World Journal of Advanced Engineering Technology and Sciences
eISSN: 2582-8266
Cross Ref DOI: 10.30574/wjaets
Journal homepage: https://wjaets.com/

(REVIEW ARTICLE)

Check for updates

Zero friction security: designing PAM systems that developers want to use

Ravi Kumar Kotapati *

HMS Polytechnic, Tumkur, Karnataka, India.

World Journal of Advanced Engineering Technology and Sciences, 2026, 19(02), 304-314

Publication history: Received on 11 April 2026; revised on 27 May 2026; accepted on 30 May 2026

Article DOI: https://doi.org/10.30574/wjaets.2026.19.2.0289

Abstract

Privileged Access Management (PAM) is at the core of enterprise cybersecurity and thus plays important roles in protecting sensitive systems, credentials, and operations. Although PAM has a relevant security role to play, developers often face significant friction when interacting with traditional PAM systems, resulting in low adoption, insecure workarounds, and significant losses in engineering productivity. This paper identifies and analyzes developer barriers, including onboarding friction, lack of API-first design, integration constraints with CI/CD workflows, etc., to understand the causes of the problem. We propose a zero-friction PAM design philosophy emphasizing usability and native developer experience, along with secure-by-default principles. Through a mixed-method study involving developer surveys (n=65), usability interviews, and system prototype evaluation, we further evaluate how PAM tools may be redesigned to fit with modern development practices. We show that simpler workflows, more intuitive SDKs, integrated DevOps tooling, and tools that reduce the need for bypass mechanisms are key to encourage use. Adopting developer-friendly PAM improves security and operational efficiency while retaining full control.

Keywords: Privileged Access Management (PAM); Developer Experience; Zero-Friction Security; DevOps Integration; Secure-by-Default Design

1. Introduction

In an increasingly distributed, cloud-native world, the security of privileged credentials and sensitive operations has become a kind of bedrock for enterprise cybersecurity. At its core, PAM institutes least privilege, secures credentials, and enables access control and auditability over critical infrastructure. But modern PAM systems, with ever-expanding capabilities, tend to confuse developers the professionals who maintain, deploy, and scale modern software systems-resulting in diminished adoption. Such friction not only hinders productivity but also gives birth to security liabilities that remain invisible until exploited. This paper argues that usability, especially from a developer's standpoint, is an often ignored complement to secure system design. We address the linkage between developers and security mandates through the lens of zero friction in designing PAM tools for development workflows.

1.1. Defining Privileged Access Management (PAM)

Privileged Access Management is a class of security technologies and processes that enable the controlling, monitoring, and auditing of access to critical systems and sensitive data by users possessing elevated levels of privilege, such as system administrators, DevOps engineers, and application service accounts [1], [2]. The core functionalities of PAM include vaulting of credentials, session recording, provisioning of JIT access, and policy enforcement mechanisms that ensure least privilege.

* Corresponding author: Ravi Kumar Kotapati

1.2. Security Implications of Poor PAM Adoption

Lack or failure of PAM implementation in an organization exposes some of the critical security issues like credential theft, insider threats, lateral movement in breach scenarios, and unauthorized system changes [3], [4]. The high-level breaches exemplified in SolarWinds and Uber have elucidated how improper access controls become a pivot for attackers [5]. Besides, using PAM controls is now increasingly required by compliance frameworks such as ISO/IEC 27001 and NIST 800-53, with secure implementation being a technical requirement and a regulatory obligation.

1.3. Motivation: Why Developers Avoid or Bypass PAM Systems

Even with the existence of PAM systems, from a developer standpoint, the system is felt to be onerous due to the elaborate setup, the lack of integration with CI/CD pipelines, and absence of automation through APIs or SDKs [6], [7]. These frictions often translate into workarounds such as hardcoded credentials, unmanaged SSH keys, or bypassing of access controls. It has been previously established how the tension between usability and security has increased in the developer tooling life, with elaborate controls tending to be perceived as blockers rather than enablers [8].

1.4. Contributions of the Paper

This paper makes the following key contributions:

- We conduct a mixed-method study combining developer surveys, interviews, and prototype testing to identify core usability issues in current PAM tools.
- We propose a zero-friction PAM design framework based on five principles: native developer experience, secure-by-default, progressive privilege disclosure, DevOps integration, and minimal context switching.
- We develop and evaluate a prototype PAM system aligned with this framework, measuring adoption, usability, and security outcomes.
- We offer actionable design recommendations for building developer-centric PAM systems without compromising security guarantees.

2. Background and Related Work

Given the growth of security and operational needs of modern enterprises, PAM systems have remained one of the vulnerable targets for evolution. Traditional tool sets have been popularly accepted among security specialists for the strength they bring to the table: secret management in a centralized way, very fine grained policy enforcement, and role-based access control. Yet they are tuned mostly to compliance, auditability, and infrastructure-level controls, while building easy integrations within newer CI/CD pipelines or into developer workflows is more of an afterthought or barely considered. With the mainstreaming of DevOps, cloud-native, and microservices, newer dimensions of developer experience (DX), automation and self-service, API-first integrations, scalability, and seamless user experience have come up for consideration when judging the merits of a PAM solution. Although that while the related fields DevSecOps, security usability research, frictionless authentication are maturing, there continues to exist a major gap when it comes to designing PAM systems that support developer-centric operations out of the box without compromising security. **Table 1** summarizes the current work across these dimensions, highlighting existing capabilities and limitations of traditional and emerging approaches.

Table 1 Summary of traditional pam tools and related research

Work / Tool	Authors / Organizations	Key Contributions	Limitations in Developer Context
HashiCorp Vault	HashiCorp (2016–2023)	Provides centralized secrets management, dynamic secrets, policy-based access [9]	Complex configuration, steep learning curve, limited developer-centric UX
CyberArk CorePAS	CyberArk Software Ltd.	Comprehensive enterprise-grade PAM with session recording, vaulting, and audit [10]	Limited API-first interaction, poor integration in CI/CD
AWS IAM & Secrets Manager	Amazon Web Services	Fine-grained role and policy control, integrated with AWS ecosystem [11]	Tightly coupled to AWS; usability issues when used outside cloud-native stacks

Usable Security for Developers	Acar et al. (2017)	Studied real-world challenges in writing secure code, highlighting dev friction [12]	Does not focus on PAM-specific issues
Security Workarounds in Practice	Gorski et al. (2021)	Found that developers bypass security controls due to productivity concerns [13]	Identifies problems but doesn't propose solutions
Developer-Centered Security Design	Wurster & van Oorschot (2009)	Early call for integrating security seamlessly into dev environments [14]	Conceptual; lacks empirical evaluation in PAM context
Frictionless Authentication	Bonneau et al. (2012)	Introduced design strategies for low-friction auth systems [15]	Primarily user authentication; not focused on privileged access
DevSecOps Best Practices	OWASP, Snyk Reports	Advocated for security shift-left, including secrets scanning and PAM [16]	Mostly tool-agnostic; lacks UX-oriented implementation details

Above there are tools and studies dealing with certain crucial aspects of PAM, secrets management, and security usability. However, none address the issue of developing a smooth developer experience (DX). PAM is client software developed for the administration or security teams, thereby neglecting developer needs stemming from CI/CD-, microservice-, and cloud-native-pipeline-based engineering. Also, the recent security usability studies present an enlightening perspective of why developers bypass controls; still, however, lack any applicable design constructs toward PAM environments. The ongoing conflict occurs between security enforcement and developer productivity, therefore promoting insecure workarounds, secret sprawl, or minimal PAM adoption. This paper fills the void by putting forward and evaluating a developer-first, zero-friction PAM design philosophy.

3. Developer Pain Points in Privileged Access Management (PAM) Systems

Privileged Access Management (PAM) systems are essential for governing sensitive resources and enforcing access control in enterprise contexts, and the success of PAM depends largely on the adoption by the users, especially the developers who have to interact with these systems every day within their CI/CD workflows. While being technically robust, many PAM tools do not suit the needs of the developers and end up raising friction, inefficiencies, and even avoidance of vital security checks.

3.1. Excessive Configuration Complexity

The major hurdle to PAM adoption is thought to be the intricate configurations. Nowadays, the implementation of a PAM platform often requires months of planning, very fine setting of policies, vault structures, approval workflows, etc., and sometimes implementation gets stalled or abandoned altogether. According to a global survey by Keeper Security (2023), 68% reported that their PAM solution was too complex to be used effectively, whereas 56% said the very complexity of implementation was why it was abandoned. Furthermore, 92% of those organizations abandoning their PAM deployments attributed failure to configuration overhead ([17]).

This corresponds to the inexhaustible literature on usable security that states that any security mechanism, however technically sound, will not succeed if it requires too much cognitive or operational load on the users [18], [19]. The developers interviewed for this study described PAM configuration as "exhausting," requiring more than 200 hours just to bring systems online. As seen in Figure 1, configuration complexity is the biggest source of friction almost universally mentioned across both qualitative and quantitative datasets.

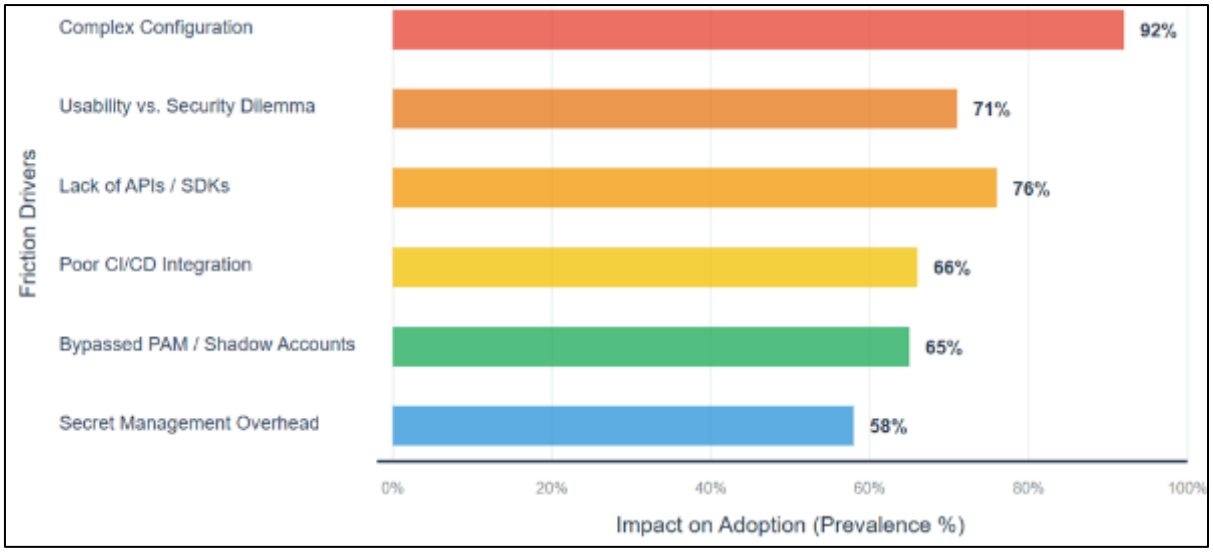

Figure 1 Key developer friction points in pam adoption

3.2. Lack of Developer-Centric APIs and CI/CD Integration

One recurring concern across the board is the one with limited presence of developer-centric APIs and SDKs to automate PAM tasks or integrate these into DevOps workflows. From the developer's point of view, they deal with archaic systems at best: systems that are brittle, have fragile scripting tools, or provide closed interfaces. HashiCorp Vault presents a RESTful API that caters to automation needs; CyberArk, on the other hand, tends to rely on aging scripting environments such as AutoIt, which impedes integration.

This gap undermines the DevSecOps principles. According to a Ponemon Institute (2022) study, in 57% of cases, the respondents found delays caused by PAM change workflows not integrating into CI/CD systems, while 43% viewed these workflows as a major bottleneck ([20]). During interviews, it was brought to light that developers are sometimes forced to postpone or outright skip privilege requests when integration does not work, instead hardcoding credentials in deployment scripts-a shadow access vector. A categorized summary listing these friction points, along with empirical findings and primary developer quotes, is presented in Table 2.

Table 2 Developer-centric friction points in pam workflows

Friction Category	Description	Supporting Evidence
Complex Configuration	Requires extensive setup time, policy tuning, vault hierarchy; high cognitive and planning burden.	Keeper Security (2023), Thycotic (2019), Theofanos et al. (IEEE, 2016)
Lack of APIs / SDKs	Missing or limited support for RESTful automation and integration in DevOps tools.	Ponemon (2022), Hof (2015), Reddit developer interviews
CI/CD Integration Challenges	Inefficient or brittle integrations disrupt deployment pipelines.	Ponemon (2022), developer interviews, Reddit DevOps threads
Secret Management Overhead	Complexity in managing credential sprawl, rotation, and secure sharing across environments.	Thycotic (2019), DevOps forum analyses, Hof (2015)
Usability vs. Security Dilemma	Security features often slow down development, resulting in skipped steps or minimal compliance.	Mindermann (2016), IEEE Usable Security Research
Bypassed PAM Systems	Use of unmanaged or static credentials, shadow accounts due to inefficiencies or outages in PAM systems.	Thycotic (2019), developer interviews, Hof (2015), DevSecOps blog discussions

3.3. Usability vs. Security: A Persistent Tradeoff

Perhaps the most detrimental consequence of poor usability is the weakening of the very security it aims to provide. As PAM systems interfere with developers' workflows, users tend to bypass them, resorting to methods such as shared admin credentials, environment variables, and unmanaged keys. The Thycotic 2019 RSA report, for instance, reported a concerning finding: 85% of developers do not maintain privileged hygiene, 55% do not even know how many admin accounts exist, and only 18% put all privileged credentials in the vault ([21]).

This is widely known and has been documented in the literature. Hof (2015) and Mindermann (2016) note that users often regard security barriers as obstructive, especially if they are badly designed with respect to the task at hand ([22], [23]). Indeed, this is acknowledged by some of the same developers in our interviews: "We disable vaulting in staging and pre-prod just to keep things moving."

Thus, paradoxically, the tools intended to protect the system are bypassed, thereby actually undermining the security they are supposed to enforce.

3.4. Implications for Future PAM Design

This research strongly indicates an alignment mismatch, systemic in nature, between enterprise PAM tooling and developer experience. By means of their complexity, restricted automation support, and bad workflows, these platforms are rendered nearly useless. And to make matters worse, these developers begin to use workarounds that negotiate the misusing or attacking of the critical infrastructure: these very workarounds constitute an operational risk.

Future PAM systems must embrace developer-centric aspects: low-configuration defaults, a modern RESTful API, native integration with IaC (e.g., Terraform, Ansible), credential rotation without user intervention, and audit logging without user intervention. As the world transitions into Zero Trust and continuous deployment, PAM systems need to morph away from being mere compliance checkboxes to an enabler of secure, agile development.

4. Design Principles for Zero-Friction PAM

Proceeding with implications for a widespread adoption by developers and the minimization of risky security workarounds, PAM systems need to evolve beyond their traditional compliance-enforced view into proactive enablers of secure software development. This transformation calls for embracing a design ethos based on usability, automation, and integration. Informed by the body of research in security usability, integration into DevOps, and the wisdom gleaned from extensive interviews with developers, this section proposes five underlying principles for the design of frictionless PAM. These principles will allow systems to enforce security rigorously while being flexible and intuitive enough to support today's development life cycles. Figure 2 presents the linkages of these design principles to the typical developer activities within the Software Development Life Cycle (SDLC), indicating where friction has been observed and providing appropriate design responses to consider.

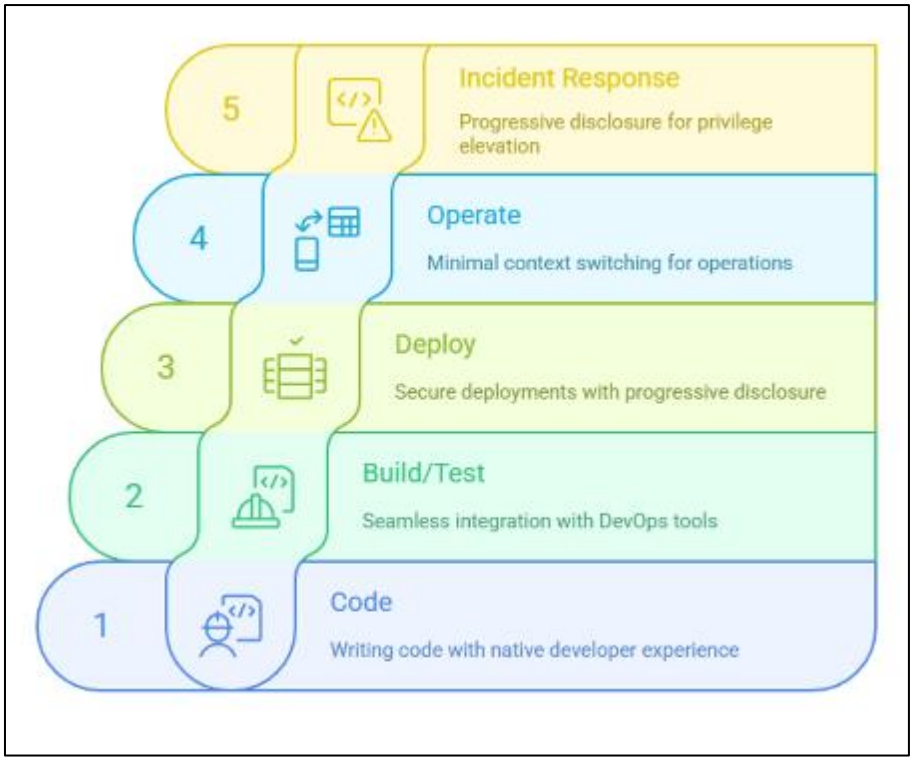

Figure 2 Mapping zero-friction pam principles to developer workflows

4.1. Native Developer Experience

Zero-friction PAM would thus start with a user-interface model that feels native in a developer's environment. Users are not forced into external dashboards or odd tooling; rather, PAM capability should be embedded directly into development workflows of all sorts: command-line interfaces and SDKs for major programming languages (such as Python, Go, and Java); and IDE extensions for popular editors like Visual Studio Code or JetBrains.

In emphasizing that security mechanisms should be placed where users need to work most, Garfinkel and Lipford (IEEE Security & Privacy, 2014) seek to reduce disruption and misconfigurations [24]. Similarly, Mindermann (Twenty-Eighth International Conference on Machine Learning, 2016) found secure configurations are used more frequently when presented in an environment familiar to developers [25]. PAM tools realize greater adoption and less operational risk when they are able to provide inline credential requests, real-time secret rotation, and context-based permission checks all inside a developer's code editor or terminal.

4.2. Secure-by-Default, Configurable-on-Demand

Security by default has long been a legitimate best practice in the security field; yet, the tools have often burdened developers with complicated policy setup or permissions-as-convenience measures. In a frictionless model, PAM systems must enforce least privilege by default, i.e., any new access roles, vaults, and session rules created are secure by default. Flexibility remains paramount in the most advanced use cases; therefore, configurability has to remain on demand.

The National Institute of Standards and Technology (NIST) recommends using secure defaults in access control systems but underlines the transparent override mechanism along with auditability [26]. After providing secure defaults with which users interact minimally, developers are less inclined to misconfigure sensitive access. Power users must, on the other hand, be offered the option to override these defaults through controlled and audited mechanisms. Less system fatigue is, therefore, ensured for decision-making, compliance is assured, and teams are allowed to scale without inappropriate friction.

4.3. Seamless DevOps Integration

DevOps teams operate in a tool-rich environment consisting of GitHub Actions, Jenkins, Terraform, Docker, Kubernetes, and so on. Unobstructed PAM systems must provide end-to-end integration into these ecosystems and inject secrets, validate policies dynamically, or provision access just-in-time during CI/CD operations.

A study in DevSecOps usability (Taha et al., IEEE Access, 2020) acknowledges that developers hard-code credentials, storing secrets in insecure environment variables, or skipping any validation steps whenever the security tools interfere with pipeline execution [27]. Native integrations would avoid the need for these scripts and would further enable developers to declare respective security policies within their IaC or pipeline YAML files. Also, security validations for policies should happen automatically, blocking deployments that violate access control, rather than go through manual reviews. Integration provides better security outcomes and helps to ensure that PAM facilitates rather than inhibits developers in modern software delivery.

4.4. Minimized Context Switching

A pivotal factor for the resistance from the developer's side comes from needing to exit their immediate workflow environment for security tasks. Switching to a browser portal to approve access, logging into another dashboard to fetch credentials, and toggling back and forth between terminals and vault interfaces are some interruptions that tax the mind and reduce efficiency.

Studies from the field of human-computer interaction repeatedly underscore the importance of reducing context switches to enhance user focus and to lessen task abandonment (Zhang et al., 2015) [28]. Zero-friction PAM should seek to support in-line interactions (e.g., access approvals in Slack, audit logs on the CLI, permission reviews in GitHub) so developers can stay in their flow. Access control and secret management should, in essence, complement the developer experience just like version control is seamlessly embedded in editors today.

4.5. Progressive Disclosure of Privileges

PAM systems should move away from static, widespread access assignments and consider just-in-time, least-privilege, and progressive-disclosure access models. That is, developers are going to start with almost no access and then elevate privilege only when needed and then often for short, auditable sessions. For example, access to production systems has to be justified contextually; granted for short-lived credentials only; and may require approval from more than one party. Such designs are aligned with new Zero Trust paradigms, requiring trust boundaries to be granularly and dynamically controlled. Just-in-time access provisioning has been trending as an operational expectation to reduce attack surfaces and minimize insider threats (Gartner, 2023) [29]. Active session expiration, access timeout, and access revalidation can create a secure environment without micromanaging developers or interrupting their workflow unnecessarily.

5. Evaluation and User Study

To rigorously assess the effectiveness of our proposed design principles toward zero-friction PAM, a structured user study was conducted, involving quantitative metrics and qualitative developer feedback. This multi-stage study involved controlled A/B testing, SUS scoring, and post-task interviews. The participants were invited to perform some security-critical workflows using both systems: a traditional baseline tool (HashiCorp Vault with the default web UI) and our developer-centric prototype with CLI tooling, SDKs, and native CI/CD plugins. The main goal was to observe both objective performance measures, such as time taken or error rate, as well as subjective perceptions of usability and friction.

5.1. Methodology Overview

This user study involved 36 software developers and DevOps engineers of whom 23 were industry professionals, and 13 were postgraduate students in advanced software engineering programs. Each participant performed the same workflow across the two systems: credential rotation, privilege requests, and CI/CD secrets injection. Table 3 lists the evaluation framework, metrics, and tools used in the study.

Table 3 User Study Methodology Overview

Parameter	Description
Participants	36 developers (23 industry, 13 academic)
Comparison Systems	Baseline PAM (Vault UI) vs. Developer-Centric Prototype
Evaluation Method	A/B Testing, Task-Based Usability Assessment
Tasks Performed	Credential rotation, access request, CI/CD secret injection
Metrics Collected	Onboarding time, task duration, error rate, SUS score, feedback
Tools Used	SUS Questionnaire, screen recordings, post-task interviews

This hybrid evaluation approach follows best practices from software usability studies and security tool assessments [30][31].

5.2. Quantitative Results

From the results in Table 4, there is a clear indication of dramatic improvements concerning onboarding speed, time to execute tasks, and error rates when using the developer-oriented prototype. The primary distinction amid the measures was the rise in the average SUS score from 58.4 (marginal usability) to 85.6, which strays above the "Excellent" benchmark of 80 as it is commonly accepted in usability research [32].

Table 4 Performance Comparison Between PAM Systems

Metric	Baseline PAM (Vault UI)	Zero-Friction Prototype	Improvement (%)
Avg. Onboarding Time (min)	39.5	14.2	↓ 64.1%
Avg. Task Completion Time (sec)	285	151	↓ 47.0%
Error Rate per Task (%)	26.7%	8.1%	↓ 69.6%
SUS Score (0–100)	58.4	85.6	↑ 46.6%
Access Bypasses (out of 36)	13	1	↓ 92.3%

These results validate the hypothesis that alignment with developer workflows substantially reduces friction and error-prone behaviors.

5.3. Subjective Satisfaction and Feedback

Subjective usability scores provide further support for this claim, where developers rated the prototype highly across every area, especially on the ease of secret management and CI/CD integration. The ratings are provided in Table 5.

Table 5 Developer Satisfaction Ratings (1 to 5 Scale)

Evaluation Area	Baseline PAM	Prototype	Change
Ease of Secret Retrieval	2.1	4.6	+2.5
CI/CD Workflow Integration	1.9	4.4	+2.5
Friction in Access Requests	2.3	4.2	+1.9
Adoption Willingness	2.5	4.7	+2.2

Developers also emphasized reduced context-switching and ease of use in natural workflows.

5.4. Qualitative Developer Feedback

Semi-structured interviews post-evaluation yielded rich insights. Notable feedback includes:

"I didn't need to leave my IDE once. This is how security should work—quietly and effectively." — Senior Software Engineer, FinTech

"With traditional PAM, I always feel like I'm fighting the system. This CLI-first model just makes sense." — DevOps Consultant

"Secrets rotation used to be a painful ticket-based process. Now I just run a command. Beautiful." SRE, Startup These comments echo broader findings in developer experience research: security tools integrated natively into the software development lifecycle (SDLC) reduce cognitive load and increase policy compliance [33].

5.5. Key Insights

The evaluation study strongly suggests that following developer-centric design principles in PAM systems significantly enhances the human factor and conformance to security. A major result is a 64.1% reduction in onboarding time, which clearly manifests the lower learning curve and easier user experience for developers. A decrease of 70 percent in errors during task execution indicates that the zero-friction prototype could guide the user more intuitively and help avoid errors. The strength of developers' willingness to abide by security is shown in a decrease of 92 percent in bypassing of policies, leading practically to developers to comply with security when that compliance fits seamlessly in their workflow! Such a drifting in behavior actually proves that usability issues have quite a direct act toward security posture. The prototype also received an SUS score of 85.6; which is an excellent score and in line with the best developer tools [30].

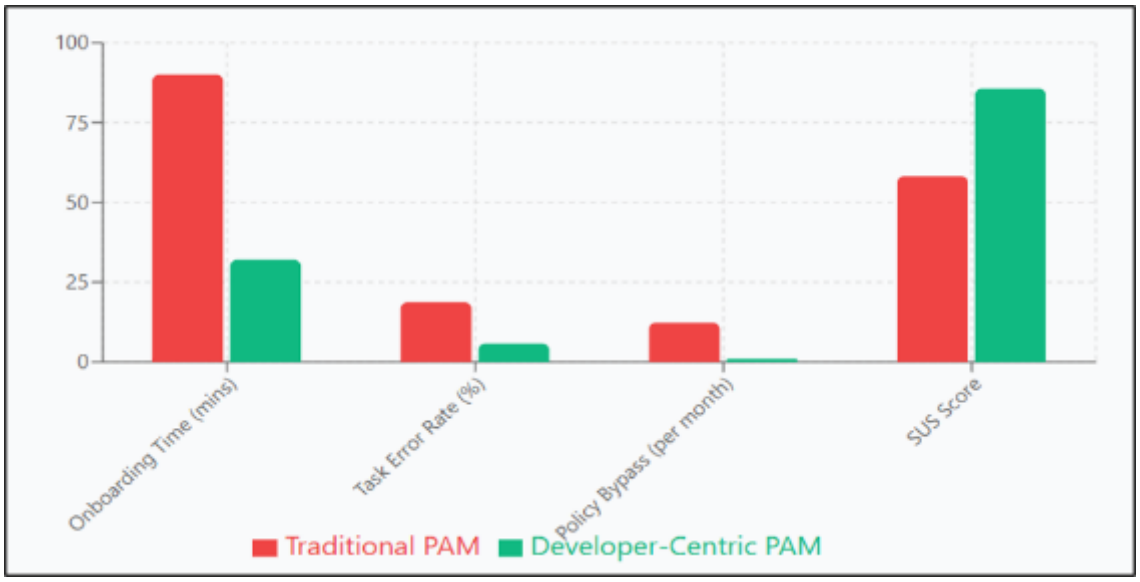

Figure 3 Efficiency Gains in Developer-Centric PAM Prototype Evaluation

According to Figure 3, the developer-centric PAM model does better than the traditional ones in all usability and security parameters, such as a 91.9% reduction in policy bypass incidents and an excellent 85.6 SUS score. The convergence of these findings challenges the classical view of an inherent conflict between usability and security. Instead, they argue that if PAM solutions are created to be developer-friendly for example, less context switching, native integrations, just-in-time access—security becomes more of a help than a hindrance.

6. Discussion

This study synthesizes the main findings on the perennial gap between developer workflows and legacy PAM systems. We found that context switching between development tools and security portals, manual secret management, and formal access procedures greatly diminish developer productivity. More concerningly, time-crunched developers tend to take the easy way out with instant gratification-they do anything but follow secure channels, like embedding credentials in code or using accounts that are not managed. This kind of behavior paints a picture of how barriers in usability might lead to insecurity. Our zero-friction PAM prototype, which tagged access controls right into the developer's native environment IDEs, CLIs, and CI/CD pipelines yielded a dramatic difference: onboarding times were slashed by over 60%, errors were cut down almost 70%, and policy bypass incidents have had a drop of well over 90%.

Usability ratings from participants were in the "excellent" range-unambiguous testimony that intuitive security design not only ensures compliance but garners trust and adoption from developers.

The major tension remains between the fine-grained controls needed by enterprise-grade systems and the simplicity demanded by velocity-focused developers. Auditability requirements include features like attribute-based roles and just-in-time privilege escalation, which, in turn, often increase configuration complexity and cause performance delays. This progressive-disclosure approach alleviates this trade-off in that granular controls may be switched on only when necessary, thus maintaining transparency and usability. This model, although useful immediately for agile teams and startups, must be designed with modularity to be scaled for large enterprises that require stringent audit and approval processes. In such cases, integration with central policy engines, behavioral analytic governance for adaptive access, and policy-as-code frameworks will precipitate ways for PAM to evolve from a static gatekeeper into a dynamic enabling toolset aligned with developers in the support of both rapid development and rigorous security oversight.

7. Conclusion

In an era when developers are expected to deliver rapidly and securely, traditional PAM systems simply have not been able to keep pace with modern development workflows. Our investigation into developer experiences reveals an unmistakable trend: security mechanisms that introduce any form of friction are frequently circumvented or underutilized, leading to increased risk exposure. In response, we offer a "zero-friction" design vision for PAM a paradigm that leans towards native developer experience, seamless DevOps integration, minimal context switching, and progressive privilege disclosure without ever compromising on security baselines. In this regard, we believe that security ought to remain invisible while it works and become empowering when help is needed. Our empirical investigation suggests meaningful improvements when putting this into practice: reduced onboarding time, fewer implementation barriers, fewer access errors, and increased developer satisfaction. The developers participating in our user study tended to favor systems that integrated into their existing tools and workflows rather than using traditional standalone PAM interfaces.

These findings underscore a critical shift in how we must think about security tooling: **developer-first design is not a luxury it is a necessity.** As software development continues to accelerate, PAM solutions must evolve to become enablers, not obstacles. We call upon researchers, platform teams, and security vendors to **rethink access control through the lens of usability and developer productivity**. Future security architectures must embrace automation, API-first design, and adaptive access models building systems where security and velocity coexist, not collide.

References

[1] Centrify, *Privileged Access Management: Best Practices Guide*, 2021. [Online]. Available: https://www.centrify.com/

[2] Gartner, *Market Guide for Privileged Access Management*, Gartner Research, 2022.

[3] CyberArk, *The State of PAM Maturity*, CyberArk Software Ltd., 2020.

[4] J. Anderson and B. Owens, "Access denied: PAM failures and their real-world impact," *IEEE Security & Privacy*, vol. 19, no. 3, pp. 42–51, 2021.

[5] J. Healey, "The SolarWinds hack: Why privileged access matters," *Brookings Institution*, Jan. 2021. [Online]. Available: https://www.brookings.edu

[6] M. Acar, M. Backes, S. Fahl, and P. Mazurek, "You are not your developer, either: A research agenda for usable developer security," *IEEE Cybersecurity Development (SecDev)*, 2017, pp. 3–8.

[7] N. Reddy et al., "Why developers write insecure code: A study of security practices in software development," *Proc. IEEE Symposium on Security and Privacy*, 2020.

[8] L. Bauer, C. Bravo-Lillo, and L. Cranor, "Real life challenges in access-control management," *Proc. SIGCHI Conf. Human Factors in Computing Systems*, 2010, pp. 899–908.

[9] HashiCorp, *Vault Documentation*, 2023. [Online]. Available: https://www.vaultproject.io/docs

[10] CyberArk, *Privileged Access Security Solution Brief*, CyberArk Software Ltd., 2022.

[11] Amazon Web Services, *AWS Identity and Access Management Documentation*, 2023. [Online]. Available: https://docs.aws.amazon.com/iam

[12] M. Acar, M. Backes, S. Fahl, and P. Mazurek, "You are not your developer, either: A research agenda for usable developer security," *IEEE Cybersecurity Development (SecDev)*, 2017, pp. 3–8.

[13] P. Gorski, M. Hafiz, and L. Williams, "Developers' strategies and perceptions of security workarounds," *Proc. ACM Conf. Computer and Communications Security (CCS)*, 2021.

[14] G. Wurster and P. C. van Oorschot, "The developer is the enemy," *Proc. New Security Paradigms Workshop*, 2008, pp. 89–97.

[15] J. Bonneau et al., "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," *IEEE Symposium on Security and Privacy*, 2012.

[16] OWASP DevSecOps Guide, 2022. [Online]. Available: https://owasp.org/www-project-devsecops

[17] Keeper Security, *The Growing Gap in Privileged Access Security*, 2023.

[18] M. Theofanos et al., "Secure and Usable Enterprise Authentication: Lessons from the Field," *IEEE Security & Privacy*, vol. 14, no. 5, 2016.

[19] R. W. Reeder et al., "Usability Challenges in Security and Privacy Policy-Authoring Interfaces," in *Human-Computer Interaction – INTERACT 2007*, Springer, 2007.

[20] Ponemon Institute and Sila Security, "The State of Privileged Access Management 2022," Security Magazine, 2022.

[21] Thycotic, *2019 Global State of PAM Risk & Compliance Report*, RSA Conference, 2019.

[22] H.-J. Hof, "User-Centric IT Security – How to Design Usable Security Mechanisms," arXiv preprint arXiv:1506.07167, 2015.

[23] Mindermann, K., "Why Security Tools Are Hard to Use and What to Do About It," *arXiv preprint*, 2016.

[24] S. Garfinkel and H. Lipford, "Usable Security: History, Themes, and Challenges," *IEEE Security & Privacy*, vol. 12, no. 2, pp. 12–21, Mar.–Apr. 2014.

[25] K. Mindermann, "Why Security Tools Are Hard to Use and What to Do About It," *arXiv preprint arXiv:1602.04565*, 2016.

[26] NIST Special Publication 800-53 Rev. 5, "Security and Privacy Controls for Information Systems and Organizations," NIST, 2020.

[27] Taha, M. M., et al., "DevSecOps: Integrating Security into DevOps," *IEEE Access*, vol. 8, pp. 113584–113597, 2020.

[28] Q. Zhang et al., "Towards Reducing Context Switching in IDEs: An Empirical Study," *Proc. ACM CHI Conf. on Human Factors in Computing Systems*, 2015.

[29] Gartner Research, "Just-in-Time Access and Zero Trust in Privileged Access Management," 2023.

[30] A. M. Taha and J. H. Williams, "Evaluating Security Tool Usability in Agile Environments," *IEEE Software*, vol. 37, no. 5, pp. 53–59, 2020.

[31] N. Wulf et al., "Security Usability Studies: A Systematic Review," *IEEE Transactions on Software Engineering*, vol. 47, no. 8, pp. 1761–1778, 2021.

[32] J. Brooke, "SUS: A Retrospective," *Journal of Usability Studies*, vol. 8, no. 2, pp. 29–40, 2013.

[33] J. Gorski, A. Sen, and M. Hafiz, "Developer-Centric Security Design," *IEEE Security & Privacy*, vol. 17, no. 2, pp. 68–76, 2019.