From Checklist to Context

LLM-Driven STIG Compliance for z/OS Environments

How generative AI uncovers mainframe security vulnerabilities that pattern-based tools cannot.

May 13, 2026

geniez

Executive summary

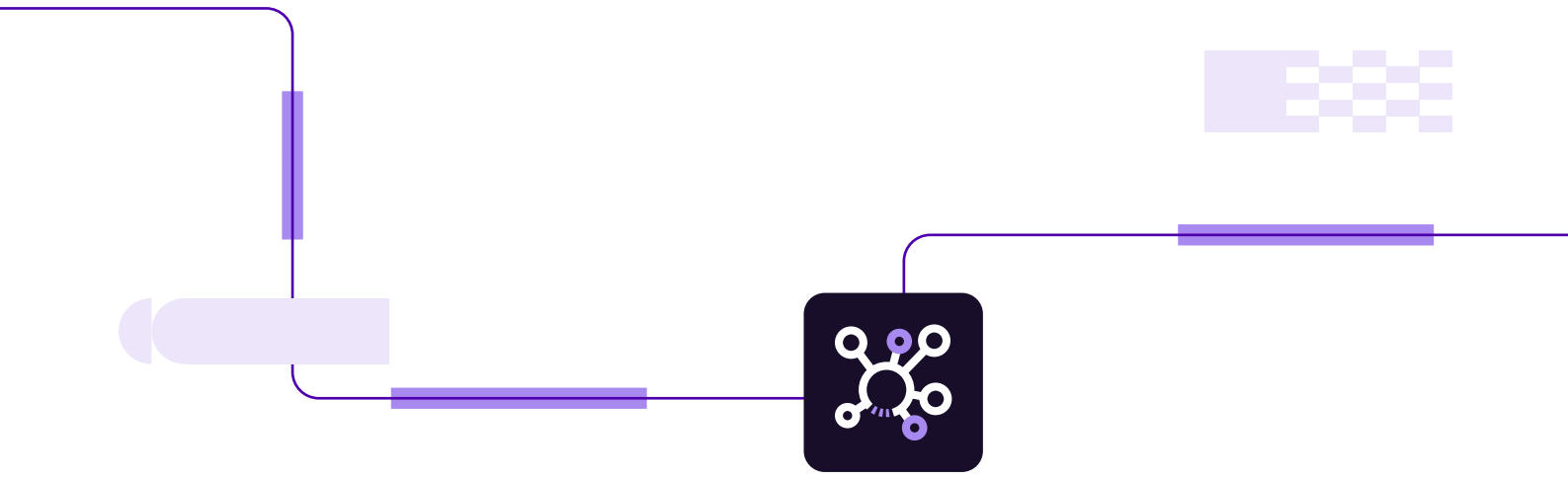
Mainframes run the world's most consequential workloads: core banking, payments, insurance policy administration, government benefits, airline reservations, and large parts of healthcare and retail. They also run on software stacks whose architectural assumptions were set in the 1960s and 1970s. After more than forty years of growth through acquisitions, conversions, and ad-hoc remediations, no large z/OS estate is fully understood by any one person. STIG checklists were created to bring discipline to this complexity, but they have always been limited by two facts: a checklist is only as good as what its authors thought to check and defined in an automated system to check.

Generative AI changes the economics of that review. Frontier large language models (LLMs) now read code, configuration, and documentation the way an experienced researcher does, reasoning about context rather than matching predefined patterns. In early 2026, a single Anthropic model identified over five hundred high-severity zero-day vulnerabilities in open-source code that had survived years of fuzzing, while AI-augmented research at Wiz produced a remote code execution flaw affecting GitHub.com itself*. These are not academic results; they are the new baseline.

This paper explains, in plain language, why LLMs detect classes of mainframe security exposure that traditional tools and manual STIG reviews routinely miss, and how Geniez delivers that capability against z/OS, RACF, ACF2, and Top Secret without compromising safety. The argument is straightforward: if a frontier model can reason about an entire codebase, it can reason about your mainframe. Geniez is the connective layer that puts the mainframe in front of the model and the guardrails that keep it within the bounds of each user's existing permissions.

---

* https://www.wiz.io/blog/github-rce-vulnerability-cve-2026-3854



The Problem with Today's STIG Reviews

DISA's z/OS Security Technical Implementation Guides exist for good reason. They codify hard-won knowledge about how to lock down a mainframe: which RACF resource classes must be active, which SETROPTS options must be set, which APF-authorized libraries should not be world-readable, which started tasks should be protected, and hundreds of other controls. Equivalent guidance exists for ACF2 and Top Secret.

In theory, automated tools and manual reviewers walk the checklist and report findings. In practice, three problems recur:

1

Checklists describe what the author thought to ask

A STIG check is a question with a predefined right answer. It can verify that AUDIT(ALL) is active for a particular resource class, but it cannot ask whether the combination of an unusual JCL exit, a permissive UACC on a dataset profile, and a started task running with elevated authority creates a privilege escalation path. The interesting vulnerabilities in mature mainframe estates almost always live in the gaps between checks.

2

Tools detect patterns, not intent

Static analyzers and configuration scanners look for known-bad strings, known-bad option values, and known-bad combinations. They do not read the surrounding REXX, COBOL, or assembler. They cannot reason about why a particular library is APF-authorized or whether the business justification still holds. A scanner that has not been told about a specific pattern is blind to it.

3

Scale and longevity defeat human review

A typical z/OS site has accumulated decades of profiles, datasets, exits, started tasks, JCL procedures, parmlib members, and security manager rules. Many were authored by people who have long since retired. The total surface area is too large for any human team to review with consistency at quarterly cadence, let alone continuously. STIG reviewers triage; they cannot exhaust.

Why this is worse on the mainframe then elsewhere

Distributed systems are rebuilt every few years; mainframes are extended, not replaced. A z/OS LPAR running today often carries configuration decisions made decades ago. The institutional memory required to evaluate those decisions has eroded faster than the systems themselves. This is precisely the kind of environment where a tireless reasoning system — rather than another pattern matcher — produces step-change improvement.

Generative AI Is Categorically Different

For readers new to this technology, the most useful framing is this: **traditional security tools are spell-checkers; LLMs are editors. A spell-checker compares words against a dictionary. An editor reads the document, understands what you meant, and tells you that paragraph three contradicts paragraph one. Both are useful. They are not the same kind of tool.**

How traditional tools work

Conventional STIG and vulnerability scanners follow a deterministic pipeline. A human expert writes a rule, for example, "the RACF SETROPTS option PROTECTALL should be set to FAIL" and the scanner mechanically checks every system against that rule. The strength of this approach is repeatability. The weakness is that the scanner is, in a precise sense, ignorant. It does not understand RACF; it understands the rule it was given about RACF.

Consequences follow directly. Every new attack technique requires a new rule. Every new STIG version requires the tool vendor to update its content. Every site-specific naming convention requires custom configuration. And every vulnerability that does not fit a pre-existing pattern — the entire long tail of business-logic and composition flaws — is invisible to the tool by construction.

How LLMs work

Frontier LLMs are trained on an enormous corpora of text, code, configuration, and technical documentation. Through that training they acquire something closer to a working model of how systems are built and how they fail. When such a model is shown a RACF database extract, a parmlib member, a JCL procedure, or an exit's source code, it can reason about that artifact in its actual context: what the artifact does, what assumptions it makes, what would happen if those assumptions were violated, and what other artifacts in the same system interact with it.

This is qualitatively different from pattern matching. A pattern matcher cannot draw a conclusion that no human ever wrote down for it. A reasoning model can and increasingly does. That is why Claude Opus 4.7 and OpenAI's GPT5.5-cyber models, were able to find vulnerabilities in open-source codebases that had absorbed millions of CPU-hours of fuzzing without yielding them.

The two properties that matter most for mainframe security

Context-awareness: the model learns your environment

- A generic STIG check has no concept of your conventions. It does not know that your batch jobs use a particular high-level qualifier, that your shop reserves certain UIDs for specific functions, or that a particular exit was added during a long-ago merger and has been quietly bypassed ever since. An LLM, given access to your configuration, picks up these patterns the same way a new senior engineer would: by reading. It can then flag deviations that a generic rule would never catch, because the rule has no notion of what is normal for your site.

A 24/7 researcher: agentic operation

- Modern LLMs can run as agents. They can hold a goal: "find STIG violations and emerging exposures in this z/OS environment" and pursue it across many small steps without supervision. They examine one profile, follow a reference into a parmlib member, check a related dataset, correlate with a JCL exit, write up a finding with evidence, and move on to the next lead. This is the work a senior security researcher does. Until recently, it was bottlenecked by the supply of senior security researchers. It no longer has to be.

Dimension	Traditional STIG Tools	LLM-Driven Approach
Detection method	Predefined rules and signatures written by humans in advance	Reasoning across configurations, code, and documentation in context
Scope	What the rule author anticipated	Anything the model can read and reason about, including unfamiliar patterns
Environment knowledge	Generic; same checks for every site	Learns your naming conventions, exits, started tasks, and historical context
Coverage of gray areas	Misses business logic flaws and combinatorial misconfigurations	Catches issues that span RACF, JCL, parmlib, and exits together
Cadence	Scheduled scans, then a backlog for human review	Continuous; works through findings 24/7
Output	Pass/fail per checklist item	Findings with explanation, evidence, severity, and remediation
Adaptability	Requires a new rule for each new threat	Generalizes to new STIG versions and emerging misconfigurations

Three Recent Examples from the Distributed World

The case for applying LLMs to mainframe security rests on results these models have already produced in other environments. Three recent findings are instructive because, in each case, the LLM found something that traditional tooling had missed despite years of trying.

Claude Opus 4.6 — 500+ zero-days in hardened open-source code

In February 2026, Anthropic's frontier red team pointed Claude Opus 4.6 at production open-source codebases with no specialized scaffolding — just Python, standard debuggers, and fuzzers. The model identified more than five hundred previously unknown high-severity vulnerabilities, each independently validated by Anthropic staff or external researchers. Several of the affected projects had accumulated millions of fuzzer CPU-hours without yielding these bugs.

The mechanism is what matters. Fuzzers throw random inputs at code until something breaks. The model read the code and reasoned about it: tracing data flows, checking commit histories for variants of partially-fixed bugs, and identifying logic that would break under specific inputs. In one case involving the GhostScript project, the model examined the Git commit history after fuzzing failed and proactively wrote its own proof-of-concept exploit. This is the same reasoning that finds layered misconfigurations in RACF or ACF2 — checking what was almost-fixed, what is inconsistent across rule sets, what a recent change might have undermined elsewhere.

Wiz Research — RCE in GitHub via AI-aided reverse engineering

In March 2026, the Wiz Research team reported CVE-2026-3854, a remote code execution vulnerability in GitHub's internal git infrastructure that affected GitHub.com and GitHub Enterprise Server. The flaw allowed any authenticated user with push access to execute arbitrary commands on backend servers using a single git push command, and on GitHub.com it gave access to shared storage nodes holding millions of repositories belonging to other organizations.

Wiz had been examining GitHub's pipeline since 2024 but could not justify the engineering cost of reverse-engineering its compiled binaries by hand. Using Claude Code with an AI-augmented reverse engineering workflow, they went from idea to working exploit in less than 48 hours. As they noted on disclosure, this is one of the first critical vulnerabilities discovered in closed-source binaries using AI — a category that closely resembles much of the mainframe stack, where load modules, exits, and vendor products are often analyzed without source.

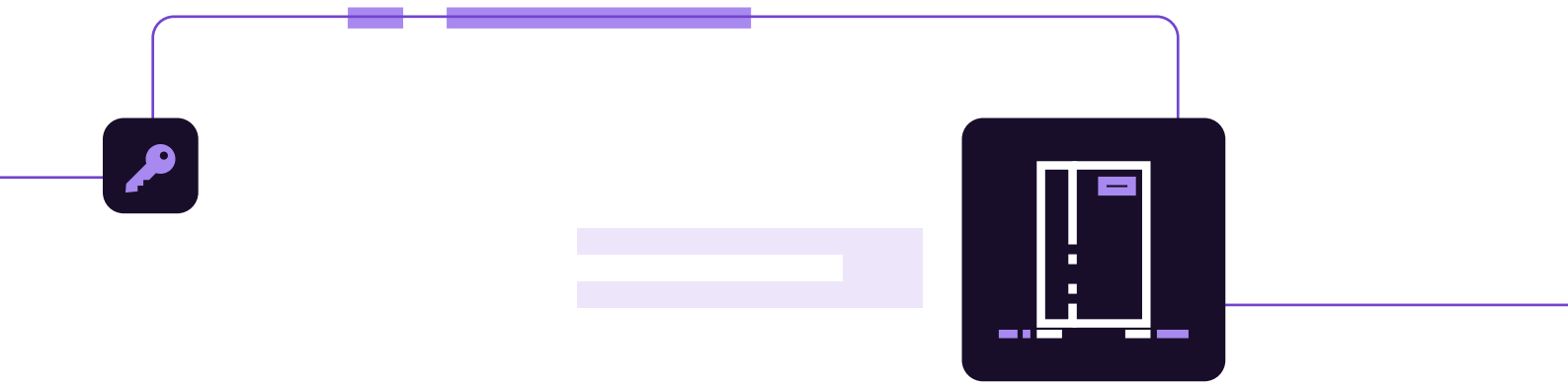
Linux kernel use-after-free — found by OpenAI o3 with no tooling

Independently, security researcher Sean Heelan used OpenAI's o3 model with no custom tooling and no agentic framework to identify CVE-2025-37899, a previously unknown use-after-free vulnerability in the Linux kernel's SMB implementation. The model analyzed over 12,000 lines of code and identified a race condition that traditional static analysis tools had consistently missed because detecting it required understanding concurrent thread interactions across multiple connections.

The pattern across all three examples is the same: in each case, the model did something that traditional tools structurally cannot — reasoning about complex, multi-component interactions in real code. The mainframe equivalents are everywhere. They are the cross-component reasoning about RACF profiles, dataset permissions, started task authorities, and exit behavior that a generic STIG check cannot perform.

Why this matters more on the mainframe

Open-source projects benefit from continuous public review by thousands of developers. Even so, decades-old bugs survived in them until an LLM was pointed at the code. A typical large z/OS estate has had nothing comparable — no public scrutiny, no millions of CPU-hours of fuzzing, often no recent line-by-line review of its exits and configuration. The vulnerabilities that survived in Firefox or GhostScript exist in greater density, and with greater longevity, **in mainframe environments. They have simply never been looked for at this resolution before.**



Alignment: What Makes Modern LLMs Safe to Deploy

A reasonable question follows naturally: if these models are powerful enough to find zero-days, are they safe to point at production mainframes? The answer rests on a body of work the AI industry calls alignment.

What alignment means

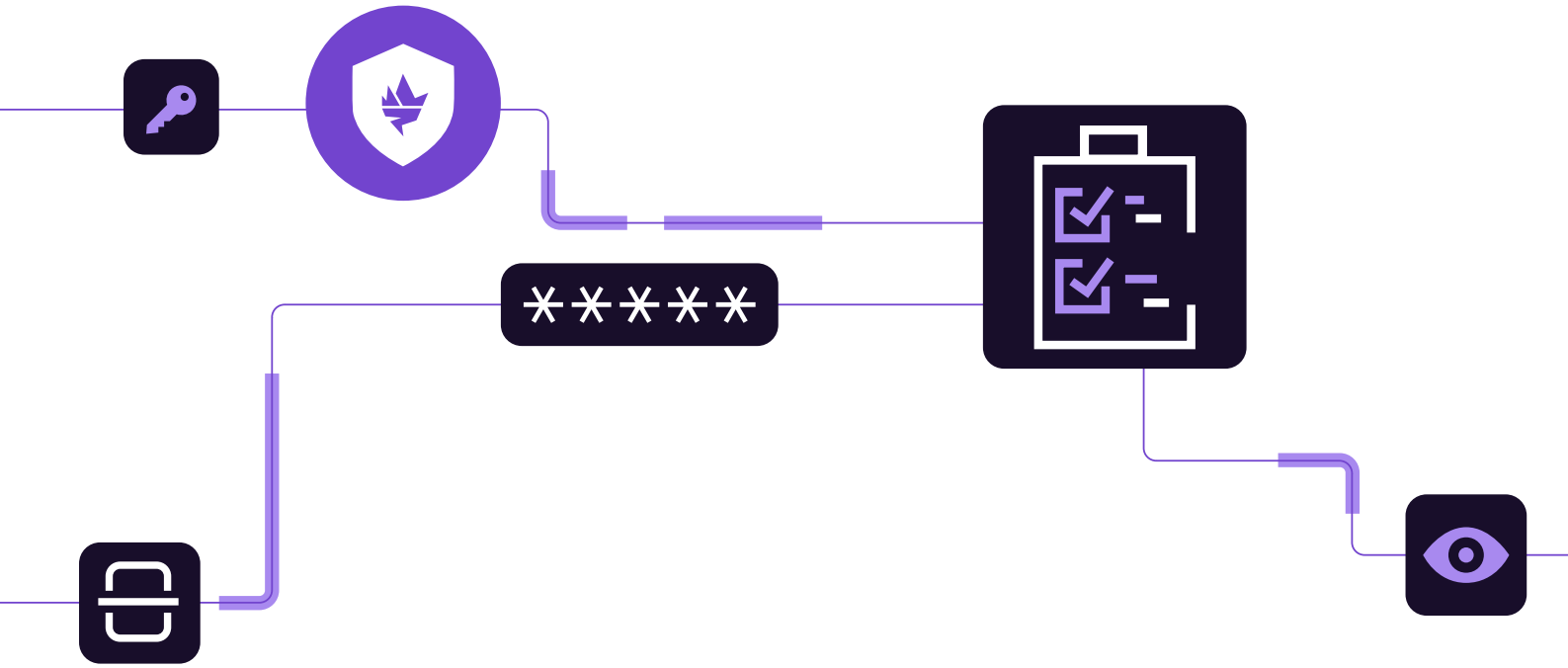
Alignment is the discipline of making AI systems behave in accordance with human intent and values — helpful, honest, and within stated boundaries. It is not a marketing layer applied after training. It is built into the training process itself, through techniques such as reinforcement learning from human feedback, constitutional AI, and adversarial red-teaming. Alignment determines, for example, that a model asked to help with a security review will produce a security review rather than weaponized exploit code, and that it will refuse instructions that fall outside its operator's intent.

Who is doing this work

Every major frontier lab maintains dedicated teams whose mandate is alignment and safety:

- **Anthropic** runs Alignment, Alignment Science, Alignment Stress-Testing, Interpretability, and Frontier Red Team groups. Their public output includes the Sleeper Agents research, model organisms of misalignment, joint evaluations with OpenAI, and the cyber-specific activation probes shipped alongside Claude Opus 4.6.
- **OpenAI** runs the Preparedness Framework and an internal safety evaluations function, and has published joint findings with Anthropic on cross-evaluating each other's models.
- **Google DeepMind, Meta, and others** maintain comparable safety functions, with growing collaboration through bodies such as the AI Safety Institutes in the US and UK.

This is the largest concentrated investment in technology safety research in the industry's history. It does not eliminate risk, but it is the reason serious enterprises can now contemplate deploying these models against sensitive systems.



The Geniez AI safety layer

Alignment is necessary but not sufficient for mainframe deployment. The mainframe demands enforcement that does not depend on the model's goodwill. Geniez adds four independent layers on top of any LLM:

- **Permissions inheritance.** Every prompt executes against the mainframe using the requesting user's own RACF, ACF2, or Top Secret credentials. The model can see and do exactly what that human user can see and do - no more. There is no service account with elevated authority quietly granting the LLM access beyond its operator.
- **Running non-authorized.** The Mainframe Databot runs as a standard userid with no escalated permissions. It is not APF-authorized, it is not UID(0). This greatly reduces the potential attack surface of an AI-agent.
- **Mandatory read-only mode.** Even if the requesting user has authority to modify resources, Geniez can enforce read-only access for LLM-initiated operations. A security review cannot accidentally — or through prompt injection — alter a single dataset, profile, or member. The blast radius is bounded at the protocol layer, not by trust in the model.
- **Detailed audit trail.** The Geniez AI Framework records every interaction in granular detail: who issued the request, the exact userid used to access the mainframe, the timestamp, the full prompt submitted to the model, the model's complete response, every resource that was read (datasets, profiles, parmlib members, exits, started tasks), the security manager calls made on the user's behalf, and the result of each call. Nothing happens through Geniez that is not logged. The audit trail is structured, queryable, and exportable to your existing SIEM — giving auditors and incident responders the same depth of evidence they expect from human reviewers, with far greater consistency.

Defense in depth

Frontier-lab alignment makes the model unwilling to do harmful things. Geniez's permissions inheritance and read-only enforcement make it unable to. Both layers fail closed: a model that misbehaves cannot exceed its user's authority, and even within that authority it cannot write.

Putting It Together: STIG Reviews with Geniez

In a mainframe environment in which Geniez AI framework is installed, a STIG review changes shape. The security team does not produce a list of pass/fail items once a quarter. They operate a continuous reasoning loop:

- The model reads the current state of the environment — RACF, ACF2, or Top Secret rules, SETROPTS or equivalent, parmlib members, started task definitions, JCL procedures, exits, and APF library contents — through Geniez's read-only interface.
- It evaluates that state against the relevant STIG version, against general mainframe security best practice, and against the site-specific patterns it has learned about the environment.
- It produces findings with severity, evidence, the affected components, the STIG item or non-checklist principle that applies, and a proposed remediation. Where the underlying issue spans multiple components — the kind of finding a checklist cannot express — it explains the chain.
- Findings flow into the security team's existing workflow for human review and approval. Remediation remains a human-authorized action. Geniez never writes; humans do.
- As new STIG versions, new advisories, or new attack patterns become known, the model incorporates them without a vendor content update. The capability improves as the underlying model improves.

What this means practically

For STIG reviewers, the deliverable is no longer a checklist with a backlog of investigations behind it. It is a curated set of findings — checklist items and non-checklist exposures together — each with the context needed to triage and act. For CISOs, it is the first time mainframe security can credibly claim continuous coverage. For auditors, it is a defensible, logged record of review activity at a depth that manual processes cannot reach.

Conclusion

STIG checklists are not wrong; they are bounded. They describe what experienced engineers already know to look for. Generative AI extends review into the territory the checklist cannot describe: the site-specific patterns, the multi-component interactions, the long tail of decisions whose original justification has been forgotten. On the mainframe — older, more accreted, less recently scrutinized than almost any other platform — that extension is where the largest unaddressed risk lives.

The capability is real today. Frontier models have already found vulnerabilities, at scale, in code that resisted decades of expert review. The same approach applies directly to z/OS, RACF, ACF2, and Top Secret. The remaining question is how to deploy it safely against systems that cannot tolerate accidents. Geniez answers that question with permission inheritance, mandatory read-only enforcement, and an architecture that improves automatically as the underlying models improve.

From checklist to context is not a slogan. It is a description of where mainframe security review is going, and how quickly. The institutions that adopt this approach early will close exposure that has been quietly accumulating for forty years. Those that do not will keep finding the same things with the same tools — and missing the same gaps.

About Geniez AI

Geniez connects any LLM to the mainframe. Whatever model your organization trusts — today's frontier or tomorrow's — Geniez is the interface that lets it read z/OS, RACF, ACF2, and Top Secret safely, with permissions inherited from each user and read-only enforcement guaranteed at the protocol layer. As frontier models improve at finding vulnerabilities, your mainframe security improves with them. Contact contact@geniez.ai for more details.

geniez.ai