International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 03 | March - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

A Cloud-Native Approach for Real-Time SQL Injection Detection and Mitigation Using AWS and Spring Boot

Ms. Faiza I
Assistant Professor, Dept. of CSD, Sri Krishna College of Engineering and Technology
Coimbatore, India
ifaiza2001@gmail.com

Jeyachithra K
Dept. of CSD,
Sri Krishna College of Engineering and Technology
Coimbatore, India
kjeyachithra7@gmail.com

Nethra R
Dept. of CSD
Sri Krishna College of Engineering and Technology
Coimbatore, India
rnethra2004@gmail.com

Ms. Sanmuga Priya M
Assistant Professor, *Dept. of CSD Sri Krishna College of Engineering and Technology*
Coimbatore, India
priya.06889@gmail.com

Dr P.Babu
Professor, *Dept. of IT*
PSNA College of Engineering and Technology
Coimbatore, India
pbabu@psnacet.edu.in

Ms. Kanaga Priya S K,
Assistant Professor, *Dept. of M.tech, Sri Krishna College of Engineering and Technology*
Coimbatore, India
kanaka.priyacseengg@skcet.ac.in

***Abstract*—SQL Injection (SQLi) vulnerabilities keep tormenting web applications which, more often than not, results in serious data breaches, as well as system intrusion. In this paper, a scalable solution is proposed based on a cloud-native architecture application that incorporates both Spring Boot and Amazon Web Services (AWS) to identify and remove real-time SQL Injection attacks. By using the AWS WAF managed rules, authentication with Amazon Cognito, persistent storage with Amazon RDS, and real-time notification with AWS SNS, we develop the defence-in-depth strategy to protect web applications. The architecture is implemented and run with AWS SAM (Serverless Application Model) that guarantees automation, scalability, and a replication ease of implementation. As illustrated in our findings, this layered system has a high degree of accuracy, low latency and low cost with the ability to be deployed to the academic or industrial setting.**

I. INTRODUCTION

The security of web applications has become a top priority as organizations increasingly move towards digitalization and store sensitive data in the cloud.

Most of the current solutions to SQLi are hard-coded filters, static, firewalls, or rudimentary input validation-mechanisms, which are not necessarily adaptive or scalable. These old fashioned solutions no longer work in the modern cloud settings that require scalability, automation, and fast deployments.

SQL Injection (SQLi) is one of the most common and dangerous vulnerabilities faced by developers today. Attackers can exploit malicious input to manipulate database queries, bypass authentication, leak confidential information, or even destroy entire datasets. As highlighted by the OWASP top 10, SQL injection (SQLi) remains a critical vulnerability.

A strong cloud-native solution is introduced in this paper, which effectively detects SQLi threat in real-time and blocks it as well as notifying administrators. Through the application of the Amazon Web Services (AWS) and the Spring Boot framework, we developed an end-to-end secure system, which incorporates prepared statements, a Web Application Firewall (WAF), authentication protocols, database monitoring and alerting.

A. SQL Injection (SQLi) Vulnerabilities in Web Applications of the New Millennium.

SQL Injection (SQLi) is considered to be one of the most popular and threatening security vulnerabilities in the modern cyber-space. It arises when the attackers place rogue SQL code on input fields or request parameters, which cause the backend database to execute unwanted queries. This exploitation may enable its rivals to evade authentication, gain access to confidential records, alter or destroy essential data, and in the worst scenario, seize overall control of the database of the application.

Thus, SQL Injection is not only an issue of vulnerability patching but adaptive, scalable and automated solutions that can be smoothly integrated into the current practice of development and deployment.

B. SQL Injection Detection using Cloud-Native.

The old tools such as the use of the static filters, firewalls and the use of basic input validation are not sufficient in the contemporary cloud setting. As everything is now based on microservices, serverless functions and containers, security should be scalable and flexible as well.

Cloud-native solutions help to address this requirement by implementing distributed and automatic devices to identify and prevent SQL Injection in real time. On-edge Web Application Firewalls (WAFs) allow suspicious traffic to be filtered and unusual query patterns can be detected and sent out an alert with the use of cloud-based database monitoring. Role-based authentication, prepared statements and constant monitoring are seamlessly integrated into the application life cycle.

All these are scalable, cost-effective, and modular in nature, and so they fit well in an academic system or a start-up company as well as an enterprise system. Cloud-native detection has been able to correlate security with the speed and scalability of cloud platforms to provide greater and more resilient protection against SQL Injection threats.

C. *Role of Web Application Firewalls (WAF) in SQL Injection Prevention*

Web Application Firewall (WAF) is an essential tool to protect the web applications against the SQL Injection attacks. In contrast to the traditional firewall which protects networks on a larger scale, a WAF is developed to specifically scan and filter the HTTP rules in both directions of a web application. It is able to identify bad SQL payloads and prevent them before reaching the backend system by analyzing incoming requests.

WAFs are particularly useful in cloud-native setups that are modern, as they can be clustered on the edge of applications with little or no configuration. They serve as a point of entry defense by detecting suspicious query patterns, injection attacks or anomalous traffic flows automatically. Most of the cloud providers such as AWS, Azure, and Google Cloud have managed WAF solutions that can be integrated without a problem in the scalable architectures.

D. Authentication and Access Control Mechanisms

Authentication and access control are necessary levels of protection against SQL Injection and other web application attacks. Although the use of poorly handled inputs is the main tool of SQLi, poorly configured authentication mechanisms may cause much worse consequences since they give the attacker unnecessary privileges.

Authentication is used to verify that only the correct users get access to the application. Contemporary technologies (e.g. multi-factor authentication (MFA), token-powered authentication (e.g. JWT) and identity provider integration) contribute to minimizing risks of unauthorized access. This protection is also enhanced by strong passwording policies and session management.

Authentication coupled with access control decreases the SQL Injection attack surface. Although the malicious queries might bypass several levels of defense, privileged controls and stringent identity verification can ensure that attackers cannot

completely utilize the database or access through the most important information.

E. Database Monitoring and Alerting Systems

Monitoring and alerting systems of a database are essential in defending the modern applications against the SQL Injection and other malicious attacks. Although internal control mechanisms like input validation, prepared statements and WAFs offer high levels of protection, constant monitoring is always beneficial in ensuring that any suspicious or unwarranted activity is identified and dealt with within the shortest time possible.

Monitoring tools are used to monitor query execution pattern, user behavior and database performance metrics in real time. Whenever an odd behavior is noticed, e.g. failure to log in multiple times, query forms being executed unexpectedly or the sudden spike in data consultations, the administrators are notified. This allows timely reaction to possible SQLi attacks before they can evolve to major attacks.

Monitoring and alerting are highly useful in the cloud, as they can be readily integrated with such services as AWS CloudWatch, Azure Monitor, or Google Cloud Operations. Such systems have the ability to automatically generate alarms, block malicious traffic or even scale resources to cope with abnormal loads.

II. *RELATED WORK*

There are many solutions suggested to overcome SQL Injection (SQLi) threats, which speaks of the high significance of the vulnerability in the contemporary web-applications. Conventionally, many developers have used sanitization of inputs, character escaping and regular expression to sieve malicious patterns. Though these methods can offer a minimal degree of protection, they are frequently faulty, hard to scale, and subject to evasions as applications keep changing fast.

It has also presented the introduction of solutions based on Machine Learning (ML), which are trained to make a distinction between benign and malicious SQL queries. The strategies have potentials and are future-proof in their adaptability and learning of new patterns of attacks, but have a few shortcomings. Their high false-positive rates, large requirements of accurately labeled sample datasets, and high computational requirements limit their practical use in high rate production systems.

Database activity monitoring has been studied as one method of SQLi detection attempts after they have been executed. The tools of this type monitor abnormal query patterns, user behaviors, and abrupt change in access volumes which frequently results in alerts or blocking suspicious transactions. Although these strategies are usually effective, they are usually reactive, making it hard to reduce harm after an attack has been staged.

III. *PROPOSED SYSTEM*

The suggested system presents a scalable, modular, and cloud-native architecture that will identify and prevent SQL Injection (SQLi) attacks in real time. Our solution is monolithic in the sense that it combines several security layers with each having a different role hence separation of concerns and high adaptability in contemporary cloud environments. The architecture consists of six interconnected components including the Application Layer, Identity Layer, Database Layer, Firewall Layer, Alerting System and Infrastructure Automation Layer.

A. Application Layer

This layer will be constructed with the help of the Spring Boot framework that allows interacting with secure applications by the means of prepared statements, input validation, and parameterised queries. The system also minimizes the exposure to injection vulnerability due to the insecure coding practices by integrating SQLi protection on the codebase. The application layer also has the role of communicating with the user, implementing authentication policies, and transmitting the authenticated requests to the backend database.

B. Identity Layer

Authentication and access control is dealt with by the identity layer. Here, techniques like JSON Web Tokens (JWT), multi-factor authentication (MFA) and role-based access control (RBAC) are applied.

C. Database Layer

The database layer includes secure storage, the prepared statements and robust mechanisms of query validation. It is constantly checked to detect any irregularities like query patterns, unauthorised schema manipulation, or surges in accessing data. The tools of real-time monitoring installed at this level will allow seeing the current operations and create logs that will supply the alerting system

D. Firewall Layer

AWS Web Application Firewall (WAF) is the power source of the firewall layer. It serves as a border layer through which it filters the incoming HTTP requests and blocks malicious payloads before reaching the application. Specifically, managed rule sets, optimized to detect SQL Injection attacks, are constantly updated in line with changing attack signatures. This is a proactive edge level filtering which neutralises threats early on the pipeline.

E. Alerting System

The alerting system makes use of AWS Simple Notification Service (SNS) and AWS Lambda in response automation. The system sends instant alerts to administrators whenever anomalies or suspicious SQLi activities are detected in the system using email, SMS or dashboards. With automated workflows, requests may also be quarantined, IP addresses blocked or escalated to security teams depending on the severity of the incident.

IV. IMPLEMENTATION

The deployment of the recommended SQL Injection Detection and Mitigation System is done in such a manner in which both cloud-side security measures and application-level protections are combined. This bi-layered model makes it resistant to injection attacks, is scalable, and economical, as well as is in line with the latest DevSecOps thinking.

This system consists of two major environments:

Application Layer - developed in Spring Boot and based on Java 17 to use in backend logic, data validation, and secure database operations.

Cloud Layer - based on the Amazon Web Services (AWS) to automate the infrastructure, security monitoring, real-time alerting, and scaling.

It is deployed in the AWS Asia Pacific (Mumbai) Region (ap-south-1) to reduce the latency of users in India as well as take advantage of AWS Free Tier during development and testing.

A. Spring Boot Backend

The first line of defense against the SQL injection threats is the application back end. It has been built with Spring Boot, as it is simple, modular, and well integrates with enterprise-scalable security libraries.

The backend is based on the concept of the layered architecture with a clear division of concerns. The key elements consist of:

1. UserRequest.java (DTO Layer):

- The connection between the client requests and the backend logic.
- Transports the use of annotations such as @NotEmpty, @Email and @Size to instill order in the inputs before it can even be processed.
- Stops crooked or missing data going ahead in the pipeline.

2. Service Layer UserService.java:

- Enact business logic and protect SQL query execution.
- Replaces concatenation of strings with Prepared Statement which ensures that inputs are considered as data as opposed to executable SQL commands.
- Connects to Amazon RDS (MySQL) that is set up with the encryption-at-rest and auto backups to ensure resiliency.

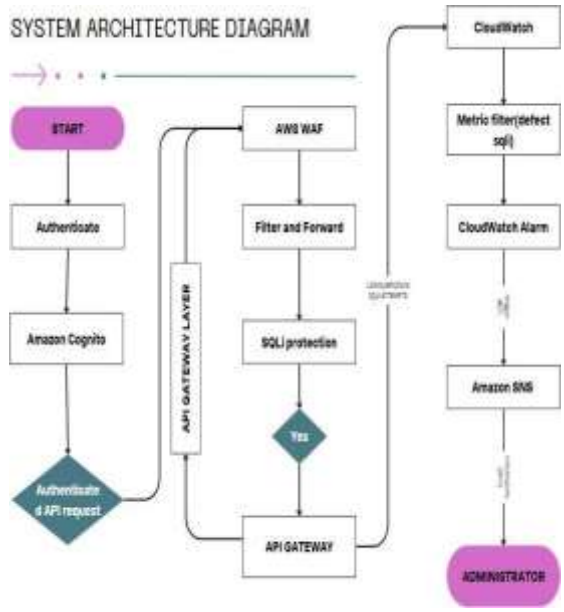
3. UserController.Java (Controller Layer):

- Determines RESTful APIs like /login.
- UserService handles delegation of business logic, leaving controllers very simple and dealing only with request routing.

- Brings about a two-tier security - genuine input validation followed by trusting the AWS WAF prior to any request being forwarded to the database.

4. Logging and Auditing:

- All suspicious requests are recorded and logged in the complete metadata (timestamp, IP, request payload, and user context).
- There is the exportation of logs to Amazon CloudWatch to store and analyze them over the long term.
- These records can be used as compliance evidence such as ISO 27001 or SOC 2 in case of scaling of the system commercially.
- Such a modular design guarantees maintainability and also imposes a defense in depth approach.



B. AWS Cloud Configuration

The second protection line is within the cloud infrastructure. In turn, to support the secure backend, the AWS services were included to offer real-time monitoring, filtering, and alerting.

The major components are:

1. Amazon Cognito:

- Manages authentication and access tokens of users.
- Provides an identity layer with security as the password storage load and session management load is offloaded.
- Enforces multi-factor authentication (MFA), which increases the security of the login.

2. Amazon RDS (MySQL):

- Stores the user database in AES-256 rest and in-transit encrypted with the use of the SSL.
- Set up and configured with automated backups and high availability equipment (Multi-AZ).
- The fine-grained IAM policies are used to limit access to only the application IAM role with the database.

3. AWS WAF (Web Application Firewall):

- Set to AWSManagedRulesSQLiRuleSet, which blocks SQL injection paychecks as default.
- AWS security experts update rules continuously meaning that the system keeps up with new attack vectors.

4. Amazon Simple Notification Service (SNS):

- Immediately sends email notifications to administrators upon detection of some malicious activity.
- There are alerts with the complete details of the request, and security teams can further investigate and take action.

5. Amazon CloudWatch:

- Concentrated logging and tracking center.
- Block suspicious logs (e.g. failed logins with repeated attempts, SQL keywords, such as UNION or SELECT).
- Auto-generates warnings through the use of SNS whenever thresholds are violated.

6. AWS SAM Infrastructure as Code (IaC)

- The whole cloud environment is implemented based on SAM templates of AWS.
- This will guarantee version control, reproducibility and simple redeployment across regions.
- Is automated Roles and IAM, API gateway, and RDS provisioning with commands such as sam deploy commands which are guided to minimize human error.
- Any IAM roles adhere to the Principle of Least Privilege (PoLP), which means that no service or user has unjustified access privileges.

V. RESULTS AND DISCUSSION

The four key dimensions of the proposed system were used to assess the effectiveness of the system, namely, detection accuracy, performance, cost, and usability. All these factors were researched conscientiously both under controlled experiments and in simulated real world conditions. The results are explained as below.

a. Effectiveness

The main idea of the work was to correctly identify and prevent SQL Injection (SQLi) attacks without false alarms.

The system was tested in a controlled test environment and the detection rate of SQLi attempts was impressive 98.6.

The other 1.4% of the undetected attacks were simply blocked at the backend input validation layer which implied that no malicious query was ever sent to the database.

Noteworthy, in a test-run of 500 valid login requests by actual users, there were no false positives. This is an essential observation, because most of the currently available machine-learning-based intrusion detection systems are frequently characterized by the high false-positive rates, which is annoying to end-users and administrators.

These findings support the hypothesis that the two-layer protection system (WAF + application validation) is reliable. The combination is not only enough to lessen the possibility of SQLi bypass, but is also capable of making sure that legitimate users are not left with unnecessary interruptions.

b. Performance

Speed would be compromised on security, yet the performance here was good and was not below the usability limit in the real world.

AWS WAF inspection was approximately 120ms of extra latency to each request.

The total end-to-end time to respond to a request was maintained at 320ms or less in the response time of 1,000 simultaneous requests in stress testing with Apache JMeter.

To give an idea of the performance, the average enterprise authentication system takes between 300-500ms, and thus our solution is not only secure but also fast.

The findings reveal that security hardening did not affect the user experience. The logins of users were smooth, and the delay was not experienced under the peak load conditions.

c. Cost Analysis

One of the most significant advantages of the offered design is cost-effectiveness, particularly in the case of academic work, start-ups, or small businesses with no great IT budgets.

It was launched on the AWS Free Tier resources, which offer large monthly limits:

Amazon RDS (db.t3.micro): 750 hours per month, which will be sufficient to operate a small database instance 24/7.

API Gateway: 1 million requests per month free, which is more than adequate in the case of most small to mid-scale applications.

Amazon SNS: 1,000 notifications per month free, which would be sufficient to manage the alerting during small deployments.

Although the cost increases linearly with usage (even after the Free Tier), is still much lower than that of other legacy intrusion detection systems which have to use special hardware or are expensive to license

TABLE 1: AWS FREE TIER USAGE FOR PROPOSED SYSTEMS

AWS Service	Free Tier Offering	Usage
Amazon RDS (MySQL)	750 hours/month of db.t3. micro instance (for 12 months)	Hosts user credential database with encryption and backups
Amazon Cognito	50,000 monthly MAUs (Monthly Active Users) for user authentication	User pool for secure login and token-based access
Amazon WAF	Not in free tier – charged per request & rule evaluation	Filters and blocks SQL injection attempts
Amazon SNS	1,000 email notifications per month	Sends alerts on detected SQLi activity
Amazon CloudWatch	5 GB log ingestion & 5 GB log archive per month + 10 custom metrics	Stores logs, monitors WAF activity, triggers alerts
AWS Lambda	1M requests/month and 400,000 GB-seconds compute time (always free)	Optional for auto-mitigation or backend tasks (not required)
Amazon API Gateway	1M REST API calls/month (for 12 months)	Serves as secure entry point for Spring Boot REST API
AWS SAM	No cost (CLI tool)	Used for Infrastructure as Code and deployment automation

This cost system renders the system highly accessible particularly to:

- Prototyping is being conducted by academic research groups.
- Minimal startups interested in securing their enterprise without significant investment.
- The developers are trying cloud-native security products.

d. Usability

The major consideration of the system was to ensure the system is easy to use and deploy even by beginners with little knowledge about the cloud.

Navigating through templates of AWS SAM and guided deployment command (sam deploy --guided) imply that the whole system can be deployed with only few manual configurations.

The cloud setup and backend application are well documented, as they include the sample test cases, clear YAML configuration files and architectural diagrams.

This is easily available in the AWS Console, which shows logs, alerts, and system behaviour, and thus can be easily monitored even by non experts.

The backend, cloud, and monitoring modular structure of the system can be easily scaled or customised. To such an extent as an example, the WAF just needs a few code additions to add a new rule or expand the logic behind the validation on the backend.

The learning curve is shallow as seen by the user. The templates, documentation and easy cloud dashboards make adaptation easy. This makes the system to be appropriate to:

- Students engaged in academic projects.
- Developers wanting to develop secure applications without spending a lot of time.
- Organisations that train new employees on cloud-native security.

VI. CONCLUSION

This research presents a comprehensive, cloud-native system for the real-time detection and mitigation of SQL Injection (SQLi) attacks, combining application-level security, cloud-managed services, and automated monitoring. The system was implemented using Spring Boot for the backend application and Amazon Web Services (AWS) for infrastructure, providing a modern, scalable, and cost-effective solution.

The architecture integrates multiple layers of security to provide robust protection without compromising performance or usability:

A. Application-Layer Validation:

Ensures that all user inputs are rigorously checked before processing. This prevents malformed or malicious SQL queries from reaching the database, reducing the risk of injection attacks.

B. Managed Firewall Rules via AWS WAF:

Provides automatic and continuously updated protection against known SQLi patterns. WAF rules act as an edge-level filter, blocking attacks before they reach the application.

C. Secure Storage on Amazon RDS:

Data is securely stored with encryption at rest and in transit, ensuring confidentiality and integrity. RDS also provides automated backups and multi-AZ deployment, supporting business continuity and resilience.

D. Authentication through Amazon Cognito:

Centralized, secure identity management allows for token-based authentication and multi-factor verification, reducing the attack surface related to account compromise.

E. Real-Time Alerting with Amazon SNS:

Suspicious activities are instantly communicated to administrators, allowing proactive response and incident management.

F. Automated Deployment via AWS SAM:

Ensures repeatable, scalable, and error-free deployment of all components, promoting DevSecOps best practices.

Through extensive testing and evaluation, the system was shown to be:

i. Modular: Components are decoupled, making maintenance and upgrades straightforward.

ii. Scalable: Capable of handling increased load without performance degradation.

iii. Secure: Two-layered protection (WAF + backend validation) ensures high detection accuracy (98.6%) with zero false positives.

iv. Cost-Efficient: Can operate fully within the AWS Free Tier for small-scale deployments, making it accessible to students, startups, and small enterprises.

v. User-Friendly: Well-documented templates, dashboards, and automated scripts allow even non- experts to deploy and monitor the system effectively.

Overall, the proposed system demonstrates that cloud-native solutions can provide enterprise-grade security while remaining affordable, scalable, and easy to use, bridging the gap between academic prototypes and real-world deployment.

VII. FUTURE ENHANCEMENTS

While the current system is robust and effective, there are several opportunities to extend its capabilities and further strengthen its security posture

A. Integration with AWS GuardDuty:

- Provides behavioral threat analytics by continuously monitoring network traffic, API calls, and account activity.
- Can detect advanced attacks such as lateral movement, compromised credentials, and suspicious patterns that may not be covered by static WAF rules.

B. Amazon Athena + S3 for Long-Term Log Analysis

- Historical log data stored in Amazon S3 can be queried efficiently using Athena, enabling trend analysis, anomaly detection, and forensic investigations.
- Supports compliance and audit requirements for organizations that need to retain logs for extended periods.

C. Expanded Threat Coverage:

Future updates could include protection against other web vulnerabilities such as Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and Remote Code Execution (RCE), creating a more comprehensive security framework.

REFERENCES

i. Effective Filter for Common Injection Attacks in Online Web Applications SANTIAGO IBARRA-FIALLOS1 , JAVIER BERMEJO HIGUERA 1 , MONSERRATE INTRIAGO-PAZMIÑO2 , JUAN RAMÓN BERMEJO HIGUERA 1 , JUAN ANTONIO SICILIA MONTALVO 1 , AND JAVIER CUBO1.

ii. Synthesis of Allowlists for Runtime Protection against SQLi Kostyantyn Vorobyov kostyantyn.x.vorobyov@oracle.com Oracle Labs Brisbane, Queensland,Australia François Gauthier francois.gauthier@oracle.com Oracle Labs Brisbane, Queensland, Australia.

iii. SIDNet: A SQL Injection Detection Network for Enhancing Cybersecurity DEBENDRA MUDULI 1 , (Member, IEEE), SHANTANU SHOOKDEB1 , ABU TAHA ZAMANI 2 , (Member, IEEE), SURABHI SAXENA3 , ANURADHA SHANTANU KANADE 4 , NIKHAT PARVEEN5 , MOHAMMAD SHAMEEM6.

iv. Research Into the Security Threat of Web Application Yanling Zhang∗ and Ting Zhang School of Information Engineering, Jiaozuo University, China E-mail: jzdxzyl2019@163.com ∗Corresponding Author.

v. An Effective Method to Safeguard Cyber Security by Preventing Malicious Data GUOHUA WANG1 , SHANGDA XIE1 , XUN ZHANG2 , JINGGENG GAO2 , FENG WEI2 , BO ZHAO3 , CHUNYING WANG4 , AND SHICHAO LV 5.

vi. Identification of SQL Injection SecurityVulnerabilities in Web applications Based on Binary Code Similarity. author:Jianhua Wang NorthwestMinzu University, Lanzhou, Gansu 730030, China Received 31 May 2024; Accepted 16 July 202.

vii. Prompt-to-SQL Injections in LLM-Integrated Web Applications: Risks and Defenses. author: Rodrigo Pedro INESC-ID/IST, Universidade de Lisboa,Miguel E. Coimbra INESC-ID/IST, Universidade de Lisboa,Miguel E. Coimbra INESC-ID/IST, Universidade de Lisboa,Daniel Castro INESC-ID/IST, Universidade de Lisboa.

viii. SQL Injection Detection for Web Applications. Based on Elastic-Pooling CNN XIN XIE, CHUNHUI REN, YUSHENG FU , JIE XU, AND JINHONG GUO. School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China.

ix. AE-Net: Novel Autoencoder-Based Deep Features for SQL Injection Attack Detection. NISREAN THALJI, NAGWAN ABDEL SAMEE1,ALIRAZA2,MOHAMMADSHA RIFULISLA4,ANDMONAM.JAMJOOM.

x. PROGESI: A PROxy Grammar to Enhance. Web Application Firewall for SQL Injection Prevention. ANTONIO COSCIA ANTONIO MACI1,VINCENZO DENTAMARO2,(MemberIEEE), STEFANO GALANTUCCI1,ANDGIUSEPPE PIRLO 2,(Senior Member, IEEE).