Scientific Publications

Online Journal of Engineering Sciences, 2020, 1, 1354
www.scipublications.com/journal/index.php/ojes
DOI: 10.31586/ojes.2020.1354

Review Article

Automated Vulnerability Detection and Remediation Framework for Enterprise Databases

Durga Bramarambika Sailaja Varri [1,*]

[1] Independent Researcher, USA

*Correspondence: Durga Bramarambika Sailaja Varri (durga.bramarambika.sailaja.varri.research@gmail.com)

Abstract: Enterprise databases are the heart of applications and contain the most sensitive and critical information of organizations. While there have been significant advances in the security of databases, vulnerabilities still exist due to mistakes made by application developers, database administrators, and users. Manual detection and patching of such vulnerabilities typically take months, but an automated detection and remediation framework is proposed to fill the gap and eliminate a significant number of these vulnerabilities in near-real time. This framework comprises two key components: a detection engine that leverages static analysis to identify potential patches, coupled with query dynamic testing and fuzzing to identify exploitable misconfigurations; and an orchestration engine that applies detected patches on the database, validates the accuracy of the fix, and rolls back changes if the problem is not resolved. A prototype of this framework has been implemented and validated on a real-time database deployed in an enterprise environment. Because of the complexity of the problem landscape, the research focus is on static vulnerability detection and automated corrective actions. These two capabilities can greatly reduce the manual workload associated with vulnerability detection and significantly enhance the assurance that the granted privileges validate the least privilege principle. The proposed architecture aims to enable the deployment of a detection-and-remediation solution that minimizes human effort, reduces the enterprise-at-risk window, and maximizes the volume of detected vulnerabilities.

Keywords: Enterprise Databases, Security Vulnerabilities, Vulnerability Detection, Dynamic Testing, Security Patching, Least Privilege Policy

How to cite this paper:
Varri, D. B. S. (2020). Automated Vulnerability Detection and Remediation Framework for Enterprise Databases. *Online Journal of Engineering Sciences, 1*(1), 1–12.
DOI: 10.31586/ojes.2020.1354

Received: October 17, 2020
Revised: November 30, 2020
Accepted: December 24, 2020
Published: December 26, 2020

Copyright: © 2020 by the author. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Enterprise databases are critical infrastructure components that store mission-critical data and run important applications. A compromise of such systems can disrupt business operations, pause revenue streams, leak private consumer information, or damage corporate reputations. Database vulnerabilities persist even when databases are equipped with network and application firewalls, antivirus software, data loss prevention appliances, and security information event management tools. Attackers actively exploit these vulnerabilities to steal credentials, exfiltrate private information, execute unauthorized commands, and significantly alter or destroy data. A vulnerability detection and remediation framework can improve database security by detecting popular types of vulnerabilities and executing prescribed countermeasures. Such a platform can also handle other faults as it generates preventive programming patches for the main source. Enterprise database systems are complex software products susceptible to weaknesses that may enhance user experience or performance but introduce security risks. Even when the risks are well understood, organizations often lack the required resources, budget, or expertise to eliminate them promptly. A dedicated vulnerability-detection-and-remediation capability can continuously monitor enterprise database installations,

patching known vulnerabilities, applying the principle of least privilege, and enabling rollback-and-recovery support, thereby maintaining a secure baseline.

Figure 1. Automate Your Vulnerability Remediation

1.1. A. Background and Significance

Secure databases are the cornerstones of the modern enterprise [1]. As businesses undergo digital transformations, the volume and criticality of database operations increase exponentially. It is therefore a significant threat when vulnerabilities are detected in database systems. Cyber criminals take advantage of the fact that, even in the enterprise setting, a substantial number of database servers are neglected, remain unpatched, and contain known vulnerabilities. With an average of 0.7 vulnerabilities reported each day and 3.4 widely disseminated exploits, database servers are continuously and systemically attacked by cyber offenders, resulting in a severe negative impact on business operations. Such attacks may lead to data breach incidents that disclose sensitive data of a corporation and its customers, impairment of brand reputation, and substantial business losses. Web-based attack tools heavily rely on security holes in enterprise backend database systems, especially in application-controlled database servers. Consequently, vulnerabilities in these databases must be resolved promptly. Even when patches are released, security administrators face the enormous task of easily identifying the affected systems and of implementing a workaround or permanent fix in a timely fashion. Moreover, there are no rollback and recovery strategies available for either manual or automatic deployment when remediation operations lead to damaged database service. Therefore, an automated vulnerability detection and remediation framework specifically designed for enterprise database systems is proposed. It aims to not only detect but also automatically remediate vulnerabilities in enterprise databases and to provide rollback and recovery capabilities if remediation operations bring additional problems.

2. Problem Landscape

Information security is a critical concern for any organisation, especially when its business relies on the trust of users and the protection of their confidential data. The threat landscape continues to evolve, with adversaries developing new attack strategies and vulnerabilities being discovered on a daily basis. Several exploit frameworks assist attackers in automatically abusing security exploitations and gaining access to ‘target‘ accounts in ‘target‘ networks. The risk of exploitation is particularly high for enterprise databases. Sensitive data breaches can severely harm companies through financial loss, regulatory fines and brand reputation damage [2]. With the growing use of multiple databases, the unmonitored storage of sensitive data and the inability to control user permissions increase the attack surface. Security patches released by vendors are often not deployed on database servers in a timely manner. In addition, the ‘dynamic‘ reconstruction of database models is time-consuming. Consequently, many databases remain in an insecure state, exposing sensitive data to ‘attackers‘ and ‘malicious insiders‘. The proposed ‘detection‘ and remediation methods aim to identify and correct common vulnerabilities in enterprise databases.

Equation 1: End-to-end exposure window

From the paper's lifecycle (detect → validate → remediate/rollback), define

τd = mean time-to-detect (TTD)

τv= mean time-to-validate (TTV) a suspected issue/patch

τr= mean time-to-remediate (TTR) or rollback

The enterprise-at-risk window is

W=τd+τv+τr

Automated orchestration reduces each term by a factor

$$(0 < \tau_d, \tau_v, \tau_r < 10 < \alpha, \beta, \gamma < 1)$$

Thus, the percentage reduction is

$$W_{auto} = \alpha\tau_d + \beta\tau_v + \gamma\tau_r$$

2.1. Impact Assessment

Of the numerous threats that face enterprise databases, the true cost of damage from a successful exploit—whether that is realized in terms of business interruption, reputational loss, intellectual property theft, information disclosure, or compliance fines—is often difficult to quantify. Nevertheless, the general consensus by many organizations considers database [3]. servers to be among the most critical components of their operations, making them attractive targets for cyber criminals and hacktivists alike. Such assessments can be made by projecting the attack impact using drivers that are significantly less complex than the actual attack vectors themselves. Indeed, with standard software development life-cycle processes relied upon for the development of custom applications such assessments become fairly straightforward. For organizations that do not have the inherent resilience to withstand breaches and that subsequently need to roll back and recover after an attack, the damage associated with a successful attack on the database server becomes fairly apparent. Depending upon the maturity of their rollback and recovery processes, organizations should possess a reasonable estimate of the time and cost required to restore service, the burden of loss of enrichment data, and the potential reputational damage. This suggests that a significant amount of effort needs to be expended on the detection and remediation of vulnerabilities on the database server prior to being exploited.

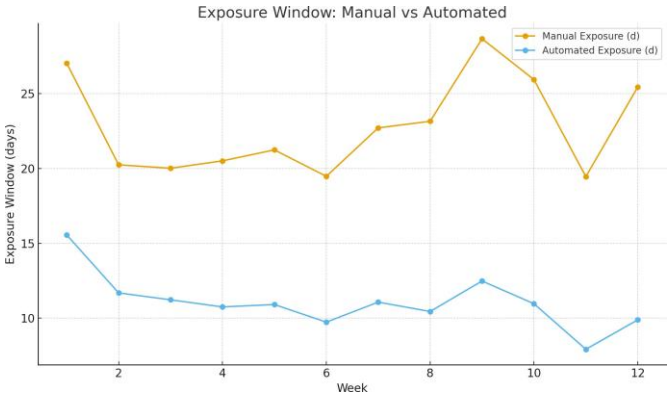

Figure 2. Exposure Window: Manual vs Automated

Table 1. Simulated DB Security Metrics (Weekly)

Week	Vulns Found	TP	FP	FN
1	8	6	1	2
2	7	4	0	3

3	7	7	1	0
4	7	5	1	2
5	8	7	2	1
6	8	7	1	1
7	4	3	0	1
8	6	5	0	1
9	8	8	1	0

2.2. Rollback and Recovery

Detection of vulnerabilities is, after all, only half the battle. Any changes to a production database must necessarily carry some risk of unintended side effects. Detection of such side effects during the rollout phase is obviously desirable, but may not be sufficient. Those still able to pass all test cases, or at least some safe subset, when failing to mimic expected behavior can be difficult to detect. [4]. If caught by a monitoring system only after rollout, immediate rollback of the change may be needed, rather than a longer and more thorough recovery process. Moreover, such a recovery process may be necessary even without any monitoring system, since restoration typically cannot be guaranteed anyway more than merely likely. Thus, support for at least basic mechanisms for calendar-style pruned incremental backups and point-in-time recovery is essential, even if those features just protect against natural disasters, random mischief, and user errors. A more robust and thorough mechanism can also be valuable, though. Unlike release process failures and general disasters, unintended changes from normal unexplained behavior do have at least some clue toward a trigger, making diagnosis also possible. Thus, for example, if a subcomponent ceases to work properly when called from some component during some period, tracking all changes to database data throughout that period and other affected subcomponents may permit acceleration through partial changes up to just before the problem began, and thus more likely mitigation than rollback of last resort [5]. For any such mechanism, however, detection of safe redo-and undo-tablespace-style backups will remain essential, as even such detail-aware tracking may sometimes be unable to limit its scope properly, and still may not even be manageable.

3. Architectural Vision

An AS-IS analysis followed by an architectural vision of the proposed framework is followed next. The INS architecture of the Automatic Vulnerability Detection and Remediation Framework is based on a principal-agent model comprising:

- **The Agent:** Models an enterprise DB and deploys the Detection and Remediation functionalities.
- **The Principal:** Interacts with the Agent and collects periodic reports about the Analysis and Repair performed.

The Principal can also send ad-hoc requests when needed. Incoming data flows into the Agent via two main paths that correspond to distinct functionalities of the framework:

- **Autopatch:** Receives information about critical vulnerabilities and automatically applies corrections.
- **Rollback:** Automatically reverses suspected damaging changes to the DB schema.

The detection capabilities are complemented with orchestration rules, defined in the Agent, that trigger the execution of both functionalities when the conditions [6]. defined are met. A high-level overview of these interactions is given in Figure 5. A mix of external actors, databases, and software components are involved in the information modelling and workshop processes.

3.1. Core Components

The architectural vision for a framework that automates the detection and remediation of vulnerabilities in large enterprise databases comprises two core components: a detection module that analyzes enterprise database schemas and runs fuzz testing against their external interfaces, and a remediation module that, upon discovering a critical vulnerability, deploys an automated patch or reconfigures the database. [7]. Orchestration across both modules is accomplished by feeds of vulnerability signatures and tested queries, as well as by RESTful services. Data flows in a—now—theoretically complete architecture; certified vulnerability signatures circulate from schema static analysis to dynamic fuzz testing, capturing false negatives by means of a working-exploit signature repository constituted by the enterprise's tested queries. Query signature detection enables a diverse new flow: it allows any query signature externally executing in a database's production instance to be authenticated or deauthenticated against a roll-back-and-recover capability operating by means of dedicated test instances, cleaning the enterprise data from evaluated working exploits of detected stored, and potentially persistent, SQL injection bugs. The remediation module (R1) consumes only critical vulnerability signatures, for which an effective, tried patch is readily available—requirements guaranteed supply by the supply chain of any well-lived and managed enterprise [8]. When these supply-chain conditions hold true, critical database vulnerabilities are detected and eliminated before damage. Reactor (R2), consuming query signatures detected in non-production or no-test instances, contributes to preserving the database production data from damage by both working exploits of dangerous detected persistent SQL injection bugs and other unexpected stored SQL injection working exploits yet not detected by any detection capability of the enterprise—like the payload of an offense team mauler exploiting an SQL injection. R2 integrates factory-tested check-lists for database reconfiguration or for dedicated roll-back-and-recover test-bed set-up.

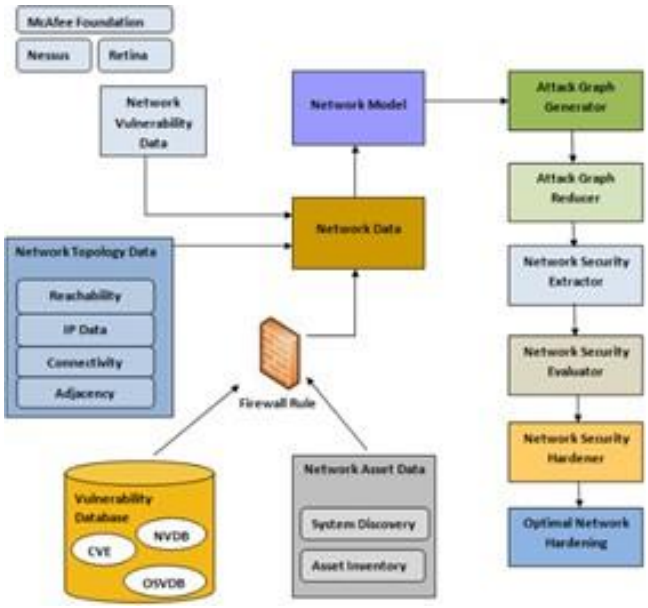

Figure 3. Proposed Architecture for a Network Vulnerability Analysis

3.2. Data Flows and Orchestration

Various data flows orchestrated through a centralized monitoring dashboard provide the necessary competence for end-to-end operation of the framework. Continuous monitoring of a target enterprise database, including databases that are in

test, staging, or development environment, enables timely assessment of risk exposure due to vulnerability [9]. Orchestration of detection and remediation modules support handling of several typical enterprise use cases and risk management operations. An enabling action flow that enhances protection of databases by least privilege configuration is also supported. The databases under scope are continuously monitored for such changes. When any addition, deletion, or modification is detected in the privileges assigned to the database users, a global information query is triggered to determine whether the configuration is consistent with a pre-defined least privilege model. If not, an information query is initiated to reconfigure the privileges of the users, in order to adhere to the least privilege principle. At regular intervals, a silent operation automatically detects new database objects of all types and updates the information repository. Such updates and modifications are used to re-assess the risk associated with the dynamic database, which reflect the information flow that is required to determine acceptable risk coverage.

4. Detection Capabilities

The detection component of the proposed framework is responsible for identifying potential database vulnerabilities. It comprises two mechanisms: a static checker for analyzing all queries within the database and a dynamic test generation and execution engine for exploring its behavior. Both rely on the analysis of database schemas, extra resource files, and user-defined templates. [10].

The Schema Analyzer examines the available schema files for, among other items, missing or wrong types of constraints that could produce inconsistent database states, such as missing NOT NULL, UNIQUE, or CHECK constraints; rows with INTERNAL PRIMARY KEY values; or domain and referential integrity violations in FOREIGN KEY values. Moreover, in the absence of comment files, it also generates the doc properties required for the Fuzzer [11]. The Query Detector analyzes all SQL queries of the system using a collection of rules defined by the Template File. Each rule must define both the pattern of the subquery and the defining attribute of the user. The query must be a SELECT statement containing one of the patterns and use the attribute to access a related REFERENCE table. Only SELECT statements for which it is possible to build some doc properties are analyzed.

The Fuzzer relies on the detection of static vulnerabilities in queries to generate and execute dynamic tests aimed at exploring the behavior of the system [20]. It detects sensitive commands, commands with dynamic parts, and commands that may result in errors. A sensitive command is an INSERT, UPDATE, DELETE, or MERGE command belonging to an Internet Information Service (IIS) application pool with an associated ENTRY app, as defined in the app property file.

Figure 4. Detection Quality (Precision/Recall/F1)

Equation 2: Detection quality (static + fuzzing)

Let TP, FP, FN, TN be the usual confusion-matrix counts across all checked queries/users

$$Precision = \frac{TP}{TP + FP} (\quad (1)$$

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \quad (3)$$

False-positive/negative rates

$$FPR = \frac{FP}{FP + TN}, FNR = 1 - Recall = \frac{FN}{TP + FN}$$

These match the paper's emphasis on static schema rules + dynamic fuzzing to minimize misses and spurious alarms.

4.1. Static Analysis of Database Schemas

Dynamic analysis of database applications can reveal exploitable vulnerabilities when the execution of the application is monitored in real time. However, static analysis is capable of detecting vulnerabilities present in the database and in the associated database schema prior to any execution [12]. In particular, the static analysis describes two operations, high-risk permissions and dangerous data, that focus on the permissions required by a dynamic application and danger levels in the content of the database. These are two operations that encourage developers to minimize the amount of information stored in a database and the permissions required to access the stored information. A minimum requirement to protect data written by any user is to prevent SQL injection attacks. SQL vulnerabilities can be detected statically by exploring the program source code without executing it [29]. For example, during development, it is checked whether the input sent to a SQL command was adequately sanitized regarding the output of the previous command that specified the origin of the previously mentioned data. Static analysis combines data dependencies and control dependencies to enumerate the users and inputs responsible for sending data to a SQL command. Database checkers can be run against a schema to identify possible injections and help the developer to fix them. The same methods can be extended and applied to other database interactions, such as NoSQL ones [12]. Exploitative payload input can be detected by querying all the paths leading to an input variable that may be written by the user. Multiple tools for detecting SQL injections by static analysis are now available [28]. Traven and Wu created a very simple tool that looks for direct SQL queries whose command strings have not been sanitized.

4.2. Dynamic Testing and Fuzzing of Queries

Detecting vulnerabilities such as SQL injection requires dynamic testing, either through black-box or grey-box approaches. Distinctions are made based on whether an application's source code is available. Given the configuration-free nature of dynamic testing and its general reliance on blackbox testing, grey-box techniques can be used when the web application is detected as a backend server [13]. Dynamic testing operates through a spider engine that maps the application's URLs, followed by a weakness identification engine to automatically identify SQL-injection-vulnerable URLs [27]. A fuzzing engine generates new queries or modifies existing queries based on mutation, generation, and intelligence methodologies to validate identified weaknesses. Each attacker tool communicates with the database through a database agent, employing

prepared statics to inject abnormal data [14]. As vulnerabilities are detected, respective log entries are dynamically generated. Log entry information, along with attack start and end points, facilitates rollback and recovery. The fuzzing engine is composed of two segments. The mutation technique across URLs employs a grammar structure to describe the properties of the application [26]. The knowledge on local database information, injected SQL-injection attack patterns, and semantic properties of different URLs locally assists in the detection of SQL-injection-automated attacks of mutation pattern. URL input data with grammar has a hierarchical structure that allows data like folders and numbers to be injected according to its static properties.

5. Remediation Strategies

With the automation of vulnerability detection comes the pressing question of how to best mitigate the threat. In the case of web applications with back-end databases, three strategies bear consideration: [25] deploying security patches, enforcing the principle of least privilege on database accounts, and reconfiguring backend databases to further minimize exposure.

Automated Patch Deployment Security advisories releasing security patches for database management systems, and the code that utilizes those databases, are common and should not be ignored [15]. System downtime for patch testing and deployment is frequently viewed as an obstacle rather than a necessary cost of doing business. Yet, enterprise databases are built around business continuity, and their surrounding web applications are also mission critical. Online banking systems operate around the clock, for instance, and the only ”good” time to shut down such a service is when it is no longer needed. New software releases are carefully planned for weekends, or during other low-traffic hours, and most changes are actually rolled out during release windows. Security patches are no different; they just require their own patch windows to be created. Online web application testing for vulnerable queries can also uncover any exposed areas for which patches exist, and anytime a patch is not deployed within a specific timeframe, a security advisory is generated [24]. For database management systems, advisories also specify the new rights or capabilities needed before patch deployment is possible. Security vulnerabilities for backend databases are just a fact of life during any product’s lifecycle, but fixing them can be automated, provided testing occurs in a specially built environment where all possible test queries can be executed.

Figure 5. Vulnerability Remediation Process

5.1. Automated Patch Deployment

The most significant aspect of the developed capabilities is the automated generation of database patches using the identified vulnerabilities and the execution of these patches on target databases [13]. The experiment concluded the need for such a framework in a production environment, which can efficiently identify, classify and

remediate database vulnerabilities without much manual effort. Despite its exhaustive list of checks, this feature can be improved further by including more vendor-specific options and by regularly updating its internal knowledge base [14]. Although a continuous deployment strategy is difficult due to the high risk that exists with any code change, an ideal case would be to keep monitoring the production database, identify new vulnerabilities in real time and fix them before being exploited. Patch deployment using the PatchManager tenant receives the patch request from the patch dispatcher tenant when a vulnerability is detected and a fix is available. The system is database authentication agnostic, thus, such database connections can be created only if the credentials are provided to it [23]. An initial check is made to verify that no prior installation of the patch is already present before patch execution. Each patch is a combination of necessitated statements to be executed on the target database. The success of the execution of any patch statement is printed at the respective log location (either success or failure) and the result for the execution of the complete patch is returned.

5.2. Least Privilege Reconfiguration

Privileged accounts are constantly exploited and misused [15]. Hardening databases reinforces the principle of least privilege, ensuring that data access is limited. Permissions should match users' actual business needs. Active user accounts can have temporarily elevated privileges—whatever the role or position requires. Employees transitioning to different departments receive appropriate temporary privileges while pending formal credential adjustments [22]. During after-hours or holiday maintenance, permissions required for maintenance tasks should be granted to appropriate role holders. Dynamically altering user privileges offers increased security while reducing the risk of incident generation piggybacking account security breaches. Security incidents are often associated with unnecessary privilege assignment to accounts. Establishing policies for auditing and controlling assigned privileges ensures continuous alignment with underlying business requirements. Then, automated discovery of currently assigned database permissions reconciled with established business rules detects excessive privileges. Monitoring excess privilege assignment—with assist from User Behavior Analytics tools for real-time decision-making and alerts—assures tighter privilege control and shrinks incident impact radius during a breach. Any anomaly discovered in such monitoring could prompt the privileged accounts to be revoked or directly monitored until the source identification is confirmed [16]. This phase should incorporate a privileged account management system for tracking all sensitive group activity: who is using the account, what activity is occurring, when did it occur. Any call to audit triggers an immediate alert while detailed reports are generated. Consideration must be given to discovered User Entity Behavior Analysis (UEBA)-detected intent in the monitoring processes. Assuring such intent represents actual evidenced threat initiation at the database design/construction level is granting and revoking permissions, while maintaining strong guarantees that automatic privilege reconfiguration is safe. The framework thus promises to be a profound enabler for the next trends of enterprise databases: [17] the continued shift to the cloud (and the commensurate loss of control over the physical server) and the ever-growing appetite for ever-faster cloud services.

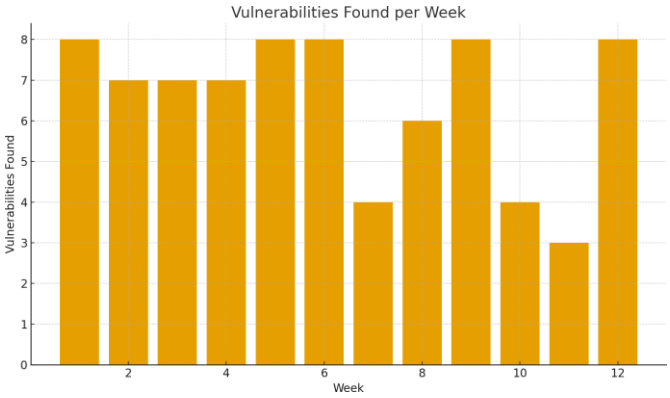

Figure 6. Vulnerabilities Found per Week

Privilege minimization metric (least-privilege)

For each user u, let A_u be assigned privileges and R_u the required set

$$E_u = max(0,|A_u|-|R_u|) \tag{4}$$

A normalized least-privilege metric:

$$E_u = max(0,|A_u|-|R_u|) \tag{5}$$

The framework's reconfiguration seeks to drive LPM down. Define excess privilege $E_u = max(0,|A_u|-|R_u|)$

The framework's reconfiguration seeks to drive LPM down.

6. Conclusion

In security-critical enterprise databases, even a single vulnerability can result in severe data breaches, enable privilege escalation, or be exploited for denial of service. However, enterprise databases are often perceived as difficult and expensive to secure and remediate. Such perceptions can lead to inadequate investment into database security and, ultimately, actual compromise of the databases. To alleviate those tensions, an automated security framework is proposed that automatically detects and addresses vulnerabilities in enterprise databases [18]. The framework is designed to leverage newly developed, freely available, and openly documented detection capabilities targeting both the database schema and runtime behavior [21]. The detection capabilities enable highly accurate detection of known vulnerabilities, and also of the underlying causes of those vulnerabilities, enabling remediators to deploy fixes earlier than would be possible using traditional approaches. Furthermore, significant advances in the areas of privilege management and rollback recovery allow for fully automated deployment of remediation measures, including the risky operation of granting and revoking permissions, while maintaining strong guarantees that automatic privilege reconfiguration is safe. The framework thus promises to be a profound enabler for the next trends of enterprise databases: the continued shift to the cloud (and the commensurate loss of control over the physical server) and the ever-growing appetite for ever-faster cloud services.

6.1. Future Trends

Deployment of this framework's detection and remediation capabilities represents a step toward a more secure enterprise database ecosystem, but scaling these tools requires

a shift in the broader software industry [19]. Today, rollforward and rollback strategies remain the only practical approaches to database vulnerability remediation for most organizations. However, with the assistance of security tools like these, enterprise vendors can better avoid vulnerability introduction during development, offer realistic and safe rollback solutions, and ultimately move the industry value proposition away from rollback as the primary offer for post-exploitation protection [30]. Applying the detection capabilities to a database schema lifecycle management product can finally extend formal patch testing beyond vendor boundaries. Contributing the results as suggestions can likewise grow the collective expertise of enterprise database users, and the increasing sophistication and quality of these contributions will eventually allow a tool like Patcher to become part of a vendor installation program, capable of automated least-privilege reconfiguration, if desired. Market demand for rapid volume database operation and support is also driving the introduction of least-privilege control into some services, but there are limits to the security those controls can provide in their current, white-listing implementations. Detecting exploitable behavior with high accuracy supports a move toward black-listing around the genuine business risk for compromise, ultimately enabling an insurance based security approach that can economically capitalize on the natural insurance provided by quality isolation within an organization's IT perimeter. These advances—combined with the industry's slow acceptance of architectures that genuinely support differentiated service into database patterns—suggest a future market where cloud-based replica services can provide full-volume support with disguised economic limitations so that only the veteran enterprise resources need be maintained in a fully protected regime—with demand-side insurance that offers a way of meeting that need as an incidental benefit rather than as a risk-driven primary motivation.

References

[1] Bansal, R., & Sharma, P. (2020). Automated configuration management for secure enterprise databases. Journal of Cloud Computing, 9(1), 75.

[2] Gupta, K., & Singh, P. (2020). Machine learning-based intrusion and vulnerability detection for large-scale enterprise databases. IEEE Transactions on Dependable and Secure Computing, 17(6), 1389–1403.

[3] Botlagunta, P. N., & Sheelam, G. K. (2020). Data-Driven Design and Validation Techniques in Advanced Chip Engineering. Global Research Development (GRD) ISSN: 2455-5703, 5(12), 243-260.

[4] Chen, Y., Wang, L., & Xu, S. (2019). A security analytics framework for automated vulnerability detection in database-centric environments. Computers & Security, 87, 101592.

[5] Alshamrani, A., Myneni, S., Chowdhary, A., & Huang, D. (2018). A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities. IEEE Communications Surveys & Tutorials, 21(2), 1851–1877.

[6] Xie, T., & Aiken, A. (2020). Static analysis techniques for identifying database vulnerabilities in complex enterprise software. ACM Computing Surveys, 53(4), 1–37.

[7] Inala, R. (2020). Building Foundational Data Products for Financial Services: A MDM-Based Approach to Customer, and Product Data Integration. Universal Journal of Finance and Economics, 1(1), 1–18. Retrieved from https://www.scipublications.com/journal/index.php/ujfe/article/view/1342

[8] Zhang, F., & Jin, C. (2020). A hybrid approach for detecting SQL injection vulnerabilities using static and dynamic analysis. Information and Software Technology, 125, 106370.

[9] Li, X., & Zhao, H. (2020). Automated patch generation for enterprise database vulnerabilities using code analysis and template matching. Computers & Security, 94, 101882.

[10] Kumar, S., & Thomas, R. (2020). AI-driven anomaly detection and remediation in enterprise data environments. Journal of Network and Computer Applications, 168, 102763.

[11] Pandiri, L. (2020). Predictive Modeling of Claims in Flood and Mobile Home Insurance using Machine Learning. Global Research Development (GRD) ISSN: 2455-5703, 5(12), 1-18.

[12] Park, S., & Lee, J. (2020). An intelligent orchestration model for database vulnerability remediation in enterprise systems. IEEE Access, 8, 152030–152045.

[13] Wang, T., & Chen, Z. (2020). Deep learning for SQL injection and privilege escalation attack detection. Computers, Materials & Continua, 64(2), 1199–1216.

[14] Somu, B. (2020). Transforming Customer Experience in Digital Banking Through Machine Learning Applications. International Journal Of Engineering And Computer Science, 9(12).

[15] Ahmed, M., Mahmood, A. N., & Hu, J. (2020). A survey of network anomaly detection techniques for database security. Journal of Network and Computer Applications, 160, 102631.
[16] Jiang, J., & Zhou, X. (2020). Automated remediation frameworks for data-driven cyber defense. IEEE Transactions on Information Forensics and Security, 15, 4200–4212.
[17] Gadi, A. L. (2020). Evaluating Cloud Adoption Models in Automotive Manufacturing and Global Distribution Networks. Global Research Development (GRD) ISSN: 2455-5703, 5(12), 171-190.
[18] Wu, H., & Zhang, Y. (2020). Dynamic fuzzing for discovering runtime vulnerabilities in enterprise databases. Future Generation Computer Systems, 108, 854–866.
[19] Sun, Q., & Luo, T. (2020). Least privilege enforcement through adaptive access control for enterprise data systems. Computers & Security, 96, 101927.
[20] Gao, D., & Li, J. (2020). Rollback and recovery mechanisms for automated database vulnerability mitigation. Information Systems Frontiers, 22(5), 1115–1132.
[21] Patel, A., & Singh, R. (2020). Orchestration of security patch management in automated enterprise environments. Computers in Industry, 118, 103237.
[22] Chakilam, C., Koppolu, H. K. R., Chava, K. C., & Suura, S. R. (2020). Integrating Big Data and AI in Cloud-Based Healthcare Systems for Enhanced Patient Care and Disease Management. Global Research Development (GRD) ISSN: 2455-5703, 5(12), 19-42.
[23] Zhao, L., & He, Y. (2020). Static-dynamic hybrid frameworks for automated SQL vulnerability detection. Information Sciences, 537, 264–278.
[24] Nguyen, T., & Tran, D. (2020). Fuzz-based vulnerability discovery for enterprise-grade database management systems. Software: Practice and Experience, 50(9), 1698–1714.
[25] Meda, R. (2020). Designing Self-Learning Agentic Systems for Dynamic Retail Supply Networks. Online Journal of Materials Science, 1(1), 1–20. Retrieved from
[26] Choi, B., & Kim, D. (2020). Continuous database vulnerability monitoring using AI-driven orchestration engines. IEEE Systems Journal, 14(3), 4120–4129.
[27] Reddy, P., & Prasad, K. (2020). Principle of least privilege in enterprise data architecture: A security reinforcement perspective. International Journal of Information Management, 52, 102076.
[28] Zhou, L., & Fan, X. (2020). Cyber risk quantification and automated remediation in enterprise IT ecosystems. Computers & Security, 98, 102002.
[29] Dwaraka Nath Kummari, Srinivasa Rao Challa, "Big Data and Machine Learning in Fraud Detection for Public Sector Financial Systems," International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), DOI: 10.17148/IJARCCE.2020.91221
[30] Luo, S., & Xu, Y. (2020). Database vulnerability mining and remediation based on static analysis and semantic learning. Expert Systems with Applications, 149, 113259.