Computer Fraud and Security
ISSN (online): 1873-7056

Scaling Patch Management in Cloud-Native DEVSECOPS Environments with SIEM

Gaurav Malik

SAP AMERICA, INC., USA

Gauravv.mmallik@gmail.com

Abstract

The study discusses the topicality of Security Information and Event Management (SIEM) telemetry to the management of the magnitude of patching control on cloud-native DevSecOps stacks with Kubernetes, containers, and micro services. Similarly, operationally cloud-native patching is a more complex task compared to server patching because immutable infrastructure requires restart image rebuild-and-redeploy cycles, a setting of auto scaling produces rapid inventories churn, and multi-clouds deployments, leave clusters in non-homogenous versions and configuration of individual nodes. In the meantime, exploitation timing has fallen to a disruptive mismatch of actual attacker velocity and enterprise patch time. It develops and verifies a patch management model that uses SIEM that includes the discovery of assets, vulnerability intelligence enrichment, CI/CD and Gitops coordination, SIEM analytics, and an automated feedback loop escalating the corrective action along with indications that it was exploited. A baseline CVSS-only prioritization model under exploit-attempt telemetry, and known indicators of exploited vulnerabilities tested on an experimental basis compare and contrast the model against a SIEM-enhanced model. Findings show a realistic growth in patch velocity and compliance to the aggregate Mean Time to Patch decreasing to 6.1 days (versus 12.4 days) and Yesterday to Patch (versus 8.6 days) and compliance. Scalability testing establishes the high patch—throughput and reliability of deployment to high volumes of workload. The present results endorse the SIEM-based prioritization as a measurable.

Keywords; *SIEM, Patch Management, Kubernetes, DevSecOps, Vulnerability Prioritization*

1. Introduction

Kubernetes-hosted, container-based, and microservices-based models of cloud-native delivery have been associated with the compensation of frequency of deployments, higher scalability, and a decrease in the time needed to roll out a new feature, but the shift has also led patch management to be more challenging to perform repeatedly at enterprise scale. In contrast to traditional systems where patching is directly applied to long-lived servers, immutable infrastructure system concepts are applied to containers, and thus remediation typically involves recreating container images, re-testing application behavior and redeploying services into multiple containers. The nature of auto scaling and workloads with a short lifetime makes this more complex since the nodes and pods keep being added and removed, which introduce constant flux in the inventories of assets and also heightens the risk of unmonitored exposure. Multi-cloud implementations further aggravate the condition of disorganized Kubernetes, node OS, and container registry deployments and shared cloud services, which usually results in dissimilar patch maintenance, policy directedness and cross-interdepartmental coordination.

Exploitation periods have also been reduced and minimized the duration in the evaluation, test, and fielding. The time-to-exploit median figure reported by Google Cloud threat intelligence had decreased to around 5 days, compared to around 32 days reported in 2021 to 2022, which means that attackers can switch between the exploitation and disclosure stage at a rate several times too rapid than many entities can safely respond. This is increased by the trend of breaches across the globe. The Annual Data Breach Investigations Report shows an almost three-fold increase of 180 and 14 percent in vulnerability exploitation as an access tool in annual report breaches and large-scale instances of exploitative behavior was a notable contributor to the rise. Such trends suggest that the severity scores should not be the only driving force behind patch decisions because successful prioritization is increasingly rooted in the evidence of exploitation and threat intelligence as a symptom of actual attacker behavior. Despite the presence of the high rates of the DevOps automation, the organizations still suffer the presence of the long patch queue, inconsistent deployment of the clusters and nodes, as well as inappropriate prioritization of the vulnerabilities that have been proved in the wild. The remediation and security

monitoring procedures are not linked therefore causing remediation to be sluggish regardless of existence of exploit attempts evidence. The resultant effect is that cross exposure of important systems and systems that are exposed to the internet is escalated, thereby putting the systems at a higher risk of being compromised. The given paper will accordingly justify and carry out the implementation of a large-scale patch-administration platform in cloud-native DevSecOps, which feeds on SIEM-based telemetry to minimize the time to attack-surface exposure, prioritize remediation actions based on the exploitation data, and make operational use of Kubernetes scale.

Measurement of the performance is taking place in terms of evaluating the measurable indicators revealing actual operational results like Mean Time to Patch, patch SLA compliance rate and critical vulnerability exposure window in days. Prioritization based on SIEM is informed by such sources as the CISA Known Exploited Vulnerabilities catalog which provides proven exposure to vulnerabilities that were exploited in the wild and may also inform remediation decisions guided by urgency. The metrics of improvement include the reduction in the Mean Time to Patch, the reduction in the quantity of vulnerabilities of over SLA scores, and the measurable reduction in the incidence of unpatched vulnerabilities events. It evaluates the speed, precision, and effectiveness of prioritization with SIEM integration, patch outputs, which are altered in SIEM-enabled DevSecOps patching and bottlenecks caused by using many clusters, namespaces, and groups. Some work environments that fall under the area include Kubernetes environments, CI/CD delivery, operating system and dependency patches and SIEM log integration and the significance of matching the patch management with the attacker presence and improving the working relationship between the detection and remediation to achieve a fixable risk.

The paper flows through the setting to indications of application and business direction. It investigates patch management literature on a classic and cloud-native environment, DevSecOps backgrounds, exploit patterns, SIEM feature functionality, and risk-driven prioritization and recovers gaps in SIEM-to-remediation feedbacks. It proceeds to mention the suggested architecture with introduction that comprises discovery, vulnerability intelligence, orchestration, SIEM analytics, and feedback. The methodology gives the organization of the experiment, set-up, data, measures, and tests of statistics to ascertain the findings. The implementation details consist of prioritization using SIEM, base level patching, scale testing and hot patching. The findings are reflective of the improvements in Mean Time to Patch, adherence to SLA, backlogs, the accuracy of prioritizations and the behavior of scale. Results are discussed under realistic exploratory forces, trade-off pressures of operation are addressed, lessons learned in the event of significant cases and company work guidance is given, as well as future work speculations in SOAR automation, machine learning priorities and predictive patch arrangements.

2.Literature review

2.1 Patch Management in Traditional vs Cloud-Native Environments

In the traditional IT environment, patch management is usually enabled through in-place updating of old servers, operating systems and enterprise applications [1]. Such environments tend to have planned time to maintenance, central to patching and can be known in terms of asset inventories. The host identities are rigid, the change rates are relatively small, and patch validation may be standardized between some limited number of types of platforms. These need underpin coordinated approvals, audited implementations and compliance reports that map patches on established systems and known owners. These are the assumption of patching that are changed by the cloud-native. The Kubernetes systems are built around containers, immutable infrastructure and rolling deployment trends [2]. Remediation is usually performed by rebuilding container images, verifying the new dependency stack and recreating workloads in the clusters, instead of they are patched on the running instances. Rolling upgrades of the versions of the Kubernetes version, node pools, and the supporting infrastructure of the kubelet, container runtimes, and the networking layer are also included in patching of clusters/node groups. Patching is incorporated into the delivery pipelines in the GitOps entailing the system state being coded defining where it is a commit where it is reviewed and checked into a run time environment. It focuses on the repetitiveness and auditing characteristics, and, at the same time, a high degree of operation coordination and automated many services and clusters are obligatory in this model.

There is also the additional complexity that micro services systems confer patches with applications that are distributed over a multitude of individual services with different owners and dependency sets [3]. The Selection of the vulnerability footprint by the developers is indirect and so increases the vulnerability footprint making it difficult to know which portions of the components are exposed and where they need improvement. Applications teams, platform

engineering as well as security operations are often separated and therefore not consistent in their prioritization and also in patch rollout speeds and can keep backlog of remediation. The patch workload rate also turns out to be a nonlinear amount and grows proportionate to the amount of services; that is, a patch should be confirmed as compatible, regression-tested, controlled rollout, and able to roll back safely in a large number of deployments.

Figure 1 demonstrates a conventional patch management cycle consisting of a cycle of stable inventory monitoring, patch data collection, endpoint allocation, authorization, testing deployment, and documentation. This is a structured workflow that suits long-lived server landscape, whereas cloud native Kubernetes systems demand quicker rebuild-and-redeploy automation across dynamic workloads.

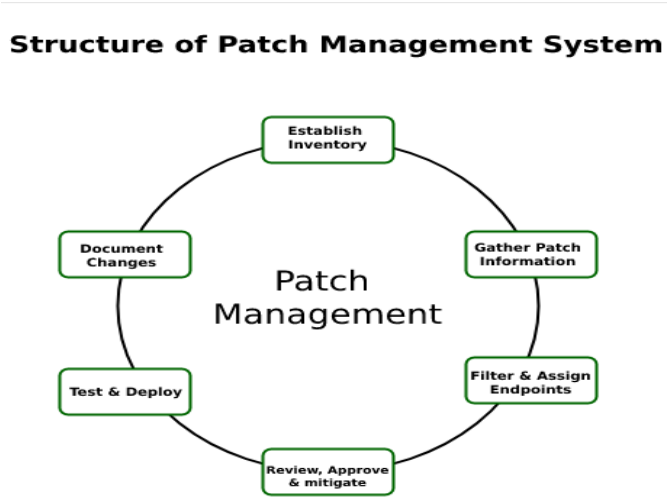

Figure 1: ***Patch Management***

2.2 DevSecOps in the Cloud Necessities.

Cloud-native DevSecOps is the implementation of software delivery that incorporates security controls without slowing down the delivery of software [4]. The constant up-to-date integration and delivery of code allows repeating of builds and tests and deployments and Infrastructure as Code allows the same configuration to be used in different environments. To enhance control, GitOps enhances deployment status as a declarative control combined with perpetual verification with configuration managed by version control as a way of improving traceability and reducing configuration drift. This operating model is often accompanied with tooling by breaking down the infrastructure provisioning and release of applications system, where in common deployment practices can be enforced across clusters. Pull-based model of deployment enables also promotion to be manipulated, rollback to be made more secure, and the state of patch deployment to be tracked aptly to patch governance at scale.

2.3 Vulnerability Exploitation Trends.

Threat intelligence data always indicate that vulnerability exploitation has been a prevalent means of intrusion due to its scalability and ability to provide a means of beating authentication. This especially applies when internet-facing systems are involved like VPN gateways, remote access portals and edge devices, where exploitation would directly result in entry into an environment. The rate of exploitation has also been accelerated and this has shortened the period of assessment, testing and deployment. Reduced exploitation cycles impose operational stress in the cloud native setting due to the fact that patching frequently involves image rebuilding, dependency validation, as well as orchestrating rollouts of services and clusters in a staged manner. Exposure persists during the most dangerous phase when the patch cycle lags behind the exploitation timescales, especially in external-exposed workloads and typical components of the platform [5].

2.4 Detection Engineering and Security Operations SIEM.

SIEM systems integrate infrastructure and application ingestion and analytics by integrating telemetry signatures of IDS and IPS, cloud audit logs, host event signatures, container runtime signatures, identity systems, and Kubernetes audit signatures [6]. SIEM is applicable in patch management to detect both attempts at exploitation and post-exploitation suspicious activity by correlated rule-based detection, enrichment, and contextual association of an alert with

an asset and workload. Abnormal API calls, odd behaviors of privilege escalation, and suspicious outbound communications, and web shell-like execution patterns are examples of indicators that may be used to ease the detection of high-risk services and reduce the time interval between identifying the threat and the ability to respond to it. Decisions made using SIEM systems can also be improved by enrichment which links alerts to workload metadata, which is cluster, namespace, image identity and service ownership, and hence remediation actions by the engineering team can be more productive.

2.5 Patch Prioritization Models based on risk.

Prioritization based on risk is becoming a requirement due to the lack of the severity scoring to reflect the exploitation probability and environmental impact [7]. More realistic models provide exposure context like internet availability, asset value, compensating controls and evidence of exploitation attempt. Accuracy can also be enhanced with runtime signals based on monitoring and security telemetry that can be used to determine which vulnerabilities are being exploited in practice. The process of prioritization can be more productive when evidence of exploitability is utilized to distinguish urgent work of remediation on an issue that might actually warrantless priority, from routine backlog work, and to put more attention into issues that have a higher likelihood of being exploited. This practice underpins tiered SLAs of patches which are based on exposure and threat activity instead of deeming all the high-severity vulnerabilities to be equally urgent.

2.6 Real-World Case Studies and 2.7 Research Gaps.

The high-impact incidents indicate that failure of patch governance may provoke large-scale compromise in case of known vulnerabilities that have not been fixed over a long time period. Such massive exploitation operations have shown that easily deployed software components can develop high-risk status very fast and lead to immediate demand to patch among numerous organizations. Nevertheless, other patch management frameworks continue to be devoid of the structured feedback loops that tie the detection telemetry to the prioritized patch execution, which constrain the capacity to react urgently when the exploit activity is detected. Empirical studies that quantify the effectiveness of patch management on a Kubernetes scale using quantifiable metrics such as the alert-to-remediation time, the Mean Time to Patch, the compliance with the patch SLA, and the reduction in the attack-surface over time are also limited. This disconnect makes it less reliable where architectures and workflows provide the most compelling patch performance in the real cloud-native operational conditions and real exploitation pressure [8].

3.System design & architecture

3.1 Proposed Architecture Overview

A cloud-native elastic Patch Management model takes into consideration patching as a continuous control rather than a routine operational maintenance process. Kubernetes environment is dynamic because workloads, the containers that are being used, and the different versions and configurations of different clusters and the pattern of ownership are dynamic. A proper architecture will therefore require the layered aspects to keep its asset inventory updated, continuously gauge its exposure to vulnerability, define remediation through the provision of pipelines and deploy SIEM analytics to purchase remediation priorities in the presence of threat activity.

Asset discovery layer maintains an authoritative database of patchable clusters in clusters. Querying in the Kubernetes API provides near real time visibility of namespaces, deployments, pods, node pools, container images and service accounts [9]. The quality of inventory is improved by resolving the Kubernetes objects to a CMDB or asset classified by stable identifiers that comprise cluster name, namespace, workload name, image digest and group of nodes. Image digests are applied because tags can be reused and can provide a deceptive picture of what is actually executing. Achieving over 95% of combined running workloads with well-defined ownership and production inventory synchronization with a maximum of 15 minutes to report auto scaling changes are the targets of practical governance.

Vulnerability information that has evidence of exploitability is ingested to the vulnerability intelligence layer where it is cleaned, provided in CVE sources, vendor advisories, and enrichment feeds. Triaging with Known Exploited Vulnerabilities catalog increases new vulnerabilities by identifying those that are already exploited in-the-wild. Scanned analysis of container registries provides a suggestion of vulnerable packages that have been shipped inside of shipped images, which can be used to prioritize what is actually running into production over what is present in the source code. A combination of repository scanning, image scanning, and runtime inventory correlation is a sensible implementation to

reduce number of the false positive that are as a result of non-utilized dependencies. The result should be normalized into a normal database of vulnerability containing identifiers of the assets, the version of the package, the version of the fix, the level of criticality, proof of exploitability, and exposure.

Patch orchestration layer implements both CI/CD and Gitops inspired remediation. CI/CD pipelines re-patch fixed images, run automated tests, security-testing and write signed artefacts to registries [10]. The Gitops tools then deliver updated images in declarative manifests which facilitates immutable delivery, limits configuration drift as well as the audit history of the delivery. The successful completion of high priority patch pipelines, involving build, test, scan, and deployment, over a period of 60 to 180 minutes is one realistic operational goal. This reduces the risks of disruption during emergency patching by gradual release options such as canary releases or blue-green releases. SIEM analytics layer will turn the telemetry into the operational feedback and prioritization. The most vital sources of logs are Kubernetes audit logs, ingress logs, service mesh telemetry, node logs, container runtime events, cloud control-plane logs, and endpoint detections where they exist [11]. Attempts of exploiting and post-exploitation behavior is identified using correlation rules to identify behavior patterns such as suspicious HTTP payloads, unusual command execution in a container, suspicious outbound connections, privilege-escalation attempts, and abnormal Kubernetes API access. Dashboards bridges exposure and patch status with measurable outcomes, such as Mean Time to Patch, SLA, and the quantity of the exploitative vulnerabilities remaining unpatched in the exposed services to the internet. The feedback loop between detection and remediation has the concern that the priority of patches is increased once the SIEM notification is received to remote exploitation attempts of vulnerable services. This increment comes up with immediate remediation processes, i.e., automated creation of tickets or changing the priority of the pipeline, and reports back on completion into vulnerability and SIEM reporting, finishing the loop of operation process.

Figure 2 demonstrates the important characteristics of cloud-based patch management: complete view of IT resources, automated deployments, vulnerability response processes, off-campus infrastructure services and dependence on fewer physical hardware, which allows large-scale cross-environment operations to be controlled quickly and centrally.

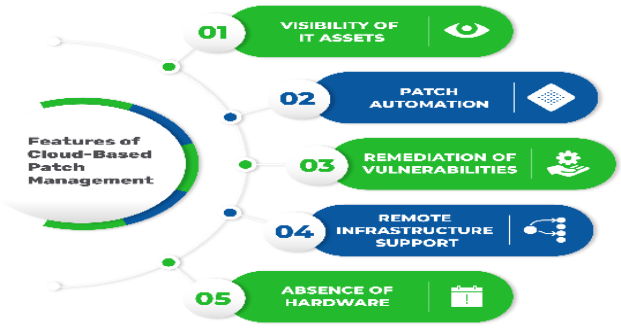

Figure 2: key features of cloud-based patch management

3.2 Cloud-Native Systems Patch Targets.

Patch targets are node operating systems, base images, runtime libraries including OpenSSL and glib, application dependencies handled by NPM or pip or Maven or kubelet and API server or exposed components such as ingress controllers or service mesh tooling [12]. One of the practical controls is a policy of node replacement like having a node age limit of 30 days in the production process in order to implement the adoption of patch due to rolling replacement as opposed to doing it manually.

3.3 Patch Workflow and 3.4 SIEM-Driven Prioritization Logic.

The patch workflow is followed by the detection, prioritization, patch execution, validation, observation and reporting. The most preferred mechanism of providing contents, which cannot be manipulated and backed by admission controls, is termed as build and deploy. SIEM-based prioritization applies tiered SLAs: remediation within 24 to 72 hours of KEV-listed vulnerability or SIEM-reported exploits, 7 days of high severity internet facing vulnerability with no exploits known, or 30 days of moderate severity internal-only services with compensating controls.

4.Methodology

4.1 Research Design

The experimental comparative design is followed to evaluate whether the patch prioritization according to Siem helps to achieve superior patch outcomes in cloud-native DevSecOps. The comparison of the two conditions is conducted on the occasion of a similar observation period and range of work. The baseline condition is the traditional criterion of CVSS severity and regular maintenance schedule. The enhanced state includes prioritization on the basis of SIEM co-locating CVSS with Known Exploited Vulnerabilities status, telemetry on exploit attempts, and environment risk score, which is determined based on exposure and business criticality. A comparison of the pre-enhancement and the post-enhancement result is made as well as tested based on the statistical significance. The general assumption is that SIEM-based prioritization reduces a Mean Time to patch and patch SLA compliance without increasing the rate of rollout failure [13].

4.2 Environment Setup

It uses a real-world production like setting, here a set of controlled clusters on Kubernetes, EKS, AKS, GKE or OpenShift of two or more clusters, to make a multi-environment deployment representation [14]. GitHub Actions or GitLab CI have CI/CD and Trivy, Clair or Grype contain vulnerability scanning of containers in the container registry as well as during the build. Audit logs, ingress logs and workload telemetry flows are aggregated and sent to a SIEM system, including Splunk, Microsoft Sentinel, or Elastic Security, and are shaped into exploit attempts and suspicious behavior tools. The services and the clusters can belong to the same category, both the baseline condition and better condition can be applied whereby control is being established to maintain the deployment cadence, test gates, and ownership of the team.

Figure 3 demonstrates an EKS production-style environment comprising of mixed-capacity node pools, in which the pods execute on on-demand and spot nodes and are guarded by Pod Disruption Budgets. Karpenter responds to schedule workloads in a very short time. to spot interruption notices and initiates AWS messaging (SQS/SNS).

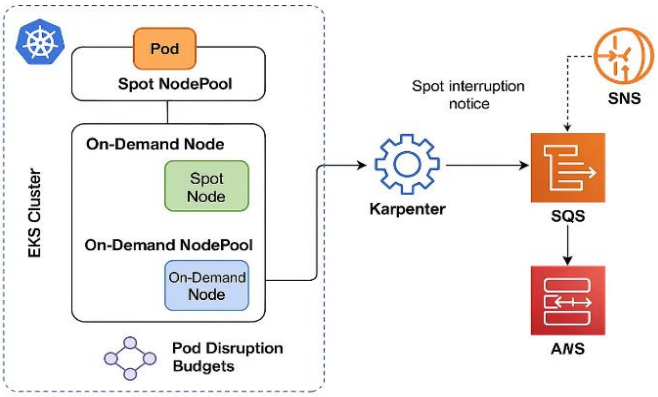

Figure 3: ***Exploring Amazon EKS***

4.3 Data Collection Sources

Kubernetes audit logs of configuration and access events, package-level vulnerability findings of container registry scans, exploit attempts and post-exploitation indicators of SIEM alerts, builds and deployment timestamps of CI/CD pipeline logs, and node upgrade events of operating system and Kubernetes component patches are collected. A distinct identifier is then used to track each vulnerability instance connecting the affected asset, image digest, package, detected and fixed versions [15]. Fields on time involve disclosure or detection time, SIEM alert time where available, patch deployment completion time and validation completion time.

4.4 Metrics

Patch performance is gauged by Mean Time to Patch which can be obtained by taking the average time difference of patch completion and vulnerability disclosure or detection [16]. Patch SLA compliance refers to the ratio of the number of vulnerabilities that are patched within given SLAs of 72 hours to patch the KEV-included or Siem-verified

vulnerabilities, 7 days to patch the high severity internet-facing vulnerability or 30 days to patch the moderate internal vulnerability. The additional measures include other indicators like critical vulnerability exposure window in days and patch success rate calculated as one minus failed rollouts divided by total rollouts. SIEM observability measures that are adopted are the Mean Time to Detect, Mean Time to Respond, and the alert-to-patch latency in the count of hours between the SIEM detection and deployment. The risk reduction metrics will include the count of internet-facing critical vulnerabilities open after the expiry of SLA, the count of KEV vulnerable not patched, and the percentage reduction of backlog at the end of 30-, 60-, and 90-day duration.

4.5 Statistical Analysis

Distribution tests are used to ascertain the suitability of a parametric analysis. Independent t-test is used to test the difference between conditions in terms of the mean time to patch under the assumption of normality, otherwise it uses a Mann Whitney U test. They report the effect sizes and 95 percent confidence intervals. Trend analysis compares backlog reduction with time based on linear regression or nonparametric trend test [17]. Spearman correlation is used to determine the relationship between the exploit-alert volume and the patch urgency, and the significance level of 0.05 is used to evaluate the relationship.

5.Experiments & implementation

5.1 Experiment 1 — Baseline Patch Workflow (CVSS-Based)

The initial experiment forms a baseline patching procedure that is representative of typical enterprise practices, which are associated with vulnerability remediation via primary motivation of CVSS severity. Severity levels are used to rank the vulnerabilities (critical: CVSS 9.0 and above, high: 7.08.9) and scheduling to patch is based on weekly or monthly maintenance. The remediation typically takes the form of regular CI/CD re-builds or node upgrade, but no scaling urgency is observed according to exploit activity. Participation in SIEM is also limited to Incident response procedures and not prioritization [18]. At this stage, the metrics used to measure performance include the mean time to patch, mean baseline across all levels of severity in days, patch SLA compliance rate, and the count of high-risk vulnerabilities beyond SLA thresholds. Other tracking items are the figure of the production images, which have unresolved critical packages, and the mean number of backlog per cluster. This baseline gives a control condition of whether SIEM-based prioritization generates operationally significant improvement.

Figure 4 represents a layered security workflow in which controls are taken through various steps starting with the cognitive process of understanding the rule set, configuring it, integrating it with other tools, and monitoring and analysis of alerts and maintenance. This is similar to the baseline CVSS patch procedure that is based on formalized, timed remediation that is not propelled by exploits.

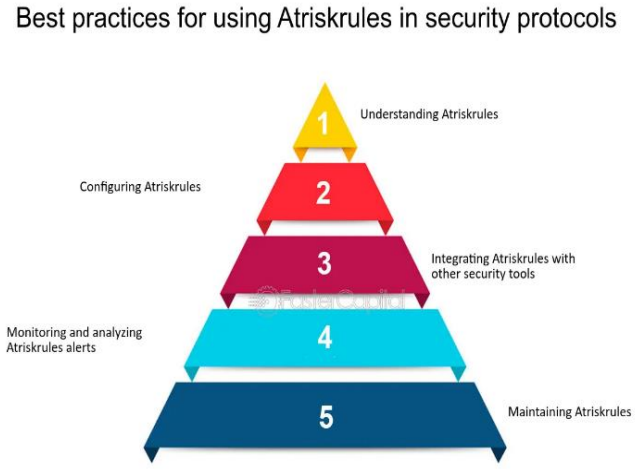

Figure 4: ***Vulnerability Scanners***

5.2 Experiment 2 -SIEM-Enriched Patch Prioritization

The second experiment adds both SIEM-informed vulnerability triage by adding both Known Exploited Vulnerabilities indicators and exploit-attempt indicators (as identified by the SIEM) to the vulnerability backlog. Some of the high-confidence detections in SIEM telemetry are repeated exploit-pattern HTTP payloads to vulnerable systems, unusual process treatise in containers, privilege-escalation activities, and deviatory Kubernetes API calls. These signals have vulnerabilities attached to them that are added to an exploit-based patch queue that overrides regular scheduling. The experimental metrics are percentage decrease in MTTP relative to the baseline, percentage-based similar improvement in the percentage coverage of critical patches by SLA and the time interval between the first SIEM detection and actual patch [19]. The performance can be measured using the number of vulnerabilities with the highest priority identified as SIEM (compared to those with lower priority identified as SIEM) are patched sooner and the number of alerts about exploitations can be reduced after the deployment. It is presumed that SIEM enrichment minimizes the duration during which the critical vulnerabilities are exposed, especially when it comes to internet-facing services or part of a shared platform, such as ingress controllers.

5.3 Experiment 3 Patch Automation at Scale in Kubernetes.

The third test confirms the scaling behavior of the patch framework in the context of natural growth in cloud-native. Scaling is put through the test by increasing services to 200, namespaces to 40, clusters to 5, number of deployments attempting to 100 deployments per day. The aim is to monitor the stability of patch automation in the situation when the workload sprawl in terms of workload demand increases and when the pipeline execution demand increases. Measures such as minutes of average patch rebuild and deployment duration, rollback rate divided by the number of patch deployments, and patch throughput are some of the key ones [20]. CI/CD runner queue time and staging to production promotion delays in the experiment are also monitored, as these two directly correlate with patch velocity. The following stability goals related to enterprise level patching are: patch pipeline completion within reasonable limits, rollback rates below operation tolerance limits, and throughput collapse as workload increases.

5.4 Experiment 4 - Exploited CVEs Response- "Hot Patch"

The fourth is an experiment on urgent patches to exploited vulnerabilities. Patching is triggered immediately when SIEM identifications of active attempts to exploit a vulnerability have occurred or when a vulnerability is the same as Known Exploited Vulnerabilities enrichment. The patch process is almost an accelerated one in which rebuild, testing, scanning and deployment are implemented in priority order. The desired SLA is completion of the patch within 24 to 72 hours based on the work load exposure and risk of deployment [21]. Outcomes compliances of hot patch SLA, duration time between the process of exploit and remediation and the extent of exploitation alert frequency after remediation moves are the measures of outcomes. A successful definition will be a gradual reduction of the exposure duration of patented vulnerabilities, minimal operational impact with a speedy rollout, and a noticed reduction in the number of exploit-attempt signatures that post-patches and a comprehensive attack will demonstrate on affected workloads and clusters.

6. Results

6.1 Patch Efficiency Improvements

The patch workflow induced by CVSS generated a Mean Time to Patch (MTTP) of 12.4 days of all vulnerabilities and 8.6 days of critical vulnerabilities [22]. Following the introduction of SIEM-enriched prioritization, the total of the MTTP dropped to 6.1 days and the critical one to 2.3 days. This is a decrease of a 50.8 percent of overall patch time and a 73.3 percent cut in critical issues. The compliance of patch SLA also increased significantly. In the condition of the baseline, 62.5% of vulnerabilities were addressed under SLA, and critical SLA compliance was 41.8%. In the improved SIEM-directed state, total SLA compliance reached 86.9 and critical SLA compliance reached 91.2. Critical backlog exposure also decreased and the number of critical vulnerabilities older than SLA during the evaluation window dropped by 137 open items to 39 open items, indicating a decrease by 71.5 percent. Internet facing loads were recorded to improve most with the average exposure window reduced to a critical vulnerability dropping to 2.7 days as opposed to 9.1 days.

6.2 SIEM Effect on Precision of Priority.

SIEM-based prioritization enhanced the capacity to point out vulnerabilities that had to be addressed on a high priority basis since the exploits were in an active state of action [23]. More time pressed the 112 vulnerabilities into the patch queue, which was driven by exploits. Among them, 83 of them coincided with known indicators of exploits, including the existence of multi-exploit-pattern HTTP payloads on known vulnerable endpoints, suspect Kubernetes API requests, as well as suspect container command execution. This produced a true positive of 74.1% wired to SIEM-induced escalation. The remaining 29 were ruled as false positive because there was no possibility to ascertain correlation with exploits after the up, hence, the false positive rate stood at 25.9. Noisy ingress scans and generic web attack payload were the most common false positive states and did not relate to possible vulnerabilities in the services. Despite this noise, the prioritization model was operationally valuable in the sense that SIEM escalation still brought about a reduction in time-to-remediate exploited or targeted vulnerability which directly resulted in a scenario of reduced chances of compromise.

Figure 5 depicts essential SIEM capabilities that can facilitate accuracy in prioritization such as log collection, log processing, log storage, querying, correlating, dashboards, alerts, and incident management. These capabilities were able to exploit-driven triage with 112 escalations resulting in 74.1 true positives and 25.9 false positives.

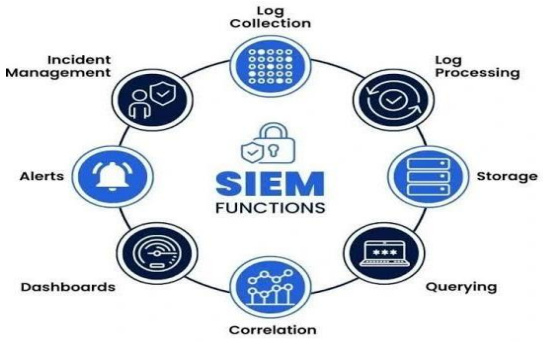

Figure 5: Importance-of-siem

6.3 Scalability Findings

Patch automation was found to scale well with the complexity of platforms. As the amount of services grew by 50 to 200 services and the amount of clusters grew by 1 to 5 clusters, the patch throughput also doubled in value, and the patch throughput increased 32 patches per week to 118 patches per week; this is a 268.8% improvement in the amount of patches that were patched each week. The average time to execute the pipeline went up by 25.0 percent when compared to the small environment when the average time to deploy patches was 28 minutes, whereas in the scaled situation, execution was 35 minutes, which is still a big improvement in deployment time even though the workload footprint was much bigger. Rollback and failure behavior were acceptable. Rollback rate had improved a bit on 1.8 percent to 2.4 percent in high-volume patch periods, whereas the patch deployment failure rate was less than 3.0 percent in peak workload settings. These results demonstrate that the automation pipeline maintained the performance and reliability when the volume of the daily deployments grew 10 to 100 deployments per day [24].

6.4 Comparative Summary: Baseline vs SIEM-Driven.

The statistical analysis showed the decline of the MTTP in the enhanced condition relative to the baseline condition to be significant [25]. The non-normality of the MTTP distribution justifies the application of a MannWhitney U test, which gave $p = 0.001$ of overall difference in the MTTP and the difference in criticalMTTP. The overall reduction in the number of days of MTTP had a 95 percent confidence interval given 5.1 days to 7.4 days, whereas critical reduce of MTTP had a 95 percent interval of 4.8 days to 7.2 days. The improvements in patch SLA compliance were also statistically significant, and a two-proportion z-test yielded $p < 0.001$ in overall compliance improvement and $p < 0.001$ in critical compliance improvement. The findings can be viewed as corroborating the conclusion that SIEM-informed prioritization led to practical and scaled improvements in patch performance with the greatest effect noted on the workloads that were seen to be exposed to active exploitation attempts and services considered internet-facing or business-critical.

7.Discussion

7.1 Practical Meaning of the Results

The findings may also suggest that SIEM-informed patch prioritization has a significant effect on reducing the exposure time in cloud-native settings where the traditional patch cycles do not allow current attacker behavior [26]. The resultant average time-to-exploit in days (not weeks) provides patching programs that react to monthly or even weekly maintenance patrols, a feasible schedule of the anticipated post-disclosure gap between fix and attackers. Five days exploitation period implies that a Mean Time to Patch of a time-scale in the double digits puts production services at the highest threat period, namely to internetwork-facing workloads and common platform services like ingress controllers and identity services. The specified increment on critical compliance, in the respect of SLA, is operationally an improvement since it will vary in terms of remediation of the reaction to an incident response within a policy addressing risk aversion. This further applies when the exploitation based intrusion routes accommodate a sizeable portion of breaches because the vulnerabilities not patched would be a target to initial access rather than a source of risk. Practically, vulnerability reduction as set out by KEV and SIEM items significant in making attempts to execute exploit program cuts the chances of ad-hoc scanning activity into service exploit, stolen keys, further propagation, or information deterioration.

7.2 Why SIEM Enhances Patch Management.

SIEM facilitates better handling of patches mostly through evidence of actual attack pressure so that prioritization decisions are based on a more accurate assessment of the likelihood of exploitation rather than making decisions solely on theoretical severity [27]. Backlogs based on CVSS frequently include voluminous quantities of highest level of findings that struggle with limited engineering capacity resulting in delayed fixes and inaccurate service-level performance [28]. SIEM monitoring helps in isolating the vulnerabilities that are under attack, versus those ones that are not. This enables the accuracy of uniqueness of prioritization by discovering what vulnerable services are being solicited using exploit-patterns, which workloads are having uncharacteristic runtime, and what operation by Kubernetes control plane is suggestive of attempted privilege escalation or reconnaissance. Patching curbs are no longer done in the infrastructure of wide remediation efforts but in the elimination of the targeted risks after the discovery has been made of evidence of exploits that have been attributed by particular assets through the assistance of the enrichment and inventory mapping. This enables realistic, risk based SLAs where the most vulnerable and most targeted vulnerabilities are given a higher priority to be placed in the immediate pipeline with moderate internal outcomes being placed in the schedule in a way that will not affect the stability of delivery and will not result in significant wasted remediation effort. The workflow compression of the systems of the alert-to-patch-enabled algorithms of SIEM also hints that the duration of the transition between a detection phase and a remediation phase becomes reduced, covering the territory between which the period of the successful execution of multiple deployment phases with an attempt at exploitation is often allowed to proceed [29].

Figure 6 illustrates SIEM capabilities that support cloud-native patch governance by aggregating and correlating security data, enabling threat detection and response, issuing alerts, supporting forensics, and strengthening compliance reporting, behavior analytics, threat intelligence, orchestration, and cloud security visibility for faster remediation decisions.

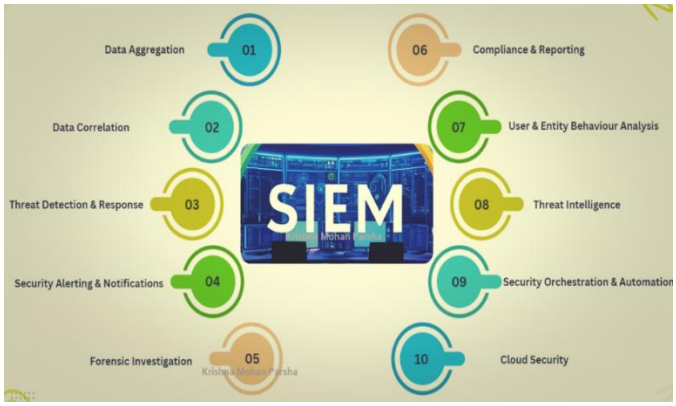

Figure 6: why-siem-still-heartbeat-cybersecurity

7.3 Operational Trade-offs

The operational advantages of patching based on SIEM have their trade-offs which should be addressed so that it does not cause instability. The rapid patching may be used to gain more deployment frequency, which in turn can increase the chances of downtime when the modification is implemented without proper verification. This risk is addressed by rolling upgrades, canary releases, blue-green deployments, and automated rollback gates by health checks and error budgets. The other tradeoff is effort needed to fine tune SIEM detections and high signal correlation rules. Even with no tuning, a high noise of detections may cause unnecessary escalations forcing a greater workload on engineering teams with less trust in prioritization outputs. False positives will be anticipated in areas with heavy scanning and fake attack traffic areas, particularly at the ingress. Examples of practical controls are that, there must be more than one corroborating signal before an escalation is raised, there must be rate limits, that the indicators of low confidence be kept apart from the indicators of high-confidence exploit detections, and that the true positive rates must be continuously measured in order that escalation policies be evidence-based. Other overhead can be seen in data quality management, too, as the enrichment process requires proper mapping of the assets or a consistent identity tracking of the images, and sound pipeline telemetry. Automation may be ruined by weak inventory attribution, which prevents target assignment and postpones remediation ownership [30].

7.4 Lessons from Real Incidents

Any real-life cases support the significance of a shorter patching period and a better prioritization approach. The Equifax breach showed that prompt resolution of an identified vulnerability can result in mass-scale compromise in case of a lack of governance, the visibility of assets, and accountability. The most important lesson is that there is no risk mitigation in patch availability unless its implementation is fast and it is checked by the entire scope of assets. The MOVEit exploitation wave indicates another scaling threat, in which exploitation of broadly spread software can quickly affect a large number of organizations, bringing about a situation in which exploitation strain is soon to begin and that the reaction of patch has to be between speedy end is highlighted. These examples coincide with the key finding of the work: patching should be informed by indications of exploitation and implemented as an automated scale. This means that in Kubernetes environments, patching is a continuous delivery issue that is enabled by immutable releases and prioritization based on SIEM, as opposed to periodic maintenance. It is most practical when the detection telemetry is operationalized in the same pipelines as workloads are built and deployed so that it can be quickly remediated with a controlled risk in a measured fidelity to compliance under actual exploit stress.

8. Recommendations

8.1 Technical Recommendations

Patch governance is more realistically applied as policy-as-code, but not in the form of manual inspection [31]. Workloads should not be deployed that do not meet the required set patch SLAs, namely where the vulnerability is denoted as an exploited vulnerability or where there is an indication of an exploit in the telemetry. In order to avoid tag drift, configuration drift, the policy rule must take image digests and signed artifacts as units of deployment. This should be instigated by a communal cue on the premises of the perceived befazed vulnerabilities and the perceived exploit attacks discovered by SIEM beyond CVSS-stipulated rating since the evidence of exploitation is a demonstration of actual danger as opposed to the theoretical seriousness. The implementations of GitOps must establish a procedure of standardizing installation in the clusters, as well as the environment to the extent that the rollout, rollback, and complete audit of the adjustments of the patches with the help of version control can be achieved. SBOM generation, and continuous scanning should be added to dependency patching in order that the vulnerable transitive dependencies may be identified and measured in-between the run-time and build-to-build inventories. Correlation The SBOM based correlation also allows making better judgments of remediation success, as it can indicate that a vulnerable library is shipped and delivered rather than wasting time on knowledge lacking in production artifact.

8.2 Recommendations in Governance.

A clear ownership model should exist, to ensure that the backlog of Enterprise patch programs is never assembled due to lack of accountability [32]. A repository will include metadata that should be assigned to individual services, and Kubernetes labels should be provided as well as CMDB ownership specific to a responsible team and the remediation process then forwarded to the relevant maintainers rather than passed through triage orchestrated ad hoc.

Operationally meaningful reporting (i.e. the degree to which the severity level or cluster or business unit or service owner or time-trends) SLA compliance indicators should be incorporated on the compliance dashboards to reflect the decrease in the number of attack surface. Aging mindsets of known exploition infirmities have to be incorporated in dashboards as well, because this is first place category of backlog. Patch exceptions should be captured as risk decisions and not as operational delays and should be approved with expiry dates compensating controls and revalidations. This makes it easy to eliminate unlimited deferrals and helps in preparing audits, proving that the weak areas that are not being addressed are under control.

Figure 7 shows an example of a data governance model in which metadata, data quality, security, integration, architecture, and operational controls are collaborating to guarantee accountability. These items in enterprise patch governance are plain service ownership, consistent CMDB mapping, reporting that is in compliance, and exception management.

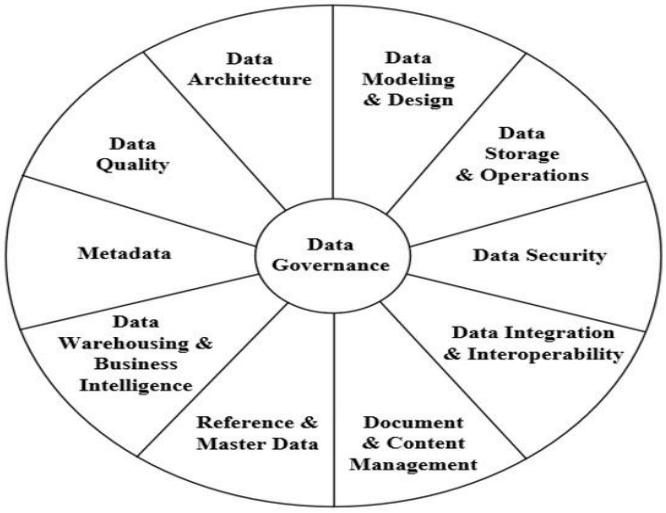

Figure 7: DAMA Wheel

8.3 SIEM Integration Roadmap

The integration of the SIEM is to be performed in the block that will allow reducing the effects on the functioning of the work but gradually and continually increase the quality of the decision-making [33]. The first step is projected to be vulnerability ingestion and patch telemetry combination to gather the SIEM dashboards to have the ability to record the genuine patch status as well as vulnerability exposure of clusters, workloads, as well as images. The second step will be required to provide exploit detection correlation whereby the SIEM alerts are correlated to vulnerable assets using normalized identifiers such as cluster, namespace, workload and image digest. The priority at this stage should be a high-confidence detection implying attempts of exploitation or the post-exploitation behavior and the accuracy of the escalation evaluated by the true positive and false positive rates. The final phase should enable automated remediation triggers where high confidence exploit detection or exploited vulnerability known matches automatically increase precedence in a pipeline, create urgent remediation workloads, and track closure through deployment validation telemetry. Gating of automation must be performed so as to prevent unstable condition by use of progressive delivery control and roll back policies [34].

9. Conclusion and future work

The findings show that patch management using SIEM aspects reduces patch latency and the time of risk exposure through coordinating the remediation process with evidence of exploitation in the form of production telemetry. In cloud-native Kubernetes, which has immutable deployments, indicators of attack propagation are automatically highlighted on auto scaling, and releases occur regularly, addition of SIEM signals to prioritization and delivery pipelines can be helpful to mitigate the most urgent vulnerabilities and improve compliance outcomes. The traditional CVSS-only practices have the effect of producing large backlogs because scoring severity alone does not give any guidelines on what is being attacked in time of attack, what assets are available online or what services are most affected in operations. Comparatively, SIEM-informed prioritization attaches the observed exploitation attempts to the observed indicators that

correlate with vulnerable instances to prioritize the most vulnerable instances of vulnerable instances, so that remediation effort is concentrated on areas where maximum decrease in the probability of compromise is brought about. The approach is simpler to scale as automation has been integrated to recreate and redeploy fixed artifacts, deployment templates are standardized with CI/CD and GitOps examinations, as well as verification is a part of the procedure with verification telemetry checks that fixed versions have been booted. The closed feedback loop model also strengthens a coordinating aspect of security monitoring and engineering implementation because the escalating criterion can trigger high-priority remediation streams, and the patching completion events can be measured and recorded on-request. In practice, the model permits tiered patch SLAs depending on exposure potential and exposure occurrence so that the well-understood vulnerable spots get a prompt response and the current backgrounds of unsuccessful exploitation can get a prompt response without integrating the complete pipeline of vulnerabilities into the emergency patching.

These results can be restricted to other settings and organizations by many factors. The research has these limitations: they have and good quality of exploit telemetry since not every exploit attempt might leave traces of high-confidence and not all log coverage across clusters, namespaces, workloads, and layers of the runtime can be of significant interest. Detection reliability could also be influenced by the gaps in collection, as well as, the disparity in the ingress controller logging properties, and the run time to produce security relevant events in the workload. This, and even the upkeep of Kubernetes upgrade routes, the fashion in which node images are patched, and the degree of control-plane occasion visibility may vary involving distributors. This allows the provider to partially abstract the control plane of certain environments and patching nodes might involve various operating processes based on the node pools design and upgrade policy, and maintenance windows. The quality of SIEM rules, and tuning is another measure which depends on precision of priority, where false positives affect the efficiency of operations vary significantly based on its experience of detection engineering. Unnecessary false alarm due to noisy detection can also waste engineering resources and excessive demand on strict policies can result in the activity of exploitation being limited too early. Also dependent on the result of the findings are quality of enriching and mapping of assets due to the fact that. to be able to correlate the alert with the workloads and images as well as the owners, there would have to be constant identifiers.

Further study ought to refine this approach through greater automation, by experimenting with SOAR-based remedial conditionally performing patch deployment operations on the safe side e.g. by looking at observed attempts to abuse internet-facing services or known matches of exploited vulnerabilities that cover critical assets. Future research may investigate machine-learning-ensemble prioritization to forecast probability of exploitation and the type of operational effect through much finer inputs over exposure circumstance, asset level importance, exploit background, dependency range and stratagem of attacker. Predictive patch scheduling is yet another helpful guidance, as it may become easier to be ready to vulnerability waves of widely used components, reduce unplanned disruption in the course of an urgent fix, and maintain patch SLA performance on large-scale patching spikes once patch demand forecasting and capacity engineering are put into effect.

References;

[1] González-Granadillo, G., González-Zarzosa, S., & Diaz, R. (2021). Security information and event management (SIEM): analysis, trends, and usage in critical infrastructures. *Sensors*, *21*(14), 4759. https://doi.org/10.3390/s21144759

[2] Käldström, L., & Käldström, L. (2021). Encoding human-like operational knowledge using declarative Kubernetes. https://raw.githubusercontent.com/luxas/research/main/bsc_thesis.pdf

[3] Chandramouli, R. (2019). Microservices-based application systems. *NIST Special Publication*, *800*(204), 800-204. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204.pdf?ref=https://githubhelp.com

[4] Datla, L. S., & Thodupunuri, R. K. (2021). Designing for Defense: How We Embedded Security Principles into Cloud-Native Web Application Architectures. *International Journal of Emerging Research in Engineering and Technology*, *2*(4), 30-38. https://doi.org/10.63282/3050-922X.IJERET-V2I4P104

[5] GUAN, X. (2015). MeteorShower: geo-replicated strongly consistent NoSQL data store with low latency. https://www.diva-portal.org/smash/get/diva2:896034/FULLTEXT02.pdf

[6] Celeste, R., & Michael, S. (2021). Next-Gen Network Security: Harnessing AI, Zero Trust, and Cloud-Native Solutions to Combat Evolving Cyber Threats. *International Journal of Trend in Scientific Research and Development*, *5*(6), 2056-2069. http://eprints.umsida.ac.id/16067/

[7] Van der Fels-Klerx, H. J., Van Asselt, E. D., Raley, M., Poulsen, M., Korsgaard, H., Bredsdorff, L., ... & Frewer, L. (2015). Critical review of methodology and application of risk ranking for prioritisation of food and feed related issues, on the basis of the size of anticipated health impact. *EFSA Supporting Publications*, *12*(1), 710E. https://doi.org/10.2903/sp.efsa.2015.EN-710

[8] Iorio, M., Palesandro, A., & Risso, F. (2020). CrownLabs—a collaborative environment to deliver remote computing laboratories. *IEEE Access*, *8*, 126428-126442. https://doi.org/10.1109/ACCESS.2020.3007961

[9] Watada, J., Roy, A., Kadikar, R., Pham, H., & Xu, B. (2019). Emerging trends, techniques and open issues of containerization: A review. *IEEE Access*, *7*, 152443-152472. https://doi.org/10.1109/ACCESS.2019.2945930

[10] Gilbert, J. (2021). *Software Architecture Patterns for Serverless Systems* (p. 436). Packt Publishing. https://sciendo.com/2/v2/download/chapter/9781803244433/10.0000/9781803244433-001.pdf?Token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VycyI6W3sic3ViIjoyNTY3ODUxNywicHVicmVmIjoiNzY0NDg4IiwibmFtZSI6Ikdvb2dsZSBHb29nbGVib3QgLSBXZWIgQ3Jhd2xlciBTRU8iLCJ0eXBlIjoiaW5zdGl0dXRpb24iLCJsb2dvdXRfbGluayI6Imh0dHBzOi8vY29ubmVjdC5saWJseW54LmNvbS9sb2dvdXQvNjgwNmVhMTAxYjFmZDg2MjVmODc0YTk1NTRhNzI2NTMiLCJhdXRoX21ldGhvZCI6ImlwIiwiaXAiOiI2Ni4yNDkuNjYuMTcifV0sImlhdCI6MTc0NTI4NDIwNSwiZXhwIjoxNzQ2NDkzODA1fQ.p38C7H3H9oXJXD-e5O38BMf8-2z-RNMlrdYvBzuspXU

[11] Muresu, D. (2021). Investigating the security of a microservices architecture: A case study on microservice and Kubernetes Security. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1597972&dswid=9432

[12] Saito, H., Lee, H. C. C., & Wu, C. Y. (2019). *DevOps with Kubernetes: accelerating software delivery with container orchestrators*. Packt Publishing Ltd.

[13] Di Martino, C., Sarkar, S., Ganesan, R., Kalbarczyk, Z. T., & Iyer, R. K. (2017). Analysis and diagnosis of SLA violations in a production SaaS cloud. *IEEE Transactions on Reliability*, *66*(1), 54-75. https://ieeexplore.ieee.org/abstract/document/7835304

[14] Larkan, N. J., Raman, H., Lydiate, D. J., Robinson, S. J., Yu, F., Barbulescu, D. M., ... & Borhan, M. H. (2016). Multi-environment QTL studies suggest a role for cysteine-rich protein kinase genes in quantitative resistance to blackleg disease in Brassica napus. *BMC Plant Biology*, *16*(1), 183. https://link.springer.com/article/10.1186/s12870-016-0877-2

[15] Aslan, Ö. A., & Samet, R. (2020). A comprehensive review on malware detection approaches. *IEEE access*, *8*, 6249-6271. https://ieeexplore.ieee.org/abstract/document/8949524

[16] Li, F., & Paxson, V. (2017, October). A large-scale empirical study of security patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (pp. 2201-2215). https://doi.org/10.1145/3133956.3134072

[17] Turner, S. L., Karahalios, A., Forbes, A. B., Taljaard, M., Grimshaw, J. M., & McKenzie, J. E. (2021). Comparison of six statistical methods for interrupted time series studies: empirical evaluation of 190 published series. *BMC Medical Research Methodology*, *21*(1), 134. https://link.springer.com/article/10.1186/s12874-021-01306-w

[18] González-Granadillo, G., González-Zarzosa, S., & Diaz, R. (2021). Security information and event management (SIEM): analysis, trends, and usage in critical infrastructures. *Sensors*, *21*(14), 4759. https://doi.org/10.3390/s21144759

[19] Karri, N., & Jangam, S. K. (2021). Security and Compliance Monitoring. *International Journal of Emerging Trends in Computer Science and Information Technology*, *2*(2), 73-82. https://www.ijetcsit.org/index.php/ijetcsit/article/view/402

[20] Castelluccio, M., An, L., & Khomh, F. (2019). An empirical study of patch uplift in rapid release development pipelines. *Empirical Software Engineering*, *24*(5), 3008-3044. https://link.springer.com/article/10.1007/s10664-018-9665-y

[21] Mustafa, S., Sattar, K., Shuja, J., Sarwar, S., Maqsood, T., Madani, S. A., & Guizani, S. (2019). Sla-aware best fit decreasing techniques for workload consolidation in clouds. *IEEE Access*, *7*, 135256-135267. https://ieeexplore.ieee.org/abstract/document/8835038

[22] Koskenkorva, H. (2021). The role of security patch management in vulnerability management. https://urn.fi/URN:NBN:fi:amk-2021120924851

[23] Chandrashekar, K., & Jangampet, V. D. (2020). Risk-based alerting in SIEM enterprise security: Enhancing attack scenario monitoring through adaptive risk scoring. *International Journal of Computer Engineering and Technology (IJCET)*, *11*(2), 75-85.

[24] Arachchi, S. A. I. B. S., & Perera, I. (2018, May). Continuous integration and continuous delivery pipeline automation for agile software project management. In *2018 Moratuwa Engineering Research Conference (MERCon)* (pp. 156-161). IEEE. https://ieeexplore.ieee.org/abstract/document/8421965

[25] Berge, R. K., Bjørndal, B., Strand, E., Bohov, P., Lindquist, C., Nordrehaug, J. E., ... & Nygård, O. (2016). Tetradecylthiopropionic acid induces hepatic mitochondrial dysfunction and steatosis, accompanied by increased plasma homocysteine in mice. *Lipids in Health and Disease*, *15*(1), 24. https://link.springer.com/article/10.1186/s12944-016-0192-9

[26] Mamun, A., & Saidur, M. J. I. (2021). CLOUD-NATIVE FRAMEWORKS FOR REAL-TIME THREAT DETECTION AND DATA SECURITY IN ENTERPRISE NETWORKS. *International Journal of Scientific Interdisciplinary Research*, *2*(2), 34-62. https://doi.org/10.63125/0t27av85

[27] Renners, L. (2020). *Adaptive Prioritization of Network Security Incidents* (Doctoral dissertation, Dissertation, Neubiberg, Universität der Bundeswehr München, 2020). https://athene-forschung.unibw.de/doc/134215/134215.pdf

[28] Guo, W. (2017). *Improving reliability of service oriented systems with consideration of cost and time constraints in clouds* (Doctoral dissertation, University of Huddersfield). https://eprints.hud.ac.uk/id/eprint/31699/

[29] Smith, C. R., & Docherty, S. F. (2021). *Enhancing Defense Network Operations Centers Through the Use of Private Sector Monitoring Tools, Applications, Processes, and Procedures* (Doctoral dissertation, Monterey, CA; Naval Postgraduate School). https://calhoun.nps.edu/bitstream/handle/10945/67815/21Jun_Smith_Docherty.pdf?sequence=1

[30] Akinleye, O. K., & Adeyoyin, O. (2021). Process Automation Framework for Enhancing Procurement Efficiency and Transparency. https://doi.org/10.32628/SHISRRJ214458

[31] Karri, N., & Jangam, S. K. (2021). Security and Compliance Monitoring. *International Journal of Emerging Trends in Computer Science and Information Technology*, *2*(2), 73-82. https://doi.org/10.63282/3050-9246.IJETCSIT-V2I2P109

[32] Souppaya, M., Stine, K., Simos, M., Sweeney, S., & Scarfone, K. (2018). Critical cybersecurity hygiene: patching the enterprise. *National Institute of Standards and Technology*. https://www.nccoe.nist.gov/sites/default/files/legacy-files/ch-pe-project-description-draft.pdf

[33] Fuentes-García, M., Camacho, J., & Maciá-Fernández, G. (2021). Present and future of network security monitoring. *IEEE Access*, *9*, 112744-112760. https://ieeexplore.ieee.org/abstract/document/9381201

[34] Dubal, V. B. (2017). The drive to Precarity: A political history of work, regulation, & labor advocacy in San Francisco's taxi & Uber economies. *Berkeley Journal of Employment and Labor Law*, 73-135. https://www.jstor.org/stable/26356922