A Case Study in AI Assisted Development and Rapid System Delivery

Building a Production Quality, NIST SP 800 53 Rev. 5 Aligned Tabletop Exercise Platform in Approximately Four Hours of Active Work

Darren Death

March 17, 2026

Contents

Figures

Tables

I built a production quality, NIST SP 800 53 Rev. 5 aligned, multi-player cybersecurity tabletop exercise platform with a real time game engine, three AI provider integrations, a full ATO documentation package, and Docker deployment in approximately four hours of active work spread across two days.

By the end of that effort, the system was running, deployed, and documented. The codebase reached 15,628 lines across the application, service layer, authentication stack, real time session handling, reporting functions, and administrative interfaces. The documentation package was generated against the actual implementation, which means the platform can be reviewed as an operational system with a defined boundary, a control structure, and supporting evidence.

Speed is part of the story, but it is not the most important part. What matters more is where the system ended up. This was a complete build with working functionality, security architecture, documentation, and operational visibility. Reaching that point would normally take much longer and would usually involve more people.

I also used this build to see what happens when cATO concepts are applied during development instead of being added after the system already exists. The platform includes an operational security dashboard that grades the security posture of the application based on implemented capabilities and observable conditions. I added that intentionally because I wanted to understand how continuous control visibility would affect the architecture, the documentation package, and the development process itself.

What was Built

CyberTabletop is a gamified incident response tabletop exercise platform built to replace the static, slide based exercises most organizations still run. The design assumption was simple. Incident response exercises work better when participants are making decisions inside a live environment with role-based inputs, immediate consequences, and measurable outcomes.

The platform supports live multiplayer sessions where each participant joins from a separate device and receives role specific scenario injects in real time. It uses a click to respond decision model so teams can make decisions quickly. Each response produces immediate AI generated feedback tied to NIST CSF and MITRE ATT&CK. The system also supports competitive scoring and leaderboards. At the end of an exercise, the platform generates an After-Action Report that reads like a formal hot wash document, and facilitators can create or modify scenarios through a built-in scenario builder rather than through code changes.

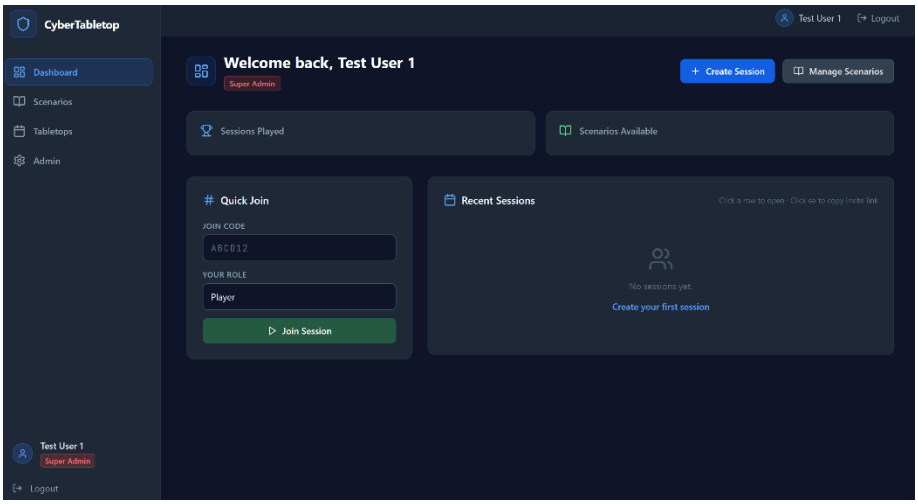

Figure 1: Main Dashboard

The initial release includes three prebuilt scenarios. Operation Black Fog addresses ransomware. Operation Silent Harvest addresses PII exfiltration. Operation Gray Hat addresses insider threat conditions. Together, these scenarios include 38 decision injects with four scored response options and scripted feedback tied to realistic operational consequences. That content is built into the platform and does not depend on a separate service offering or custom consulting engagement.

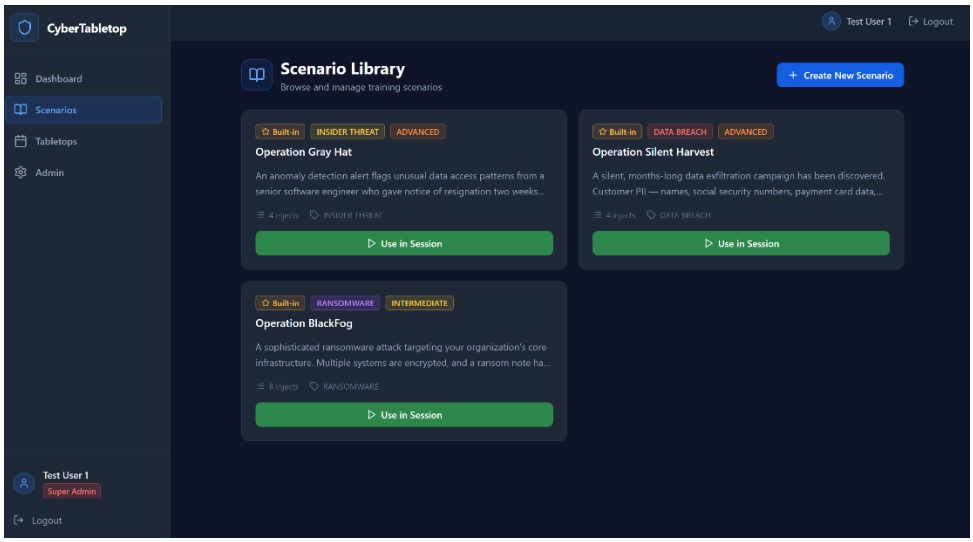

Figure 2: Built-in Scenario Library

By the Numbers

Metric	Value
Total lines of code	15,628
Backend TypeScript, API, services, auth	6,149 lines
Frontend TypeScript / React, UI, pages	9,479 lines
Frontend pages	14 pages
Backend API route files	6 files
Backend service modules	13 modules
Database models, Prisma / PostgreSQL	10 models
AI provider integrations	3, local model, cloud model, scripted fallback
Editable AI prompt templates	Unlimited
NIST SP 800 53 compliance documents	13 documents
Deployment targets documented	Onprem, AWS, Azure
Pre built scenarios with full content	3 scenarios / 38 injects
RBAC roles	4
Auth methods	3
Real time infrastructure	live session engine
Containerization	Docker Compose, 5 services
Security visibility capability	Operational security dashboard with application grading

Table 1: Application Metrics

How the Build Actually Worked

The total elapsed time across both days was approximately eight hours. My active work, writing requirements, reviewing generated code, testing the running application, and describing bugs, amounted to roughly four hours. The remaining time was spent by the AI generating implementation.

That distinction matters because the human work changed. I was not spending the full period writing code file by file. I was defining requirements, validating structure, testing the running system, and identifying issues precisely enough for the implementation to be corrected. The implementation workload shifted to the AI. The human workload shifted to intent, review, architecture judgment, and validation.

Before code generation started, I described the functional requirements, user roles, compliance expectations, deployment model, and security posture in plain language. The AI generated a build plan covering the technology stack, schema, project structure, security model, and implementation sequence within minutes. In a conventional build, that level of design work would typically consume a significant amount of senior engineering time before implementation even started.

The platform implemented a structured security model for authentication, authorization, and auditability, with controls designed to support access enforcement, identity assurance, and traceable user activity. Sensitive administrative and user actions were recorded in a protected audit trail, and the overall control design aligned to the access control, identification and authentication, and audit expectations reflected in NIST SP 800 53 Rev. 5. This was built as part of the application architecture.

The real time engine manages inject delivery, decision collection, and score updates across active participants while giving facilitators immediate visibility into team responses as they occur. The scoring model evaluates participant actions against the scenario logic in real time and adjusts outcomes based on response quality, timing, role context, and learning mode behavior. The frontend spans fourteen pages and includes administrative functions for user oversight, prompt management, audit review, active session visibility, and security control monitoring. The debrief workflow generates an in browser After Action Report with participant details, NIST CSF heat maps, MITRE ATT&CK references, scenario replay, and a structured hot wash suitable for operational review.

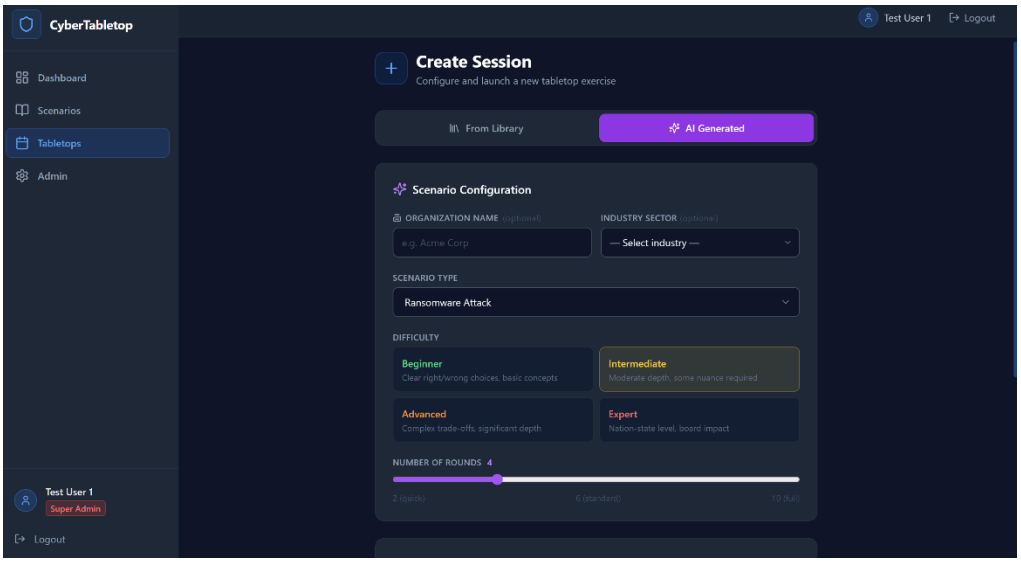

Figure 3: AI Generated Scenarios

The platform also integrates three provider paths behind an abstraction layer so it can operate in connected and constrained environments. It supports a primary cloud model, a local model for controlled deployments, and a scripted fallback that maintains continuity when generation is unavailable. The prompt structures were built to generate technically credible injects with ATT&CK mappings, NIST CSF function tags, scored response options, and consequence narratives. Those templates are editable by administrators from within the application.

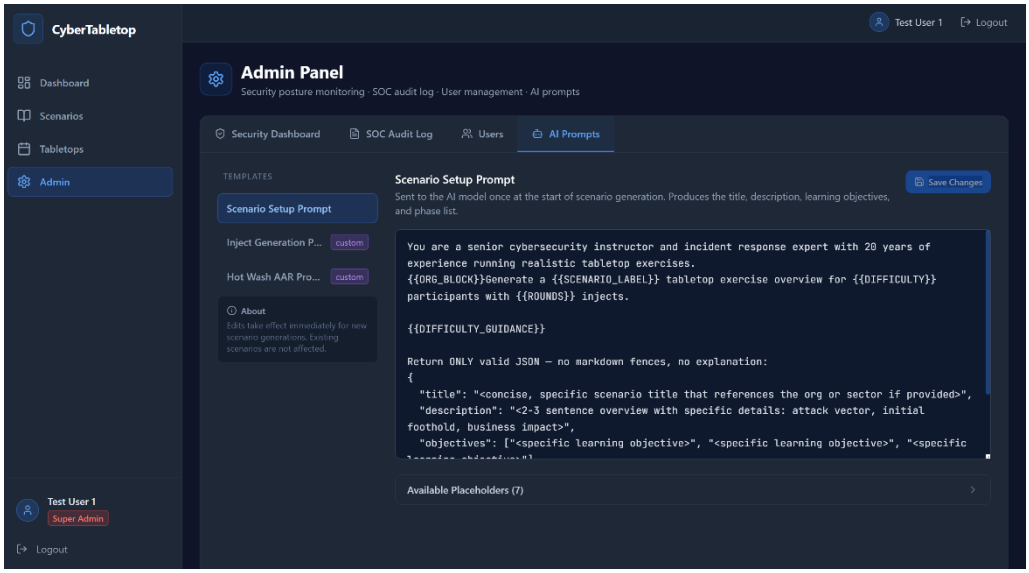

Figure 4: AI Model Prompt Managment

The compliance package includes 13 documents, including the System Security Plan, Privacy Impact Assessment, Security Assessment Plan, Security Assessment Report template, Plan of Action and Milestones, Contingency Plan, Incident Response Plan, Configuration Management Plan, Risk Assessment, FIPS 199 Categorization, a NIST SP 800 53 Rev. 5 control matrix, a boundary diagram, and a data flow diagram. The documentation was produced in minutes and aligned to the actual system architecture rather than being assembled from disconnected boilerplate.

cATO Concepts in the Build

I did not treat security documentation and operational visibility as downstream deliverables. I used this build to see what changes when cATO concepts are applied as part of development and implementation. That meant the application, the control evidence structure, and the operational security view were developed together.

The platform includes an operational security dashboard that grades the security posture of the application. That grading is tied to implemented capabilities and system conditions rather than to static policy language. I wanted to understand the effect of that visibility on the design itself. It pushed the system toward clearer control boundaries, stronger traceability between features and controls, and better discipline around what had actually been implemented versus what would normally be deferred to the documentation phase.

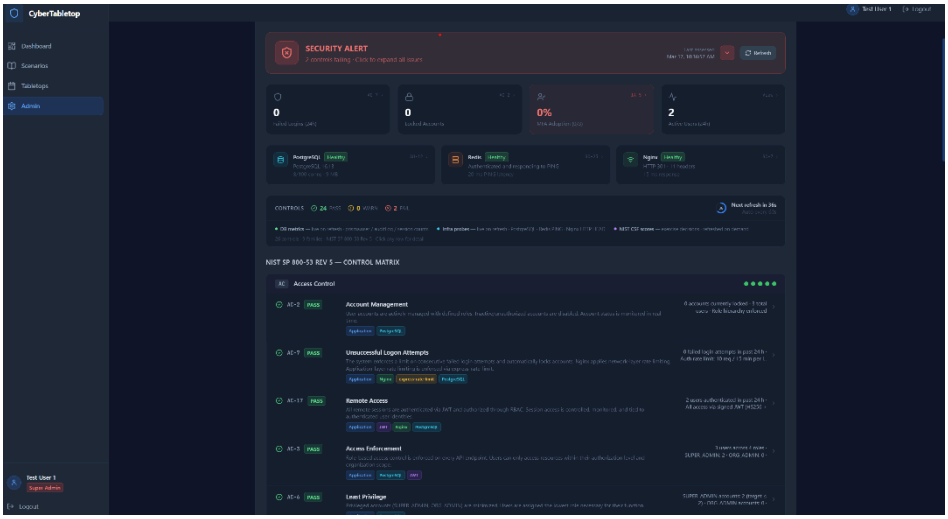

Figure 5: Security Dashboard

Time Analysis

Assuming this system would take four to six months with a two person team, roughly 552 to 804 developer hours. My actual active human investment was approximately four hours, spread across eight hours of elapsed time over two days.

Component	Traditional Development	My Active Time
Architecture and system design	40 to 60 hrs	15 to 20 min
Authentication system	40 to 56 hrs	20 to 30 min
Backend API and services	64 to 96 hrs	30 to 40 min
Real time session engine	24 to 40 hrs	10 to 15 min
Frontend, 14 pages	120 to 160 hrs	60 to 75 min
AI integration and prompt engineering	40 to 56 hrs	15 to 20 min
Pre built scenario content, 38 injects	64 to 96 hrs	10 to 15 min
NIST documentation package, 13 documents	80 to 120 hrs	10 to 15 min
Docker, deployment, Nginx	16 to 24 hrs	10 to 15 min
Seed data and database migrations	24 to 40 hrs	5 to 10 min
Feature iterations and bug fixes	40 to 56 hrs	45 to 60 min
TOTAL	552 to 804 hrs	Approximately 4 hrs active / 8 hrs elapsed

Table 2: Time Analysis

At a blended contractor rate of $150 per hour, a traditional build would fall between $82,800 and $120,600. My active labor investment was the equivalent of about $600. On that basis, the reduction in direct human labor was roughly 138x at the low end and just over 200x at the high end. Using the midpoint of the estimated traditional range, the reduction was approximately 161x.

The midpoint is probably the cleanest number to use because it shows the scale of the change without overstating the case. The better way to frame it is that the workflow reduced direct human labor by roughly 160x while still producing a deployed system with a documentation package, a defined control structure, and operational security visibility.

To be precise about what those four hours looked like, I wrote requirements in plain English, reviewed generated code for correctness, opened the application to test behavior, and described bugs accurately when they appeared. Most of the implementation keystrokes were produced by the AI, but the product decisions, architectural judgment, security review, and acceptance decisions remained mine.

What Made the Workflow Effective

AI assisted development only works if the workflow around it is disciplined. In this case, the value was not just that the code was generated quickly. The value was that I could define the requirements clearly, test the system while it was running, and correct issues as soon as the implementation drifted from the intended behavior.

The process worked because the requirements were specific, the review cycle was immediate, and the output was validated against the actual application instead of being accepted at face value. The AI handled most of the implementation across the codebase. I handled the product decisions, the security expectations, the testing, and the judgment about whether the result was actually good enough to keep. That is what made the pace possible without losing control of the build.

Why This Matters Across the Business

The impact is bigger than one security platform or one development workflow. When implementation speed changes this much, the effect does not stay inside engineering. It changes how business teams think about capability gaps, how quickly leaders can move from an identified need to a working system, and how much justification is required before someone starts building.

CyberTabletop is a good example of that shift. In a traditional environment, an application like this would usually be broken into phases, scoped into future releases, and pushed through a long sequence of design, development, and compliance steps. The exercise engine would likely come first. The reporting functions would come later. The documentation package would be treated as a separate workstream. The operational security dashboard might never make it into the initial release at all. What

changed here was not just the speed of code generation. What changed was the ability to build the application, the security model, the documentation, and the operational visibility together as one system.

That has implications across the business. Security teams can build tools that reflect their actual operating environment instead of adapting their work to whatever a commercial product supports. Compliance and audit functions can review systems that already contain supporting evidence and control traceability instead of waiting for documentation to be assembled after the fact. Program and mission teams can move more quickly when they need a focused capability that would not justify a major procurement effort but still needs to be built correctly. Leadership can evaluate a working system with real functionality, actual controls, and observable behavior instead of making funding decisions based only on requirements documents and vendor claims.

It also changes the economics of internal development. A capability that would have been treated as too expensive, too slow, or too difficult to justify can now be developed and tested directly against the actual use case. In the case of CyberTabletop, that meant building a platform tailored to how incident response exercises should operate, not how slide based exercises have traditionally been tolerated. The same dynamic applies well beyond tabletop exercises. It applies to internal workflow tools, decision support applications, reporting systems, security dashboards, intake platforms, and other systems that business units often need but struggle to get built through normal delivery timelines.

The bigger point is that the barrier between identifying a need and delivering a functioning application is lower than it used to be. That does not eliminate the need for strong developers, sound architecture, or disciplined review. It does change what the organization should expect once requirements are clear and the work is being led by someone who can direct the build, validate the output, and keep the implementation aligned to the intended result.

How Development Processes Need to Change

The development process cannot stay the same if implementation speed changes this much. Organizations still need talented developers. That requirement does not go away. What changes is where their time should be spent and how quickly delivery is expected to move once requirements are defined.

Teams need stronger technical judgment, not less. They need people who can define architecture clearly, recognize weak implementations, test behavior in a running system,

and make good decisions about security boundaries, data handling, and control enforcement. That is where the value shifts. The bottleneck is less about typing code and more about whether the team can direct, review, and validate the system effectively.

Timelines also have to shrink. A process built around long implementation cycles, delayed review points, and late stage security involvement will not hold up when working capability can be assembled in hours or days. Development governance, security review, and documentation practices have to move closer to the build itself. If they do not, organizations will either slow themselves down artificially or end up with systems moving faster than their control processes can handle.

Security additions also need to be treated differently. They should not be risk managed away because they were not properly planned for. If a control is necessary for the application to operate securely, then it is a build requirement. It is not a discretionary enhancement to be negotiated down after timelines slip or scope gets compressed. Faster development should reduce excuses for deferring security implementation, not create new ones.

That is one of the more important lessons from this project. When implementation becomes faster, the standard for planning has to get better. Requirements need to be clearer at the start. Security needs to be designed into the system from the beginning. Evidence needs to be generated as the system takes shape. If teams keep treating security as something to bolt on later, they will move quickly in the wrong direction.

What Comes Next

CyberTabletop is a working platform, but the larger point is not limited to one application or one use case. It demonstrates what changes when AI assisted development becomes part of the delivery model for internal systems. The practical effect is that organizations can move from a defined requirement to a functioning application much faster than traditional development timelines would suggest, as long as the work is being led by someone who can set direction, evaluate the output, and validate the system as it takes shape.

That has implications well beyond cybersecurity. The same development model can be applied to internal business applications, workflow tools, reporting platforms, operational dashboards, intake systems, and other capabilities that often sit in backlog because the projected timeline, staffing model, or procurement path makes them hard to justify. When implementation speed increases at this level, organizations have an opportunity

to build systems around actual operational needs instead of waiting for a vendor roadmap or accepting a manual workaround.

For CyberTabletop, the more important point is not whether this specific platform becomes a long term product. I built it as an experiment to understand what AI assisted development can actually produce when the requirements are clear, the review process is disciplined, and the system is built with security, documentation, and operational visibility in scope from the beginning. I may continue to develop it, or I may apply the same approach to other use cases. Either way, the value of the effort was in testing the capability, understanding its impact on the development process, and seeing how far a single person could move a real system in a very short period of time.