International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)
https://ijcjournal.org/index.php/InternationalJournalOfComputer/index

Reducing “Unknown Certificate” Risk at Scale: A Comparative Evaluation of TLS Certificate Discovery Architectures

Vasilii Turuntaev*

Independent Infrastructure Security Expert, Spokane, WA, 99216, USA
Email: poloz942@gmail.com

Abstract

TLS/SSL certificates are a foundational trust primitive in modern digital infrastructures, yet certificate-related outages persist despite the widespread adoption of monitoring tools and enterprise certificate lifecycle management platforms. This paper addresses this puzzle by (1) classifying and analyzing existing certificate monitoring and management approaches to identify their systematic architectural limitations; (2) designing an alternative automated monitoring method that targets these gaps by rethinking certificate discovery; and (3) evaluating its effectiveness relative to conventional approaches using controlled cyber-threat modeling in a virtualized environment. We develop a human-error-aware simulation that applies identical, time-indexed infrastructure change events to multiple monitoring architectures across legacy and cloud scenarios, enabling direct comparison of inventory completeness, omission risk, and discovery latency. The results show that configuration-bound monitoring approaches degrade with infrastructure scale and dynamism, whereas an auto-discovery model grounded in L7 load balancer and front-end state maintains substantially higher visibility and lower omission rates, particularly in cloud environments. These findings suggest that reducing “unknown certificate” risk requires architectural innovation in discovery mechanisms rather than incremental improvements to configuration discipline, with important implications for organizational risk management and national cybersecurity resilience.

Keywords: certificate lifecycle management; auto-discovery; cloud security; infrastructure monitoring; cybersecurity risk.

Received: 11/29/2025
Accepted: 1/29/2026
Published: 2/7/2026

** Corresponding author.*

1. Introduction

TLS/SSL certificates are a foundational trust mechanism in modern distributed systems, providing confidentiality, service authentication, and integrity of data in transit. They underpin public websites, internal enterprise services, APIs, and machine-to-machine communication. Their scale is substantial: in 2025, Let’s Encrypt alone serves over 700 million websites worldwide, issuing up to 10 million certificates per day [1]. Browser telemetry confirms this transition to ubiquitous encryption: HTTPS accounts for over 85% of page loads in major browsers, including Chromium-based clients, reflecting the near-universal enforcement of encrypted transport at the application layer Reference [2].

Concurrently, the infrastructure environment in which certificates are deployed has undergone significant transformation. The widespread adoption of cloud computing, microservice-based architectures, containerization, and Infrastructure as Code (IaC) practices has dramatically increased both the number of certificates in use and the complexity of their lifecycle management. Certificates are no longer tied to a small, stable set of long-lived hosts but instead proliferate across ephemeral virtual machines, containers, load balancers, and dynamically created environments. Regulatory and governance developments have further intensified this pressure. Under the CA/Browser Forum Baseline Requirements, the maximum validity period for publicly trusted TLS certificates has already been reduced to 398 days. Moreover, the SC-081 ballot adopted in 2025 mandates a phased reduction of certificate lifetimes to approximately 47 days by 2029 [3].

Failures in certificate lifecycle management have immediate and often severe operational consequences. An expired or misconfigured certificate typically results in service outages or significant degradation of availability due to failed TLS handshakes or client-side security errors. The prevalence of such incidents is well documented in industry-wide empirical studies. In a large-scale survey of approximately 1,200 organizations, 72% of respondents experienced at least one certificate-related outage in the past year, with 67% reporting monthly disruptions and 45% facing them weekly [4]. The cost of downtime is staggering: industry analysis pegs the cost at around $9,000 per minute of outage, which can translate to losses of several million dollars for a multi-hour outage caused by an expired certificate [5].

In response, a distinct market segment has emerged around certificate lifecycle management (CLM) and PKI platforms, offering centralized inventories, automated issuance and rotation, and policy-based governance. Existing solutions range from lightweight expiration monitors for predefined domains to enterprise-grade CLM/PKI platforms (such as those offered by DigiCert, Entrust, and Sectigo) providing centralized control and CA integration. Many tools incorporate auto-discovery mechanisms — such as network scanning, imports from CA and cloud provider APIs, and integrations with certificate transparency logs — to address incomplete visibility. However, these tools are often limited in scope, configuration-intensive, or poorly aligned with highly dynamic IaC-driven environments. A growing corpus of research underscores that while automation adoption reduces human error and improves compliance, it introduces architectural and governance complexities that remain insufficiently resolved in practice [6, 7, 8].

This tension motivates the central puzzle of this study: why do certificate-related incidents persist despite widespread adoption of specialized CLM tools and auto-discovery mechanisms, particularly in dynamic IaC-oriented infrastructures? What structural and technical limitations prevent existing approaches from reliably reducing TLS/SSL-related availability and security risks, and how can these limitations be addressed? To answer these questions, we (1) classify and analyze existing certificate monitoring and management approaches to identify systematic limitations; (2) design an alternative automated monitoring method, EmTool, targeting these gaps; and (3) evaluate its effectiveness relative to conventional approaches using controlled cyber-threat modeling in a virtualized environment.

2. Existing Approaches to TLS Certificate Monitoring and Management

The evolution of large-scale distributed infrastructures and the tightening of security governance requirements have led to the emergence of a diverse ecosystem of tools concerned with TLS certificate monitoring and lifecycle management. These tools can be grouped into three analytically distinct classes, which differ in architectural assumptions, scope of visibility, and reliance on automation: (1) lightweight monitoring services and utilities; (2) certificate monitoring functionality embedded in traditional infrastructure monitoring systems; and (3) enterprise-grade certificate lifecycle management (CLM) and machine identity platforms. Below, each class is examined with respect to its functional capabilities, typical deployment context, and systematic limitations.

2.1. Lightweight Monitoring Services and Utilities

The simplest and historically earliest approach to TLS certificate monitoring relies on lightweight services or utilities that periodically check the status of certificates for a predefined set of domain names. In this model, operators manually enumerate domains of interest, after which the tool establishes TLS connections to corresponding endpoints and extracts certificate metadata, most notably the expiration date. Alerts are triggered when the remaining validity period falls below a configured threshold. From an operational perspective, this approach offers clear advantages: minimal setup cost, no requirement for infrastructure deployment, and rapid adoption for monitoring publicly exposed services. Similar mechanisms are discussed in early operational studies of HTTPS deployment, where manual certificate tracking was common in small-scale environments [9].

However, the architectural premise of this class is explicit enumeration. A certificate is monitored only if its associated domain is known in advance and manually registered. Empirical studies of configuration drift and shadow infrastructure demonstrate that this assumption rarely holds in practice, even in moderately complex systems [10]. Temporary environments, internal APIs, ad hoc subdomains, and short-lived test deployments frequently escape manual inventories. Even research surveys focused on industrial or embedded networking contexts confirm that rudimentary checks cannot scale to dynamic environments where certificates are frequently created, deprecated, or reassigned without centralized tracking [11]. Consequently, despite their utility for targeted monitoring of selected public domains, these tools fail to provide a comprehensive inventory of certificates across heterogeneous infrastructures.

2.2. Certificate Monitoring in Traditional Infrastructure Monitoring Systems

A more structured approach integrates certificate monitoring into classical infrastructure monitoring frameworks such as Nagios, Zabbix, and Prometheus. In these systems, plugins or exporters extract certificate attributes (e.g., validity period, issuer, and subject) as part of routine service availability checks. The extracted data are represented as metrics and evaluated against alerting rules. Research into automation of certificate lifecycle tasks emphasizes that embedding certificate status checks within broader observability systems improves operational awareness and aligns TLS monitoring with other performance and reliability metrics [12].

Nevertheless, this class of solutions inherits a core limitation from lightweight tools: visibility is constrained by configuration. A certificate is monitored only if the corresponding endpoint is explicitly defined in the monitoring system. While some platforms support limited forms of auto-discovery (such as template-based checks or scanning predefined address ranges), these mechanisms require careful manual configuration and are typically secondary features. In IaC-oriented infrastructures, where services and endpoints are created and destroyed continuously, monitoring configurations frequently lag behind actual system state. Empirical research on configuration drift in cloud environments shows that such lag — caused by divergence between Infrastructure as Code definitions and runtime states — is a dominant source of monitoring blind spots, particularly for ephemeral or experimental resources [13]. Thus, while such integrations enhance operational efficiency for known assets, they do not resolve the core challenge of dynamic certificate inventory in ephemeral or IaC-driven environments.

2.3. Enterprise Certificate Lifecycle and Machine Identity Management Platforms

The third and most functionally comprehensive class consists of enterprise-oriented certificate lifecycle management and machine identity platforms. These systems aim to provide centralized inventories, automate the full certificate lifecycle (from request and issuance to deployment, rotation, and revocation) and enforce policy across hybrid and distributed infrastructures. This category includes both traditional CLM/PKI platforms and cloud- or DevOps-oriented enterprise systems integrated with orchestrators, load balancers, and cloud-native certificate services. A defining feature of this class is the extensive use of auto-discovery mechanisms. Platforms combine active network scanning, agent-based collection, imports from certificate authority logs, cloud provider APIs, and analysis of TLS termination points such as load balancers and reverse proxies. Some systems additionally leverage Certificate Transparency logs to detect certificates issued outside centralized processes [14].

At a conceptual level, enterprise platforms come closest to addressing the visibility problem identified in prior work on PKI mismanagement [9]. However, empirical and qualitative studies of enterprise security tooling reveal several persistent structural limitations. First, effective discovery remains contingent on deep integration with network inventories, orchestration services, and organizational metadata; without tight coupling to authoritative service registries, auto-discovery mechanisms produce incomplete inventories, especially in contexts with segmented networks, multiple cloud tenants, or transient workloads [15].Second, while automation protocols such as the Automatic Certificate Management Environment (ACME) provide standardized mechanisms for certificate issuance, their applicability is largely limited to automated CA interactions and does not, by itself, address cross-domain inventory and policy enforcement across heterogeneous certificate sources. Third, organizational factors remain critical. Enterprise platforms require disciplined onboarding of new environments, consistent ownership assignment, and continuous synchronization between architectural changes and platform configuration. Research on human and organizational dimensions of security consistently shows that such discipline is difficult to sustain over time, leading to persistent gaps even in technically advanced systems [16, 17].

Table 1: Classes of TLS Certificate Management Tools and Systematic Limitations

Tool class	Primary focus and strengths	Typical scope and deployment context	Systematic limitations
Lightweight monitoring services and utilities	Rapid deployment; low operational overhead; basic expiration alerts for public certificates	External-facing domains; small or medium infrastructures; auxiliary monitoring for critical services	Full dependence on manual domain enumeration; no systemic auto-discovery; no coverage of internal, ephemeral, or ad hoc services; high sensitivity to human error
Certificate monitoring in traditional infrastructure monitoring systems (Nagios/Zabbix/Prometheus)	Integration with existing monitoring dashboards and alerting; unified escalation workflows	Known services and endpoints; relatively stable infrastructures	Coverage depends entirely on explicit configuration; limited and manual auto-discovery; weak support for IaC-driven dynamics; human factors determine inventory completeness
Enterprise CLM and machine identity platforms	Centralized inventory; full lifecycle automation; multiple discovery channels; CA and cloud integrations	Large organizations; hybrid and multi-cloud infrastructures; formalized security governance	High configuration and integration complexity; incomplete visibility of temporary and high-churn environments; fragmented inventories across domains; strong dependence on sustained organizational discipline

3. EmTool as an Alternative Monitoring Model

The comparative analysis summarized in Table 1 reveals a consistent structural pattern across existing classes of TLS certificate monitoring and management tools. Lightweight utilities, classical infrastructure monitoring systems, and enterprise CLM platforms differ substantially in scope and sophistication, yet all rely (explicitly or implicitly) on incomplete discovery mechanisms and sustained manual coordination. In each class, certificate visibility ultimately depends on either explicit enumeration, configuration-bound monitoring targets, or complex integrations whose coverage must be continuously maintained. As infrastructure scale and dynamism increase, these assumptions systematically break down, producing unmanaged or “unknown” certificates even in environments with advanced tooling.

This observation motivates the development of an alternative automated monitoring method that rethinks certificate discovery as a first-order design problem rather than an auxiliary feature. he central premise of the proposed approach is that certificate inventories should not be derived from operator-curated domain lists or static monitoring configurations, but from infrastructure components that objectively reflect which services are actively terminating TLS traffic. To address these requirements, we design EmTool, an enterprise-oriented TLS/SSL and domain monitoring platform focused on automated discovery and continuous validation of certificates at scale. The design of EmTool is guided by four principles derived directly from the limitations identified in existing approaches:

- Real TLS termination points (most notably load balancers and web front ends) are treated as the primary source of truth for certificate discovery, rather than manually enumerated domains.
- Auto-discovery must operate uniformly across cloud, on-premises, virtualized, and bare-metal environments.
- The number of manual actions required to onboard and maintain coverage must be minimized, as organizational discipline has been shown to be difficult to sustain over time.
- Certificate validation must operate at the level of complete TLS chains and integrate seamlessly with existing monitoring stacks to preserve established operational workflows.

3.1. System Architecture

EmTool is implemented as a multi-component platform composed of thirteen microservices written in Go. Each microservice is responsible for a narrowly scoped function — data collection, auto-discovery, inventory normalization and storage, TLS chain validation, integration with external systems, and API exposure — communicating through internal APIs and message queues. This modular architecture enables horizontal scaling of discovery and validation components as the number of monitored endpoints and certificates grows. At the core of the platform is a TLS validation engine that actively probes discovered endpoints and reconstructs the full trust chain from the leaf certificate to the root. Validation includes checks of certificate validity periods, cryptographic algorithms, key lengths, hostname matching, and intermediate certificate correctness. By validating the entire chain rather than only the leaf certificate, EmTool detects misconfigurations that often remain latent until expiration or client-side enforcement failures occur.

The platform can be deployed either as a cloud-hosted service or on-premises within a customer's infrastructure, accommodating regulatory and data-locality constraints. To execute checks from appropriate network vantage points, EmTool uses lightweight agents (referred to as gates) deployed within customer environments (e.g., as Docker containers). Gates initiate TLS checks locally and transmit structured results to the central services, allowing discovery and validation to respect network segmentation and access boundaries.

3.2. Auto-Discovery via Load Balancers and Front-End Configurations

The defining feature of EmTool is its auto-discovery model, which is built around load balancers and front-end configurations as the most reliable indicators of which services are actually exposed and terminating TLS traffic.

In cloud environments, EmTool provides provider-specific integrations deployed via Infrastructure-as-Code tooling. Once installed in a cloud account, an auto-discovery add-on continuously queries cloud APIs to import load balancers, listeners, associated domains, and bound certificates, and then tracks configuration changes over time. Creation or removal of services, modification of listener configurations, and certificate updates are automatically propagated into the EmTool inventory without manual intervention. In this model, the load balancer effectively functions as a centralized registry of externally reachable services.

For legacy and on-premises environments, where no centralized cloud API exists, EmTool combines two mechanisms. First, gate agents deployed within relevant network segments perform localized discovery and validation of TLS endpoints. Second, EmTool includes a dedicated integration module for Nginx that subscribes to configuration change events. Upon each configuration update, the module parses virtual host definitions and TLS settings and transmits an updated set of domains and endpoints to EmTool. This approach is particularly important for bare-metal and virtualized systems, where front-end configurations remain the most faithful representation of actively served domains. By anchoring discovery to load balancers and front-end configurations, EmTool avoids reliance on arbitrary domain lists and instead derives its inventory from components that directly govern traffic flow. This design sharply reduces the number of operator actions required to maintain coverage and shifts omission risk from human enumeration to machine-level discovery fidelity.

3.3. Continuous Inventory Maintenance and Monitoring Integration

At the inventory level, EmTool maintains continuous synchronization with its discovery sources. Domain names are automatically added and removed based on DNS data and front-end configurations, while certificates enter or exit the inventory as they appear or disappear in the infrastructure. This eliminates the need for manual lifecycle bookkeeping and ensures that the inventory reflects the current operational state rather than an administratively maintained snapshot. Active TLS validation is performed continuously. While operators may specify individual hostnames explicitly, the dominant source of monitored endpoints is automated discovery. Validation results can be inspected directly within the EmTool interface or exported into existing monitoring systems such as Zabbix, Prometheus, or Dynatrace. In these integrations, EmTool acts as an external metric provider, supplying certificate and domain expiration data while preserving existing alerting logic and escalation procedures. This design choice lowers adoption barriers and minimizes disruption to established operational practices.

3.4. Formal Model of Discovery Errors

To compare EmTool with monitoring-based approaches analytically, we model certificate omission as the outcome of two distinct error-generating processes: human-mediated configuration actions and machine-mediated auto-discovery.

Let N denote the number of TLS-terminating services present in the infrastructure. For each service i, we define a binary variable O_i, where $O_i = 1$ indicates that the certificate is omitted from monitoring. In monitoring-centric approaches, each service introduction typically induces one or more manual actions. Let k_i denote the number of such actions for service i, and let p_h denote the probability that a single action results in an omission-relevant error. The probability of omission due to human error is therefore $1-(1-p_h)^{k_i}$, implying that expected omissions grow rapidly with infrastructure scale.

In EmTool, manual actions are concentrated at the level of integration onboarding rather than per-service enumeration. Let p_e denote the probability that automated discovery fails for a given service due to incomplete integration coverage or event-processing errors. Under stable integrations, the expected number of omissions scales approximately as $N \cdot p_e$, decoupling omission risk from the number of services added. This formalization captures the architectural distinction between the approaches: monitoring-based systems scale omission risk with the number of human actions, whereas EmTool shifts risk toward machine-level discovery fidelity. This model also motivates two testable hypotheses:

H1. *As infrastructure scale increases, the expected number of certificate omissions grows significantly more slowly under EmTool than under monitoring-based approaches.*

H2. *In cloud environments, a statistically significant fraction of certificates is discovered exclusively through load balancer and cloud API data, resulting in fewer omissions relative to monitoring-based approaches.*

4. Materials and Methods

To evaluate the hypotheses, we employ a human-error-aware virtual infrastructure simulation designed to compare TLS certificate monitoring approaches under controlled yet realistic conditions. The simulation explicitly models how architectural differences in certificate discovery translate into inventory completeness, discovery latency, and omission risk under infrastructure dynamism and imperfect execution. Rather than benchmarking specific commercial products, the study compares four architectural treatment groups corresponding to the classes identified in Table 1, using identical infrastructure traces and ground-truth certificate sets.

Infrastructure state includes services, domain names, TLS termination points, and certificates, as well as the relationships among them. Certificates are associated with one or more domains and are bound to specific termination points (e.g., web servers, reverse proxies, or load balancers). Infrastructure evolution is driven by a predefined sequence of events, including service creation and deletion, service reconfiguration, certificate issuance and renewal, and certificate expiration. The same event sequence is applied to all treatment groups.

Each treatment group is implemented as a distinct discovery model that determines how certificates enter and leave the monitoring inventory in response to infrastructure events. Table 2 summarizes how each treatment group is operationalized in the simulation.

Table 2: Formal Modeling of Treatment Groups in the Simulation

Dimension	Lightweight monitoring services and utilities	Traditional infrastructure monitoring systems	Enterprise CLM and machine identity platforms	EmTool (proposed)
Primary discovery condition	Domain explicitly presents in manual domain list	Service endpoint explicitly presents in monitoring configuration	Certificate located within an onboarded discovery scope	Certificate bound to an integrated TLS termination point
Formal inclusion rule	$D_c(t) = 1 \iff d \in L(t)$	$D_c(t) = 1 \iff$ $M_s(t)$ exists and is correct	$D_c(t) = 1 \iff c \in \cup_j E_j(t)$	$D_c(t) = 1 \iff \exists T : c \leftrightarrow T \land T$ integrated
Trigger for discovery	Manual list update	Configuration update in monitoring system	Periodic automated discovery within onboarded scopes	Event-driven updates from termination configuration
Human error	p_h per domain addition	p_h per configuration action	p_h per scope onboarding	p_h per termination integration
Machine-level discovery error	Not applicable	Not applicable	p_e per certificate within scope	p_e per certificate at termination
Scaling of omission risk with infrastructure size	Linear in number of domains	Superlinear in number of services and changes	Proportional to number of environments	Approximately constant once terminations integrated

All architectures are evaluated against identical synthetic infrastructures with complete ground-truth observability:

- ***Legacy scenario:*** the infrastructure consists of multiple network segments hosting web services, reverse proxies, and internal APIs. TLS termination occurs primarily at web servers and proxies. Certificates are issued by both internal and external CAs. Services are dynamically added, migrated, and retired.
- ***Cloud scenario:*** the infrastructure consists of cloud-managed L7 load balancers serving as the primary TLS termination points. Domains and certificates are bound to listeners and backend groups. The scenario includes multiple environments and frequent creation and teardown of services.

For each scenario, a time-indexed ground-truth set of certificates $N_{true}(t)$ and a sequence of infrastructure change events are generated. These events include service creation, deletion, and migration; domain and routing reconfiguration; creation or modification of TLS termination points; and certificate lifecycle operations such as issuance, renewal, replacement, and expiration. Each event deterministically updates the infrastructure state and, consequently, the set of certificates actively bound to TLS termination points at time t.

Against this evolving ground truth, each monitoring architecture produces a time-varying certificate inventory reflecting what is actually discovered and monitored. Comparing these inventories to the ground-truth set at each time step allows us to quantify discovery completeness, omission risk, and detection latency under identical infrastructure dynamics. Table 3 summarizes the evaluation metrics used for this comparison.

Table 3: Evaluation Metrics

Metric	Symbol/Formula	Definition and interpretation
Inventory coverage	$coverage = \frac{N_{found}}{N_{true}}$	Fraction of certificates present in the ground-truth infrastructure that are successfully discovered and included in the monitoring inventory. Values closer to 1 indicate more complete visibility.
Number of missed certificates	$N_{missed} = N_{true} - N_{found}$	Absolute number of certificates that exist in the infrastructure but are not discovered by the monitoring approach.
Missed certificate ratio	$missed\_ratio = \frac{N_{missed}}{N_{true}}$	Proportion of certificates omitted from monitoring relative to the total number of certificates present in the infrastructure.
Time to discovery	T_{disc}	Time elapsed between the moment a certificate appears in the infrastructure (e.g., issuance or binding to a TLS termination point) and the moment it is first detected by the monitoring system. Reported as mean and distribution across simulation runs.
Sensitivity to human error	$N_{missed}(p_h)$	Dependence of inventory coverage and number of missed certificates on the probability of human error. Used to compare how rapidly omission risk increases as human reliability decreases across treatment groups.
Cloud-specific discovery contribution	$\alpha_{LB} = \frac{N_{LB-only}}{N_{true}}$	Fraction of certificates discovered exclusively via load balancer configurations and cloud provider APIs. These certificates would remain undiscovered under monitoring-based approaches for the same p_h

5. Results

Across both simulated infrastructure scenarios, the results provide strong and consistent support for both hypotheses. Figure 1 illustrates inventory coverage as a function of infrastructure scale for all four architectural treatment groups under legacy and cloud conditions. Two systematic patterns emerge. First, as infrastructure scale increases, monitoring-based architectures exhibit a pronounced degradation in certificate inventory coverage, whereas EmTool maintains near-complete visibility even under high churn. In the legacy scenario, lightweight monitoring utilities decline from approximately 81% coverage at small scale to roughly 62% at large scale, while traditional infrastructure monitoring drops from about 89% to 76%. Enterprise CLM platforms perform substantially better but still exhibit gradual erosion of coverage as scale increases, decreasing from approximately

94% to 91%. In contrast, EmTool remains stable and close to complete coverage across all scales, increasing slightly from about 97% at small scale to nearly 99% at large scale. This divergence widens monotonically with scale, indicating that omission risk compounds with the number of configuration-driven discovery actions required by monitoring-centered architectures.

The same qualitative pattern is observed in the cloud scenario, although overall coverage levels are higher for all approaches. Lightweight utilities decline from approximately 83% to 67% as scale increases, and traditional monitoring decreases from about 91% to 81%. Enterprise CLM again shows strong but imperfect performance, with coverage decreasing modestly from roughly 95.5% to 93.6%. EmTool, by contrast, remains essentially invariant to scale, achieving coverage between 98% and 99.2% across all infrastructure sizes. The visual gap between EmTool and all comparator architectures increases with scale in both scenarios, directly supporting Hypothesis H1: the expected number of certificate omissions grows significantly more slowly under EmTool than under monitoring-based approaches.

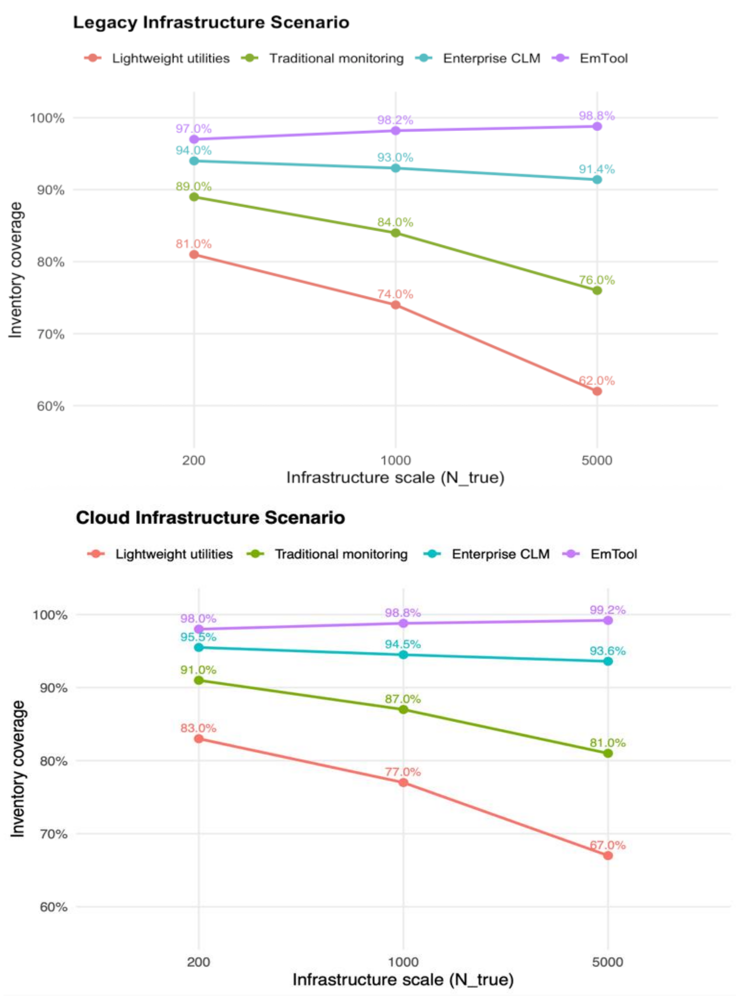

Figure 1: Inventory Coverage of TLS Certificates as a Function of Infrastructure Scale

These visual trends are corroborated by formal statistical tests using the number of missed certificates as the dependent variable at large scale (Table 4). In both scenarios, two-sample Welch t-tests comparing EmTool to each baseline architecture yield extremely large test statistics and vanishingly small p-values. In the legacy scenario, EmTool produces dramatically fewer omissions than lightweight utilities ($t = 45.70$, $p < 1\times10^{-20}$, $d = 11.80$), traditional monitoring ($t = 34.57$, $p < 1\times10^{-20}$, $d = 8.93$), and enterprise CLM platforms ($t = 22.21$, $p < 1\times10^{-15}$, $d = 5.73$). Analogous results hold in the cloud scenario, with EmTool again outperforming lightweight utilities ($t = 41.92$, $p < 1\times10^{-20}$, $d = 10.82$), traditional monitoring ($t = 31.06$, $p < 1\times10^{-20}$, $d = 8.02$), and enterprise CLM ($t = 18.96$, $p < 1\times10^{-12}$, $d = 4.89$). All effect sizes are exceptionally large, indicating not only statistical significance but also substantive differences in omission risk.

Table 4: Two-sample Welch t-test results (K = 30, N_{true}= 5000)

Scenario	Comparison (DV = N_{missed})	t	df	p	Effect size (Cohen's d)
Legacy	EmTool vs Lightweight	45.70	29.27	< 1e-20	11.80
Legacy	EmTool vs Traditional monitoring	34.57	29.40	< 1e-20	8.93
Legacy	EmTool vs Enterprise CLM	22.21	30.61	< 1e-15	5.73
Cloud	EmTool vs Lightweight	41.92	29.19	< 1e-20	10.82
Cloud	EmTool vs Traditional monitoring	31.06	29.33	< 1e-20	8.02
Cloud	EmTool vs Enterprise CLM	18.96	30.30	< 1e-12	4.89

Hypothesis H2 predicts that in cloud environments, a statistically significant fraction of certificates is discovered exclusively through load balancer configurations and cloud provider APIs, resulting in fewer omissions relative to monitoring-based approaches. Results from the cloud scenario provide strong empirical support for this hypothesis. Under EmTool, the cloud-specific discovery contribution (α_{LB}) is consistently large across all infrastructure scales, with a mean value of 0.18 and low dispersion (SD = 0.03). Substantively, this implies that approximately 18% of certificates present in the infrastructure are identified exclusively through load balancer metadata and cloud API signals. These certificates are not attached to statically enumerated domains or explicitly configured monitoring targets and would therefore remain undiscovered under monitoring-centric architectures operating at the same level of human reliability.

Two complementary statistical tests confirm both components of H2. First, a one-sample t-test evaluates whether the exclusive discovery share under EmTool differs from zero and shows an overwhelmingly significant effect ($t = 32.86$, df = 29, $p < 10^{-20}$), demonstrating that load balancer/API–only discovery is a systematic property of the architecture rather than a stochastic artifact of simulation noise. Second, a two-sample Welch t-test comparing EmTool to traditional monitoring architectures reveals that the exclusive discovery share is nearly an order of magnitude larger under EmTool (mean α_{LB}= 0.18 versus 0.02; $t = 27.71$, df = 35.37, $p < 10^{-20}$), confirming that this capability is not replicated by configuration-driven monitoring approaches.

Importantly, this exclusive discovery channel is directly associated with reduced omission counts at scale. At N_{true}= 5000, EmTool omits approximately 40 certificates on average, whereas traditional monitoring omits roughly 950 certificates under identical infrastructure dynamics and human error assumptions. The magnitude of this difference closely corresponds to the number of certificates discovered exclusively via load balancer and cloud API signals, indicating that this channel is not merely additive but is a primary mechanism driving EmTool’s superior performance.

6. Conclusion

The central technical contribution evaluated in this work is the use of Layer-7 load balancers and front-end routing configurations as the primary source of truth for TLS certificate discovery. Existing monitoring architectures, including enterprise CLM and machine identity platforms, derive their inventories from combinations of manually enumerated endpoints, environment-level scopes, periodic network scans, and certificate authority integrations. While effective within declared boundaries, these mechanisms remain structurally indirect: certificates are discovered only insofar as operators correctly specify where discovery should occur. As infrastructures grow and evolve, this assumption systematically breaks down. Layer-7 load balancers occupy a qualitatively different position. They explicitly encode which domains are reachable, which certificates are bound to those domains, and when bindings change. This information exists independently of naming conventions, deployment topology, or organizational ownership. By anchoring discovery at L7 termination points (as operationalized in EmTool) certificate inventories are derived directly from the infrastructure elements that actually govern encrypted traffic, rather than from administratively maintained abstractions.

The empirical results demonstrate that this architectural shift fundamentally alters how omission risk scales. Across both legacy and cloud scenarios, monitoring-centered approaches exhibit sharply increasing numbers of missed certificates as infrastructure size grows, reflecting the accumulation of human configuration errors under frequent change. Enterprise CLM platforms attenuate this effect but remain sensitive to incomplete onboarding of discovery scopes and configuration drift across environments. In contrast, the load-balancer-centric discovery model maintains near-complete coverage even at large scale, because certificates enter the inventory precisely when they are attached to TLS-terminating components. The cloud scenario provides particularly strong evidence for the value of this approach. A substantial fraction of certificates is discovered exclusively through load balancer and cloud API data, objects that remain invisible to monitoring-based architectures operating under comparable conditions. These certificates are not edge cases: they correspond to dynamically created services, ephemeral environments, and automated routing changes that are characteristic of cloud-native systems. The sharp reduction in omission counts observed under the EmTool-style model aligns directly with the presence of this exclusive discovery channel.

From a practical standpoint, the results have direct implications for both organizational risk management and national cybersecurity. TLS certificate failures remain a recurrent cause of large-scale service disruptions affecting financial systems, healthcare platforms, government services, and critical digital infrastructure. Importantly, a substantial share of these incidents originates not from renewal errors, but from certificates that were never incorporated into monitoring inventories and therefore expired or misconfigured without detection. As certificate lifetimes continue to shorten and cloud-native infrastructures expand, the attack surface created by “unknown certificates” grows, increasing the likelihood of cascading outages, loss of service availability, and exploitable trust failures at a national scale.

The findings of this study demonstrate that mitigating these risks requires architectural innovation rather than stricter procedures or increased human oversight. Discovery mechanisms anchored in Layer-7 load balancer state (where domain exposure and certificate bindings are authoritatively defined) systematically reduce blind spots that persist even in advanced enterprise certificate management systems. The alternative auto-discovery approach developed and evaluated in this research shows how such architectures can materially lower omission risk without increasing organizational burden. More broadly, the results suggest that the adoption of load-balancer-centric, event-driven certificate discovery should be treated as a strategic component of modern cybersecurity policy and infrastructure design, particularly for organizations operating services whose reliability is critical to national economic stability and public trust.

References

[1] Internet Security Research Group. "2025 ISRG Annual Report." Internet: https://www.abetterinternet.org/documents/2025-ISRG-Annual-Report.pdf, 2025 [Dec. 21, 2025].

[2] Google. "HTTPS Encryption on the Web." Internet: https://transparencyreport.google.com/https/overview?hl=en, 2025 [Dec. 21, 2025].

[3] CA/Browser Forum. "Ballot SC-081v3: Introduce Schedule of Reducing Validity and Data Reuse Periods." Internet: https://cabforum.org/2025/04/11/ballot-sc081v3-introduce-schedule-of-reducing-validity-and-data-reuse-periods/, Apr. 11, 2025 [Dec. 21, 2025].

[4] CyberArk Software Ltd. "2025 State of Machine Identity Security Report." Internet: https://www.cyberark.com/CyberArk-2025-state-of-machine-identity-security-report.pdf, May 5, 2025 [Dec. 21, 2025].

[5] Keyfactor, Inc. "2024 PKI and Digital Trust Report." Internet: https://www.keyfactor.com/2024-pki-and-digital-trust-report/, 2024 [Dec. 21, 2025].

[6] A. Sun, J. Lin, W. Wang, Z. Liu, B. Li, S. Wen, Q. Wang, and F. Li. "Certificate Transparency Revisited: The Public Inspections on Third-party Monitors." Network and Distributed System Security (NDSS) Symposium 2024. Internet: https://www.ndss-symposium.org/wp-content/uploads/2024-834-paper.pdf, 2024 [Dec. 21, 2025].

[7] N. Kataria. "Impact of Certificates in Multi-Plane Architectures." International Journal for Multidisciplinary Research, vol. 7, no. 5, pp. 1–12, Oct. 2025.

[8] N. Shaik. "Automated TLS Certificate Lifecycle Management: A Policy-Driven Framework for Kubernetes Security Hardening." Global Journal of Engineering and Technology Advances, vol. 23, no. 01, pp. 250–257, 2025.

[9] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. "Analysis of the HTTPS Certificate

Ecosystem." Proceedings of the Internet Measurement Conference (IMC), pp. 291–304, Oct. 2013.

[10] S. Klotz, A. Kopper, M. Westner, and S. Strahringer. "Causing Factors, Outcomes, and Governance of Shadow IT and Business-Managed IT: A Systematic Literature Review." International Journal of Information Systems and Project Management, vol. 7, no. 1, pp. 15–43, 2019.

[11] J. Göppert, A. Walz, and A. Sikora. "A Survey on Life-Cycle-Oriented Certificate Management in Industrial Networking Environments." Journal of Sensor and Actuator Networks, vol. 13, no. 2, art. 26, 2024.

[12] P. S. Yadav. "Automation of Digital Certificate Lifecycle: Improving Efficiency and Security in IT Systems." Journal of Mathematics & Computer Applications, vol. 2, no. 4, pp. 1–4, Oct. 2023.

[13] G. Thiyagarajan, V. Bist, and P. Nayak. "AI-Driven Configuration Drift Detection in Cloud Environments." International Journal of Communication Networks and Information Security, vol. 16, no. 5, pp. 721–743, 2024.

[14] National Institute of Standards and Technology (NIST). Security and Privacy Controls for Information Systems and Organizations: NIST Special Publication 800-53 Revision 5. Gaithersburg, MD: NIST, 2020.

[15] Kumara, M. Garriga, A. U. Romeu, D. Di Nucci, F. Palomba, D. A. Tamburri, and W.-J. van den Heuvel. "The Do's and Don'ts of Infrastructure Code: A Systematic Gray Literature Review." Information and Software Technology, vol. 137, art. 106593, Sept. 2021.

[16] S. Kraemer, P. Carayon, and J. Clem. "Human and Organizational Factors in Computer and Information Security: Pathways to Vulnerabilities." Computers & Security, vol. 28, no. 7, pp. 509–520, Oct. 2009.

[17] J. G. Proudfoot, W. A. Cram, and S. Madnick. "Weathering the Storm: Examining How Organisations Navigate the Sea of Cybersecurity Regulations." European Journal of Information Systems, vol. 34, no. 3, pp. 436–459, 2025.