongest cyber models it had assessed, able to complete one of its multi-step attack simulations end to end. The strategic question is no longer which model is "the best cyber model." It is that several frontier models are now capable enough to change the economics and tempo of vulnerability discovery, validation, and response, and access to that capability is widening.

AISI added a caution that points directly at where enterprises have leverage. Its cyber ranges had no active defenders and no defensive tooling, so the results should not be read as proof that these models can walk through a well-defended environment. Mozilla made a related point: a high-severity bug is not automatically a usable exploit. In a defense-in-depth architecture, an attacker often has to chain several weaknesses across sandbox, process, identity, and operating-system boundaries before a finding becomes a compromise. That is the hinge for everything that follows. AI sharply increases discovery pressure, but architecture, runtime controls, and containment still decide whether a discovered weakness ever becomes a business-impacting event.

2. Why Conventional Vulnerability Management Breaks

If the lifecycle is compressing, and even hardened code turns out to be full of discoverable weaknesses, the question for a security leader is immediate: what does that do to a program built for human-speed software?

It breaks two assumptions that most vulnerability management programs were quietly built on. Those programs assumed that software changes at a manageable rate, and that vulnerability discovery is rate-limited by human researchers, scheduled tests, and scanner coverage. AI invalidates both at once, and the compression shows up in two places: the software you ship, and the weaknesses attackers find in it.

The software you ship is growing faster

AI-assisted development is making it easier for engineering teams to produce and change software, and that productivity is genuinely valuable. It helps teams modernize old systems, write tests, improve documentation, and remediate faster. But it also increases the volume of code and configuration entering the environment: first-party application code, generated glue code and internal tools, APIs and service integrations, infrastructure-as-code, dependency and transitive-dependency updates, and agent-written scripts that automate operational work.

Each of these can introduce defects. Most will be low risk; some will be reachable and exploitable. The issue is not that "AI writes insecure code" - human developers do too. The issue is that the review, testing, ownership, and remediation machinery has to scale with a rate of change it was never sized for.

The weaknesses attackers find are surfacing faster

At the same time, AI is improving at exactly the tasks used to find exploitable weaknesses: code reasoning, reverse engineering, patch diffing, exploit validation, fuzzing assistance, log analysis, and attack-chain planning. Defenders gain from this, reviewing their own code faster, validating exploitability, generating safer patches, and writing detections. But attackers gain too. They can triage targets faster, weaponize public advisories faster, and hunt for zero-days in widely deployed software, and they do not need complete enterprise context to succeed. They need one path that is reachable, exploitable, and insufficiently contained.

The result is that these two pressures collide with a third constraint: the rate at which an organization can safely reduce risk. Conventional vulnerability management breaks at the mismatch between all three: how fast software changes, how fast vulnerabilities are discovered, and how fast the organization can actually mitigate. The first two are accelerating; the third is not. That gap produces more findings, shorter exploit windows, more no-patch scenarios, and more sustained pressure on AppSec, engineering, cloud, SOC, and incident response teams.

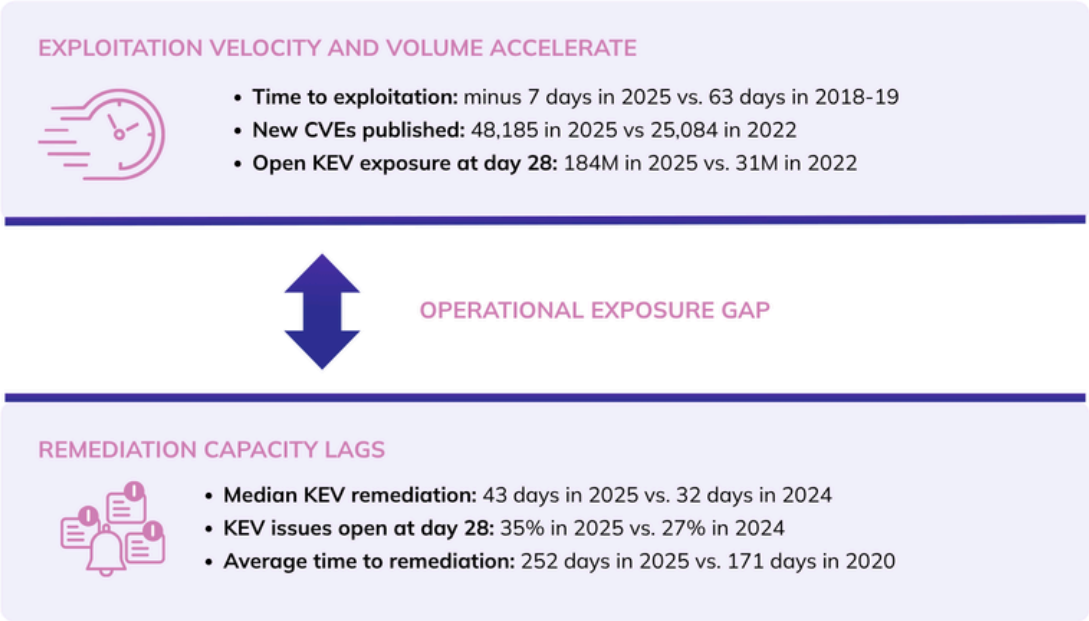

Fig. 3: *Rising exploitation pressure vs. remediation capacity challenges (sources: Verizon DBIR, NVD, Mandiant, Veracode)*

The instinct is to patch faster. That instinct is right, but incomplete. Unsupported software, exposed critical vulnerabilities, and neglected dependency backlogs will all be punished faster in an AI-accelerated environment, so patching matters more than ever. It simply cannot carry the program alone, because patch-first programs rest on a set of assumptions that AI turns from weak to brittle:

1. Severity scores are a reliable proxy for business risk.
2. Human triage can keep up with finding volume.
3. Every AI-generated finding deserves an engineering ticket.
4. A vendor patch will exist before exploitation begins.
5. Engineering can safely deploy every urgent fix immediately.
6. Supplier compromise is someone else's incident.
7. Detection and response are separate from vulnerability management.
8. The board should measure vulnerability risk by open finding count.

The signal-to-noise problem makes this worse. AI will generate more findings, but not all of them will be real, reachable, or urgent. The gap between raw findings and genuine business risk is already large and getting larger. In one environment Averlon analyzed, spanning roughly 140,000 assets, about 255,000 raw vulnerabilities reduced to roughly 500 validated reachable and exploitable exposures, concentrated on 100 assets. That is the funnel security teams must operate: from hundreds of thousands of findings, to the few hundred that are real and reachable, to the few dozen that sit on critical paths. Routing speculative results straight into engineering queues burns scarce developer attention and erodes trust in the program. Mature programs will validate before they assign work: exploitability evidence, reachability context, deduplication, confidence scoring, and clear ownership.

So the useful question is no longer "How many critical vulnerabilities do we have?" It is:

Which reachable, exploitable weaknesses could create business impact before we can patch, and what controls reduce that risk now?

That question changes the operating model.

3. The New Operating Model: Exposure-Led VulnOps

Security leaders should evolve from vulnerability management as a ticketing function to Vulnerability Operations, or VulnOps*: a continuous operating model that combines discovery, prioritization, remediation, mitigation, detection, and response.

The core principle is exposure-led prioritization.

Every meaningful vulnerability decision should start with exposure. Is the affected asset internet-facing or reachable from an exposed path? Is the vulnerable function actually used in production? Can attacker-controlled input reach it? From there, teams should weigh known exploitation, credible exploit code, business criticality, supplier ownership, compensating controls, and the organization's ability to patch, mitigate, isolate, or recover.

AI can help at several points in this workflow. It can validate whether a finding is likely exploitable in the organization's implementation, analyze dependency changes and patch impact, generate remediation PRs for human review, write regression and security tests, summarize advisory impact for asset owners, convert vulnerability context into detection logic, and assist incident responders during exploitation.

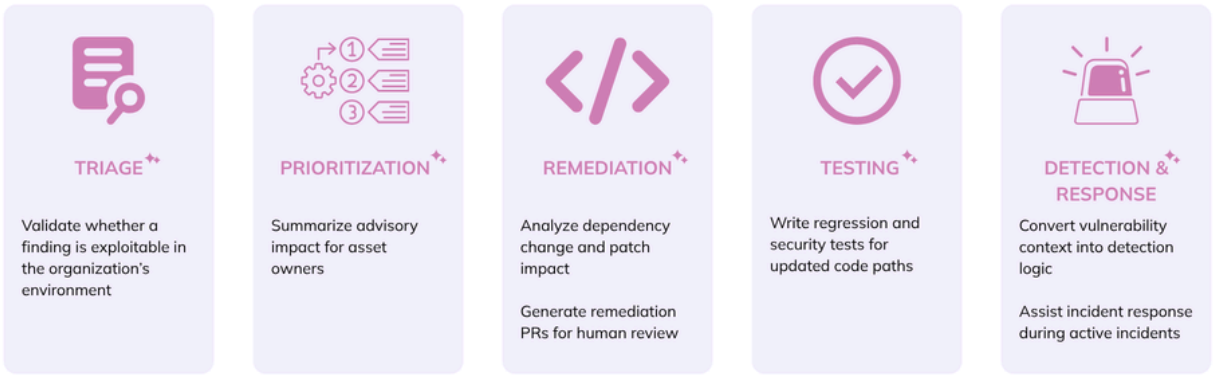

Fig. 4: *AI aiding VulnOps*

** Also referred to as Remediation Operations (RemOps)*

But AI should not become a new black box. AI-generated fixes and exploitability judgments need review, test evidence, and clear ownership. The operating model should use AI to increase throughput and context, not to remove accountability.

Reachability is the most important filter. The presence of a vulnerable component somewhere in the estate is not the same as business exposure. Security teams need to know whether the vulnerable code path is invoked, whether external or lower-trust input can reach it, and whether the asset sits on a path to sensitive systems or data. This is especially important for shared libraries and open-source dependencies, where a single vulnerable package may appear in hundreds of applications but be exploitable in only a subset.

Compensating controls should change priority, not merely document exceptions. A severe vulnerability behind effective segmentation, runtime blocking, and no attacker-controlled path should be treated differently from a lower-severity vulnerability that is reachable on an exposed business system.

From tickets to risk-reduction actions

The unit of work should shift from "file a vulnerability ticket" to "reduce exploitable exposure." Sometimes that means patching. Sometimes it means disabling a vulnerable feature, blocking an attack path at the edge, rotating credentials, isolating a workload, limiting egress, applying a configuration change, or deploying additional runtime detection.

This is where vulnerability management must merge with cloud security, application security, identity, SOC, and resilience engineering. In the AI era, a vulnerability that cannot be patched quickly is still a security operations problem.

4. The Second-Order Risk: Suppliers, OSS, and Trusted Dependencies

Even if an enterprise does a strong job managing its own vulnerabilities, it can still be compromised through the software and services it trusts. ***This is the second-order effect of AI-accelerated vulnerability discovery.*** Attackers will not only look for flaws in the enterprise's first-party applications. They will look for high-leverage weaknesses in commercial software, SaaS platforms, identity providers, CI/CD tools, open-source packages, MSPs, and update channels. A single supplier compromise can become many customer compromises.

This does not mean enterprises should try to rip out open source. That is not realistic, and it would often make security worse by replacing mature community-reviewed components with less tested internal code. Nor should the strategy depend on every open-source maintainer producing perfect SBOM or VEX data. Most will not.

The practical approach is different for commercial suppliers and open source.

For commercial suppliers

Security leaders should treat critical suppliers as part of the vulnerability management surface. This means maintaining an inventory of suppliers by privilege, connectivity, data access, business dependency, and update mechanism.

For the most critical suppliers, enterprises should require security notification and escalation paths that work during an emergency, clear vulnerability disclosure and patch communication processes, evidence of secure development practices, contractual expectations for incident notification and support during active exploitation, and recovery planning for supplier outage or compromise. SBOM and VEX data can be useful where available, but neither should be treated as a complete answer.

For open source

Enterprises should assume they must generate their own transparency. That means producing SBOMs from actual builds, maintaining dependency inventories tied to application ownership and production exposure, using SCA and reachability analysis, monitoring package reputation and exploit intelligence, pinning and verifying dependencies, using controlled artifact repositories and package firewalls, applying cooldown periods for new package versions where appropriate, watching for abandoned or single-maintainer projects in critical paths, and contributing fixes or funding where the business depends on critical OSS projects.

The most important action is to identify which dependencies actually matter. A critical vulnerability in an unused code path is not the same as a medium-severity flaw in a reachable authentication path for an internet-facing application.

Supply chain security in this context is not a procurement checklist. It is operational exposure management.

5. The No-Patch Window: Runtime Protection & Compensating Controls

AI-driven vulnerability discovery increases the likelihood that organizations will face no-patch windows.

A no-patch window can happen because the vulnerability is a true zero-day and no vendor fix exists; because the supplier has a fix but the enterprise cannot deploy it safely yet; because the affected system is legacy, fragile, or operationally constrained; because the vulnerable component is embedded in a product, appliance, or third-party service; or because the exposed path is known before ownership, patch impact, or asset scope is understood.

In these situations, "patch faster" is necessary but insufficient. Simply compressing patch SLAs does not remove the engineering work required to test, deploy, and roll back safely. If regression testing takes a day, a two-hour patch target either fails or creates pressure to skip testing and risk breaking production. Organizations need a mitigation control plane that can reduce exposure while remediation catches up.

Runtime protection is central to that control plane. For internet-facing applications and APIs, that may mean WAF, WAAP, virtual patching, API protection, abuse detection, or application-layer exploit blocking. For cloud and workload environments, it may mean container runtime protection, EDR and XDR tied to vulnerability context, egress filtering, microsegmentation, least privilege, just-in-time access, and scoped tokens. For high-risk integrations, it may mean canary tokens, deception, feature flags, kill switches, or emergency configuration changes that disable risky paths.

The goal is not to pretend these controls replace remediation. They buy time. They reduce exploitability, limit blast radius, and create detection opportunities when a patch is unavailable, delayed, or incomplete.

The strategic goal is to make the disclosure-to-patch gap matter less. Architecture and runtime controls should make it harder for an attacker to reach vulnerable code, harder to turn a bug into privilege, harder to move laterally, and harder to exfiltrate data. In an AI-accelerated threat environment, this matters as much as reducing mean time to patch.

This is where many vulnerability management programs are underdeveloped. They can identify a vulnerable asset, but they cannot quickly answer: "What can we block, isolate, disable, or monitor in the next four hours?"

That question should become a standard part of vulnerability triage.

6. Detection, Containment, and Recovery Are Now Part of Vuln Management

Vulnerability resilience cannot stop at remediation. Security teams should assume some vulnerabilities will be exploited before they are patched.

This makes detection, containment, and recovery part of the same program.

Detection

Detection should be built from vulnerability context. Security teams need to know which assets are exposed, which endpoints, APIs, or functions are vulnerable, what exploitation would look like in logs, traces, process activity, network flow, identity events, or cloud control-plane telemetry, and which signals indicate scanning, exploit attempts, post-exploitation, or data access.

AI can help translate advisory detail into candidate detections, but detections must be tested and tuned. The strongest programs will connect vulnerability intelligence, asset context, runtime telemetry, and SOC workflows.

Containment

Containment actions should be pre-authorized for high-risk scenarios. During a no-patch window, teams should not need to negotiate from scratch whether they can isolate a workload, restrict egress, disable an integration, revoke tokens, block an endpoint, or apply an emergency WAF rule.

The playbook should specify who can make the call, what evidence is needed, what business impact is acceptable, and how exceptions are approved.

Recovery

Recovery is often treated as a business continuity function separate from vulnerability management. That separation is no longer defensible.

If AI reduces exploit timelines, then critical vulnerable systems need tested recovery paths: known-good backups, clean rebuild processes, golden images and hardened baselines, tested restoration for critical services, credential rotation procedures, data integrity validation, and manual workarounds for critical business processes.

For the board, recovery time and containment time are now vulnerability risk metrics.

7. A 90-Day+ Action Plan for CISOs

This week

1	**Change the executive narrative**	Explain that AI affects vulnerability management through both software velocity and discovery velocity, and set the expectation that patching remains essential but will not be enough.
2	**Identify the top exposed business systems**	Start with internet-facing assets, identity systems, remote access, CI/CD, critical APIs, and crown-jewel applications.
3	**Start exposure-led reprioritization**	Re-rank the highest-risk slice of the backlog for the top exposed business systems by reachability, exploitability, business criticality, and available compensating controls. Use CISA KEV and exploit intelligence as input, not the only input, and expand the model over the next 30-90 days.
4	**Start a no-patch mitigation playbook**	Define emergency controls such as WAF rules, egress restrictions, isolation, token revocation, feature disablement, and increased monitoring.
5	**Validate supplier emergency contacts**	Confirm who to call for the top 10-25 critical suppliers during active exploitation or urgent patch wave.

First 30 days

6	**Build an exposure-led scorecard**	Track reachable exploitable vulnerabilities, internet-facing risk, critical supplier dependencies, and mitigation status.
7	**Map critical supplier and OSS dependencies**	Focus on systems with high privilege, high connectivity, sensitive data, or direct business dependency. Generate your own dependency transparency where supplier data is incomplete.
8	**Put AI into the defensive workflow**	Use authorized AI tooling for secure code review, triage, exploitability analysis, patch impact analysis, detection drafting, and remediation PRs. Add a validation gate before AI-generated findings reach engineering queues.

First 30 days (continued)

9	**Establish runtime protection coverage for critical exposed services**	Prioritize external applications, APIs, high-risk workloads, and systems where patching is historically slow.
10	**Run a no-patch tabletop**	Use a scenario where a critical supplier or open-source component is actively exploited before a patch is available.
11	**Inventory recovery readiness for crown-jewel systems**	Confirm which critical systems have known-good backups, tested restoration paths, and documented credential-rotation procedures, and flag the gaps for closure.

30-90 days

12	**Stand up a VulnOps operating cadence**	Bring AppSec, cloud security, platform engineering, SOC, identity, and incident response into one risk-reduction workflow.
13	**Automate the path from advisory to exposure answer**	The target question is: "Where are we exposed, what is reachable, and what can we mitigate today?"
14	**Build emergency change paths**	Accelerate security changes while preserving test evidence, rollback, and auditability.
15	**Expand runtime and compensating controls**	Integrate WAF/WAAP, API protection, workload runtime telemetry, EDR/XDR, egress filtering, and segmentation into vulnerability response.
16	**Test containment and recovery**	Measure time to isolate a vulnerable service, disable a supplier integration, rotate exposed credentials, and restore a critical system.

Post 90 days

17	**Mature VulnOps into a permanent function**	Staff it like an engineering and operations function, not a ticket queue.
18	**Build automated remediation pipelines**	Generate fixes, tests, patch-impact analysis, and change requests with human approval and rollback.
19	**Manage critical OSS as strategic infrastructure**	Identify, monitor, fund, fork, replace, or isolate the few components that create material enterprise risk.
20	**Shift board reporting from activity to resilience**	Report risk reduction, mitigation speed, runtime coverage, containment readiness, and recovery evidence.

8. Board Metrics That Matter Now

Traditional vulnerability metrics are not useless, but they are incomplete. The board needs to see whether the organization can reduce risk at the speed required.

The core board view should focus on a small set of resilience metrics:

1. Reachable, exploitable vulnerabilities on critical assets.
2. Internet-facing critical vulnerabilities without compensating controls.
3. Mean time to mitigate, not only mean time to patch.
4. Percentage of critical exposed services covered by runtime protection.
5. Percentage of critical vulnerabilities with detection coverage.
6. Number of no-patch exposures with approved compensating controls.
7. Mean time to contain exploitation of a vulnerable asset.
8. Mean time to restore critical services to a known-good state.

Operating teams should track additional indicators underneath the board view, including supplier-originated critical exposures by business process, time to isolate or disable a compromised supplier integration, critical OSS dependencies with owner and exposure review, emergency change lead time for critical security fixes, vulnerability exceptions older than 30, 60, and 90 days, and AI-assisted triage and remediation coverage.

The board should not only ask, "Are we patching?" It should ask, "How quickly can we know where we are exposed, reduce exploitability, contain impact, and recover?"

9. The Security Leader's Mandate

The next phase of vulnerability management will not be won by buying more scanners or producing larger backlogs.

Security leaders need to make four commitments.

1. Prioritize what can be exploited

Replace severity-only queues with exposure-led prioritization. Focus on reachable, exploitable, business-relevant weaknesses.

2. Use AI defensively, with governance

Defenders should use AI to scale review, triage, remediation, detection, and response. But AI outputs must be governed with identity, logging, human review, and test evidence.

3. Build the no-patch control plane

Treat runtime protection, virtual patching, segmentation, egress filtering, least privilege, and deception as vulnerability management controls. These are the controls that matter when remediation is unavailable or delayed.

4. Measure resilience, not just remediation

Containment and recovery are no longer downstream incident response topics. They are essential measures of vulnerability risk.

Conclusion

Mythos and GPT-5.5 are not isolated events. They are signals of a broader shift in software security. AI is increasing the speed at which software is created and changed. AI is also increasing the speed at which vulnerabilities can be found, validated, and operationalized. Between those two forces, conventional vulnerability management programs will face more findings, shorter response windows, more supplier exposure, and more zero-day or no-patch scenarios.

The answer is not panic. It is an operating-model change.

Security leaders should patch faster where they can, but also assume that patching will sometimes be late, unavailable, or outside their control. The programs that handle this transition best will know what matters, reduce exposure quickly, use AI to accelerate defenders, deploy runtime protection during the no-patch window, and prove they can contain and recover when exploitation occurs.

That is the shift from vulnerability management to vulnerability resilience.

Selected Sources

1. Anthropic, Project Glasswing: Securing critical software for the AI era
2. Anthropic, Assessing Claude Mythos Preview's cybersecurity capabilities
3. Anthropic, Coordinated Vulnerability Disclosure Dashboard
4. Mozilla, The zero-days are numbered
5. Mozilla Hacks, Behind the Scenes Hardening Firefox with Claude Mythos Preview
6. UK AI Security Institute, Our evaluation of Claude Mythos Preview's cyber capabilities
7. UK AI Security Institute, Our evaluation of OpenAI's GPT-5.5 cyber capabilities
8. OpenAI, GPT-5.5 System Card
9. JPMorganChase, Fortifying the enterprise: 10 actions to take now for AI-ready cyber resilience
10. OWASP, Virtual Patching Cheat Sheet
11. NIST, SP 800-218 Secure Software Development Framework
12. NIST, SP 800-161 Rev. 1 Cybersecurity Supply Chain Risk Management Practices
13. CISA, Known Exploited Vulnerabilities Catalog
14. Verizon, Data Breach Investigation Report, 2026
15. Veracode, State of Software Security, 2025

ABOUT AVERLON

Averlon was built to solve the hardest part of vulnerability management: getting from findings to fixes without breaking production or burning out engineering teams. Its Remediation Operations platform reduces real risk and closes the Exposure Window across code, cloud, and infrastructure. Founded in 2022, Averlon is trusted by security teams at startups and global enterprises and backed by Salesforce Ventures, Voyager Capital, and Outpost Ventures.

info@averlon.io