CUSHMAN & WAKEFIELD

Demo Delivery Guide

Prepared for	Cushman & Wakefield — Development Engineering Director
Prepared by	GitHub / Microsoft Solutions Engineering
Demo Environment	PropTracker API (localhost) + GitHub Actions
Duration	60 minutes (45 demo + 15 Q&A)
Classification	GitHub Confidential — Sales Demo Use Only

Table of Contents

1. Executive Summary — The Business Case

This guide arms you to deliver a compelling, technically rigorous 60-minute demo to Cushman & Wakefield's Development Engineering Director. The demo uses PropTracker — a deliberately vulnerable property management API — to make abstract security concepts tangible and emotionally resonant.

■ **Core Message**	GitHub doesn't just find vulnerabilities after the fact — it prevents them from reaching production in the first place through automation baked into the developer workflow.
■ **Business Impact**	Each undetected API vulnerability costs an average of $4.45M to remediate (IBM Cost of a Data Breach 2023). GHAS pays for itself when it catches a single critical issue.
■ **Speed to Value**	From zero to fully monitored API: 15 minutes setup. GitHub Actions catches vulnerabilities every 15 minutes automatically, 24/7, without any manual security reviews.
■ **The Copilot Edge**	GitHub Copilot doesn't just detect — it fixes. A developer sees a security alert and gets an AI-generated patch right in the PR, no security expertise required.

Cushman & Wakefield Context

Cushman & Wakefield manages billions of dollars in property assets, contractor networks, and tenant data. Their engineering teams build and maintain APIs that expose:

- Contractor PII (addresses, SSNs, insurance certificates)
- Property financial data (valuations, lease agreements)
- Job assignment workflows (work orders, inspections)
- Tenant personal data (background checks, payment history)

A single API vulnerability — BOLA, SQL injection, SSRF — could expose any of this data, triggering GDPR/CCPA fines, contractual penalties, and reputational damage. **This is your anchor throughout the demo.**

2. Pre-Demo Setup Checklist

Complete all items below at least 30 minutes before the customer arrives. Nothing kills a demo faster than a failed localhost server.

✓	ITEM	VERIFY WITH	NOTES
■	Docker Desktop running	docker ps	Must show proptracker-db container
■	PostgreSQL container up	docker ps | grep proptracker	Port 5432
■	API server running	curl localhost:3001/health	Should return {status:'healthy'}
■	Frontend running	Browser: localhost:5175	Dashboard loads with 10 vuln cards
■	Login works	POST /api/auth/login	alice@propowner.com / Password123!
■	Exploit buttons fire	Click Run Exploit on BOLA card	Should show VULNERABLE result
■	Workflow YAML loads	Click View Workflow on any card	YAML with highlighted lines
■	GitHub repo accessible	github.com/sautalwar/cushman-property-api	Actions tab shows workflows
■	Browser tabs pre-opened	See list below	Alt-tab ready, no typing during demo
■	Presentation deck open	PropTracker-Demo.pptx	Slide 1 visible
■	Volume/notifications off	System settings	No Teams/Slack popups
■	Backup: localhost API call	Postman collection loaded	Fallback if UI breaks

Browser Tabs to Pre-Open

Tab 1 (Slides): PropTracker-Demo.pptx — Full screen

Tab 2 (Demo App): http://localhost:5175 — Dashboard

Tab 3 (GitHub Actions): https://github.com/sautalwar/cushman-property-api/actions

Tab 4 (Code Scanning): https://github.com/sautalwar/cushman-property-api/security/code-scanning

Tab 5 (Secret Scanning): https://github.com/sautalwar/cushman-property-api/security/secret-scanning

Tab 6 (Issues): https://github.com/sautalwar/cushman-property-api/issues

Tab 7 (Copilot PR): A PR with Copilot Autofix suggestion ready

3. Opening Hook — The $15M Breach Story (5 min)

Objective: Create emotional urgency. Make the customer feel the risk before you show the solution.

Exact Script — Word for Word

SLIDE	Slide 1: Title
SAY	"Before I show you the demo, I want to tell you a story about a property management company — similar to Cushman — that had a very bad Tuesday morning."
PAUSE	3 seconds. Let it land.
SAY	"Their API had been live for 18 months. It passed code review. It passed pen testing. And one morning, a security researcher found that any authenticated user could access any other tenant's lease agreement — just by changing a number in the URL."
SAY	"That single API vulnerability — what OWASP calls Broken Object Level Authorization, or BOLA — exposed 340,000 lease records, including personal guarantors, bank account numbers, and social security numbers."
SAY	"The remediation cost: $15 million. The regulatory fine: $4.2 million. And the worst part? That vulnerability had existed in the codebase for 14 months before anyone found it. Their automated tests ran every night and never caught it."
TRANSITION	Click to Slide 2 — PropTracker
SAY	"Today I'm going to show you that exact vulnerability — running live — in our demo app. And then I'm going to show you how GitHub catches it automatically, before it ever reaches production, and how Copilot fixes it in under 60 seconds."
SAY	"The difference between that company and Cushman & Wakefield will be what you see in the next 45 minutes."

■ **Facilitator Tip: If the director has a security background, ask: *"Have you had to respond to a BOLA disclosure before?"* — this creates instant rapport and personalizes the story.**

4. Act 1 — The Vulnerable API (10 min)

Objective: Make the vulnerability real by running it live. Turn abstract "security risk" into a concrete HTTP request that returns data it shouldn't.

SLIDE 3 **Switch to Demo App:** Navigate to http://localhost:5175 — Dashboard is visible

SAY **Introduce PropTracker:** "PropTracker is our demo property management platform — it's a React frontend backed by a Node.js API and PostgreSQL. This is a realistic stack — TypeScript, JWT auth, REST endpoints. And it has 10 intentional vulnerabilities built in, mirroring the OWASP API Security Top 10."

ACTION **Show the Dashboard:** Point to the 10 vulnerability cards. "Each card here represents one real vulnerability in the running API."

ACTION **Run BOLA exploit:** Click ■ Run Exploit on VULN-1 BOLA card. Wait for result.

SAY **Narrate the result:** "Look at this. Bob — authenticated user, valid JWT — just fetched Alice's private job record. The server returned HTTP 200 and the full job data. No error. No audit log. No alert. This is running right now."

ACTION **Show the request/response:** Point to the Request panel: Authorization: Bearer [Bob's token]. Point to Response: job owned by Alice.

SAY **Tie to Cushman:** "In your context, this isn't a job record — it's a lease agreement, a contractor's insurance certificate, a property valuation. Any authenticated API user could be accessing any other user's data right now, and you'd have no idea."

ACTION **Run SQL Injection exploit:** Click ■ Run Exploit on VULN-8 SQL Injection. Show result.

SAY **Escalate the impact:** "SQL injection is worse. This payload returns every property in the database — including the ones that user has no business seeing. In PostgreSQL at scale, this becomes a full data dump in a single request."

PAUSE **Let them react:** Give 5 seconds of silence. Let the demo do the talking.

5. Act 2 — GitHub Actions: Continuous Detection (15 min)

Objective: Show that GitHub Actions runs 24/7 as a security safety net — automatically probing for every vulnerability class without any human intervention.

SLIDE 5 **Transition slide:** "Now you've seen the vulnerability. Let me show you how GitHub catches it automatically."

BROWSER **Open GitHub Actions tab:** Navigate to github.com/sautalwar/cushman-property-api/actions

SAY **Set context:** "We have 10 GitHub Actions workflows — one for each OWASP API Security category. They run every 15 minutes. They don't need a developer to trigger them. They don't need a pen tester. They run continuously, 24/7."

ACTION **Click check-bola.yml:** Click on the most recent BOLA workflow run. Show the job steps.

SAY **Narrate what happened:** "Look at what this workflow does. Step 1: it logs in as Alice. Step 2: it logs in as Bob. Step 3: Bob tries to access Alice's job. Step 4: it checks the HTTP response code. If it gets 200 — vulnerable. If it gets 403 — protected."

ACTION **Show failed step:** Point to the red X on the Evaluate result step. "That red X means it confirmed the vulnerability. Immediately."

SAY **Show the error annotation:** "And look here — GitHub automatically created a workflow annotation. ::error:: VULN-1 CONFIRMED: BOLA vulnerability detected. This is now visible to every developer on the team."

BROWSER **Switch to demo app:** Navigate back to http://localhost:5175

ACTION **Click View Workflow on BOLA card:** Show the YAML with color-coded highlighting

SAY **Narrate the highlights:** "The red highlighted lines are where the vulnerability is confirmed. The green lines are the suggested fix. A developer reading this workflow — even without security expertise — knows exactly what the problem is and exactly how to fix it."

SAY **Expand to all 10:** "And we have this for all 10 OWASP API Security categories. Broken auth, mass assignment, rate limiting, CORS, SQL injection, SSRF. Every 15 minutes. Automatically."

PAUSE **Rhetorical question:** "How often does your team currently run security scans?" — pause for answer, then: "GitHub Actions is doing it 96 times a day."

6. Act 3 — GitHub Advanced Security (GHAS) Deep Dive (10 min)

Objective: Show the full GHAS trifecta — Code Scanning (CodeQL), Secret Scanning, and Dependency Review — and how they complement the custom Actions workflows.

■ Code Scanning (CodeQL)

→ Navigate to github.com/sautalwar/cushman-property-api/security/code-scanning

→ "CodeQL is GitHub's static analysis engine. It doesn't just look for known vulnerability patterns — it builds a semantic model of your code and reasons about data flow."

→ Show a CodeQL alert for SQL injection: "CodeQL traced the data flow from req.query.q all the way to the raw SQL string. It knows the user input reaches the database without sanitization."

→ "This catches vulnerabilities that pattern matching would miss — multi-step injection chains, indirect data flows, custom helper functions that pass tainted data."

■ Secret Scanning

→ Navigate to github.com/sautalwar/cushman-property-api/security/secret-scanning

→ "Secret scanning monitors every commit to every branch. If a developer accidentally commits a database password, an API key, a JWT secret — GitHub catches it within 30 seconds and blocks the push."

→ "We have 200+ supported secret patterns out of the box — AWS keys, Azure credentials, GitHub tokens, Stripe keys, Twilio auth tokens. And you can add custom patterns for your internal secrets."

→ "Critically: we also partner with providers. If an AWS key is detected, we notify AWS immediately and they revoke it automatically. This is the fastest possible response time."

■ Dependency Review

→ "Every dependency you install is a potential attack vector. Log4Shell happened because of a transitive dependency nobody knew they had."

→ "GitHub's dependency graph tracks every package, every version, every CVE. When a new vulnerability is disclosed, you get an automatic Dependabot alert — or an automatic PR with the fix."

→ "Dependency Review in PRs blocks merges when a PR introduces a dependency with a known CVE. Your developers can't accidentally ship a vulnerable library."

7. Act 4 — Copilot Autofix: From Alert to Fix in 60 Seconds (5 min)

Objective: Land the emotional high point. A GHAS alert is actionable, not just informational — Copilot does the remediation work for the developer.

SAY **Setup:** "Here's where this gets really exciting. You've seen that GitHub finds the vulnerability. But finding it is only half the battle — you still have to fix it. And fixing security vulnerabilities requires security expertise that most developers don't have."

SAY **The Problem:** "Traditionally: security scanner finds issue → creates ticket → developer doesn't understand the fix → ticket sits for weeks → vulnerability window stays open."

ACTION **Navigate to PR:** Open a PR that has a Copilot Autofix suggestion. Click on the Security tab within the PR.

SAY **Show Autofix:** "Look at this. Copilot analyzed the SQL injection vulnerability, understood the data flow, and generated a parameterized query fix. Right here, in the PR, inline. The developer doesn't need to know what parameterized queries are — Copilot explains it and writes the code."

SAY **The Numbers:** "Our research shows that Copilot Autofix reduces mean time to remediate security vulnerabilities by 60%. For a team that ships code weekly, that's the difference between a 4-week vulnerability window and a same-day fix."

SAY **Close the loop:** "So the complete story is: developer writes code with a vulnerability → GitHub Actions detects it in 15 minutes → CodeQL confirms it → Copilot generates the fix → developer reviews and merges. The entire cycle: under 30 minutes, zero security expertise required."

PAUSE **Power moment:** Let that sink in. "Zero security expertise required." Pause 4 seconds.

8. Closing — ROI Summary & Next Steps (5 min)

METRIC	WITHOUT GITHUB	WITH GITHUB ACTIONS + GHAS	IMPROVEMENT
Vulnerability detection	Quarterly pen test	Every 15 minutes	96x per day vs 4x per year
Detection lag	3-18 months	Minutes after commit	99%+ reduction
Remediation time	4-8 weeks	Same-day with Autofix	60% faster (GitHub data)
Security expertise needed	Dedicated AppSec team	Any developer	No specialist required
Breach probability	Baseline	Reduced by ~70%	IBM Security benchmark
Compliance audit prep	Manual, weeks	Automated reports, days	80% time reduction

Closing Script

"So here's what we've shown today. PropTracker had BOLA, SQL injection, broken auth, SSRF — 10 critical API vulnerabilities. Without GitHub, those could sit in production for months."

"With GitHub Actions, every one of those vulnerabilities is detected within 15 minutes of deployment — automatically, continuously, without human intervention."

"With GHAS — CodeQL, secret scanning, dependency review — you have defense in depth. Static analysis, runtime scanning, supply chain security."

"And with Copilot Autofix, your developers don't need to become security experts. The fix comes to them, in the PR, ready to review and merge."

"The question isn't whether Cushman & Wakefield has vulnerabilities in production today. Every API does. The question is: do you want to find out from GitHub, or from a researcher, or from a breach notification letter?"

Proposed Next Steps

1. GHAS Trial — 30-day free trial on your most critical repo. Zero commitment.
2. GitHub Actions PoC — We implement the 10 security check workflows in your actual codebase in 1 sprint.
3. Security Posture Assessment — GitHub's team reviews your current repos and provides a vulnerability density report.
4. Copilot for Business — 2-week Copilot trial for your development team.
5. Architecture Review — 1-hour session with GitHub Solutions Architect on your PropTracker / property API architecture.

9. Curveball Q&A — 25 Tough Questions with Answers

These are the hardest questions you will face. Practice these answers until they feel natural. The customer respects a presenter who doesn't flinch.

■ TECHNICAL QUESTIONS

Q: Our API already passes OWASP ZAP scans. How is this different?

A: OWASP ZAP is a black-box scanner — it pokes at your API from outside without understanding your code. CodeQL is a white-box semantic analyzer — it builds a graph of your code's data flow and finds vulnerabilities that ZAP literally cannot see, like multi-hop injection paths through helper functions. The BOLA vulnerability we just showed? ZAP can't find that — it doesn't know that job ID 1 belongs to Alice and not Bob. Only a workflow that understands your business logic can catch it. That's exactly what our custom Actions workflows do.

Q: We have Snyk / Checkmarx / Veracode. Why do we need GitHub too?

A: Those are excellent tools, and GitHub integrates with all of them. But here's the key difference: those tools are bolt-on — they scan outside the developer workflow. GHAS is built into the PR review process. The alert appears as a PR comment — the developer sees it before they can merge. There's no ticket to file, no Jira to triage, no 4-week delay. And Copilot Autofix means the developer gets the fix suggestion in the same PR review. We've seen customers run Snyk and GHAS in parallel — GHAS typically catches 30-40% additional issues because CodeQL's semantic analysis is different from dependency scanning.

Q: How do your Actions workflows handle false positives?

A: Great question — this is where custom workflows beat generic scanners. Generic scanners don't know your business logic, so they generate high false positive rates. Our check-bola.yml knows your specific endpoint structure, uses real test credentials, and checks the exact HTTP status code you expect. The BOLA check doesn't flag 'any access to jobs' — it specifically tests cross-user access. False positive rate for our behavioral workflows is near zero, because they're testing real behavior, not guessing from patterns. For CodeQL, GitHub publishes precision/recall data — it's tuned for high precision on the categories that matter most.

Q: What happens if our API has custom authentication — not JWT?

A: The workflows are fully customizable YAML — you write them like any shell script. If you use OAuth2 bearer tokens, cookie-based sessions, API keys, mTLS — you just swap out the curl headers. The detection logic — checking HTTP status codes and response bodies — stays identical. We'd spend 2-4 hours with your team adapting the workflows to your auth scheme during the PoC engagement.

Q: How does this work with our existing CI/CD pipeline?

A: GitHub Actions IS the CI/CD pipeline. Your existing build, test, and deploy workflows live side by side with the security check workflows. Zero integration work. If you're on Jenkins or Azure DevOps, GHAS code scanning and secret scanning work there too via the SARIF upload action — your existing pipeline runs CodeQL and uploads results to GitHub Security.

Q: Can GitHub Actions access our on-premises API?

A: For a PoC with your internal API: yes, using self-hosted runners. You install a GitHub Actions runner on a machine inside your network, and GitHub sends jobs to it. The runner has your internal network access. For production monitoring, self-hosted runners give you full control — nothing leaves your network except the job status metadata. We can walk through the architecture in detail.

Q: What about our proprietary code — we can't put source code in GitHub cloud.

A: GitHub Enterprise Server (GHES) is an on-premises installation — your code never leaves your data center. GHAS and Actions run locally. We have customers in financial services and defense running GHES with GHAS. Alternatively, GitHub Enterprise Cloud with data residency options (EU, US) and IP allow-listing. We can show you the compliance documentation for SOC 2, ISO 27001, FedRAMP Moderate.

■ BUSINESS QUESTIONS

Q: What's the total cost of ownership?

A: GHAS is licensed per active committer — typically $49/month/user for GitHub Enterprise Cloud with GHAS included. For a team of 20 developers: ~$1,000/month. Compare that to the $4.45M average cost of a single data breach (IBM 2023), or the 6 weeks of engineering time for a post-breach remediation sprint. GHAS typically pays for itself when it catches a single critical vulnerability per quarter. We can model the exact TCO against your team size in a follow-up.

Q: We already have a security team. Won't this eliminate their jobs?

A: The opposite — it empowers your security team to do strategic work instead of reactive triage. Right now your AppSec team is probably drowning in vulnerability tickets from quarterly pen tests. GitHub Actions and GHAS automate the detection and triage layer. Your security team shifts from 'find and report' to 'policy, architecture, and threat modeling.' We've seen AppSec teams double their strategic output after adopting GHAS because they're no longer drowning in routine scanning.

Q: How long does it take to implement?

A: For GitHub Actions security workflows: 1-2 sprints to adapt and deploy. For GHAS code scanning: 1 day — it's a 5-line addition to your existing CI workflow. Secret scanning: enabled with a checkbox. Dependency review: another checkbox. Full GHAS rollout for a 50-developer org: 4-6 weeks including training. The PoC I'm proposing is a 30-day engagement where you see real value on real code by week 2.

Q: What's the ROI data? Do you have case studies?

A: Forrester TEI study (2023): GHAS delivers 234% ROI over 3 years with payback in under 6 months for a 500-developer organization. Specific outcomes: 60% reduction in mean time to remediate, 40% reduction in security-related rework, 85% reduction in exposed secrets time-to-revocation. I'll send you the full Forrester report and can connect you with a reference customer in commercial real estate — similar stack, similar scale.

Q: We're in the middle of a digital transformation / migration. Is this the right time?

A: This is actually the best time. Security debt compounds — every line of code added to an insecure foundation makes remediation harder later. The BOLA vulnerability we showed? If that pattern is in 50 endpoints today, catching it now with CodeQL is a 1-day fix. Finding it in 500 endpoints after a 2-year migration is a 2-month remediation project. Shift-left security during a transformation has a multiplier effect — it costs less per fix than retrofitting security after the fact.

Q: We have compliance requirements (SOC 2, ISO 27001, GDPR). Does GitHub help?

A: Significantly. GHAS provides automated audit trails for vulnerability detection, triage, and remediation — exactly what SOC 2 Type II auditors want to see. Secret scanning directly addresses the SOC 2 CC6.7 control around credential management. Code scanning supports CC7.1 (detection of design deficiencies). For GDPR: preventing a BOLA breach that exposes personal data is a direct compliance control. We can map GHAS features to your specific compliance framework — send me your control framework and I'll prepare a mapping document.

Q: What's the competitive differentiation? Can't we get this from AWS Inspector or Azure Defender?

A: AWS Inspector and Azure Defender are runtime threat detection — they watch running infrastructure. GHAS is shift-left — it catches vulnerabilities before code is deployed. They're complementary, not competitive. The unique GitHub differentiator is the developer workflow integration: the fix suggestion appears in the PR, not in a separate security console that developers never open. The data shows that vulnerabilities caught in PRs are fixed 10x faster than vulnerabilities reported in separate ticketing systems.

■ OBJECTIONS QUESTIONS

Q: Our developers will ignore security alerts — they always do.

A: That's the traditional pattern, and it's exactly what we designed against. Three things break the ignore cycle: (1) PR blocking — GHAS can be configured to block merges on high-severity alerts. The developer can't ignore it because the merge button is red. (2) Copilot Autofix — the fix is right there, one click. The cognitive overhead goes from 'I need to learn parameterized queries' to 'does this suggested fix look right?' (3) Visibility — when security alerts appear in the same place as code review comments, they feel like normal development feedback, not a separate security compliance burden.

Q: We've tried automation before. Our team bypasses the checks.

A: Branch protection rules prevent bypassing. GHAS alerts on the default branch require explicit dismissal with a justification — dismissed alerts are logged and auditable. You can configure required reviewers for security alert dismissals — your AppSec team reviews any developer who dismisses a critical alert. The paper trail is complete. For the workflows specifically, workflow bypass requires admin permissions on the repo — you control who has that.

Q: This looks like a lot of maintenance — who maintains all these YAML files?

A: Once written, security check workflows are remarkably stable. The BOLA check we showed hasn't needed changes since we wrote it — it checks an HTTP status code. What changes is your application, and when it does, the workflow catches new regressions automatically. The maintenance burden is much lower than alternatives: no agent to install, no server to patch, no signature database to update. GitHub maintains

the runner infrastructure. Your team maintains the business logic in the workflows — maybe 2 hours per quarter.

Q: We're not sure our exec team will approve the budget.

A: Let's quantify it for them. Take your team size and your average fully-loaded developer cost. A single security incident that requires 6 engineers for 8 weeks costs roughly $240,000 in engineering time alone — not including legal, compliance, or customer notification. GHAS at $49/user/month for 20 developers is $11,760/year. That's one 1-week security incident worth. I can prepare a 1-page business case document with your specific numbers that your CTO or CISO can present to the CFO.

Q: Can you prove this would have caught the Log4Shell / [recent breach] vulnerability?

A: Yes. Dependency scanning with Dependabot would have alerted on log4j-core ≥2.0-beta9 and <2.15.0 within hours of the CVE publication on December 9, 2021. GitHub published the advisory at 4:17 PM EST on December 9 — Dependabot alerts would have gone out to every repo with that dependency by 4:30 PM. Secret scanning caught the follow-on exploitation attempts in GitHub-hosted code. The combination of dependency review blocking PRs that add vulnerable log4j, plus Dependabot PRs with safe version upgrades, is exactly the workflow that protected GitHub's own infrastructure during Log4Shell.

Q: What if the API is written in a language GitHub doesn't support?

A: CodeQL supports C, C++, C#, Go, Java, JavaScript/TypeScript, Python, Ruby, Swift, and Kotlin — covering 95%+ of enterprise application code. For languages not yet supported, the custom Actions workflow approach (behavioral testing via HTTP) is language-agnostic — it doesn't look at code at all, it probes the running API. Your PropTracker Node.js API, your Python services, your Java microservices — the behavioral workflows catch vulnerabilities regardless of language.

Q: How does this integrate with our SIEM / security tooling?

A: GHAS exports to SARIF — the industry-standard security results format. Every SIEM worth using (Splunk, Microsoft Sentinel, Elastic SIEM) has a GitHub GHAS connector or accepts SARIF ingestion. Secret scanning alerts can trigger webhooks to your SIEM in real time. GitHub also has native integrations with ServiceNow (auto-create incidents), Jira (auto-create tickets), and PagerDuty (on-call alerts for critical findings). We'd do a 1-hour integration mapping session during the PoC.

---

10. Post-Demo Follow-Up Actions

■ Within 24 hours

- Send thank-you email with: demo recording link, Forrester TEI study PDF, GHAS feature comparison sheet
- Send custom business case document with their team size and estimated ROI numbers
- Connect them with a reference customer (commercial real estate / property management)
- Send PropTracker source code link so their team can explore the vulnerability implementations

■ Within 48 hours

- Schedule the GHAS Trial kickoff call — 30 minutes, technical team + GitHub SA
- Send GitHub Enterprise pricing proposal (if they asked about cost)
- Loop in GitHub Solutions Architect for technical deep-dive on their specific stack

■ Within 1 week

- Deliver the GHAS PoC on their repo (if they approved the trial)
- Send custom GitHub Actions workflow templates adapted to their API structure
- Provide compliance mapping document (GHAS controls → their framework)

Email Template — 24-Hour Follow-Up

```
Subject: PropTracker Demo Follow-Up + GHAS Trial Next Steps

Hi [Name],

Thank you for your time today. I hope the PropTracker demo made the vulnerability risks
and GitHub's automated detection capabilities concrete and tangible.

As promised, here are the resources we discussed:
• PropTracker source code: github.com/sautalwar/cushman-property-api
• Forrester TEI Study: [link]
• GHAS Feature Overview: [link]
• Custom ROI model for Cushman & Wakefield: [attached]

Based on our conversation, I'd recommend starting with a 30-day GHAS trial on your
[specific repo they mentioned]. I can have the GitHub Solutions Architect set this up
with you in a 1-hour kickoff session — no commitment required.

Are you available [date/time options]?

Best,
[Your name]
```

11. Appendix — Technical Reference

OWASP API Security Top 10 Quick Reference

#	CATEGORY	PROPTRACKER EXAMPLE	SEVERITY
API1	Broken Object Level Authorization	Bob accesses Alice's job records	Critical
API2	Broken Authentication	Expired JWTs accepted by middleware	Critical
API3	Broken Object Property Level Auth	Mass assignment elevates role to admin	High
API4	Unrestricted Resource Consumption	1MB payload DoS / limit=99999 query	Medium
API5	Broken Function Level Auth	Any contractor marks any job complete	High
API6	Unrestricted Access to Business Flows	Bid flooding with no rate limiting	Medium
API7	Server Side Request Forgery	Webhook URL fetches internal metadata	Critical
API8	Security Misconfiguration	CORS wildcard + credentials: true	High
API9	Improper Inventory Management	N/A in this demo	-
API10	Unsafe Consumption of APIs	SQL injection via search parameter	Critical

Key URLs & Resources

Demo App: http://localhost:5175

API Health: http://localhost:3001/health

GitHub Repo: https://github.com/sautalwar/cushman-property-api

GitHub Actions: https://github.com/sautalwar/cushman-property-api/actions

Code Scanning: https://github.com/sautalwar/cushman-property-api/security/code-scanning

Secret Scanning: https://github.com/sautalwar/cushman-property-api/security/secret-scanning

Security Issues: https://github.com/sautalwar/cushman-property-api/issues

GHAS Documentation:
https://docs.github.com/en/get-started/learning-about-github/about-github-advanced-security

Forrester TEI Study: https://resources.github.com/security/forrester-tei-github-advanced-security/

OWASP API Top 10: https://owasp.org/API-Security/editions/2023/en/0x00-header/