cytix

SECURITY LEADER GUIDE

Continuous Offensive Security Testing

A practical guide for AppSec leaders

Point-in-time penetration testing was designed for a world where software changed slowly. That world no longer exists. This guide covers what Continuous Offensive Security Testing (COST) is, why it matters, and how to implement it without breaking what already works.

20 min read Updated April 2026 Cytix Security Research

01 — OVERVIEW

What is Continuous Offensive Security Testing?

Continuous Offensive Security Testing (COST) is a security methodology that integrates adversarial testing — penetration testing, threat modelling, and exploit simulation — directly into the software development lifecycle, so that security assessment happens whenever code changes, not on an arbitrary schedule.

The word "offensive" is deliberate. COST doesn't just look for known vulnerability patterns or scan for misconfigurations. It actively attempts to exploit weaknesses, the way a real attacker would. The word "continuous" is equally deliberate — not because every test runs 24/7, but because no meaningful code change escapes scrutiny before it reaches production.

> **THE SIMPLEST WAY TO THINK ABOUT IT**
>
> *Traditional pentesting asks: "Is our application secure as of this quarter?" COST asks: "Is this specific change introducing risk before it goes live?" The difference in those two questions determines the entire shape of your security programme.*

COST isn't a product. It's a discipline — one that can be built with a combination of tooling, process, and expertise. What makes it timely is that the combination of AI-assisted development, accelerating delivery cadences, and rising compliance expectations has made the old model genuinely inadequate for most teams.

How COST relates to existing practices

COST doesn't replace the security work you're already doing — it extends it. Static analysis (SAST), dependency scanning (SCA), and dynamic analysis (DAST) all remain useful. What COST adds is the offensive layer: active exploitation attempts, business logic testing, and human judgment that automated scanners cannot replicate.

🔍

SAST / SCA

Finds known patterns and vulnerable dependencies. Fast and automated. Doesn't test whether something is actually exploitable.

🌐

DAST

Tests running applications against known vulnerability classes. Better than SAST for runtime issues, but schedule-driven and surface-level.

📋

Periodic pentesting

Deep, manual testing by skilled testers. Excellent coverage — but only of the application as it stood when the test ran.

⚡

COST

Offensive testing triggered by code changes. Combines the depth of manual pentesting with the frequency of automated tooling.

02 — THE PROBLEM

Why periodic pentesting falls short

For a long time, the annual or quarterly penetration test was a reasonable approach to application security assurance. Delivery cycles were measured in months, attack surfaces were relatively stable, and the gap between tests was an acceptable risk.

All three of those conditions have changed.

The security–delivery gap

Most engineering teams are shipping code weekly, often daily. Each deployment introduces changes — new features, refactored logic, updated dependencies, modified authentication flows. Each creates potential exposure.

The gap between the last test and the current state of production is where most real-world breaches begin. Security teams reviewing a pentest report need to evaluate findings against a version of the application that no longer exists. Developers are asked to remediate issues in code they barely remember writing.

194 days
average time to identify a breach
IBM Cost of a Data Breach Report 2024

77%
of web app attacks involve stolen credentials
Verizon DBIR 2024

AI-assisted development changes the equation

The emergence of AI-assisted development has accelerated this challenge significantly. When developers can generate and ship functional code in minutes, the volume of changes entering production increases dramatically. The issue isn't that AI-generated code is inherently less secure; it's that the sheer volume outpaces any security review process designed for slower delivery cadences.

Security teams that were already stretched are now facing an exponential increase in surface area with no corresponding increase in resource. Either the volume of code slows down (it won't), or the security model evolves to meet it.

If any of these feel familiar, you're already experiencing the problem COST is designed to address:

- → Your last pentest covered a version of the app that is months out of date
- → Developers are shipping features that no one in security has reviewed at the code level
- → AI-assisted coding is accelerating your delivery cadence beyond what your review process can absorb
- → Your DAST tool runs weekly but deployments happen daily
- → Compliance requirements are asking for evidence of continuous testing, not just point-in-time reports

03 — CORE PRINCIPLES

The four principles of COST

COST isn't defined by a specific tool or technology stack — it's defined by a set of principles that should shape how you think about security testing. Any mature implementation will reflect all four.

1. Testing is triggered by change, not by time

The fundamental shift in COST is moving from a schedule-driven testing model to a change-driven one. Security assessment isn't triggered by the calendar — it's triggered by code changes that actually matter.

This doesn't mean every single commit triggers a full penetration test. It means that **every change with security implications is identified, assessed, and tested before it reaches production**. The first challenge — and where most teams struggle — is reliably identifying which changes actually carry security risk.

> **PRACTICAL NOTE**
> *Not all code changes are security-relevant. Authentication changes, permission logic, API endpoint modifications, payment flows, data handling — these carry real risk. CSS tweaks, copy changes, internal tooling updates — usually don't. A well-implemented COST programme distinguishes between them automatically, so testing effort concentrates where it's actually needed.*

2. Every risky change gets a threat model

Knowing that a change is risky isn't enough — you need to know *how* it's risky before you can test it effectively. Threat modelling is the bridge between identifying a change and deploying testing against it.

In a COST programme, threat modelling happens continuously, at the change level. Each significant change is analysed: what attack vectors does this introduce? What trust assumptions does it make? What business logic does it affect?

The output doesn't need to be a formal document — it needs to be a structured set of hypotheses that informs the testing plan. The goal is that when a tester — human or AI — sits down to test a change, they're not starting from scratch.

3. Human expertise and AI capability work together

AI has fundamentally changed what's possible in offensive security. Automated agents can now generate test cases, identify attack vectors, and attempt exploitation at a speed no human team can match. For many categories of vulnerability, AI-driven testing is not just faster — it's more thorough.

But there are things AI cannot do. It cannot understand the business significance of a vulnerability in context. It cannot exercise the lateral thinking that finds novel attack chains. And — critically for regulated industries — it cannot carry the professional accountability that comes with a human expert reviewing and signing off on findings.

🤖

Where AI excels

Speed and coverage. Systematic testing of known vulnerability classes, API fuzzing, authentication bypass attempts, dependency chain analysis — at scale, across every change.

🧠

Where humans add irreplaceable value

Business logic flaws. Novel attack chains. Contextual risk assessment. Professional accountability. Governance and regulatory sign-off that pure-AI tools cannot provide.

The right model isn't humans or AI — it's **AI doing the work, with human expertise overseeing and stepping in where judgment is required**. This keeps the speed of continuous testing while retaining the quality and accountability that enterprise security demands.

> "
>
> *Big organisations aren't looking for a piece of software to take over all the responsibility. If something goes wrong, the auditor isn't going to blame the tool. They're going to ask who was responsible, and whether there's a documented record of the decisions that were made.*

4. Every test produces a compliance-ready record

Security testing that doesn't produce evidence is security theatre. For regulated industries — financial services, healthcare, insurance, legal tech — the ability to demonstrate what was tested, when, and by whom is as important as the testing itself.

A mature COST programme produces a continuous, structured record: which changes were tested, what the threat model identified, what the test plan covered, what was found, and what was remediated. This record should be machine-readable for GRC tooling and human-readable for auditors and regulators.

Rather than producing a point-in-time report that becomes stale the moment it's published, a COST programme produces continuous assurance — the ability to tell a regulator or customer "this application was tested before it went live, and here's the evidence."

---

04 — IMPLEMENTATION

How to implement COST in your organisation

Implementation doesn't have to be an all-or-nothing transformation. The teams that succeed with COST tend to move through recognisable phases — each one delivering value before the next begins.

1 Establish your baseline

Before you can move to continuous testing, you need a clear picture of where you are. Run a thorough assessment of your current application portfolio — what's in scope, what's changed recently, and where the existing security controls sit. The goal is to identify your highest-risk attack surfaces so that continuous testing effort can be prioritised intelligently from day one.

2 Integrate with your development workflow

The defining characteristic of COST is SDLC integration — security testing triggered by the same events that drive development. This means connecting your security testing capability to your version control system and issue tracking (GitHub, GitLab, Jira, Azure DevOps). When a pull request is raised, the system automatically assesses whether it carries security risk, and a testing workflow begins scoped to that change.

3 Build the feedback loop

Integration alone isn't sufficient — findings need to reach developers quickly and in a format they can act on. The goal is security feedback at the speed of development: a developer who pushes a change should receive actionable security information within minutes, not days. A vulnerability finding is infinitely more useful when the developer still has the relevant code in their head.

4 Govern and assure

As COST matures, the focus shifts from implementation to assurance. This phase is about ensuring the programme remains effective as the application evolves — reviewing test coverage, calibrating risk scoring, and producing the audit evidence that compliance and leadership require.

What you need to make it work

Component	What it does	Build or buy?
Change risk detection	Identifies which code changes carry security risk	Typically bought — requires significant ML capability
Automated threat modelling	Generates attack hypotheses per change	Typically bought — specialist domain knowledge required
AI pentesting agent	Executes test cases and attempts exploitation	Typically bought — rapidly evolving capability
Human expertise	Reviews findings, validates novel attacks, signs off on compliance	Internal team or external CREST-accredited provider
SDLC integration	Connects the testing workflow to your dev toolchain	Configuration task — most platforms provide this
Audit trail / reporting	Produces structured evidence of testing activity	Should be automatic in any mature COST platform

05 — MEASUREMENT

How to measure COST effectiveness

A COST programme that can't be measured can't be improved — and can't be justified to leadership. The metrics that matter fall into two categories: coverage metrics that tell you what's being tested, and outcome metrics that tell you whether the programme is delivering security value.

Coverage metrics

- → **Change coverage rate** — what percentage of security-relevant changes are being tested before they reach production
- → **Time to test** — how long between a change being identified as risky and a test being deployed against it
- → **Test plan completion** — what percentage of threat model hypotheses are covered by actual test cases

Outcome metrics

- → **Vulnerability density** — findings per 1,000 lines of changed code, tracked over time to identify trends
- → **Mean time to remediation (MTTR)** — how long between a finding being raised and the fix being validated
- → **Escape rate** — how many vulnerabilities reach production without being caught by the COST programme
- → **Risk-weighted backlog** — the composition of outstanding findings by severity

FOR SECURITY LEADERS PRESENTING TO THE BOARD

The most compelling metric for non-technical stakeholders isn't a number — it's a statement: "Every feature release this quarter was tested for security vulnerabilities before it went live." That's the outcome of a functioning COST programme, and it resonates with boards, auditors, and enterprise customers alike.

06 — ENTERPRISE & COMPLIANCE

COST in regulated and enterprise environments

For security leaders in regulated industries — financial services, insurance, healthcare, legal tech — COST isn't just a security improvement. It's becoming a compliance expectation.

Frameworks like DORA in financial services, and increasingly stringent interpretations of ISO 27001 and SOC 2, are moving toward continuous assurance requirements rather than point-in-time audit snapshots. The question from regulators is no longer just "did you test?" — it's "when did you test, what did you test, and how do you know you've tested the things that matter?"

The accreditation question

One of the most important decisions in designing a COST programme for a regulated environment is determining what level of human oversight and accreditation the programme requires.

Fully automated security testing tools can identify vulnerabilities and produce reports. But for many regulated buyers, those reports lack the professional accountability that comes with human expert involvement. **CREST accreditation** — the industry standard for penetration testing quality — requires human testers in the loop. For banks, financial services firms, and government-adjacent organisations, this isn't optional: regulators and auditors expect CREST-accredited reports, not outputs from automated systems alone.

The practical implication: a COST programme for a regulated environment should include qualified humans reviewing AI-generated findings, validating the threat model, and signing off on the security status of high-risk changes.

What auditors actually need

- → A timestamped record of what was tested and when
- → Evidence that significant changes were tested before production deployment
- → Named, qualified individuals who reviewed and approved security decisions
- → A clear process for how findings were triaged and remediated
- → Reports in a format that can be shared with regulators and enterprise customers

A well-implemented COST programme produces all of this automatically, as a byproduct of normal operation.

07 — COMMON QUESTIONS

Questions security leaders ask about COST

"We have a small security team. Is COST realistic for us?"

Yes — and in some ways, COST is more valuable for smaller teams than for large ones. When you have limited security headcount, you need every hour of that expertise to be spent where it matters most. COST automates the triage and prioritisation work that currently consumes security analysts, and deploys testing automatically against the changes that carry real risk.

"We already do quarterly pentesting. Why isn't that enough?"

It depends what "enough" means. If your goal is compliance with requirements written before continuous delivery was common, quarterly testing might technically satisfy them. If your goal is to genuinely know whether the application in production is secure right now, quarterly testing cannot achieve that. The application has changed since the last test ran.

"Won't continuous testing create alert fatigue for our developers?"

Only if it's implemented poorly. A well-designed COST programme is highly targeted — it tests security-relevant changes, not every commit. When a finding is surfaced, it's specific to the change that introduced it, with the relevant context attached. In practice, the most common developer feedback is that this model is *less* disruptive than the status quo, not more.

"How does COST interact with our existing bug bounty programme?"

Complementarily. Bug bounty is reactive — it relies on external researchers finding issues in production. COST is proactive — it finds issues before they reach production. A mature security programme uses both: COST to catch issues in the development pipeline, and bug bounty as a further layer of coverage for issues that get through.

"What's the difference between COST and DevSecOps?"

DevSecOps is a philosophy — the idea that security should be integrated into the development process rather than bolted on at the end. COST is a specific practice within that philosophy: the continuous, change-driven application of offensive security testing. A full DevSecOps programme, taken seriously, leads naturally to COST — because everything else is defensive, and defence alone isn't sufficient.

---

08 — GETTING STARTED

Where to begin

The most common mistake in adopting COST is trying to boil the ocean. Start with the highest-risk application, or the most actively developed one, and prove the model there first.

1 **Identify your highest-risk application**

Which application carries the most sensitive data, has the most external exposure, or is changing fastest? Start there. COST delivers the most visible value where the combination of change velocity and security risk is highest.

2 **Map your current testing coverage**

Before adding anything new, understand what's already in place. What does your existing pentest cover? How frequently does your DAST run? Where are the gaps between test frequency and deployment frequency? This map becomes the case for change.

3 **Choose your model**

COST can be delivered fully in-house, fully outsourced through a managed continuous testing provider, or as a hybrid of both. For most security leaders, the hybrid model — where an external specialist platform handles the automation and AI layer, with your team governing and escalating — is the right starting point.

4 **Run a pilot, measure it, expand**

Pilot COST on a single application for 90 days. Measure coverage rate, MTTR, and vulnerability density. Use those numbers to build the business case for expanding the programme. The data from a real pilot is far more persuasive than any analysis done in a spreadsheet.

See COST in practice with Cytix

Cytix is built specifically for continuous offensive security testing — integrating with your SDLC, threat modelling every risky change, and deploying both AI and CREST-accredited human testing before code reaches production.

cytix.io/demo