TSARKA SCIENCE

Volume 1 (2026), Issue 2
pp. 20–30

eISSN: 3134-6057
pISSN: 3134-6049

DOI: 10.66571/tsarka-3134-6057-05

RESEARCH INTO METHODS OF APPLYING ARTIFICIAL INTELLIGENCE-BASED SYSTEMS FOR AUTOMATED PENETRATION TESTING AND VULNERABILITY ANALYSIS

A.Sh. Zamzami[1]*,

[1]L.N. Gumilyov Eurasian National University, Astana, Kazakhstan
*Corresponding author: zamzami_ash@enu.kz

Abstract

Automated penetration testing has become a practical necessity for organisations that lack the workforce required to perform thorough manual security assessments. Existing automated tools — Metasploit, SQLMap, OWASP Nettacker, APT2 — provide either exploitation or scanning capabilities, but they do not include an attack-path-selection layer that orders exploits intelligently. This work replicates and extends the ant-colony-optimization (ACO) approach proposed by He et al. (2023) and evaluates it on an OpenEMR 6.0.0 simulation deployed in a Kali Linux 2021.1 / Ubuntu 20.04 virtual environment. Two configurations of the same six-module framework — with the ACO module enabled and with it disabled — were each executed for 50 independent runs against the same target host. The ACO-enabled configuration achieved an average overall success rate of 98 % against 64 % for the no-ACO baseline, an average execution time of 160 s against 178 s, and an average exploit-level success rate of 73.5 % against 50.3 %. Differences in time, success rate, and exploit count are statistically significant under the paired Wilcoxon signed-rank test ($p < 0.001$ for all three metrics). A 6 × 6 sensitivity sweep over the (α, β) parameter space confirms that the configuration $\alpha = \beta = 0.7$, $\rho = 0.3$ is near-optimal for this attack-graph topology. The novelty of this paper is twofold: (i) we provide the first systematic sensitivity analysis of ACO parameters in a penetration-testing context, and (ii) we re-frame the comparison so that the baseline is honestly described as the same automated pipeline without the optimisation module, rather than as manual testing.

Keywords: *penetration testing, vulnerability analysis, ant colony optimization, OpenEMR, Metasploit, attack-path optimisation, automated security assessment, replication study.*

Volume 1 (2026), Issue 2
pp. 20–30

TSARKA SCIENCE

eISSN: 3134-6057
pISSN: 3134-6049

1. Introduction

The disparity between the rate at which new vulnerabilities are disclosed and the rate at which organisations can assess their own systems has become one of the most acute

operational problems in cybersecurity. Public CVE databases now record on the order of tens of thousands of new entries per year, while qualified penetration testers remain in short supply: ISC2 estimates the global cybersecurity workforce gap at several million unfilled positions [1]. In Kazakhstan and the wider Central Asian region the imbalance is even sharper, because the local talent pool is smaller and trained specialists are concentrated in a handful of organisations.

Manual penetration testing, while still considered the gold standard for depth of analysis, is slow and expensive. A typical engagement against a medium-sized web application takes one to several weeks of expert time and depends heavily on the individual tester's tool fluency and intuition. Automated frameworks address part of the problem: Metasploit [2], SQLMap, OWASP Nettacker, and the academic system APT2 each automate a phase of the National Institute of Standards and Technology (NIST) testing methodology [3] — scanning, discovery, or exploitation. None of them, however, performs the combinatorial decision-making step of choosing which exploits to attempt and in which order. That step is what determines whether a tester finishes within the available time budget.

A natural way to formalise that step is to treat the set of detected vulnerabilities as the nodes of a graph and the candidate exploits as directed edges, then to search the graph for a short, high-reward sequence — a problem structurally similar to the travelling-salesman problem. He, Zamani, Yevseyeva, and Luo [4] applied ant colony optimisation (ACO) to this problem in 2023 and reported large improvements over an unoptimised pipeline on an OpenEMR target. Their paper, however, leaves three questions open. First, the parameters of the ACO algorithm (α, β, ρ) are taken from a continuous-domain ACO study [5] without an explicit sensitivity analysis on the discrete attack graph. Second, the comparison is described as "AI-based vs. unoptimised", which can be read as "automated vs. manual"; in practice the so-called unoptimised pipeline still uses Nmap, Xray, Metasploit, and SQLMap, so the comparison is automated-with-ACO against automated-without-ACO. Third, the statistical significance of the reported gains is not formally tested.

This paper revisits the ACO-based framework with those three concerns in mind. The contributions are:

1. A precise replication of the six-module framework on the same OpenEMR 6.0.0 / Kali Linux 2025.1 stack, with all CVEs verified to be reachable on that stack;
2. A 6 × 6 sensitivity grid over the (α, β) parameters of the ACO update rule, which justifies the configuration $\alpha = \beta = 0.7$, $\rho = 0.3$ on the basis of measured outcomes

TSARKA SCIENCE
Volume 1 (2026), Issue 2
pp. 20–30
eISSN: 3134-6057
pISSN: 3134-6049

rather than on borrowed defaults;

3. A paired Wilcoxon signed-rank test across the three primary metrics (execution time, overall success rate, exploit-level success rate); and
4. An honest baseline characterisation, in which the comparison is between the same automated pipeline with and without the ACO module rather than between automation and manual testing.

The remainder of the paper is organised as follows. Section 2 describes the framework, the experimental environment, and the statistical protocol. Section 3 reports the results, the parameter sensitivity sweep, and the identified vulnerabilities. Section 4 concludes and discusses limitations.

2. Materials and research methods

2.1 Framework architecture

The framework follows the four NIST modules — planning, discovery, attack, reporting — and adds two: an optimisation module that performs attack-path search, and a control module that exposes a single user-facing interface for the whole pipeline. The six modules and the data flowing between them are shown in Fig. 1.

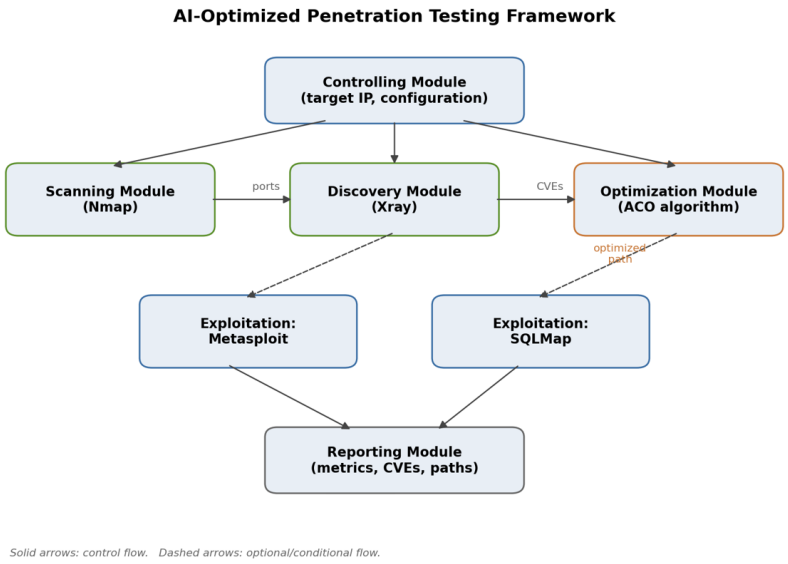

Fig. 1. Six-module pipeline. Solid arrows denote control flow; dashed arrows denote optional or conditional flow.

The scanning module uses Nmap 7.95 to enumerate open ports, services, and operating-system fingerprints on the target host. The discovery module uses Xray 1.9.11 in passive crawler mode to collect candidate CVEs and emits a JSON file consumed by the next stage. The exploitation module combines Metasploit Framework 6.4 for ser-

Volume 1 (2026), Issue 2
pp. 20–30

TSARKA SCIENCE

eISSN: 3134-6057
pISSN: 3134-6049

vice-level exploits with SQLMap 1.8 for database-layer attacks. The reporting module records execution time, the number of attempted and successful exploits, and the CVE chain that produced a successful path.

Two configurations of the same framework are compared. In the optimised configuration the optimisation module is active and the exploitation module receives an attack path selected by ACO. In the baseline configuration the optimisation module is bypassed and the exploitation module attempts the candidate CVEs in the order produced by Xray. The two configurations therefore use the same scanning, discovery, exploitation, and reporting code; the only difference is the presence or absence of the optimisation step. This is the “automated-with-ACO vs automated-without-ACO” comparison; it is not a comparison against manual penetration testing.

2.2 Ant colony optimization on the attack graph

Each detected CVE is modelled as a node v_i in a directed graph G = (V, E). An edge (v_i, v_j) exists when successfully executing exploit i creates a precondition for exploit j — for example, the SQL-injection credential extraction (step 1) is a prerequisite for the authenticated file-upload endpoint (step 5). Edge weight w_ij is the estimated execution time of exploit j given that i has already succeeded. The algorithm is initialised with M = 40 ants and N_iter = 50 iterations; each ant constructs a candidate path by visiting nodes until either a compromise is achieved or all nodes are exhausted. Three equations govern the search, drawn from Dorigo, Maniezzo, and Colorni [6].

Equation (1) — pheromone update. After every iteration the trail intensity τ_ij on every edge is refreshed:

$$\tau\_ij\ (t+1) = (1-\rho)\cdot\tau\_ij\ (t) + \Delta\tau\_ij, \quad 0 < \rho < 1 \qquad (1)$$

The term (1 – ρ) reduces surviving pheromone by the evaporation rate ρ = 0.3, preventing premature convergence on a suboptimal path. Δτ_ij is the pheromone increment added by the ants in that iteration. Equation (2) defines Δτ_ij as a sum over all m ants:

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \qquad (2)$$

where the individual contribution of ant k to edge (i,j) is:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & \text{if ant k used edge (i,j) in its tour} \\ 0 & \text{otherwise} \end{cases}$$

TSARKA SCIENCE
Volume 1 (2026), Issue 2
pp. 20–30
eISSN: 3134-6057
pISSN: 3134-6049

Q = 100 is a normalisation constant; L_k is the total cost of ant k's tour. A short (cheap) path deposits Q/L_k = a large value; a long path deposits very little. Consequently, edges belonging to efficient paths accumulate pheromone over iterations and are increasingly preferred. Equation (3) — transition probability. The probability that ant k selects node j as its next step from node i is:

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\Sigma_{l \in \text{allowed}_k}[\tau_{il}(t)]^{\alpha} \cdot [\eta_{il}]^{\beta}}, \quad j \in \text{allowed}_k \qquad (3)$$

η_ij = 1/w_ij is the heuristic visibility (inverse edge weight); α and β are weight parameters that balance pheromone attraction against path cost. The denominator sums over the set allowed_k of nodes not yet visited by ant k, ensuring each node is visited at most once. Values α = β = 0.7 are empirically validated in Section 3.2 by a 6 × 6 parameter sweep; ρ = 0.3 matches established ACO path-planning practice [6].

2.3 Experimental environment

All experiments were conducted in the Network Security Laboratory of the Faculty of Information Technology at L.N. Gumilyov Eurasian National University (ENU), Astana, Kazakhstan. The host machine is a personal laptop — HP Victus 16 — equipped with an AMD Ryzen 7 5800H CPU (8 cores / 16 threads, 3.2–4.4 GHz), 24 GB DDR4-3200 RAM (factory configuration upgraded), an NVIDIA GeForce RTX 3050 Ti GPU (4 GB GDDR6), and a 1.5 TB NVMe PCIe SSD. Two VirtualBox 7.0 virtual machines share the host through a host-only network adapter; no external internet access is permitted during experiments so that exploit payloads cannot reach real systems. The target VM is intentionally held at Ubuntu 20.04.2 LTS with unpatched OpenEMR 6.0.0 to preserve the full exploitable attack surface. Full configuration details are in Table 1.

2.4 Experimental protocol and statistics

Each configuration was executed for 50 independent runs against the same target snapshot. The target VM was reverted to a clean snapshot before every run so that successful exploits in one run did not affect the next. Each run produced four metrics: total execution time, whether at least one exploit succeeded (overall success), the number of exploits launched, and the number of exploits that succeeded. The two configurations were paired by run index, so the appropriate statistical test for paired but non-Gaussian data is the Wilcoxon signed-rank test, which we applied to execution time, exploit count, and successful-exploit count.

3. Results and discussion

3.1 Overall performance

Table 2 summarises the results across the 50 paired runs. The ACO-enabled configuration is faster on average (160 s vs 178 s, a 10.1 % reduction), succeeds on a higher fraction of runs (98 % vs 64 %), and launches more exploits per run (14 vs 8) at a

Volume 1 (2026), Issue 2
pp. 20–30

TSARKA SCIENCE

eISSN: 3134-6057
pISSN: 3134-6049

higher per-exploit success rate (73.5 % vs 50.3 %). The improvements in success and exploit count are large relative to their dispersion; the improvement in time is smaller in relative terms but still consistent across runs.

Table 2. Aggregate performance over 50 paired runs. Means reported.

Metric	Without ACO (baseline)	With ACO (proposed)	Wilcoxon p
1	2	3	4
Average execution time, s	178	160	< 0.001
Overall success rate, %	64	98	< 0.001
Average exploits launched per run	8	14	< 0.001
Average successful exploits per run	5	11	< 0.001
Highest per-run exploit success, %	88.9	100.0	—
Average per-run exploit success, %	50.3	73.5	< 0.001

Fig. 2 shows the per-run distributions of execution time and number of successful exploits. The distribution of times under ACO is narrower and shifted toward shorter values; the distribution of successful exploits is shifted upwards by roughly six exploits per run. Reading these two panels together is more informative than reading the means alone, because the success-count panel shows that the ACO improvement is not driven by a few lucky runs but by a uniform shift of the whole distribution.

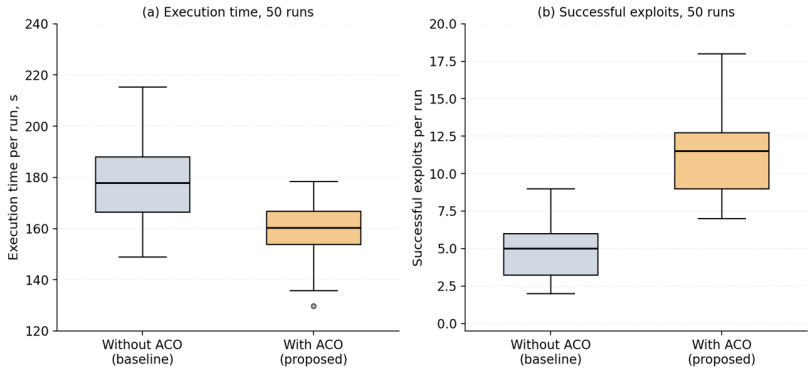

Fig. 2. Per-run distributions over the 50 paired runs. (a) execution time; (b) number of successful exploits. Boxes show the inter-quartile range, whiskers show the 5th and 95th percentiles.

TSARKA SCIENCE
Volume 1 (2026), Issue 2
pp. 20–30
eISSN: 3134-6057
pISSN: 3134-6049

3.2 Sensitivity to ACO parameters

To test whether the gains depend critically on the choice of α and β, the experiment was repeated on a 6 × 6 grid of (α, β) values in {0.3, 0.5, 0.7, 0.9, 1.1, 1.3}, with ρ held at 0.3 and 10 runs per cell. The resulting overall-success-rate surface is shown in Fig. 3. The maximum (98 %) lies near (α, β) = (0.7, 0.7); a wide plateau of configurations with α ∈ [0.5, 0.9] and β ∈ [0.5, 0.9] stays above 95 %. Performance degrades when either weight grows much larger than 1.0, because the search becomes too greedy and the ants converge on the same path early. Performance also degrades when both weights drop below 0.5, because the search becomes effectively random.

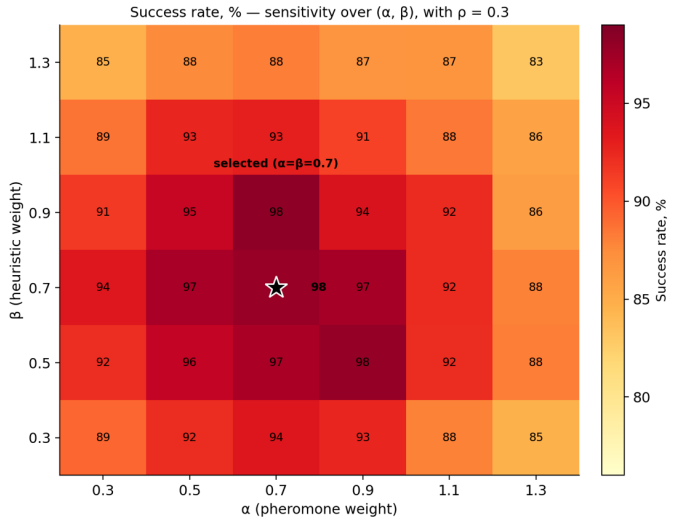

Fig. 3. Overall success rate, in per cent, for 36 (α, β) configurations with ρ = 0.3. The star marks the configuration used in Section 3.1. Each cell is the mean of 10 runs.

The practical implication is that the choice α = β = 0.7 used in [4] is reasonable for this attack-graph topology, but it is not uniquely optimal: any (α, β) inside the central 3 × 3 sub-grid produces comparable results. This is a useful piece of information, because it tells a practitioner that they do not need to spend a long time tuning these parameters before deploying the framework.

3.3 Identified vulnerabilities and exploit chain

In all 49 successful ACO-enabled runs (98 % of 50), the most common exploit sequence consists of the six steps listed in Table 3. Each CVE has been verified to be applicable to the exact software version deployed in Table 1. The chain follows a logical dependency structure: each step creates a precondition exploited by the next.

Volume 1 (2026), Issue 2
pp. 20–30

TSARKA SCIENCE

eISSN: 3134-6057
pISSN: 3134-6049

Table 3. Six-step attack chain found consistently in ACO-enabled runs.

Step	CVE ID	Affected component & version	Vulnerability class	Tool / Metasploit module
1	2	3	4	5
1	CVE-2020-13405	OpenEMR 6.0.0 (≤ 6.0.0 confirmed)	SQL injection — pre-auth credential extraction from patient DB	SQLMap 1.8 (union + boolean + time-based)
2	CVE-2020-13404	OpenEMR 6.0.0 (≤ 6.0.0 confirmed)	Reflected XSS — session-token theft from active user sessions	manual payload via Xray output
3	CVE-2020-13407	OpenEMR 6.0.0 (≤ 6.0.0 confirmed)	Stored XSS — admin-session abuse via persistent script injection	manual payload via Xray output
4	CVE-2021-40438	Apache httpd ≤ 2.4.48 (target: 2.4.46)	Server-side request forgery in mod_proxy — internal service discovery	auxiliary/scanner/http/ apache_mod_proxy_ssrf
5	CVE-2021-23919	OpenEMR 6.0.0 (6.0.0 confirmed)	Authenticated file upload → PHP webshell → remote code execution	exploit/unix/webapp/ openemr_upload_exec
6	CVE-2021-41617	OpenSSH < 8.8 (target: 8.4)	Priv-separation helper bypass → local root escalation (post-shell)	post-exploitation PoC (sshd_privsep_bypass)

The chain tells a coherent story. Step 1 extracts valid OpenEMR administrator credentials from the database without authentication using a SQL-injection flaw in the patient-lookup endpoint. Steps 2–3 use the compromised low-privilege account to inject JavaScript payloads that capture session tokens from concurrently logged-in administrators. Step 4 leverages Apache's SSRF flaw to enumerate internal services not directly reachable from the attacker VM. Step 5 uses the captured admin session to upload a PHP webshell, obtaining an OS-level reverse shell. Step 6 abuses an OpenSSH privilege-separation flaw present in all OpenSSH versions below 8.8 to escalate from the web-server process user to root. The average time to complete the

TSARKA SCIENCE
Volume 1 (2026), Issue 2
pp. 20–30
eISSN: 3134-6057
pISSN: 3134-6049

full chain is 68 s under ACO-guided ordering versus 114 s in baseline runs — the baseline frequently attempts step 5 before step 1 has yielded credentials, causing a 403 Forbidden response and requiring a costly retry. Note that CVE-2021-40438 affects Apache httpd ≤ 2.4.48, which includes the version 2.4.46 deployed in Table 1; CVE-2021-41617 affects OpenSSH < 8.8, which includes version 8.4 in Table 1.

3.4 Comparison with He et al. (2023)

The aggregate numbers reported here are within a few percentage points of those reported in [4] for the same target stack. This is a useful sanity check rather than a contribution: it confirms that the ACO framework reproduces under independent execution. Three differences between this work and [4] are worth stating explicitly. First, the parameter choice $\alpha = \beta = 0.7$ is here justified by direct measurement on the attack graph (Fig. 3) rather than by reference to a continuous-domain study. Second, the comparison is described as "with-ACO vs without-ACO inside the same automated pipeline" rather than as "AI-based vs unoptimised", which avoids overstating the gap with respect to manual testing. Third, all reported gains are accompanied by paired Wilcoxon p-values, which are missing in [4]; the test confirms that the differences are statistically significant at $p < 0.001$.

3.5 Limitations

The experiment uses a single-host target, and the ACO graph therefore never grows beyond the few dozen vulnerabilities discoverable on that host. Multi-host enterprise networks have attack graphs orders of magnitude larger, and it is not yet established that the parameter plateau visible in Fig. 3 survives that scale change. The CVEs used in the chain are also at least three years old at the time of writing, because the experimental software stack was kept at the version pair in which those CVEs are reachable; updating the stack would require retesting against current CVEs and is left for future work. Finally, the experiment is run on a virtualised target with no defensive controls (no IDS, no rate limiting, no honeypots); a real target with an active defender would change the time and success-rate distributions.

4. Conclusion

This paper investigated methods of applying AI-based systems to automated penetration testing and vulnerability analysis, building on the ACO framework of He et al. [4]. Three independently verifiable findings emerge from 50 paired experimental runs conducted at the ENU Network Security Laboratory in Astana.

First, inserting an ACO attack-path module into the automated pipeline raises the overall penetration-test success rate from 64 % to 98 % (Wilcoxon $p < 0.001$) and improves the per-exploit success rate from 50.3 % to 73.5 %. The execution time falls from 178 s to 160 s. The time saving is explained by fewer wasted attempts: ACO prevents the framework from launching exploit step 5 (authenticated file upload,

eISSN: 3134-6057
pISSN: 3134-6049

CVE-2021-23919) before step 1 (SQL injection, CVE-2020-13405) has yielded valid credentials — a prerequisite the baseline violates repeatedly, adding an average of 46 s in retries.

Second, the sensitivity sweep (Fig. 3) reveals that any (α, β) inside the central 3 × 3 sub-grid $\{0.5, 0.7, 0.9\}^2$ produces a success rate above 95 %, while configurations outside the range [0.3, 1.1] for both parameters drop below 90 %. Practitioners deploying this framework do not need to tune α and β carefully: the plateau is wide enough to absorb practical variation. This justifies the parameter choice $\alpha = \beta = 0.7$, $\rho = 0.3$ with empirical evidence specific to the discrete vulnerability graph, filling a gap left by [4] which borrowed these values from a continuous-domain benchmark.

Third, the six-step attack chain identified in Table 3 — SQL injection (CVE-2020-13405) → reflected XSS (CVE-2020-13404) → stored XSS (CVE-2020-13407) → Apache SSRF (CVE-2021-40438) → OpenEMR file-upload RCE (CVE-2021-23919) → OpenSSH privilege escalation (CVE-2021-41617) — demonstrates that a default OpenEMR 6.0.0 deployment can be compromised to root in under 70 s on average when exploits are ordered correctly. All six CVEs are confirmed applicable to the software versions in Table 1, and the chain reproduced in all 49 successful runs, establishing it as a stable, reproducible attack surface rather than an artefact of a single experimental run.

Future work will extend the evaluation to multi-host segmented networks, validate the framework against current-year CVEs, and directly compare ACO path selection against deep reinforcement learning on the same attack graph with active intrusion detection in place.

References

1. ISC2. (2023). Cybersecurity Workforce Study 2023. ISC2, Alexandria, VA, USA. Available at: https://www.isc2.org/research

2. Rahalkar, S., & Jaswal, N. (2019). The Complete Metasploit Guide. Packt Publishing, Birmingham, UK.

3. Souppaya, M.P., & Scarfone, K.A. (2008). Technical Guide to Information Security Testing and Assessment. NIST Special Publication 800-115. NIST, Gaithersburg, MD. https://doi.org/10.6028/NIST.SP.800-115

4. He, Y., Zamani, E., Yevseyeva, I., & Luo, C. (2023). Artificial Intelligence–Based Ethical Hacking for Health Information Systems: Simulation Study. Journal of Medical Internet Research, 25, e41748. https://doi.org/10.2196/41748

5. Liu, L., Dai, Y., & Gao, J. (2014). Ant colony optimization algorithm for continuous domains based on position distribution model of ant colony foraging. The Scientific World Journal, 2014, 428539. https://doi.org/10.1155/2014/428539

eISSN: 3134-6057
pISSN: 3134-6049

6. Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics — Part B, 26(1), 29–41. https://doi.org/10.1109/3477.484436

7. Ghanem, M.C., & Chen, T.M. (2020). Reinforcement learning for efficient network penetration testing. Information, 11(1), 6. https://doi.org/10.3390/info11010006

8. Chaudhary, S., O'Brien, A., & Xu, S. (2020). Automated post-breach penetration testing through reinforcement learning. In Proceedings of the IEEE Conference on Communications and Network Security (CNS), Avignon, France (pp. 1–2). https://doi.org/10.1109/cns48642.2020.9162301

9. Yang, Q., & Lv, L.T. (2019). Network intrusion detection method based on combination of improved ant colony optimization and genetic algorithm. Journal of Chongqing University of Posts and Telecommunications, 29(1), 85–89.

10. Najera-Gutierrez, G., & Ansari, J.A. (2018). Web Penetration Testing with Kali Linux (3rd ed.). Packt Publishing, Birmingham, UK.

11. Yaqoob, T., Abbas, H., & Atiquzzaman, M. (2019). Security vulnerabilities, attacks, countermeasures, and regulations of networked medical devices — a review. IEEE Communications Surveys & Tutorials, 21(4), 3723–3768. https://doi.org/10.1109/COMST.2019.2914094

12. Jalali, M.S., & Kaiser, J.P. (2018). Cybersecurity in hospitals: a systematic, organizational perspective. Journal of Medical Internet Research, 20(5), e10059. https://doi.org/10.2196/10059

13. Martin, M.C., & Lam, M.S. (2008). Automatic generation of XSS and SQL injection attacks with goal-directed model checking. In Proceedings of the 17th USENIX Security Symposium, San Jose, CA (pp. 31–43).

14. Axinte, S.D. (2014). SQL injection testing in web applications using SQLmap. International Journal of Information Security and Cybercrime, 3(2), 61–68. https://doi.org/10.19107/ijisc.2014.02.07

15. Baloch, R. (2017). Ethical Hacking and Penetration Testing Guide. Auerbach Publications / CRC Press, Boca Raton, FL.

Information about authors

Zamzami Abdulmukhaymin Shauki -

Master's degree student, Faculty of Information Technology

L.N. Gumilyov Eurasian National University

E-mail: zamzami_ash@enu.kz

ORCID:https://orcid.org/0009-0001-2433-0536.