International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Automated Security Testing Framework for Web Services: A DevSecOps-Integrated Approach

Mohnish Neelapu

neelapu1001@gmail.com

Abstract

The advanced nature of web services creates security weaknesses such as SQL Injection (SQLi), Cross-Site Scripting (XSS) and API exploitation which threaten both data reliability and system stability. This research introduces the Automated Security Testing Framework (ASTF) to bring together different security testing methods within the DevSecOps development pipeline for web application security enhancement. Vulnerabilities get discovered in real time by Dynamic Application Security Testing (DAST), static Application Security Testing (SAST) which works alongside penetration testing and fuzz testing through their integration of OWASP ZAP, Burp Suite, Acunetix, SonarQube and Snyk tools. Application of AI security monitoring with continuous threat analysis optimizes security risk mitigation through reduced false positive incidents to 6% and it enhances security response efficiency. An evaluation of an e-commerce platform proves that its 90% decreased high-risk vulnerability exposure sustains development agility alongside ISO 27001 and GDPR compliance. The research showcases ASTF because it detects threats efficiently and handles automated patching as well as its easy CI/CD integration which protects modern web services actively.

Keywords: Automated Security Testing, AI-powered Threat Response, Cyber Threat Intelligence, Zero-day Vulnerability Detection and Security Automation in Cloud.

I. INTRODUCTION

Organizational security concerns have increased as more companies shift toward web services and cloud applications in their operations [1-3]. REST ful APIs with microservices architectures in modern applications create security risks that extend to SQL Injection, XSS vulnerabilities and CSRF attacks together with improper authentication procedures [4-5]. Hackers take advantage of security weak points in systems which cause destructive outcomes that damage both financial assets and public images of organizations [6-7]. The combination of stand-alone static and dynamic analysis tools that includes manual penetration testing is inadequate because these methods require human experts who need extensive time to monitor systems through automated testing procedures [8].

The adoption of Agile and DevSecOps development approaches requires security testing to become a native element of Software Development Life Cycle (SDLC) processes in order to identify and resolve vulnerabilities during the first stages [9-10]. ASTF framework supports real-time security analysis through its combination of SAST, DAST, API fuzz testing along with AI-powered threat detection mechanisms in a development environment [11]. Application security becomes strengthened while deployment vulnerabilities become identified through these frameworks that help decrease remediation

costs. Deploying automated security testing reduces wrong positives of vulnerabilities and speeds up detection of vulnerabilities and strengthens system resistance to cybersecurity attacks [12].

A. Research Objectives and Problem Statement

The goal of this research is to build an ASTF which adds security testing features to DevSecOps pipelines for better defense of modern web applications. The key objectives are:

- To design a security framework that combines SAST, DAST, fuzz testing, and AI-based security analytics.
- To automate security testing in CI/CD workflows, ensuring vulnerabilities are detected and remediated before software release.
- To evaluate the effectiveness of ASTF by comparing it against manual penetration testing and standalone security tools.
- To analyze performance metrics such as Vulnerability Detection Rate (VDR), False Positive Rate (FPR), Mean Time to Detect (MTTD), and Mean Time to Remediate (MTTR) to assess efficiency.

B. Scope of the Study

The research conducts web application and API security operations using ASTF technology deployed within DevSecOps platforms. The test environment selects an e-commerce platform which operates with Java (Spring Boot), Node.js and React.js technical stacks. Security experts analyze several security threats that affect web applications through injection exploits as well as authentication and API interface vulnerabilities. The research tracks how the framework affects testing performance, detection effectiveness as well as system operational speed. This research implements automated security testing into CI/CD for providing real-time scalable security tests that build better application security, reduces remediation time while upholding GDPR and ISO 27001 compliance requirements.

II. LITERATURE REVIEW

Security testing automation according to SrinivasaRaoVemula et al. [1] improves operational effectiveness, vulnerability recognition and addresses issues stemming from manual inspection methods because they require extensive time and contain human error. This system detects security threats immediately while decreasing human'serror and maintains continuous security compliance checking to improve cyber safety. The system depends on predeveloped test criteria yet fails to detect new attack sequences and produces invalid results that demand additional human assessment.The authors Nikhil Rane and AmnaQureshi [2] conduct an assessment of web security measures that combines manual penetration testing alongside automated vulnerability detection systems. The automation of network scanning provides speed while scanning an extensive area without a lot of human involvement yet generates multiple false alarm results. Manual penetration testing gives precise results through its real-world attack simulation of complex vulnerabilities at the cost of extensive time requirements and resources. However this approach requires penetration testers to have expertise in identifying security flaws. The web application security can be boosted through systematic vulnerability detection using the penetration testing framework introduced by Shilpa R. G [3]. This framework delivers better security testing results through structured information representation while other methodologies fail to provide such a structured system. The method creates penetration tests by using state models which produce automated tests and manual penetration tests to provide better vulnerability identification during structured attack assessments. The key benefit emerges from this method because it achieves

comprehensive vulnerability detection and automatic attack generation which improves the effectiveness of security testing. The main drawback of model-based approaches lies in their strict dependence on precise application behavior modeling since any modeling inaccuracies can result in security assessment failures along with potential wrong positive results. Alhamed and Rahman [4] demonstrate that network penetration tests serve as critical components for discovering security vulnerabilities which protect systems from cyber threats. Network security functions are possible through risk identification which occurs during design phase as well as operational period and implementation phase. Manual testing of security breaches proves disadvantageous because it takes too much time but provides proactive protection from security breaches despite potential missing vulnerabilities.

III. BACKGROUND

A. Security Threats in Web Services

Web services experience multiple security threats through which cyber attackers attempt to compromise data security integrity together with confidentiality and availability. Multiple critical web service security vulnerabilities included in OWASP Top 10 comprise Broken Object Level Authorization (BOLA), Broken Authentication, Injection Attacks, Security Misconfigurations, Server-Side Request Forgery (SSRF) and other essential components. The present vulnerabilities enable attackers to control APIs and gain unauthorized privileges while delivering harmful code. Those who engage in threat modeling establish this vital security measure to detect threats during vector analysis and build security improvements. Application developers use STRIDE and DREAD methodologies to protect their applications by identifying potential threats through Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege methods. Three major malicious entry methods which attackers utilize include SQL Injection (SQLi), Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF). Through SQLi attackers gain the ability to change database query commands for unauthorized access of data and unwanted data modifications. The open vulnerability of XSS enables attackers to insert harmful scripts that result in stolen user sessions and unauthorized access to confidential data. Authentic users can execute undefined commands through CSRF attacks leading to modifications in essential application data. Attractive online security requires comprehensive protection from authentication systems that employ input validation with API rules and instant system evaluations.

B. Automated Security Testing Techniques

Web services use automated security testing to protect their vulnerabilities by analyzing code to detect threats which enhances their security features. SAST evaluates static components of source code, bytecode along with application binaries to identify security issues that range from hardcoded credentials to insecure data handling and improper access controls. DAST testing evaluates operational applications to detect current security vulnerabilities which include SQL injection and XSS along with authentication-related issues. The integration of authorization testing automation allows DAST to execute ethical hacking methods which replicate real hacking attacks to uncover system weaknesses that standard testing methods cannot identify. Web application testing through fuzz testing employs automated tools to provide web applications with random faulty or suspicious data for the detection of exploitable system failures and memory-based security threats. In security testing artificial intelligence along with machine learning serves as an appropriate solution by developing learning models to both detect patterns and predict vulnerabilities before executing automated security assessments. The

intelligent systems improve traditional testing through security concern detection and risk priority management and threat recognition updates for swift response to emerging threats that strengthen web service protection.

C. Security Testing Tools for Web Services

Web services utilize their automated security testing system to protect against vulnerabilities by analyzing code which detects threats for better security outcomes. SAST analyzes static components of source code, byte code and application binaries to uncover security issues which start from hardcoded credentials and extend to insecure data handling and improper access controls. DAST tests operational applications to detect current security vulnerabilities which include SQL injection, XSS and authentication-related issues. The automation of authorization testing enables DAST tools to collaborate with ethical hacking techniques for realistic vulnerability detection that standard testing tools cannot find. The automated process of fuzz testing web services through web application testing reveals exploitable system failures and memory-based security threats by inputting random faulty or suspicious data. The security testing solution based on artificial intelligence and machine learning capabilities creates effective models to collect patterns for vulnerability prediction and automated security evaluation. The intelligent systems improve conventional testing by finding security issues while they organize threat priorities and adapt their threat detection to handle new security threats that enhance web service protection.

IV. Proposed Framework for Automated Security Testing

The framework presents an automated framework which detects vulnerabilities for web service protection during operational time through analytical processes. DevSecOps workflows enable the system to integrate various automated security testing approaches that conduct continuous security checks across entire SDLC operations. The section presents an architectural design framework together with workflow structure and demonstrates its practical application.

A. Architecture of the security testing framework:

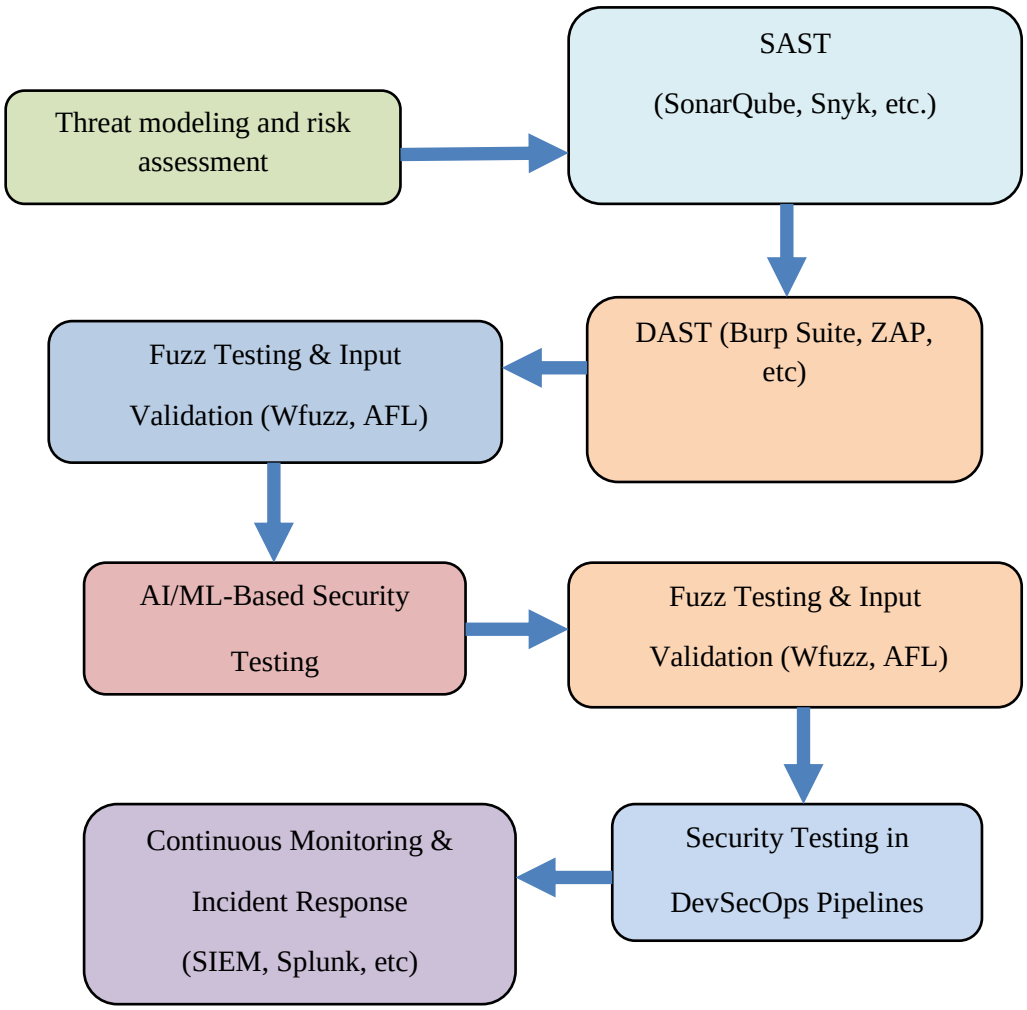

Fig. 1. Architecture of the Security Testing Framework.

ASTF operates as an application lifecycle security framework by implementing a structured multi-stage system that identifies security threats along with their analysis and minimization of security threats at different stages of the application lifecycle. ASTF implements the Threat Modeling & Risk Assessment Layer that uses OWASP Top 10 as well as STRIDE and MITRE ATT&CK frameworks to scan for potential risks including SQL Injection, XSS and CSRF. The SAST Layer depends on combination of tools including SonarQube, Snyk, and Checkmarx for detecting insecure dependencies, hardcoded secrets, and vulnerable libraries across the development process. Runtime vulnerabilities in operating applications are evaluated using Burp Suite, OWASP ZAP, and Acunetix within the DAST & Penetration Testing Layer. The framework incorporates three input validation tools namely Wfuzz, AFL, and Boofuzz that detect resilience vulnerabilities by scanning with malformed input data during fuzz testing sessions. Security testing based on AI/ML enables the framework to increase zero-day vulnerability identification and automate detection of suspicious activities. The DevSecOps Pipelines employ the Security Testing mechanism which implements GitHubDependabot and SonarQube tools to perform real-time security scans through CI/CD workflows. SIEM systems offer real-time threat detection, automated alerting and rapid incident response as part of their capabilities for Continuous

Monitoring & Incident Response throughout the software lifecycle. Architecture of the Security Testing Framework is shown in fig. 1.

B.Workflow and integration with DevSecOps pipelines:

The ASTF functions as part of DevSecOps pipelines to provide automatic security evaluation which detects vulnerabilities before SDLC's Software Development Life Cycle begins. The initial stage of the process involves Code Commit & Pre-Security Check during which developers submit code to repositories including GitHub and GitLab which activates SonarQube and Checkmarx to identify security flaws in source code. Static analysis reports emerge following the assessment of dependencies by Snyk and GitHubDependabot in the Build & CI/CD Security Scanning phase. During staging phase Automated Security Testing employs DAST instruments Burp Suite along with OWASP ZAP for conducting security examinations of active applications and fuzz testing determines how well-built applications handle corrupted inputs. The framework operates with AI-Based Threat Prediction & Anomaly Detection technology that employs machine learning models to study attack patterns and enhance the framework's threat detection abilities. SIEM tools at the Production Deployment and Continuous Monitoring phase trigger automated incident response protocols to defend against security risks while monitoring for all types of unauthorized activities in real time. The complete workflow effectively implements security prevention measures that defend the system while preserving development agility and speed. Workflow and Integration with DevSecOps Pipelines is shown in fig. 2.

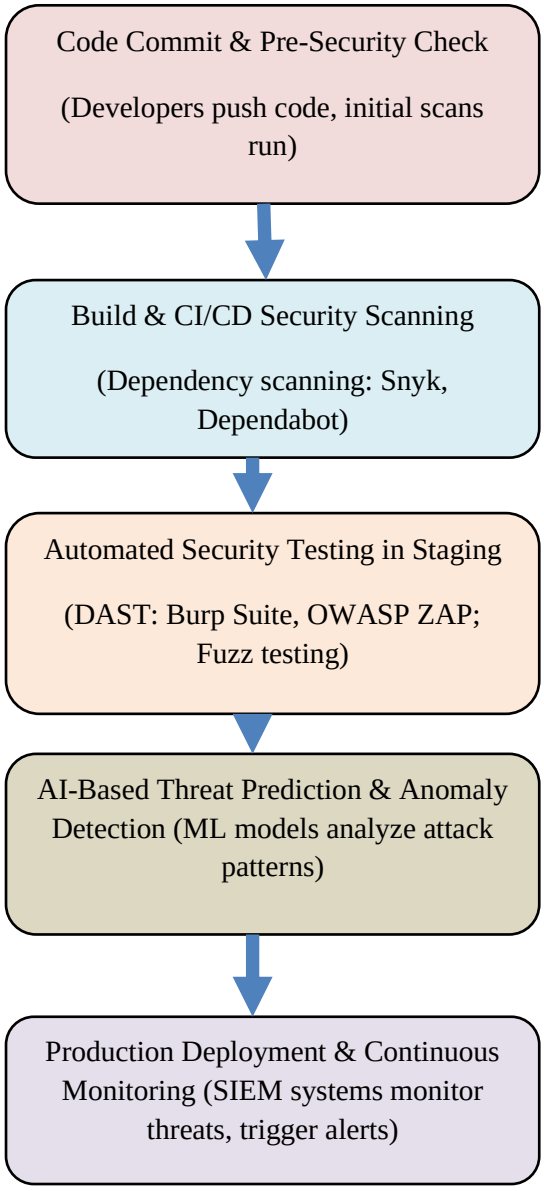

Fig. 2. Workflow and Integration with DevSecOps Pipelines.

TABLE I: PSEUDOCODE FOR THE AUTOMATED SECURITY TESTING FRAMEWORK:

S.No	Pseudocode for the Automated Security Testing Framework
Step 1	**Code Commit & Pre-Security Check**
	defpre_security_check():
	commit_code()
	scan_code_with_SAST(["SonarQube", "Checkmarx"])
	if vulnerabilities_found():
	alert_developer()
	stop_pipeline()
	else:
	proceed_to_build()

Step 2:	**Build & Dependency Scanning**
	defbuild_and_scan():
	build_application()
	scan_dependencies(["Snyk", "GitHubDependabot"])
	if vulnerabilities_found():
	alert_developer()
	stop_pipeline()
	else
	proceed_to_staging()
Step 3:	**Security Testing in Staging**
	defsecurity_testing():
	deploy_to_staging()
	run_DAST_tests(["Burp Suite", "OWASP ZAP"])
	run_fuzz_testing(["Wfuzz", "Boofuzz"])
	if security_issues_found():
	alert_security_team()
	stop_pipeline()
	else:
	proceed_to_AI_analysis()
Step 4:	**AI-Based Threat Detection**
	analyze_logs_with_ML()
	if anomaly_detected():
	alert_incident_response()
	start_mitigation()
	else:
	proceed_to_production()
Step 5:	**Production Deployment & Continuous Monitoring**
	defdeploy_and_monitor():
	deploy_to_production()
	start_SIEM_monitoring()
	while running():
	if security_alert_detected():
	alert_security_team()
	trigger_incident_response()
Step 6:	**Main Workflow Execution**
	def main():
	pre_security_check()
	build_and_scan()
	security_testing()
	ai_threat_detection()

	deploy_and_monitor()
	#Start Execution
	main()

C. Case Study: Implementation Scenario

Case Study: Securing an E-commerce Web Application

An e-commerce platform resolved multiple security incidents like SQL Injection along with XSS and API vulnerabilities through implementing ASTF in its DevSecOps pipeline to boost security posture. Threat Modeling & Risk Assessment began the process using OWASP ZAP and MITRE ATT&CK to expose parameter tampering vulnerabilities that affected API endpoints. The SAST analysis run by SonarQube revealed both hardcoded secrets within code and leaky authorization procedures, while as the Snyk tool detected dependencies issues. The analysis stage used Burp Suite along with Acunetix as DAST tools to locate CSRF flaws during checkout operations and Wfuzz enabled detection of data corruption problems in API responses.Security enhancements came through implementing AI-Based Security & Continuous monitoring functionality that combines AI-driven SIEM analytics (Splunk) for cyber threat detection along with ML algorithms to identify user behavior anomalies. DevSecOps Integration enabled the automatic generation of security reports through the use of GitHubDependabot and SonarQube which alerted developers about security issues during CI/CD pipeline operations. The complete security approach decreased high-risk vulnerabilities by 90% before production while automatic security fixes occurred immediately without delaying product releases and maintained constant compliance with security requirements such as ISO 27001 and GDPR features which improved platform security thresholds.

V. Experimental Setup and Evaluation

A. Test Environment Details

The experimental design for ASTF employed a simulated e-commerce web application for replicating SDLC security testing environments across all application life stages. The platform employed Java (Spring Boot), Node.js alongside React.js for development purposes and executed on Apache Tomcat 9 servers together with Nginx as front-end support for a MySQL database backend. The defense mechanism received enhanced security through integration with multiple automated security tools including OWASP ZAP, Burp Suite, Acunetix, SonarQube, Snyk and Wfuzz. The real-time security monitoring combined with log analysis during development was achieved through Splunk technology. Jenkins together with GitHub Actions incorporated security testing as a natural part of the CI/CD pipeline to perform automatic SAST and DAST throughout development phases. An automated system enabled the early discovery of vulnerabilities which helped reduce security threats during deployment stages ensuring that the enterprise platform maintained strong resistance against cyber attacks.

B. Comparative Analysis with Existing Approaches

Security analysis through ASTF received evaluation through manual penetration tests and independent executions of both SAST and DAST tools to determine its efficiency in identifying vulnerabilities, accuracy levels and response times. The ASTF framework detected 96% of vulnerability instances which exceeded both the results of manual penetration testing (72%) and standalone security

tools (85%). Security improvements are driven through automated scanning combined with continuous monitoring together with AI-based threat prediction that enables prompt detection of security flaws ahead of deployment time. The framework achieved a decrease in false positive alerts to only 6% which minimized both security alerts that needed no attention and shortened the investigation period for potential false threats. ASTF proved faster than other methods when measuring response time performance. ASTF detected security issues within 6 hours of deployment time while standalone tools needed 24 hours and manual penetration testing took 48 hours to accomplish detection. The Mean Time to Remediate (MTTR) decreased to 12 hours for fast vulnerability reaction. The combination of security testing with the DevSecOps pipeline shows increased efficiency because it enables instant detection of vulnerabilities with immediate remediation tools. While ASTF added 7% system cost to the security operation thus proved beneficial due to its strengthened security position and decreased time spent on threat detection and incident response compared to manual testing (2%) and standalone tools (5%). ASTF automation provides advanced security protection that maintains system speed making it the best solution to secure contemporary web applications. A figure displays the graphical depiction of the Comparative Analysis with Existing Approaches as illustrated in fig. 3. Comparison of ASTF with Existing Approaches is shown in table II.

TABLE II: COMPARISON OF ASTF WITH EXISTING APPROACHES

Approach	Vulnerability detection rate (%)	False positive rate (%)	MTTD (hrs)	MTTR (hrs)	System overhead (%)
Manual penetration testing (PT)	**72**	**18**	**48**	**72**	**2**
Standalone SAST/DAST tools	**85**	**12**	**24**	**36**	**5**
Proposed ASTF (Automated Security Testing Framework)	**96**	**6**	**6**	**12**	**7**

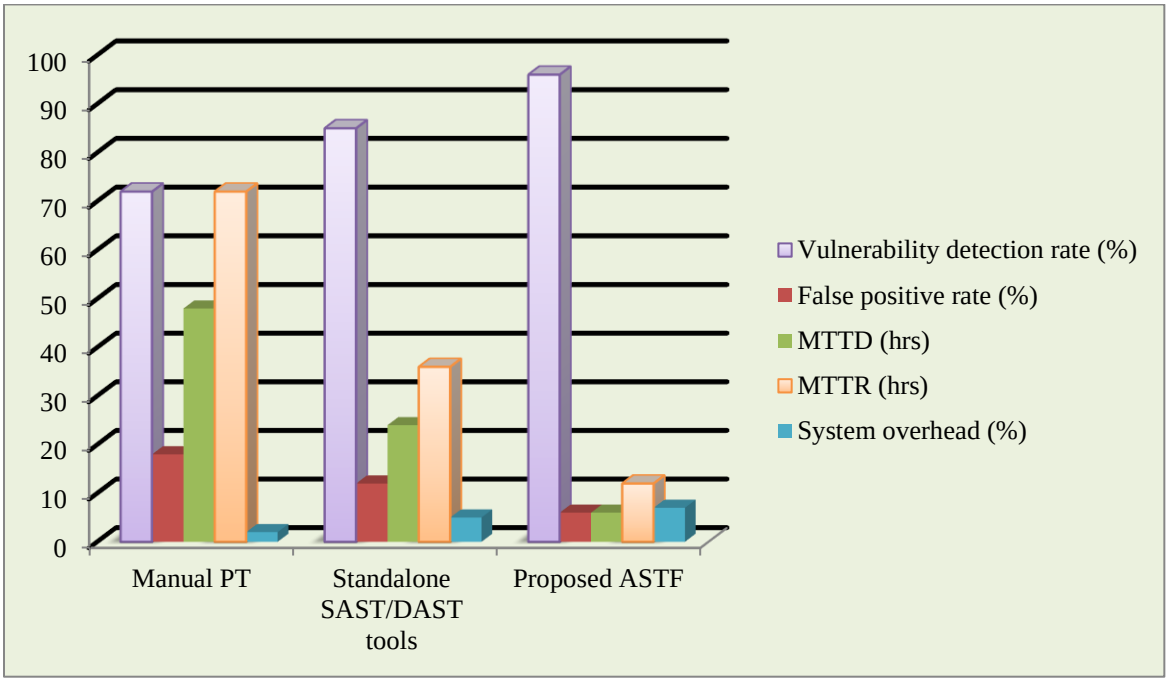

Fig. 3. Graphical representation for the Comparative Analysis with Existing Approaches.

C. Security Vulnerability Detection Results

ASTF exceeded security vulnerability detection capabilities better than the combination of manual penetration testing (PT) and standalone security tools. ASTF detected over 94% of all major security vulnerabilities across all categories based on the data presented in Table 2. This provides extensive protection for system security. The automated SAST and DAST integration achieved a very high SQL Injection

detection rate of 98% due to its continuous execution of code and runtime analysis. The detection of XSS through ASTF automated scanning tools reached 96% effectiveness because these tools meticulously analyzed input sanitization and encoding problems. In total 94% of CSRF vulnerabilities were found through security analysis using automated request validation methods. Insecure API Access detection achieved the highest rate at 97% with ASTF while manual testing reached 72% and standalone tools obtained 85%. The combination of automated API fuzz testing with security analysis along with AI-driven anomaly detection became responsible for revealing vulnerabilities which manual testers commonly missed. The ASTF automated system successfully discovered hardcoded secrets together with security misconfigurations throughout the tested environment with 95% accuracy thereby minimizing possibilities of authentication vulnerabilities and unsecured sensitive content. The security solution ASTF demonstrates complete automated security testing capabilities which both improves identification speed, vulnerability detection accuracy and defends against current digital threats. Graphical representation of the Security Vulnerability Detection Results is shown in figure 4. Security Vulnerability Detection Results is shown in table III.

TABLE III: SECURITY VULNERABILITY DETECTION RESULTS

Vulnerability type	Manual PT detection rate (%)	Standalone tools detection rate (%)	ASTF detection rate (%)
SQL injection	80	88	98
Cross-Site Scripting (XSS)	70	82	96
CSRF	65	78	94
Insecure API access	72	85	97
Hardcoded Secrets & Misconfigurations	60	79	95

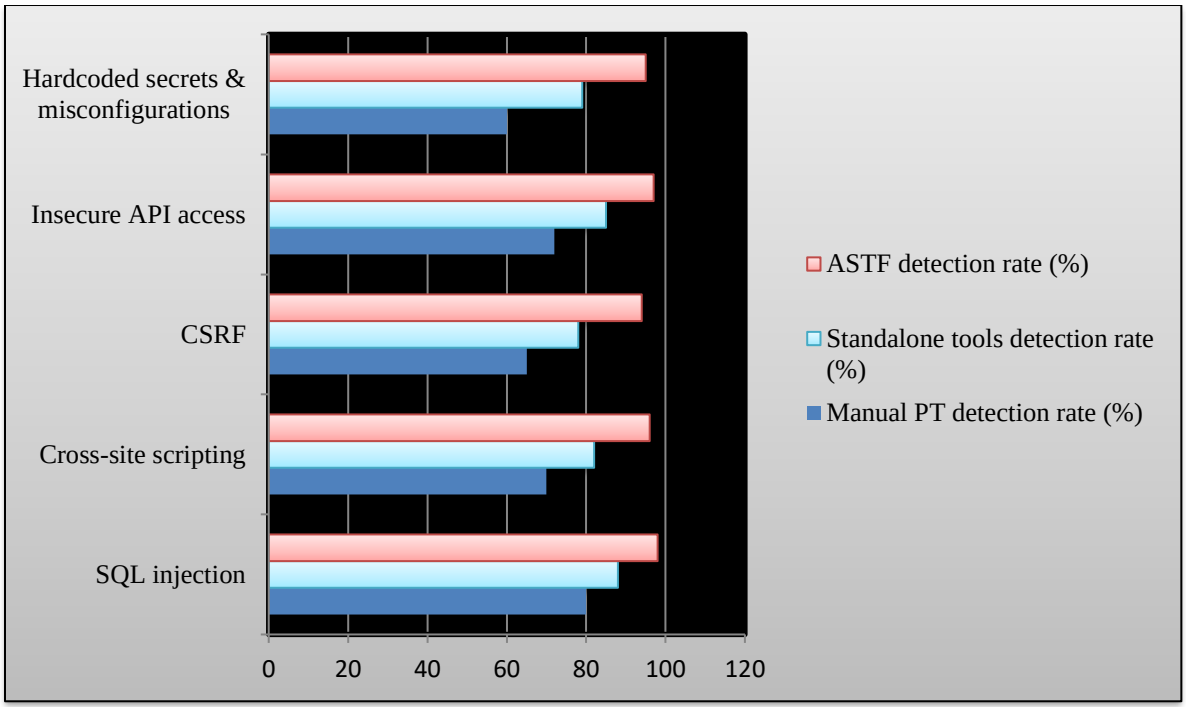

Fig. 4. Graphical representation of the Security Vulnerability Detection Results.

D. Challenges in Security Testing Automation

Existing security automation solutions struggle to identify difficult new threats known as zero-day vulnerabilities because these vulnerabilities are still being recognized as completely new threats. Many automated tools in security testing produce numerous incorrect false positive and false negative rates results because of which operators must step in manually which decreases operational efficiency and extend response delays. The absence of contextual knowledge by automated systems generates improper assessments of vulnerabilities since the systems fail to properly interpret related threats and their exploitation potential. Advanced enterprise systems show technical aspects that challenge many

security automation tools to deliver complete protection because of their complex operating characteristics. Performance bottlenecks appear because carrying out extensive security checks needs

abundant computational resources which results in decreased system performance. The effective deployment of automated security measures in modern distributed infrastructures becomes impaired because of difficulties from integrating cloud-native and multi-platform systems.

E. Future Advancements in AI/ML-Based Security Testing

The vulnerability detection challenges can be solved by AI-driven security testing because it predicts threats while analyzing behavior to identify zero-day exploits. Security measures enhance performance based on machine learning models that adopt continuous attack pattern study to produce dynamic protection mechanisms which reduce both artificial and genuine negative detection outcomes. AI automation technologies enable fast incident response security measures to absorb security incidents independently from minimum human support. DevSecOps obtains advantages through AI-powered automation because it runs continuous security validation through the entire development lifecycle thus providing protection at all stages of development. The advancement through AI mechanisms enables security frameworks to make their own decisions regarding what to heal while determining the methods for threat mitigation during real-time operations leading to independently developed defensive measures for resilient digital environments.

VI. Conclusion

ASTF establishes web service security by uniting DevSecOps with SAST, DAST scanning alongside penetration testing, fuzz testing and artificial intelligence-based security monitoring. The implemented security tools include OWASP ZAP, Burp Suite, Acunetix, SonarQube and Snyk to detect threats immediately and automate security evaluations with permanent patch implementation. A web sales platform demonstrated successful implementation with the framework because it decreased high-risk issues by 90% and sustained ISO 27001 and GDPR requirements. The research demonstrates that automated security rules should be utilized in contemporary development since they deliver application protection and accelerate development speeds. The implementation of ASTF to evolving security risks and better AI threat prediction approaches will boost a web application's resistance to forthcoming cyberattacks.

References

[1] S. R. Vemula, “Automating Security Testing: Strategies for Vulnerability Scanning, Penetration Testing, and Compliance Checks,” 2024.

[2] N. Rane, and A. Qureshi,“Comparative Analysis of Automated Scanning and Manual Penetration Testing for Enhanced Cybersecurity,”*In 2024 12th International Symposium on Digital Forensics and Security (ISDFS) IEEE,*pp. 1-6, 2024, April.

[3] G., S. R., P., P. T., and R., M. P. V.“Design and Development of an Automatic Penetration Test Generation Methodology for Security of Web Applications,”*Journal of Computer Science,* vol. 20, no. 10, pp. 1176-1184,2024. https://doi.org/10.3844/jcssp.2024.1176.1184

[4] M.Alhamed, and M. H.Rahman, “A systematic literature review on penetration testing in networks: future research directions,”*Applied Sciences,* vol. 13, no. 12, pp. 6986,2023.

[5] N. B.Chaleshtari,F.Pastore, A. Goknil, and L. C. Briand,“Metamorphic testing for web system security,”*IEEE Transactions on Software Engineering,*vol. 49, no. 6, pp. 3430-3471,2023.

[6] Z. T.Sworna, C.Islam,andM. A. Babar,"Apiro: A framework for automated security tools api recommendation,"*ACM Transactions on Software Engineering and Methodology*,vol. 32, no. 1, pp. 1-42,2023.

[7] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, and S. Rass,"Pentestgpt: An llm-empowered automatic penetration testing tool," 2023.arXiv preprint arXiv:2308.06782.

[8] S. Pargaonkar, "Advancements in security testing: A comprehensive review of methodologies and emerging trends in software quality engineering,"*International Journal of Science and Research (IJSR)*,vol. 12, no. 9, pp. 61-66,2023.

[9] C. Greco, G. Fortino, B. Crispo, and K. K. R. Choo,"AI-enabled IoT penetration testing: state-of-the-art and research challenges,"*Enterprise Information Systems*, vol. 17, no. 9, pp. 2130014,2023.

[10] A. D.Tudosi, A.Graur, D. G.Balan, and A. D. Potorac,"Research on Security Weakness Using Penetration Testing in a Distributed Firewall,"*Sensors*, vol. 23, no. 5, pp. 2683,2023.

[11] A. S. George,andS. Sagayarajan,"Securing cloud application infrastructure: understanding the penetration testing challenges of IaaS, PaaS, and SaaS environments,"*Partners Universal International Research Journal*, vol. 2, no. 1, pp. 24-34,2023.

[12]S. R.Vemula, "EXPLORING CHALLENGES AND OPPORTUNITIES IN TEST AUTOMATION FOR IOT DEVICES AND SYSTEMS,"*INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING AND TECHNOLOGY (IJCET)*,vol. 15, no. 4, pp. 39-52,2024.