WHITEPAPER · SECURITY

Continuous Security Testing for Small Teams

How to make security testing a low-noise, weekly habit without a security team or an annual pentest budget.

CodeReviewer

A product of Techware Lab · codereviewer.cloud

CONTENTS

Inside this paper

01 Why small teams skip security testing

02 Three layers: passive, active, modelled

03 Only scan what you own

04 An illustrative cost-of-timing model

05 Threat modelling without a workshop

06 A 30-day rollout plan

07 How CodeReviewer helps

About this paper

A practical guide for teams without a dedicated security function. It argues that security testing works best as a continuous, mostly automatic habit attached to work you already do, not as an annual event. Figures in the model are illustrative. Replace them with your own.

SECTION 01

Why small teams skip security testing

Ask a small product team when they last tested their application’s security and the honest answer is often "before the big customer asked for a pentest report". Security testing is treated as an event: expensive, scheduled, run by outsiders, and finished with a PDF that is out of date by the next release.

The reasons are understandable. Security tools have a reputation for noise. Active scanners can damage data or trip alarms if pointed at the wrong place. Nobody on the team owns the results, so findings pile up in a spreadsheet. And the work competes with features that customers are actually asking for.

Meanwhile the application changes every week. A new endpoint forgets an authorisation check; a dependency update pulls in a library with a known vulnerability; a header is dropped during a framework upgrade. None of these waits for the annual test.

> **The core claim**
>
> Security findings are cheapest when they are found close to the change that caused them. Continuous, low-noise testing beats occasional, exhaustive testing for most small teams.

SECTION 02

Three layers: passive, active, modelled

It helps to think of application security testing as three layers with different costs and risks. A small team should run the first continuously, the second deliberately, and the third whenever the architecture shifts.

Layer	What it does	Risk to your app	How often
Passive checks	Observes normal traffic: transport security (HSTS), content security policy, clickjacking protection, cookie flags, libraries with known CVEs	None. No attack traffic is sent	Continuously, on a schedule
Active testing (DAST)	Sends crafted requests to probe for injection, misconfiguration and other OWASP Top 10 classes, e.g. with OWASP ZAP	Real. Can create data or trigger alerts	Deliberately, against staging you own
Threat modelling	Reasons about what an attacker would try, from code and architecture, e.g. using STRIDE and MITRE ATT&CK	None	When architecture or data flows change

Passive checks catch a surprising amount of real-world exposure: the missing header, the session cookie without *Secure* or *SameSite*, the outdated front-end library. And they do it at almost zero risk. They are the right default.

Active testing finds deeper issues but must be controlled. Threat modelling finds the issues no scanner can: business-logic flaws, missing authorisation between tenants, trust boundaries nobody drew.

SECTION 03

Only scan what you own

Active scanning is, mechanically, an attack. Pointed at a system you do not own it may be illegal; pointed at production without warning it may corrupt data or page your on-call engineer at 3 a.m. Responsible tooling therefore needs a hard rule, not a checkbox.

- **Prove ownership first.** Verify the domain with a DNS TXT record or a file served from a well-known path before any active scan is allowed.
- **Re-verify automatically.** Ownership can lapse when a domain or environment is handed over; checks should repeat and scans should stop when proof disappears.
- **Prefer staging.** Run active scans against a production-like environment with disposable data.
- **Tell people.** Let whoever watches your alerts know when scans run, so findings are not mistaken for an incident.

> **Why this matters for trust**
>
> A tool that will happily attack any URL you type is a liability to you and to everyone else on the internet. Verification is what makes active testing safe to automate.

SECTION 04

An illustrative cost-of-timing model

Illustrative: replace with your own numbers

The hours below are assumptions chosen to show the shape of the calculation. They are not measurements from any customer or industry survey.

Expected annual remediation effort = V × sum over stages of (p_s × c_s)

where **V** is the number of security-relevant defects introduced per year, $\mathbf{p_s}$ the share found at stage *s*, and $\mathbf{c_s}$ the human hours to handle one defect found at that stage (diagnosis, fix, review, deploy, and any customer communication).

Stage found	Hours per defect (c)	Occasional testing (p)	Continuous testing (p)
In review, before merge	1	10%	45%
Scheduled scan of staging	4	20%	40%
Annual external test	8	40%	10%
In production, by someone else	40	30%	5%

With V = 30 defects a year, occasional testing gives 30 × (0.1×1 + 0.2×4 + 0.4×8 + 0.3×40) = **483 hours**. Continuous testing gives 30 × (0.45×1 + 0.4×4 + 0.1×8 + 0.05×40) = **146 hours**.

The saving comes almost entirely from the last row. A defect found in production carries costs a scan never does: incident response, customer communication, and the lost trust that no hour count captures. Shifting even a modest share of discoveries earlier dominates the result.

SECTION 05

Threat modelling without a workshop

Traditional threat modelling is a whiteboard session with a security specialist. Small teams rarely have one. The underlying method is simple enough to apply routinely.

STRIDE category	Question to ask of every feature
Spoofing	Can someone pretend to be another user or service?
Tampering	Can data be changed in transit, in storage, or through a parameter we trust?
Repudiation	Could a user deny an action because we keep no reliable record?
Information disclosure	Can one tenant, role or anonymous user see data meant for another?
Denial of service	Can one request or user exhaust something everyone shares?
Elevation of privilege	Can a member reach an admin action, or a user reach another tenant?

Mapping each credible threat to a MITRE ATT&CK technique adds a shared vocabulary and connects the model to how real attackers operate. The most useful models cross-reference live evidence: a threat that a passive scan has already observed (say, a session cookie without *SameSite*) deserves attention before a purely theoretical one.

SECTION 06

A 30-day rollout plan

- **Week 1: baseline.** Run passive checks against production and staging. Record the grade; fix the headers and cookie flags, which are usually one-line changes.
- **Week 2: ownership.** Verify the domains you control. Agree who is told when active scans run.
- **Week 3: active, carefully.** Run the first active scan against staging. Triage findings into real, accepted-risk and false positive; turn each real one into a tracked issue with an owner.
- **Week 4: model and schedule.** Produce a first threat model of the main application. Put passive checks on a daily schedule and active scans on a weekly schedule.
- **Ongoing.** Re-model when you add a new data store, integration or tenant boundary. Review the grade monthly.

> **Rule of thumb**
>
> A finding without an owner and a due date is not a finding; it is a future incident with extra steps.

SECTION 07

How CodeReviewer helps

CodeReviewer attaches security testing to work your team already does.

- Passive security checks by default (transport, CSP, clickjacking, cookie flags and known-CVE libraries), with an A to F grade and no attack traffic.
- Active DAST with OWASP ZAP, only on domains you have verified by DNS or a well-known file, with automatic re-verification.
- AI threat models that read your code and your scan results and map threats to STRIDE and MITRE ATT&CK.
- One consolidated report with severities, remediation guidance and an executive summary.
- Findings turned into issues, located in your code and fixed with one click.
- Scans ride your existing test suites and never consume AI runs to re-run. Load testing is on the roadmap.

Start a 15-day free trial at **codereviewer.cloud**.

See it on your own codebase.

Start a 15-day free trial. No card needed.

codereviewer.cloud