



Zero-Day, Zero Delay: Your Mythos & AI-Era Vulnerability Guide

Preparing for, responding to, and surviving the attacks you never saw coming — in an era when your adversaries move at machine speed.



Contents

Foreword: The Rules Just Changed	3
Part One: Understanding the Threat	4
Part Two: Before It Happens	8
Part Three: Anatomy of a Bad Response	12
Part Four: The Hour-by-Hour Playbook	14
Part Five: How Harness Compresses Zero-Day Response	15
Conclusion: The Difference Between Hours and Forever	17
Appendix: Zero-Day Preparedness Checklist	18

FOREWORD

The Rules Just Changed

There has always been a gap between when a vulnerability is disclosed and when attackers exploit it. For years, that gap was measured in days or weeks — uncomfortable, certainly, but manageable with practiced response teams and solid processes.

That gap has collapsed.

In early 2026, Anthropic broke the news of [Mythos and Project Glasswing](#). This announcement demonstrated something the security industry had long anticipated but never fully reckoned with: a frontier AI model autonomously discovering thousands of zero-day vulnerabilities, including critical flaws in every major operating system and web browser, many of which had survived decades of human review and millions of automated security tests.

The defensive implications are profound. The offensive implications are just as significant. The same capabilities that allow an AI to find and patch a 27-year-old vulnerability in OpenBSD are available — or will soon be — to adversaries who have no interest in responsible disclosure. Attackers who once needed weeks to develop a usable exploit after a CVE dropped will soon need hours. Some already do.

This guide was written for that world.

Everything in these pages — the playbooks, the checklists, the response frameworks — is designed around a simple premise: in a zero-day event, your speed of understanding is your speed to survival. Not the speed of your patch. Not the skill of your engineers. Your ability to answer “what do we have, where is it exposed, and who owns the fix” within minutes, not days, is what separates a contained incident from a catastrophic one.

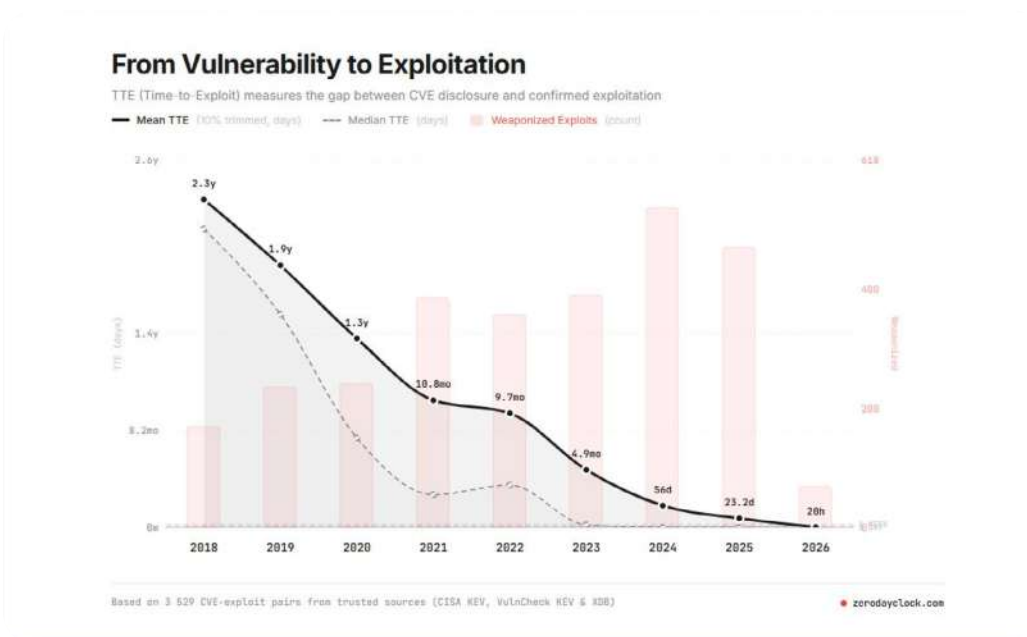
That has always been true. It is truer now than ever before.

PART ONE

Understanding the Threat

To prepare for a zero-day exploit, start by setting aside everything you think you know about security breaches. Zero-day events are fundamentally different from incidents triggered by known CVEs, and they require a fundamentally different posture.

When a zero-day is disclosed, you are immediately in two races simultaneously: a race to understand your exposure, and a race against adversaries who are already developing exploits. In the AI era, that second race is no longer a fair fight. Attackers with access to capable AI models can go from disclosure to working exploit in hours. The organizations that survive zero-day events are the ones that have done the preparation work before the clock starts.



Log4Shell: The Zero-Day Anatomy Lesson

No modern case study illustrates zero-day dynamics better than Log4Shell. It remains the clearest example of how an unknown flaw becomes a global crisis — and a long-tail risk that organizations are still managing years later.

The vulnerability lived in Apache Log4j, in a feature called JNDI lookups that allowed log entries to dynamically fetch and execute remote data. The capability required no authentication and could be triggered through simple user input — an HTTP header, a form field, a chat message. Untrusted input was treated as executable behavior.

When a proof-of-concept was posted to GitHub in December 2021, attackers didn't wait. Within hours, they were scanning web servers, APIs, and cloud services at scale. The initial burst of exploitation was fast and chaotic. What followed was worse: weeks of incomplete patches, bypassed mitigations, and repeated patching cycles. Attackers used the foothold to deploy malware, establish persistence, and move laterally across environments that lacked segmentation or least-privilege controls.

The deepest problem wasn't the vulnerability itself. It was the invisibility of it. Log4j was everywhere, embedded in frameworks, bundled in third-party software, and included in SaaS platforms. Organizations didn't know they were using it. They couldn't search for what they didn't know they had.

Zero-days don't end when patches are released. They become long-term attack vectors in every environment that wasn't fully remediated and 'fully remediated' is harder than it sounds.

Even today, unpatched Log4j instances exist in forgotten containers, legacy systems, and old images. The lesson is as relevant now as it was in 2021: the blast radius of a zero-day is determined by how well you know your own environment, and most organizations know theirs less well than they think.

Now layer on what AI changes. A Mythos-class model could autonomously discover a Log4Shell-equivalent flaw before public disclosure, develop a working exploit, and identify vulnerable targets at a scale and speed no human attacker team could match. The 27-year-old vulnerability in OpenBSD that Project Glasswing's model found — in a system with a reputation for extraordinary security hardening — is exactly the kind of deeply buried flaw that becomes the next Log4Shell. The difference is that it was found by a defender first. That won't always be the case.

Why Modern Supply Chains Amplify Blast Radius

Zero-day exploits are almost purpose-built to harm today's interconnected software world. Modern applications are layered compositions of open source libraries and frameworks, each with their own dependencies — a structure that means one vulnerability can live in thousands of applications across hundreds of organizations, buried so deep that it's nearly invisible.

The attack surface has expanded in every direction. Third-party SaaS providers, managed services, APIs, and vendor software are all part of your perimeter now. If they're vulnerable, your

organization is vulnerable — but you have no control over their response timelines and no visibility into their systems.

Container images persist indefinitely. CI/CD pipelines, often in the hundreds or thousands, propagate changes across environments at speed, including vulnerabilities. Few organizations maintain current Software Bills of Materials, which means they lack an accurate inventory of the components running in production.

And now add AI-augmented adversaries. An attacker with access to a capable AI model can autonomously traverse dependency trees, identify which transitive libraries contain a vulnerable component, and prioritize targets by exposure and privilege. The structural vulnerabilities that have always defined modern supply chains now face an adversary with the patience and analytical capability to exploit all of them simultaneously.

Modern supply chains were designed for speed, reuse, and scale — not containment. Without visibility, control, and context, a single vulnerability can cascade across systems, teams, and vendors almost instantly.

Direct vs. Transitive Dependencies: Where Real Risk Lives

One of the most consequential distinctions in supply chain security is the one between dependencies you chose and dependencies you inherited. Direct dependencies are intentional. Transitive dependencies — the libraries your libraries depend on, and their dependencies in turn — define your actual risk profile, and they are almost always less visible.

When a zero-day hits a transitive dependency, the response timeline immediately bifurcates. Direct dependencies can be patched quickly. Transitive exposure requires discovery first, mapping which components in which applications pull in the vulnerable library, often several layers deep and then waiting for upstream maintainers to release fixes before you can act. That discovery-to-action gap is where attackers have historically operated most freely.

The practical implication: your true security posture in a zero-day event is defined less by what you build and more by what your dependency chain carries with it. Organizations that have mapped this landscape before a crisis have a decisive advantage over those mapping it under fire.

The Three Phases: Panic, Triage, Recovery — And How to Compress Them

Every organization moves through the same three stages in a zero-day event. Understanding them in advance is the first step toward moving through them faster.

Panic

Slack fills up. The C-suite wants answers. Engineers search codebases manually while security pulls incomplete reports. The defining characteristic of this phase is that exposure level is unknown — and without a reliable dependency inventory, it stays unknown far longer than it should.

The Panic phase is made worse by something most teams underestimate: the moment a zero-day disclosure goes public, attackers know it too. In an era of AI-assisted exploitation, they are retooling within minutes. Every hour spent in undirected searching is an hour that gap narrows.

Triage

Teams begin generating lists of potentially vulnerable systems. Scanners produce alerts. The challenge is that volume without context is noise. Teams need to know not just what contains the vulnerable component, but what's reachable from the internet, what privilege level is at stake, and whether the vulnerable code path is actually being executed. Without that context, triage becomes educated guessing — and the result is patching everything indiscriminately, which is both slow and prone to breaking things.

Compensating controls matter here. A WAF configured to block exploit patterns can buy critical time while patches are developed and tested. Don't wait until triage to know how to deploy those controls.

Recovery

Patches are applied, but this phase is rarely clean. Initial patches are sometimes incomplete. Mitigations get bypassed. The full scope of which systems were patched, which weren't, and whether the exploit was used before remediation is often unclear. Organizations with strong pipeline automation can rebuild and redeploy systematically. Those without it patch manually, inconsistently, and never reach a confident "all clear" moment.

**The speed of your response is not determined by how fast you can patch.
It's determined by how fast you can understand and control your exposure.**

Compressing these phases requires three things working together: visibility into what you have and where it's exposed, context that lets you prioritize what matters most, and pipeline automation that makes remediation systematic rather than heroic. None of these can be built during the incident. They must exist before it.

PART TWO

Before It Happens

The organizations that respond fastest to zero-day events are not the ones with the most skilled incident responders. They are the ones that did the quiet, unglamorous preparation work when nothing was on fire. Here is what that work looks like.

Dependency Inventory: Know What You Have

You cannot search for what you don't know exists. The single most valuable investment an organization can make in zero-day preparedness is maintaining a living, accurate inventory of every dependency in production, both direct and transitive.

That inventory needs to span application-level dependencies, infrastructure and platform dependencies, and the runtime context that tells you which version of what is actually running where. It needs to be current, not just a snapshot from the last quarterly review, but updated with every build. And it needs to be queryable, so that when a zero-day drops at 11pm on a Friday, your team can answer "do we use this?" in minutes, not days.

SBOMs: Your Incident Response Shortcut

A Software Bill of Materials is the mechanism that makes dependency visibility operational. An SBOM for every application gives you something concrete to query when a vulnerability is disclosed.

The most effective way to maintain SBOMs is to generate them automatically as part of every CI/CD build. This removes the compliance theater of quarterly SBOM generation and ensures the inventory reflects what's actually running. Store them in a centralized location every member of the team knows how to access. Confirm they include transitive dependencies, not just direct ones.

An SBOM you can't find when you need it is not better than no SBOM. The storage, access, and query workflow matters as much as the generation.

Incident Response Roles: Decide Before the Fire

The worst moment to discover who owns the incident commander role is during the incident. Designate roles in advance and make sure every person understands their responsibilities:

- Incident commander — owns the decision-making process and timeline
- Security lead — drives technical assessment and prioritization
- Application and service owners — provide context on affected systems
- Infrastructure and platform lead — coordinates environment-level response
- Observability and telemetry lead — owns log and trace analysis
- Communications lead — manages executive and external stakeholder updates
- Scribe — maintains the timeline and audit trail for post-incident review

These roles need to be documented, known, and rehearsed — not discovered under pressure.

Severity Thresholds and SLAs

Define your response standards before an incident forces the conversation. The following framework provides a starting point — treat these as maximum acceptable windows, not comfortable targets. AI-assisted exploitation means the gap between disclosure and active attack is narrower than it has ever been.

Severity 1 — Critical / Active Zero-Day

Trigger conditions: known active exploitation in the wild, internet-facing service affected, remote code execution or auth bypass capability, high-confidence exposure in your environment.

- Incident declared: ≤ 15 minutes
- Triage initiated: immediate
- Containment action deployed: ≤ 1 hour
- Mitigation (patch or compensating control): target 4–8 hours; maximum 24 hours
- Stakeholder updates: every 30–60 minutes

This is a drop-everything response. No other work takes priority.

Severity 2 — High Risk, No Confirmed Exploitation

Critical vulnerability, no confirmed active exploitation, internal services exposed, exploit likely but not observed.

- Incident declared: ≤ 1 hour
- Triage: ≤ 2 hours
- Mitigation: ≤ 48 hours
- Updates: every 2–4 hours

Severity 3 — Moderate Risk / Limited Exposure

- Vulnerability requires complex conditions to exploit
- Limited blast radius, compensating controls exist
- Triage: ≤ 24 hours | Mitigation: ≤ 3 –5 days

Severity 4 — Low Risk

No immediate action required. Address in normal patch cycles.

The Zero-Day Fire Drill

You will not perform well in an incident you have never rehearsed. Run a zero-day simulation at least quarterly. Make it realistic, which means making it messy, ambiguous, and chaotic. The simulation that feels uncomfortable is the one that actually prepares your team.

Start with a realistic scenario drawn from your actual stack: a vulnerable transitive dependency in a widely used library, a compromised container base image, an auth bypass in an API gateway, a supply chain compromise in CI/CD. Reveal information progressively rather than all at once. Use your real incident channels, your real roles, and your real SLA thresholds.

Include an AI-speed attacker scenario where the window between disclosure and active exploitation is measured in hours, not days. Your team's responses will be different when they feel the actual time pressure.

The goal is not to test whether your team can find the vulnerability. It's to test whether they can find it fast, prioritize correctly, make good decisions under pressure, and coordinate without confusion. Run a structured debrief afterward and implement changes before the next drill.

Blast Radius Calculation

Before a zero-day hits is the right time to build your blast radius model, a repeatable, data-backed framework for estimating how far a given exploit can spread and how much damage it can do. The formula:

$$\text{Affected Assets} \times \text{Exposure} \times \text{Privilege} \times \text{Connectivity} \times \text{Data Sensitivity}$$

Score each dimension 1–5 for each affected system. Exposure is usually the biggest multiplier: a public internet-facing service scores 5; an isolated internal system scores 1. Add a spread-potential multiplier (2–3×) for shared libraries across many services, common base images, shared credentials, or flat network architecture.

Map the results visually. Service dependency maps and network connectivity graphs reveal what numbers alone don't: which nodes, if compromised, cascade into everything else. That's where containment effort should concentrate.



PART THREE

Anatomy of a Bad Response

Understanding what a failed zero-day response looks like in detail is valuable preparation. Most failures are not caused by a single catastrophic decision. They are caused by the accumulation of small structural gaps that compound under pressure.

Stage 1: The Ownership Vacuum

The zero-day is disclosed publicly. Multiple teams notice it. No one owns it. Between 15 minutes and an hour passes before anyone formally declares an incident. In that window, the gap between disclosure and exploitation is closing. The root causes: no incident commander designated in advance, no clear severity threshold to trigger response, over-reliance on CVSS scores that don't reflect actual exploitability in the specific environment.

Stage 2: Blind Triage

Security begins analyzing the vulnerability while engineering teams manually grep codebases and scan manifests. Hours or days pass just identifying affected systems. Multiple teams do duplicative work. No one has a clear picture because there is no SBOM, no dependency map, and no way to distinguish direct from transitive exposure quickly.

Stage 3: Scope Without Context

Some teams report they are affected. Others say they are not. Leadership wants an impact assessment. The organization oscillates between panic and false reassurance. No blast radius model exists. No one knows which systems are internet-facing, what data they touch, or how connected they are to other services. The absence of runtime context means theoretical exposure cannot be translated into prioritized action.

Stage 4: Decision Paralysis

With incomplete information and no pre-approved decision authority, teams debate rather than act. Patch vs. workaround. Shut down vs. stay up. Wait for guidance vs. act now. The vulnerability is known. Time is passing. The exploit window is open. Nothing is being done because the decision framework that should have been built before the incident doesn't exist.

Stage 5: Fragmented Containment

Teams eventually act, but without coordination. Some patch, some block traffic, and some do nothing but wait. High-risk internet-facing systems may remain exposed while low-risk internal tools get patched first. There is no centralized view of what has been done, what hasn't, and what still needs to happen. Containment is incomplete by design.

Stage 6: Communication Breakdown

Executives demand updates. Customers begin asking questions. Engineers are pulled off response work to brief stakeholders. Multiple teams send conflicting messages. There is no communications lead, no update cadence, and no single source of truth. The response itself is being degraded by the noise around it.

Stage 7: Incomplete Recovery

Patches are eventually applied, systems come back online, and the incident is declared closed. But there is no forensic confidence about whether the exploit was used before remediation. Some systems were missed. There is no rollback strategy if a patch causes regressions. The “all clear” moment is a social construct, not a verified state.

Stage 8: Permanent Residual Risk

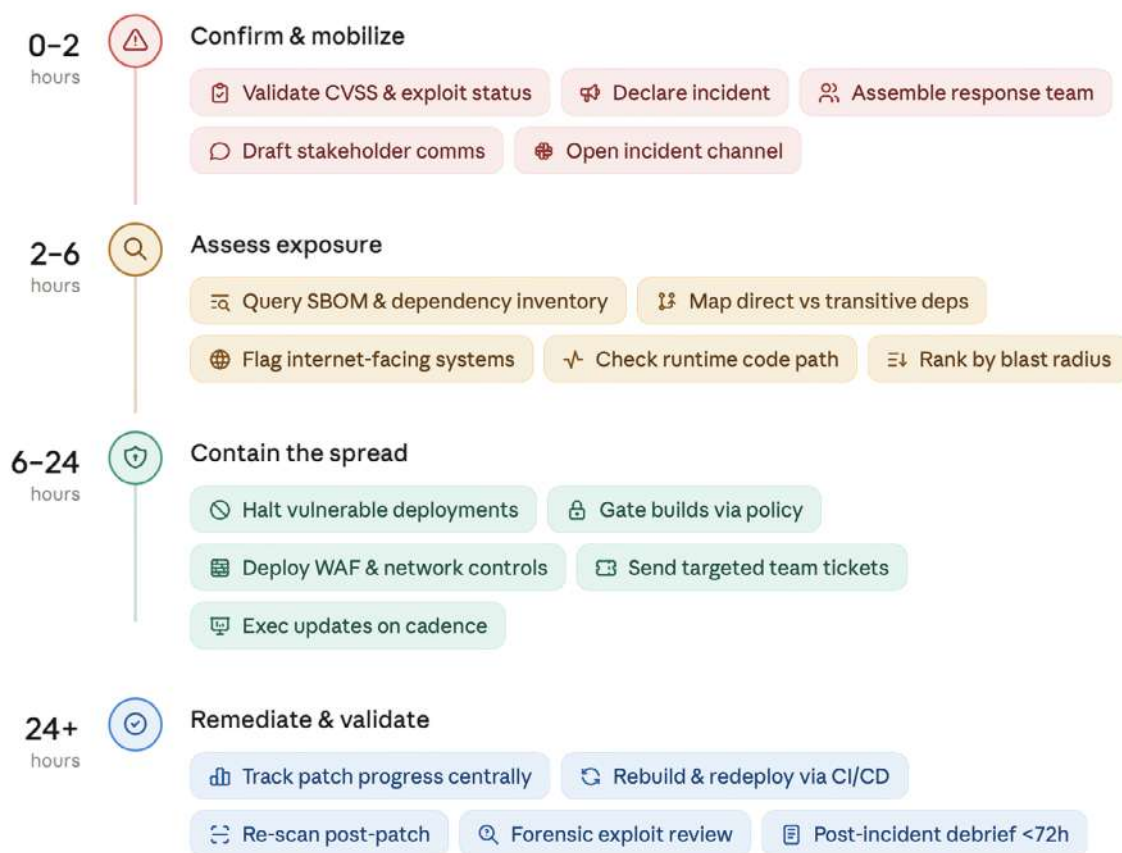
The incident is over on the timeline. It is not over in the environment. Forgotten containers, old images, and unpatched legacy systems carry the vulnerability forward. Long-tail cleanup continues for months, managed informally and without tracking. The next disclosure finds the same gaps.

In an AI-era incident, each of these failure modes costs more than it used to. A 15-minute ownership vacuum, hours of manual searching, fragmented containment — these gaps that once cost hours may now cost the entire incident. Build your response process assuming your adversary is moving at machine speed, because increasingly, they are.

PART FOUR

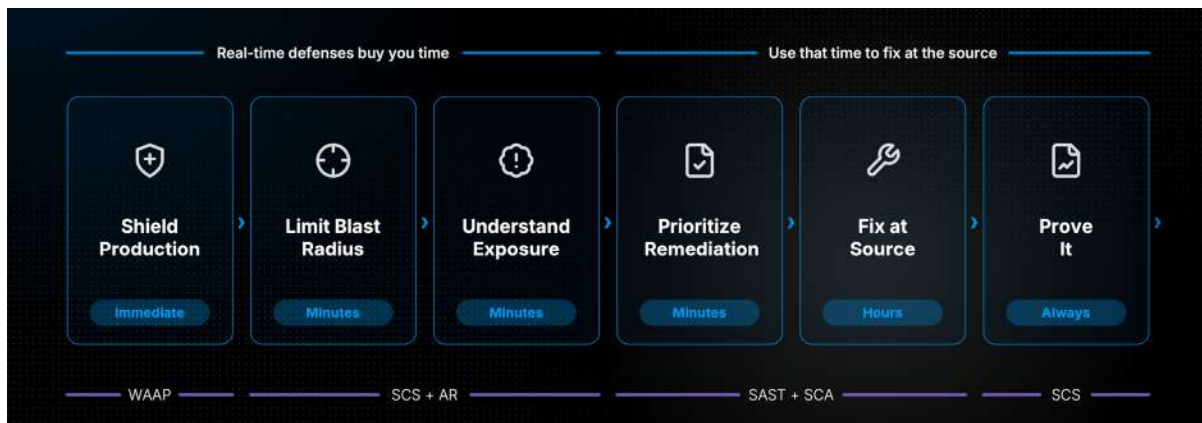
The Hour-by-Hour Playbook

When a zero-day drops, decisions made in the first two hours determine the character of the entire incident. Here is the framework for moving through each phase with purpose.



How Harness Compresses Zero-Day Response

Every part of this guide points toward the same conclusion: the organizations that respond fastest to zero-day events are the ones that have already done the work of building visibility, context, and pipeline control into their day-to-day operations. Harness is designed to make that work systematic.



Find Your Exposure in Minutes, Not Days

Harness Supply Chain Security generates SBOMs automatically as part of every build and stores them centrally. When a zero-day is disclosed, you don't search — you query. Finding every pipeline, container, and artifact that references an affected library goes from a multi-day manual exercise to a task measured in minutes. That compression matters most at the precise moment you can least afford delays.

Mobilize the Right Engineers Immediately

Harness SCS automatically creates scoped, trackable tickets for every affected pipeline — routed to the engineers who own them. During a live incident, the alternative is a security engineer manually filing dozens of tickets while simultaneously trying to triage the vulnerability itself. Automated mobilization at scale is not a convenience feature. When the exploit window is measured in hours, it is a core response capability.

Track Remediation Across the Entire Organization

A remediation dashboard that gives security leadership real-time fix progress across every affected team eliminates the most corrosive dynamic in zero-day response: status uncertainty. When no one knows what has been patched and what hasn't, organizations oscillate between false confidence and unnecessary panic. Central visibility replaces Slack archaeology with a source of truth.

Policy Enforcement That Gates Vulnerable Builds

In the containment phase, the priority is stopping the vulnerable component from spreading further while remediation proceeds. Harness pipeline policies can block any build that introduces the affected library version — automatically, across every pipeline, without requiring manual intervention at each one. This is containment at machine speed, which is the only speed that matters when your adversary operates the same way.

As AI-powered vulnerability discovery becomes available to both defenders and adversaries, the operational readiness that Harness enables is no longer a competitive advantage. It is the baseline survival infrastructure.

CONCLUSION

The Difference Between Hours and Forever

Zero-day incidents have always been won or lost in the preparation period, not during the incident itself. The teams that respond well are the ones that mapped their dependencies before the disclosure, wrote their playbooks before the Slack explosion, defined their SLAs before the C-suite started asking questions.

What has changed is the cost of not having done that work.

We are in a new era of cybersecurity, one defined by AI systems capable of finding and exploiting vulnerabilities at a scale and speed that no human-paced process can match. The same frontier capabilities that defenders are beginning to deploy are available to adversaries who play by no rules at all. **The window between disclosure and exploitation has collapsed. The blast radius of a supply chain vulnerability has never been larger. The tools to map, contain, and remediate that exposure have never been more capable.**

The path forward is preparation, not heroics. It is the quiet work of building SBOM pipelines, mapping blast radius, running fire drills, and giving your teams the visibility they need to answer the question that decides every zero-day event: “what do we have, where is it exposed, and who owns the fix?”

Build the infrastructure to answer that question in minutes. Then, when the clock starts ticking — and it will — you will already be ahead.

APPENDIX

Zero-Day Preparedness Checklist

A practical assessment framework for security and engineering leaders. If any item below is a gap, that gap is your highest-priority preparation investment.

1. Asset and Dependency Visibility

Inventory

- Real-time inventory of all applications, services, APIs, endpoints, and infrastructure
- Ownership mapped for every system: team, on-call contact, escalation path

Dependency Mapping

- SBOM generated automatically for every application — built and purchased
- SBOMs include both direct and transitive dependencies
- All third-party library versions tracked and current
- SBOMs updated automatically on every CI/CD build
- SBOM storage location known to all relevant team members

Exposure Awareness

- All internet-facing systems identified and mapped
- Internal vs. external service boundaries documented
- High-risk entry points catalogued: auth, logging, parsing, deserialization, API gateways

2. Exposure Reduction

- API gateways or WAFs in place and configured to filter malicious payloads
- Schema validation enforced on all public APIs
- AI-assisted proactive vulnerability scanning explored or in use

- Least-privilege access enforced across services and environments
- Network segmentation in place to limit lateral movement

3. Telemetry and Observability

- Logs, metrics, and traces collected for all critical systems
- Telemetry enriched with service name, environment, version, and identity
- Able to answer within minutes: “Where is this library used?”
- Able to answer within minutes: “Which affected services are internet-facing?”
- Able to answer within minutes: “Is this vulnerable code path being executed in production?”

4. Incident Response Readiness

- Incident commander and all response roles designated and documented
- On-call rotations current and tested
- Zero-day response playbook written, distributed, and reviewed
- Severity thresholds and SLAs defined and agreed upon
- Pre-approved decision authority for Severity 1 actions (shutdown, isolation, traffic blocking)
- Incident communication channels designated and tested
- Executive and stakeholder update templates drafted

5. Rapid Mitigation Controls

- Ability to block specific payload patterns at the perimeter within minutes
- Ability to isolate services or disable external access without full incident response mobilization
- Pipeline policy enforcement capable of blocking builds referencing vulnerable components

6. Patch and Remediation Readiness

- Automated dependency updates enabled in CI/CD pipelines
- Vulnerability checks enforced before deployment

- Ability to rebuild images and redeploy across environments rapidly and systematically
- Rollback capability confirmed and tested
- Centralized remediation tracking available to security leadership in real time

7. Supply Chain Security

- Trusted repositories and registries in use; dependency versions pinned
- Third-party libraries audited on a regular cadence
- Critical vendor dependencies tracked; security SLAs documented

8. Testing and Simulation

- Zero-day fire drills conducted at least quarterly
- Simulations include AI-speed attacker scenarios (hours, not days, to exploitation)
- Scenarios cover transitive dependency vulnerabilities, not just direct
- Structured debriefs completed and action items tracked after every drill

9. Metrics and KPIs

- Time to identify all affected systems: target < 15 minutes
- Time to prioritize exploitable exposure: target < 1 hour
- Time to deploy compensating controls: target < 1 hour
- Time to full mitigation for Severity 1: target 4–8 hours
- MTTD and MTTR tracked and reviewed after every incident and drill

10. Post-Incident Learning

- Postmortem conducted within 72 hours of incident close
- What failed, what slowed response, and what improved outcomes documented
- Playbooks, thresholds, and controls updated before next quarter

The Readiness Question

Ask your team these five questions. Every “no” is a gap that deserves immediate investment.

1. Can we identify all affected systems in under 15 minutes?
2. Can we determine which are actively exploitable in under 1 hour?
3. Can we reduce exposure immediately without waiting for a patch?
4. Can we coordinate responses across every affected team without confusion?
5. Are our SLA targets calibrated for an adversary operating at machine speed?

Diagram from the [Zero Day Clock](#), by Sergej Epp, demonstrating the collapsing time to exploitation, which is now down to hours.



The AI-Native Software Delivery Platform™

Follow us on

 /harnessio

 /harnessinc

Contact us on

www.harness.io

