

TECH & AI CASE STUDY - E-COMMERCE - 2026

DevSecOps Automation That Cut Critical Vulnerabilities 83% for an E-commerce Platform

Based on a delivery engagement with Trailhouse Outfitters

-83%CRITICAL VULNERABILITIES PER
QUARTER**32 days to 5**

TIME TO REMEDIATE CRITICALS

zero

LIVE SECRETS IN GIT

<10 min

PIPELINE SECURITY BUDGET

A fillable practice edition: the full story, 8 actions,
6 reflections and a 90-day plan - work it with your team.

Free - no email requiredemersontechnology.com

MAKE THIS PRACTICE BOOK YOURS

Work it, do not just read it

This practice book turns the Trailhouse Outfitters case study into a working session for your team. Read the full story first - what happened, what was built and what it changed. Then tick the 8 actions you can commit to, write short answers to the 6 reflections, and finish with a 90-day plan. Every field in this PDF is fillable: open it in Adobe Acrobat Reader, Chrome, Edge or macOS Preview, type directly into the boxes and save.

YOUR NAME

ROLE

COMPANY

TEAM / DEPARTMENT

DATE

EMAIL (OPTIONAL)

MY GOAL FOR THIS PRACTICE SESSION

- 01

The story - What happened at Trailhouse Outfitters - before, during and after the project.
- 02

The results - The numbers that changed, plus industry benchmarks from public research.
- 03

The 8-action checklist - Tick boxes as your organization completes each step. Owner and date fields included.
- 04

The 6 reflections - Fillable questions that pressure-test your own roadmap.
- 05

My 90-day plan - Three fillable phase cards that turn this case study into your roadmap.

RUN IT AS A 90-MINUTE TEAM SESSION

25 min READ Read the story together. Ask everyone to note the one number that surprised them most.	40 min PRACTICE Work the 8-action checklist and the 6 reflections. Assign an owner and a date to every action you commit to.	25 min COMMIT Fill in the 90-day plan. One goal per phase, one number that tells you it worked. Book a review date before you leave the room.
--	--	---

THE STORY

What happened, and why it worked

01 The Company

Trailhouse Outfitters is an outdoor retailer whose e-commerce platform sells gear and apparel direct to consumers, backed by a 40-engineer technology organization. The platform handles the full retail lifecycle — product catalog, search, cart and checkout, order management, loyalty and a marketplace integration that puts Trailhouse products in front of partner audiences. The engineering team practices modern delivery: deployments go out daily, features move from idea to production in days, and the company's commerce operation depends on that velocity to run seasonal campaigns, drop new product lines and respond to demand shifts quickly.

Trailhouse's customers are also security-relevant by default. The platform stores accounts, payment tokens and order histories, and it integrates with marketplace partners who audit their suppliers' security practices as a condition of doing business. In retail, security failures are not abstract risks — they surface as checkout downtime during the highest-revenue weeks of the year, as card-testing fraud, or as a partner review that stalls a revenue channel. Verizon's Data Breach Investigations Report consistently finds web applications among the most common breach vectors, with errors and credential abuse leading the patterns, which describes Trailhouse's exposure almost exactly: a web application, a large dependency surface and a fast-moving pipeline.

02 Where Things Stood

Trailhouse shipped daily, but security testing lived outside the shipping process, and the gap showed up with ritual regularity. Every quarterly penetration test found the same vulnerability classes: cross-site scripting in newly shipped pages, outdated JavaScript dependencies with known CVEs, and API keys that had been committed to git history and quietly copied between repositories. Each pen test generated a PDF report, the report generated a triage meeting, the triage meeting generated a backlog — and the backlog lost to feature work every sprint, because nothing forced the issues to be fixed where and when the code was written.

The cost of that gap arrived during peak season, when an exploited dependency forced an emergency weekend patch. The incident cost an estimated \$310,000 in delayed features and overtime, but the number that stung most came out of the incident review: the vulnerable package had been flagged weeks earlier by a vulnerability scanner whose output nobody read. The tooling existed; the process made it ignorable. Around the same time, a marketplace partner's due-diligence review demanded evidence of a secure development lifecycle — documented controls, scan coverage, remediation SLAs — and the team could not produce it, not because the engineering was careless but because nothing was systematic. Security testing, in short, ran late, separately and silently: late in the development cycle, separate from the pipeline, and silent until something broke.

03 The Challenge

The challenge was to make security move at the speed of the pipeline without breaking the pipeline. Trailhouse's engineers were not resistant to security; they were resistant to friction, and rightly so — Gartner has predicted that through 2025, 45% of organizations worldwide will have experienced an attack on their software supply chain, three times more than in 2021, so both speed and scrutiny were necessary. The design tension was real: gate too little and the vulnerability classes that every pen test found would keep returning; gate too much and developers would route around the checks, and bypassed security is worse than none because it manufactures false confidence.

Three specific problems had to be solved as a system. First, findings arrived too late and in the wrong place — in PDF exports days after the code was written, rather than in the pull request while the author still had the context loaded. Second, the noise problem: dependency scanners report every published CVE, and most of them are not

reachable from Trailhouse's actual code paths, so a naive gate would flood engineers with work that does not matter. Third, the secrets problem was historical as well as procedural — years of committed API keys sat in git history, so prevention alone would not clean up what prevention had failed to stop. IBM Security's 2024 breach report finds organizations with extensive security AI and automation reportedly shorten breach lifecycles by roughly 100 days on average, and the team took that as the design brief: automate the detection, shorten the exposure, and put the fix in front of the person who can apply it.

04 Discovery and Diagnosis

Emerson Technology ran a three-week diagnostic that combined pipeline observation, repository archaeology and interviews across the engineering organization. The pipeline itself was mapped end to end: build times, test stages, and where a security stage could be inserted without breaching a ten-minute total budget, which the team set as a hard constraint after observing that any slower pipeline would erode the daily deployment rhythm the business depends on. Repository analysis quantified the debt concretely — 41 critical vulnerabilities per quarter on average across the last four pen-test cycles, live credentials sitting in git history across multiple repositories, and a dependency inventory where a meaningful share of flagged CVEs were not actually reachable from executed code paths.

The interviews diagnosed the human half of the system. Developers reported that they wanted to fix findings quickly but learned about them at the wrong time — often weeks after merging, when the code was no longer in their heads. Security review was performed by one senior engineer whose review queue had become a bottleneck, and the quarterly pen-test findings competed for sprint capacity against revenue features and lost, predictably, every time. The remediation data told the same story: mean time to remediate a critical finding was 32 days, not because anyone was negligent but because the finding surfaced in a backlog rather than in a pull request. The diagnosis concluded that Trailhouse did not need more scanning — it needed the existing detection to arrive earlier, speak louder for criticals and stay quiet for noise, which became the design principle for everything that followed.

05 The Solution

We embedded security directly into the existing CI/CD pipeline under a ten-minute budget, so that every merge is checked and no check is slow enough to be worth bypassing.

The gated pipeline

Every pull request now runs static analysis for code vulnerabilities, software-composition analysis with reachability checks, secrets scanning and container and infrastructure-as-code scanning in parallel lanes. The gate is deliberately narrow: only critical findings fail the merge, and everything else is triaged by a defined SLA, because blocking merges on every warning teaches developers to ignore the pipeline entirely. Fast scans run first so quick failures surface early, and heavier jobs ride parallel lanes without extending the critical path. The result is a gate that developers respect precisely because it rarely cries wolf — when the pipeline blocks a merge, the finding is real.

Dependency and secrets hygiene

Dependency drift, the source of the \$310,000 weekend, is now handled by routine rather than heroics: automated dependency updates open reviewed pull requests daily with tests attached, so upgrades arrive as small, boring, frequent changes instead of emergency patches during peak traffic season. Reachability analysis prioritizes vulnerable libraries that Trailhouse's code paths actually execute, which cuts remediation noise dramatically and focuses engineers on dependencies that are genuinely exploitable. The historical secrets problem was closed in two moves: pre-commit hooks and server-side push protection now stop credentials before they reach git at all,

and a one-time, carefully sequenced rotation eliminated everything already sitting in history — because preventing the next commit is far cheaper than rewriting the past, but the past still had to be dealt with once.

Triage that developers actually use

Findings surface as pull-request comments with concrete suggested fixes attached, not as exports to a scanner console nobody opens. Secure-coding feedback is attached to real merge requests, using the code the engineer just wrote as the teaching moment instead of a standalone annual training course. A vulnerability dashboard lives in the team's existing observability stack, charting critical findings, remediation time and reopen rates where engineers see them daily. Critical remediation carries a five-day SLA backed by leadership review, which is what gives the SLA teeth without a security team chasing tickets manually — the escalation is visible, not personal.

06 The First 90 Days

The first month was spent on instrumentation and the pipeline skeleton: scans were integrated into the pull-request flow in parallel lanes, the ten-minute budget was enforced from day one, and the team ran the pipeline against the existing codebase in report-only mode to calibrate thresholds before any gate went live. That calibration week mattered enormously — it revealed that a naive gate would have blocked roughly a third of open pull requests, most on unreachable CVEs, and it produced the reachability configuration that made the eventual gate credible. Pre-commit hooks and push protection were rolled out in week two across every repository, and the secrets rotation was sequenced for a low-traffic week to avoid entangling credential cutover with a campaign. Weeks five through eight brought the gate live for critical findings only, the daily automated dependency-update flow into service, and the first monthly burn-down review with leadership. The early numbers were noisy but directional: critical findings introduced per week dropped sharply once new code was scanned at pull-request time, and the first dependency-update batches removed three genuinely reachable critical CVEs without a single emergency. Weeks nine through twelve completed the rollout — the vulnerability dashboard moved into the observability stack, the five-day critical SLA took effect with its leadership-review escalation, and the secure-coding feedback loop was attached to real merges. By day 90, the mean time to remediate criticals had already fallen from 32 days to single digits, and the team's weekly pipeline performance report showed the full scan suite finishing inside budget on every merge.

07 The Results

Critical vulnerabilities per quarter fell 83% — from 41 to 7 — and the seven that remain are typically exotic findings that reachability analysis still flags as low exposure. Mean time to remediate criticals dropped from 32 days to 5, comfortably inside the SLA, and the remediation now happens in the pull request where the issue was introduced, while the author still has full context. Git history is free of live secrets after the one-time rotation, with push protection preventing regressions, and the pipeline — including every security scan — still completes inside its ten-minute budget. The marketplace partner's secure-development review passed with the pipeline artifacts themselves as evidence, and the following peak season shipped without an emergency security patch for the first time in years.

METRIC	BEFORE	AFTER
Critical vulnerabilities per quarter	41	7
Time to remediate criticals	32 days	5 days
Live secrets in git	Present	Zero
Pipeline security budget	-	Under 10 min

The cultural shift is the durable result. Security stopped being a quarterly event and became a property of everyday engineering: developers fix findings in the flow of their own work, dependency updates arrive as ordinary reviewed pull requests, and the vulnerability dashboard is checked the same way the team checks error rates. The security review that once bottlenecked on one senior engineer is now distributed across the pipeline, freeing that engineer to handle design-level threats instead of ticket triage. And the \$310,000 emergency — the incident that justified the whole program — has a clear successor story: the same class of vulnerability appeared again in a new dependency months later, and the fix shipped as a routine automated pull request the next morning, noticed by no one, costing nothing.

08 What Your Business Should Do

The Trailhouse pattern works because it treats security as pipeline design rather than as policing. Gate merges on critical findings only, and triage everything else by a defined SLA — blocking merges on every warning teaches developers to ignore the pipeline entirely. Keep the pipeline under ten minutes by running fast scans first and pushing heavier jobs to parallel lanes or nightly runs, because security that slows merges gets bypassed, and bypassed security is worse than none. Check dependency reachability rather than raw CVE counts, prioritizing libraries your code paths actually execute; reachability analysis cuts remediation noise dramatically and focuses engineers on what is genuinely exploitable. Stop secrets before they reach git with pre-commit hooks and server-side push protection, then rotate everything already in history once — the rotation is a one-time project, and prevention makes it the last one.

Build the operating rhythm that keeps the system honest. Automate dependency updates daily, opening reviewed pull requests with tests attached, because small frequent upgrades beat emergency weekend patches during peak season every single time. Deliver findings as pull-request comments with concrete suggested fixes, since findings that appear where developers already work get fixed, while exports to another tool just get archived. Attach secure-coding feedback to real merge requests rather than standalone annual courses — feedback tied to the code you just wrote changes habits the fastest. Finally, track the vulnerability burn-down publicly: chart critical findings, remediation time and reopen rates on a dashboard engineers see daily, because numbers everyone can see keep SLAs honest and future budgets funded. As you plan your own program, sit with the questions this engagement surfaced — what your last emergency security patch really cost, how many days a critical waits in your backlog today, whether you could prove a secure development lifecycle to a partner tomorrow, where findings surface for your developers, and how many live credentials sit in your git history right now. Work through these eight steps with your team using the action checklist in this practice book, and revisit the reflection questions each quarter to keep the pipeline — and the confidence it buys — honest.

REAL-WORLD PROOF & BENCHMARKS

Gartner has predicted that through 2025, 45% of organizations worldwide will have experienced an attack on their software supply chain — three times more than in 2021. - Gartner, 2021

The Verizon Data Breach Investigations Report consistently finds web applications among the most common breach vectors, with errors and credential abuse leading the patterns. - Verizon DBIR, 2024

IBM's 2024 breach report finds organizations with extensive security AI and automation reportedly shorten breach lifecycles by roughly 100 days on average. - IBM Security, 2024

THE PRACTICE PAGES

The 8-action checklist

Work through the list with your team. Tick each box as your organization completes the step, and fill in an owner and a target date so every action has a name and a deadline.

☐

1. Gate merges on critical findings only

Fail the build on critical severity and triage everything else by a defined SLA. Blocking merges on every warning teaches developers to ignore the pipeline entirely.

OWNER

TARGET DATE

STATUS

--	--	--

☐

2. Keep the pipeline under ten minutes

Run fast scans first and push heavier jobs to parallel lanes or nightly runs. Security that slows merges gets bypassed, and bypassed security is worse than none.

OWNER

TARGET DATE

STATUS

--	--	--

☐

3. Check dependency reachability, not just CVEs

Prioritize vulnerable libraries that your code paths actually execute, not every CVE bulletin. Reachability analysis cuts remediation noise dramatically and focuses engineers on dependencies that are genuinely exploitable.

OWNER

TARGET DATE

STATUS

--	--	--

☐

4. Stop secrets before they reach git

Install pre-commit hooks and server-side push protection, then rotate everything already sitting in history once. Preventing the commit is far cheaper than rewriting git history later.

OWNER

TARGET DATE

STATUS

--	--	--

☐

5. Automate dependency updates daily

Open reviewed upgrade pull requests every day with tests attached automatically. Small, frequent updates beat emergency weekend patches during peak traffic season every single time.

OWNER

TARGET DATE

STATUS

--	--	--



6. Deliver findings as PR comments

Surface scanner output as pull-request comments with concrete suggested fixes attached. Findings that appear where developers already work get fixed; exports to another tool just get archived.

OWNER

TARGET DATE

STATUS

--	--	--



7. Attach security feedback to real merges

Use actual merge requests as secure-coding teaching moments instead of standalone annual training courses. Feedback tied to the code you just wrote changes habits the fastest.

OWNER

TARGET DATE

STATUS

--	--	--



8. Track vulnerability burn-down publicly

Chart critical findings, remediation time and reopen rates on a dashboard that engineers see daily. Numbers everyone can see keep SLAs honest and future budgets funded.

OWNER

TARGET DATE

STATUS

--	--	--

AFTER THE CHECKLIST: OUR TOP 3 PRIORITIES

From the eight actions above, write down the three you will start first - and the single number you expect each one to move. Three priorities with owners beat eight without any.

PRIORITY 1 NUMBER IT SHOULD MOVE	PRIORITY 2 NUMBER IT SHOULD MOVE	PRIORITY 3 NUMBER IT SHOULD MOVE
--	--	--

THE PRACTICE PAGES

Reflections for your team

Use these questions in your next planning session and type short answers into the boxes. The gaps you find are usually the fastest wins.

Q1. What did your last emergency security patch cost in delayed features, overtime and peak-season revenue?

Q2. How many days does a critical vulnerability wait in your backlog before someone owns the fix?

Q3. Could you prove a secure development lifecycle to a partner today, or would the review fail?

Q4. Where do security findings surface for your developers — in the pull request, or in a scanner nobody opens?

Q5. How many live credentials are sitting in your git history right now, and who would know?

Q6. If your pipeline blocked on every warning tomorrow, would developers fix issues or start bypassing it?

THE PRACTICE PAGES

My 90-day plan

Finish the session by committing to a plan. Three phases, one goal each, one number that tells you it worked. Bring this page to your next leadership review - or to a free consultation with Emerson Technology, and we will pressure-test it with you.

Days 1-30 Focus: audit and alignment

PRIMARY GOAL

KEY ACTIONS

HOW WE MEASURE SUCCESS

OWNER

Days 31-60 Focus: build and prove

PRIMARY GOAL

KEY ACTIONS

HOW WE MEASURE SUCCESS

OWNER

Days 61-90 Focus: scale and measure

PRIMARY GOAL

KEY ACTIONS

HOW WE MEASURE SUCCESS

OWNER

Tip: save a copy per team so each group keeps its own answers. Revisit the reflections every quarter - the questions stay the same even as the numbers move.

BOOK THE REVIEW BEFORE YOU CLOSE THE BOOK

Plans without a review date drift. Before you close this practice book, agree on when you will come back to these pages, who will run that session, and when you will check the numbers again. If you want an outside pair of eyes, bring this page to a free consultation with Emerson Technology and we will pressure-test the plan with you.

REVIEW DATE	REVIEWED BY	NEXT CHECK-IN
------------------------------------	------------------------------------	--------------------------------------

Ready for a transformation like Trailhouse Outfitters's?

Emerson Technology designs and builds the data, AI, cloud and automation systems behind results like these. On a free consultation call we will map this exact playbook to your business - your data, your constraints, your numbers.

Book a free consultation

support@emersontechnology.com

or start online: emersontechnology.com/contact