Σχολή Θετικών Σπουδών Και Τεχνολογίας

Πρόγραμμα Σπουδών Πληροφορικής

Πτυχιακή Εργασία

Μελέτη Εντοπισμού Επιθέσεων σε Web Εφαρμογές με Αξιοποίηση Knowledge Graphs, Vector Databases και LLMs πάνω σε TCP/IP Δεδομένα

Βασίλειος Χατζητούρνος

Επιβλέπων καθηγητής: Δρ. Βάιος Παπαϊωάννου

Πάτρα, Αύγουστος 2026

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Βασίλειου Χατζητούρνου που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο συγγραφέας/δημιουργός εκχωρεί στο ΕΑΠ, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού. Ο συγγραφέας/δημιουργός διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.

Μελέτη Εντοπισμού Επιθέσεων σε Web Εφαρμογές με Αξιοποίηση Knowledge Graphs, Vector Databases και LLMs πάνω σε TCP/IP Δεδομένα

Βασίλειος Χατζητούρνος

Επιτροπή Επίβλεψης Πτυχιακής Εργασίας

Επιβλέπων Καθηγητής:

Βάιος Παπαϊωάννου

Μέλος Εργαστηριακού Διδακτικού Προσωπικού (Ε.ΔΙ.Π) Τμήματος Μηχανικών Η/Υ & Πληροφορικής Πανεπιστήμιο Πατρών

Συν-Επιβλέπων Καθηγητής:

Χρήστος Κούτσικας

ΣΕΠ ΕΑΠ

Συν-Επιβλέπων Καθηγητής:

Νικόλαος Δημητρίου

Εντεταλμένος Ερευνητής, Ινστιτούτο Πληροφορικής & Τηλεπικοινωνιών, ΕΚΕΦΕ «Δημόκριτος»

Πάτρα, Αύγουστος 2026

Θα ήθελα να ευχαριστήσω τον κ. Βάιο Παπαϊωάννου για την καθοδήγηση, την εμπιστοσύνη και την ελευθερία που μου έδωσε κατά την εκπόνηση της πτυχιακής μου εργασίας. Η εμπειρία και τα εφόδια που αποκόμισα μου δίνουν αυτοπεποίθηση για τα επόμενα μου βήματα.

Επίσης ευχαριστώ τους γονείς μου, τα αδέρφια μου και όλους τους στενούς μου φίλους που για έξι χρόνια παρά τις αυξημένες υποχρεώσεις και τον περιορισμένο διαθέσιμο χρόνο μεταξύ μας, με στήριξαν και ήταν δίπλα μου σε αυτή μου τη προσπάθεια.

Τέλος, θα ήθελα να ευχαριστήσω ιδιαίτερα τον παιδικό μου φίλο Ανδρέα Εμερετλή, για την βοήθεια, την υποστήριξη και τις χρήσιμες συμβουλές του, συμβάλλοντας ουσιαστικά στην ολοκλήρωση της εργασίας μου.

Περίληψη

Η ασφάλεια των εφαρμογών διαδικτύου αποτελεί κρίσιμο ζήτημα για οργανισμούς και χρήστες, καθώς οι εφαρμογές αυτές διαχειρίζονται σημαντικό όγκο δεδομένων και αποτελούν συχνό στόχο κυβερνοεπιθέσεων. Μία από τις πλέον σημαντικές κατηγορίες επιθέσεων είναι η SQL injection (SQLi), κατά την οποία κακόβουλα δεδομένα εισόδου μπορούν να επηρεάσουν τα ερωτήματα που εκτελεί μια εφαρμογή προς τη βάση δεδομένων της. Τέτοιες επιθέσεις ενδέχεται να οδηγήσουν σε μη εξουσιοδοτημένη πρόσβαση, αποκάλυψη ή αλλοίωση δεδομένων, γεγονός που καθιστά αναγκαία την αποτελεσματική ανίχνευση και διερεύνησή τους.

Η παρούσα πτυχιακή εργασία εξετάζει την αξιοποίηση σύγχρονων τεχνικών τεχνητής νοημοσύνης και αναπαράστασης γνώσης για την υποστήριξη της ανάλυσης πιθανών SQL injection επιθέσεων σε δεδομένα δικτυακής κίνησης. Προτείνεται μια υβριδική αρχιτεκτονική που συνδυάζει τη δομημένη αναπαράσταση των στοιχείων της επικοινωνίας σε γράφο γνώσης, τη σημασιολογική ανάκτηση τεχνικής γνώσης από διανυσματική βάση δεδομένων και τη χρήση μεγάλων γλωσσικών μοντέλων για την ερμηνεία των διαθέσιμων τεκμηρίων. Η λειτουργία του συστήματος οργανώνεται μέσω πολλαπλών εξειδικευμένων agents, οι οποίοι ελέγχουν τη συνάφεια του ερωτήματος, επιλέγουν την κατάλληλη πηγή γνώσης, συνθέτουν την απάντηση και ελέγχουν την τεκμηρίωσή της.

Η προτεινόμενη προσέγγιση αξιολογήθηκε σε ελεγχόμενο εργαστηριακό περιβάλλον με δεδομένα δικτυακής κίνησης που περιλάμβαναν κανονική δραστηριότητα, οριακές ή μη κακόβουλες περιπτώσεις και σενάρια με HTTP artifacts συμβατά με διαφορετικές τεχνικές SQL injection. Τα αποτελέσματα έδειξαν ότι, στο συγκεκριμένο πειραματικό σύνολο, το σύστημα μπορεί να αναγνωρίζει και να διακρίνει τις εξεταζόμενες περιπτώσεις, παρέχοντας παράλληλα επεξηγηματικές απαντήσεις συνδεδεμένες με τα διαθέσιμα τεχνικά στοιχεία. Επιπλέον, η αξιολόγηση ανέδειξε ότι η ορθή ταξινόμηση της ύπαρξης ενδείξεων SQL injection μπορεί να συνοδεύεται από διαφορετικό βαθμό λεπτομέρειας ως προς τον ακριβή τύπο της πιθανής επίθεσης, ιδίως όταν παρατηρούνται πολλαπλά ή σύνθετα μοτίβα δραστηριότητας.

Συνολικά, η εργασία δείχνει ότι η συνδυασμένη αξιοποίηση γράφων γνώσης, διανυσματικής ανάκτησης και συστημάτων πολλαπλών agents μπορεί να λειτουργήσει ως χρήσιμο εργαλείο υποστήριξης αναλυτών ασφαλείας. Η προτεινόμενη προσέγγιση δεν αποσκοπεί στην

αντικατάσταση καθιερωμένων μηχανισμών προστασίας, αλλά στην ενίσχυση της διερεύνησης περιστατικών μέσω της συσχέτισης τεχνικών τεκμηρίων και της παροχής κατανοητών, τεκμηριωμένων απαντήσεων.

Λέξεις – Κλειδιά

Γράφοι Γνώσης, Διανυσματικές Βάσεις Δεδομένων, Μεγάλα Γλωσσικά Μοντέλα (LLMs), SQL injection, κίνηση HTTP

Abstract

Web application security is a critical concern for organizations and users, as such applications manage significant volumes of data and constitute frequent targets of cyberattacks. One of the most important categories of attacks is SQL injection (SQLi), in which malicious input data can affect the queries executed by an application against its database. Such attacks may lead to unauthorized access, disclosure, or modification of data, making their effective detection and investigation necessary.

This thesis examines the use of modern artificial intelligence and knowledge representation techniques to support the analysis of potential SQL injection attacks in network traffic data. A hybrid architecture is proposed that combines the structured representation of communication elements in a knowledge graph, the semantic retrieval of technical knowledge from a vector database, and the use of large language models to interpret the available evidence. The operation of the system is organized through multiple specialized agents, which assess the relevance of the user query, select the appropriate knowledge source, synthesize the response, and verify its grounding.

The proposed approach was evaluated in a controlled laboratory environment using network-traffic data that included normal activity, borderline or non-malicious cases, and scenarios containing HTTP artifacts consistent with different SQL injection techniques. The results showed that, within the examined experimental dataset, the system can identify and distinguish these cases while also providing explanatory responses linked to the available technical evidence. Furthermore, the evaluation indicated that correct classification of the presence of SQL injection indications may be accompanied by varying levels of detail regarding the precise type of the possible attack, particularly when multiple or complex activity patterns are observed.

Overall, the thesis demonstrates that the combined use of knowledge graphs, vector retrieval, and multi-agent systems can function as a useful support tool for security analysts. The proposed approach is not intended to replace established protection mechanisms, but rather to enhance incident investigation through the correlation of technical evidence and the provision of understandable, well-grounded responses.

Keywords

Knowledge Graphs, Vector Databases, Large Language Models (LLMs), SQL injection, HTTP traffic

Περιεχόμενα

Περίληψη v
Abstract vii
Περιεχόμενα ix
Κατάλογος Εικόνων xi
Κατάλογος Πινάκων xii
Συντομογραφίες & Ακρωνύμια xiii
1. Εισαγωγή 1
1.1 Αντικείμενο και Στόχοι της Εργασίας 2
1.2 Ερευνητικό Πρόβλημα και Αναγκαιότητα της Μελέτης 3
1.3 Διάρθρωση της Εργασίας 4
2. Κίνδυνοι Εφαρμογών Διαδικτύου και SQL Injection 5
2.1 Το OWASP Top 10 ως Πλαίσιο Κινδύνων 5
2.2 SQL Injection: Μηχανισμός και Επιπτώσεις 6
2.3 Τεχνικές SQL Injection και Ανιχνεύσιμα Ίχνη 7
2.4 Ενδείξεις, Διερεύνηση και Μέτρα Αντιμετώπισης 9
2.5 Συμπεράσματα 11
3. AI και Ανάκτηση Γνώσης στην Κυβερνοασφάλεια 13
3.1 Μεγάλα Γλωσσικά Μοντέλα και Περιορισμοί Αξιοπιστίας 14
3.2 Retrieval Augmented Generation (RAG) 16
3.2.1 Η Εξελικτική Διαδρομή: Naive, Advanced και Modular RAG 17
3.2.2 Στρατηγικές Ανάκτησης (Retrieval Strategies) 20
3.2.3 Διαχείριση και Αναπαράσταση Γνώσης 22
3.3 Agentic RAG και Συστήματα Πολλαπλών Agents 24
3.3.1 Agents, Εργαλεία και Δυναμική Δρομολόγηση 25
3.3.2 Ροές Επαλήθευσης και Διορθωτικής Ανάκτησης 26
3.4 Εφαρμογή στην Κυβερνοασφάλεια 28
3.4.1 Σχετική Βιβλιογραφία και Τοποθέτηση της Προτεινόμενης Προσέγγισης 29
3.4.2 Threat Hunting και Ανάλυση Περιστατικών 29
3.4.2 Σύνδεση με το NIST CSF 2.0 31
4. Σχεδιασμός και Αρχιτεκτονική Συστήματος 32
4.1 Συνολική Αρχιτεκτονική Συστήματος 32
4.2 Διαδικασία Εισαγωγής και Προεπεξεργασίας Δεδομένων (Data Ingestion and Preprocessing Pipeline) – 1ο Επίπεδο (1st Layer) 34
4.2.1 Ροή Εισαγωγής και Προεπεξεργασίας Δεδομένων στη Διανυσματική Βάση Qdrant 35
4.2.2 Ροή Εισαγωγής και Προεπεξεργασίας Δεδομένων στη Γραφική Βάση Neo4j 36
4.3 Εξαγωγή Γνώσης (Knowledge Extraction) - 2ο Επίπεδο (2nd Layer) 38

4.3.1 Διανυσματοποίηση και Δημιουργία Embeddings 38
4.3.2. Αναγνώριση Οντοτήτων και Εξαγωγή Σχέσεων 39
4.4 Υβριδική Υποδομή Γνώσης (Hybrid Knowledge Infrastructure) - 3ο Επίπεδο (3rd Layer) 42
4.4.1 Διανυσματική Βάση Δεδομένων Qdrant 42
4.4.2 Γραφική Βάση Δεδομένων Neo4j 47
4.5 Υποσύστημα Agentic RAG (Multi Agent System - MAS) – 4ο Επίπεδο (4th Layer) 50
4.5.1 Αρχιτεκτονική Συστήματος Πολλαπλών Agents 51
4.5.2 Relevance Checker Agent 52
4.5.3 RAG Agent 53
4.5.4 Research Agent 54
4.5.5 Verification Agent 55
4.5.6 Ενορχήστρωση των Agents 56
4.6 Διεπαφή Χρήστη και Υποστηρικτικές Υπηρεσίες – 5ο Επίπεδο (5th Layer) 60
4.6.1 Διεπαφή Χρήστη (UI) 60
4.6.2 Επίπεδο Διεπαφής Προγραμματισμού Εφαρμογών (API) 62
5. Πειραματική Διαδικασία και Αξιολόγηση 64
5.1 Πειραματικό Περιβάλλον και Σύνολο Δεδομένων (Dataset & Setup) 64
5.1.1 Εργαστηριακή Υποδομή 65
5.1.2 Σύνολο Δικτυακών Δεδομένων PCAP 66
5.1.3 Συλλογή Εξειδικευμένης Γνώσης SQL Injection 69
5.2 Μεθοδολογία Αξιολόγησης 70
5.2.1 Πλαίσιο Εκτέλεσης και Αξιολόγησης με LangSmith 70
5.2.2 Σύνολα Αξιολόγησης και Κριτήρια Βαθμολόγησης 71
5.3 Αξιολόγηση Λειτουργίας του Agentic RAG 73
5.4 Αξιολόγηση Αναγνώρισης και Ταξινόμησης SQL Injection 77
5.4.1 Συγκεντρωτική Ερμηνεία Αποτελεσμάτων SQLi Ανίχνευσης και Ταξινόμησης 79
5.4.2 Αξιολόγηση Δυαδικής Ανίχνευσης 80
5.4.3 Ανάλυση Χαρακτηριστικών Περιπτώσεων 81
5.5 Συζήτηση Αποτελεσμάτων 83
6. Συμπεράσματα & Μελλοντικές Προεκτάσεις 86
6.1 Κύρια Συμπεράσματα 86
6.2 Περιορισμοί της Προσέγγισης 88
6.3 Προτάσεις για Μελλοντική Έρευνα 89
Βιβλιογραφία 92
Παράρτημα Α: Κώδικας Υλοποίησης της Εργασίας 96
Παράρτημα Β: Σύνολα Αξιολόγησης 129

Κατάλογος Εικόνων

Εικόνα 1 Κατηγορίες Επιθέσεων SQLI 8
Εικόνα 2 Ανάλυση Modular RAG 19
Εικόνα 3 Σύγκριση μεταξύ Iterative, Recursive & Adaptive Retrieval (Y. Gao et al., 2023) 21
Εικόνα 4 Ροή Εργασίας Evaluator-Optimizer (Building Effective AI Agents \ Anthropic, n.d.) 27
Εικόνα 5 Συνολική Αρχιτεκτονική Συστήματος 34
Εικόνα 6 Σχηματική Απεικόνιση της Ιεραρχικής Δομής του HNSW 44
Εικόνα 7 Σχήμα Neo4j Graph DB 50
Εικόνα 8 Αρχιτεκτονική Multi-Agent Sub-System 52
Εικόνα 9 Στατική απεικόνιση του γράφου ενορχήστρωσης των Agents με χρήση LangGraph 57
Εικόνα 10 Διεπαφή χρήστη του συστήματος ανάλυσης SQLi, με περιοχή συνομιλίας και στατική απεικόνιση του Agentic workflow 61
Εικόνα 11 Ροή Καταγραφής Δικτυακής Κίνησης και Εισαγωγής στη Neo4j 66
Εικόνα 12 Αποτελέσματα Αξιολόγησης Agentic RAG Quality 74
Εικόνα 13 Συνολική Απόδοση Agentic Συστήματος 74
Εικόνα 14 Αξιολόγηση Ανίχνευσης και Ταξινόμησης SQLi Επιθέσεων 78
Εικόνα 15 Συνολική Απόδοση Συστήματος στην Αναγνώριση και Ταξινόμηση SQLi Επιθέσεων 79

Κατάλογος Πινάκων

Πίνακας 1 Τεχνικές Προδιαγραφές Αρχιτεκτονικής Συστήματος 33
Πίνακας 2 Βασικά Πεδία WorkflowState Συστήματος Πολλαπλών Agents 58
Πίνακας 3 Σύνολο Αρχείων PCAP και Εκτελεσθέντα Σενάρια SQLi 68
Πίνακας 4 Μετρικές Βαθμολόγησης Agentic RAG Συστήματος 72
Πίνακας 5 Μετρικές Αξιολόγησης Ανίχνευσης SQLi Επιθέσεων 73
Πίνακας 6 Πίνακας Σύγχυσης για την Αναγνώριση SQLi σε Επίπεδο PCAP 80
Πίνακας 7 Αποτελέσματα Μετρικών Δυαδικής Ταξινόμησης 81

Συντομογραφίες & Ακρωνύμια

AI	Artificial Intelligence
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
DBMS	Data Base Management System
DNS	Domain Name System
HTTP	Hyper Text Transfer Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
LDAP	Lightweight Directory Access Protocol
LLM	Large Language Model
NER	Named Entity Recognition
NLP	Natural Language Process
OWASP	Open Worldwide Application Security Project (OWASP)
OWL	Web Ontology Language
PCAP	Packet Capture
RAG	Retrieval Augmented Generation
RDF	Resource Description Framework
SIEM	Security Information and Event Management
SOAR	Security Orchestration Automation and Response
SQL	Structured Query Language
SQLi	SQL injection
TLS	Transport Layer Security
URI	Uniform Resource Identifier
WAF	Web Application Firewall
XDR	Extended Detection and Response

1. Εισαγωγή

Οι εφαρμογές διαδικτύου αποτελούν βασική υποδομή για την παροχή ψηφιακών υπηρεσιών, καθώς υποστηρίζουν ηλεκτρονικές συναλλαγές, πληροφοριακά συστήματα οργανισμών, υπηρεσίες υγείας, πλατφόρμες κοινωνικής δικτύωσης και εφαρμογές ηλεκτρονικής διακυβέρνησης. Η διαχείριση δεδομένων μέσω διαδικτυακών εφαρμογών δημιουργεί αυξημένες απαιτήσεις ασφάλειας, ιδιαίτερα όταν η εφαρμογή επικοινωνεί με βάσεις δεδομένων και δέχεται δεδομένα εισόδου από μη έμπιστους χρήστες.

Μία από τις πλέον διαδεδομένες και κρίσιμες κατηγορίες ευπαθειών σε εφαρμογές διαδικτύου είναι η SQL injection (SQLi). Μέσω της εισαγωγής κατάλληλα διαμορφωμένων δεδομένων σε πεδία εισόδου, ένας επιτιθέμενος μπορεί να επηρεάσει τα SQL ερωτήματα που εκτελεί η εφαρμογή και, υπό προϋποθέσεις, να αποκτήσει μη εξουσιοδοτημένη πρόσβαση σε δεδομένα, να αλλοιώσει εγγραφές ή να ανακτήσει πληροφορίες σχετικά με το σχήμα και τη λειτουργία της βάσης δεδομένων. Η διερεύνηση τέτοιων περιστατικών δεν περιορίζεται στην αναγνώριση ενός μεμονωμένου payload, αλλά απαιτεί την αξιοποίηση του ευρύτερου πλαισίου της δικτυακής κίνησης και επικοινωνίας επιπέδου εφαρμογής.

Η αυξανόμενη διαθεσιμότητα δεδομένων καταγραφής, όπως αρχεία PCAP, HTTP καταγραφές και στοιχεία επιπέδου εφαρμογής, δημιουργεί τη δυνατότητα συστηματικότερης ανάλυσης ύποπτης δραστηριότητας. Παράλληλα, η χρήση μεγάλων γλωσσικών μοντέλων (LLMs), τεχνικών Retrieval-Augmented Generation και υποδομών γνώσης μπορεί να υποστηρίξει την ερμηνεία τεχνικών artifacts από διαφορετικές πηγές. Ειδικότερα, οι διανυσματικές βάσεις δεδομένων επιτρέπουν την ανάκτηση σημασιολογικά συναφούς τεχνικής γνώσης, ενώ οι γράφοι γνώσης διευκολύνουν τη ρητή αναπαράσταση και συσχέτιση οντοτήτων, όπως αρχεία PCAP, αιτήματα HTTP , διευθύνσεις IP, hosts και παράμετροι.

Η παρούσα εργασία εξετάζει την ανάπτυξη ενός Agentic HybridRAG συστήματος υποστήριξης διερεύνησης πιθανών SQL injection περιστατικών σε εφαρμογές διαδικτύου. Η προτεινόμενη υλοποίηση εξάγει στοιχεία HTTP από αρχεία PCAP, τα μετασχηματίζει σε γράφο γνώσης και αξιοποιεί παράλληλα συλλογή δεδομένων εξειδικευμένης γνώσης SQL injection σε διανυσματική βάση. Πάνω σε αυτή την υβριδική υποδομή, ένα σύστημα πολλαπλών agents αναλαμβάνει τον έλεγχο συνάφειας του ερωτήματος, την επιλογή της

κατάλληλης στρατηγικής ανάκτησης, τη σύνθεση τεκμηριωμένης απάντησης και την επαλήθευση της.

Η προτεινόμενη προσέγγιση δεν αποσκοπεί στην αντικατάσταση συστημάτων IDS, WAF ή εξειδικευμένων μηχανισμών incident response. Αντίθετα, σχεδιάζεται ως εργαλείο υποστήριξης αναλυτών ασφαλείας, το οποίο συνδέει παρατηρούμενα HTTP artifacts με τεχνική γνώση SQL injection και παράγει επεξηγηματικές απαντήσεις βασισμένες στα διαθέσιμα τεκμήρια.

1.1 Αντικείμενο και Στόχοι της Εργασίας

Αντικείμενο της παρούσας εργασίας είναι η μελέτη, σχεδίαση, υλοποίηση και αξιολόγηση μιας αρχιτεκτονικής υποστήριξης της διερεύνησης ενδείξεων πιθανών επιθέσεων SQL injection σε εφαρμογές διαδικτύου. Η αρχιτεκτονική αξιοποιεί δεδομένα HTTP που εξάγονται από αρχεία PCAP, γράφο γνώσης, διανυσματική βάση τεχνικής γνώσης και ροή εργασίας πολλαπλών agents βασισμένη σε μεγάλα γλωσσικά μοντέλα.

Οι ειδικότεροι στόχοι της εργασίας είναι οι ακόλουθοι:

- Η εξαγωγή χαρακτηριστικών HTTP από αρχεία PCAP, συμπεριλαμβανομένων URI, παραμέτρων, μεθόδων HTTP, request bodies, cookies, διευθύνσεων IP, hosts και χρονικών σημάνσεων.
- Η μετατροπή των εξαγόμενων στοιχείων σε γράφο γνώσης, με σκοπό τη συσχέτιση των τεχνικών artifacts και την υποστήριξη ανάκτησης μέσω του γράφου.
- Η κατασκευή συλλογής εξειδικευμένης γνώσης SQL injection στη διανυσματική βάση, με αξιοποίηση πυκνών και αραιών διανυσματικών αναπαραστάσεων για σημασιολογική και λεξική ανάκτηση.
- Η υλοποίηση HybridRAG μηχανισμού που μπορεί να αξιοποιεί ανεξάρτητα ή συνδυαστικά τη γραφική και τη διανυσματική πηγή γνώσης.
- Η ανάπτυξη υποσυστήματος πολλαπλών agents για έλεγχο συνάφειας, επιλογή εργαλείων ανάκτησης, σύνθεση απάντησης και έλεγχο τεκμηρίωσης.

Με βάση τους παραπάνω επιμέρους στόχους, κύριος σκοπός της εργασίας είναι η διερεύνηση του τρόπου με τον οποίο σύγχρονες τεχνικές Retrieval Augmented Generation και Agentic AI

μπορούν να αξιοποιηθούν για την υποστήριξη της ανίχνευσης και της αναλυτικής διερεύνησης πιθανών επιθέσεων SQL injection σε εφαρμογές διαδικτύου. Ειδικότερα, εξετάζεται κατά πόσο η σύζευξη μιας υβριδικής υποδομής γνώσης, η οποία συνδυάζει γραφική αναπαράσταση των παρατηρούμενων στοιχείων δικτυακής κίνησης με διανυσματική ανάκτηση εξειδικευμένης τεχνικής γνώσης, μπορεί να ενισχύσει την ποιότητα, την τεκμηρίωση και την επεξηγησιμότητα των απαντήσεων RAG. Παράλληλα, διερευνάται η συμβολή μιας αρχιτεκτονικής πολλαπλών agents στη δυναμική επιλογή στρατηγικών ανάκτησης, στη σύνθεση των ευρημάτων και στον έλεγχο της τελικής απόκρισης. Κατ' επέκταση, η υλοποίηση της εφαρμογής λειτουργεί ως πειραματικό πλαίσιο για την αξιολόγηση του κατά πόσο οι προσεγγίσεις Agentic HybridRAG μπορούν να μετατρέψουν τεχνικά artifacts από δεδομένα HTTP και PCAP σε χρήσιμη, τεκμηριωμένη και κατανοητή πληροφορία για την υποστήριξη αναλυτών ασφαλείας.

1.2 Ερευνητικό Πρόβλημα και Αναγκαιότητα της Μελέτης

Το ερευνητικό πρόβλημα αφορά την υποστήριξη της διερεύνησης SQL injection περιστατικών όταν τα διαθέσιμα τεχνικά στοιχεία είναι κατανεμημένα μεταξύ δεδομένων δικτυακής κίνησης, παραμέτρων εφαρμογής και εξωτερικής εξειδικευμένης γνώσης. Οι συμβατικοί μηχανισμοί ανίχνευσης βασίζονται συχνά σε στατικές υπογραφές, κανόνες ή ταξινομητές, οι οποίοι μπορούν να αναγνωρίσουν συγκεκριμένα μοτίβα, αλλά δεν παρέχουν απαραίτητα ολοκληρωμένη ερμηνεία του πλαισίου ενός περιστατικού, ούτε σύνδεση της παρατηρούμενης δικτυακής δραστηριότητας με τεκμηριωμένη γνώση για τις αντίστοιχες τεχνικές επίθεσης.

Η αναγκαιότητα της μελέτης προκύπτει από την ανάγκη περιορισμού του γνωστικού φόρτου κατά τη διερεύνηση ύποπτης δικτυακής δραστηριότητας. Ένας αναλυτής καλείται συχνά να συνδυάσει στοιχεία από αρχεία PCAP, αρχεία καταγραφών, παραμέτρους HTTP και τεχνική γνώση από διαφορετικές πηγές. Η χρήση συστημάτων όπως αυτό της προτεινόμενης υλοποίησης, δεν υποκαθιστά την ανθρώπινη κρίση, αλλά μπορεί να προσφέρει οργανωμένο μηχανισμό ανάκτησης, συσχέτισης και παρουσίασης τεκμηρίων, διευκολύνοντας την αρχική ανάλυση και την επιθεώρηση πιθανών περιστατικών SQLi.

Στην παρούσα εργασία, η ανάλυση περιορίζεται στην ανάδειξη και ερμηνεία HTTP artifacts που είναι συμβατά με πιθανή δραστηριότητα SQL injection. Η παρουσία τέτοιων στοιχείων σε

αρχεία PCAP δεν αποτελεί από μόνη της επιβεβαίωση ότι εκτελέστηκε επιτυχώς κακόβουλο SQL ερώτημα ή ότι προκλήθηκε παραβίαση της εφαρμογής ή της βάσης δεδομένων.

1.3 Διάρθρωση της Εργασίας

Η εργασία οργανώνεται σε έξι κεφάλαια. Το δεύτερο κεφάλαιο οριοθετεί το πρόβλημα της ασφάλειας εφαρμογών διαδικτύου και επικεντρώνεται στις επιθέσεις SQL injection, στις βασικές τεχνικές τους, στα τεχνικά ίχνη που μπορούν να εμφανιστούν σε επικοινωνία HTTP και στα διαθέσιμα μέτρα πρόληψης και διερεύνησης.

Το τρίτο κεφάλαιο παρουσιάζει το τεχνολογικό υπόβαθρο της προτεινόμενης προσέγγισης. Εξετάζονται οι περιορισμοί των μεγάλων γλωσσικών μοντέλων, οι τεχνικές Retrieval-Augmented Generation, οι διανυσματικές και γραφικές αναπαραστάσεις γνώσης, η προσέγγιση HybridRAG και οι ροές εργασίας Agentic RAG. Η παρουσίαση αυτή αιτιολογεί τις βασικές επιλογές της αρχιτεκτονικής που υλοποιείται στη συνέχεια.

Στο τέταρτο κεφάλαιο παρουσιάζεται αναλυτικά ο σχεδιασμός και η υλοποίηση του συστήματος. Περιγράφονται η εξαγωγή στοιχείων HTTP από αρχεία PCAP, η δημιουργία του γράφου γνώσης, η οργάνωση του συνόλου γνώσης στη διανυσματική βάση δεδομένων, η αρχιτεκτονική και η ενορχήστρωση των agents και η διεπαφή χρήστη.

Το πέμπτο κεφάλαιο παρουσιάζει το εργαστηριακό περιβάλλον, τα σύνολα αξιολόγησης και τα αποτελέσματα των πειραμάτων. Αξιολογούνται τόσο η λειτουργία της ροή εργασίας του Agentic RAG, όσο και η αναγνώριση και ταξινόμηση SQL injection σε επίπεδο αρχείου PCAP. Τέλος, στο έκτο κεφάλαιο συνοψίζονται τα συμπεράσματα, οι περιορισμοί της προσέγγισης και οι προτάσεις για μελλοντική έρευνα.

2. Κίνδυνοι Εφαρμογών Διαδικτύου και SQL Injection

Στη σημερινή εποχή η χρήση των εφαρμογών διαδικτύου έχει αυξηθεί με ραγδαίους ρυθμούς λόγω της μεγάλης επέκτασης του διαδικτύου που έχει σημειωθεί τις τελευταίες δύο δεκαετίες. Όλο και περισσότεροι χρήστες χρησιμοποιούν καθημερινά αντίστοιχες εφαρμογές, καθώς πολλές από τις καθημερινές μας ανάγκες ή δραστηριότητες έχουν κάνει τη μετάβαση τους στο ψηφιακό κόσμο. Ηλεκτρονικές αγορές, χρήση κοινωνικών δικτύων, ακόμη και τραπεζικές συναλλαγές αποτελούν ορισμένες από αυτές και όλο και περισσότεροι χρήστες επιλέγουν την πραγματοποίηση τους μέσω εφαρμογών διαδικτύου.

Η λειτουργία των εφαρμογών αυτών περιλαμβάνει και τον χειρισμό τεράστιου αριθμού δεδομένων, τα οποία δύναται να είναι ευαίσθητα και προσωπικά όπως π.χ. ιατρικά δεδομένα. Όπως γίνεται αντιληπτό η ασφάλεια των εν λόγω εφαρμογών είναι μείζονος σημασίας και περιλαμβάνει και την προστασία των δεδομένων που επεξεργάζονται (Tadhani et al., 2024).

2.1 Το OWASP Top 10 ως Πλαίσιο Κινδύνων

Το OWASP Top 10 αποτελεί ένα ευρέως χρησιμοποιούμενο πλαίσιο αναφοράς για τους σημαντικότερους κινδύνους ασφάλειας εφαρμογών διαδικτύου. Παρουσιάζει κατηγορίες αδυναμιών που αφορούν, μεταξύ άλλων, τον έλεγχο πρόσβασης, τη μη ασφαλή διαμόρφωση, τις αστοχίες κρυπτογραφίας, την ελλιπή καταγραφή συμβάντων και τις ευπάθειες έγχυσης. Ο σκοπός του δεν είναι να λειτουργήσει ως πλήρης κατάλογος όλων των πιθανών απειλών, αλλά να υποστηρίξει την ιεράρχηση κινδύνων και την εφαρμογή κατάλληλων μέτρων ασφάλειας κατά τον σχεδιασμό, την ανάπτυξη και τη λειτουργία εφαρμογών διαδικτύου (*OWASP Top Ten Web Application Security Risks | OWASP Foundation*, n.d.).

Για την παρούσα εργασία, ιδιαίτερη σημασία έχει η κατηγορία **Injection**, η οποία στην έκδοση OWASP Top 10:2025 περιλαμβάνει ευπάθειες όπου μη έμπιστα δεδομένα εισόδου ερμηνεύονται από κάποιον διερμηνευτή ως εντολές ή τμήμα εντολών. Στην κατηγορία αυτή εντάσσονται, μεταξύ άλλων, η SQL injection, η NoSQL injection, η LDAP injection και η command injection. Η SQL injection συνδέεται ειδικότερα με την αδυναμία ορθού διαχωρισμού των δεδομένων εισόδου από τον κώδικα SQL που εκτελεί η εφαρμογή προς τη βάση δεδομένων (*OWASP Top 10:2025*, n.d.).

Η ταξινόμηση της SQL injection στο πλαίσιο του OWASP αναδεικνύει τη σημασία της ως απειλής για εφαρμογές που δέχονται δεδομένα χρήστη και επικοινωνούν με βάσεις δεδομένων. Ωστόσο, η διερεύνηση μιας πιθανής επίθεσης SQLi δεν περιορίζεται στην αναγνώριση της ευπάθειας σε επίπεδο κώδικα. Απαιτεί επίσης την ανάλυση τεχνικών artifacts, όπως URI, παράμετροι HTTP, request bodies, cookies, χρονικές σημάνσεις και άλλα στοιχεία της δικτυακής κίνησης ή επικοινωνίας επιπέδου εφαρμογής. Η επόμενη ενότητα παρουσιάζει τη φύση της SQL injection και τις βασικές μορφές με τις οποίες μπορεί να εμφανιστεί σε περιβάλλοντα διαδικτύου.

2.2 SQL Injection: Μηχανισμός και Επιπτώσεις

Η SQL injection (SQLi) αποτελεί ευπάθεια που εμφανίζεται όταν μια εφαρμογή διαδικτύου ενσωματώνει δεδομένα εισόδου του χρήστη σε εντολές SQL χωρίς να εφαρμόζει κατάλληλο διαχωρισμό μεταξύ δεδομένων και κώδικα. Σε μια τέτοια περίπτωση, ένας επιτιθέμενος μπορεί να διαμορφώσει ειδικά κατασκευασμένες τιμές εισόδου, ώστε να επηρεάσει τη σύνταξη ή τη λογική του ερωτήματος που αποστέλλεται από την εφαρμογή προς τη βάση δεδομένων. Η ευπάθεια δεν σχετίζεται με τη γλώσσα SQL αυτή καθαυτή, αλλά με τον μη ασφαλή τρόπο με τον οποίο η εφαρμογή χειρίζεται μη έμπιστα δεδομένα εισόδου κατά τη δημιουργία ερωτημάτων προς τη βάση (Clarke, 2012).

Μια επιτυχημένη επίθεση SQLi μπορεί να επιτρέψει την παράκαμψη μηχανισμών αυθεντικοποίησης, την ανάκτηση ευαίσθητων δεδομένων, την αποκάλυψη πληροφοριών σχετικά με το σχήμα της βάσης ή τη μεταβολή και διαγραφή αποθηκευμένων δεδομένων. Ανάλογα με τα δικαιώματα του λογαριασμού που χρησιμοποιεί η εφαρμογή για τη σύνδεσή της με τη βάση, η επίπτωση ενδέχεται να επεκτείνεται πέρα από την πρόσβαση σε μεμονωμένους πίνακες και να επηρεάζει συνολικότερα τη διαθεσιμότητα, την ακεραιότητα και την εμπιστευτικότητα των δεδομένων. Ενδεικτικά, η SQLi μπορεί να οδηγήσει σε αποκάλυψη ονομάτων χρηστών, κωδικών πρόσβασης, στοιχείων επικοινωνίας ή οικονομικών δεδομένων, καθώς και σε μη εξουσιοδοτημένες ενέργειες πάνω στα αποθηκευμένα δεδομένα (Clarke, 2012).

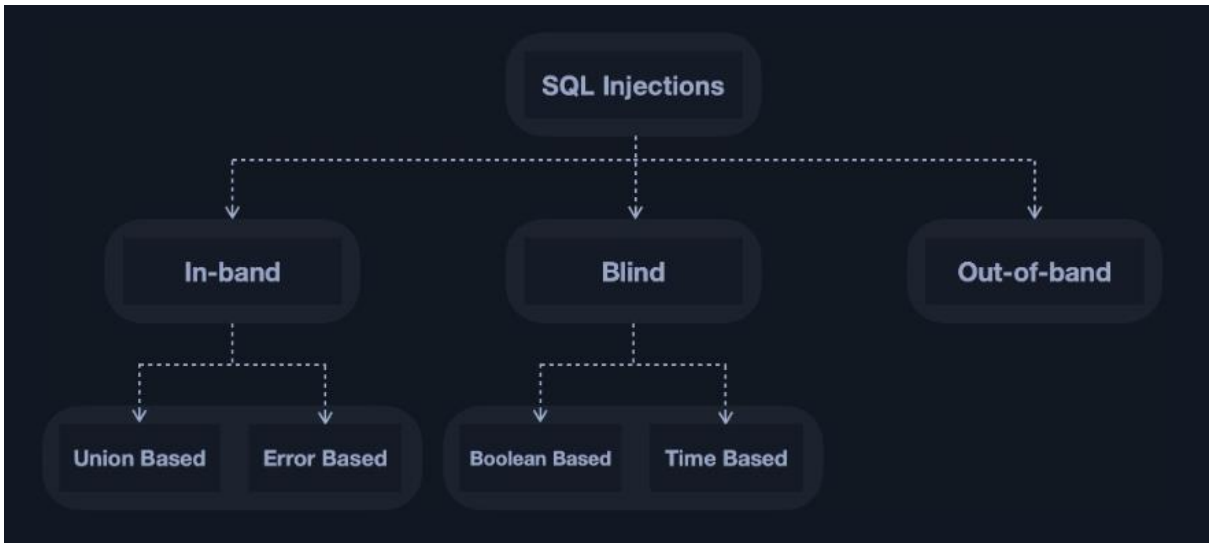
Η SQL injection αντιστοιχεί στην αδυναμία CWE-89, με τίτλο *Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')*. Η αντιστοίχιση αυτή αναδεικνύει ότι το πρόβλημα έγκειται στην ανεπαρκή εξουδετέρωση ειδικών στοιχείων που μπορούν να μετατρέψουν την είσοδο ενός χρήστη σε εκτελέσιμο τμήμα ενός SQL ερωτήματος. Παρότι η SQLi μπορεί να εμφανίζεται με διαφορετικές μορφές και διαφορετικό βαθμό πολυπλοκότητας, η κοινή της βάση είναι η δυνατότητα αλλοίωσης της προβλεπόμενης λογικής μιας εφαρμογής μέσω κακόβουλων δεδομένων εισόδου. Η SQLi είναι μια επίθεση χαμηλής συχνότητας, αλλά μεγάλης επίδρασης με πάνω από 14 χιλιάδες CVEs (*OWASP Top 10:2025*, n.d.).

Στο πλαίσιο της παρούσας εργασίας, η SQL injection εξετάζεται από την πλευρά των ανιχνεύσιμων τεχνικών artifacts που μπορεί να αφήνει στη δικτυακή κίνηση και επικοινωνία επιπέδου εφαρμογής. Τα στοιχεία αυτά μπορεί να περιλαμβάνουν τιμές παραμέτρων σε URI, query strings, request bodies ή cookies, οι οποίες περιέχουν λέξεις κλειδιά SQL, λογικές εκφράσεις, συναρτήσεις βάσης δεδομένων ή άλλες συμβολοσειρές που σχετίζονται με προσπάθεια χειρισμού ενός SQL ερωτήματος. Η παρουσία ενός τέτοιου στοιχείου δεν αρκεί από μόνη της για να επιβεβαιώσει την ύπαρξη επίθεσης, καθώς απαιτείται η συνεκτίμηση του ευρύτερου πλαισίου του αιτήματος HTTP και των υπόλοιπων διαθέσιμων τεκμηρίων. Στην συνέχεια παρουσιάζονται οι βασικές τεχνικές SQL injection και τα αντίστοιχα ανιχνεύσιμα ίχνη τους.

2.3 Τεχνικές SQL Injection και Ανιχνεύσιμα Ίχνη

Οι επιθέσεις SQL injection μπορούν να κατηγοριοποιηθούν με βάση τον τρόπο αλληλεπίδρασης του επιτιθέμενου με την εφαρμογή και το κανάλι από το οποίο προκύπτει η πληροφορία σχετικά με τη συμπεριφορά της βάσης δεδομένων. Μία ευρέως χρησιμοποιούμενη κατηγοριοποίηση διακρίνει τις επιθέσεις σε **In-Band**, **Blind** και **Out-of-Band** SQL injection, όπως φαίνεται στην Εικόνα 1. Στις In-Band επιθέσεις, τα αποτελέσματα της εκμετάλλευσης επιστρέφονται μέσω του ίδιου καναλιού που χρησιμοποιήθηκε για την αποστολή του κακόβουλου αιτήματος. Στις Blind επιθέσεις, ο επιτιθέμενος δεν λαμβάνει απευθείας τα δεδομένα της βάσης, αλλά εξάγει συμπεράσματα από διαφοροποιήσεις στην

απόκριση της εφαρμογής ή στον χρόνο απόκρισης. Τέλος, στις Out-of-Band επιθέσεις, η εξαγωγή δεδομένων ή η επιβεβαίωση της εκμετάλλευσης μπορεί να πραγματοποιείται μέσω διαφορετικού εξερχόμενου καναλιού, όπως HTTP ή DNS (Lee, 2019).



Εικόνα 1 Κατηγορίες Επιθέσεων SQLI

Στην κατηγορία In-Band εντάσσονται κυρίως οι **Error-based** και οι **UNION-based** επιθέσεις. Η Error-based SQLi αξιοποιεί μηνύματα σφάλματος που παράγονται από τη βάση δεδομένων ή την εφαρμογή, προκειμένου να αποκαλυφθούν πληροφορίες σχετικά με το σχήμα της βάσης, τον τύπο του συστήματος διαχείρισης βάσεων δεδομένων ή ακόμη και επιμέρους δεδομένα. Η UNION-based SQLi βασίζεται στη χρήση του τελεστή UNION, ώστε ένα κακόβουλα διαμορφωμένο ερώτημα να συνδυάζεται με το αρχικό ερώτημα της εφαρμογής και να επιστρέφει δεδομένα που κανονικά δεν θα ήταν προσβάσιμα στον χρήστη (Al-Olaqi et al., 2025; Lee, 2019).

Οι **Boolean-based blind** και **Time-based blind** επιθέσεις ανήκουν στην κατηγορία Blind SQLi. Στην πρώτη περίπτωση, ο επιτιθέμενος διαμορφώνει λογικές συνθήκες που αξιολογούνται ως αληθείς ή ψευδείς και παρατηρεί διαφορές στο περιεχόμενο, στην κατάσταση ή στη συμπεριφορά της απόκρισης της εφαρμογής. Στη δεύτερη περίπτωση, η επιβεβαίωση βασίζεται στην πρόκληση τεχνητής καθυστέρησης κατά την εκτέλεση του ερωτήματος, μέσω συναρτήσεων ή εντολών χρονικής αναμονής, και στην παρατήρηση του αντίστοιχου χρόνου απόκρισης. Οι Blind τεχνικές είναι συνήθως δυσκολότερο να διερευνηθούν, επειδή δεν απαιτούν την άμεση εμφάνιση SQL σφαλμάτων ή δεδομένων στην απόκριση HTTP (Al-Olaqi et al., 2025; Lee, 2019).

Η διερεύνηση SQL injection σε αρχεία PCAP δεν επιτρέπει πάντοτε την επιβεβαίωση της πραγματικής εκτέλεσης ενός ερωτήματος SQL ή της επιτυχούς εξαγωγής δεδομένων. Επιτρέπει, ωστόσο, την αναζήτηση ενδείξεων σε στοιχεία επιπέδου εφαρμογής, όπως το URI, το query string, οι τιμές παραμέτρων, τα request bodies και τα cookies. Ενδεικτικά, οι Boolean-based τεχνικές μπορεί να εμφανίζουν λογικές εκφράσεις όπως OR NOT, AND 1=1 ή OR 1=1, οι Error-based τεχνικές συναρτήσεις όπως EXTRACTVALUE, UPDATEXML ή ELT, οι Time-based τεχνικές συναρτήσεις καθυστέρησης όπως SLEEP, BENCHMARK ή WAITFOR DELAY, ενώ οι UNION-based τεχνικές εκφράσεις όπως UNION SELECT ή UNION ALL SELECT. Η παρουσία των μοτίβων αυτών αποτελεί τεχνική ένδειξη πιθανής SQLi δραστηριότητας, η οποία απαιτεί να συσχετίζεται με το υπόλοιπο πλαίσιο της επικοινωνίας και όχι να αντιμετωπίζεται ως αυτοτελής απόδειξη επίθεσης.

Στο πλαίσιο της παρούσας εργασίας, η αξιολόγηση εστιάζει στις Boolean-based blind, Error-based, Time-based blind και UNION-based SQLi τεχνικές, καθώς και σε σενάρια που συνδυάζουν περισσότερες από μία από αυτές. Οι τεχνικές αυτές επιλέχθηκαν επειδή παράγουν αναγνωρίσιμα ίχνη στις τιμές των παραμέτρων HTTP και μπορούν να διερευνηθούν μέσα από τα στοιχεία που εξάγονται από τα αρχεία PCAP. Η επιλογή αυτή δεν σημαίνει ότι το σύστημα μπορεί να επιβεβαιώσει πλήρως την επιτυχία μιας επίθεσης ή να καλύψει όλες τις μορφές SQLi. Αντίθετα, το σύστημα λειτουργεί ως μηχανισμός εντοπισμού και ερμηνείας πιθανών ενδείξεων, υποστηρίζοντας τον αναλυτή στην περαιτέρω διερεύνηση του περιστατικού.

Η συστηματική αξιοποίηση των παραπάνω ιχνών απαιτεί κατάλληλη ανάκτηση και συσχέτιση δεδομένων από διάφορες πηγές. Στην επόμενη ενότητα παρουσιάζονται οι βασικές ενδείξεις SQL injection, οι διαθέσιμες προσεγγίσεις ανίχνευσης και η σημασία της ανάλυσης τεχνικών artifacts για τη διερεύνηση πιθανών περιστατικών.

2.4 Ενδείξεις, Διερεύνηση και Μέτρα Αντιμετώπισης

Η ανίχνευση πιθανών επιθέσεων SQL injection μπορεί να βασιστεί σε στατικές, δυναμικές, συνδυαστικές και βασισμένες στη μηχανική μάθηση προσεγγίσεις. Οι στατικές τεχνικές αξιοποιούν συνήθως προκαθορισμένες υπογραφές, κανόνες, regular expressions ή μοτίβα γνωστών payloads, ενώ οι δυναμικές τεχνικές εξετάζουν τη συμπεριφορά της εφαρμογής υπό

διαφορετικές τιμές εισόδου. Παράλληλα, προσεγγίσεις μηχανικής και βαθιάς μάθησης επιδιώκουν την αναγνώριση κακόβουλων μοτίβων μέσα από χαρακτηριστικά των αιτημάτων ή της δικτυακής κίνησης. Οι παραπάνω κατηγορίες δεν είναι αμοιβαία αποκλειόμενες και συχνά συνδυάζονται, ώστε να ενισχύεται η αξιοπιστία του εντοπισμού (Nasereddin et al., 2021; Paul et al., 2024).

Για τη διερεύνηση ενός πιθανού περιστατικού SQLi, ιδιαίτερη σημασία έχουν τα διαθέσιμα τεχνικά artifacts. Ενδείξεις μπορεί να προκύψουν από αρχεία καταγραφών των εξυπηρετητών, αιτήματα HTTP, παραμέτρους URI, query strings, request bodies, cookies, χρονικές σημάνσεις και στοιχεία δικτυακής επικοινωνίας. Πρόσθετη πληροφορία μπορεί να προέρχεται από αρχεία καταγραφών των βάσεων δεδομένων, όπως καταγραφές συναλλαγών και πλάνων εκτέλεσης, καθώς και χρονικών σημάνσεων αντικειμένων τους. Η συνδυασμένη εξέταση αυτών των πηγών βοηθά στη διάκριση μεταξύ μιας απλής ύποπτης εισόδου και μιας τεκμηριωμένης προσπάθειας ή επιτυχούς εκμετάλλευσης μίας ευπάθειας του συστήματος (Clarke, 2012).

Στο επίπεδο της επικοινωνίας HTTP, ενδείξεις πιθανής SQLi περιλαμβάνουν τιμές παραμέτρων που περιέχουν λέξεις-κλειδιά SQL, λογικές εκφράσεις, συναρτήσεις βάσης δεδομένων, σχόλια SQL ή συμβολοσειρές σχετικές με το σχήμα της βάσης. Ωστόσο, η παρουσία ενός μεμονωμένου ειδικού χαρακτήρα ή ενός όρου SQL δεν αποτελεί από μόνη της επαρκή απόδειξη κακόβουλης δραστηριότητας. Για παράδειγμα, μια απόστροφος μπορεί να εμφανίζεται σε ένα νόμιμο όνομα χρήστη, ενώ ένας όρος όπως «UNION» ενδέχεται να αποτελεί μέρος μη κακόβουλου κειμένου. Η αξιολόγηση πρέπει επομένως να λαμβάνει υπόψη το πλήρες πλαίσιο του αιτήματος, τη θέση της τιμής, τη συχνότητα και την αλληλουχία παρόμοιων αιτημάτων, καθώς και τη συνύπαρξη πολλαπλών ενδείξεων SQLi.

Η επιβεβαίωση της επιτυχίας μιας SQL injection απαιτεί συνήθως επιπλέον στοιχεία πέρα από την παρατήρηση ύποπτου payload. Τέτοια στοιχεία μπορεί να είναι η εμφάνιση σφαλμάτων SQL, η μη εξουσιοδοτημένη ανάκτηση δεδομένων, η εκτέλεση εντολών μεταβολής δεδομένων ή η δημιουργία νέων αντικειμένων στη βάση. Επομένως, όταν η διαθέσιμη πληροφορία προέρχεται αποκλειστικά από PCAP ή κίνηση HTTP, το συμπέρασμα πρέπει να διατυπώνεται ως ένδειξη ή πιθανή απόπειρα SQLi και όχι ως πλήρης επιβεβαίωση επιτυχημένης εκμετάλλευσης, εκτός εάν υπάρχουν πρόσθετα τεκμήρια από την εφαρμογή ή τη βάση δεδομένων (Clarke, 2012).

Σε επίπεδο πρόληψης, η βασική πρακτική είναι η χρήση παραμετροποιημένων ερωτημάτων (*parameterized queries*) ή prepared statements από τις εφαρμογές διαδικτύου, ώστε τα δεδομένα εισόδου να μη μπορούν να ερμηνευθούν ως τμήμα του SQL κώδικα. Συμπληρωματικά, απαιτούνται έλεγχος και κανονικοποίηση εισόδου, περιορισμός δικαιωμάτων των λογαριασμών σύνδεσης με τη βάση δεδομένων, ασφαλής χειρισμός σφαλμάτων και κατάλληλη καταγραφή συμβάντων. Τα μέτρα αυτά μειώνουν την πιθανότητα εκμετάλλευσης, χωρίς όμως να αναιρούν την ανάγκη παρακολούθησης και διερεύνησης ύποπτης δραστηριότητας (Abdullayev & Chauhan, 2023; *SQL Injection Prevention - OWASP Cheat Sheet Series*, n.d.).

Στο προτεινόμενο σύστημα της παρούσας εργασίας, η διερεύνηση επικεντρώνεται σε στοιχεία HTTP που εξάγονται από αρχεία PCAP. Τα στοιχεία αυτά συνδέονται μεταξύ τους και αναλύονται ως τεχνικά τεκμήρια πιθανής δραστηριότητας SQLi. Η προσέγγιση αυτή δεν επιχειρεί να υποκαταστήσει τους μηχανισμούς πρόληψης ή τα συστήματα ανίχνευσης εισβολών, αλλά να υποστηρίξει την αναλυτική διερεύνηση και την ερμηνεία ύποπτων συμβάντων.

2.5 Συμπεράσματα

Η SQL injection παραμένει σημαντική απειλή για τις εφαρμογές διαδικτύου, επειδή μπορεί να επηρεάσει άμεσα την εμπιστευτικότητα, την ακεραιότητα και τη διαθεσιμότητα των δεδομένων που διαχειρίζονται. Η συγκεκριμένη κατηγορία επιθέσεων αν και χαμηλής συχνότητας, μπορεί να είναι ιδιαίτερα επιβλαβής για μία επιχείρηση ή ένα οργανισμό, το οποίο αποτελεί και ένα από του λόγους που πέρα από τη λίστα του OWASP Top 10, βρίσκεται επίσης και στη δεύτερη θέση των 25 πιο επικίνδυνων αδυναμιών της λίστας του MITRE (*CWE - CWE Top 25 Most Dangerous Software Weaknesses*, n.d.). Χαρακτηριστικό παράδειγμα τέτοιας επίθεσης, είναι η αυτή του 2024 στα συστήματα αυθεντικοποίησης εισόδου που χρησιμοποιεί η TSA (Transport Security Administration) για να εξακριβώσει τα πληρώματα καμπίνας στα σημεία ελέγχου αεροδρομίων. (*Researchers Say a Bug Let Them Add Fake Pilots to Rosters Used for TSA Checks | The Verge*, n.d.) κ.α.. Όπως γίνεται αντιληπτό η ανάπτυξη

αποτελεσματικών συστημάτων άμυνας εναντίον των επιθέσεων SQLi είναι ιδιαίτερα υψηλής σημασίας.

Η πολυμορφία των τεχνικών SQLi και η δυνατότητα χρήσης διαφορετικών payloads καθιστούν ανεπαρκή την αξιολόγηση ενός περιστατικού αποκλειστικά με βάση την εμφάνιση μεμονωμένων λέξεων-κλειδιών ή ειδικών χαρακτήρων. Αντίθετα, απαιτείται συνδυασμός τεχνικών ενδείξεων, συσχέτιση δεδομένων επικοινωνίας και αξιοποίηση εξειδικευμένης γνώσης για τις τεχνικές επίθεσης (Clarke, 2012; Nasereddin et al., 2021).

Η ανάγκη αυτή συνδέεται άμεσα με το πρόβλημα που αντιμετωπίζει η παρούσα εργασία. Τα αρχεία PCAP μπορούν να περιέχουν χρήσιμα στοιχεία επιπέδου επικοινωνίας εφαρμογής, όμως η ερμηνεία τους απαιτεί γνώση των τεχνικών SQLi και του πλαισίου στο οποίο εμφανίζονται. Επιπλέον, η παρουσία ενός ύποπτου μοτίβου δεν ισοδυναμεί πάντοτε με επιβεβαιωμένη επίθεση, γεγονός που καθιστά αναγκαία την τεκμηριωμένη και προσεκτική παραγωγή συμπερασμάτων.

Για τον σκοπό αυτό, στο επόμενο κεφάλαιο παρουσιάζονται οι τεχνολογίες μεγάλων γλωσσικών μοντέλων, Retrieval-Augmented Generation και Agentic RAG. Οι προσεγγίσεις αυτές επιτρέπουν τον συνδυασμό των παρατηρούμενων HTTP artifacts με εξωτερική τεχνική γνώση, υποστηρίζοντας την ανάκτηση, τη συσχέτιση και την ερμηνεία πιθανών ενδείξεων SQL injection.

3. AI και Ανάκτηση Γνώσης στην Κυβερνοασφάλεια

Η αυξανόμενη πολυπλοκότητα των σύγχρονων πληροφοριακών υποδομών και ο μεγάλος όγκος δεδομένων ασφαλείας δημιουργούν σημαντικές απαιτήσεις για αποτελεσματική συλλογή, συσχέτιση και ερμηνεία τεχνικών τεκμηρίων. Σε περιβάλλοντα κυβερνοασφάλειας, οι αναλυτές καλούνται συχνά να εξετάσουν δεδομένα από διαφορετικές πηγές, όπως αρχεία καταγραφής, δικτυακή κίνηση, αιτήματα HTTP, ειδοποιήσεις ασφαλείας και πληροφορίες σχετικά με γνωστές τεχνικές επίθεσης. Η διαδικασία αυτή είναι ιδιαίτερα απαιτητική, καθώς η αξιολόγηση ενός συμβάντος προϋποθέτει όχι μόνο την αναγνώριση ύποπτων μοτίβων, αλλά και την ερμηνεία τους στο κατάλληλο τεχνικό πλαίσιο.

Η τεχνητή νοημοσύνη και ειδικότερα τα Μεγάλα Γλωσσικά Μοντέλα μπορούν να υποστηρίξουν τέτοιες διαδικασίες μέσω της επεξεργασίας φυσικής γλώσσας, της σύνθεσης πληροφορίας και της παραγωγής επεξηγηματικών απαντήσεων. Η αξιοποίησή τους στην κυβερνοασφάλεια μπορεί να ενισχύσει την ανάλυση δεδομένων, να περιορίσει μέρος του γνωστικού φόρτου των αναλυτών και να υποστηρίξει την ταχύτερη διερεύνηση ενδείξεων πιθανής κακόβουλης δραστηριότητας (Kaur et al., 2023). Ωστόσο, η χρήση LLMs σε αυτό το πεδίο απαιτεί ιδιαίτερη προσοχή, επειδή τα παραγόμενα συμπεράσματα πρέπει να στηρίζονται σε πραγματικά και ελέγξιμα τεχνικά τεκμήρια.

Συνεπώς, οι τεχνικές Retrieval Augmented Generation επιτρέπουν τη σύνδεση των γλωσσικών μοντέλων με εξωτερικές πηγές γνώσης, όπως έγγραφα, βάσεις δεδομένων και γράφους γνώσης. Η προσέγγιση αυτή περιορίζει την εξάρτηση από τη γνώση που είναι ενσωματωμένη στο ίδιο το μοντέλο και διευκολύνει την παραγωγή απαντήσεων που βασίζονται σε ανακτημένα στοιχεία. Επιπλέον, η ενσωμάτωση agents επιτρέπει τη δυναμική επιλογή πηγών γνώσης, την οργάνωση σύνθετων ροών ανάκτησης και την επαλήθευση της τελικής απάντησης. Οι έννοιες αυτές αποτελούν το θεωρητικό υπόβαθρο της προτεινόμενης αρχιτεκτονικής, η οποία παρουσιάζεται αναλυτικά στο επόμενο κεφάλαιο.

3.1 Μεγάλα Γλωσσικά Μοντέλα και Περιορισμοί Αξιοπιστίας

Τα Μεγάλα Γλωσσικά Μοντέλα (*Large Language Models — LLMs*) αποτελούν μοντέλα τεχνητής νοημοσύνης που έχουν εκπαιδευτεί σε μεγάλης κλίμακας συλλογές κειμένου και είναι ικανά να κατανοούν, να παράγουν και να μετασχηματίζουν φυσική γλώσσα. Η σύγχρονη ανάπτυξή τους βασίζεται κυρίως στην αρχιτεκτονική *Transformer* και στον μηχανισμό *self-attention*, ο οποίος επιτρέπει στο μοντέλο να αξιοποιεί τις σχέσεις μεταξύ διαφορετικών τμημάτων μιας ακολουθίας κειμένου κατά την επεξεργασία της (Vaswani et al., 2017). Η αύξηση του μεγέθους των μοντέλων, των συνόλων εκπαίδευσης και της υπολογιστικής ισχύος έχει επιτρέψει την αξιοποίησή τους σε σύνθετες εργασίες φυσικής γλώσσας, όπως η περίληψη, η απάντηση ερωτήσεων, η εξαγωγή πληροφορίας και η υποστήριξη διαδικασιών λήψης αποφάσεων (C. Zhao et al., 2024).

Στον τομέα της κυβερνοασφάλειας, τα LLMs μπορούν να υποστηρίξουν την ανάλυση τεχνικών κειμένων, την ερμηνεία αρχείων καταγραφής, τη σύνθεση πληροφοριών από ετερογενείς πηγές και τη δημιουργία κατανοητών αναφορών για ύποπτα συμβάντα. Η δυνατότητα αυτή είναι ιδιαίτερα χρήσιμη όταν ένας αναλυτής χρειάζεται να συνδυάσει στοιχεία από διαφορετικές πηγές και να αποδώσει νόημα σε μεγάλες ποσότητες τεχνικής πληροφορίας. Η αξιοποίησή τους όμως, δεν πρέπει να αντιμετωπίζεται ως υποκατάσταση της ανθρώπινης κρίσης, αλλά ως μηχανισμός υποστήριξης κατά τη διερεύνηση και τεκμηρίωση περιστατικών ασφαλείας (Kaur et al., 2023; Nourmohammadzadeh Motlagh et al., 2024).

Παρά τη σημαντική τους εξέλιξη όμως, τα LLMs παρουσιάζουν και σημαντικούς περιορισμούς που επηρεάζουν την αξιοπιστία τους. Σημαντικές προκλήσεις που αντιμετωπίζουν τα LLMs και σχετίζονται με τις τεχνολογίες στις οποίες βασίζονται σύμφωνα με τους (Fui-Hoon Nah et al., 2023), είναι τα παρακάτω:

- Ποιότητα των δεδομένων εκπαίδευσης. Η ποιότητα ενός μοντέλου Generative AI εξαρτάται σε μεγάλο βαθμό από την ποιότητα των δεδομένων στα οποία έχει εκπαιδευτεί. Δεδομένα τα οποία είναι ατελή ή περιέχουν λανθασμένες πληροφορίες ή ακόμη και προκατάληψη, μπορεί να εμφανιστούν στις απαντήσεις του μοντέλου.

- Εξηγησιμότητα (explainability). Μια ακόμη ανησυχία σχετικά με τα αποτελέσματα στις απαντήσεις των LLMs είναι ή έλλειψη διαφάνειας της συλλογιστικής τους

πορείας, που ακολούθησαν προκειμένου να καταλήξουν στο εξαγόμενο αποτέλεσμα. Αυτό το γεγονός επηρεάζει την εμπιστοσύνη του συστήματος από το χρήστη.

- Αυθεντικότητα (Authenticity). Με τη συνεχή εξέλιξη και βελτίωση των LLMs, παραγόμενα αποτελέσματα όπως φωτογραφία, video, ήχος είναι όλο και πιο δύσκολο να διακρίνεις το αν προήλθαν από κάποιο LLM ή είναι προϊόν ανθρώπινης δημιουργίας. Το γεγονός αυτό έχει εγείρει έντονους προβληματισμούς στην καλλιτεχνική κοινότητα. Επίσης μεγάλη ανησυχία δημιουργεί το ότι αντίστοιχα αποτελέσματα μπορούν να χρησιμοποιηθούν για κακόβουλη χρήση, όπως ψευδείς πληροφορίες σε κοινωνικά δίκτυα και στο διαδίκτυο.

- Prompt Engineering. Με τη ευρεία διάδοση εφαρμογών όπως το ChatGPT, η αποτελεσματική διάδραση των χρηστών με τα LLMs έχει αποκτήσει μεγάλη σημασία. Συνήθως αυτό επιτυγχάνεται μέσω prompts προς το LLM και η μη σωστή κατασκευή τους μπορεί να οδηγήσει σε εσφαλμένα αποτελέσματα και παρεξηγήσεις. Συνεπώς εδώ είναι που εισέρχεται ο όρος prompt engineering, ο οποίος αναφέρεται στη σωστή κατασκευή των prompts από το χρήστη, προκειμένου να έχει τα επιθυμητά αποτελέσματα.

- Παραισθήσεις (Hallucinations). Ένα από τα πιο σημαντικά προβλήματα των LLMs, είναι τα λεγόμενα «hallucinations». Με τον όρο αυτό εννοούμε την εξαγωγή αποτελεσμάτων από το μοντέλο τα οποία δεν βασίζονται σε πραγματικές πηγές και δεν είναι υπαρκτά, αλλά έχουν κατασκευαστεί από το μοντέλο. Τέτοια παραδείγματα, μπορεί να είναι γεγονότα που δεν έχουν συμβεί, πρόσωπα που δεν έχουν υπάρξει στην πραγματική ζωή κ.α.. Τις παραισθήσεις βασιζόμενοι στη σχέση μεταξύ της εισόδου που δίνεται στο LLM και της παραγόμενης εξόδου, μπορούμε να τις διακρίνουμε σε εσωτερικές (intrinsic hallucinations), όπου το μοντέλο αντιφάσκει με το παρεχόμενο prompt, καθώς και τις εξωτερικές (extrinsic hallucinations), όπου το μοντέλο κατασκευάζει πληροφορίες που δεν υπάρχουν στα δεδομένα του (W. X. Zhao et al., 2026). Πιθανά αίτια των παραισθήσεων μπορεί να είναι τα δεδομένα που χρησιμοποιήθηκαν για την εκπαίδευση του μοντέλου, οι μέθοδοι εκπαίδευσης και οι στρατηγικές που ακολουθεί το μοντέλο για την παραγωγή των απαντήσεων. Μία από τις πιο σημαντικές τεχνικές αντιμετώπισης του προβλήματος των παραισθήσεων (hallucinations), είναι η τεχνική Retrieval Augmented Generation (RAG).

Όπως γίνεται αντιληπτό, αυτοί οι περιορισμοί των γλωσσικών μοντέλων μπορούν να δημιουργήσουν σημαντικά προβλήματα, ειδικά σε περιπτώσεις όπου τέτοια μοντέλα αξιοποιούνται για την υποστήριξη πληροφοριακών συστημάτων που σχετίζονται με την

ασφάλεια οργανισμών, επιχειρήσεων και διαφόρων υποδομών. Σε περιβάλλοντα κυβερνοασφάλειας, ένας μη τεκμηριωμένος ισχυρισμός μπορεί να οδηγήσει σε λανθασμένη αξιολόγηση ενός συμβάντος, σε άσκοπη διερεύνηση ή σε εσφαλμένη αίσθηση ασφάλειας. Τα τελευταία χρόνια, διάφορες τεχνικές και εργαλεία έχουν χρησιμοποιηθεί για να αντιμετωπίσουν ορισμένα από αυτά τα προβλήματα με το RAG να είναι μία από τις πλέον διαδεδομένες. Η συγκεκριμένη τεχνική, η οποία παρουσιάζεται στην επόμενη ενότητα, αποτελεί ένα από τα σημαντικότερα συστατικά του προτεινόμενου συστήματος.

3.2 Retrieval Augmented Generation (RAG)

Τα μεγάλα γλωσσικά μοντέλα ενώ παρουσιάζουν εντυπωσιακές δυνατότητες σε διάφορες εργασίες που σχετίζονται με την επεξεργασία της φυσικής γλώσσας, παρουσιάζουν ένα θεμελιώδη περιορισμό λόγω της εξάρτησης τους από τα δεδομένα εκπαίδευσης τους, τα οποία είναι στατικά και σε ορισμένες περιπτώσεις είτε έχουν μεταβληθεί ή δεν ισχύουν πλέον. Αποτέλεσμα αυτού του περιορισμού είναι οι απαντήσεις των μοντέλων να μην είναι πάντα έγκυρες ή ακόμη και να είναι αναληθείς (Sharma, 2025).

Λύση για την αντιμετώπιση των παραπάνω προβλημάτων, υπόσχεται να δώσει η τεχνική RAG. Η εν λόγω τεχνική επαυξάνει τα LLMs ενσωματώνοντας εξωτερική γνώση στη διαδικασία παραγωγής κατά το χρόνο της εξαγωγής συμπερασμάτων (Singh et al., 2025). Η εν λόγω εξωτερική γνώση μπορεί να προέρχεται από διάφορες πηγές, όπως βάσεις δεδομένων, APIs ή και πηγές διαδικτύου και με αυτό τον τρόπο να παρέχονται στο «πλαίσιο παραθύρου» (context window) του μοντέλου επιπλέον δεδομένα, τα οποία είναι ενήμερα, σχετιζόμενα με το επιθυμητό αποτέλεσμα ή και επικεντρωμένα σε ένα συγκεκριμένο τομέα (πχ. κυβερνοασφάλεια). Έτσι η τεχνική RAG έχει επιδείξει σημαντική επιτυχία στην αντιμετώπιση των προαναφερθέντων προκλήσεων των LLMs, χωρίς να απαιτείται επανεκπαίδευση τους (Peng et al., 2026).

Η αρχιτεκτονική των RAG συστημάτων σύμφωνα με τους (Singh et al., 2025) αποτελείται από τα εξής τρία βασικά συστήματα:

- Ανάκτηση (Retrieval): είναι υπεύθυνο για την ανάκτηση δεδομένων από εξωτερικές πηγές, όπως βάσεις γνώσης, APIs ή διανυσματικές βάσεις δεδομένων. Εξελιγμένα

συστήματα ανάκτησης προκειμένου να βελτιώσουν την ακρίβεια τους, καθώς και την σημασιολογική συνάφεια των δεδομένων ανάκτησης, αξιοποιούν δυνατότητες όπως την αναζήτηση πυκνών διανυσμάτων και μοντέλα βασισμένα σε Transformers.

- Επαύξηση (Augmentation): σχετίζεται με την επεξεργασία των ανακτημένων δεδομένων, έτσι ώστε να εξαχθούν οι πιο σχετικές πληροφορίες που ανταποκρίνονται σημασιολογικά στο αντίστοιχο ερώτημα του χρήστη.

- Παραγωγή (Generation): σε αυτό το στάδιο γίνεται ο συνδυασμός του LLM με την ανακτημένη πληροφορία από τις προηγούμενες φάσεις του Retrieval και του Augmentation, με σκοπό να αξιοποιηθεί ο συνδυασμός της ανακτημένης πληροφορίας με τη γνώση του μοντέλου, που προέρχεται από την εκπαίδευση του, ώστε να παραχθεί μια τελική απάντηση η οποία θα είναι απαλλαγμένη από «παραισθήσεις» (hallucinations) και επιπλέον θα είναι σημασιολογικά σχετική με το ερώτημα του χρήστη.

3.2.1 Η Εξελικτική Διαδρομή: Naive, Advanced και Modular RAG

Σύμφωνα με τους (Y. Gao et al., 2023), η εξέλιξη των αρχιτεκτονικών RAG μπορεί να χωριστεί σε τρία στάδια, το Naive RAG, το Advanced RAG και το Modular RAG. Η διάκριση αυτή αποτυπώνει τη μετάβαση από απλές, γραμμικές ροές ανάκτησης και παραγωγής σε περισσότερο προσαρμοστικές αρχιτεκτονικές, στις οποίες τα επιμέρους στοιχεία της διαδικασίας μπορούν να βελτιστοποιούνται και να συνδυάζονται, προκειμένου να αντιμετωπιστούν περιορισμοί και θέματα επεκτασιμότητας των πρώτων προσεγγίσεων. Προκειμένου να γίνει καλύτερα κατανοητή η αρχιτεκτονική του Modular RAG, πάνω στην οποία βασίζεται το προτεινόμενο σύστημα της εργασίας, ακολουθεί μία αναφορά και στις υπόλοιπες:

- Naïve RAG:

Αποτελεί την απλούστερη μορφή RAG και ακολουθεί συνήθως μια γραμμική ροή τριών σταδίων, την ευρετηρίαση (*indexing*), την ανάκτηση (*retrieval*) και την παραγωγή (*generation*). Κατά την ευρετηρίαση, τα αρχικά δεδομένα μετατρέπονται σε ενιαία μορφή, τεμαχίζονται σε μικρότερα αποσπάσματα και αποθηκεύονται σε κατάλληλη υποδομή γνώσης. Στη συνέχεια, ένα ερώτημα χρήστη χρησιμοποιείται για την ανάκτηση των σχετικότερων αποσπασμάτων, τα οποία ενσωματώνονται στο αρχικό ερώτημα του χρήστη και στέλνονται

στο γλωσσικό μοντέλο για την παραγωγή της τελικής απάντησης. Η τεχνική του Naive RAG ήταν η πρώτη προσέγγιση για την αντιμετώπιση του προβλήματος των «παραισθήσεων» (hallucinations), χαρακτηρίζεται από την απλότητα και την ευκολία της υλοποίησης (Singh et al., 2025), ωστόσο παρουσιάζει αρκετούς περιορισμούς, όπως την αποτυχία να ανακτήσει την σωστή πληροφορία σύμφωνα με τη σημασία του αρχικού ερωτήματος, προβλήματα επεκτασιμότητας (scalability) καθώς και με την παραγωγή της τελικής απάντησης, η οποία μπορεί να είναι πολύ γενικευμένη ή τελείως άσχετη σε σχέση με το αρχικό ερώτημα.

- Advanced RAG:

Η τεχνική του Advanced RAG εισήγαγε αλλαγές προκείμενου να ξεπεραστούν τα προβλήματα και οι περιορισμοί του Naive RAG. Η συγκεκριμένη τεχνική επεκτείνει τη βασική ροή με μηχανισμούς βελτιστοποίησης πριν και μετά την ανάκτηση. Στο στάδιο προ-ανάκτησης (*pre-retrieval*) μπορούν να εφαρμοστούν τεχνικές βελτίωσης της ευρετηρίασης, επαναδιατύπωσης ή επέκτασης του ερωτήματος, ενώ στο στάδιο μετά-ανάκτησης (*post-retrieval*) αξιοποιούνται τεχνικές όπως η ανακατάταξη των αποτελεσμάτων (*re-ranking*) και η συμπίεση του πλαισίου (*context compression*). Οι μηχανισμοί αυτοί επιδιώκουν να αυξήσουν τη συνάφεια των τεκμηρίων που τελικά προωθούνται στο LLM και να περιορίσουν τον πλεονασμό ή την εισαγωγή άσχετης πληροφορίας στο διαθέσιμο πλαίσιο (Y. Gao et al., 2023).

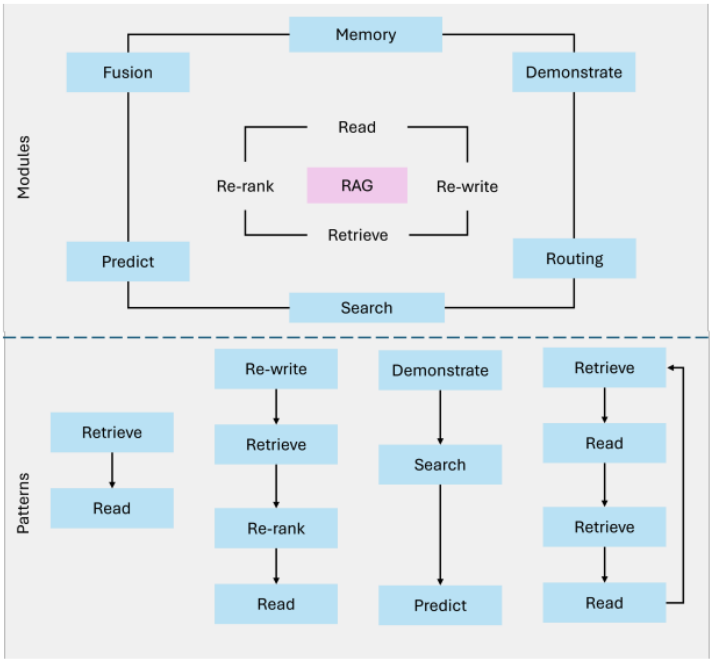
- Modular RAG:

Η συγκεκριμένη μεθοδολογία RAG είναι η τελευταία εξέλιξη όσο αφορά την αρχιτεκτονική των συστημάτων RAG, βασίζεται πάνω στις προηγούμενες και δίνει ιδιαίτερη έμφαση στην προσαρμοστικότητα και στην παραμετροποίηση (Εικόνα 2), γεγονός που την καθιστά ιδανική για σύνθετες και πολυδιάστατες εργασίες προσφέροντας επεκτασιμότητα και ακρίβεια. Οι πλέον σημαντικές καινοτομίες της συγκεκριμένης μεθοδολογίας, σύμφωνα με τους (Singh et al., 2025) είναι οι εξής:

1. **Στρατηγικές Υβριδικής Ανάκτησης (Hybrid Retrieval Strategies)** οι οποίες συνδυάζουν μεθόδους σποραδικής ανάκτησης (sparse encoder-BM25) με πυκνής ανάκτησης (π.χ. DPR – Dense Passage Retrieval), προκειμένου να μεγιστοποιήσουν την ακρίβεια για διάφορους τύπους ερωτημάτων.

2. **Ενσωμάτωση Εργαλείων (Tool Integration).** Ενσωμάτωση εξωτερικών APIs, βάσεων δεδομένων ή υπολογιστικών εργαλείων τα οποία διεκπεραιώνουν εξειδικευμένες εργασίες.

3. **Συνθέσιμες Ροές (Composable Pipelines).** Στο Modular RAG δίνεται η δυνατότητα αντικατάστασης, επαύξησης ή ανεξάρτητης επαναπαραμετροποίησης διαφόρων υποσυστημάτων (πχ retrievers, generators), προσφέροντας υψηλή προσαρμοστικότητα ανάλογα την περίπτωση χρήσης.



Εικόνα 2 Ανάλυση Modular RAG

Η προτεινόμενη λύση της παρούσας εργασίας ακολουθεί τη λογική του Modular RAG. Η αρχιτεκτονική δεν περιορίζεται σε μία στατική διαδικασία ανάκτησης από μία μόνο πηγή δεδομένων, αλλά αποτελείται από διακριτά και συνεργαζόμενα υποσυστήματα για την εισαγωγή δεδομένων, την αναπαράσταση γνώσης, τη διανυσματική και γραφική ανάκτηση, τη σύνθεση απαντήσεων και την επαλήθευση των παραγόμενων αποτελεσμάτων. Η διακριτή οργάνωση των λειτουργιών αυτών επιτρέπει την αξιοποίηση διαφορετικών πηγών γνώσης και στρατηγικών ανάκτησης, καθώς και την προσαρμογή της ροής επεξεργασίας στις απαιτήσεις κάθε ερωτήματος.

3.2.2 Στρατηγικές Ανάκτησης (Retrieval Strategies)

Οι στρατηγικές ανάκτησης σε μία ροή RAG είναι υψηλής σημασίας, καθώς καθορίζει ποια τεκμήρια θα επιλεγούν από τις διαθέσιμες πηγές γνώσης και θα αξιοποιηθούν κατά την παραγωγή της τελικής απάντησης. Η επιλογή της κατάλληλης στρατηγικής εξαρτάται από τη φύση του ερωτήματος, τη δομή των διαθέσιμων δεδομένων και τον βαθμό πολυπλοκότητας της απαιτούμενης διερεύνησης. Ενώ στις απλούστερες υλοποιήσεις RAG η ανάκτηση πραγματοποιείται σε ένα μόνο βήμα, νεότερες προσεγγίσεις υιοθετούν επαναληπτικές, υπό συνθήκη ή προσαρμοστικές ροές (Εικόνα 3), ώστε η διαδικασία να ανταποκρίνεται αποτελεσματικότερα στις απαιτήσεις διαφορετικών ερωτημάτων. Σύμφωνα με τους (Huang & Huang, 2026) υπάρχουν πέντε κύριες στρατηγικές ανάκτησης στα συστήματα RAG:

- Βασική Στρατηγική Ανάκτησης (Basic Retrieval Strategy)

Στη βασική στρατηγική η ανάκτηση ακολουθεί μία γραμμική/ακολουθιακή ροή εργασιών μεταξύ των φάσεων της προ-ανάκτησης (pre-retrieval), της ανάκτησης (retrieval) και της μετά-ανάκτησης (post-retrieval), όπου η ανακτημένη πληροφορία χρησιμοποιείται απευθείας στη διαδικασία της παραγωγής (generation).

- Στρατηγική Επαναληπτικής Ανάκτησης (Iterative Retrieval Strategy)

Η συγκεκριμένη στρατηγική χρησιμοποιείται σε πιο περίπλοκα σενάρια και η ανάκτηση της πληροφορίας πραγματοποιείται σε πολλαπλά βήματα, καθένα από αυτά ενημερώνεται από τα αποτελέσματα του προηγούμενου βήματος. Η επαναληπτική στρατηγική έχει υλοποιηθεί με διάφορους τρόπους, όπως συνδυασμό ανάκτησης με αλυσιδωτό συλλογισμό (chain-of-thought reasoning), χρήσιμο για περιπτώσεις επίλυσης προβλημάτων που απαιτούν χρήση πολλαπλών βημάτων, καθώς και βελτιστοποίησης του παραγόμενου αποτελέσματος με επαναληπτική παραγωγή ενδιάμεσων αποτελεσμάτων.

- Στρατηγική Αναδρομικής Ανάκτησης (Recursive Retrieval Strategy)

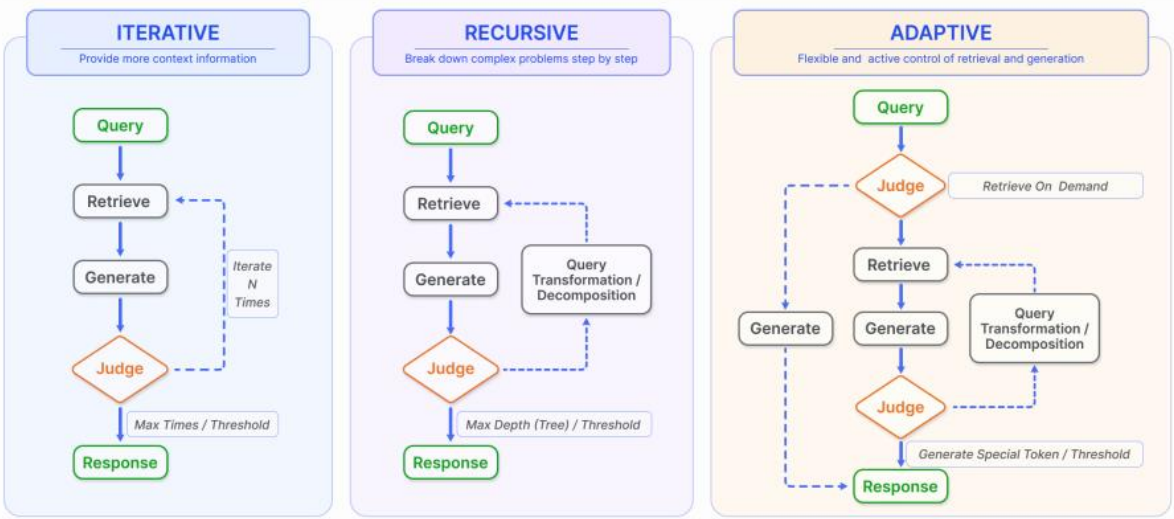
Στην αναδρομική ανάκτηση, η διαδικασία της ανάκτησης μπορεί να καλέσει τον εαυτό της, δημιουργώντας κατ' αυτό τον τρόπο ένα ιεραρχικό δέντρο αναδρομικών ανακτήσεων. Μια τέτοια μέθοδος μπορεί να χειριστεί ιεραρχική και πολυεπίπεδη πληροφορία, ανάγοντας τα σύνθετα ερωτήματα σε απλούστερα υπό-ερωτήματα και μπορεί να φανεί ιδιαίτερη χρήσιμη στη ιεραρχική διερεύνηση πληροφορίας, καθώς και στην κατασκευή βάσης γνώσης.

- Στρατηγική υπό Συνθήκη Ανάκτησης (Conditional Retrieval Strategy)

Στην εν λόγω στρατηγική η ανάκτηση της πληροφορίας διέπεται από συγκεκριμένους κανόνες και καταστάσεις, οι οποίες μπορεί να είναι προκαθορισμένες ή δυναμικά καθορισμένες κατά τη διάρκεια της διαδικασίας ανάκτησης. Με τον τρόπο αυτό διασφαλίζεται ότι η πληροφορία που ανακτάται συμμορφώνεται με συγκεκριμένους περιορισμούς ή κριτήρια, καθιστώντας έτσι αυτή την στρατηγική ιδανική για συστήματα συστάσεων βασισμένα σε κανόνες (rule-based), καθώς και συστήματα ελέγχου συμμόρφωσης (compliance checking).

- Στρατηγική Δυναμικής Ανάκτησης (Adaptive Retrieval Strategy)

Στην δυναμική ανάκτηση, η στρατηγική ανάκτησης προσαρμόζεται δυναμικά αναλόγως τη φύση και τη σημασία του ερωτήματος ή των έως τώρα ανακτημένων δεδομένων. Με αυτό τον τρόπο η ανάκτηση προσαρμόζεται σε πραγματικό χρόνο, προκείμενου να βελτιωθεί η ακρίβεια και η σχετικότητα, κάτι που αποδεικνύεται ιδιαίτερα χρήσιμο σε προσωποποιημένες μηχανές αναζήτησης, σε συστήματα τα οποία εκπαιδεύονται δυναμικά και σε συστήματα αποφάσεων πραγματικού χρόνου (real-time).



Εικόνα 3 Σύγκριση μεταξύ Iterative, Recursive & Adaptive Retrieval (Y. Gao et al., 2023)

Στην υλοποίηση της παρούσας εργασίας, η διαδικασία ανάκτησης δεν ακολουθεί μία αποκλειστικά στατική ροή. Ανάλογα με το περιεχόμενο του ερωτήματος, το σύστημα μπορεί να αξιοποιήσει διανυσματική, γραφική ή υβριδική ανάκτηση, εφαρμόζοντας τη στρατηγική της δυναμικής ανάκτησης. Σε περιπτώσεις όπου η επαλήθευση της απάντησης αναδείξει

ανεπαρκή τεκμηρίωση, είναι δυνατό να ενεργοποιηθεί συμπληρωματικός κύκλος ανάκτησης, εφαρμόζοντας με αυτό τον τρόπο στρατηγικές επαναληπτικής και υπό συνθήκη ανάκτησης. Η επιλογή της κατάλληλης πηγής γνώσης και η διαχείριση των επαναληπτικών βημάτων και γενικά η διαχείριση της συνολικής ροής εργασίας του RAG, πραγματοποιούνται από υποσύστημα πολλαπλών agents, το οποίο παρουσιάζεται σε επόμενες ενότητες.

3.2.3 Διαχείριση και Αναπαράσταση Γνώσης

Η αποτελεσματικότητα ενός συστήματος RAG εξαρτάται όχι μόνο από τη στρατηγική ανάκτησης, αλλά και από τον τρόπο με τον οποίο η διαθέσιμη γνώση οργανώνεται, αποθηκεύεται και καθίσταται προσβάσιμη. Η αναπαράσταση της γνώσης επηρεάζει άμεσα το είδος των ερωτημάτων που μπορεί να υποστηρίξει το σύστημα, την ποιότητα των ανακτημένων τεκμηρίων και τη δυνατότητα συσχέτισης διαφορετικών στοιχείων. Στο πλαίσιο των σύγχρονων αρχιτεκτονικών RAG, ιδιαίτερη σημασία έχουν η διανυσματική αναπαράσταση μη δομημένου περιεχομένου, η γραφική αναπαράσταση οντοτήτων και σχέσεων, καθώς και ο συνδυασμός των δύο προσεγγίσεων.

Η προσέγγιση **VectorRAG** βασίζεται στην αποθήκευση τεκμηρίων ή τμημάτων κειμένου ως διανυσματικές αναπαραστάσεις (*embeddings*) σε διανυσματικές βάσεις δεδομένων. Οι αναπαραστάσεις αυτές επιτρέπουν την ανάκτηση περιεχομένου με βάση τη σημασιολογική εγγύτητα του ερωτήματος προς τα αποθηκευμένα τεκμήρια, αντί αποκλειστικά με βάση την ακριβή αντιστοίχιση λέξεων-κλειδιών. Με τον τρόπο αυτό, ένα VectorRAG σύστημα μπορεί να ανακτήσει σχετική γνώση ακόμη και όταν η διατύπωση του ερωτήματος διαφέρει από εκείνη του αρχικού κειμένου (S. Gao et al., 2026; Huang & Huang, 2026). Η προσέγγιση αυτή είναι ιδιαίτερα κατάλληλη για την αξιοποίηση τεχνικής τεκμηρίωσης, περιγραφών επιθέσεων, παραδειγμάτων payloads και οδηγιών πρόληψης, δηλαδή για περιεχόμενο που διατίθεται κυρίως σε μορφή φυσικής γλώσσας ή μη δομημένων εγγράφων.

Παρότι η διανυσματική ανάκτηση υποστηρίζει αποτελεσματικά τη σημασιολογική αναζήτηση, δεν αποτυπώνει ρητά τις σχέσεις μεταξύ των επιμέρους οντοτήτων. Για παράδειγμα, η σημασιολογική ομοιότητα μεταξύ δύο τμημάτων κειμένου δεν εκφράζει από μόνη της ότι ένα συγκεκριμένο αίτημα HTTP περιέχεται σε ένα αρχείο PCAP, ότι προέρχεται από συγκεκριμένη διεύθυνση IP ή ότι περιλαμβάνει συγκεκριμένη παράμετρο και τιμή. Σε τέτοιες

περιπτώσεις, η διαχείριση της γνώσης μέσω γραφικών δομών προσφέρει συμπληρωματικές δυνατότητες.

Η προσέγγιση **GraphRAG** αξιοποιεί γράφους γνώσης, στους οποίους οι βασικές οντότητες αναπαρίστανται ως κόμβοι και οι σχέσεις τους ως ακμές. Με τον τρόπο αυτό, η πληροφορία οργανώνεται ως δίκτυο συσχετίσεων και μπορεί να διερευνηθεί μέσω πλοήγησης μεταξύ οντοτήτων ή μέσω ανάκτησης μονοπατιών που συνδέουν διαφορετικά στοιχεία. Η γραφική αναπαράσταση είναι ιδιαίτερα χρήσιμη όταν η απάντηση σε ένα ερώτημα δεν βρίσκεται σε ένα μόνο απόσπασμα κειμένου, αλλά προκύπτει από τη σύνθεση σχέσεων μεταξύ πολλών τεχνικών artifacts. Επιπλέον, οι γράφοι γνώσης υποστηρίζουν συλλογισμό πολλαπλών βημάτων, καθώς επιτρέπουν τη διερεύνηση ακολουθιών σχέσεων μεταξύ κόμβων (Peng et al., 2026).

Παρόλα αυτά και η παραπάνω τεχνική, όπως και το VectorRAG παρουσιάζει κάποιες αδυναμίες. Συγκεκριμένα και λόγω του τρόπου κατασκευής ενός γράφου όπου τα δεδομένα απεικονίζονται πλέον σε οντότητες και συσχετίσεις μεταξύ αυτών, ένα σύστημα αρχιτεκτονικής GraphRAG αντιμετωπίζει δυσκολίες σε ερωτήσεις που αφορούν περίληψη ή δεν αναφέρεται ρητά μία οντότητα στο αρχικό ερώτημα του χρήστη (Sarmah et al., 2024).

Όπως είναι πλέον αντιληπτό κάθε μία από τις παραπάνω αρχιτεκτονικές παρουσιάζει τα δικά της προτερήματα, καθώς και αδυναμίες. Συνεπώς, η προσέγγιση **HybridRAG** συνδυάζει διανυσματική και γραφική αναπαράσταση γνώσης, επιδιώκοντας να αξιοποιήσει τα πλεονεκτήματα των δύο μοντέλων. Η διανυσματική υποδομή μπορεί να παρέχει σημασιολογικά σχετική και εξειδικευμένη γνώση από μη δομημένα τεκμήρια, ενώ ο γράφος γνώσης μπορεί να ανακτά συγκεκριμένα, συσχετισμένα και παρατηρούμενα στοιχεία από ένα σύνολο δεδομένων (Sarmah et al., 2024). Ο συνδυασμός αυτός είναι χρήσιμος σε περιπτώσεις όπου η απάντηση απαιτεί αφενός τεχνική ερμηνεία ενός μοτίβου και αφετέρου τεκμηρίωση ότι το αντίστοιχο στοιχείο εμφανίστηκε σε συγκεκριμένη επικοινωνία ή γεγονός .

Η συμπληρωματικότητα των δύο μορφών γνώσης αποτελεί τη βάση της υβριδικής προσέγγισης που υλοποιείται στην παρούσα εργασία. Η υβριδική αναπαράσταση γνώσης αξιοποιείται για τη σύνδεση των παρατηρούμενων HTTP artifacts με εξειδικευμένη γνώση SQL injection. Η γραφική υποδομή αποτυπώνει τις σχέσεις μεταξύ αρχείων PCAP, αιτημάτων HTTP, διευθύνσεων IP, hosts και παραμέτρων, ενώ η διανυσματική υποδομή περιλαμβάνει τεχνική και θεωρητική γνώση για τεχνικές SQLi, payloads και τρόπους αντιμετώπισης. Επομένως, η πρώτη μπορεί να υποστηρίξει ερωτήματα σχετικά με το τι παρατηρήθηκε στα

εισαγμένα δεδομένα, ενώ η δεύτερη μπορεί να υποστηρίξει την ερμηνεία της πιθανής σημασίας των ευρημάτων.

Η συνύπαρξη διαφορετικών μορφών γνώσης δημιουργεί την ανάγκη δυναμικής επιλογής της κατάλληλης πηγής ανάκτησης, καθώς και ελέγχου της ποιότητας των παραγόμενων απαντήσεων. Η ανάγκη αυτή αποτελεί βασικό κίνητρο για την αξιοποίηση Agentic RAG και ροών εργασίας πολλαπλών agents, οι οποίες παρουσιάζονται στην επόμενη ενότητα.

3.3 Agentic RAG και Συστήματα Πολλαπλών Agents

Η αξιοποίηση πολλαπλών πηγών γνώσης σε ένα σύστημα RAG δημιουργεί την ανάγκη για δυναμική επιλογή της κατάλληλης στρατηγικής ανάκτησης, καθώς και για έλεγχο της επάρκειας των τεκμηρίων που χρησιμοποιούνται κατά την παραγωγή μιας απάντησης. Σε απλές αρχιτεκτονικές RAG, η διαδικασία ανάκτησης και παραγωγής ακολουθεί συνήθως μια σταθερή ακολουθία βημάτων. Ωστόσο, σε σύνθετα ερωτήματα μπορεί να απαιτείται επιλογή μεταξύ διαφορετικών εργαλείων, συμπληρωματική ανάκτηση από περισσότερες από μία πηγές ή επαναληπτική αναθεώρηση μιας απάντησης όταν τα διαθέσιμα στοιχεία δεν είναι επαρκή.

Το Agentic RAG επεκτείνει τη συμβατική προσέγγιση RAG, ενσωματώνοντας agents που αναλαμβάνουν διακριτούς ρόλους στη δρομολόγηση ερωτημάτων, στην επιλογή εργαλείων, στην ανάκτηση πληροφοριών, στη σύνθεση απαντήσεων και στην αξιολόγηση των αποτελεσμάτων. Οι agents αυτοί αξιοποιούν γλωσσικά μοντέλα, εξωτερικές πηγές γνώσης και μηχανισμούς διαχείρισης κατάστασης, ώστε η συνολική ροή να προσαρμόζεται στις ανάγκες του εκάστοτε ερωτήματος. Η συνεργασία εξειδικευμένων agents επιτρέπει τον διαχωρισμό των επιμέρους λειτουργιών και υποστηρίζει περισσότερο ελεγχόμενη παραγωγή απαντήσεων σε σύγκριση με μια ενιαία, μη διακριτή ροή επεξεργασίας (Sapkota et al., 2026; Singh et al., 2025).

Η Agentic RAG προσέγγιση, στο προτεινόμενο σύστημα χρησιμοποιείται για τον συνδυασμό της διανυσματικής και της γραφικής υποδομής γνώσης με μια ελεγχόμενη ροή ερωτήματος–ανάκτησης–σύνθεσης–επαλήθευσης. Στη συνέχεια παρουσιάζονται οι βασικές αρχές των agents, της δυναμικής δρομολόγησης και των ροών επαλήθευσης που υποστηρίζουν την προτεινόμενη αρχιτεκτονική.

3.3.1 Agents, Εργαλεία και Δυναμική Δρομολόγηση

Οι agents αποτελούν οντότητες λογισμικού που αξιοποιούν μεγάλα γλωσσικά μοντέλα για την εκτέλεση καθορισμένων εργασιών, με δυνατότητα αξιοποίησης εξωτερικών εργαλείων και πηγών γνώσης. Σε αντίθεση με μια απλή αλληλεπίδραση ερωτήματος–απάντησης, ένας agent μπορεί να ερμηνεύει το αίτημα του χρήστη, να επιλέγει ενέργειες σύμφωνα με τον ρόλο του και να χρησιμοποιεί δεδομένα ή υπηρεσίες εκτός του γλωσσικού μοντέλου. Τα βασικά στοιχεία ενός agent περιλαμβάνουν το γλωσσικό μοντέλο, τον καθορισμένο ρόλο ή στόχο, την πρόσβαση σε εργαλεία και έναν μηχανισμό διαχείρισης της κατάστασης της εκτέλεσης (Sapkota et al., 2026; Singh et al., 2025).

Η πρόσβαση σε εργαλεία επιτρέπει στους agents να υπερβαίνουν τα όρια της γνώσης που είναι ενσωματωμένη στο ίδιο το γλωσσικό μοντέλο. Τα εργαλεία μπορούν να περιλαμβάνουν μηχανισμούς αναζήτησης στο διαδίκτυο, βάσεις δεδομένων και APIs. Μέσω αυτών, ένας agent μπορεί να ανακτά ενημερωμένα ή εξειδικευμένα δεδομένα, να εκτελεί αναζητήσεις σε εξωτερικές πηγές και να αξιοποιεί τα αποτελέσματα ως τεκμήρια για την επόμενη ενέργεια ή την τελική απάντηση. Η δυνατότητα αυτή είναι ιδιαίτερα σημαντική σε Agentic RAG αρχιτεκτονικές, όπου η ποιότητα της απάντησης εξαρτάται από την ορθή επιλογή και χρήση των διαθέσιμων πηγών γνώσης (Singh et al., 2025).

Η δυναμική δρομολόγηση (*routing*) αφορά τη διαδικασία κατά την οποία το σύστημα αξιολογεί το εισερχόμενο ερώτημα και το προωθεί προς την κατάλληλη διαδικασία, agent ή εργαλείο. Η λειτουργία αυτή επιτρέπει τη διαχείριση διαφορετικών τύπων ερωτημάτων με διακριτές στρατηγικές, αντί της εφαρμογής της ίδιας ροής ανάκτησης σε όλες τις περιπτώσεις. Για παράδειγμα, ένα ερώτημα γενικής γνώσης μπορεί να απαιτεί αναζήτηση σε τεχνικά έγγραφα, ενώ ένα ερώτημα που αφορά συγκεκριμένο περιστατικό δικτυακής κίνησης μπορεί να απαιτεί διερεύνηση δομημένων τεχνικών στοιχείων. Η δρομολόγηση βελτιώνει την αποδοτικότητα και τη συνάφεια της επεξεργασίας, επειδή περιορίζει την άσκοπη ενεργοποίηση εργαλείων και επιτρέπει την προσαρμογή της ροής στις απαιτήσεις του ερωτήματος (Singh et al., 2025).

Σε συστήματα πολλαπλών agents, οι παραπάνω λειτουργίες κατανέμονται σε εξειδικευμένους ρόλους. Η προσέγγιση αυτή επιτρέπει τον διαχωρισμό της αξιολόγησης του ερωτήματος, της ανάκτησης τεκμηρίων, της σύνθεσης της απάντησης και του ελέγχου της τεκμηρίωσής της. Αντί να αναλαμβάνει ένα μόνο μοντέλο όλες τις ενέργειες, κάθε agent επικεντρώνεται σε

συγκεκριμένη υπο-εργασία και ανταλλάσσει τα αποτελέσματά του με τα υπόλοιπα στοιχεία της ροής μέσω μιας κοινής κατάστασης εκτέλεσης. Η οργάνωση αυτή υποστηρίζει την αρθρωτότητα, την επεκτασιμότητα και την καλύτερη επιθεώρηση των ενδιάμεσων αποφάσεων του συστήματος (Sapkota et al., 2026).

Στην προτεινόμενη αρχιτεκτονική, η δρομολόγηση εφαρμόζεται αρχικά μέσω ελέγχου της θεματικής συνάφειας του ερωτήματος. Ερωτήματα που σχετίζονται με SQL injection, ύποπτη HTTP ή δικτυακή δραστηριότητα και τα εισαγμένα δεδομένα του συστήματος προωθούνται στα επόμενα στάδια επεξεργασίας, ενώ προφανώς άσχετα ερωτήματα τερματίζονται χωρίς ενεργοποίηση μηχανισμών ανάκτησης. Στη συνέχεια, το σύστημα επιλέγει διανυσματική, γραφική ή υβριδική ανάκτηση, ανάλογα με το αν το ερώτημα απαιτεί κυρίως τεχνική γνώση SQL injection, στοιχεία από την παρατηρούμενη δικτυακή επικοινωνία ή συνδυασμό των δύο. Με αυτό τον τρόπο το σύστημα αξιοποιεί τις δυνατότητες της δυναμικής δρομολόγησης που προσφέρουν τα συστήματα πολλαπλών agents, καθώς και την πρόσβαση στη χρήση εξωτερικών εργαλείων που μπορεί να εκμεταλλευτεί ο κάθε agent. Η κατανομή αυτών των λειτουργιών στους επιμέρους agents παρουσιάζεται αναλυτικά στο Κεφάλαιο 4.

3.3.2 Ροές Επαλήθευσης και Διορθωτικής Ανάκτησης

Η παραγωγή μιας απάντησης σε ένα σύστημα RAG δεν εξασφαλίζει από μόνη της ότι το περιεχόμενό της είναι πλήρως τεκμηριωμένο, συναφές με το ερώτημα ή απαλλαγμένο από μη υποστηριζόμενους ισχυρισμούς (Sapkota et al., 2026). Η ποιότητα της τελικής απόκρισης επηρεάζεται από τη συνάφεια των ανακτημένων τεκμηρίων, την πληρότητα του διαθέσιμου πλαισίου και τον τρόπο με τον οποίο το γλωσσικό μοντέλο αξιοποιεί την πληροφορία κατά τη σύνθεση της απάντησης. Για τον λόγο αυτό, σύγχρονες προσεγγίσεις Agentic RAG ενσωματώνουν μηχανισμούς αξιολόγησης, ανατροφοδότησης και επανάληψης της ανάκτησης ή της παραγωγής, όταν τα διαθέσιμα στοιχεία κρίνονται ανεπαρκή.

Ένα χαρακτηριστικό πρότυπο είναι το **Evaluator–Optimizer**, στο οποίο ένα αρχικό αποτέλεσμα παράγεται από ένα υποσύστημα παραγωγής και στη συνέχεια αξιολογείται από ξεχωριστό μηχανισμό. Ο αξιολογητής εξετάζει εάν η απάντηση ανταποκρίνεται στο ερώτημα, εάν στηρίζεται στα διαθέσιμα τεκμήρια και εάν απαιτεί βελτίωση. Εφόσον εντοπιστούν αδυναμίες, η ανατροφοδότηση μπορεί να αξιοποιηθεί για την αναθεώρηση της απάντησης ή

για την ενεργοποίηση συμπληρωματικής αναζήτησης. Με τον τρόπο αυτό, η τελική παραγωγή δεν αντιμετωπίζεται ως αμετάβλητο αποτέλεσμα ενός μόνο βήματος, αλλά ως τμήμα μιας επαναληπτικής διαδικασίας βελτίωσης, όπως φαίνεται στην Εικόνα 4 (Singh et al., 2025).

Εικόνα 4 Ροή Εργασίας Evaluator-Optimizer *(*Building Effective AI Agents \ Anthropic, *n.d.)*

Σχετική προσέγγιση αποτελεί το **Corrective RAG**, το οποίο ενσωματώνει μηχανισμούς αξιολόγησης της ποιότητας ή της συνάφειας της ανακτημένης πληροφορίας. Όταν το διαθέσιμο πλαίσιο θεωρείται ανεπαρκές ή μη κατάλληλο για την απάντηση του ερωτήματος, το σύστημα μπορεί να προχωρήσει σε διορθωτικές ενέργειες, όπως επαναδιατύπωση του ερωτήματος, νέα ανάκτηση από διαφορετική πηγή γνώσης ή αξιοποίηση εξωτερικών πηγών. Η λογική αυτή υποστηρίζει τη δυναμική προσαρμογή της ροής ανάκτησης και περιορίζει την παραγωγή απαντήσεων που βασίζονται σε ελλιπή ή άσχετα τεκμήρια(Singh et al., 2025).

Αντίστοιχη λογική υλοποιείται μέσω διακριτών σταδίων σύνθεσης και επαλήθευσης και στην υλοποίηση του προτεινόμενου συστήματος. Αρχικά, η ανακτημένη πληροφορία αξιοποιείται για την παραγωγή ενός προσχεδίου απάντησης. Στη συνέχεια, η απάντηση συγκρίνεται με το διαθέσιμο πλαίσιο, ώστε να εντοπιστούν μη υποστηριζόμενοι ισχυρισμοί, πιθανές αντιφάσεις ή ανεπαρκής κάλυψη του αρχικού ερωτήματος. Αν η απάντηση κριθεί επαρκής, οριστικοποιείται. Αν απαιτεί μόνο βελτίωση της διατύπωσης ή αφαίρεση μη τεκμηριωμένου περιεχομένου, επιστρέφει στο στάδιο αναθεώρησης. Αν, αντίθετα, διαπιστωθεί ότι λείπουν ουσιώδη τεκμήρια, ενεργοποιείται νέος κύκλος ανάκτησης με ειδικότερη καθοδήγηση ως προς την απαιτούμενη πληροφορία.

Η παραπάνω ροή συμβάλλει στον περιορισμό του κινδύνου παραγωγής μη τεκμηριωμένων απαντήσεων και ενισχύει τη διαφάνεια της διαδικασίας. Η δυνατότητα επιστροφής σε προηγούμενο στάδιο διαχωρίζει τις περιπτώσεις όπου χρειάζεται νέα πληροφορία από εκείνες όπου απαιτείται μόνο βελτίωση της υπάρχουσας απάντησης. Η λεπτομερής υλοποίηση της συγκεκριμένης ροής, οι ρόλοι των agents και οι κανόνες δρομολόγησης παρουσιάζονται στο Κεφάλαιο 4.

3.4 Εφαρμογή στην Κυβερνοασφάλεια

Οι τεχνικές Retrieval-Augmented Generation και Agentic RAG μπορούν να αξιοποιηθούν σε εργασίες κυβερνοασφάλειας όπου απαιτείται η συλλογή, η συσχέτιση και η ερμηνεία ετερογενών τεχνικών δεδομένων. Σε ένα πραγματικό περιβάλλον, η διερεύνηση ενός πιθανού περιστατικού μπορεί να απαιτεί συνδυασμό πληροφοριών από δικτυακή κίνηση, αρχεία καταγραφής εφαρμογών και συστημάτων, ειδοποιήσεις ασφαλείας, βάσεις γνώσης και τεκμηρίωση σχετικά με γνωστές τεχνικές επιθέσεων. Η αξιοποίηση γλωσσικών μοντέλων σε συνδυασμό με εξωτερικές πηγές γνώσης μπορεί να υποστηρίξει τη σύνθεση αυτών των στοιχείων σε πιο κατανοητές και τεκμηριωμένες αναλύσεις, χωρίς να υποκαθιστά την κρίση του αναλυτή ασφαλείας (Kaur et al., 2023; Mohamed, 2025).

Η εφαρμογή τέτοιων τεχνικών είναι ιδιαίτερα χρήσιμη σε διαδικασίες όπου η τελική απόφαση δεν προκύπτει από ένα μόνο γεγονός, αλλά από τη συσχέτιση πολλών επιμέρους ενδείξεων. Η ανάλυση αιτημάτων HTTP, των παραμέτρων εισόδου, των χρονικών σημάνσεων, των διευθύνσεων δικτύου και των σχετικών καταγραφών μπορεί να συμβάλει στην αναγνώριση πιθανών μοτίβων κακόβουλης δραστηριότητας. Η δυνατότητα ανάκτησης εξειδικευμένης τεχνικής γνώσης παράλληλα με τα παρατηρούμενα artifacts υποστηρίζει την ερμηνεία των ευρημάτων και τη διατύπωση τεκμηριωμένων υποθέσεων για τη φύση ενός περιστατικού.

Στις επόμενες ενότητες παρουσιάζεται η σύνδεση της προτεινόμενης προσέγγισης με τις διαδικασίες Threat Hunting και ανάλυσης περιστατικών, καθώς και η θέση της στο πλαίσιο των λειτουργιών Detect και Respond του NIST Cybersecurity Framework 2.0.

3.4.1 Σχετική Βιβλιογραφία και Τοποθέτηση της Προτεινόμενης Προσέγγισης

Οι προσεγγίσεις HybridRAG και Agentic RAG έχουν ήδη μελετηθεί στη σχετική βιβλιογραφία, τόσο ως προς τον συνδυασμό γράφων γνώσης και διανυσματικής ανάκτησης (Sarmah et al., 2024), όσο και ως προς τη χρήση agents για τη δρομολόγηση, την ανάκτηση, τη σύνθεση και την επαλήθευση απαντήσεων (Singh et al., 2025). Επιπλέον, αντίστοιχες Agentic RAG προσεγγίσεις έχουν εφαρμοστεί στην κυβερνοασφάλεια για την υποστήριξη ταξινόμησης και αναφοράς κυβερνοεπιθέσεων (Blefari et al., 2026).

Η παρούσα εργασία δεν παρουσιάζει το HybridRAG ή το Agentic RAG ως νέες αρχιτεκτονικές προσεγγίσεις. Η συμβολή της εντοπίζεται στην εφαρμογή τους στο ειδικό πρόβλημα της διερεύνησης πιθανών επιθέσεων SQL injection, με αξιοποίηση στοιχείων HTTP που εξάγονται από αρχεία PCAP, αναπαρίστανται σε γράφο γνώσης και συνδυάζονται με εξωτερική γνώση SQLi μέσω διανυσματικής ανάκτησης. Οι agents αξιοποιούνται για την επιλογή της κατάλληλης πηγής γνώσης, τη σύνθεση της απάντησης και τον έλεγχο της τεκμηρίωσής της, ώστε το σύστημα να συνδέει παρατηρούμενα δικτυακά artifacts με τεχνική ερμηνεία σχετική με SQL injection.

Με βάση αυτή την τοποθέτηση, οι επόμενες ενότητες εξετάζουν τον τρόπο με τον οποίο η ανάλυση τεχνικών artifacts και η συσχέτιση ετερογενών δεδομένων εντάσσονται σε διαδικασίες Threat Hunting και διερεύνησης περιστατικών.

3.4.2 Threat Hunting και Ανάλυση Περιστατικών

Το Threat Hunting αποτελεί διαδικασία προληπτικής και αναλυτικά καθοδηγούμενης αναζήτησης πιθανών απειλών που δεν έχουν ακόμη εντοπιστεί ή επιβεβαιωθεί από τους διαθέσιμους μηχανισμούς ασφαλείας. Σε αντίθεση με την αποκλειστική απόκριση σε ειδοποιήσεις που έχουν ήδη παραχθεί, η διαδικασία αυτή βασίζεται στη διερεύνηση ασθενών ενδείξεων κακόβουλης δραστηριότητας, στη διατύπωση υποθέσεων και στη συσχέτιση δεδομένων από διαφορετικές πηγές. Οι αναλυτές μπορούν να αξιοποιούν ενδείξεις παραβίασης, γνωστές τεχνικές επιτιθέμενων ή ασυνήθιστα πρότυπα συμπεριφοράς, ώστε να αναζητούν στοιχεία που μπορεί να υποδεικνύουν ένα υπό εξέλιξη ή μη ανιχνευμένο περιστατικό (*What Is Threat Hunting? | IBM*, n.d.).

Η ανάλυση ενός περιστατικού ασφαλείας αποσκοπεί στην κατανόηση της φύσης, της έκτασης και του πιθανού αντικτύπου ενός ύποπτου ή επιβεβαιωμένου συμβάντος. Η διαδικασία αυτή περιλαμβάνει τον εντοπισμό των σχετικών τεχνικών artifacts, τη συσχέτιση των επιμέρους γεγονότων και την ανασύνθεση της ακολουθίας ενεργειών που οδήγησε στο περιστατικό. Στο πλαίσιο εφαρμογών διαδικτύου, χρήσιμα τεκμήρια μπορεί να προέρχονται από αρχεία καταγραφής εξυπηρετητών (servers), αιτήματα HTTP και αποκρίσεις, αρχεία PCAP, δεδομένα από WAF/IDS, καθώς και από αρχεία καταγραφής της εφαρμογής και της βάσης δεδομένων (*What Is Threat Detection and Response (TDR)? | IBM*, n.d.).

Η ανάλυση αρχείων PCAP μπορεί να υποστηρίξει τη διερεύνηση της επικοινωνίας σε επίπεδο δικτύου και εφαρμογής, ιδίως όταν είναι διαθέσιμο το περιεχόμενο αιτημάτων HTTP. Στοιχεία όπως οι διευθύνσεις IP, οι θύρες, οι μέθοδοι HTTP, τα URI, οι παράμετροι query string, τα request bodies, τα cookies και οι χρονικές σημάνσεις μπορούν να συμβάλουν στην αναγνώριση και συσχέτιση πιθανών ενδείξεων SQL injection. Η διερεύνηση αυτή δεν αποδεικνύει πάντοτε την επιτυχία μιας επίθεσης, αλλά μπορεί να υποστηρίξει την αναγνώριση ύποπτων payloads και τη διαμόρφωση τεκμηριωμένων υποθέσεων για περαιτέρω έλεγχο.

Όπως είναι αντιληπτό σε ένα πραγματικό περιβάλλον ο όγκος των ειδοποιήσεων (alerts) οποίος παράγεται από εργαλεία όπως SIEM,XDR,EDR κτλ., είναι τεράστιος και αρκετές από αυτές απαιτούν εξειδικευμένη επικύρωση. Αυτό έχει ως αποτέλεσμα πολλές φορές όταν μία ειδοποίηση (alert) δημιουργείται, υψηλά εξειδικευμένοι αναλυτές ασφαλείας να πρέπει να ερευνήσουν το γεγονός που τη προκάλεσε, γεγονός το οποίο δυνητικά να αποσπά την προσοχή από μία πραγματικά κρίσιμη κυβερνοαπειλή. Λύση σε αυτά τα προβλήματα, υπόσχεται να δώσει η χρήση των μεγάλων γλωσσικών μοντέλων, τα οποία λόγω των αξιοσημείωτων δυνατοτήτων που παρουσιάζουν στην επεξεργασία της φυσικής γλώσσας (NLP-Natural Language Processing), μπορούν να βοηθήσουν τους αναλυτές στην ανάλυση των απαραίτητων δεδομένων και ειδοποιήσεων γενικότερα (Blefari et al., 2026).

Σε αυτό το πλαίσιο, οι τεχνικές RAG και Agentic RAG μπορούν να λειτουργήσουν υποστηρικτικά προς τους αναλυτές ασφαλείας. Η ανάκτηση παρατηρούμενων στοιχείων από την επικοινωνία, σε συνδυασμό με τεχνική γνώση για γνωστές μεθόδους επίθεσης, διευκολύνει την ερμηνεία ύποπτων artifacts και τη σύνθεση μιας κατανοητής αναφοράς διερεύνησης. Η αξία της προσέγγισης δεν βρίσκεται στην αντικατάσταση της ανθρώπινης κρίσης ή των

υφιστάμενων μηχανισμών προστασίας, αλλά στη μείωση του χρόνου που απαιτείται για την αρχική συσχέτιση και ερμηνεία των διαθέσιμων στοιχείων.

3.4.2 Σύνδεση με το NIST CSF 2.0

Το NIST Cybersecurity Framework 2.0 παρέχει ένα πλαίσιο υψηλού επιπέδου για την οργάνωση και διαχείριση κινδύνων κυβερνοασφάλειας από οργανισμούς διαφορετικού μεγέθους και τομέα. Ο πυρήνας του πλαισίου οργανώνεται στις λειτουργίες Govern, Identify, Protect, Detect, Respond και Recover, οι οποίες δεν αποτελούν αυστηρή χρονική ακολουθία, αλλά αλληλοσυνδεόμενες περιοχές δραστηριότητας για τη διαχείριση του κινδύνου κυβερνοασφάλειας (The NIST Cybersecurity Framework (CSF) 2.0, 2024).

Η προτεινόμενη αρχιτεκτονική συνδέεται κυρίως με τις λειτουργίες Detect και Respond. Η λειτουργία Detect αφορά την έγκαιρη αναγνώριση και ανάλυση ανωμαλιών, ενδείξεων παραβίασης και άλλων πιθανών κακόβουλων συμβάντων. Στο πλαίσιο της παρούσας εργασίας, η επεξεργασία στοιχείων HTTP από αρχεία PCAP και η συσχέτισή τους με γνωστά μοτίβα SQL injection υποστηρίζουν την ανάδειξη πιθανών ενδείξεων κακόβουλης δραστηριότητας. Η λειτουργία Respond αφορά ενέργειες που ακολουθούν την αναγνώριση ενός περιστατικού, συμπεριλαμβανομένης της ανάλυσης, της διαχείρισης και της επικοινωνίας των σχετικών ευρημάτων. Η παραγωγή τεκμηριωμένων απαντήσεων και η παρουσίαση των αντίστοιχων HTTP artifacts μπορούν να υποστηρίξουν το αναλυτικό στάδιο της διερεύνησης, χωρίς να υποκαθιστούν τις επιχειρησιακές αποφάσεις περιορισμού ή απόκρισης.

Η προτεινόμενη υλοποίηση δεν αποτελεί πλήρη πλατφόρμα διαχείρισης περιστατικών ούτε αυτοματοποιημένο μηχανισμό απόκρισης. Λειτουργεί ως εργαλείο υποστήριξης της αρχικής διερεύνησης πιθανών ενδείξεων SQL injection, παρέχοντας δυνατότητες ανάκτησης, συσχέτισης και ερμηνείας τεχνικών τεκμηρίων. Με βάση το παραπάνω θεωρητικό πλαίσιο, το επόμενο κεφάλαιο παρουσιάζει τον σχεδιασμό και την αρχιτεκτονική του συστήματος, από την επεξεργασία των αρχείων PCAP και την οργάνωση της γνώσης έως τη λειτουργία του Agentic RAG υποσυστήματος.

4. Σχεδιασμός και Αρχιτεκτονική Συστήματος

Το παρόν κεφάλαιο εξετάζει τη σχεδίαση και την αρχιτεκτονική του προτεινόμενου συστήματος για την υποστήριξη της ανίχνευσης και ανάλυσης δραστηριότητας συμβατής με επιθέσεις SQLi σε εφαρμογές διαδικτύου μέσω αρχείων PCAP, με αξιοποίηση τεχνικών HybridRAG και Agentic RAG. Η αρχιτεκτονική του συστήματος θεμελιώνεται στη λογική του Modular RAG, καθώς συγκροτείται από επιμέρους λειτουργικές ενότητες οι οποίες συνεργάζονται για την εισαγωγή, αναπαράσταση, ανάκτηση και αξιοποίηση της πληροφορίας.

Κεντρικό στοιχείο της προτεινόμενης προσέγγισης αποτελεί η υβριδική οργάνωση της γνώσης, μέσω του συνδυασμού διανυσματικής και γραφικής αναπαράστασης των δεδομένων. Η επιλογή αυτή αποσκοπεί στην υποστήριξη τόσο σημασιολογικής ανάκτησης βάσει ομοιότητας όσο και συσχετιστικής διερεύνησης βάσει οντοτήτων και σχέσεων, ενισχύοντας τη δυνατότητα διαχείρισης σύνθετων ερωτημάτων και πολυβηματικού συλλογισμού.

Περαιτέρω, η αρχιτεκτονική του συστήματος οργανώνεται γύρω από ένα σύνολο agents, οι οποίοι αναλαμβάνουν διακριτές αρμοδιότητες κατά την ανάλυση του ερωτήματος, την επιλογή στρατηγικής ανάκτησης, τη σύνθεση των αποτελεσμάτων και την παραγωγή της τελικής απόκρισης. Υπό το πρίσμα αυτό, η αρχιτεκτονική δεν περιορίζεται σε μία στατική ακολουθία βημάτων, αλλά υλοποιεί μία δυναμική ροή επεξεργασίας, προσαρμοζόμενη στα χαρακτηριστικά του εκάστοτε ερωτήματος του χρήστη.

Στις ενότητες που ακολουθούν παρουσιάζονται διαδοχικά η συνολική αρχιτεκτονική του συστήματος, η διαδικασία εισαγωγής και προεπεξεργασίας των δεδομένων, το υβριδικό επίπεδο γνώσης, η δομή του Agentic σχήματος και η ροή εκτέλεσης των ερωτημάτων. Μέσω της ανάλυσης αυτής επιδιώκεται η συστηματική αποτύπωση των επιμέρους δομικών και λειτουργικών χαρακτηριστικών της προτεινόμενης υλοποίησης.

4.1 Συνολική Αρχιτεκτονική Συστήματος

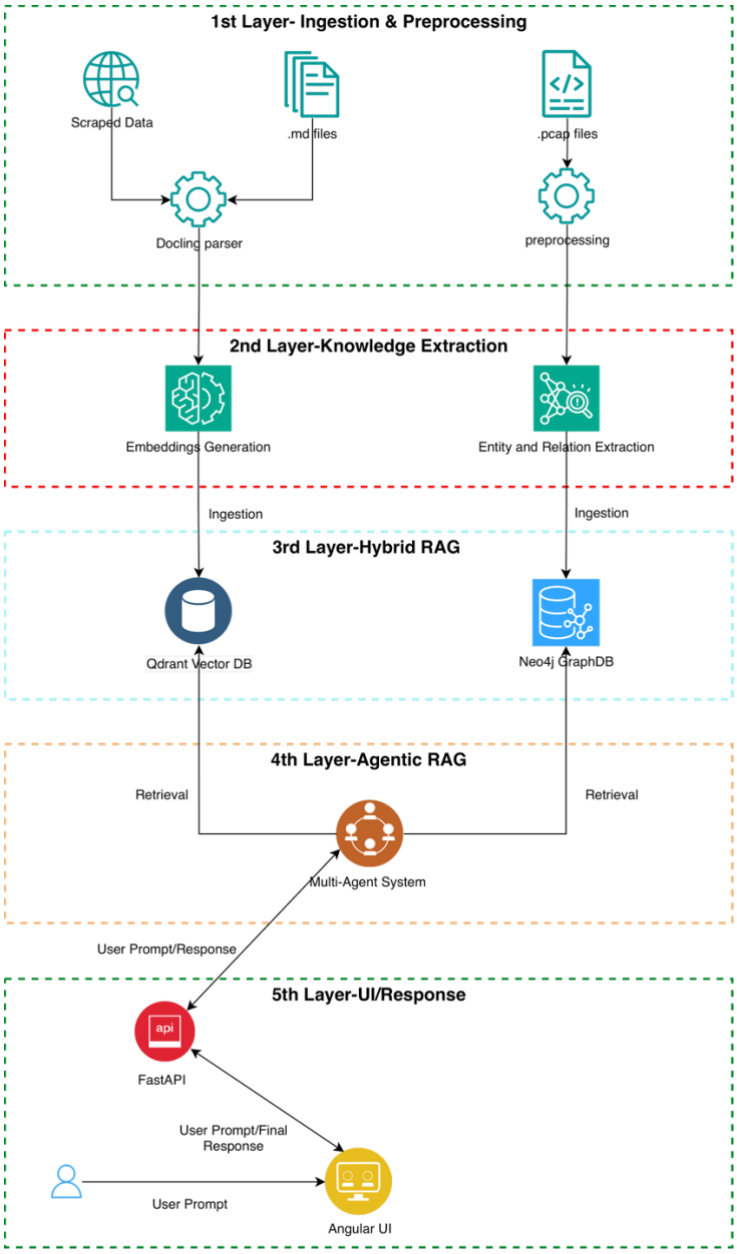
Η αρχιτεκτονική του προτεινόμενου συστήματος υποστήριξης διερεύνησης και ανάλυσης ενδείξεων πιθανής δραστηριότητας SQLi σε εφαρμογές διαδικτύου μέσα από αρχεία PCAP, έχει σχεδιαστεί ως αρχιτεκτονική πέντε επιπέδων (Εικόνα 5):

1) Εισαγωγή και προεπεξεργασία δεδομένων (Ingestion and Preprocessing)
2) Εξαγωγή Γνώσης (Knowledge Extraction)
3) Υβριδική Υποδομή Γνώσης (HybridRAG προσέγγιση)
4) Υποσύστημα Agentic RAG (Multi Agent System - MAS)
5) Διεπαφή Χρήστη (UI) και τελική απάντηση LLM

Στον παρακάτω πίνακα (Πίνακας 1) παρουσιάζονται οι τεχνικές προδιαγραφές του κάθε επιπέδου της αρχιτεκτονικής του συστήματος:

Επίπεδο		**Λειτουργικές Μονάδες**	**Κύριες Τεχνολογίες**	**Έξοδοι Επιπέδου**
1	Ingestion & Preprocessing	web crawler/scraper, parser, chunking and metadata tagger	Crawl4AI, Docling	Structured chunks
2	Knowledge Extraction	Entity and Relation Extraction Pipeline, Embeddings generation	Neo4j Cypher Query, HuggingFace Embeddings	Entity List, Relation Triples,
3	HybridRAG	Vector Database Graph Database	QdrantDB Neo4j	Vectors Knowledge Graph
4	Agentic RAG	Multi Agent System	Langchain/LangGraph/ LangSmith/Hugging Face Router	Retrieval Results
5	UI/Response	Frontend/Backend App	Angular/FastAPI	Final Results, Natural language responses

Πίνακας 1 Τεχνικές Προδιαγραφές Αρχιτεκτονικής Συστήματος



Εικόνα 5 Συνολική Αρχιτεκτονική Συστήματος

4.2 Διαδικασία Εισαγωγής και Προεπεξεργασίας Δεδομένων (Data Ingestion and Preprocessing Pipeline) – 1ο Επίπεδο (1st Layer)

Η διαδικασία της εισαγωγής και της προεπεξεργασίας των δεδομένων αποτελεί το πρώτο επίπεδο της αρχιτεκτονικής 5-επιπέδων της προτεινόμενης υλοποίησης. Από το παραπάνω

διάγραμμα της συνολικής αρχιτεκτονικής (Εικόνα 5), μπορούμε να διακρίνουμε δύο διαφορετικές ροές εισαγωγής και επεξεργασίας. Τη ροή επεξεργασίας που σχετίζεται με τα δεδομένα που θα αποτελέσουν τη γνώση που θα αποθηκευτεί στη διανυσματική βάση (Qdrant) και τη δεύτερη ροή, η οποία αφορά την εισαγωγή και επεξεργασία των δεδομένων με βάση τα οποία θα κατασκευασθεί ο γράφος γνώσης (Neo4j). Οι δύο αυτές ροές είναι ανεξάρτητες μεταξύ τους και απαραίτητες για τη λειτουργία του συστήματος, κάτι που σημαίνει ότι η μία δεν αποτελεί εναλλακτική της άλλης.

4.2.1 Ροή Εισαγωγής και Προεπεξεργασίας Δεδομένων στη Διανυσματική Βάση Qdrant

Η συγκεκριμένη ροή είναι αυτή που αξιοποιείται από το σύστημα έτσι ώστε ακατέργαστα δεδομένα να επεξεργασθούν και να μετατραπούν σε διανύσματα (embeddings) προκειμένου να αποθηκευτούν στη διανυσματική βάση Qdrant. Τα δεδομένα που εισάγονται στη συγκεκριμένη βάση δεδομένων αποτελούν την εξειδικευμένη γνώση βάσει της οποίας το σύστημα μπορεί να ερμηνεύει τεχνικά στοιχεία και να αξιολογεί κατά πόσο αυτά συνιστούν ενδείξεις πιθανής δραστηριότητας συμβατής με SQL injection.

Προκειμένου να αρχικοποιηθεί η διανυσματική βάση δεδομένων με όλη την απαραίτητη γνώση, έχουν επιλεχθεί δύο τύποι δεδομένων, αρχεία markdown (.md) τα οποία περιέχουν πληροφορίες και παραδείγματα επιθέσεων SQLi, καθώς και πηγές διαδικτύου που περιέχουν το θεωρητικό υπόβαθρο αυτών των επιθέσεων. Τα βασικά εργαλεία που χρησιμοποιούνται για την εισαγωγή και την προεπεξεργασία των δεδομένων είναι τα εξής:

- Crawl4AI:

Για τη συλλογή δεδομένων από το διαδίκτυο χρησιμοποιείται το Crawl4AI, μια open-source βιβλιοθήκη Python για web crawling/scraping, η οποία υποστηρίζει ασύγχρονη ανάκτηση ιστοσελίδων και εξαγωγή καθαρού περιεχομένου κατάλληλου για συστήματα RAG και LLM agents (*Crawl4AI Documentation (v0.8.x)*, n.d.). Αφού συλλεχθούν τα δεδομένα, επεξεργάζονται και στέλνονται με μορφή markdown (.md) στο Docling parser.

- Docling Parser:

Για την ανάλυση και προεπεξεργασία των εγγράφων χρησιμοποιείται το Docling, μια open-source βιβλιοθήκη της Python, η οποία αξιοποιείται τόσο για την επεξεργασία

(parsing) όσο και για τον τεμαχισμό (chunking) του περιεχομένου. Ειδικότερα, μετατρέπει αρχεία διαφορετικών μορφών σε ενοποιημένη και δομημένη αναπαράσταση, διευκολύνοντας την εξαγωγή κειμένου, πινάκων, πληροφοριών διάταξης και άλλων στοιχείων χρήσιμων για την περαιτέρω επεξεργασία από το σύστημα (*Docling*, n.d.). Στην προτεινόμενη υλοποίηση, το Docling εφαρμόζεται στα αρχεία markdown που παράγονται από τον web crawler Crawl4AI, καθώς και στα markdown έγγραφα που εισάγονται απευθείας στο σύστημα. Στη συνέχεια, με αξιοποίηση του Hybrid Chunker της ίδιας βιβλιοθήκης, το περιεχόμενο τεμαχίζεται σε συνεκτικά τμήματα, με τρόπο που λαμβάνει υπόψη τόσο τη δομή του εγγράφου όσο και περιορισμούς σχετικούς με το μέγεθος των αποσπασμάτων, ώστε τα παραγόμενα κομμάτια (chunks) να είναι κατάλληλα για μετατροπή σε διανύσματα (embeddings).

Η εισαγωγή των δεδομένων με την παραπάνω ροή, πραγματοποιείται κατά την αρχική παραμετροποίηση του συστήματος και την εκτέλεση του script initial_ingestion_qdrant.py (Παράρτημα Α: Κώδικας Υλοποίησης της Εργασίας).

4.2.2 Ροή Εισαγωγής και Προεπεξεργασίας Δεδομένων στη Γραφική Βάση Neo4j

Η δεύτερη ροή εισαγωγής και προεπεξεργασίας δεδομένων αφορά την επεξεργασία αρχείων καταγραφής πακέτων δικτυακής κίνησης σε μορφή PCAP, με σκοπό την εξαγωγή δομημένων δεδομένων κατάλληλων για εισαγωγή στη γραφική βάση Neo4j. Για την υλοποίηση της συγκεκριμένης διαδικασίας αξιοποιείται το PyShark, βιβλιοθήκη Python η οποία λειτουργεί ως διεπαφή προς το TShark, δηλαδή το εργαλείο γραμμής εντολών του Wireshark, και επιτρέπει την ανάλυση αρχείων PCAP μέσω των μηχανισμών αποκωδικοποίησης πρωτοκόλλων που διαθέτει το οικοσύστημα του Wireshark(*PyShark Usage Documentation*, n.d.). Η επιλογή του PyShark αιτιολογείται από το γεγονός ότι παρέχει αξιόπιστη και συστηματική πρόσβαση σε πεδία πολλαπλών επιπέδων της δικτυακής επικοινωνίας, διευκολύνοντας τον μετασχηματισμό της ακατέργαστης δικτυακής κίνησης σε αναπαράσταση κατάλληλη για γραφική μοντελοποίηση.

Στο πλαίσιο της παρούσας εργασίας, η επεξεργασία των αρχείων PCAP εστιάζει στο υποσύνολο της επικοινωνίας HTTP, καθώς αυτό συγκεντρώνει το κύριο ενδιαφέρον για την ανάλυση ενδείξεων που σχετίζονται με πιθανές επιθέσεις SQL injection σε περιβάλλον εφαρμογών διαδικτύου. Για κάθε σχετικό πακέτο εξάγονται αρχικά βασικά χαρακτηριστικά

ταυτοποίησης της επικοινωνίας, όπως η διεύθυνση IP πηγής και προορισμού, οι θύρες προέλευσης και προορισμού, καθώς και η χρονική σήμανση του πακέτου. Τα χαρακτηριστικά αυτά επιτρέπουν την αποτύπωση του πλαισίου της επικοινωνίας και τη συσχέτιση επιμέρους αιτημάτων με συγκεκριμένες ροές δικτυακής δραστηριότητας.

Σε δεύτερο επίπεδο, εξάγονται στοιχεία επιπέδου εφαρμογής που αφορούν το ίδιο το αίτημα HTTP, όπως η μέθοδος αιτήματος, το host, το URI, το πλήρες URI, το user-agent, το referer, τα cookies και το σώμα του αιτήματος, όταν αυτό είναι διαθέσιμο. Ιδιαίτερη σημασία αποδίδεται στην εξαγωγή των επιμέρους παραμέτρων του αιτήματος, οι οποίες αναλύονται από διαφορετικά σημεία εισόδου, και ειδικότερα από το query string του URI, από το σώμα αιτημάτων τύπου POST, καθώς και από τα cookies. Η ανάλυση αυτή επιτρέπει τον περαιτέρω διαχωρισμό της πληροφορίας σε επιμέρους οντότητες και ιδιότητες, στοιχείο κρίσιμο για τη μεταγενέστερη αναπαράστασή τους στον γράφο γνώσης.

Η παραγόμενη πληροφορία δεν παραμένει σε επίπεδο απλής αποθήκευσης μεταδεδομένων, αλλά αξιοποιείται ως ενδιάμεση αναπαράσταση για τη διαδικασία αναγνώρισης οντοτήτων και σχέσεων. Με τον τρόπο αυτό καθίσταται εφικτή η κατασκευή ενός γράφου γνώσης που αποτυπώνει όχι μόνο μεμονωμένα αιτήματα HTTP, αλλά και τις μεταξύ τους συσχετίσεις με αρχεία PCAP, δικτυακά άκρα, παραμέτρους, τιμές παραμέτρων και λοιπά στοιχεία της επικοινωνίας. Η γραφική αυτή αναπαράσταση υποστηρίζει αποτελεσματικότερα την αναζήτηση σύνθετων μοτίβων, την ανάδειξη ύποπτων αλληλουχιών ενεργειών και τη συσχέτιση ενδείξεων που σχετίζονται με επιθέσεις SQL injection.

Η εισαγωγή των δεδομένων μέσω της συγκεκριμένης ροής πραγματοποιείται κατά το στάδιο αρχικής παραμετροποίησης του συστήματος, μέσω της εκτέλεσης του script initial_ingestion_neo4j.py (Παράρτημα Α: Κώδικας Υλοποίησης της Εργασίας). Κατά το στάδιο αυτό, τα αρχεία PCAP υποβάλλονται σε επεξεργασία, οι εγγραφές HTTP μετασχηματίζονται σε κατάλληλα δομημένα αντικείμενα και στη συνέχεια προωθούνται στο στάδιο κατασκευής και αποθήκευσης του γράφου γνώσης στη Neo4j.

4.3 Εξαγωγή Γνώσης (Knowledge Extraction) - 2ο Επίπεδο (2nd Layer)

Η εξαγωγή της γνώσης αποτελεί το δεύτερο επίπεδο της συνολικής αρχιτεκτονικής και στην πράξη τη συνέχεια των ροών εισαγωγής και προεπεξεργασίας, οι οποίες αναλύθηκαν παραπάνω. Στο συγκεκριμένο επίπεδο, το σύστημα δέχεται τα εξαγόμενα στοιχεία της προεπεξεργασίας του προηγούμενου επιπέδου και με κατάλληλες διαδικασίες εξαγωγής της απαραίτητης γνώσης από αυτά, παράγει τα τελικά δεδομένα που θα αποθηκευτούν στην υβριδική υποδομή γνώσης, που αποτελεί το τρίτο επίπεδο της αρχιτεκτονικής σχεδίασης. Το δεύτερο επίπεδο, καθώς αποτελεί φυσική εξέλιξη των παραπάνω ροών, είναι προτιμότερο να εξετασθεί από τη σκοπιά των δύο ανεξάρτητων αυτών ροών.

4.3.1 Διανυσματοποίηση και Δημιουργία Embeddings

Στη παρούσα φάση της ροής εισαγωγής των δεδομένων που προορίζονται για τη διανυσματική βάση Qdrant, τα κομμάτια (chunks) των δεδομένων που έχουν δημιουργηθεί στο προηγούμενο στάδιο της προεπεξεργασίας, πρέπει να μετατραπούν σε διανύσματα προκειμένου να αποθηκευτούν στη βάση δεδομένων. Τα συγκεκριμένα chunks μετατρέπονται σε διανύσματα με τη χρήση ενός μοντέλου embeddings και αποθηκεύονται στη συνέχεια στη βάση. Η αποθήκευση των chunks των δεδομένων με διανυσματική αναπαράσταση, δίνει τη δυνατότητα στο σύστημα να πραγματοποιεί αποτελεσματικές αναζητήσεις ομοιότητας κατά τη διάρκεια της διαδικασίας ανάκτησης δεδομένων (Y. Gao et al., 2023).

Στην προτεινόμενη υλοποίηση, η αποθήκευση των chunks γίνεται σε δύο διαφορετικούς τύπους διανυσμάτων, πυκνά (dense) και αραιά (sparse), με σκοπό να υποστηρίζεται υβριδική διανυσματική ανάκτηση (Hybrid Retrieval) των δεδομένων. Η υβριδική ανάκτηση των δεδομένων από μία διανυσματική βάση, με τις τελευταίες τεχνολογικές εξελίξεις, θεωρείται ως βέλτιστη πρακτική, καθώς συνδυάζει τα πλεονεκτήματα δύο διαφορετικών τεχνικών, τη βασισμένη σε λέξεις κλειδιά (keyword-based) και τη σημασιολογική (semantic).

Τα πυκνά διανύσματα (dense vectors) παράγονται από προεκπαιδευμένα μοντέλα, βασισμένα στην αρχιτεκτονική των Transformer και αναπαρίστανται σε χώρους Ευκλείδειας απόστασης χαμηλότερων διαστάσεων, επιτρέποντας τη μοντελοποίηση σημασιολογικών (Mei et al., 2025). Για τη δημιουργία τους στο προτεινόμενο σύστημα χρησιμοποιείται το μοντέλο embeddings «sentence-transformers/all-MiniLM-L12-v2».

Τα αραιά διανύσματα (sparse vectors) βασίζονται παραδοσιακά στη λεκτική ταύτιση. Τα μοντέλα δημιουργίας αυτών των διανυσμάτων αξιοποιούν αλγορίθμους όπως τον BM25 που εστιάζει στη συχνότητα εμφάνισης και παρουσίας των όρων (Huang & Huang, 2026). Για τη δημιουργία τους στο σύστημα χρησιμοποιείται η Python βιβλιοθήκη FastEmbed με τη χρήση του μοντέλου «Qdrant/bm25».

Οι δύο παραπάνω προσεγγίσεις είναι συμπληρωματικές, καθώς τα αραιά διανύσματα βοηθούν τα πυκνά να διαχειριστούν ερωτήματα που περιέχουν σπάνιες οντότητες ή εξειδικευμένους όρους, βελτιώνοντας τη στιβαρότητα του συστήματος (Y. Gao et al., 2023). Συνεπώς, με τη στρατηγική της υβριδικής διανυσματικής ανάκτησης, μεγιστοποιείται η ακρίβεια τόσο σε σημασιολογικό όσο και σε λεκτικό επίπεδο, με τα πυκνά διανύσματα (embeddings) να προσφέρουν το γενικότερο πλαίσιο των επιθέσεων SQLi, και τα αραιά να βοηθούν στον εντοπισμό των εξειδικευμένων όρων και των συνηθισμένων payloads των διαφόρων τύπων SQLi επιθέσεων.

4.3.2. Αναγνώριση Οντοτήτων και Εξαγωγή Σχέσεων

Μετά το πρώτο στάδιο επεξεργασίας των αρχείων PCAP και εξαγωγής των επιμέρους εγγραφών HTTP, ακολουθεί το στάδιο μετασχηματισμού και εισαγωγής της πληροφορίας στη γραφική βάση Neo4j. Στο στάδιο αυτό, η δομημένη αναπαράσταση που έχει παραχθεί από το PyShark δεν παραμένει σε μορφή απλών εγγραφών, αλλά μετατρέπεται σε σύνολο οντοτήτων και συσχετίσεων, οι οποίες αποτυπώνονται στον γράφο γνώσης μέσω της γλώσσας ερωτημάτων Cypher της Neo4j. Η συγκεκριμένη διαδικασία συνιστά τον πυρήνα του σταδίου αναγνώρισης οντοτήτων και εξαγωγής σχέσεων, καθώς επιτρέπει τη μετάβαση από την ακολουθιακή αναπαράσταση αιτημάτων HTTP σε μία γραφική δομή κατάλληλη για σημασιολογική διερεύνηση και πολυβηματικό συλλογισμό.

Η υλοποίηση της εισαγωγής βασίζεται σε προκαθορισμένο σχήμα γράφου, στο οποίο αποτυπώνονται οι βασικές οντότητες που προκύπτουν από την επικοινωνία HTTP. Ειδικότερα, δημιουργούνται κόμβοι που αντιστοιχούν στο ίδιο το αρχείο PCAP, στο αίτημα HTTP, στις διευθύνσεις IP, στον στόχο τύπου host και στις παραμέτρους του αιτήματος. Η επιλογή των συγκεκριμένων τύπων οντοτήτων αποσκοπεί στην αναπαράσταση τόσο του τεχνικού πλαισίου της επικοινωνίας όσο και των επιμέρους στοιχείων εισόδου που είναι δυνατόν να συνδέονται με ύποπτη ή κακόβουλη δραστηριότητα.

Παράλληλα, ορίζονται και οι αντίστοιχες συσχετίσεις μεταξύ των ανωτέρω οντοτήτων, ώστε να αποτυπώνεται η εσωτερική δομή της δικτυακής δραστηριότητας. Συγκεκριμένα, κάθε κόμβος ημέρας συνδέεται με τα αντίστοιχα αρχεία PCAP, κάθε αρχείο PCAP με τα αιτήματα HTTP που περιέχει, ενώ κάθε αίτημα HTTP συσχετίζεται με τη διεύθυνση IP προέλευσης, τη διεύθυνση IP προορισμού, τον στόχο τύπου host και το σύνολο των παραμέτρων που εξήχθησαν από το query string, το σώμα του αιτήματος ή τα cookies. Με τον τρόπο αυτό, το σύστημα δεν αποθηκεύει απλώς μεμονωμένα μεταδεδομένα, αλλά οργανώνει τη γνώση σε μορφή γράφου συσχετίσεων, κατάλληλου για γραφική ανάκτηση και ανάλυση σύνθετων προτύπων.

Για τη διασφάλιση της συνέπειας και της μοναδικότητας των δεδομένων κατά την εισαγωγή, αξιοποιούνται περιορισμοί ακεραιότητας στη Neo4j για βασικούς τύπους κόμβων, όπως η ημερομηνία, το όνομα του αρχείου PCAP, η διεύθυνση IP, η τιμή του host, το μοναδικό αναγνωριστικό του αιτήματος HTTP και η ταυτότητα κάθε παραμέτρου. Η χρήση των περιορισμών αυτών είναι κρίσιμη, καθώς αποτρέπει την εισαγωγή πλεοναζόντων κόμβων και εξασφαλίζει ότι η ίδια οντότητα αναπαρίσταται ενιαία στον γράφο, ακόμη και όταν εμφανίζεται επανειλημμένα σε διαφορετικά πακέτα ή αρχεία καταγραφής. Κατ' επέκταση, υποστηρίζεται μια πιο καθαρή και συνεκτική γραφική αναπαράσταση της γνώσης.

Η εισαγωγή των δεδομένων στον γράφο πραγματοποιείται με τη χρήση της εντολής UNWIND του Cypher, ώστε πολλαπλές εγγραφές HTTP να επεξεργάζονται ως ενιαίο σύνολο μέσα σε κάθε κύκλο εισαγωγής. Για κάθε εγγραφή εφαρμόζονται πράξεις MERGE, μέσω των οποίων είτε δημιουργούνται νέοι κόμβοι και σχέσεις είτε επαναχρησιμοποιούνται ήδη υπάρχοντα στοιχεία του γράφου, όταν αυτά ταυτίζονται με τα χαρακτηριστικά της νέας εγγραφής. Η επιλογή του MERGE έναντι καθαρών εντολών δημιουργίας είναι ουσιώδης, επειδή επιτρέπει την προοδευτική οικοδόμηση του γράφου γνώσης, χωρίς απώλεια συνέπειας και χωρίς ανεξέλεγκτο πολλαπλασιασμό όμοιων οντοτήτων.

Ιδιαίτερη σημασία έχει το γεγονός ότι η διαδικασία εισαγωγής οργανώνεται σε παρτίδες (batches) εγγραφών και όχι σε μεμονωμένες συναλλαγές για κάθε αίτημα HTTP. Η πρακτική αυτή μειώνει το κόστος επικοινωνίας με τη βάση και βελτιώνει τη συνολική απόδοση της συγκεκριμένης ροής, ιδίως όταν το σύστημα επεξεργάζεται μεγάλο πλήθος πακέτων προερχόμενων από πολλαπλά αρχεία PCAP. Κάθε παρτίδα μετασχηματίζεται αρχικά σε

κατάλληλη ενδιάμεση μορφή και στη συνέχεια προωθείται στη Neo4j ως ενιαίο σύνολο εγγραφών, γεγονός που καθιστά την όλη διαδικασία περισσότερο αποδοτική και κλιμακώσιμη.

Πέραν της ομαδοποίησης σε παρτίδες, για την περαιτέρω επιτάχυνση της ροής εισαγωγής αξιοποιείται πολυνηματική επεξεργασία (multithreading). Ειδικότερα, το σύστημα χρησιμοποιεί ομάδα worker threads, τα οποία αναλαμβάνουν παράλληλα την επεξεργασία διαφορετικών αρχείων PCAP, επιτρέποντας την ταυτόχρονη εκτέλεση πολλαπλών εργασιών εισαγωγής. Η επιλογή αυτή κρίνεται ιδιαίτερα σημαντική στο στάδιο της αρχικής φόρτωσης του γράφου γνώσης, καθώς μειώνει τον συνολικό χρόνο επεξεργασίας μεγάλων συλλογών αρχείων και καθιστά την υποδομή πιο κατάλληλη για πρακτική αξιοποίηση σε ρεαλιστικά σύνολα δεδομένων.

Για την ασφαλή λειτουργία της παράλληλης επεξεργασίας, υλοποιούνται επιπλέον μηχανισμοί παρακολούθησης της προόδου και του αποτελέσματος της εισαγωγής. Συγκεκριμένα, το σύστημα διατηρεί ξεχωριστά αρχεία καταγραφής για τα αρχεία PCAP που έχουν ήδη εισαχθεί επιτυχώς και για εκείνα των οποίων η επεξεργασία απέτυχε, ώστε να υποστηρίζεται η επανεκκίνηση της διαδικασίας χωρίς επαναλαμβανόμενη εισαγωγή των ίδιων δεδομένων. Παράλληλα, πριν από την επεξεργασία κάθε αρχείου πραγματοποιείται έλεγχος εγκυρότητας μέσω του TShark. Οι λειτουργίες αυτές ενισχύουν την ανθεκτικότητα και την επιχειρησιακή αξιοπιστία του μηχανισμού αρχικής εισαγωγής.

Συνολικά, το στάδιο αυτό λειτουργεί ως γέφυρα ανάμεσα στην εξαγωγή τεχνικών χαρακτηριστικών από τα αρχεία PCAP και στη μοντελοποίησή τους σε γράφο γνώσης. Η αναγνώριση οντοτήτων και η εξαγωγή σχέσεων δεν υλοποιούνται ως ανεξάρτητη γλωσσική διεργασία τύπου NER σε φυσικό κείμενο, αλλά ως δομικός μετασχηματισμός των εγγραφών HTTP σε κόμβους και σχέσεις της Neo4j, με βάση προκαθορισμένους κανόνες χαρτογράφησης. Με την προσέγγιση αυτή, ο παραγόμενος γράφος αποκτά τη μορφή οργανωμένης υποδομής γνώσης, ικανής να υποστηρίξει τόσο τη μεταγενέστερη ανάκτηση πληροφορίας όσο και την ανάλυση σύνθετων μοτίβων που σχετίζονται με επιθέσεις SQL injection.

4.4 Υβριδική Υποδομή Γνώσης (Hybrid Knowledge Infrastructure) - 3ο Επίπεδο (3rd Layer)

Όσο αφορά την υποδομή γνώσης, δηλαδή το σύστημα αποθήκευσης και ανάκτησης των δεδομένων του συστήματος, το σύστημα ακολουθεί την προσέγγιση της υβριδικής αρχιτεκτονικής, γνωστή ως HybridRAG στην οποία αξιοποιούνται συνδυαστικά τα πλεονεκτήματα τόσο του VectorRAG, όσο και του GraphRAG. Για την αποθήκευση των διανυσμάτων που παράγονται από το μοντέλο embeddings, χρησιμοποιείται η διανυσματική βάση Qdrant, ενώ για την αποθήκευση του γράφου γνώσης χρησιμοποιείται η γραφική βάση δεδομένων της Neo4j.

4.4.1 Διανυσματική Βάση Δεδομένων Qdrant

Η Qdrant είναι μια open-source διανυσματική βάση δεδομένων, η οποία έχει σχεδιαστεί για την αποθήκευση και αποδοτική αναζήτηση διανυσμάτων υψηλής διάστασης που προκύπτουν από μοντέλα αναπαράστασης δεδομένων. Παρέχει δυνατότητες αναζήτησης βάσει ομοιότητας, καθώς και συνδυασμού της αναζήτησης με φίλτρα μεταδεδομένων (metadata), γεγονός που την καθιστά κατάλληλη για σύγχρονες εφαρμογές σημασιολογικής ανάκτησης πληροφορίας. Στο πλαίσιο της προτεινόμενης υλοποίησης, τα δεδομένα αποθηκεύονται με δύο διαφορετικές μορφές διανυσμάτων, αραιά (sparse) και πυκνά (dense) έτσι ώστε να υποστηρίζεται η υβριδική διανυσματική ανάκτηση. Επίσης αυτή η τακτική δίνει τη δυνατότητα αν απαιτηθεί να γίνει επιλογή οποιαδήποτε μορφής ανάκτησης, μόνο αραιά (sparse retrieval) ή μόνο πυκνά (dense retrieval), καθώς τα διανύσματα έχουν αποθηκευτεί και με τους δύο τρόπους (*Qdrant Integration - Docs by LangChain*, n.d.).

Η μετρική απόστασης που χρησιμοποιείται στο προτεινόμενο σύστημα για τα πυκνά διανύσματα και βάση της οποίας αποφασίζεται το σκορ ομοιότητας, είναι η συνημιτονική ομοιότητα (cosine similarity), καθώς είναι η προτιμότερη επιλογή έναντι της μετρικής εσωτερικού γινομένου (inner/dot product) και Ευκλείδειας απόστασης, για ένα σύστημα επεξεργασίας φυσικής γλώσσας (Ma et al., 2023). Επιπλέον καθώς ο διανυσματικός χώρος που δημιουργείται από την αποθήκευση των πυκνών διανυσμάτων των δεδομένων τείνει να αυξάνεται σε μεγάλο βαθμό, η Qdrant για την πραγματοποίηση αναζήτησης ομοιότητας (similarity search) χρησιμοποιεί τον αλγόριθμο Hierarchical Navigable Small World (HNSW),

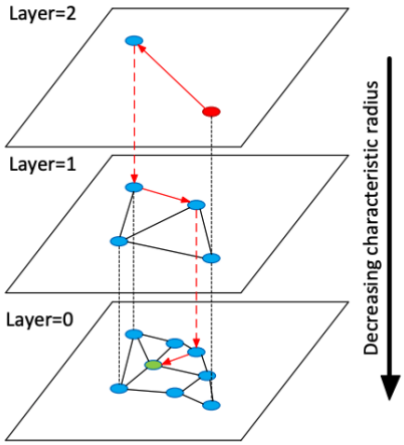
όποιος αποτελεί τύπο αλγορίθμου ANNS (Approximate Nearest Neighbor Search) (Ma et al., 2023).

Hierarchical Navigable Small World (HNSW):

Ο συγκεκριμένος αλγόριθμος αποτελεί μία αποδοτική μέθοδο προσεγγιστικής αναζήτησης πλησιέστερων γειτόνων, η οποία βασίζεται σε μία ιεραρχική δομή γράφου πολλαπλών επιπέδων. Η δομή αυτή αποτελείται από διαδοχικά επίπεδα (layers) γειτνίασης, στα οποία τα ανώτερα επίπεδα είναι αραιότερα και περιλαμβάνουν συνδέσεις μεγαλύτερης κλίμακας, ενώ τα κατώτερα επίπεδα είναι πυκνότερα και αποτυπώνουν πιο τοπικές σχέσεις μεταξύ των διανυσμάτων (Malkov & Yashunin, 2020).

Η αναζήτηση στον HNSW ξεκινά από ένα σημείο εισόδου στο ανώτερο επίπεδο και εκτελείται με άπληστη (greedy) πλοήγηση, δηλαδή με επαναλαμβανόμενη μετάβαση προς τον γειτονικό κόμβο που ελαχιστοποιεί την απόσταση από το ερώτημα. Όταν η διαδικασία εντοπίσει ένα τοπικό ελάχιστο σε ένα επίπεδο, συνεχίζει στο αμέσως κατώτερο επίπεδο, χρησιμοποιώντας ως νέο σημείο εκκίνησης τον καλύτερο κόμβο που βρέθηκε προηγουμένως. Η ίδια διαδικασία επαναλαμβάνεται μέχρι το επίπεδο 0, όπου πραγματοποιείται η πιο λεπτομερής αναζήτηση.

Η βασική σχεδιαστική αρχή του HNSW είναι ο διαχωρισμός των συνδέσεων ανά κλίμακα απόστασης, ώστε τα ανώτερα επίπεδα (layers) να επιτρέπουν γρήγορη προσέγγιση της σχετικής περιοχής του χώρου αναζήτησης και τα κατώτερα να υποστηρίζουν ακριβέστερη τοπική βελτιστοποίηση (Εικόνα 6). Με αυτόν τον τρόπο, ο αλγόριθμος επιτυγχάνει σημαντική επιτάχυνση σε σχέση με την εξαντλητική αναζήτηση, διατηρώντας παράλληλα υψηλή ποιότητα ανάκτησης. Η ιεραρχική αυτή οργάνωση επιτρέπει στο HNSW να παρουσιάζει περίπου λογαριθμική κλιμάκωση της αναζήτησης ως προς το μέγεθος του συνόλου δεδομένων, γεγονός που το καθιστά ιδιαίτερα κατάλληλο για μεγάλες συλλογές διανυσματικών αναπαραστάσεων.



Εικόνα 6 Σχηματική Απεικόνιση της Ιεραρχικής Δομής του HNSW

Για την ανάκτηση των αραιών διανυσμάτων (sparse retrieval), η προτεινόμενη υλοποίηση αξιοποιεί λεξική αναζήτηση βασισμένη στο BM25, ένα από τα πλέον καθιερωμένα μοντέλα ανάκτησης πληροφορίας.

Best Matching 25 (BM25):

Σε αντίθεση με τις πυκνές αναπαραστάσεις, οι οποίες βασίζονται σε σημασιολογική εγγύτητα στον διανυσματικό χώρο, ο αλγόριθμος BM25 στηρίζεται στην ακριβέστερη αντιστοίχιση όρων του ερωτήματος με τους όρους που εμφανίζονται στα αποθηκευμένα τεκμήρια. Για τον λόγο αυτό θεωρείται ιδιαίτερα χρήσιμο σε περιπτώσεις όπου η ακριβής παρουσία συγκεκριμένων λέξεων-κλειδιών, τεχνικών όρων ή συμβολοσειρών έχει αυξημένη σημασία για την ποιότητα της ανάκτησης (Singh et al., 2025).

Η λειτουργία του BM25 βασίζεται στη συχνότητα εμφάνισης ενός όρου μέσα σε ένα τεκμήριο, καθώς και στη σπανιότητά του στο σύνολο της συλλογής. Με άλλα λόγια, ένας όρος αποκτά μεγαλύτερη βαρύτητα όταν εμφανίζεται αρκετά συχνά σε ένα συγκεκριμένο τεκμήριο, αλλά ταυτόχρονα δεν είναι ιδιαίτερα κοινός σε ολόκληρο το σώμα δεδομένων. Παράλληλα, το BM25 ενσωματώνει μηχανισμό κανονικοποίησης ως προς το μήκος του τεκμηρίου, ώστε να μην ευνοούνται αδικαιολόγητα τα μεγαλύτερα κείμενα μόνο και μόνο επειδή περιέχουν περισσότερες εμφανίσεις των ίδιων όρων.

Στο πλαίσιο του προτεινόμενου συστήματος, η χρήση του BM25 είναι ιδιαίτερα σημαντική επειδή επιτρέπει την ανάδειξη τεκμηρίων που περιέχουν ακριβείς λεκτικές ενδείξεις σχετικές με επιθέσεις SQLi, όπως συγκεκριμένους όρους, παραμέτρους, μοτίβα ή συμβολοσειρές που

ενδέχεται να μην αποτυπώνονται εξίσου αποτελεσματικά μόνο μέσω της σημασιολογικής ανάκτησης.

Στην περίπτωση της υβριδικής διανυσματικής ανάκτησης, κατόπιν της πυκνής ανάκτησης με τη βοήθεια του HNSW, καθώς και της αραιής ανάκτησης με την αξιοποίηση της λεξικής αντιστοίχισης όρων από την βάση δεδομένων, το δύσκολο είναι να συνδυαστούν σε ένα αποτέλεσμα, οι διαφορετικές αναπαραστάσεις των ίδιων δεδομένων (*Hybrid Queries - Qdrant*, n.d.). Η Qdrant διαθέτει διάφορους τρόπους συνδυασμού των αποτελεσμάτων (fusion), όπως ο Reciprocal Rank Fusion (RRF), Weighted RRF και Distribution-Based Score Fusion (DBSF). Στο προτεινόμενο σύστημα χρησιμοποιείται ο RRF.

Reciprocal Rank Fusion (RRF):

Ο Reciprocal Rank Fusion (RRF) είναι ένας αλγόριθμος συνδυασμού κατατάξεων (rank fusion), ο οποίος προτάθηκε από τους Cormack et al. (2009) και χρησιμοποιείται για την ενοποίηση των αποτελεσμάτων πολλαπλών ανεξάρτητων μεθόδων ανάκτησης σε μία ενιαία κατάταξη (Jeong, 2026).

Η βασική ιδέα του αλγορίθμου είναι ότι δεν συνδυάζει τις αρχικές βαθμολογίες των επιμέρους retrievers, αλλά τις θέσεις κατάταξης των εγγράφων σε κάθε λίστα αποτελεσμάτων, ώστε να παράγεται μία τελική ενιαία κατάταξη. Ο υπολογισμός του RRF score για ένα έγγραφο d δίνεται από τον τύπο:

$$Score(d) = \sum_{r \in R} \frac{1}{k + rank_r(d) + 1}$$

όπου R είναι το σύνολο των επιμέρους retrievers, $rank_r(d)$ είναι η θέση του εγγράφου d στη λίστα αποτελεσμάτων του retriever r, και k είναι μια σταθερά εξομάλυνσης που μειώνει την επίδραση πολύ μικρών διαφορών μεταξύ των κορυφαίων θέσεων.

Στην πράξη, ο RRF αποδίδει μεγαλύτερη βαθμολογία στα έγγραφα που εμφανίζονται ψηλά σε περισσότερες από μία λίστες αποτελεσμάτων. Έτσι, ευνοούνται τα έγγραφα που παρουσιάζουν σταθερά καλή απόδοση σε διαφορετικούς μηχανισμούς ανάκτησης, αντί να επιλέγονται μόνο όσα είχαν την υψηλότερη βαθμολογία σε μία μόνο μέθοδο. Στο πλαίσιο της υβριδικής ανάκτησης που πραγματοποιείται στη Qdrant, αυτό σημαίνει ότι ένα έγγραφο μπορεί να αναδειχθεί ψηλά στην τελική κατάταξη είτε επειδή είναι σημασιολογικά σχετικό μέσω πυκνής ανάκτησης, είτε επειδή ταιριάζει λεξικά μέσω αραιής, αλλά κυρίως όταν εμφανίζεται ως

σημαντικό και στις δύο προσεγγίσεις. Με αυτόν τον τρόπο, ο RRF συμβάλλει σε πιο ισορροπημένη και αξιόπιστη συγχώνευση αποτελεσμάτων για το σύστημα.

Τέλος, στην ενιαία λίστα που προκύπτει από την εφαρμογή του RRF, εφαρμόζεται ο αλγόριθμος MMR, προκειμένου να διασφαλιστεί η ποικιλομορφία της ανακτώμενης πληροφορίας.

Maximal Marginal Relevance (MMR)

Ο αλγόριθμος Maximal Marginal Relevance (MMR) αποτελεί κριτήριο επανακατάταξης (reranking), το οποίο εισήχθη από τους Carbonell και Goldstein με στόχο να συνδυάσει τη συνάφεια προς το ερώτημα με τη διαφορετικότητα της πληροφορίας που επιστρέφεται στον χρήστη. Η βασική ιδέα του αλγορίθμου είναι ότι ένα αποτέλεσμα δεν πρέπει να επιλέγεται μόνο επειδή είναι πολύ σχετικό με το ερώτημα, αλλά και επειδή προσφέρει νέα πληροφορία σε σχέση με τα αποτελέσματα που έχουν ήδη επιλεγεί. Με τον τρόπο αυτό, ο MMR επιδιώκει να μειώσει τον πλεονασμό και να αυξήσει την ποικιλία των ανακτώμενων τεκμηρίων ή αποσπασμάτων (Carbonell, 1999).

Πιο συγκεκριμένα, ο MMR λειτουργεί επαναληπτικά επιλέγοντας κάθε φορά το τεκμήριο που επιτυγχάνει τον καλύτερο συμβιβασμό μεταξύ δύο κριτηρίων, αφενός της ομοιότητάς του με το αρχικό ερώτημα και αφετέρου της ανομοιότητάς του, ως προς τα ήδη επιλεγμένα αποτελέσματα. Η ισορροπία μεταξύ των δύο αυτών συνιστωσών ελέγχεται από μια παράμετρο λ, η οποία καθορίζει αν το σύστημα θα δώσει μεγαλύτερη έμφαση στη συνάφεια ή στη διαφορετικότητα. Όταν η τιμή της παραμέτρου αυξάνεται, το σύστημα ευνοεί περισσότερο τα άμεσα σχετικά αποτελέσματα, ενώ όταν μειώνεται, δίνεται μεγαλύτερη βαρύτητα στην αποφυγή επαναλαμβανόμενης πληροφορίας.

Στο προτεινόμενο σύστημα, ο MMR μπορεί να αξιοποιηθεί μετά την αρχική ανάκτηση και τη συγχώνευση των αποτελεσμάτων, ώστε η τελική επιλογή των chunks που θα προωθηθούν στο γλωσσικό μοντέλο να μην περιέχει έντονα αλληλεπικαλυπτόμενη πληροφορία. Αυτό είναι ιδιαίτερα σημαντικό σε περιβάλλοντα RAG, όπου πολλά από τα κορυφαία αποτελέσματα μπορεί να είναι πολύ κοντινά μεταξύ τους, είτε επειδή προέρχονται από παρόμοια τμήματα κειμένου είτε επειδή αναφέρονται επανειλημμένα στο ίδιο συμβάν. Επομένως, ο MMR δεν αντικαθιστά την αρχική ανάκτηση ή τη διαδικασία fusion, αλλά λειτουργεί ως

συμπληρωματικός μηχανισμός βελτίωσης της τελικής λίστας αποτελεσμάτων, ενισχύοντας την κάλυψη πληροφορίας και περιορίζοντας την περιττή επανάληψη.

4.4.2 Γραφική Βάση Δεδομένων Neo4j

Η Neo4j χρησιμοποιείται στο προτεινόμενο σύστημα ως η γραφική βάση στην οποία αποθηκεύεται η δομημένη γνώση που προκύπτει από την επεξεργασία των αρχείων καταγραφής δικτυακής κίνησης. Πιο συγκεκριμένα, τα PCAP αρχεία δεν αποθηκεύονται αυτούσια ως αδόμητα τεκμήρια, αλλά αναλύονται και από αυτά εξάγονται κρίσιμα στοιχεία των HTTP επικοινωνιών, τα οποία στη συνέχεια μετασχηματίζονται σε κόμβους και σχέσεις ενός γράφου γνώσης. Με τον τρόπο αυτό, η πληροφορία δεν παραμένει σε επίπεδο απλής καταγραφής πακέτων, αλλά αποκτά δομημένη μορφή που επιτρέπει πλοήγηση, συσχέτιση και σημασιολογική ερμηνεία των επιμέρους συμβάντων.

Στην παρούσα υλοποίηση, ο όρος «γράφος γνώσης» αναφέρεται σε έναν **domain-specific property graph** στη Neo4j, ο οποίος μοντελοποιεί τα HTTP και δικτυακά artifacts που εξάγονται από τα αρχεία PCAP. Η υλοποίηση δεν αποτελεί οντολογικό γράφο RDF/OWL και δεν αξιοποιεί μηχανισμό τυπικού σημασιολογικού συλλογισμού. Ο συγκεκριμένος γράφος οργανώνεται γύρω από τους βασικούς τύπους κόμβων DAY, PCAP_FILE, HTTP_REQUEST, IP, HOST και PARAM (Εικόνα 7), ώστε να αποτυπώνεται τόσο η προέλευση κάθε καταγραφής όσο και τα επιμέρους γνωρίσματα των αιτημάτων HTTP που εξάγονται από τα αρχεία PCAP. Η επιλογή αυτού του σχήματος επιτρέπει στο σύστημα να συνδέει το χρονικό πλαίσιο εκτέλεσης του αιτήματος, το αρχείο προέλευσης, το ίδιο το αίτημα HTTP, τις εμπλεκόμενες διευθύνσεις δικτύου, τον στόχο του αιτήματος και τις παραμέτρους που μεταφέρονται μέσω URI, query string ή body.

Πιο συγκεκριμένα, για κάθε ημερομηνία δημιουργείται ένας κόμβος DAY με μοναδική ιδιότητα «date», ενώ για κάθε αρχείο καταγραφής δημιουργείται ένας κόμβος PCAP_FILE με μοναδική ιδιότητα «name». Κάθε αρχείο συνδέεται με την αντίστοιχη ημερομηνία μέσω της σχέσης HAS_PCAP, ώστε να διατηρείται η ομαδοποίηση των δεδομένων σε επίπεδο ημέρας και πηγής συλλογής. Επιπλέον, στον κόμβο του αρχείου αποθηκεύεται και η ημερομηνία του PCAP, καθώς και χρονική πληροφορία δημιουργίας μέσω της ιδιότητας «created_at», ώστε να υποστηρίζεται η ιχνηλασιμότητα της διαδικασίας εισαγωγής.

Ο βασικός πυρήνας του γράφου είναι ο κόμβος HTTP_REQUEST, ο οποίος αναπαριστά κάθε μεμονωμένο αίτημα HTTP που εξάγεται από τα αρχεία PCAP. Κάθε τέτοιος κόμβος ταυτοποιείται μοναδικά με την ιδιότητα «request_id» και συνοδεύεται από ένα σύνολο ιδιοτήτων που περιγράφουν λειτουργικά και δικτυακά χαρακτηριστικά του αιτήματος, όπως «timestamp», «method», «uri», «full_uri», «query_string», «user_agent», «referer», «cookie», «raw_body», «src_port» και «dst_port». Με αυτόν τον τρόπο, κάθε αίτημα HTTP αποθηκεύεται ως αυτόνομη οντότητα του γράφου, στην οποία συγκεντρώνεται τόσο η πληροφορία επιπέδου εφαρμογής όσο και κρίσιμα στοιχεία της συνολικής επικοινωνίας.

Η σύνδεση μεταξύ του αρχείου προέλευσης και των επιμέρους αιτημάτων HTTP πραγματοποιείται μέσω της σχέσης CONTAINS, η οποία δηλώνει ότι ένα συγκεκριμένο PCAP_FILE περιέχει ένα συγκεκριμένο HTTP_REQUEST. Η σχέση αυτή είναι σημαντική, διότι επιτρέπει την αναδρομή από το επίπεδο του επιμέρους αιτήματος, πίσω στο αρχείο συλλογής από το οποίο προήλθε. Έτσι, καθίσταται δυνατή όχι μόνο η αναλυτική εξέταση ενός μεμονωμένου αιτήματος, αλλά και η ένταξή του στο ευρύτερο αποδεικτικό και χρονικό πλαίσιο του αρχείου PCAP από το οποίο εξήχθη.

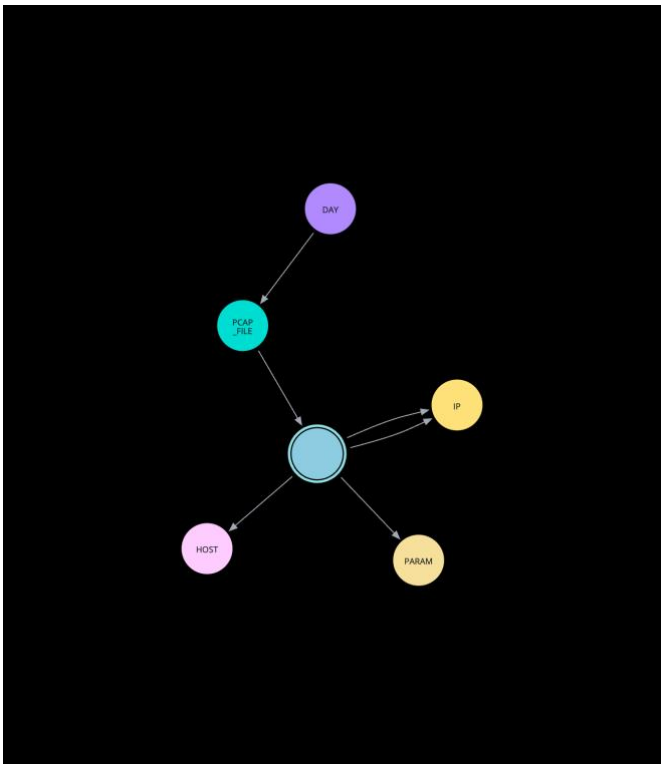
Παράλληλα, για τις διευθύνσεις δικτύου δημιουργούνται κόμβοι τύπου IP, οι οποίοι φέρουν ως μοναδική ιδιότητα τη διεύθυνση «address». Κάθε HTTP_REQUEST μπορεί να συνδέεται με έναν κόμβο πηγής μέσω της σχέσης FROM_IP και με έναν κόμβο προορισμού μέσω της σχέσης TO_IP, εφόσον οι αντίστοιχες τιμές «src_ip» και «dst_ip» είναι διαθέσιμες στα εξαγόμενα δεδομένα. Με την αναπαράσταση αυτή, η πληροφορία του δικτυακού επιπέδου αποσυνδέεται από το ίδιο το αίτημα ως απλό πεδίο και μετατρέπεται σε ξεχωριστή οντότητα του γράφου, γεγονός που επιτρέπει την εύρεση κοινών μοτίβων, επαναλαμβανόμενων επικοινωνιών και συσχετίσεων μεταξύ πολλαπλών αιτημάτων που εμπλέκουν τις ίδιες IP διευθύνσεις.

Αντίστοιχα, για το πεδίο προορισμού δημιουργούνται κόμβοι τύπου HOST, με μοναδική ιδιότητα την τιμή «value», και κάθε αίτημα συνδέεται με τον αντίστοιχο «host» μέσω της σχέσης TARGETS_HOST. Η μοντελοποίηση αυτή επιτρέπει την αποτύπωση του λογικού στόχου του αιτήματος HTTP ως διακριτής οντότητας, ανεξάρτητης από τη φυσική δικτυακή διεύθυνση. Ως αποτέλεσμα, καθίσταται ευκολότερη η ομαδοποίηση πολλών αιτημάτων που στοχεύουν στον ίδιο «host», ακόμα και όταν αυτά εμφανίζονται σε διαφορετικά αρχεία PCAP ή χρονικές στιγμές.

Ιδιαίτερα κρίσιμη είναι και η αποθήκευση των παραμέτρων κάθε αιτήματος μέσω κόμβων τύπου PARAM. Κάθε παράμετρος χαρακτηρίζεται από τα πεδία «request_id», «key», «value» και «location», όπου το «location» δηλώνει το σημείο εμφάνισής της, όπως query string ή body. Ο κόμβος της παραμέτρου συνδέεται με το αντίστοιχο αίτημα μέσω της σχέσης HAS_PARAM, επιτρέποντας την άμεση ανάκτηση όλων των παραμέτρων που σχετίζονται με ένα αίτημα αλλά και την αναζήτηση παραμέτρων με συγκεκριμένο όνομα, τιμή ή θέση εμφάνισης σε ολόκληρο το γράφημα.

Για τη διασφάλιση της ακεραιότητας του γράφου, η υλοποίηση ορίζει ένα σύνολο από περιορισμούς μοναδικότητας πριν από την εισαγωγή των δεδομένων. Συγκεκριμένα, επιβάλλεται μοναδικότητα στα πεδία «date» για τους κόμβους DAY, «name» για τους κόμβους PCAP_FILE, «address» για τους κόμβους IP, «value» για τους κόμβους HOST και «request_id» για τους κόμβους HTTP_REQUEST, ενώ για τους κόμβους PARAM ορίζεται σύνθετος περιορισμός μοναδικότητας στα πεδία «request_id», «key», «value» και «location». Η πρακτική αυτή αποτρέπει τη δημιουργία διπλότυπων οντοτήτων και εξασφαλίζει ότι η επαναλαμβανόμενη εισαγωγή δεδομένων δεν αλλοιώνει τη συνοχή του γραφήματος.

Η διαδικασία εισαγωγής υλοποιείται με χρήση της εντολής MERGE, ώστε κάθε κόμβος και κάθε σχέση να δημιουργείται μόνο όταν δεν υπάρχει ήδη στη βάση. Έτσι, το σύστημα επιτυγχάνει ταυτοδύναμη εισαγωγή δεδομένων (idempotent ingestion), δηλαδή μπορεί να επανεκτελεί τη διαδικασία φόρτωσης χωρίς να παράγει πλεονάζουσες εγγραφές. Η επιλογή αυτή είναι ιδιαίτερα κατάλληλη σε συστήματα ανάλυσης καταγραφών, όπου είναι πιθανό να απαιτηθεί επανεπεξεργασία ενός PCAP ή επαναφόρτωση ενός συνόλου δεδομένων μετά από αλλαγές στη ροή προεπεξεργασίας.



Εικόνα 7 Σχήμα Neo4j Graph DB

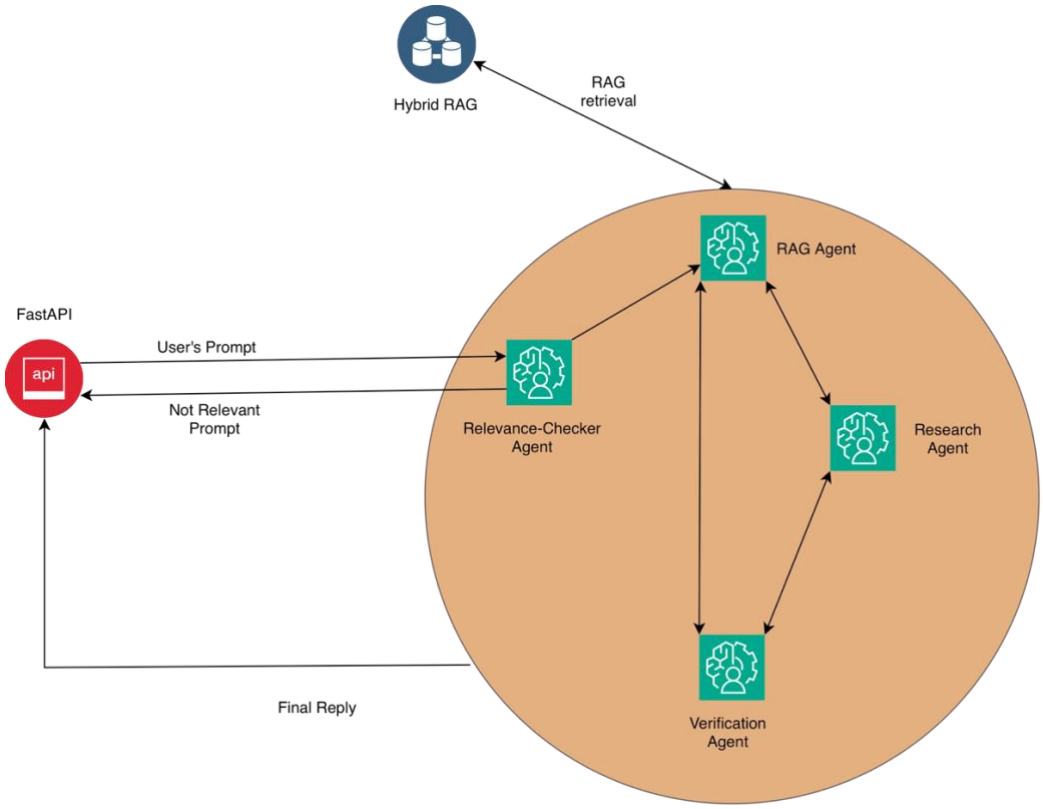
4.5 Υποσύστημα Agentic RAG (Multi Agent System - MAS) – 4ο Επίπεδο (4th Layer)

Η αρχιτεκτονική του συγκεκριμένου υποσυστήματος είναι ένα Agentic RAG σύστημα πολλαπλών agents, το οποίο προσφέρει όλα τα πλεονεκτήματα που αναφέρθηκαν στο προηγούμενο κεφάλαιο για τα συστήματα που ακολουθούν την συγκεκριμένη αρχιτεκτονική. Για την παρούσα υλοποίηση, η συγκεκριμένη αρχιτεκτονική εισάγει τέσσερις διαφορετικούς agents, οι οποίοι βοηθούν στην αξιολόγηση του ερωτήματος του χρήστη, δηλαδή το πόσο σχετικό είναι με τις επιθέσεις SQLi, στη χρήση εργαλείων αναζήτησης της υβριδικής υποδομής γνώσης, καθώς και στον έλεγχο της κάθε απάντησης προκειμένου να αποφευχθούν περιπτώσεις «παραισθήσεων» (hallucinations).

4.5.1 Αρχιτεκτονική Συστήματος Πολλαπλών Agents

Το συγκεκριμένο υποσύστημα (Εικόνα 8) αποτελείται από τέσσερις διαφορετικούς agents, και συγκεκριμένα τον Relevance Checker agent, τον RAG agent, τον Research agent και τον Verification agent. Ο Relevance Checker είναι ο agent που δέχεται τα ερωτήματα του χρήστη μέσω του FastAPI και πραγματοποιεί μία αρχική εκτίμηση αν το ερώτημα είναι σχετικό με το θέμα των SQLi επιθέσεων. Εφόσον το ερώτημα είναι έγκυρο, προωθείται και οι υπόλοιποι agents αναλαμβάνουν την ανάκτηση επιπλέον πληροφορίας από την υβριδική υποδομή γνώσης, την αρχική απάντηση του ερωτήματος και την αξιολόγηση της ως προς την πληρότητα και εγκυρότητα. Προκειμένου οι agents να εκπληρώσουν το έργο τους, αξιοποιούν το γλωσσικό μοντέλο Deepseek-V3 μέσω API κλήσεων που γίνονται προς τον δρομολογητή του Hugging Face (https://router.huggingface.co/v1).

Επιπλέον, στο προτεινόμενο σύστημα, έχει χρησιμοποιηθεί το LangChain ως το βασικό πλαίσιο διασύνδεσης του Agentic υποσυστήματος με τα υπόλοιπα δομικά μέρη της εφαρμογής. Το LangChain αποτελεί ένα open-source framework το οποίο απλοποιεί την ανάπτυξη εφαρμογών με μεγάλα γλωσσικά μοντέλα (LLMs) και πιο συγκεκριμένα χρησιμοποιήθηκε για την ενοποίηση της αλληλεπίδρασης μεταξύ των agents, του γλωσσικού μοντέλου, της διανυσματικής βάσης, της γραφικής βάσης και των επιμέρους μηχανισμών ανάκτησης, επιτρέποντας την υλοποίηση μίας συνεκτικής ροής επεξεργασίας από το αρχικό ερώτημα έως την τελική παραγωγή απάντησης. Στο πλαίσιο αυτό, το LangChain δεν λειτουργεί απλώς ως βιβλιοθήκη κλήσεων προς εξωτερικά συστήματα, αλλά ως ενδιάμεσο επίπεδο αφαίρεσης που διευκολύνει τον συντονισμό εργαλείων, τη διαχείριση της ροής πληροφορίας και τον καθορισμό των επιμέρους ρόλων των agents. Η χρήση του κρίνεται ιδιαίτερα σημαντική, καθώς επιτρέπει την αρθρωτή ανάπτυξη του συστήματος, την ευέλικτη ενσωμάτωση ετερογενών πηγών γνώσης και την αποδοτική υποστήριξη σύνθετων Agentic ροών εργασίας, μέσα στις οποίες οι επιμέρους agents μπορούν να συνεργάζονται για την ανάκτηση, αξιολόγηση, συγχώνευση και αξιοποίηση της πληροφορίας με συστηματικό και επεκτάσιμο τρόπο.



Εικόνα 8 Αρχιτεκτονική Multi-Agent Sub-System

Παρακάτω αναλύεται εκτενέστερα ο ρόλος που επιτελεί ο κάθε agent στη λειτουργία του συνολικού συστήματος, προκειμένου στη συνέχεια να γίνει κατανοητό το πώς πραγματοποιείται η ενορχήστρωσή τους, καθώς και τη ροή που ακολουθείται προκειμένου να απαντηθούν επιτυχώς τα ερωτήματα του χρήστη.

4.5.2 Relevance Checker Agent

Ο Relevance Checker agent αποτελεί τον αρχικό μηχανισμό αξιολόγησης των ερωτημάτων εισόδου στο συγκεκριμένο υποσύστημα. Αποστολή του είναι να εξετάζει εάν το ερώτημα του χρήστη εμπίπτει στο γνωστικό και λειτουργικό πεδίο της εφαρμογής, το οποίο αφορά την ανάλυση επιθέσεων SQLi, τη διερεύνηση ύποπτης δικτυακής κίνησης και κίνησης επιπέδου εφαρμογής, καθώς και την αξιοποίηση των δεδομένων που έχουν εξαχθεί από τα εισαγμένα αρχεία PCAP. Η αξιολόγηση πραγματοποιείται αποκλειστικά ως προς τη θεματική συνάφεια του ερωτήματος και δεν εξαρτάται από το εάν έχουν ήδη ανακτηθεί επαρκή δεδομένα ή τεκμήρια για την παραγωγή απάντησης. Κατ' αυτόν τον τρόπο, ο agent διαχωρίζει τον έλεγχο

καταλληλότητας του ερωτήματος από τα επόμενα στάδια ανάκτησης, ανάλυσης και τεκμηρίωσης.

Η απόφασή του αποτυπώνεται σε τρεις κατηγορίες: **RELEVANT**, όταν το ερώτημα σχετίζεται σαφώς με SQLi, ύποπτα αιτήματα, payloads, παραμέτρους, κίνηση HTTP ή στοιχεία του εισαγμένου συνόλου δεδομένων, **PARTIALLY_RELEVANT** όταν υφίσταται έμμεση ή ασθενής σύνδεση με κυβερνοασφάλεια, δικτυακή κίνηση ή ύποπτη συμπεριφορά, χωρίς ρητή αναφορά σε SQL injection και **IRRELEVANT**, όταν το περιεχόμενο βρίσκεται εκτός του πεδίου της εφαρμογής. Η ύπαρξη της ενδιάμεσης κατηγορίας επιτρέπει την ανοχή σε ασαφή ή γενικά διατυπωμένα ερωτήματα, τα οποία ενδέχεται να είναι χρήσιμα για διερεύνηση, χωρίς να απαιτείται η χρήση αυστηρής και εξειδικευμένης ορολογίας SQLi. Προκειμένου ο agent να κρίνει αποτελεσματικά τη συνάφεια του ερωτήματος, πραγματοποιεί κλήση προς το API του γλωσσικού μοντέλου στέλνοντας το ερώτημα του χρήστη μαζί με επιπλέον οδηγίες (prompt template), οι οποίες κατευθύνουν το LLM ως προς τον τρόπο που πρέπει να κρίνει το ερώτημα και ως προς την επιστρεφόμενη τιμή.

Συνεπώς, ο Relevance Checker agent λειτουργεί ως φίλτρο εισόδου που προστατεύει τη συνοχή της ροής επεξεργασίας, αποτρέπει την ενεργοποίηση των επόμενων agents για προφανώς εκτός πεδίου αιτήματα και περιορίζει την άσκοπη κατανάλωση υπολογιστικών πόρων.

4.5.3 RAG Agent

Ο RAG agent αποτελεί τον μοναδικό agent του υποσυστήματος που διαθέτει άμεση πρόσβαση και δυνατότητα αξιοποίησης των εργαλείων ανάκτησης γνώσης του συστήματος. Ο ρόλος του περιορίζεται στην επιλογή και ενεργοποίηση της κατάλληλης στρατηγικής ανάκτησης, χωρίς να παράγει ο ίδιος την τελική απάντηση προς τον χρήστη. Με βάση το περιεχόμενο και τη φύση του εισερχόμενου ερωτήματος, αξιολογεί ποια μορφή τεκμηρίωσης είναι καταλληλότερη, ώστε να συγκεντρωθούν τα πιο σχετικά και αξιόπιστα στοιχεία για τα επόμενα στάδια επεξεργασίας. Επομένως, λειτουργεί ως ο συνδετικός κρίκος μεταξύ του Agentic επιπέδου και των υποσυστημάτων αποθήκευσης και ανάκτησης γνώσης, ενεργοποιώντας με ελεγχόμενο τρόπο τους διαθέσιμους μηχανισμούς πρόσβασης στα δεδομένα.

Ειδικότερα, ο RAG agent μπορεί να χρησιμοποιήσει τρία εργαλεία ανάκτησης, το **vector_search_tool**, το **kg_search_tool** και το **hybrid_search_tool**. Το πρώτο επιλέγεται για

ερωτήματα που απαιτούν σημασιολογική ανάκτηση από τα τεκμήρια και τα τμηματοποιημένα κείμενα (chunks) της διανυσματικής βάσης, όπως ερωτήματα για έννοιες, τεχνικές, μοτίβα και μηχανισμούς επιθέσεων SQL injection. Το δεύτερο χρησιμοποιείται όταν το ερώτημα αφορά στοιχεία που έχουν παρατηρηθεί στα εισαγμένα δεδομένα δικτυακής κίνησης, όπως ημερομηνίες, διευθύνσεις IP, hosts, αιτήματα HTTP, παραμέτρους και σχέσεις μεταξύ οντοτήτων, αξιοποιώντας τη δομή του γράφου γνώσης στη Neo4j. Τέλος, το hybrid_search_tool επιλέγεται όταν απαιτείται συνδυασμός σημασιολογικής γνώσης και γραφικής τεκμηρίωσης, ιδίως σε ερωτήματα που διερευνούν την πιθανότητα εμφάνισης δραστηριότητας SQL injection σε συγκεκριμένο χρονικό διάστημα ή σύνολο δικτυακών δεδομένων, που απαιτούν παράλληλα ερμηνεία των ευρημάτων. Με αυτόν τον μηχανισμό δυναμικής επιλογής εργαλείων, ο RAG agent διασφαλίζει ότι η συλλογή τεκμηρίων προσαρμόζεται στις απαιτήσεις κάθε ερωτήματος, χωρίς να παράγεται ή να συμπληρώνεται μη τεκμηριωμένη πληροφορία.

4.5.4 Research Agent

Ο Research agent είναι υπεύθυνος για τη μετατροπή της ανακτημένης πληροφορίας σε μία αρχική, συνεκτική και τεκμηριωμένη απάντηση. Αξιοποιεί το αρχικό ερώτημα, το πρόσφατο ιστορικό της συνομιλίας και τα στοιχεία που έχουν συγκεντρωθεί από τον RAG agent, προκειμένου να συνθέσει ένα πρώτο προσχέδιο απάντησης. Η λειτουργία του βασίζεται στην αρχή της τεκμηριωμένης παραγωγής, καθώς το περιεχόμενο της απάντησης πρέπει να προκύπτει αποκλειστικά από τα διαθέσιμα ανακτημένα δεδομένα, χωρίς προσθήκη μη επαληθευμένων πληροφοριών ή συμπερασμάτων που δεν υποστηρίζονται από τα ευρήματα. Ιδιαίτερη έμφαση δίνεται στην ερμηνεία των στοιχείων από πλευράς κυβερνοασφάλειας και όπου είναι δυνατόν, στην αξιολόγηση ενδείξεων που μπορεί να σχετίζονται με δραστηριότητα SQL injection.

Πέρα από την παραγωγή του αρχικού προσχεδίου, ο Research agent αναλαμβάνει και την αναθεώρηση μιας ήδη παραχθείσας απάντησης όταν αυτή αξιολογηθεί από το στάδιο επαλήθευσης. Στη λειτουργία αναθεώρησης λαμβάνει υπόψη το ερώτημα του χρήστη, την ανακτημένη τεκμηρίωση (RAG retrieval), την τρέχουσα μορφή της απάντησης και την έκθεση επαλήθευσης (verification report), με στόχο να αφαιρέσει μη υποστηριζόμενους ή αντιφατικούς ισχυρισμούς και να βελτιώσει την ακρίβεια και τη σαφήνεια του τελικού

κειμένου. Επομένως, ο Research agent λειτουργεί ως ο βασικός μηχανισμός σύνθεσης και βελτίωσης της απάντησης, διατηρώντας σαφή διαχωρισμό από τη διαδικασία ανάκτησης δεδομένων και από τις αποφάσεις ελέγχου του υποσυστήματος.

4.5.5 Verification Agent

Ο Verification agent αποτελεί τον μηχανισμό ελέγχου ποιότητας και τεκμηρίωσης των απαντήσεων που παράγονται από το υποσύστημα. Δέχεται ως είσοδο το αρχικό ερώτημα του χρήστη, το προσχέδιο απάντησης που έχει συνταχθεί από τον Research agent και το διαθέσιμο ανακτημένο πλαίσιο (RAG retrieval), με σκοπό να αξιολογήσει κατά πόσο το περιεχόμενο της απάντησης υποστηρίζεται από τα αντίστοιχα στοιχεία. Η λειτουργία του επικεντρώνεται στον εντοπισμό άμεσης ή έμμεσης τεκμηρίωσης των ισχυρισμών, στην αναγνώριση πληροφοριών που δεν προκύπτουν από τα διαθέσιμα δεδομένα και στον έλεγχο πιθανών αντιφάσεων μεταξύ της απάντησης και του ανακτημένου πλαισίου. Με αυτόν τον τρόπο, λειτουργεί ως ενδιάμεσο στάδιο αξιοπιστίας πριν από την παράδοση της απάντησης στον χρήστη.

Το αποτέλεσμα της αξιολόγησης αποτυπώνεται σε ένα δομημένο **verification report**, το οποίο περιλαμβάνει ένδειξη υποστήριξης της απάντησης, κατάλογο μη τεκμηριωμένων ισχυρισμών, ενδεχόμενες αντιφάσεις και προτεινόμενη επόμενη ενέργεια της ροής. Η ενέργεια αυτή μπορεί να είναι η οριστικοποίηση της απάντησης (finalize), η αναθεώρησή της (revise_answer) ή η ανάκτηση πρόσθετης πληροφορίας (retrieve_more).

Η επιλογή finalize χρησιμοποιείται όταν η απάντηση είναι επαρκώς τεκμηριωμένη, συναφής και δεν απαιτεί ουσιώδεις διορθώσεις. Αντίθετα, η επιλογή revise_answer ενεργοποιείται όταν η απάντηση είναι κατά βάση ορθή, αλλά απαιτεί βελτίωση στη διατύπωση, αποσαφήνιση ή αφαίρεση επιμέρους μη υποστηριζόμενων στοιχείων. Όταν το διαθέσιμο πλαίσιο δεν περιέχει επαρκή τεκμήρια για την παραγωγή αξιόπιστης απάντησης, ο agent προτείνει retrieve_more, συνοδεύοντας την απόφαση με συγκεκριμένες οδηγίες για το είδος της πληροφορίας που πρέπει να ανακτηθεί.

Συνεπώς, ο Verification agent δεν παράγει ούτε συμπληρώνει απευθείας την τελική απάντηση, αλλά υποστηρίζει τον έλεγχο της εγκυρότητας και της πληρότητάς της. Η αναφορά επαλήθευσης που δημιουργεί αξιοποιείται για να καθοριστεί εάν η απάντηση μπορεί να παραδοθεί, εάν πρέπει να επιστρέψει στον Research agent για αναθεώρηση ή εάν απαιτείται νέα διαδικασία ανάκτησης μέσω του RAG agent. Έτσι, ενισχύεται η τεκμηριωμένη λειτουργία

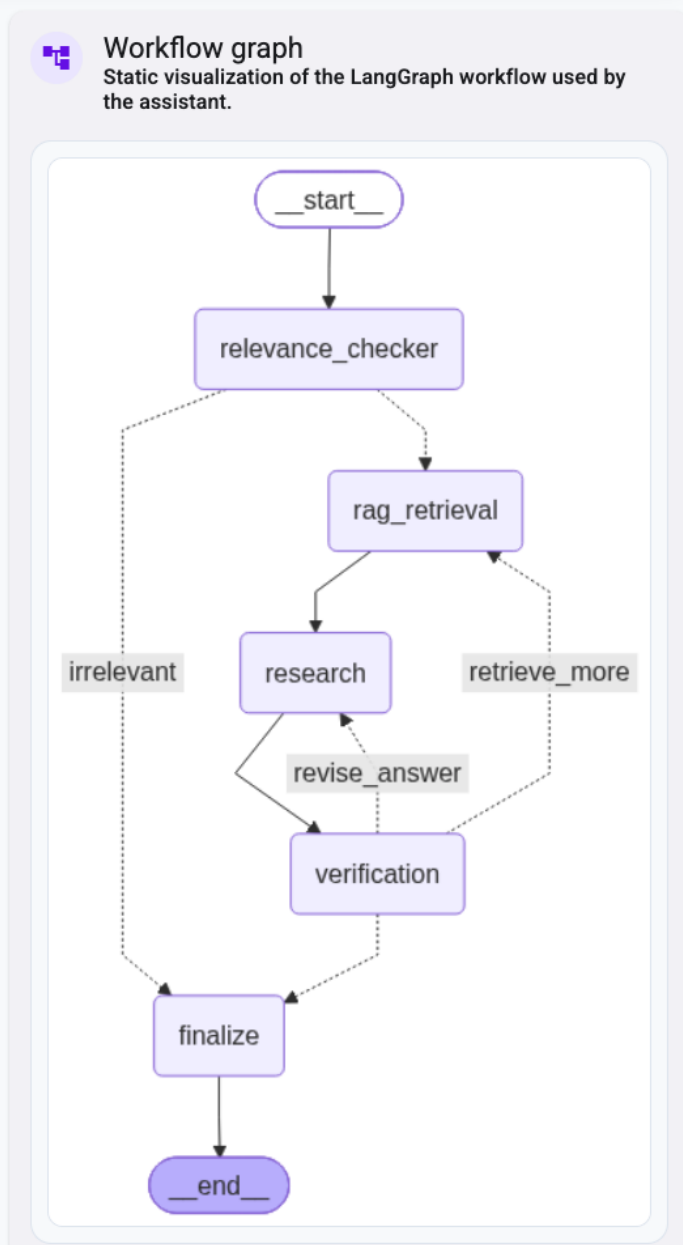
του συστήματος, περιορίζεται ο κίνδυνος παραγωγής μη υποστηριζόμενων συμπερασμάτων και βελτιώνεται η συνολική αξιοπιστία των απαντήσεων που παρέχονται στον χρήστη.

4.5.6 Ενορχήστρωση των Agents

Για την ενορχήστρωση των Agents χρησιμοποιείται το **LangGraph**, ένα open-source framework του οικοσυστήματος LangChain για την κατασκευή εφαρμογών βασισμένων σε ροές εργασίας με τη μορφή γράφων. Η βασική του αρχή είναι ότι οι επιμέρους λειτουργίες αναπαρίστανται ως κόμβοι (*nodes*), ενώ οι μεταβάσεις και οι κανόνες δρομολόγησης μεταξύ τους ορίζονται ως ακμές (*edges*). Η γραφική αυτή προσέγγιση υποστηρίζει τόσο γραμμικές ακολουθίες επεξεργασίας όσο και συνθήκες διακλάδωσης ή επαναληπτικούς βρόχους, καθιστώντας το κατάλληλο για συστήματα πολλαπλών agents στα οποία η πορεία επεξεργασίας δεν είναι πάντοτε προκαθορισμένη. Ειδικότερα, το LangGraph υποστηρίζει μηχανισμούς διαχείρισης κατάστασης, επανάληψης και αυτοδιόρθωσης, οι οποίοι είναι σημαντικοί για την υλοποίηση σύνθετων Agentic RAG ροών.

Στην παρούσα υλοποίηση, οι agents οργανώνονται σε έναν κατευθυνόμενο γράφο με τους κόμβους relevance_checker, rag_retrieval, research, verification και finalize. Κάθε κόμβος αντιστοιχεί σε μία διακριτή λειτουργική φάση, όπως έλεγχος συνάφειας του ερωτήματος, ανάκτηση τεκμηρίων, παραγωγή ή αναθεώρηση προσχεδίου απάντησης, έλεγχος τεκμηρίωσης και τελική παράδοση της απάντησης. Οι ακμές του γράφου καθορίζουν τη βασική διαδοχή των σταδίων, ενώ οι υπό συνθήκη ακμές επιτρέπουν τη μεταβολή της ροής ανάλογα με την απόφαση του Relevance Checker agent ή το αποτέλεσμα του Verification agent. Έτσι, η αρχιτεκτονική δεν περιορίζεται σε μία στατική ακολουθία ανάκτησης και παραγωγής απάντησης, αλλά ενσωματώνει επαναληπτική αξιολόγηση και δυνατότητα επιστροφής σε προηγούμενα στάδια όταν τα διαθέσιμα στοιχεία δεν επαρκούν. Η προσέγγιση αυτή συνάδει με διορθωτικές και αξιολογητικές ροές εργασίας Agentic RAG, όπου η ανάκτηση, η σύνθεση και η επαλήθευση αλληλοτροφοδοτούνται για τη βελτίωση της τεκμηρίωσης των παραγόμενων απαντήσεων.

Η συνολική δομή του γράφου της ροή εργασίας και οι δυνατές μεταβάσεις μεταξύ των επιμέρους κόμβων παρουσιάζονται παρακάτω (Εικόνα 9):



Εικόνα 9 Στατική απεικόνιση του γράφου ενορχήστρωσης των Agents με χρήση LangGraph

Κοινή κατάσταση ροής (Workflow State)

Η επικοινωνία μεταξύ των κόμβων πραγματοποιείται μέσω ενός κοινού αντικειμένου κατάστασης, του WorkflowState, το οποίο λειτουργεί ως ενιαίο μέσο μεταφοράς πληροφοριών καθ' όλη τη διάρκεια εκτέλεσης της ροής εργασίας. Κάθε agent λαμβάνει την τρέχουσα κατάσταση, χρησιμοποιεί μόνο τα πεδία που είναι απαραίτητα για τον ρόλο του και επιστρέφει τις ενημερώσεις που παράγει. Με αυτόν τον τρόπο, η ανταλλαγή δεδομένων μεταξύ agents δεν υλοποιείται με άμεση αλληλεξάρτηση μεταξύ των επιμέρους κλάσεων, αλλά μέσω ενός σαφώς

καθορισμένου σχήματος κοινής κατάστασης. Η διατήρηση κατάστασης είναι ουσιώδης στην πολυβηματική Agentic ροή εργασίας, επειδή επιτρέπει τη συνέχιση της ροής με συνεπή πρόσβαση στο ερώτημα, στα ανακτημένα ευρήματα, στο προσχέδιο απάντησης και στις αποφάσεις επαλήθευσης.

Τα βασικά πεδία του WorkflowState παρουσιάζονται συνοπτικά στον ακόλουθο πίνακα (Πίνακας 2):

Πεδίο	Περιεχόμενο και ρόλος
messages	Διατηρεί τα μηνύματα της τρέχουσας αλληλεπίδρασης, συμπεριλαμβανομένου του αρχικού ερωτήματος και της τελικής απόκρισης.
query	Περιέχει το ερώτημα που υποβάλλει ο χρήστης και αποτελεί το βασικό δεδομένο εισόδου της ροής.
relevance_decision	Αποθηκεύει την απόφαση του Relevance Checker Agent ως RELEVANT, PARTIALLY_RELEVANT ή IRRELEVANT.
rag_retrieval	Περιλαμβάνει τα αποτελέσματα ανάκτησης από τη διανυσματική βάση, τον γράφο γνώσης ή την υβριδική διαδικασία ανάκτησης.
draft_answer	Περιέχει το αρχικό ή αναθεωρημένο προσχέδιο απάντησης του Research Agent.
verification_report	Περιλαμβάνει την έκθεση του Verification Agent, με έλεγχο υποστήριξης, μη τεκμηριωμένους ισχυρισμούς, αντιφάσεις και προτεινόμενη επόμενη ενέργεια.
retry_count	Καταγράφει τον αριθμό επαναληπτικών προσπαθειών ανάκτησης, ώστε να περιορίζεται η πιθανότητα απεριόριστου βρόχου.
final_answer	Περιέχει την τελική απάντηση που επιστρέφεται στον χρήστη μετά την ολοκλήρωση της ροής.

Πίνακας 2 Βασικά Πεδία WorkflowState Συστήματος Πολλαπλών Agents

Στην αρχή κάθε εκτέλεσης δημιουργείται μία αρχική κατάσταση που περιλαμβάνει το ερώτημα του χρήστη, το αντίστοιχο μήνυμα συνομιλίας και μηδενικό αριθμό επαναλήψεων. Στη συνέχεια, κάθε κόμβος ενημερώνει συγκεκριμένα πεδία της κατάστασης: ο Relevance Checker agent ενημερώνει το relevance_decision, ο RAG agent το rag_retrieval, ο Research agent το draft_answer και ο Verification agent το verification_report. Τέλος, ο κόμβος finalize καταχωρεί το αποτέλεσμα στο final_answer και ενημερώνει το ιστορικό μηνυμάτων.

Παράλληλα, χρησιμοποιείται μηχανισμός InMemorySaver ως checkpointer, ο οποίος συνδέει την κατάσταση με το thread_id της συνομιλίας και υποστηρίζει τη διατήρηση της ροής στο πλαίσιο της συγκεκριμένης συνεδρίας.

Ροή Επεξεργασίας Ερωτημάτων

Η ροή ξεκινά από τον κόμβο START και προωθεί αρχικά κάθε ερώτημα στον Relevance Checker agent. Αν το ερώτημα χαρακτηριστεί ως IRRELEVANT, η ροή παρακάμπτει την ανάκτηση και οδηγείται απευθείας στον κόμβο finalize, όπου παράγεται ένα κατάλληλο μήνυμα ενημέρωσης προς τον χρήστη. Αντίθετα, όταν η απόφαση είναι RELEVANT ή PARTIALLY_RELEVANT, το ερώτημα μεταφέρεται στον κόμβο rag_retrieval. Εκεί καλείται το υπογράφημα ανάκτησης του RAG agent, ο οποίος επιλέγει δυναμικά την κατάλληλη στρατηγική ανάκτησης αξιοποιώντας τα τρία διαθέσιμα εργαλεία (vector_search_tool, kg_search_tool, hybrid_search_tool) και ενημερώνει το πεδίο rag_retrieval με τα συγκεντρωμένα τεκμήρια.

Έπειτα, ο Research agent αξιοποιεί το ερώτημα, το ιστορικό της συνομιλίας και τα ανακτημένα δεδομένα για να συνθέσει το αρχικό προσχέδιο απάντησης (draft answer). Το προσχέδιο προωθείται στον Verification agent, ο οποίος εξετάζει εάν οι ισχυρισμοί του υποστηρίζονται από τα διαθέσιμα τεκμήρια (rag retrieval), εάν υπάρχουν αντιφάσεις και εάν η απάντηση καλύπτει επαρκώς το αρχικό ερώτημα. Με βάση το verification report που παράγει ο τελευταίος, το LangGraph δρομολογεί τη ροή σε μία από τρεις εναλλακτικές διαδρομές:

- προς τον κόμβο finalize όταν η απάντηση είναι επαρκής
- πίσω στον Research agent (research_node) όταν απαιτείται αναδιατύπωση ή διόρθωση
- πίσω στον RAG agent (rag_retrieval_node) όταν απαιτείται συμπληρωματική ανάκτηση τεκμηρίων.

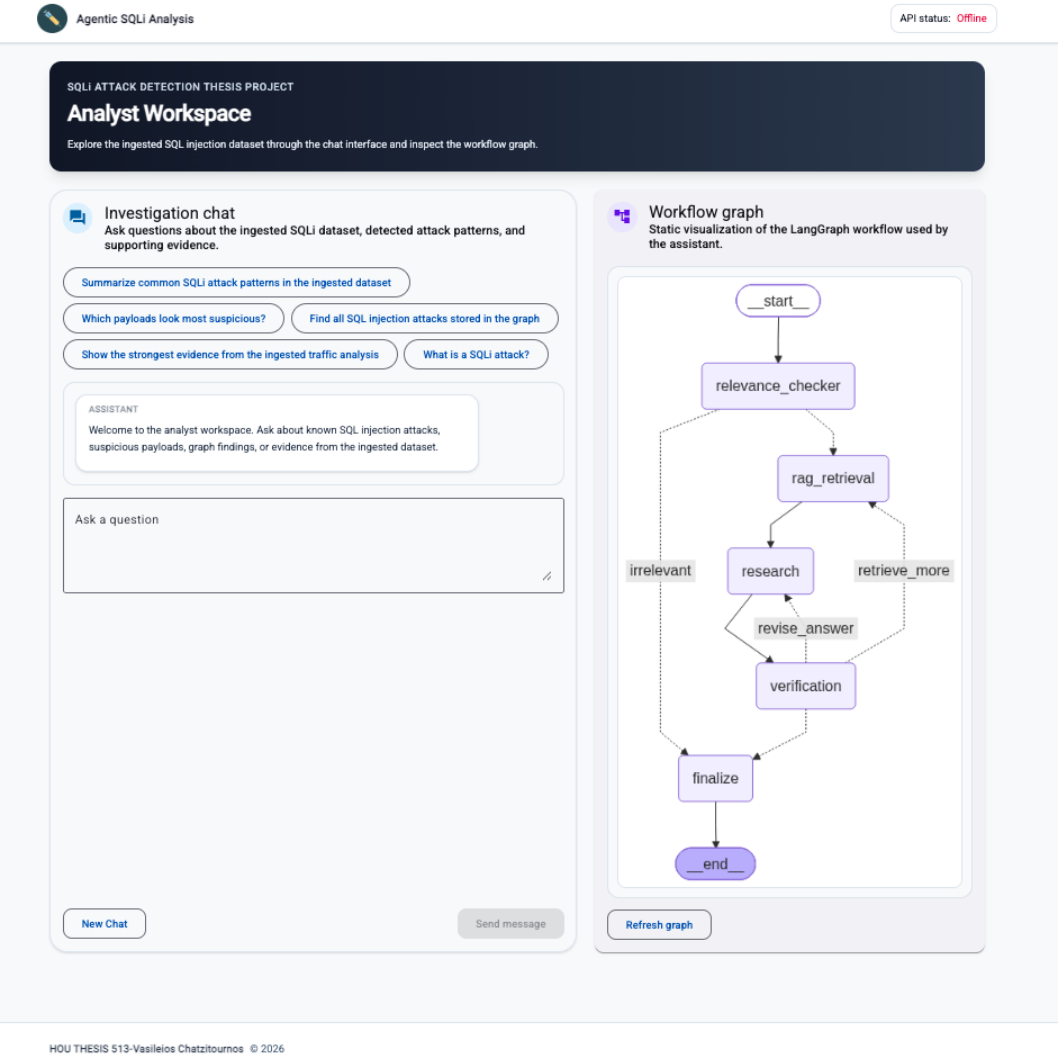
Η τελευταία περίπτωση δημιουργεί έναν ελεγχόμενο βρόχο ανάκτησης–σύνθεσης–επαλήθευσης. Εάν ο μέγιστος επιτρεπτός αριθμός επαναλήψεων έχει επιτευχθεί, η ροή δεν εκτελεί νέα ανάκτηση αλλά οδηγείται σε αναθεώρηση της υπάρχουσας απάντησης, περιορίζοντας το υπολογιστικό κόστος και αποτρέποντας ατέρμονη επανάληψη.

4.6 Διεπαφή Χρήστη και Υποστηρικτικές Υπηρεσίες – 5ο Επίπεδο (5th Layer)

Το τελικό επίπεδο της αρχιτεκτονική του συστήματος, συμπληρώνεται από το επίπεδο παρουσίασης και το επίπεδο διεπαφής προγραμματισμού εφαρμογών (Application Programming Interface - API), τα οποία εξασφαλίζουν την επικοινωνία του χρήστη με το Agentic υποσύστημα και τις υποκείμενες βάσεις δεδομένων. Η διεπαφή χρήστη (User Interface - UI) υλοποιείται ως εφαρμογή διαδικτύου που αναπτύχθηκε με το framework της Angular, ενώ η επικοινωνία με το backend πραγματοποιείται μέσω REST API που αναπτύχθηκε με FastAPI. Η διάκριση αυτή επιτρέπει τον σαφή διαχωρισμό μεταξύ παρουσίασης, επιχειρησιακής λογικής και αποθήκευσης δεδομένων, διατηρώντας την εφαρμογή αρθρωτή και ευκολότερα επεκτάσιμη.

4.6.1 Διεπαφή Χρήστη (UI)

Η διεπαφή χρήστη υλοποιήθηκε ως ένα διαδραστικό περιβάλλον εργασίας, βασισμένο στο framework Angular. Σκοπός της είναι να παρέχει ένα ενιαίο και απλό περιβάλλον, μέσω του οποίου ο χρήστης μπορεί να υποβάλλει ερωτήματα σχετικά με το εισαγμένο σύνολο δεδομένων και να λαμβάνει τις απαντήσεις του Agentic συστήματος. Όπως παρουσιάζεται στην Εικόνα 10, η κεντρική οθόνη οργανώνεται σε δύο κύριες περιοχές: την περιοχή συνομιλίας (*Investigation chat*) και την περιοχή οπτικοποίησης του γράφου ροής εργασίας του LangGraph (*Workflow graph*).



Εικόνα 10 Διεπαφή χρήστη του συστήματος ανάλυσης SQLi, με περιοχή συνομιλίας και στατική απεικόνιση του Agentic workflow

Η περιοχή συνομιλίας λειτουργεί ως το κύριο σημείο αλληλεπίδρασης με το σύστημα. Παρέχει πεδίο εισαγωγής ερωτήματος, δυνατότητα έναρξης νέας συνομιλίας και περιοχή εμφάνισης των απαντήσεων που παράγονται από το backend. Επιπλέον, περιλαμβάνει ενδεικτικές προτάσεις ερωτημάτων, όπως η αναζήτηση ύποπτων payloads, η εμφάνιση επιθέσεων SQL injection που έχουν καταγραφεί στον γράφο γνώσης ή η παρουσίαση των ισχυρότερων ενδείξεων από την ανάλυση της δικτυακής κίνησης. Οι προτάσεις αυτές διευκολύνουν την καθοδήγηση του χρήστη ως προς ερωτήματα εντός του πεδίου εφαρμογής και αναδεικνύουν τις βασικές δυνατότητες του συστήματος.

Παράλληλα, η περιοχή *Workflow graph* προβάλλει τη στατική οπτικοποίηση του γράφου LangGraph που ενορχηστρώνει τους agents. Η απεικόνιση περιλαμβάνει τους κόμβους

relevance_checker, rag_retrieval, research, verification και finalize, καθώς και τις δυνατές μεταβάσεις μεταξύ τους. Επομένως, η διεπαφή δεν χρησιμοποιείται μόνο για την υποβολή και λήψη απαντήσεων, αλλά συνεισφέρει και στην επεξηγησιμότητα της εφαρμογής, παρέχοντας στον χρήστη ορατότητα ως προς τη λογική της Agentic ροής εργασίας. Η ένδειξη κατάστασης του API και η λειτουργία ανανέωσης του γραφήματος υποστηρίζουν επιπλέον τον βασικό έλεγχο διαθεσιμότητας του backend και την επικαιροποίηση της παρουσιαζόμενης δομής.

4.6.2 Επίπεδο Διεπαφής Προγραμματισμού Εφαρμογών (API)

Το API υλοποιήθηκε με χρήση του FastAPI, ενός ασύγχρονου Python framework για την ανάπτυξη διαδικτυακών APIs. Το FastAPI λειτουργεί ως το ενδιάμεσο επίπεδο μεταξύ της Angular εφαρμογής και της επιχειρησιακής λογικής του συστήματος, παρέχοντας ελεγχόμενη πρόσβαση στις υπηρεσίες του Agentic υποσυστήματος. Κατά την εκκίνηση της εφαρμογής, μέσω του μηχανισμού lifespan, δημιουργείται ένα στιγμιότυπο της υπηρεσίας WorkflowService, το οποίο περιλαμβάνει το compiled LangGraph workflow και διατηρείται στην κατάσταση της εφαρμογής. Με αυτόν τον τρόπο, κάθε εισερχόμενο αίτημα μπορεί να αξιοποιεί την ίδια διαχειριζόμενη υπηρεσία χωρίς να απαιτείται επαναδημιουργία ολόκληρου του Agentic υποσυστήματος.

Το κύριο endpoint της εφαρμογής είναι το POST /api/v1/chat/query. Το endpoint δέχεται ένα αντικείμενο ChatQueryRequest, το οποίο περιλαμβάνει το ερώτημα του χρήστη και το thread_id της συνομιλίας. Στη συνέχεια, καλεί τη μέθοδο ask() της WorkflowService, η οποία ενεργοποιεί τη ροή του LangGraph και επιστρέφει την τελική απάντηση που προκύπτει μετά την ολοκλήρωση των σταδίων ελέγχου συνάφειας, ανάκτησης, παραγωγής προσχεδίου και επαλήθευσης. Η απάντηση επιστρέφεται στο frontend με τη μορφή ChatQueryResponse, διατηρώντας το ίδιο thread_id, ώστε να υποστηρίζεται η σύνδεση των μηνυμάτων με τη συγκεκριμένη συνεδρία συνομιλίας.

Εκτός από το endpoint συνομιλίας, η εφαρμογή περιλαμβάνει router ελέγχου διαθεσιμότητας (*health endpoint*), ο οποίος χρησιμοποιείται από το frontend για την εμφάνιση της κατάστασης του API, καθώς και router για την ανάκτηση ή ανανέωση της οπτικοποίησης του γράφου ροής εργασίας.

Τέλος, το σύνολο της εφαρμογής έχει αναπτυχθεί σε περιβάλλον Docker Compose. Η σύνθεση περιλαμβάνει ανεξάρτητα containers για την Angular εφαρμογή, το FastAPI backend, τη

διανυσματική βάση Qdrant και τη γραφική βάση Neo4j. Η υπηρεσία frontend εκτίθεται στη θύρα 4200 και το backend στη θύρα 8000, ενώ οι υπηρεσίες Qdrant και Neo4j λειτουργούν στις αντίστοιχες θύρες τους και διατηρούν τα δεδομένα τους σε μόνιμους τόμους αποθήκευσης. Η χρήση Docker Compose επιτρέπει την επαναλήψιμη εγκατάσταση, τη σαφή απομόνωση των επιμέρους υπηρεσιών και την απλούστευση της ανάπτυξης του συστήματος σε διαφορετικά περιβάλλοντα εκτέλεσης.

5. Πειραματική Διαδικασία και Αξιολόγηση

Το προηγούμενο κεφάλαιο παρουσίασε τον σχεδιασμό και την υλοποίηση της προτεινόμενης αρχιτεκτονικής, από την εισαγωγή και επεξεργασία των αρχείων PCAP έως την οργάνωση της υβριδικής υποδομής γνώσης και την ενορχήστρωση των agents. Με βάση την αρχιτεκτονική αυτή, στο παρόν κεφάλαιο εξετάζεται η λειτουργία του συστήματος σε ελεγχόμενο εργαστηριακό περιβάλλον και αξιολογείται η ικανότητά του να αξιοποιεί τα διαθέσιμα τεχνικά τεκμήρια για την ανάλυση πιθανών επιθέσεων SQL injection.

Αρχικά παρουσιάζονται η εργαστηριακή υποδομή, ο τρόπος δημιουργίας των αρχείων PCAP και η συλλογή εξειδικευμένης γνώσης που χρησιμοποιήθηκε για την αρχικοποίηση της Qdrant και της Neo4j. Στη συνέχεια περιγράφεται η μεθοδολογία αξιολόγησης και αναλύονται τα αποτελέσματα τόσο της λειτουργίας του υποσυστήματος Agentic RAG, όσο και της αναγνώρισης και ταξινόμησης SQL injection σε επίπεδο αρχείου PCAP. Τέλος, τα ευρήματα συζητούνται σε σχέση με τους στόχους της εργασίας, τις δυνατότητες της προτεινόμενης προσέγγισης και τους περιορισμούς του πειραματικού πλαισίου.

5.1 Πειραματικό Περιβάλλον και Σύνολο Δεδομένων (Dataset & Setup)

Η αξιολόγηση της προτεινόμενης αρχιτεκτονικής πραγματοποιήθηκε σε ελεγχόμενο εργαστηριακό περιβάλλον, με σκοπό τη διερεύνηση της δυνατότητας εντοπισμού, ανάκτησης και ερμηνείας ενδείξεων επιθέσεων SQL injection μέσα από αρχεία καταγραφής δικτυακής κίνησης σε μορφή PCAP. Η επιλογή ελεγχόμενου περιβάλλοντος κρίθηκε απαραίτητη, καθώς επέτρεψε την παραγωγή δικτυακής κίνησης με γνωστή προέλευση, σαφώς καθορισμένα σενάρια και τεκμηριωμένη ground-truth επισήμανση ανά αρχείο καταγραφής.

Επιπλέον, η αξιολόγηση οργανώθηκε γύρω από δύο διακριτά αλλά αλληλοσυμπληρούμενα σύνολα δεδομένων. Το πρώτο σύνολο αφορά πραγματικά δεδομένα δικτυακής κίνησης σε μορφή PCAP, τα οποία χρησιμοποιούνται για την κατασκευή του γράφου γνώσης στη Neo4j. Το δεύτερο σύνολο αφορά τεχνική και θεωρητική γνώση σχετικά με επιθέσεις SQL injection, η οποία επεξεργάστηκε σε μορφή chunks και αποθηκεύθηκε στη διανυσματική βάση Qdrant.

Η διάκριση αυτή είναι ουσιώδης για την αξιολόγηση της HybridRAG αρχιτεκτονικής. Η Neo4j παρέχει στοιχεία που έχουν παρατηρηθεί στο πειραματικό traffic, όπως URI, μεθόδους HTTP, query parameters, request bodies, IP διευθύνσεις, cookies και χρονικές σημάνσεις. Αντίθετα, η Qdrant παρέχει το εξωτερικό γνωστικό πλαίσιο που απαιτείται ώστε το σύστημα να ερμηνεύει την πιθανή σημασία ενός payload, να διακρίνει τεχνικές SQLi και να παράγει τεκμηριωμένες εξηγήσεις ή προτάσεις αντιμετώπισης.

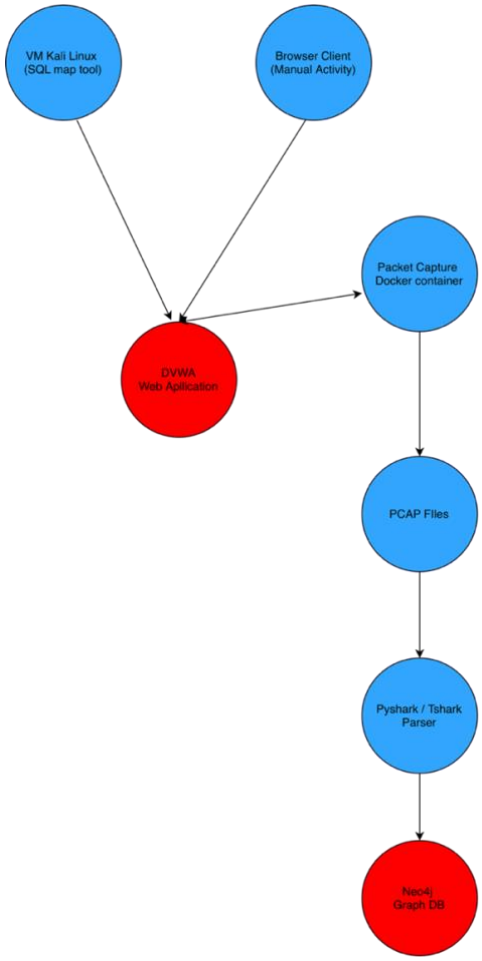
5.1.1 Εργαστηριακή Υποδομή

Το εργαστηριακό περιβάλλον υλοποιήθηκε σε υπολογιστή Mac mini (CPU M4/RAM 16GB) με χρήση Docker Desktop. Η εφαρμογή Damn Vulnerable Web Application (DVWA) εκτελέστηκε σε container Docker, ενώ η βάση δεδομένων MariaDB εκτελέστηκε ως ξεχωριστή υπηρεσία. Η DVWA επιλέχθηκε επειδή επιτρέπει την ελεγχόμενη δημιουργία σεναρίων SQL injection σε εφαρμογή που προορίζεται για εκπαιδευτική και πειραματική χρήση.

Η συλλογή πακέτων πραγματοποιήθηκε μέσω ειδικού container καταγραφής, το οποίο μοιραζόταν το δικτυακό namespace του DVWA container. Με αυτόν τον τρόπο καταγράφηκαν HTTP requests και responses που έφθαναν στην ευάλωτη εφαρμογή, ενώ τα αποτελέσματα αποθηκεύονταν ως αρχεία PCAP.

Για την παραγωγή αυτοματοποιημένης επιθετικής κίνησης χρησιμοποιήθηκε εικονική μηχανή Kali Linux. Η εικονική μηχανή επικοινωνούσε με τη DVWA μέσω ιδιωτικού δικτύου και το εργαλείο sqlmap εκτελέστηκε αποκλειστικά εναντίον της τοπικά αναπτυγμένης εφαρμογής. Δεν πραγματοποιήθηκε κανένας έλεγχος ή επίθεση εναντίον συστήματος που δεν ανήκε στο ελεγχόμενο περιβάλλον της εργασίας.

Η ροή του πειράματος για την καταγραφή της δικτυακής κίνησης σε αρχεία PCAP είναι η παρακάτω (Εικόνα 11):



Εικόνα 11 Ροή Καταγραφής Δικτυακής Κίνησης και Εισαγωγής στη Neo4j

5.1.2 Σύνολο Δικτυακών Δεδομένων PCAP

Το πρώτο σύνολο δεδομένων δημιουργήθηκε ειδικά για την παρούσα εργασία και αποτελείται από αρχεία PCAP που συλλέχθηκαν κατά την εκτέλεση benign και SQLi δραστηριότητας στην DVWA. Κάθε αρχείο PCAP αντιστοιχεί σε μία καθορισμένη συνεδρία ή σε ένα ελεγχόμενο σενάριο δραστηριότητας, ώστε να μπορεί να αποδοθεί μία αξιόπιστη ground-truth ετικέτα. Η ground-truth επισήμανση περιγράφει το σενάριο που εκτελέστηκε κατά την εργαστηριακή συλλογή και χρησιμοποιείται για τη σύγκριση των αποτελεσμάτων της αξιολόγησης. Δεν προκύπτει αποκλειστικά από την παρουσία μεμονωμένων payloads στο PCAP ούτε ισοδυναμεί σε πραγματικό περιβάλλον με απόδειξη επιτυχούς εκμετάλλευσης μιας ευπάθειας.

Οι επιθέσεις SQL injection δημιουργήθηκαν είτε χειροκίνητα είτε αυτοματοποιημένα μέσω του εργαλείου sqlmap. Τα αυτοματοποιημένα σενάρια περιλαμβάνουν Boolean-based blind, Error-based, Time-based blind και UNION-based SQL injection. Επιπλέον, δημιουργήθηκαν

mixed captures που περιέχουν περισσότερες από μία τεχνικές SQLi, καθώς και captures κανονικής χρήσης της εφαρμογής.

Για την ενίσχυση της αξιολόγησης false positives, συμπεριλήφθηκαν benign και hard-negative σενάρια. Τα τελευταία περιλαμβάνουν τιμές οι οποίες δεν είχαν κακόβουλη πρόθεση, αλλά μπορούν να εμφανίζουν στοιχεία που μοιάζουν επιφανειακά με SQLi, όπως χρήση αποστρόφου σε username. Η χρήση τέτοιων σεναρίων είναι απαραίτητη, επειδή ένας ειδικός χαρακτήρας από μόνος του δεν αποτελεί επαρκές αποδεικτικό στοιχείο SQL injection.

Ο παρακάτω πίνακας (Πίνακας 3) παρουσιάζει το σύνολο των PCAP αρχείων, που συλλέχθηκαν και τα οποία στη συνέχεια αποτέλεσαν τη απαραίτητη γνώση για τη δημιουργία του γράφου στη Neo4j:

Capture ID	Αρχείο PCAP	Ground-truth κλάση	Τύπος επίθεσης	Εργαλείο παραγωγής	Παράμετρος στόχος	Ground-truth επισήμανση
capture-001	capture-001.pcap	Benign	None	Manual	None	Benign
capture-002	capture-002.pcap	Benign	None	Browser	None	Benign
capture-003	capture-003.pcap	SQLi	Boolean-based	sqlmap	id	SQLi
capture-004	capture-004.pcap	SQLi	Error-based	sqlmap	id	SQLi
capture-005	capture-005.pcap	SQLi	Time-based	sqlmap	id	SQLi
capture-006	capture-006.pcap	SQLi	UNION-based	sqlmap	id	SQLi
capture-007	capture-007.pcap	SQLi	Mixed	sqlmap	id	SQLi
capture-008	capture-008.pcap	Benign	None	Manual	id	Benign
capture-009	capture-009.pcap	Benign	None	Manual	id	Benign
capture-010	capture-010.pcap	Benign	None	Manual	id	Benign

Capture ID	Αρχείο PCAP	Ground-truth κλάση	Τύπος επίθεσης	Εργαλείο παραγωγής	Παράμετρος στόχος	Ground-truth επισήμανση
capture-011	capture-011.pcap	Benign-Invalid	None	Manual	id	Benign
capture-012	capture-012.pcap	Benign-Hard-Negative	None	Manual	id	Benign
capture-013	capture-013.pcap	SQLi	Boolean-based	sqlmap	id	SQLi
capture-014	capture-014.pcap	SQLi	Error-based	sqlmap	id	SQLi
capture-015	capture-015.pcap	SQLi	UNION-based	sqlmap	id	SQLi
capture-016	capture-016.pcap	SQLi	Time-based	sqlmap	id	SQLi
capture-017	capture-017.pcap	SQLi	Mixed	sqlmap	id	SQLi
capture-018	capture-018.pcap	SQLi	Mixed	sqlmap	id	SQLi

Πίνακας 3 Σύνολο Αρχείων PCAP και Εκτελεσθέντα Σενάρια SQLi

Η εφαρμογή DVWA χρησιμοποιήθηκε με ρύθμιση ασφαλείας *Low*, ώστε να επιτρέπει την εκτέλεση ελεγχόμενων σεναρίων SQL injection. Η επιλογή αυτή δεν αποσκοπεί στην προσομοίωση ενός πλήρως ρεαλιστικού παραγωγικού περιβάλλοντος, αλλά στην ελεγχόμενη παραγωγή καθαρών και ανιχνεύσιμων αιτημάτων HTTP, τα οποία περιλαμβάνουν SQLi payloads σε μη κρυπτογραφημένη μορφή.

Η καταγραφή πραγματοποιήθηκε σε HTTP και όχι HTTPS, ώστε οι μέθοδοι HTTP, τα URI, οι query παράμετροι, τα POST request bodies και τα cookies να είναι ορατά στο PCAP. Η επιλογή αυτή συνδέεται με τον στόχο της εργασίας, δηλαδή την ανάλυση αποδεικτικών στοιχείων επιπέδου εφαρμογής μέσα από αρχεία PCAP. Επίσης, Οι ground-truth ετικέτες διατηρήθηκαν σε ξεχωριστό αρχείο manifest.csv και όχι μέσα στο όνομα των αρχείων PCAP, ώστε η αξιολόγηση του συστήματος να βασίζεται στο περιεχόμενο της δικτυακής κίνησης και όχι σε πληροφορία που υποδηλώνεται από το όνομα του αρχείου.

5.1.3 Συλλογή Εξειδικευμένης Γνώσης SQL Injection

Το δεύτερο σύνολο δεδομένων αφορά το σύνολο εξειδικευμένης γνώσης που χρησιμοποιείται από τη διανυσματική βάση Qdrant. Σε αντίθεση με τα αρχεία PCAP, τα οποία περιέχουν παρατηρημένη δικτυακή δραστηριότητα, το συγκεκριμένο σύνολο δεδομένων αποτελείται από τεχνικό και θεωρητικό περιεχόμενο για τους διάφορους τύπους SQL injection.

Η γνώση αυτή προήλθε από δύο κύριες κατηγορίες πηγών:

- Αρχεία Markdown από το αποθετήριο **PayloadsAllTheThings – SQL Injection** (*Swisskyrepo/PayloadsAllTheThings: A List of Useful Payloads and Bypass for Web Application Security and Pentest/CTF*, n.d.).

- Περιεχόμενο ιστοσελίδων που συλλέχθηκε με το Crawl4AI, ξεκινώντας από τρεις αρχικές πηγές και επεκτεινόμενο έως δύο επίπεδα συνδέσμων.

Οι markdown πηγές από το αποθετήριο PayloadsAllTheThings περιλαμβάνουν δομημένο υλικό για τους διάφορους τύπους SQL injection, όπως MySQL, PostgreSQL, Oracle, SQLite, SQLmap, UNION-based, Error-based, Boolean-based blind, Time-based, Out-of-Band και Second-order SQL injection. Επιπλέον είναι κατηγοριοποιημένες, αναλόγως του DBMS που επιχειρούν να παραβιάσουν, δηλαδή το πως εκτελούνται σε MySQL, PostgreSQL, Oracle, SQLite κ.α. Περιλαμβάνουν επίσης τεχνικές παράκαμψης WAF και παραδείγματα ειδικών payload patterns.

Οι αρχικές διαδικτυακές πηγές που χρησιμοποιήθηκαν για το web crawling ήταν οι ακόλουθες:

- https://owasp.org/Top10/2025/A05_2025-Injection/
- https://www.nature.com/articles/s41598-023-48845-4
- https://www.sciencedirect.com/science/article/pii/S0167404823000032

Το crawling περιορίστηκε σε βάθος δύο επιπέδων συνδέσμων από κάθε αρχική σελίδα. Η επιλογή αυτή έγινε ώστε να επιτευχθεί ισορροπία μεταξύ κάλυψης σχετικής γνώσης και περιορισμού άσχετου ή υπερβολικά εκτεταμένου περιεχομένου.

Η Qdrant δεν λαμβάνει ως είσοδο τα ίδια τα αρχεία PCAP. Αντίθετα, λειτουργεί ως επιπρόσθετη γνώση. Όταν ο RAG agent ανακτά από τη Neo4j ένα URI ή μια παράμετρο που περιέχει ύποπτη συμβολοσειρά, η Qdrant μπορεί να παρέχει πληροφορία για το τι ενδέχεται να σημαίνει το pattern. Για παράδειγμα, ένα payload που περιέχει UNION SELECT μπορεί να

συσχετιστεί με πληροφορία για UNION-based SQL injection, ενώ ένα payload που περιέχει SLEEP() μπορεί να συσχετιστεί με Time-based blind SQL injection.

5.2 Μεθοδολογία Αξιολόγησης

Η αξιολόγηση της προτεινόμενης υλοποίησης οργανώθηκε σε δύο διακριτές αλλά συμπληρωματικές πειραματικές διαδικασίες. Η πρώτη εξετάζει την ποιότητα λειτουργίας του Agentic RAG υποσυστήματος ως προς την κατανόηση των ερωτημάτων, την επιλογή εργαλείων ανάκτησης, τη συνάφεια των ανακτημένων τεκμηρίων, την τεκμηρίωση των απαντήσεων και την ορθή διαχείριση ερωτημάτων εκτός πεδίου. Η δεύτερη εξετάζει την ικανότητα του συστήματος να αναγνωρίζει και να ταξινομεί πιθανές επιθέσεις SQL injection σε επίπεδο αρχείου PCAP, αξιοποιώντας τα στοιχεία HTTP που έχουν μετασχηματιστεί και αποθηκευτεί στον γράφο γνώσης.

Η διάκριση αυτή θεωρείται ουσιώδης, καθώς η επιτυχία ενός Agentic RAG συστήματος δεν εξαντλείται στην τελική ορθότητα της απάντησης. Απαιτείται επιπλέον να αξιολογηθεί εάν το σύστημα επιλέγει κατάλληλα την πηγή γνώσης, ανακτά τεκμήρια που σχετίζονται με το ερώτημα, αποφεύγει μη υποστηριζόμενους ισχυρισμούς και εφαρμόζει σωστά τη ροή ελέγχου συνάφειας. Αντίστοιχα, στην περίπτωση της ανίχνευσης SQLi, είναι απαραίτητο να διαχωριστεί η απλή αναγνώριση κακόβουλης δραστηριότητας από την ακριβή αναγνώριση του τύπου επίθεσης και από την ποιότητα της τεχνικής αιτιολόγησης που συνοδεύει την απόφαση.

5.2.1 Πλαίσιο Εκτέλεσης και Αξιολόγησης με LangSmith

Για την υλοποίηση της πειραματικής διαδικασίας χρησιμοποιήθηκε το **LangSmith**, πλατφόρμα παρατηρησιμότητας, αξιολόγησης και ανάλυσης εφαρμογών που βασίζονται σε μεγάλα γλωσσικά μοντέλα. Το LangSmith αξιοποιήθηκε αφενός για την καταγραφή των traces της εκτέλεσης του Agentic συστήματος και αφετέρου για τη συστηματική εκτέλεση των συνόλων δεδομένων αξιολόγησης. Μέσω της πλατφόρμας καταγράφηκαν το ερώτημα εισόδου, οι κλήσεις των agents, τα εργαλεία που επιλέχθηκαν, το ανακτημένο πλαίσιο από τη Qdrant και τη Neo4j, οι ενδιάμεσες απαντήσεις και η τελική απόκριση του συστήματος. Η δυνατότητα αυτή είναι ιδιαίτερα σημαντική σε μια αρχιτεκτονική πολλαπλών agents, καθώς

επιτρέπει την αξιολόγηση όχι μόνο του τελικού αποτελέσματος, αλλά και της πορείας που ακολούθησε το σύστημα μέχρι την παραγωγή του.

Η εκτέλεση των πειραμάτων απομονώθηκε σε ανεξάρτητο container Docker, με την ονομασία sqli_eval_runner. Το container χρησιμοποιεί την ίδια εικόνα με το backend της εφαρμογής, ώστε να διατηρείται κοινό περιβάλλον εξαρτήσεων, μοντέλων, μεταβλητών ρυθμίσεων και μηχανισμών πρόσβασης στις βάσεις Qdrant και Neo4j. Η επιλογή αυτή βελτιώνει την επαναληψιμότητα της διαδικασίας, καθώς η αξιολόγηση εκτελείται σε περιβάλλον συνεπές με εκείνο της κύριας εφαρμογής, χωρίς να απαιτείται χειροκίνητη ενεργοποίηση του frontend ή του πλήρους fastAPI.

Η αξιολόγηση των παραγόμενων απαντήσεων πραγματοποιήθηκε με την τεχνική **LLM-as-a-Judge**. Στην προσέγγιση αυτή, ένα γλωσσικό μοντέλο αναλαμβάνει τον ρόλο του αξιολογητή και συγκρίνει την απόκριση του συστήματος με την αναμενόμενη αναφορά, το αρχικό ερώτημα και, όπου απαιτείται, το ανακτημένο πλαίσιο. Η αξιολόγηση δεν βασίζεται στην ακριβή λεκτική ταύτιση της παραγόμενης απάντησης με μια προκαθορισμένη αναφορά, αλλά στη σημασιολογική αντιστοίχιση, στην τεκμηρίωση και στη συνέπεια με τα διαθέσιμα στοιχεία. Η προσέγγιση αυτή είναι κατάλληλη για συστήματα RAG και Agentic RAG, όπου μπορεί να υπάρχουν περισσότερες από μία ισοδύναμες ορθές διατυπώσεις και όπου η ποιότητα της απάντησης εξαρτάται τόσο από το περιεχόμενο όσο και από τη χρήση των ανακτημένων τεκμηρίων. Για την εκτέλεση των πειραμάτων αξιολόγησης, ως LLM-as-a-Judge χρησιμοποιήθηκε το μοντέλο Qwen/Qwen3-8B.

Παράλληλα, τα αποτελέσματα του LLM-as-a-Judge πρέπει να ερμηνεύονται ως αυτοματοποιημένη αξιολόγηση υποστήριξης και όχι ως απόλυτη αντικατάσταση ανθρώπινης κρίσης. Η αξιολόγηση αυτή εξαρτάται από το prompt του κριτή, από την ποιότητα των απαντήσεων αναφοράς και από τη δυνατότητα του μοντέλου να ερμηνεύει ορθά το διαθέσιμο τεχνικό πλαίσιο. Για τον λόγο αυτό, η ποσοτική βαθμολόγηση συνδυάζεται στις επόμενες ενότητες με ποιοτική εξέταση χαρακτηριστικών περιπτώσεων.

5.2.2 Σύνολα Αξιολόγησης και Κριτήρια Βαθμολόγησης

Το πρώτο σύνολο, με τίτλο Agentic RAG Quality (Παράρτημα Β: Σύνολα Αξιολόγησης), περιλαμβάνει δέκα ελεγχόμενες περιπτώσεις ερωτημάτων. Οι περιπτώσεις καλύπτουν ερωτήματα γενικής γνώσης για επιθέσεις SQLi, επεξηγηματικά ερωτήματα σχετικά με

τεχνικές επίθεσης, άσχετα ερωτήματα, μερικώς σχετικά ερωτήματα, ερωτήματα που απαιτούν ανάκτηση από τα εισαγμένα δεδομένα, περιπτώσεις αναζήτησης συγκεκριμένου τύπου SQLi και ένα hard-negative σενάριο. Για κάθε περίπτωση ορίζονται το ερώτημα του χρήστη, μία απάντηση αναφοράς, τα αναμενόμενα εργαλεία ανάκτησης και, όπου κρίθηκε αποδεκτό, εναλλακτικά εργαλεία.

Στην αξιολόγηση Agentic RAG χρησιμοποιήθηκαν πέντε μετρικές βαθμολόγησης (Πίνακας 4) από 0 έως 1:

Μετρική	Περιγραφή
Relevance	Αξιολογεί εάν η τελική απάντηση, απαντά άμεσα στο ερώτημα του χρήστη
Groundedness	Αξιολογεί εάν οι πραγματολογικοί ισχυρισμοί υποστηρίζονται από το ανακτημένο πλαίσιο
Retrieval Quality	Αξιολογεί εάν τα ανακτημένα στοιχεία είναι χρήσιμα και συναφή με την απάντηση
Tool Use	Αξιολογεί εάν χρησιμοποιήθηκαν τα αναμενόμενα ή αποδεκτά εναλλακτικά εργαλεία
Refusal	Αξιολογεί εάν το σύστημα χειρίστηκε ορθά σχετικά και άσχετα ερωτήματα

Πίνακας 4 Μετρικές Βαθμολόγησης Agentic RAG Συστήματος

Το δεύτερο σύνολο αξιολόγησης (Παράρτημα Β: Σύνολα Αξιολόγησης), με τίτλο SQLi Detection, περιλαμβάνει δεκαοκτώ περιπτώσεις αξιολόγησης, μία για κάθε αρχείο PCAP του πειραματικού συνόλου. Κάθε περίπτωση περιέχει το όνομα του αρχείου PCAP, το ερώτημα ανάλυσης, την ground-truth ετικέτα benign ή sqli, τον αναμενόμενο τύπο επίθεσης και την αναμενόμενη τεχνική αιτιολόγηση. Το σύνολο περιλαμβάνει benign δικτυακή κίνηση, σενάρια διαφόρων τύπων επιθέσεων SQLi, Error-Based, Time-Based, Union-Based, Boolean-Based, καθώς και SQLi σενάρια πολλαπλών τεχνικών ταυτόχρονα. Για ορισμένα αρχεία PCAP επιτρέπονται περισσότεροι από ένας αποδεκτός τύπος επίθεσης, όπως boolean-based or multi-technique, επειδή η πραγματική δικτυακή κίνηση ενδέχεται να περιέχει τόσο την κυρίαρχη τεχνική όσο και επιπλέον χειροκίνητα payloads άλλων τεχνικών.

Η αξιολόγηση ανίχνευσης SQLi βασίστηκε σε τρεις μετρικές (Πίνακας 5):

Μετρική	Περιγραφή
Detection Correctness	Ελέγχει εάν το PCAP ταξινομήθηκε σύμφωνα με την εργαστηριακή ground-truth επισήμανση ως benign ή ως κίνηση που περιέχει artifacts συμβατά με SQLi.
Attack Type Accuracy	Ελέγχει εάν ο τύπος δραστηριότητας SQLi που αποδίδεται από το σύστημα αντιστοιχεί στην εργαστηριακή ετικέτα αναφοράς ή σε μία από τις αποδεκτές εναλλακτικές.
Justification Quality	Ελέγχει εάν η τεχνική αιτιολόγηση είναι συμβατή με την ταξινόμηση και αναφέρεται σε πραγματικά στοιχεία του PCAP

Πίνακας 5 Μετρικές Αξιολόγησης Ανίχνευσης SQLi Επιθέσεων

Στην περίπτωση benign κίνησης, πλήρης βαθμολογία αποδίδεται όταν το σύστημα δεν αναφέρει SQL injection ή άλλο κακόβουλο συμβάν και δεν αποδίδει τύπο επίθεσης. Στις περιπτώσεις που η εργαστηριακή ground truth επισήμανση αντιστοιχεί σε σενάριο SQLi, πλήρης βαθμολογία για τον τύπο αποδίδεται όταν η απάντηση αναγνωρίζει έναν αποδεκτό τύπο δραστηριότητας SQLi χωρίς να προσθέτει μη υποστηριζόμενες κατηγορίες. Μερική βαθμολογία μπορεί να αποδοθεί όταν η απάντηση αναγνωρίζει ορθά την ύπαρξη ενδείξεων συμβατές με SQL injection, αλλά παραμένει γενική ως προς τον τύπο της επίθεσης ή περιλαμβάνει πρόσθετους μη επαρκώς τεκμηριωμένους τύπους. Ο αναλυτικός τρόπος με τον οποίο αξιολογεί τα αποτελέσματα και των δύο συνόλων ο LLM-as-a-Judge αποτυπώνεται στο prompt που του δίνεται από το σύστημα.

5.3 Αξιολόγηση Λειτουργίας του Agentic RAG

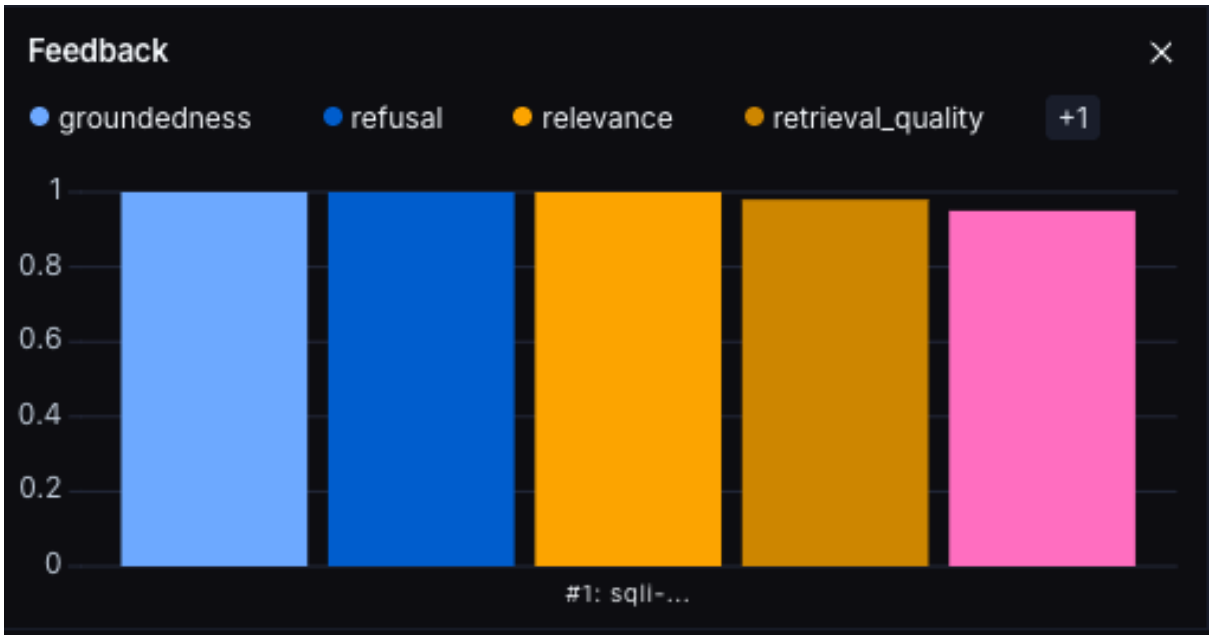
Η πρώτη αξιολόγηση εξετάζει τη συνολική συμπεριφορά της ροής εργασίας Agentic RAG σε διαφορετικούς τύπους ερωτημάτων. Ο στόχος δεν είναι μόνο να ελεγχθεί αν το σύστημα παράγει μια αποδεκτή τελική απάντηση, αλλά και αν ενεργοποιεί σωστά τους επιμέρους μηχανισμούς, όπως τον έλεγχο συνάφειας, την ορθή επιλογή εργαλείων, την αποτελεσματική ανάκτηση από τη διανυσματική βάση ή το γράφο γνώσης, την σύνθεση απάντησης και τον έλεγχο τεκμηρίωσης.

Στη παρακάτω εικόνα (Εικόνα 12) φαίνονται το πως έχει αξιολογήσει το LLM-as-a-Judge τα αποτελέσματα του συστήματος επί των δέκα ερωτήσεων του συνόλου Agentic RAG Quality στις 5 διαφορετικές μετρικές:

	Inputs	groundedness 1.00 AVG	refusal 1.00 AVG	relev... 1.00 AVG	retri... 0.98 AVG	tool_... 0.95 AVG
1	In capture-014.pcap, can you detect any malicious attempt... f yes, what kind of attack is it and what is the justification?	1.00	1.00	1.00	1.00	1.00
2	Can you find any time-based SQLi attacks in your data and... n which captures?	1.00	1.00	1.00	1.00	1.00
3	Can you explain me the key differences between SQLi and ... SS attacks?	1.00	1.00	1.00	0.80	1.00
4	In capture-012.pcap traffic is there any evidence of SQLi at... cks? If yes, what kind of attack is it and what is the justificati	1.00	1.00	1.00	1.00	1.00
5	In capture-018.pcap, can you detect any malicious attempt... f yes, what kind of attack is it and what is the justification?	1.00	1.00	1.00	1.00	0.50
6	What is a SQLi attack?	1.00	1.00	1.00	1.00	1.00
7	How is a UNION-based SQLi attack performed?	1.00	1.00	1.00	1.00	1.00
8	Can you explain me how a XSS attack works?	1.00	1.00	1.00	1.00	1.00
9	Is SQLi attacks and gelato related?	1.00	1.00	1.00	1.00	1.00
10	I want some pizza	1.00	1.00	1.00	1.00	1.00

Εικόνα 12 Αποτελέσματα Αξιολόγησης Agentic RAG Quality

Ενώ στην επόμενη εικόνα (Εικόνα 13) φαίνεται η συνολική απόδοση του Agentic συστήματος, έχοντας υπολογίσει το μέσο όρο για κάθε μετρική επί του συνόλου των ερωτήσεων:



Εικόνα 13 Συνολική Απόδοση Agentic Συστήματος

Από τις διαθέσιμες εγγραφές προκύπτει ότι το σύστημα χειρίστηκε ορθά τα προφανώς άσχετα ερωτήματα. Για παράδειγμα, στην περίπτωση agentic_003, με ερώτημα «I want some pizza», το σύστημα επιστρέφει μήνυμα ότι το ερώτημα δεν σχετίζεται με SQLi και ζητά σχετικό ερώτημα για περαιτέρω βοήθεια. Δεν ενεργοποιείται μηχανισμός ανάκτησης, γεγονός που συμφωνεί με τον ρόλο του Relevance Checker agent και αποτρέπει άσκοπη κατανάλωση πόρων. Η περίπτωση αυτή αξιολογήθηκε θετικά ως προς τις μετρικές «relevance», «refusal» και «tool use».

Θετική είναι επίσης η συμπεριφορά σε ερωτήματα εννοιολογικής γνώσης, όπου απαιτείται διανυσματική ανάκτηση. Στην περίπτωση agentic_007, το σύστημα απάντησε στο ερώτημα για τις διαφορές SQLi και XSS, διαχωρίζοντας σωστά τον στόχο των δύο επιθέσεων, η SQLi στοχεύει το επίπεδο της βάσης δεδομένων, ενώ η XSS το περιβάλλον εκτέλεσης του περιηγητή (browser) του χρήστη. Η αξιολόγηση κατέγραψε υψηλές τιμές στις μετρικές «relevance», «groundedness», «tool use» και «refusal», ενώ η αξιολόγηση στη μετρική «retrieval quality» εμφανίζει ελαφρώς χαμηλότερη βαθμολογία, καθώς η ανάκτηση των δεδομένων περιέχει και κάποια στοιχεία που δεν ήταν συναφή στην ερώτηση, χωρίς όπως να επηρεάσουν σημαντικά την τελική απάντηση του συστήματος.

Ενδιαφέρον παρουσιάζουν επίσης τα ερωτήματα που απαιτούν συνδυασμό γραφικής και σημασιολογικής γνώσης. Στην περίπτωση agentic_004, το σύστημα ανέδειξε HTTP artifacts συμβατά με time-based SQLi στο capture-005.pcap, αναφέροντας ότι οι παρατηρούμενες τιμές παραμέτρων περιέχουν τη συνάρτηση SLEEP, η οποία συνδέεται με τεχνικές σκόπιμης καθυστέρησης απόκρισης. Ο γράφος παρείχε συγκεκριμένα χαρακτηριστικά μοτίβα της συγκεκριμένης SQLi τεχνικής από αιτήματα HTTP, τιμές της παραμέτρου «id» και το όνομα του αντίστοιχου αρχείου PCAP, ενώ η διανυσματική βάση παρείχε το θεωρητικό πλαίσιο που συνδέει τις συναρτήσεις SLEEP και BENCHMARK με time-based SQLi τεχνικές. Η περίπτωση αυτή είναι ενδεικτική της επιδιωκόμενης λειτουργίας HybridRAG, ο γράφος γνώσης απαντά στο ερώτημα «πού παρατηρήθηκε το στοιχείο», ενώ η διανυσματική βάση υποστηρίζει την τεχνική του ερμηνεία.

Η hard-negative περίπτωση agentic_009 είναι επίσης πολύ σημαντική για τον έλεγχο ψευδώς θετικών αποτελεσμάτων. Στο capture-012.pcap ανακτήθηκε τιμή χρήστη O'Reilly, η οποία περιλαμβάνει απόστροφο. Το σύστημα δεν θεώρησε το μεμονωμένο αυτό στοιχείο επαρκή ένδειξη SQLi και κατέληξε ότι δεν υπάρχει τεκμηριωμένη απόδειξη επίθεσης. Η αιτιολόγηση

συνδέει τη διαπίστωση αυτή με την απουσία άλλων μοτίβων, όπως UNION, SELECT, DROP ή λογικών εκφράσεων τύπου OR 1=1. Η συμπεριφορά αυτή είναι ουσιώδης, επειδή αποδεικνύει ότι η αρχιτεκτονική δεν αντιμετωπίζει κάθε ειδικό χαρακτήρα ως κακόβουλο payload, αλλά αξιολογεί το συνολικό πλαίσιο του αιτήματος HTTP.

Στην περίπτωση agentic_010, η οποία αφορά την ανάλυση του capture-018.pcap, η τελική απάντηση κρίθηκε πλήρως επιτυχής ως προς τις μετρικές «relevance», «groundedness» και «retrieval quality». Το σύστημα ανέδειξε στοιχεία του γράφου γνώσης που είναι συμβατά με SQL injection, όπως τιμές της παραμέτρου id που περιείχαν το μοτίβο UNION ALL SELECT και άλλες SQL συντακτικές δομές. Ωστόσο, ως προς τη μετρική «tool use» έλαβε μερική βαθμολογία 0,5, επειδή ενεργοποίησε το kg_search_tool, το οποίο είχε οριστεί ως αποδεκτή εναλλακτική επιλογή, αντί του αναμενόμενου hybrid_search_tool. Κατά συνέπεια, η απάντηση βασίστηκε αποκλειστικά στα πραγματικά τεχνικά τεκμήρια που ανακτήθηκαν από το γράφο γνώσης και στην εσωτερική γνώση του γλωσσικού μοντέλου για την ερμηνεία των παρατηρούμενων SQL μοτίβων, χωρίς να ανακτήσει συμπληρωματικό θεωρητικό πλαίσιο από τη διανυσματική βάση. Το αποτέλεσμα αυτό δείχνει ότι η γραφική ανάκτηση μπορεί να είναι επαρκής για την τεκμηρίωση μιας σαφώς παρατηρούμενης επίθεσης, αλλά ταυτόχρονα αναδεικνύει απόκλιση μεταξύ της επιλεγμένης στρατηγικής ανάκτησης και της επιθυμητής υβριδικής αξιοποίησης των δύο πηγών γνώσης.

Συνολικά, η αξιολόγηση του Agentic RAG υποσυστήματος δείχνει ότι το προτεινόμενο σύστημα μπορεί να χειρίζεται ερωτήματα γενικής γνώσης, ερωτήματα που αφορούν τα εισαγμένα δεδομένα δικτυακής κίνησης, άσχετα ερωτήματα και hard-negative περιπτώσεις. Οι υψηλές βαθμολογίες στις μετρικές συνάφειας, τεκμηρίωσης και ποιότητας ανάκτησης υποδεικνύουν ότι, στις εξεταζόμενες περιπτώσεις, το σύστημα κατάφερε να παράγει απαντήσεις που σχετίζονται με το ερώτημα και στηρίζονται στα διαθέσιμα τεχνικά τεκμήρια. Παράλληλα, οι αποκλίσεις που καταγράφηκαν στη μετρική «tool use» δείχνουν ότι η τελική ορθότητα μιας απάντησης δεν συνεπάγεται πάντοτε την επιλογή της επιθυμητής στρατηγικής ανάκτησης. Ειδικότερα, σε περιπτώσεις όπου ο γράφος γνώσης επιστρέφει επαρκή και σαφή στοιχεία, ο RAG agent μπορεί να επιλέξει αποκλειστικά γραφική ανάκτηση αντί της αναμενόμενης υβριδικής προσέγγισης. Η συμπεριφορά αυτή δεν υποβαθμίζει απαραίτητα την ποιότητα της τελικής απάντησης, αλλά αναδεικνύει την ανάγκη για περαιτέρω βελτίωση των κανόνων δρομολόγησης και επιλογής εργαλείων, ώστε η αξιοποίηση της διανυσματικής βάσης

να ενεργοποιείται συστηματικότερα όταν απαιτείται πρόσθετη θεωρητική ερμηνεία ή πληρέστερη κατηγοριοποίηση των ευρημάτων SQLi.

5.4 Αξιολόγηση Αναγνώρισης και Ταξινόμησης SQL Injection

Η δεύτερη πειραματική διαδικασία αξιολογεί το σύστημα ως προς την αναγνώριση και ταξινόμηση HTTP artifacts και δικτυακής δραστηριότητας συμβατής με SQL injection σε επίπεδο PCAP. Σε αντίθεση με την προηγούμενη αξιολόγηση, η οποία εξετάζει ευρύτερα τη συμπεριφορά του Agentic RAG συστήματος, εδώ το ερώτημα είναι τυποποιημένο και ζητά τρία πεδία, ταξινόμηση benign ή sqli σύμφωνα με το σχήμα αξιολόγησης, τύπο πιθανής SQLi δραστηριότητας εφόσον αναδεικνύονται σχετικά artifacts και τεχνική αιτιολόγηση βασισμένη στα ανακτημένα δεδομένα. Επίσης αξιολογείται και η επεξήγηση του συστήματος με βάση το ανακτημένο πλαίσιο, καθώς μεγάλης σημασίας είναι όχι μόνο η σωστή ταξινόμηση, αλλά και η σωστά τεκμηριωμένη ταξινόμηση.

Η αξιολόγηση πραγματοποιήθηκε με βάση την ground-truth επισήμανση που ορίστηκε κατά τη δημιουργία του εργαστηριακού συνόλου PCAP αρχείων. Στην παρακάτω εικόνα (Εικόνα 14) φαίνονται το πως έχει αξιολογήσει το LLM-as-a-Judge τα αποτελέσματα του συστήματος επί των δεκαοχτώ διαφορετικών περιπτώσεων στις 3 διαφορετικές μετρικές:

	Inputs	attack_type_a... 0.83 AVG	detec... 1.00 AVG	justif... 0.97 AVG
1	Analyze capture-015.pcap. Return exactly three fields: Det... tion (benign or sqli), Attack type (or null if benign), and justif	1.00	1.00	0.50
2	Analyze capture-013.pcap. Return exactly three fields: Det... tion (benign or sqli), Attack type (or null if benign), and justif	1.00	1.00	1.00
3	Analyze capture-001.pcap. Return exactly three fields: Det... tion (benign or sqli), Attack type (or null if benign), and justif	1.00	1.00	1.00
4	Analyze capture-014.pcap. Return exactly three fields: Det... tion (benign or sqli), Attack type (or null if benign), and justif	1.00	1.00	1.00
5	Analyze capture-011.pcap. Return exactly three fields: Dete... ion (benign or sqli), Attack type (or null if benign), and justify	1.00	1.00	1.00
6	Analyze capture-012.pcap. Return exactly three fields: Det... tion (benign or sqli), Attack type (or null if benign), and justif	1.00	1.00	1.00
7	Analyze capture-008.pcap. Return exactly three fields: Det... ction (benign or sqli), Attack type (or null if benign), and justi	1.00	1.00	1.00
8	Analyze capture-018.pcap. Return exactly three fields: Det... tion (benign or sqli), Attack type (or null if benign), and justif	1.00	1.00	1.00
9	Analyze capture-006.pcap. Return exactly three fields: Det... ction (benign or sqli), Attack type (or null if benign), and justi	1.00	1.00	1.00
10	Analyze capture-016.pcap. Return exactly three fields: Det... tion (benign or sqli), Attack type (or null if benign), and justif	1.00	1.00	1.00
☐	Analyze capture-009.pcap. Return exactly three fields: Det... ction (benign or sqli), Attack type (or null if benign), and justi	1.00	1.00	1.00
12	Analyze capture-003.pcap. Return exactly three fields: Det... ction (benign or sqli), Attack type (or null if benign), and justi	0.00	1.00	1.00
13	Analyze capture-002.pcap. Return exactly three fields: Det... ction (benign or sqli), Attack type (or null if benign), and justi	1.00	1.00	1.00
14	Analyze capture-017.pcap. Return exactly three fields: Dete... tion (benign or sqli), Attack type (or null if benign), and justif	1.00	1.00	1.00
15	Analyze capture-010.pcap. Return exactly three fields: Det... tion (benign or sqli), Attack type (or null if benign), and justif	1.00	1.00	1.00
16	Analyze capture-004.pcap. Return exactly three fields: Det... ction (benign or sqli), Attack type (or null if benign), and justi	0.00	1.00	1.00
17	Analyze capture-007.pcap. Return exactly three fields: Det... tion (benign or sqli), Attack type (or null if benign), and justif	1.00	1.00	1.00
18	Analyze capture-005.pcap. Return exactly three fields: Det... ction (benign or sqli), Attack type (or null if benign), and justi	0.00	1.00	1.00

Εικόνα 14 Αξιολόγηση Ανίχνευσης και Ταξινόμησης SQLi Επιθέσεων

Ενώ παρακάτω στην Εικόνα 15, φαίνεται η συνολική απόδοση του Agentic συστήματος έχοντας υπολογίσει το μέσο όρο για κάθε μετρική επί του συνόλου των ερωτήσεων:

Εικόνα 15 Συνολική Απόδοση Συστήματος στην Αναγνώριση και Ταξινόμηση SQLi Επιθέσεων

5.4.1 Συγκεντρωτική Ερμηνεία Αποτελεσμάτων SQLi Ανίχνευσης και Ταξινόμησης

Τα συγκεντρωτικά αποτελέσματα του συγκεκριμένου ελεγχόμενου πειραματικού συνόλου δείχνουν μέση βαθμολογία **1,00** στη μετρική «detection correctness», **0,83** στη μετρική «attack type accuracy» και **0,97** στη μετρική «justification quality». Ειδικότερα, στα δεκαοκτώ εξεταζόμενα αρχεία PCAP, το σύστημα ταξινόμησε ορθά όλες τις περιπτώσεις ως benign ή SQLi. Δεν καταγράφηκε περίπτωση benign PCAP που να ταξινομήθηκε ως SQLi, ούτε περίπτωση PCAP με SQL injection που να ταξινομήθηκε ως benign. Η ταξινόμηση αυτή εκφράζει τη συμφωνία της απόκρισης του συστήματος με την εργαστηριακή ground truth του συγκεκριμένου συνόλου αξιολόγησης. Δεν ισοδυναμεί, με βάση μόνο τα στοιχεία των αρχείων PCAP, με ανεξάρτητη επιβεβαίωση επιτυχούς εκμετάλλευσης της ευπάθειας. Επιπρόσθετα, το εύρημα αυτό αφορά αποκλειστικά το συγκεκριμένο εργαστηριακό σύνολο αξιολόγησης και δεν συνιστά γενικευμένη εκτίμηση της απόδοσης του συστήματος σε παραγωγική ή μη ελεγχόμενη δικτυακή κίνηση.

Η χαμηλότερη, αλλά ακόμη υψηλή, επίδοση στη μετρική «attack type accuracy» δείχνει ότι η κύρια δυσκολία δεν αφορά την αναγνώριση της ύπαρξης κακόβουλης δραστηριότητας, αλλά την πλήρη και ακριβή κατηγοριοποίηση του τύπου SQL injection. Σε ορισμένες περιπτώσεις το σύστημα αναγνώρισε σωστά την παρουσία ενδείξεων SQLi, αλλά επέλεξε γενικότερο χαρακτηρισμό, όπως «multi-technique», ή ανέφερε επιπλέον τεχνικές πέραν της κυρίαρχης

ετικέτας αναφοράς. Η συμπεριφορά αυτή παρατηρήθηκε σε αρχεία PCAP που περιέχουν μεγάλο αριθμό αιτημάτων HTTP και συνδυασμό διαφορετικών μοτίβων payload, ιδιαίτερα όταν η ανάκτηση από τον γράφο επιστρέφει περισσότερα από ένα σχετικά artifacts.

Η μέση βαθμολογία 0,97 στη μετρική «justification quality» υποδεικνύει ότι οι απαντήσεις του συστήματος συνδέθηκαν στις περισσότερες περιπτώσεις με συγκεκριμένα στοιχεία του ανακτημένου πλαισίου, όπως τιμές παραμέτρων id, URI, SQL συναρτήσεις, λογικές εκφράσεις και λέξεις-κλειδιά της SQL. Επομένως, η τελική απόφαση δεν περιορίστηκε σε έναν αδιαφανή χαρακτηρισμό benign ή sqli, αλλά συνοδεύτηκε από τεχνική αιτιολόγηση βασισμένη σε παρατηρούμενα HTTP artifacts.

5.4.2 Αξιολόγηση Δυαδικής Ανίχνευσης

Πέραν της αξιολόγησης από το LLM-as-a-Judge, υπολογίστηκαν οι κλασικές μετρικές ταξινόμησης για τη δυαδική διάκριση μεταξύ benign και sqli. Στην παρούσα ανάλυση, ως θετική κλάση ορίζεται η παρουσία δραστηριότητας SQL injection. Από τα δεκαοκτώ αρχεία PCAP, τα δώδεκα έχουν ground-truth ετικέτα SQLi και τα έξι ετικέτα benign.

Πραγματική κλάση / Πρόβλεψη	SQLi	Benign
SQLi	12	0
Benign	0	6

Πίνακας 6 Πίνακας Σύγχυσης για την Αναγνώριση SQLi σε Επίπεδο PCAP

Με βάση τον παραπάνω πίνακα (Πίνακας 6), το σύστημα παρήγαγε 12 αληθώς θετικές προβλέψεις (*True Positives*), 6 αληθώς αρνητικές προβλέψεις (*True Negatives*), χωρίς ψευδώς θετικά (*False Positives*) ή ψευδώς αρνητικά (*False Negatives*) αποτελέσματα. Στον πίνακα αυτό, ως θετική κλάση ορίζεται η εργαστηριακή επισήμανση του κάθε αρχείου PCAP που περιέχει σενάριο SQLi. Η συγκεκριμένη κατηγοριοποίηση χρησιμοποιείται για την αξιολόγηση της αναγνώρισης HTTP artifacts συμβατών με SQL injection και όχι ως άμεση απόδειξη επιτυχούς εκτέλεσης κακόβουλων SQL ερωτημάτων.

Οι αντίστοιχες μετρικές δυαδικής ταξινόμησης παρουσιάζονται στον επόμενο πίνακα (Πίνακας 7):

Μετρική	Τιμή
Accuracy	1,00
Precision	1,00
Recall	1,00
F1-score	1,00

Πίνακας 7 Αποτελέσματα Μετρικών Δυαδικής Ταξινόμησης

Η ακρίβεια (*precision*) εκφράζει το ποσοστό των αρχείων PCAP που ταξινομήθηκαν ως SQLi και πράγματι περιείχαν SQLi δραστηριότητα. Η ανάκληση (*recall*) εκφράζει το ποσοστό των πραγματικών SQLi PCAP που αναγνωρίστηκαν επιτυχώς από το σύστημα. Τέλος, το F1-score αποτελεί τον αρμονικό μέσο precision και recall και χρησιμοποιείται ως ενιαία μέτρηση της ισορροπίας μεταξύ των δύο αυτών μετρικών. Στο συγκεκριμένο πείραμα, οι τιμές 1,00 στις μετρικές accuracy, precision, recall και F1-score δείχνουν ότι το σύστημα ταξινόμησε σωστά όλες τις εξεταζόμενες περιπτώσεις του συνόλου αξιολόγησης ως benign ή SQLi. Το αποτέλεσμα αυτό αποτυπώνει την απόδοση της προτεινόμενης ροής στο συγκεκριμένο ελεγχόμενο εργαστηριακό σύνολο και δεν πρέπει να ερμηνεύεται ως γενικευμένη μέτρηση της ικανότητας ανίχνευσης πιθανής δραστηριότητας SQL injection σε διαφορετικά ή παραγωγικά περιβάλλοντα.

Τα παραπάνω αποτελέσματα πρέπει να ερμηνεύονται με προσοχή. Το σύνολο αξιολόγησης αποτελείται από δεκαοκτώ PCAP που δημιουργήθηκαν σε ελεγχόμενο εργαστηριακό περιβάλλον και περιλαμβάνουν γνωστές μορφές benign και SQLi δραστηριότητας. Επομένως, οι τιμές αυτές δείχνουν ότι η προτεινόμενη ροή λειτούργησε επιτυχώς στις συγκεκριμένες πειραματικές συνθήκες, αλλά δεν αποτελούν γενικεύσιμη απόδειξη απόδοσης ή υπεροχής έναντι παραγωγικών IDS, WAF ή εξειδικευμένων μηχανισμών ανίχνευσης SQLi.

5.4.3 Ανάλυση Χαρακτηριστικών Περιπτώσεων

Η εξέταση επιμέρους περιπτώσεων επιβεβαιώνει τη σύνδεση της τελικής ταξινόμησης με τα τεχνικά στοιχεία του γράφου γνώσης. Στην benign περίπτωση capture-011.pcap, το σύστημα επέστρεψε Detection: benign και Attack type: null, αναφέροντας ότι η αναζήτηση στον γράφο

γνώσης δεν ανέδειξε παραμέτρους με γνωστά μοτίβα SQL injection. Η απόκριση αυτή συμφωνεί με την ground-truth ετικέτα και έλαβε πλήρη βαθμολογία και στις τρεις μετρικές.

Αξιοσημείωτη είναι η περίπτωση capture-012.pcap, η οποία λειτουργεί ως hard-negative σενάριο. Το PCAP περιλαμβάνει username με τιμή O'Reilly, δηλαδή ειδικό χαρακτήρα που μπορεί επιφανειακά να συσχετιστεί με SQL injection. Το σύστημα όμως, δεν ταξινόμησε την κίνηση ως SQLi, καθώς η ανάκτηση δεν έδειξε συμπληρωματικά μοτίβα όπως UNION SELECT, EXTRACTVALUE, SLEEP, OR NOT ή άλλες ενδείξεις απόπειρας SQLi . Η σωστή benign απόφαση δείχνει ότι το σύστημα δεν βασίζεται αποκλειστικά στην εμφάνιση ειδικών χαρακτήρων, αλλά αξιολογεί το ευρύτερο πλαίσιο των παραμέτρων HTTP.

Στην περίπτωση capture-015.pcap, το σύστημα ταξινόμησε το capture ως SQLi και απέδωσε τα παρατηρούμενα artifacts σε union-based δραστηριότητα, με βάση την παρουσία του μοτίβου UNION ALL SELECT στην τιμή παραμέτρου HTTP, το οποίο αντιστοιχεί άμεσα σε τεχνική συνένωσης αποτελεσμάτων SQL queries. Η reference label αποδέχεται είτε τον χαρακτηρισμό union-based είτε τον χαρακτηρισμό multi-technique, λόγω της παρουσίας πρόσθετων δοκιμών στο PCAP. Επομένως, η συγκεκριμένη απάντηση αξιολογήθηκε ως πλήρως ορθή, καθώς αναγνώρισε έναν αποδεκτό τύπο επίθεσης και τον συνέδεσε με συγκεκριμένο payload artifact.

Αντίθετα, η περίπτωση capture-005.pcap αναδεικνύει τον λόγο για τον οποίο η μέση επίδοση στην μετρική «attack type accuracy» είναι χαμηλότερη από την επίδοση στη βασική αναγνώριση. Η ground-truth επισήμανση περιγράφει time-based blind SQLi, με βασικό τεχνικό στοιχείο τη συνάρτηση SLEEP. Το σύστημα ανέδειξε στοιχεία συμβατά με SQLi και αναφέρθηκε σε time-based μοτίβα, όπως η συνάρτηση SLEEP, αλλά πρόσθεσε και union-based χαρακτηρισμό, επιστρέφοντας τελικά τον γενικότερο τύπο multi-technique. Η απόφαση αναγνώρισης παρέμεινε ορθή, όμως η επιπλέον κατηγοριοποίηση δεν αντιστοιχούσε πλήρως στην ετικέτα αναφοράς της συγκεκριμένης περίπτωσης. Επομένως, η περίπτωση αυτή επηρεάζει κυρίως τη μετρική «attack type accuracy» και δευτερευόντως την ποιότητα της αιτιολόγησης.

Συνολικά, η αξιολόγηση δείχνει ότι στο συγκεκριμένο ελεγχόμενο πειραματικό σύνολο, η προτεινόμενη αρχιτεκτονική ταξινόμησε ορθά τις εξεταζόμενες περιπτώσεις benign και SQLi δικτυακής κίνησης και συνόδευσε τις αποφάσεις της με τεχνικά τεκμήρια από τα HTTP artifacts του γράφου γνώσης. Το εύρημα αυτό αφορά τις συνθήκες, τις τεχνικές SQLi και τα

δεδομένα που χρησιμοποιήθηκαν στην παρούσα αξιολόγηση. Η κύρια περιοχή περαιτέρω βελτίωσης αφορά την πιο ακριβή επιλογή και παρουσίαση του τύπου επίθεσης SQLi, ιδίως σε PCAP με πολλαπλές τεχνικές ή μεγάλο όγκο payloads. Σε αυτές τις περιπτώσεις, η ενίσχυση της ανάκτησης από το γράφο γνώσης με μηχανισμούς ομαδοποίησης ανά τεχνική, συχνότητα εμφάνισης ή χρονική ακολουθία αιτημάτων θα μπορούσε να βελτιώσει τη διάκριση μεταξύ της κυρίαρχης τεχνικής και της γενικότερης «multi-technique» δραστηριότητας.

5.5 Συζήτηση Αποτελεσμάτων

Η πειραματική αξιολόγηση δείχνει ότι η προτεινόμενη Agentic HybridRAG αρχιτεκτονική λειτούργησε σύμφωνα με τον σχεδιασμό της στο ελεγχόμενο σύνολο δεδομένων από αρχεία PCAP. Ειδικότερα, ως προς τη βασική δυαδική διάκριση μεταξύ benign και SQLi δραστηριότητας, το σύστημα ταξινόμησε σωστά και τις δεκαοκτώ εξεταζόμενες περιπτώσεις, χωρίς να καταγραφούν ψευδώς θετικές ή ψευδώς αρνητικές ταξινομήσεις στο συγκεκριμένο σύνολο αξιολόγησης. Η επίδοση αυτή αφορά αποκλειστικά το περιορισμένο, εργαστηριακά παραγόμενο σύνολο δεδομένων και δεν επιτρέπει συμπεράσματα για την αναμενόμενη απόδοση της αρχιτεκτονικής σε ευρύτερα ή παραγωγικά περιβάλλοντα.

Επιπλέον, η υψηλή μέση βαθμολογία στη μετρική ποιότητας αιτιολόγησης δείχνει ότι οι αποφάσεις δεν περιορίστηκαν σε έναν απλό χαρακτηρισμό της κίνησης, αλλά συνδέθηκαν στις περισσότερες περιπτώσεις, με συγκεκριμένα τεχνικά στοιχεία που ανακτήθηκαν από τον γράφο γνώσης.

Θετικό εύρημα αποτελεί η ορθή διαχείριση των hard-negative περιπτώσεων. Το σύστημα δεν ταξινόμησε ως SQLi την παρουσία ενός μεμονωμένου ειδικού χαρακτήρα, όπως η απόστροφος στην τιμή «O'Reilly», όταν δεν υπήρχαν πρόσθετες ενδείξεις χειρισμού SQLi. Η συμπεριφορά αυτή υποδεικνύει ότι η τελική απόφαση δεν βασίζεται αποκλειστικά σε επιφανειακή αντιστοίχιση λέξεων-κλειδιών ή χαρακτήρων, αλλά λαμβάνει υπόψη το ευρύτερο πλαίσιο του HTTP αιτήματος και τα διαθέσιμα τεκμήρια από τον γράφο γνώσης.

Η αξιοποίηση του γράφου γνώσης αποδείχθηκε σημαντική για την ανάδειξη πραγματικών τεχνικών artifacts από την καταγεγραμμένη επικοινωνία HTTP. Το σύστημα μπορεί να συσχετίζει το αρχείο PCAP, το αίτημα HTTP, τις διευθύνσεις IP, τον host, τη μέθοδο HTTP,

το URI, τη χρονική σήμανση και τις επιμέρους παραμέτρους του αιτήματος. Έτσι, η παραγόμενη απάντηση μπορεί να τεκμηριώνεται με βάση τις πραγματικές τιμές των παραμέτρων που εμφανίζονται στην κίνηση, όπως εκφράσεις UNION ALL SELECT, συναρτήσεις EXTRACTVALUE ή SLEEP, αντί να στηρίζεται αποκλειστικά στη γενική γνώση του γλωσσικού μοντέλου.

Παράλληλα, η αξιολόγηση της ροής εργασίας του Agentic RAG έδειξε ότι το σύστημα μπορεί να διαχειρίζεται τόσο ερωτήματα γενικής γνώσης όσο και ερωτήματα διερεύνησης της εισαγμένης δικτυακής κίνησης, καθώς και να απορρίπτει ερωτήματα που βρίσκονται εκτός του πεδίου του συστήματος. Η διάκριση ρόλων μεταξύ του Relevance Checker agent, του RAG agent, του Research agent και του Verification agent συνέβαλε στην ελεγχόμενη παραγωγή απαντήσεων, γεγονός το οποίο διαπιστώθηκε και από την επιθεώρηση των ενδιάμεσων σταδίων της ροής μέσω των traces του LangSmith.

Η κύρια περιοχή βελτίωσης αφορά την ακριβέστερη ταξινόμηση του τύπου της πιθανής επίθεσης SQLi, όπου η μέση βαθμολογία ήταν χαμηλότερη από εκείνη της βασικής αναγνώρισης. Σε PCAP που περιλαμβάνουν μεγάλο αριθμό αιτημάτων ή πολλαπλά μοτίβα payload, το σύστημα μπορεί να αναγνωρίσει ορθά ότι υπάρχει SQL injection, αλλά να επιλέξει έναν γενικότερο χαρακτηρισμό, όπως «multi-technique», ή να συμπεριλάβει επιπλέον τεχνικές πέραν της κυρίαρχης. Επιπλέον, σε ορισμένες περιπτώσεις ο RAG agent αξιοποίησε αποκλειστικά το kg_search_tool, επειδή τα διαθέσιμα στοιχεία της Neo4j επαρκούσαν για την τεκμηρίωση της απάντησης, αντί να χρησιμοποιήσει το αναμενόμενο hybrid_search_tool. Η επιλογή αυτή δεν επηρέασε απαραίτητα την ορθότητα της τελικής απόκρισης, αλλά περιόρισε τη συστηματική αξιοποίηση της θεωρητικής γνώσης που είναι αποθηκευμένη στη διανυσματική βάση δεδομένων.

Συνεπώς, η μελλοντική βελτίωση της αρχιτεκτονικής μπορεί να εστιάσει σε πιο αυστηρούς κανόνες δρομολόγησης προς την υβριδική ανάκτηση, σε μηχανισμούς ομαδοποίησης των ευρημάτων ανά τεχνική SQLi και σε στρατηγικές που διακρίνουν την κυρίαρχη από τη δευτερεύουσα ή συμπληρωματική επιθετική δραστηριότητα. Παρά τους περιορισμούς του μικρού και ελεγχόμενου εργαστηριακού συνόλου PCAP, τα αποτελέσματα δείχνουν ότι η προτεινόμενη αρχιτεκτονική μπορεί να λειτουργήσει ως χρήσιμο εργαλείο υποστήριξης αναλυτών ασφαλείας, συνδυάζοντας την αναζήτηση τεχνικών τεκμηρίων, τη συσχέτιση

δεδομένων και την παραγωγή επεξηγηματικών απαντήσεων για την διερεύνηση πιθανών περιστατικών SQL injection.

6. Συμπεράσματα & Μελλοντικές Προεκτάσεις

Η πειραματική διαδικασία και η ανάλυση των αποτελεσμάτων του προηγούμενου κεφαλαίου ανέδειξαν τον τρόπο με τον οποίο η προτεινόμενη αρχιτεκτονική μπορεί να υποστηρίξει την αναγνώριση και ερμηνεία πιθανών επιθέσεων SQL injection. Τα αποτελέσματα έδειξαν ότι ο συνδυασμός γράφου γνώσης, διανυσματικής ανάκτησης και συστήματος πολλαπλών agents μπορεί να συνδέσει παρατηρούμενα HTTP artifacts με εξειδικευμένη τεχνική γνώση και να παράγει τεκμηριωμένες απαντήσεις.

Στο παρόν κεφάλαιο συνοψίζονται τα βασικά συμπεράσματα που προκύπτουν από τον σχεδιασμό, την υλοποίηση και την αξιολόγηση του συστήματος. Παράλληλα, εξετάζονται οι περιορισμοί της προσέγγισης, όπως το ελεγχόμενο και περιορισμένης κλίμακας σύνολο δεδομένων, η χρήση μη κρυπτογραφημένης HTTP κίνησης και η εξάρτηση από τη διαθέσιμη γνώση και το γλωσσικό μοντέλο. Τέλος, παρουσιάζονται προτάσεις για μελλοντική έρευνα και επέκταση της αρχιτεκτονικής σε ευρύτερα σύνολα δεδομένων, πρόσθετες κατηγορίες επιθέσεων και περισσότερο ρεαλιστικά περιβάλλοντα λειτουργίας.

6.1 Κύρια Συμπεράσματα

Η παρούσα εργασία διερεύνησε την εφαρμογή μιας υβριδικής αρχιτεκτονικής για την υποστήριξη της ανάλυσης ενδείξεων πιθανής δραστηριότητας SQL injection σε δεδομένα δικτυακής κίνησης. Η προτεινόμενη προσέγγιση συνδυάζει ανάλυση αρχείων PCAP, γράφο γνώσης στη Neo4j, διανυσματική βάση γνώσης Qdrant, τεχνικές Retrieval-Augmented Generation και ενορχήστρωση πολλαπλών agents με χρήση LangGraph.

Η υλοποίηση έδειξε ότι η μετατροπή στοιχείων HTTP από αρχεία PCAP σε γράφο γνώσης επιτρέπει τη δομημένη αναπαράσταση των τεχνικών artifacts που σχετίζονται με πιθανές επιθέσεις SQL injection. Τα στοιχεία αυτά περιλαμβάνουν μεθόδους HTTP, URI, full URI, query string, parameters, POST body τιμές, cookies, διευθύνσεις IP, θύρες, hosts και χρονικές σημάνσεις. Μέσω της Neo4j, η πληροφορία αυτή δεν διατηρείται μόνο ως ακολουθία πακέτων, αλλά ως σύνολο συσχετισμένων οντοτήτων, το οποίο μπορεί να διερευνηθεί με ερωτήματα Cypher και να αξιοποιηθεί από τον Agentic μηχανισμό.

Η Qdrant συμπληρώνει τη γραφική αναπαράσταση παρέχοντας τεχνική και θεωρητική γνώση για SQL injection. Το σύνολο εξειδικευμένης γνώσης της διανυσματικής βάσης περιλαμβάνει υλικό με payloads και τεχνικές SQLi, καθώς και περιεχόμενο που συλλέχθηκε από επιλεγμένες διαδικτυακές πηγές. Με τη χρήση αραιών και πυκνών διανυσμάτων, η Qdrant επιτρέπει την ανάκτηση γνώσης τόσο βάσει σημασιολογικής συνάφειας όσο και βάσει ακριβούς λεκτικής αντιστοίχισης τεχνικών όρων και payload μοτίβων.

Η βασική συμβολή της προτεινόμενης αρχιτεκτονικής είναι ότι συνδυάζει δύο διαφορετικές μορφές γνώσης. Η Neo4j απαντά στο ερώτημα «τι παρατηρήθηκε στη δικτυακή κίνηση;», ενώ η Qdrant υποστηρίζει το ερώτημα «πώς μπορεί να ερμηνευθεί τεχνικά το συγκεκριμένο artifact;». Το υποσύστημα πολλαπλών agents συνθέτει αυτές τις δύο πηγές ώστε να παράγει απαντήσεις, οι οποίες συνδέουν συγκεκριμένα στοιχεία HTTP με γνώση σχετική με τεχνικές SQL injection, πιθανές επιπτώσεις και προτεινόμενα μέτρα αντιμετώπισης.

Η αξιοποίηση αρχιτεκτονικής Agentic RAG με ενορχήστρωση πολλαπλών agents, επέτρεψε την κατανομή των επιμέρους λειτουργιών σε διακριτούς ρόλους. Ο Relevance Checker agent διαχειρίζεται τη θεματική συνάφεια του ερωτήματος, ο RAG agent επιλέγει τη στρατηγική ανάκτησης, ο Research agent συνθέτει την αρχική απάντηση και ο Verification agent ελέγχει την τεκμηρίωση της. Η οργάνωση αυτή συμβάλλει στην ελεγχόμενη παραγωγή απαντήσεων και περιορίζει τον κίνδυνο παραγωγής μη υποστηριζόμενων συμπερασμάτων.

Η δημιουργία ελεγχόμενου συνόλου δεδομένων PCAP αποδείχθηκε κατάλληλη για την αξιολόγηση της συνολικής ροής του συστήματος. Τα benign, hard-negative, manual SQLi και sqlmap-generated captures επέτρεψαν την εξέταση διαφορετικών ελεγχόμενων σεναρίων με HTTP artifacts συμβατά με Boolean-based, Error-based, Time-based και UNION-based SQL injection. Η χρήση ground-truth επισημάνσεων επέτρεψε την αξιολόγηση των αποτελεσμάτων σε επίπεδο συνεδρίας PCAP και όχι ως μεμονωμένων, μη συσχετισμένων πακέτων.

Συνολικά, τα αποτελέσματα της ελεγχόμενης αξιολόγησης του προτεινόμενου συστήματος, δείχνουν ότι ένας Agentic HybridRAG μηχανισμός μπορεί να λειτουργήσει ως εργαλείο υποστήριξης αναλυτών ασφαλείας για την αρχική διερεύνηση πιθανών ενδείξεων SQL injection. Η κύρια αξία του δεν βρίσκεται στην αντικατάσταση των παραδοσιακών μηχανισμών IDS, αλλά στην ενίσχυση της διερεύνησης περιστατικών μέσω της συσχέτισης δικτυακών ευρημάτων, οργανωμένης γνώσης και επεξηγηματικής φυσικής γλώσσας.

6.2 Περιορισμοί της Προσέγγισης

Η αξιολόγηση και η προτεινόμενη υλοποίηση παρουσιάζουν ορισμένους περιορισμούς, οι οποίοι πρέπει να λαμβάνονται υπόψη κατά την ερμηνεία των αποτελεσμάτων.

Πρώτον, το σύνολο αρχείων PCAP είναι μικρής κλίμακας και δημιουργήθηκε σε ελεγχόμενο εργαστηριακό περιβάλλον. Ως αποτέλεσμα, δεν καλύπτει όλη την ποικιλία πραγματικών εφαρμογών διαδικτύου, διαφορετικών frameworks, συστημάτων διαχείρισης βάσεων δεδομένων, μηχανισμών WAF, δικτυακών τοπολογιών ή τεχνικών απόκρυψης payloads που είναι δυνατόν να εμφανίζονται σε παραγωγικά περιβάλλοντα.

Επιπλέον, η αξιολόγηση βασίζεται σε ελεγχόμενα σενάρια SQL injection και σε ερωτήματα που καλύπτουν τις λειτουργίες για τις οποίες σχεδιάστηκε το πρωτότυπο. Κατά συνέπεια, τα αποτελέσματα δείχνουν κυρίως ότι η προτεινόμενη αρχιτεκτονική λειτουργεί στις συγκεκριμένες συνθήκες και για τις εξεταζόμενες μορφές HTTP artifacts και SQLi. Απαιτείται περαιτέρω αξιολόγηση με ανεξάρτητα και ετερογενή σύνολα δεδομένων, διαφορετικές εφαρμογές και επιπλέον τεχνικές επίθεσης, ώστε να μελετηθεί η δυνατότητα γενίκευσής της σε πραγματικά παραγωγικά περιβάλλοντα.

Δεύτερον, η DVWA αποτελεί εσκεμμένα ευάλωτη εφαρμογή εκπαίδευσης. Επομένως, η συμπεριφορά της, οι μηχανισμοί χειρισμού λαθών και τα μηνύματα σφάλματος SQL που μπορεί να επιστρέφει ενδέχεται να είναι περισσότερο εμφανή σε σύγκριση με πραγματικές εφαρμογές. Σε ένα παραγωγικό περιβάλλον ενδέχεται να χρησιμοποιούνται prepared statements, parameterized queries, input validation, μηχανισμοί WAF, περιορισμός μηνυμάτων σφάλματος και άλλες πρακτικές άμυνας.

Τρίτον, η ανάλυση βασίζεται σε κίνηση HTTP χωρίς κρυπτογράφηση TLS. Σε πραγματικές εφαρμογές, όπου η πλειονότητα της επικοινωνίας χρησιμοποιεί HTTPS, τα URI, τα headers και τα request bodies δεν είναι άμεσα διαθέσιμα σε ένα PCAP. Η αντίστοιχη ανάλυση θα απαιτούσε κατάλληλο σημείο τερματισμού TLS, πρόσβαση σε αποκρυπτογραφημένη κίνηση ή κατάλληλη διαχείριση session keys.

Τέταρτον, το σύνολο δεδομένων της Qdrant προέρχεται από επιλεγμένα αρχεία markdown και από στοχευμένο web crawling περιορισμένου βάθους. Παρότι περιλαμβάνει ευρύ φάσμα τεχνικών SQLi, μοτίβα αναλόγως DBMS, παραδειγμάτων payload και πληροφορία πρόληψης, δεν αποτελεί πλήρη ή εξαντλητική αναπαράσταση της διαθέσιμης γνώσης SQL injection. Η

ποιότητα της τελικής απάντησης επηρεάζεται από την ποιότητα και την κάλυψη των chunks, τον τρόπο τμηματοποίησης του περιεχομένου, το μοντέλο embeddings, τη στρατηγική υβριδικής ανάκτησης και τη συνάφεια της ανακτημένης γνώσης.

Πέμπτον, οι μετρικές ταξινόμησης προκύπτουν από αξιολόγηση σε επίπεδο συνεδρίας PCAP και όχι από μεγάλο, στατιστικά αντιπροσωπευτικό benchmark. Ως εκ τούτου, τα αποτελέσματα πρέπει να ερμηνεύονται ως ένδειξη ότι η προτεινόμενη αρχιτεκτονική λειτούργησε επιτυχώς στο συγκεκριμένο πειραματικό πλαίσιο και ως βάση για περαιτέρω αξιολόγηση σε ευρύτερα σύνολα δεδομένων. Επίσης, τα αποτελέσματα δεν αποτελούν γενικευμένη μέτρηση της απόδοσης ή της υπεροχής της έναντι υφιστάμενων IDS, WAF ή εξειδικευμένων μηχανισμών ταξινόμησης SQLi.

Έκτον, η ποιότητα του Agentic συστήματος επηρεάζεται από την επιλογή του LLM, τη διατύπωση των ερωτημάτων, τη μη ντετερμινιστική συμπεριφορά του μοντέλου και την επάρκεια των αποτελεσμάτων ανάκτησης. Για τον λόγο αυτό, η χρήση του LangSmith, των αναφορών επιβεβαίωσης και των επαναλαμβανόμενων ερωτημάτων αποτελεί ουσιώδη μηχανισμό παρακολούθησης, αποσφαλμάτωσης και τεκμηρίωσης της λειτουργίας του συστήματος.

Τέλος, η προτεινόμενη υλοποίηση αποτελεί σύστημα υποστήριξης διερεύνησης και όχι αυτόνομο μηχανισμό λήψης αποφάσεων ασφαλείας. Η τελική αξιολόγηση ενός ύποπτου περιστατικού, ιδίως όταν πρόκειται να οδηγήσει σε blocking, incident response ή άλλες επιχειρησιακές ενέργειες, πρέπει να παραμένει υπό την εποπτεία εξειδικευμένου αναλυτή ασφαλείας.

6.3 Προτάσεις για Μελλοντική Έρευνα

Η προτεινόμενη αρχιτεκτονική μπορεί να επεκταθεί σε διάφορες κατευθύνσεις, με στόχο την ενίσχυση της πληρότητας των διαθέσιμων τεκμηρίων, την αξιολόγηση σε ευρύτερα περιβάλλοντα και την υποστήριξη περισσότερων κατηγοριών επιθέσεων. Ορισμένες προτάσεις μελλοντικής έρευνας είναι οι κάτωθι:

- **Συσχέτιση επιπλέον πηγών δεδομένων για επιβεβαίωση επιτυχίας SQLi.** Άμεση κατεύθυνση μελλοντικής εργασίας αποτελεί η εισαγωγή και συσχέτιση επιπλέον δεδομένων, πέρα από τα αρχεία PCAP. Ενδεικτικά, μπορούν να αξιοποιηθούν αρχεία καταγραφών εξυπηρετητών και επιπέδου εφαρμογής, καταγραφές WAF, καθώς και

καταγραφές της βάσης δεδομένων της εφαρμογής. Η συνδυαστική ανάλυση αυτών των πηγών θα επέτρεπε στο σύστημα να μην αξιολογεί μόνο αν παρατηρήθηκε πιθανή απόπειρα SQL injection, αλλά και αν η απόπειρα ήταν επιτυχής, αν προκάλεσε SQL error, αν οδήγησε σε μη εξουσιοδοτημένη ανάκτηση δεδομένων ή αν δημιούργησε μεταβολές στη βάση δεδομένων. Η συσχέτιση θα μπορούσε να βασίζεται σε χρονικές σημάνσεις, διευθύνσεις IP, αναγνωριστικά συνεδρίας, παραμέτρους HTTP, χρήστες της βάσης δεδομένων και άλλους κοινούς δείκτες.

- **Επέκταση σε πρόσθετες κατηγορίες επιθέσεων εφαρμογών διαδικτύου.** Η αρχιτεκτονική μπορεί να επεκταθεί ώστε να υποστηρίζει την ανάλυση και άλλων επιθέσεων, όπως Cross-Site Scripting (XSS), command injection, path traversal, file inclusion, Server-Side Template Injection (SSTI) και authentication attacks. Η επέκταση αυτή απαιτεί εμπλουτισμό τόσο του συνόλου δεδομένων PCAP, όσο και της εξειδικευμένης γνώσης της Qdrant με σχετικό τεχνικό υλικό, payloads, παραδείγματα και τεκμηρίωση. Με αυτόν τον τρόπο, η εφαρμογή θα μπορούσε να εξελιχθεί από σύστημα επικεντρωμένο σε SQL injection σε ευρύτερο εργαλείο υποστήριξης ανάλυσης επιθέσεων σε εφαρμογές διαδικτύου.

- **Επέκταση και διαφοροποίηση του πειραματικού συνόλου δεδομένων.** Η συλλογή περισσότερων αρχείων PCAP από διαφορετικές εσκεμμένα ευάλωτες εφαρμογές, όπως OWASP Juice Shop και WebGoat, θα επέτρεπε την αξιολόγηση της αρχιτεκτονικής σε διαφορετικά endpoints, μεθόδους HTTP, μορφές εισόδου, τεχνολογίες backend και συστήματα διαχείρισης βάσεων δεδομένων. Επιπλέον, η χρήση επιπρόσθετων benign, hard-negative και mixed-activity σεναρίων θα ενίσχυε τη μελέτη false positives και false negatives.

- **Υποστήριξη ανάλυσης δικτυακής κίνησης HTTPS.** Η παρούσα αξιολόγηση βασίστηκε σε HTTP traffic χωρίς κρυπτογράφηση, ώστε τα στοιχεία επιπέδου εφαρμογής να είναι διαθέσιμα μέσα στα αρχεία PCAP. Μελλοντικά, η υλοποίηση μπορεί να προσαρμοστεί για χρήση σε περιβάλλοντα HTTPS μέσω συλλογής δεδομένων από reverse proxy, WAF, API gateway ή σημείο τερματισμού TLS. Η προσέγγιση αυτή θα επέτρεπε την ανάλυση ρεαλιστικότερων παραγωγικών συνθηκών, όπου η πλειονότητα της επικοινωνίας πραγματοποιείται με χρήση TLS.

- **Συγκριτική αξιολόγηση διαφορετικών επιλογών ανάκτησης και γλωσσικών μοντέλων (LLMs).** Μελλοντική μελέτη μπορεί να συγκρίνει διαφορετικά μοντέλα embeddings, στρατηγικές chunking, παραμετροποιήσεις ανάκτησης πυκνών/αραιών διανυσμάτων και τεχνικές συγχώνευσης (fusion) ή ανακατάταξης (reranking). Επιπλέον,

μπορούν να αξιολογηθούν διαφορετικά LLMs και διαφορετικές παραμετροποιήσεις της Agentic ροής εργασίας ως προς την ποιότητα ανάκτησης, τη πιστότητά των απαντήσεων, τον χρόνο απόκρισης και την ικανότητα τεκμηριωμένης ερμηνείας των ευρημάτων.

Οι παραπάνω κατευθύνσεις μπορούν να ενισχύσουν σταδιακά την προτεινόμενη αρχιτεκτονική, μετατρέποντάς την σε πιο ολοκληρωμένο και επεκτάσιμο μηχανισμό υποστήριξης αναλυτών ασφαλείας για τη διερεύνηση περιστατικών σε εφαρμογές διαδικτύου.

Βιβλιογραφία

Abdullayev, V., & Chauhan, A. S. (2023). SQL Injection Attack: Quick View. In *Mesopotamian Journal of CyberSecurity* (Vol. 2023, pp. 30–34). Mesopotamian Academic Press. https://doi.org/10.58496/MJCS/2023/006

Al-Olaqi, M., Al-Gailani, A., & Rahman, M. M. H. (2025). Comprehensive Study of SQL Injection Attacks Mitigation Methods and Future Directions. *Journal of Cyber Security and Risk Auditing*, *2025*(4), 347–365. https://doi.org/10.63180/jcsra.thestap.2025.4.11

Blefari, F., Cosentino, C., Pironti, F. A., Furfaro, A., & Marozzo, F. (2026). CyberRAG: An agentic RAG cyber attack classification and reporting tool. *Future Generation Computer Systems*, *176*. https://doi.org/10.1016/j.future.2025.108186

Building Effective AI Agents \ Anthropic. (n.d.). Retrieved May 31, 2026, from https://www.anthropic.com/engineering/building-effective-agents

Carbonell, J. (1999). *The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries*. https://doi.org/10.1145/290941.291025

Clarke, J. (2012). *SQL injection attacks and defense (2nd ed.). Syngress.*

Crawl4AI Documentation (v0.8.x). (n.d.). Retrieved June 4, 2026, from https://docs.crawl4ai.com/

CWE - CWE Top 25 Most Dangerous Software Weaknesses. (n.d.). Retrieved May 2, 2026, from https://cwe.mitre.org/top25/index.html

Docling. (n.d.). Retrieved June 4, 2026, from https://www.docling.ai/

Fui-Hoon Nah, F., Zheng, R., Cai, J., Siau, K., & Chen, L. (2023). Generative AI and ChatGPT: Applications, challenges, and AI-human collaboration. *Journal of Information Technology Case and Application Research*, *25*(3), 277–304. https://doi.org/10.1080/15228053.2023.2233814

Gao, S., Zhao, S., Jiang, X., Duan, L., Chng, Y. X., Chen, Q.-G., Luo, W., Zhang, K., Bian, J.-W., & Gong, M. (2026). Scaling Beyond Context: A Survey of Multimodal Retrieval-Augmented Generation for Document Understanding. In M. Liakata, V. P. Moreira, J. Zhang, & D. Jurgens (Eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 4458–4489). Association for Computational Linguistics. https://doi.org/10.18653/v1/2026.acl-long.204

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., & Wang, H. (2023). *Retrieval-augmented generation for large language models: A survey [Preprint]. arXiv.* https://doi.org/10.48550/arXiv.2312.10997

Huang, Y., & Huang, J. X. (2026). A Survey on Retrieval-Augmented Text Generation for Large Language Models. *ACM Comput. Surv.*, *58*(12). https://doi.org/10.1145/3805774

Hybrid Queries - Qdrant. (n.d.). Retrieved June 20, 2026, from https://qdrant.tech/documentation/search/hybrid-queries/

Jeong, C. (2026). *A hybrid RAG system integrating knowledge graph and vector retrieval: Based on solution technical documents [Preprint]. Research Square.* https://doi.org/10.21203/rs.3.rs-9476748/v1

Kaur, R., Gabrijelčič, D., & Klobučar, T. (2023). Artificial intelligence for cybersecurity: Literature review and future research directions. *Information Fusion*, *97*, 101804. https://doi.org/10.1016/J.INFFUS.2023.101804

Lee, C. H. (2019). *A study of out-of-band structured query language injection [Preprint]. Zenodo.* https://doi.org/10.5281/ZENODO.3556347

Ma, L., Zhang, R., Han, Y., Yu, S., Wang, Z., Ning, Z., Zhang, J., Xu, P., Li, P., Qiao, Z., Ju, W., Chen, C., Wang, D., Liu, K., Wang, P., Wang, P., Fu, Y., Liu, C., Zhou, Y., & Lu, C.-T. (2023). *A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge [Preprint]. arXiv.* https://doi.org/10.48550/arXiv.2310.11703

Malkov, Y. A., & Yashunin, D. A. (2020). Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, *42*(4), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

Mei, L., Mo, S., Yang, Z., & Chen, C. (2025). *A Survey of Multimodal Retrieval-Augmented Generation [Preprint]. ArXiv.* https://doi.org/10.48550/arXiv.2504.08748

Mohamed, N. (2025). Artificial intelligence and machine learning in cybersecurity: a deep dive into state-of-the-art techniques and future paradigms. *Knowledge and Information Systems*, *67*(8), 6969–7055. https://doi.org/10.1007/s10115-025-02429-y

Nasereddin, M., ALKhamaiseh, A., Qasaimeh, M., & Al-Qassas, R. (2021). A systematic review of detection and prevention techniques of SQL injection attacks. In *Information Security Journal: A Global Perspective* (Vol. 32, Number 4, pp. 252–265). Taylor and Francis Ltd. https://doi.org/10.1080/19393555.2021.1995537

Nourmohammadzadeh Motlagh, F., Hajizadeh, M., Majd, M., Najafi, P., Cheng, F., & Meinel, C. (2024). *Large Language Models in Cybersecurity: State-of-the-Art [Preprint]. ArXiv.* https://doi.org/10.48550/arXiv.2402.00891

OWASP Top 10:2025. (n.d.). Retrieved April 28, 2026, from https://owasp.org/Top10/2025/

OWASP Top Ten Web Application Security Risks | OWASP Foundation. (n.d.). Retrieved April 30, 2026, from https://owasp.org/www-project-top-ten/

Paul, A., Sharma, V., & Olukoya, O. (2024). SQL injection attack: Detection, prioritization & prevention. *Journal of Information Security and Applications*, *85*, 103871. https://doi.org/10.1016/J.JISA.2024.103871

Peng, B., Zhu, Y., Liu, Y., Bo, X., Shi, H., Hong, C., Zhang, Y., & Tang, S. (2026). Graph Retrieval-Augmented Generation: A Survey. *ACM Transactions on Information Systems*, *44*(2), 1–52. https://doi.org/10.1145/3777378

PyShark Usage Documentation. (n.d.). Retrieved August 2, 2026, from https://pyshark-packet-analysis.readthedocs.io/en/latest/

Qdrant integration - Docs by LangChain. (n.d.). Retrieved June 9, 2026, from https://docs.langchain.com/oss/python/integrations/vectorstores/qdrant#hybrid-vector-search

Researchers say a bug let them add fake pilots to rosters used for TSA checks | The Verge. (n.d.). Retrieved May 2, 2026, from

https://www.theverge.com/2024/9/8/24239026/airline-security-bug-tsa-security-database-sql-injection-hack

Sapkota, R., Roumeliotis, K. I., & Karkee, M. (2026). AI Agents vs. Agentic AI: A Conceptual taxonomy, applications and challenges. *Information Fusion*, *126*, 103599. https://doi.org/10.1016/j.inffus.2025.103599

Sarmah, B., Mehta, D., Hall, B., Rao, R., Patel, S., & Pasquali, S. (2024). HybridRAG: Integrating Knowledge Graphs and Vector Retrieval Augmented Generation for Efficient Information Extraction. *Proceedings of the 5th ACM International Conference on AI in Finance, ICAIF '24*, 608–616. https://doi.org/10.1145/3677052.3698671

Sharma, C. (2025). *Retrieval-Augmented Generation: A Comprehensive Survey of Architectures, Enhancements, and Robustness Frontiers [Preprint]. ArXiv.* https://doi.org/10.48550/arXiv.2506.00054

Singh, A., Ehtesham, A., Kumar, S., Khoei, T. T., & Vasilakos, A. V. (2025). *Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG [Preprint]. ArXiv.* https://doi.org/10.48550/arXiv.2501.09136

SQL Injection Prevention - OWASP Cheat Sheet Series. (n.d.). Retrieved May 2, 2026, from https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

swisskyrepo/PayloadsAllTheThings: A list of useful payloads and bypass for Web Application Security and Pentest/CTF. (n.d.). Retrieved August 25, 2026, from https://github.com/swisskyrepo/PayloadsAllTheThings

Tadhani, J. R., Vekariya, V., Sorathiya, V., Alshathri, S., & El-Shafai, W. (2024). Securing web applications against XSS and SQLi attacks using a novel deep learning approach. *Scientific Reports*, *14*(1). https://doi.org/10.1038/s41598-023-48845-4

The NIST Cybersecurity Framework (CSF) 2.0. (2024). https://doi.org/10.6028/NIST.CSWP.29

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17*, 6000–6010. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

What Is Threat Detection and Response (TDR)? | IBM. (n.d.). Retrieved May 30, 2026, from https://www.ibm.com/think/topics/threat-detection-and-response#2086344954

What Is Threat Hunting? | IBM. (n.d.). Retrieved May 30, 2026, from https://www.ibm.com/think/topics/threat-hunting?utm_medium=OSocial&utm_source=Youtube&utm_content=RSRWW&utm_id=YT-101-What-is-Threat-Hunting

Zhao, C., Agrawal, G., Zhang, F., Kumarage, T., Tan, Z., Deng, Y., Chen, Y.-C., & Liu, H. (2024). *Ontology-Aware RAG for Improved Question-Answering in Cybersecurity Education [Preprint]. ArXiv.* https://doi.org/10.48550/arXiv.2412.14191

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu,

Z., Wen, J.-R. (2026). A Survey of Large Language Models. *Frontiers of Computer Science*, *20*(12). https://doi.org/10.1007/s11704-026-60308-3

Παράρτημα Α: Κώδικας Υλοποίησης της Εργασίας

Ο κώδικας του προτεινόμενου συστήματος της παρούσας υλοποίησης είναι αναρτημένος στο παρακάτω GitHub repository:

https://github.com/ChatzitBill/WebAttackDetection-KG-VectorDB-LLMs/releases/tag/v1.0-thesis-HOU513

Ακολουθούν κάποια βασικά τμήματα της υλοποίησης.

initial_ingestion_qdrant.py:

Επεξήγηση: Script για την εισαγωγή των δεδομένων στη διανυσματική βάση

```
import json
import asyncio
import uuid
from pathlib import Path
from app.domain.ingestion.web_crawl import run_web_crawler
from app.domain.vector_search.chunker_qdrant import chunk_crawl_results, chunk_local_files
from app.domain.vector_search.embeddings_qdrant import get_embeddings
from app.domain.vector_search.vector_store  import create_vector_store, get_existing_vector_store
from app.db.qdrant import get_qdrant_client, get_or_create_collection, ensure_collection_exists, initialize_qdrant
from app.core.config import settings as Settings

def _load_websites_for_ingestion() -> list[str]:
    project_root = Path(__file__).resolve().parents[3]
    json_path = project_root / "data_for_ingestion" / "websites_ingestion_qdrantdb.json"

    if not json_path.exists():
        raise FileNotFoundError(f"Websites JSON file not found at {json_path}")

    with json_path.open("r", encoding="utf-8") as f:
        return json.load(f)

def _path_local_files_for_ingestion() -> Path:
    project_root = Path(__file__).resolve().parents[3]
    local_files_path = project_root / "data_for_ingestion" / "SQLi_md_files"
```

```
    if not local_files_path.exists() or not local_files_path.is_dir():
        raise FileNotFoundError(f"Local files directory not found at {local_files_path}")

    return local_files_path

def _stable_chunk_id(source_type: str, source_id: str, chunk_index: int) -> str:
    seed = f"{source_type}:{source_id}:{chunk_index}"
    return str(uuid.uuid5(uuid.NAMESPACE_URL, seed))


def main():

    #initial setup
    #initailze Qdrant DB
    initialize_qdrant()

    #get embeddings model
    embeddings = get_embeddings()

    #get client and collection for vector store
    client = get_qdrant_client()
    collection_name = Settings.COLLECTION_NAME
    if not ensure_collection_exists(client):
        vectorstore = create_vector_store(client, collection_name, embeddings)
    else:
        vectorstore = get_existing_vector_store(collection_name, embeddings)

    #Start of ingestion pipeline
    #load websites for crawling
    websites_for_crawl = _load_websites_for_ingestion()
    if not websites_for_crawl:
        print("No websites found for crawling. Please check the JSON file.")
        return

    crawl_results = asyncio.run(run_web_crawler(websites_for_crawl))

    # Further processing: chunking, embedding, and storing in Qdrant
    #chunking the crawl results and adding to vector store
```

```
    for crawl_result in crawl_results:
        chunks = chunk_crawl_results(crawl_result.markdown.fit_markdown, crawl_result.url)

        for i, chunk in enumerate(chunks):
            chunk.metadata["source"] = crawl_result.url
            chunk.metadata["source_type"] = "url"
            chunk.metadata["chunk_index"] = i

        ids = [_stable_chunk_id("url", crawl_result.url, i) for i in range(len(chunks))]
        vectorstore.add_documents(chunks, ids=ids)

    # chunking local files and adding to vector store
    local_files_path = _path_local_files_for_ingestion()

    for file_path in local_files_path.glob("*.md"):
        chunks = chunk_local_files(str(file_path))
        print(f"Generated {len(chunks)} chunks for file: {file_path.name}")

        for i, chunk in enumerate(chunks):
            chunk.metadata["source_file"] = file_path.name
            chunk.metadata["source_path"] = str(file_path)
            chunk.metadata["chunk_index"] = i

        ids = [_stable_chunk_id("file", file_path.name, i) for i in range(len(chunks))]
        vectorstore.add_documents(chunks, ids=ids)


if __name__ == "__main__":
    main()
```

initial_pcap_ingestion

Επεξήγηση: Script για την αρχική επεξεργασία των PCAP και την εισαγωγή των εξαχθέντων από αυτά στοιχείων στη Neo4j

```
from __future__ import annotations
import subprocess
import argparse
```

```
import re
from datetime import datetime
from pathlib import Path
from typing import Iterable
from threading import Lock
from concurrent.futures import ThreadPoolExecutor, as_completed
from app.domain.ingestion.pcap_ingestion import PCAPIngestionService
from app.services.logging_service import logger


processed_file_lock = Lock()
failed_file_lock = Lock()

def _is_pcap_with_tshark(file_path: Path) -> bool:
    try:
        result = subprocess.run(
            ["tshark", "-r", str(file_path), "-c", "1", "-q"],
            capture_output=True,
            text=True,
            check=False
        )

        if result.returncode != 0:
            logger.warning(f"Not a valid capture file: {file_path}")

        return result.returncode == 0
    except FileNotFoundError:
        return False

def _iter_pcap_files(folder: Path, recursive: bool):
    iterator = folder.rglob("*") if recursive else folder.glob("*")

    for file_path in iterator:
        if not file_path.is_file():
            continue
        if file_path.name.startswith("."):
            continue
        if _is_pcap_with_tshark(file_path):
```

```
            yield file_path


def _load_tracked_paths(file_path: Path) -> set[str]:
    if not file_path.exists():
        return set()

    return {
        line.strip()
        for line in file_path.read_text(encoding="utf-8").splitlines()
        if line.strip()
    }


def _append_tracked_path(file_path: Path, tracked_path: Path) -> None:
    with file_path.open("a", encoding="utf-8") as f:
        f.write(str(tracked_path.resolve()) + "\n")


def _extract_pcap_date_from_path(
    file_path: Path,
) -> str | None:
    return None


def _parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(
        description="PCAP ingestion into Neo4j graph database"
    )
    parser.add_argument(
        "--folder",
        required=True,
        help="Path to the folder containing PCAP files"
    )
    parser.add_argument(
        "--pcap-date",
        required=True,
        help="Capture date for the lab PCAP files, e.g. 2026-08-08",
    )
    parser.add_argument(
```

```
        "--recursive",
        action="store_true",
        help="Scan subfolders recursively"
    )
    parser.add_argument(
        "--limit",
        type=int,
        default=None,
        help="Optional max number of files to ingest"
    )
    parser.add_argument(
        "--processed-file",
        default="processed_pcaps.txt",
        help="File used to track already ingested PCAPs"
    )
    parser.add_argument(
        "--failed-file",
        default="failed_pcaps.txt",
        help="File used to track failed PCAPs"
    )
    parser.add_argument(
        "--workers",
        type=int,
        default=4,
        help="Number of worker threads for parallel PCAP ingestion"
    )
    return parser.parse_args()

def _process_single_file(
    file_path: Path,
    service: PCAPIngestionService,
    processed_file: Path,
    failed_file: Path,
    already_failed: set[str],
    args,
) -> tuple[str, int, str]:
    resolved = str(file_path.resolve())
```

```
extracted_date = _extract_pcap_date_from_path(file_path)
pcap_date = extracted_date or args.pcap_date

if not pcap_date:
    with failed_file_lock:
        if resolved not in already_failed:
            _append_tracked_path(failed_file, file_path)
            already_failed.add(resolved)

    logger.error(
        f"Could not determine date for file: {file_path}. "
        f"No folder date found and no --pcap-date fallback provided."
    )
    return resolved, 0, "failed"

logger.info(f"Processing file: {file_path} | pcap_date={pcap_date}")

try:
    count = service.ingest_file(
        pcap_path=str(file_path),
        pcap_date=pcap_date,
    )

    if count <= 0:
        logger.error(
            f"No records ingested from file: {file_path}"
        )

        with failed_file_lock:
            if resolved not in already_failed:
                _append_tracked_path(
                    failed_file,
                    file_path,
                )
                already_failed.add(resolved)

        return resolved, 0, "failed"
```

```
        with processed_file_lock:
            _append_tracked_path(
                processed_file,
                file_path,
            )

        logger.success(
            f"Finished file: {file_path.name} | "
            f"pcap_date: {pcap_date} | "
            f"ingested records: {count}"
        )

        return resolved, count, "success"

    except Exception as exc:
        with failed_file_lock:
            if resolved not in already_failed:
                _append_tracked_path(failed_file, file_path)
                already_failed.add(resolved)

        logger.exception(
            f"Failed to ingest file: {file_path} | pcap_date: {pcap_date} | error: {exc}"
        )
        return resolved, 0, "failed"


def main() -> None:
    args = _parse_args()
    folder = Path(args.folder)
    processed_file = Path(args.processed_file)
    failed_file = Path(args.failed_file)

    if not folder.exists() or not folder.is_dir():
        raise ValueError(f"Invalid folder path: {folder}")

    processed = _load_tracked_paths(processed_file)
    already_failed = _load_tracked_paths(failed_file)
```

```
service = PCAPIngestionService(batch_size=100)

total_files = 0
total_records = 0
failed_files = 0
skipped_files = 0

logger.info(f"Starting ingestion from folder: {folder}")
logger.info(f"Processed tracker file: {processed_file}")
logger.info(f"Failed tracker file: {failed_file}")
logger.info(f"Worker threads: {args.workers}")

candidate_files = []

for file_path in _iter_pcap_files(folder, args.recursive):
    resolved = str(file_path.resolve())

    if resolved in processed:
        skipped_files += 1
        logger.info(f"Skipping already processed file: {file_path}")
        continue

    candidate_files.append(file_path)

    if args.limit is not None and len(candidate_files) >= args.limit:
        logger.info(f"Reached limit={args.limit}, stopping file discovery")
        break

with ThreadPoolExecutor(max_workers=args.workers) as executor:
    logger.info(
    f"Discovered candidate PCAP files: "
    f"{len(candidate_files)}"
)

    for candidate in candidate_files:
        logger.info(f"Candidate PCAP: {candidate}")

    futures = [
```

```
            executor.submit(
                _process_single_file,
                file_path,
                service,
                processed_file,
                failed_file,
                already_failed,
                args,
            )
            for file_path in candidate_files
        ]

        for future in as_completed(futures):
            _, count, status = future.result()

            if status == "success":
                total_files += 1
                total_records += count
            else:
                failed_files += 1

    logger.info(
        "Ingestion complete | "
        f"succeeded: {total_files} | "
        f"failed: {failed_files} | "
        f"skipped: {skipped_files} | "
        f"total records: {total_records}"
    )


if __name__ == "__main__":
    main()
```

workflow_graph.py

Επεξήγηση: Workflow ενορχήστρωσης των agents με τη χρήση του framework LangGraph

```
import json
from typing import TypedDict, Annotated, Any, Literal
import operator
from langchain_core.messages import AIMessage, BaseMessage, HumanMessage
from langgraph.graph import StateGraph, START, END, add_messages
from langgraph.checkpoint.memory import InMemorySaver

from app.domain.vector_search.models import VectorSearchResult
from app.domain.kg_search.models import KGSearchResult
from app.domain.hybrid_search.models import HybridSearchResult
from app.domain.agents.verification_agent.models import VerificationReport
from app.domain.agents.relevance_checker_agent.relevance_checker_agent import RelevanceCheckerAgent
from app.domain.agents.relevance_checker_agent.models import RelevanceLabel
from app.domain.agents.verification_agent.verification_agent import VerificationAgent
from app.domain.agents.research_agent.research_agent import ResearchAgent
from app.domain.agents.rag_agent.subgraph_workflow import build_rag_retrieval_subgraph
from app.core.config import settings as Settings

class WorkflowState(TypedDict, total=False):
    messages: Annotated[list, add_messages]
    query: str
    relevance_decision: str
    rag_retrieval: list[VectorSearchResult | KGSearchResult | HybridSearchResult]
    draft_answer: str
    final_answer: str
    verification_report: VerificationReport
    retry_count: int
    classification_task: bool = False


class AgentWorkflow:
    def __init__(self,  llm):
        self.llm = llm
        self.relevance_checker_agent = RelevanceCheckerAgent(self.llm)
        self.verification_agent = VerificationAgent(self.llm)
        self.research_agent = ResearchAgent(self.llm)
```

```
    self.rag_retrieval_subgraph = build_rag_retrieval_subgraph(self.llm)
    self.checkpoint_saver = InMemorySaver()
    self.compiled_workflow = self.build_workflow()

def build_workflow(self):
    workflow = StateGraph(WorkflowState)

    #add nodes
    workflow.add_node("relevance_checker", self._relevance_checker_node)
    workflow.add_node("rag_retrieval", self._rag_retrieval_node)
    workflow.add_node("research", self._research_node)
    workflow.add_node("verification", self._verification_node)
    workflow.add_node("finalize", self._finalize_node)

    #add edges
    workflow.add_edge(START, "relevance_checker")

    workflow.add_conditional_edges(
        "relevance_checker",
        self._route_after_relevance,
        {
            "rag_retrieval": "rag_retrieval",
            "irrelevant": "finalize",
        }
    )

    workflow.add_edge("rag_retrieval", "research")
    workflow.add_edge("research", "verification")

    workflow.add_conditional_edges(
        "verification",
        self._route_after_verification,
        {
            "retrieve_more": "rag_retrieval",
            "revise_answer": "research",
            "finalize": "finalize",
        }
    )
```

```
        workflow.add_edge("finalize", END)

        return workflow.compile(checkpointer=self.checkpoint_saver)

    def _relevance_checker_node(self,state: WorkflowState) -> dict:
        response = self.relevance_checker_agent.check_relevance(state["query"])

        if response.relevance_decision.relevance_response == RelevanceLabel.IRRELEVANT:
            draft_answer = "The query is irrelevant to SQLi attacks. Please provide a query related to SQLi attacks for further assistance."

            updated_messages = list(state.get("messages", []))
            updated_messages.append(AIMessage(content=draft_answer))
            updated_messages = updated_messages[-Settings.MAX_MESSAGES_PER_SESSION:]
            return {
                "relevance_decision": response.relevance_decision.relevance_response.value,
                "messages": updated_messages,
                "draft_answer": draft_answer
            }

        return {
            "relevance_decision": response.relevance_decision.relevance_response.value
        }


    def _rag_retrieval_node(self, state: WorkflowState) -> dict:
        verification_report = state.get("verification_report")
        retrieval_guidance = (
            verification_report.additional_details
            if verification_report is not None
            else "None"
        )

        result = self.rag_retrieval_subgraph.invoke({
            "query": state["query"],
            "retrieval_guidance": retrieval_guidance,
        })
```

```
    return {
        "rag_retrieval": result.get("rag_retrieval", []),
        "retry_count": state.get("retry_count", 0) + 1,
    }

def _research_node(self, state: WorkflowState) -> dict:
    if "draft_answer" not in state:
        if state["classification_task"] == True:
            classification_result = self.research_agent.classify_capture(
                query=state["query"],
                rag_retrieval=state.get("rag_retrieval", []),
                messages=state.get("messages", [])
            )
            draft_answer = json.dumps({
                "detection": classification_result.detection,
                "attack_type": classification_result.attack_type,
                "justification": classification_result.justification
            })
            return {
                "draft_answer": draft_answer,
                "classification_task": False
            }

        result = self.research_agent.generate_draft(
                query=state["query"],
                rag_retrieval=state.get("rag_retrieval", []),
                messages=state.get("messages", [])
            )
        return {
            "draft_answer": result.draft_answer_result.answer
    }
    else:
        result = self.research_agent.revise_answer(
            query=state["query"],
            rag_retrieval=state["rag_retrieval"],
            draft_answer=state.get("draft_answer", ""),
            verification_report=state.get("verification_report", None),
```

```
            messages=state.get("messages", []),
        )
        return {
            "draft_answer": result.revised_answer_result.answer
        }


def _verification_node(self, state: WorkflowState) -> dict:
    response = self.verification_agent.verify_answer(
        query=state["query"],
        draft_answer=state.get("draft_answer", ""),
        rag_retrieval=state.get("rag_retrieval", []),
    )
    return {
        "verification_report": response.verification_report
    }

def _finalize_node(self, state: WorkflowState) -> dict:
    updated_messages = list(state.get("messages", []))

    final_answer = state.get("draft_answer") or "I could not generate an answer for this request."

    updated_messages.append(AIMessage(content=final_answer))
    updated_messages = updated_messages[-Settings.MAX_MESSAGES_PER_SESSION:]

    return {
        "messages": updated_messages,
        "final_answer": final_answer,
    }

def _route_after_relevance(self, state: WorkflowState) -> Literal["rag_retrieval", "irrelevant"]:
    if state["relevance_decision"] in ("RELEVANT", "PARTIALLY_RELEVANT"):
        return "rag_retrieval"
    return "irrelevant"

def _route_after_verification(self, state: WorkflowState) -> str:
    verification_report = state["verification_report"]
    decision = verification_report.workflow_action
```

```
    # MAX RETRY COUNT : if it is reached, do not retrieve more, instead go to research revision
    if state.get("retry_count", 0) >= Settings.MAX_RETRIEVALS_RETRIES and decision == "retrieve_more":
        return "revise_answer"

    return decision


def execute(self, query: str, thread_id: str) -> dict:
    initial_state: WorkflowState = {
        "query": query,
        "messages": [HumanMessage(content=query)],
        "retry_count": 0
    }
    final_state = self.compiled_workflow.invoke(initial_state, config={"configurable": {"thread_id": thread_id}})
    return final_state
```

Relevance Checker Agent

```
from app.domain.agents.relevance_checker_agent.models import (
    RelevanceAgentOutput,
    RelevanceAgentResponse,
    RelevanceLabel,
)
from app.domain.agents.relevance_checker_agent.prompts import RELEVANCE_PROMPT
from app.services.logging_service import logger
from langchain_core.language_models import BaseLanguageModel


class RelevanceCheckerAgent:
    def __init__(self, llm: BaseLanguageModel):
        self.llm = llm

    def check_relevance(self, query: str) -> RelevanceAgentResponse:
        logger.info(f"RelevanceCheckerAgent.check_relevance called with query='{query}'")

        structured_llm = self.llm.with_structured_output(RelevanceAgentOutput)
        chain = RELEVANCE_PROMPT | structured_llm
```

```
    try:
        output = chain.invoke({"query": query})
        logger.info(f"LLM relevance response: {output}")
        return RelevanceAgentResponse(
            query=query,
            relevance_decision=output
        )
    except Exception as e:
        logger.error(f"Relevance classification failed: {e}")
        output=RelevanceAgentOutput(relevance_response=RelevanceLabel.IRRELEVANT)
        return RelevanceAgentResponse(
            query=query,
            relevance_decision=output
        )
```

```
from langchain_core.prompts import ChatPromptTemplate

RELEVANCE_PROMPT = ChatPromptTemplate.from_messages([
  (
    "system",
    """
        You are a relevance classifier for a multi-agent hybrid RAG system.

        Your task is to classify the USER QUERY only.
        Do NOT judge whether enough evidence has already been retrieved.
        Do NOT require the query to contain exact SQL injection terminology.
        Judge whether the query is within the scope of this system.

        System scope:
        - Explains SQL injection (SQLi) attacks, concepts, payloads, techniques, indicators, and defenses.
        - Analyzes ingested PCAP / TCP/IP / HTTP / application traffic data to identify possible SQL injection attacks.
        - Investigates suspicious requests, parameters, payloads, entities, relationships, evidence, and graph findings related to SQLi.
        - Answers questions about whether the user's ingested dataset may contain SQLi activity.
```

```
        Labels:
        - RELEVANT: The query is clearly within the system scope.
        - PARTIALLY_RELEVANT: The query is loosely related to cybersecurity, network traffic, application traffic, attacks, suspicious behavior, or database-related abuse, but the SQLi connection is weak or implicit.
        - IRRELEVANT: The query is outside the system scope.

        Important rules:
        - If the user asks about attacks, suspicious payloads, requests, parameters, database abuse, HTTP/TCP-IP/PCAP traffic, or evidence in their dataset, that is usually RELEVANT or PARTIALLY_RELEVANT.
        - Queries about the user's traffic or ingested data should usually be accepted unless clearly unrelated.
        - General everyday questions, unrelated programming help, food, entertainment, weather, or unrelated topics are IRRELEVANT.
        - When uncertain between RELEVANT and IRRELEVANT, prefer PARTIALLY_RELEVANT.

        Examples:
        - "What is SQL injection?" -> RELEVANT
        - "Find suspicious payloads in my dataset" -> RELEVANT
        - "Is there any SQLi attack in my TCP/IP traffic?" -> RELEVANT
        - "Show attack evidence from the ingested PCAP" -> RELEVANT
        - "Are there suspicious requests in my HTTP logs?" -> PARTIALLY_RELEVANT
        - "Can you explain XSS?" -> PARTIALLY_RELEVANT
        - "Can I have some pizza?" -> IRRELEVANT
        - "What is the weather today?" -> IRRELEVANT

        Return only the label in the structured output.
        """
        ),
        (
            "human",
            "User query: {query}"
        ),
])
```

RAG Agent

```
from langchain_openai import ChatOpenAI
from langchain_core.documents import Document
from langgraph.prebuilt import ToolNode
```

```
from langchain_core.language_models import BaseLanguageModel
from typing import Literal, Any
import json

from app.domain.agents.rag_agent.prompts import RAG_AGENT_PROMPT
from app.services.logging_service import logger
from app.domain.vector_search.models import VectorSearchResult
from app.domain.kg_search.models import KGSearchResult
from app.domain.hybrid_search.models import HybridSearchResult
from app.domain.agents.rag_agent.models import RAGAgentResponse
from app.domain.vector_search.vector_search_tool import vector_search_tool
from app.domain.kg_search.kg_search_tool import kg_search_tool
from app.domain.hybrid_search.hybrid_search_tool import hybrid_search_tool
from app.domain.agents.rag_agent.subgraph_state import RagRetrievalState
from langsmith import traceable


class RagAgent:
    def __init__(self, llm: BaseLanguageModel):
        self.llm = llm
        self.tools = [
            vector_search_tool,
            kg_search_tool,
            hybrid_search_tool,
        ]
        self.llm_with_tools = self.llm.bind_tools(self.tools)
        self.tool_node = ToolNode(self.tools)


    def reason(self, state: RagRetrievalState) -> dict:
        existing_messages = state.get("messages", [])

        if not existing_messages:
            prompt_value = RAG_AGENT_PROMPT.invoke({
                "query": state["query"],
                "retrieval_guidance": state.get("retrieval_guidance", "None"),
            })
            messages = prompt_value.to_messages()
```

```
    else:
        messages = existing_messages

    response = self.llm_with_tools.invoke(messages)
    return {"messages": [response]}

def should_continue_retrieval(self, state: RagRetrievalState) -> Literal["tools", "collect"]:
    last_message = state["messages"][-1]

    if getattr(last_message, "tool_calls", None):
        return "tools"

    return "collect"

def collect_retrieved_evidence(self, state: RagRetrievalState) -> dict:
    retrieval_results = []

    for msg in state.get("messages", []):
        if getattr(msg, "type", None) != "tool":
            continue

        tool_name = getattr(msg, "name", None)
        content = msg.content

        if isinstance(content, str):
            try:
                content = json.loads(content)
            except Exception:
                pass

        if tool_name == "vector_search_tool" and isinstance(content, dict):
            retrieval_results.append(VectorSearchResult.model_validate(content))
        elif tool_name == "kg_search_tool" and isinstance(content, dict):
            retrieval_results.append(KGSearchResult.model_validate(content))
        elif tool_name == "hybrid_search_tool" and isinstance(content, dict):
            retrieval_results.append(HybridSearchResult.model_validate(content))
        else:
            retrieval_results.append(content)
```

```
    return {"rag_retrieval": retrieval_results}
```

```
from langchain_core.prompts import ChatPromptTemplate

RAG_AGENT_PROMPT = ChatPromptTemplate.from_messages([
  (
    "system",
    """You are the RAG Agent of a cybersecurity analysis system.

    Your job is retrieval orchestration only.
    Do not produce the final answer for the user.
    Your role is to decide which retrieval tool(s) should be used to gather the best evidence.

    Available tools:
    1. vector_search_tool
    - Use for semantic/background knowledge from documents and chunked text.
    - Use when the question asks about concepts, explanations, attack patterns, or SQL injection knowledge.

    2. kg_search_tool
    - Use for graph-based evidence from Neo4j.
    - Use when the question is about days, IPs, hosts, HTTP requests, parameters, relationships, or observed traffic behavior.
    - The KG tool returns raw graph evidence (rows), not summaries or classifications. Downstream agents will perform the detection and labeling.


    3. hybrid_search_tool
    - Use when the question requires both semantic knowledge and graph evidence.
    - Prefer this for questions such as whether SQL injection activity occurred on a specific day and what evidence supports that conclusion.

    Rules:
    - Focus on retrieving the most relevant evidence.
    - You may call one tool or multiple tools if needed.
    - If the query clearly requires both graph evidence and semantic interpretation, prefer hybrid_search_tool.
    - Do not invent evidence.
```

```
        - Do not produce a final user-facing answer; only retrieve and organize evidence.
        - For capture classification, treat KG raw_result as the authoritative evidence.
          Vector results may explain techniques but must not change the detection label.
          Generic cybersecurity documentation can explain a technique, but it cannot prove that the technique
occurred in this capture.
          Treat generic vector-search results as background knowledge, not as evidence of the capture's content.
        """
    ),
    (
        "human",
        """User query:
        {query}

        Additional retrieval guidance:
        {retrieval_guidance}
        """
    ),
])
```

Research Agent

```
from langchain_core.language_models import BaseLanguageModel

from app.domain.agents.research_agent.models import ClassificationResult, DraftAnswerResult,
RevisedAnswerResult, ResearchAgentResponse
from app.domain.agents.research_agent.prompts import (
    CLASSIFICATION_PROMPT,
    RESEARCH_DRAFT_PROMPT,
    RESEARCH_REVISION_PROMPT,
)
from app.domain.agents.utils import serialize_messages, serialize_rag_retrieval
from app.domain.vector_search.models import VectorSearchResult
from app.domain.kg_search.models import KGSearchResult
from app.domain.hybrid_search.models import HybridSearchResult
from app.core.config import settings as Settings

class ResearchAgent:
    def __init__(self, llm: BaseLanguageModel):
```

```
        self.llm = llm



    def generate_draft(self, query: str, rag_retrieval: list[VectorSearchResult | KGSearchResult |
HybridSearchResult], messages: list) -> ResearchAgentResponse:
        serialized_evidence = serialize_rag_retrieval(rag_retrieval)
        chat_history = serialize_messages(messages[-Settings.MAX_MESSAGES_PER_SESSION:])

        structured_llm = self.llm.with_structured_output(DraftAnswerResult)
        chain = RESEARCH_DRAFT_PROMPT | structured_llm

        draft_answer = chain.invoke({"query": query,"chat_history": chat_history,"retrieved_evidence":
serialized_evidence})

        return ResearchAgentResponse(
            draft_answer_result=draft_answer,
            revised_answer_result=None,
            context=serialized_evidence
        )

    def revise_answer(
        self,
        query: str,
        rag_retrieval: list[VectorSearchResult | KGSearchResult | HybridSearchResult],
        draft_answer: DraftAnswerResult,
        verification_report: str,
        messages: list
    ) -> ResearchAgentResponse:
        serialized_evidence = serialize_rag_retrieval(rag_retrieval)
        chat_history = serialize_messages(messages[-Settings.MAX_MESSAGES_PER_SESSION:])

        structured_llm = self.llm.with_structured_output(RevisedAnswerResult)
        chain = RESEARCH_REVISION_PROMPT | structured_llm

        revised_answer_result = chain.invoke({
            "query": query,
            "rag_retrieval": serialized_evidence,
```

```
        "draft_answer": draft_answer,
        "chat_history": chat_history,
        "verification_report": verification_report,
    })

    return ResearchAgentResponse(
        draft_answer_result=None,
        revised_answer_result=revised_answer_result,
        context=serialized_evidence
    )

def classify_capture(
    self,
    query: str,
    rag_retrieval: list[VectorSearchResult | KGSearchResult | HybridSearchResult],
    messages: list | None = None,
) -> ClassificationResult:
    """
    Structured classification for benchmark-style capture tasks.
    This does NOT change your generic answer flow; it's an extra method
    you call only when you need a structured label.
    """
    serialized_evidence = serialize_rag_retrieval(rag_retrieval)
    chat_history = serialize_messages(
        (messages or [])[-Settings.MAX_MESSAGES_PER_SESSION:]
    )

    structured_llm = self.llm.with_structured_output(ClassificationResult)
    chain = CLASSIFICATION_PROMPT | structured_llm

    return chain.invoke({
        "query": query,
        "chat_history": chat_history,
        "retrieved_evidence": serialized_evidence,
    })
```

```
from langchain_core.prompts import ChatPromptTemplate
```

```
RESEARCH_DRAFT_PROMPT = ChatPromptTemplate.from_messages([
  (
    "system",
      """You are the Research Agent of a cybersecurity analysis system.

          Your job is to create a grounded draft answer using only the retrieved evidence.
          Do not invent facts that are not supported by the evidence.

          Rules:
          - Answer the user's question clearly.
          - Base the answer only on the retrieved evidence.
          - If the evidence is incomplete or uncertain, say that clearly.
          - Focus especially on cybersecurity interpretation, including whether the evidence suggests SQL
injection activity.
          - Do not mention internal tools, agents, workflows, or implementation details.

          For classification tasks for a capture, always produce this structure:

          Detection: benign | sqli
          Attack type: <normalized subtype> | null
          Justification: <one or two sentences based only on retrieved evidence>

          If malicious evidence is present, identify the most specific supported SQLi subtype.
          Use these normalized labels when applicable:
          - boolean-based
          - error-based
          - union-based
          - time-based blind
          - multi-technique

          Do not output only the generic label "SQL injection" when the evidence supports
          a more specific subtype.

          If the query returns no matching malicious records, classify the capture as
          benign and state that no malicious pattern was detected. Do not describe the
          result as impossible to determine unless the retrieval itself failed.
```

```
            For example, the ideal answer for capture-XXX.pcap is:
            Detection: sqli
            Attack type: time-based blind
            Justification: The capture contains repeated SLEEP(...) payloads in the id
            parameter, which are characteristic of time-based blind SQL injection.
        """
    ),
    (
        "human",
        """User query:
            {query}

        Recent conversation:
            {chat_history}

        Retrieved evidence:
            {retrieved_evidence}
        """
    ),
])



RESEARCH_REVISION_PROMPT = ChatPromptTemplate.from_messages([
    (
        "system",
            """You are the Research Agent of a cybersecurity analysis system.

            Your job is to revise a draft answer using:
            - the user query,
            - the retrieved evidence,
            - and the verification report.

            Rules:
            - Improve the answer based on the verification report.
            - Remove unsupported or contradictory content.
            - Stay grounded in the retrieved evidence.
```

```
            - Do not invent missing evidence.
            - Do not decide workflow control actions.
        """
    ),
    (
        "human",
        """User query:
            {query}

        Rag Retrieval:
            {rag_retrieval}

        Current draft answer:
            {draft_answer}

        Recent conversation:
            {chat_history}

        Verification report:
            {verification_report}
        """
        ),
])

CLASSIFICATION_PROMPT = ChatPromptTemplate.from_messages([
    (
        "system",
        """You are a classification assistant for SQL injection detection.

        Your task is to read the user query and retrieved evidence, then produce a structured classification.

        Rules:
        - detection must be exactly "benign" or "sqli".
        - attack_type must be exactly one of:
        - "boolean-based"
        - "error-based"
        - "union-based"
        - "time-based blind"
```

```
    - "multi-technique"
    - null
    - If evidence supports two or more distinct SQLi techniques, attack_type must be "multi-technique".
    - If detection is "benign", attack_type must be null.
    - If detection is "sqli", attack_type cannot be null.
    - The justification must be 1 to 2 sentences, grounded only in the retrieved evidence.
    - Do not mention tools, agents, or internal implementation details.

    Special rule for knowledge graph (KG) evidence:

    - If the retrieved evidence contains a KG result with "raw_result": [] (empty list),
    this means the knowledge graph found NO matching SQL injection patterns
    for this capture.
    - In this situation:
    - You must set detection = "benign".
    - You must set attack_type = null.
    - Any claim that SQL injection was detected is unsupported and must not be made.
    - Treat "raw_result": [] as strong evidence of no attack, not as missing data.
    - If you see:
        Knowledge Graph Evidence:
            status: success
            raw_result: []
            # No matching SQL injection patterns found...
        then treat this as evidence of benign traffic.

        - If you see:
        Knowledge Graph Evidence:
            status: error
            # Retrieval failed...
        then do not treat this as benign; treat as insufficient evidence.
    """
),
(
    "human",
    """User query:
        {query}

        Recent conversation:
```

```
        {chat_history}

        Retrieved evidence:
        {retrieved_evidence}
        """
    ),
])
```

Verification Agent

```
from langchain_openai import ChatOpenAI
from langchain_core.documents import Document
from app.domain.agents.verification_agent.prompts import VERIFICATION_PROMPT
from app.domain.agents.verification_agent.models import VerificationAgentResponse, VerificationReport
from app.services.logging_service import logger
from app.domain.vector_search.models import VectorSearchResult
from app.domain.kg_search.models import KGSearchResult
from app.domain.hybrid_search.models import HybridSearchResult
from langchain_core.language_models import BaseLanguageModel
from app.domain.agents.utils import serialize_rag_retrieval


class VerificationAgent:
    def __init__(self, llm: BaseLanguageModel):
        self.llm = llm

    def verify_answer(self, query: str, draft_answer: str, rag_retrieval: list[VectorSearchResult | KGSearchResult | HybridSearchResult]) -> VerificationAgentResponse:
        """
        Use the LLM to verify the answer generated by the research agent based on the retrieved documents from RAG (VectorDB/KG).

        Process:
        1. Construct a prompt that includes the answer and the retrieved documents.
        2. Ask the LLM to evaluate whether the answer is supported by the retrieved documents, if there are any unsupported claims,
           contradictions, and if the answer is relevant to the question based on the context.
        3. Return a structured response with the verification report.
```

```
    """

    logger.info("VerificationAgent.verify_answer called with generated answer and retrieved documents for verification.")

    #serilalize the retrieved documents for the prompt
    serialized_evidence = serialize_rag_retrieval(rag_retrieval)

    # Construct the prompt for the LLM
    structured_llm = self.llm.with_structured_output(VerificationReport)

    chain = VERIFICATION_PROMPT | structured_llm


    try:
        output = chain.invoke({ "query": query, "draft_answer": draft_answer, "context": serialized_evidence })
        logger.info(f"LLM verification response: {output}")
    except (IndexError, KeyError) as e:
        logger.error(f"Unexpected response structure: {e}")
        return VerificationAgentResponse(
            verification_report=VerificationReport(
                supported=False,
                unsupported_claims=[],
                contradictions=[],
                workflow_action="retrieve_more",
                additional_details="Failed to verify the answer."
            ),
            context=serialized_evidence
        )

    return VerificationAgentResponse(
        verification_report=output,
        context=serialized_evidence
    )
```

```
from langchain_core.prompts import ChatPromptTemplate

VERIFICATION_PROMPT = ChatPromptTemplate.from_messages([
    (
        "system",
            """You are a verification agent.

            Verify the accuracy and relevance of the answers based on provided context.

            **Instructions:**
            - Verify the following answer against the provided context.
            - Check for:
            1. Direct/indirect factual support (YES/NO)
            2. Unsupported claims (list any if present)
            3. Contradictions (list any if present)
            4. Decide the next workflow action
            5. Provide specific retrieval guidance describing what evidence is missing or specific revision guidance describing what should change in the answer, if applicable.
            - Respond in the exact format specified below without adding any unrelated information.

            Special rule for knowledge graph (KG) evidence:

                - If the context contains a KG result with "raw_result": [] (empty list),
                    this means the knowledge graph found NO matching SQL injection patterns
                    for this capture.
                - In this situation:
                    - Any claim that SQL injection was detected is UNSUPPORTED.
                    - The correct classification is "benign" unless other evidence clearly
                    contradicts this.
                - Treat "raw_result": [] as strong evidence of no attack, not as missing data.
                - If you see:
                    Knowledge Graph Evidence:
                        status: success
                        raw_result: []
                        # No matching SQL injection patterns found...
                    then treat this as evidence of benign traffic.

                    - If you see:
```

```
                Knowledge Graph Evidence:
                  status: error
                  # Retrieval failed...
                then do not treat this as benign; treat as insufficient evidence.

        **Format:**
        Supported: YES/NO
        Unsupported Claims: [item1, item2, ...]
        Contradictions: [item1, item2, ...]
        Workflow Action: [finalize | revise_answer | retrieve_more]
        Additional Details: [Any extra information or explanations]

        Decision rules:
            - Choose "finalize" if the answer is supported, relevant, and does not require meaningful correction.
            - Choose "revise_answer" if the answer is mostly grounded but needs clarification, correction of
wording, removal of minor unsupported content, or better alignment with the question.
            - Choose "retrieve_more" if the current context is insufficient to support a reliable answer, or if
important evidence is missing.
            - If the reference or KG evidence clearly indicates no SQLi patterns (e.g. raw_result=[]), and the answer
claims SQL injection,
              set Supported: NO and choose "revise_answer" or "retrieve_more" as appropriate.

            Additional rules:
            - If you choose "retrieve_more", provide specific retrieval guidance describing what evidence is
missing.
            - If you choose "revise_answer", provide specific revision guidance describing what should change in
the answer.
            - Do not invent evidence.
            - Base your judgment only on the provided answer and context.
        """
    ),
    (
      "human",
        """
        Query: {query}

        Answer: {draft_answer}
```

```
        Context: {context}"""
    ),
])
```

Παράρτημα Β: Σύνολα Αξιολόγησης

```
AGENTIC_CASES = [
    {
        "case_id": "agentic_001",
        "type": "general_knowledge",
        "user_input": "What is a SQLi attack?",
        "reference": """SQL Injection (SQLi) is a type of security vulnerability that allows an attacker to interfere with the queries that an application makes
                        to its database. SQL Injection is one of the most common and severe types of web application vulnerabilities, enabling attackers
                        to execute arbitrary SQL code on the database. This can lead to unauthorized data access, data manipulation, and, in some cases,
                        full compromise of the database server.""",
        "expected_tools": ["vector_search_tool"],
        "alternative_tools": []
    },
    {
        "case_id": "agentic_002",
        "type": "explanation",
        "user_input": "How is a UNION-based SQLi attack performed?",
        "reference": """In a standard SQL query, data is retrieved from one table. The UNION operator allows multiple SELECT statements to be combined.
                        If an application is vulnerable to SQL injection, an attacker can inject a crafted SQL query that appends a UNION statement to the original query.

                        Let's assume a vulnerable web application retrieves product details based on a product ID from a database:
                            SELECT product_name, product_price FROM products WHERE product_id = 'input_id';
                        An attacker could modify the input_id to include the data from another table like users.
                            1' UNION SELECT username, password FROM users --
                        After submitting our payload, the query become the following SQL:
                            SELECT product_name, product_price FROM products WHERE product_id = '1' UNION SELECT username, password FROM users --';
                        The 2 SELECT clauses must have the same number of columns.""",
        "expected_tools": ["vector_search_tool"],
        "alternative_tools": []
    },
```

```
  {
    "case_id": "agentic_003",
    "type": "irrelevant",
    "user_input": "I want some pizza",
    "reference": "The query is irrelevant to SQLi attacks. Please provide a query related to SQLi attacks for
further assistance.",
    "expected_tools": [],
    "alternative_tools": []
  },
  {
    "case_id": "agentic_004",
    "type": "tool_use",
    "user_input": "Can you find any time-based SQLi attacks in your data and in which captures?",
    "reference": """Yes, I found a time-based SQLi attack in the data and specifically in capture-005 and in
capture-016. The attack was
                performed using a payload that included a time delay function, which this indicates possible time-
based SQL injection attacks.""",
    "expected_tools": ["hybrid_search_tool"],
    "alternative_tools": ["kg_search_tool"]
  },
  {
    "case_id": "agentic_005",
    "type": "partially_relevant",
    "user_input": "Can you explain me how a XSS attack works?",
    "reference": """A Cross-Site Scripting (XSS) attack is a web application vulnerability that allows attackers to
inject malicious code into a web page
                viewed by other users. This malicious code can steal sensitive information like cookies or session
data, redirect users to attacker-controlled
                websites, or perform other harmful actions under the guise of the vulnerable site.""",
    "expected_tools": ["vector_search_tool"],
    "alternative_tools": []
  },
  {
    "case_id": "agentic_006",
    "type": "detection-analysis",
    "user_input": "In capture-014.pcap, can you detect any malicious attempt? If yes, what kind of attack is it
and what is the justification?",
```

```
    "reference": """The traffic in capture-014.pcap contains Error-based SQL injection patterns targeting the id GET parameter.
                The attack uses MySQL's EXTRACTVALUE() function with XPath syntax error exploitation to extract data through error messages.
                Specifically, sqlmap injected payloads like id=1' AND EXTRACTVALUE(7267,CONCAT(0x5c,0x7170626a71,(SELECT (ELT(7267=7267,1))),0x7178627871))-- ZxGr,
                which forces the database to return data inside XPATH syntax error messages. The back-end DBMS was identified as MySQL >= 5.1 (MariaDB fork).""",
    "expected_tools": ["hybrid_search_tool"],
    "alternative_tools": ["kg_search_tool"]
  },
  {
    "case_id": "agentic_007",
    "type": "partially_relevant",
    "user_input": "Can you explain me the key differences between SQLi and XSS attacks?",
    "reference": """SQLi (SQL Injection) and XSS (Cross-Site Scripting) are both web application vulnerabilities, but they target different aspects
                of a web application and have distinct characteristics:
                1. Target: SQLi targets the database layer of a web application, allowing attackers to manipulate SQL queries and access or modify data.
                  XSS targets the client-side (user's browser), allowing attackers to inject malicious scripts that execute in the context of the user's session.
                2. Impact: SQLi can lead to unauthorized data access, data manipulation, and potentially full control over the database. XSS can lead to session hijacking,
                  defacement, and the execution of malicious scripts in the user's browser, potentially stealing sensitive information.""",
    "expected_tools": ["vector_search_tool"],
    "alternative_tools": []
  },
  {
    "case_id": "agentic_008",
    "type": "irrelevant",
    "user_input": "Is SQLi attacks and gelato related?",
    "reference": "The query is irrelevant to SQLi attacks. Please provide a query related to SQLi attacks for further assistance.",
    "expected_tools": [],
    "alternative_tools": []
  },
```

```
  {
    "case_id": "agentic_009",
    "type": "edge_case",
    "user_input": "In capture-012.pcap traffic is there any evidence of SQLi attacks? If yes, what kind of attack is it and what is the justification?",
    "reference": """In capture-012.pcap traffic, there is no evidence of SQLi attacks. The traffic appears to be benign. The username O'Reilly is used and even if
                the single quote in specific case might be an inidicator of a potential SQLi attack, in this case it is not.""",
    "expected_tools": ["hybrid_search_tool"],
    "alternative_tools": ["kg_search_tool"]
  },
  {
    "case_id": "agentic_010",
    "type": "detection-analysis",
    "user_input": "In capture-018.pcap, can you detect any malicious attempt? If yes, what kind of attack is it and what is the justification?",
    "reference": """The traffic in capture-018.pcap contains a multi-technique SQL injection attack targeting the id GET parameter. Sqlmap identified four distinct injection techniques:

                Boolean-based blind: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment) with payload id=1' OR NOT 6152=6152#

                Error-based: MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE) with payload id=1' AND EXTRACTVALUE(5338,CONCAT(0x5c,0x71766a7671,(SELECT (ELT(5338=5338,1))),0x7171786271))-- QMZy

                Time-based blind: MySQL >= 5.0.12 AND time-based blind (query SLEEP) with payload id=1' AND (SELECT 5974 FROM (SELECT(SLEEP(5)))uGGO)-- BzgM

                UNION query: MySQL UNION query (NULL) - 2 columns with payload id=1' UNION ALL SELECT NULL,CONCAT(0x71766a7671,0x5a634d4c694962534a4d536b476359567263794454416 24e5a637870656577484147526976677473,0x7171786271)#

                The attack required 146 HTTP requests to fully enumerate the vulnerability. The back-end DBMS was identified as MySQL >= 5.1 (MariaDB fork) running on Linux Debian with Apache 2.4.67 and PHP 8.5.6.""",
    "expected_tools": ["hybrid_search_tool",],
    "alternative_tools": ["kg_search_tool"]
```

```
    }

]

# Dataset 2: SQLi Detection
DETECTION_CASES = [
    {
        "case_id": "detect_001",
        "pcap_identifier": "capture-001.pcap",
        "user_input": """Analyze capture-001.pcap. Return exactly three fields:
                        Detection (benign or sqli),
                        Attack type (or null if benign),
                        and justify your classification decision of each possible attack type based on the retrieved
evidence.""",
        "reference_label": "benign",
        "reference_attack_type": None,
        "reference_justification": "No SQL injection patterns detected in the traffic.",
    },
    {
        "case_id": "detect_002",
        "pcap_identifier": "capture-002.pcap",
        "user_input": """Analyze capture-002.pcap. Return exactly three fields:
                        Detection (benign or sqli),
                        Attack type (or null if benign),
                        and justify your classification decision of each possible attack type based on the retrieved
evidence.""",
        "reference_label": "benign",
        "reference_attack_type": None,
        "reference_justification": "No SQL injection patterns detected in the traffic.",
    },
    {
        "case_id": "detect_003",
        "pcap_identifier": "capture-003.pcap",
        "user_input": """Analyze capture-003.pcap. Return exactly three fields:
                        Detection (benign or sqli),
                        Attack type (or null if benign),
                        and justify your classification decision of each possible attack type based on the retrieved
evidence.""",
```

```
    "reference_label": "sqli",
    "reference_attack_type": "boolean-based",
    "reference_justification": "Boolean-based blind SQLi detected in 'id' GET parameter using OR NOT technique (payload: id=1' OR NOT 8720=8720#).",
  },
  {
    "case_id": "detect_004",
    "pcap_identifier": "capture-004.pcap",
    "user_input": """Analyze capture-004.pcap. Return exactly three fields:
                Detection (benign or sqli),
                Attack type (or null if benign),
                and justify your classification decision of each possible attack type based on the retrieved evidence.""",
    "reference_label": "sqli",
    "reference_attack_type": "error-based",
    "reference_justification": "Error-based SQLi detected in 'id' GET parameter using EXTRACTVALUE technique (payload: id=1' AND EXTRACTVALUE(7267,CONCAT(0x5c,0x7170626a71,(SELECT (ELT(7267=7267,1))),0x7178627871))-- ZxGr).",
  },
  {
    "case_id": "detect_005",
    "pcap_identifier": "capture-005.pcap",
    "user_input": """Analyze capture-005.pcap. Return exactly three fields:
                Detection (benign or sqli),
                Attack type (or null if benign),
                and justify your classification decision of each possible attack type based on the retrieved evidence.""",
    "reference_label": "sqli",
    "reference_attack_type": "time-based blind",
    "reference_justification": "Time-based blind SQLi detected in 'id' GET parameter using SLEEP function (payload: id=1' AND (SELECT 8265 FROM (SELECT(SLEEP(5)))vKVh)-- UgVG).",
  },
  {
    "case_id": "detect_006",
    "pcap_identifier": "capture-006.pcap",
    "user_input": """Analyze capture-006.pcap. Return exactly three fields:
                Detection (benign or sqli),
                Attack type (or null if benign),
```

```
                and justify your classification decision of each possible attack type based on the retrieved evidence.""",
        "reference_label": "sqli",
        "reference_attack_type": "union-based",
        "reference_justification": "UNION-based SQLi detected in 'id' GET parameter using UNION ALL to append a second SELECT (payload: id=1' UNION ALL SELECT NULL,CONCAT(0x7178626a71,0x73784961584a69447871497a785273786c4759514678785769567 84c714d79714151727367496c79,0x716a766271)-- -).",
    },
    {
        "case_id": "detect_007",
        "pcap_identifier": "capture-007.pcap",
        "user_input": """Analyze capture-007.pcap. Return exactly three fields:
                Detection (benign or sqli),
                Attack type (or null if benign),
                and justify your classification decision of each possible attack type based on the retrieved evidence.""",
        "reference_label": "sqli",
        "reference_attack_type": "multi-technique",
        "reference_justification": "Multi-technique SQLi detected in 'id' GET parameter using boolean-based blind, error-based (EXTRACTVALUE), time-based blind (SLEEP), and UNION query techniques (146 HTTP requests).",
    },
    {
        "case_id": "detect_008",
        "pcap_identifier": "capture-008.pcap",
        "user_input": """Analyze capture-008.pcap. Return exactly three fields:
                Detection (benign or sqli),
                Attack type (or null if benign),
                and justify your classification decision of each possible attack type based on the retrieved evidence.""",
        "reference_label": "benign",
        "reference_attack_type": None,
        "reference_justification": "No SQL injection patterns detected in the traffic.",
    },
    {
        "case_id": "detect_009",
        "pcap_identifier": "capture-009.pcap",
```

```
    "user_input": """Analyze capture-009.pcap. Return exactly three fields:
                Detection (benign or sqli),
                Attack type (or null if benign),
                and justify your classification decision of each possible attack type based on the retrieved
evidence.""",
    "reference_label": "benign",
    "reference_attack_type": None,
    "reference_justification": "No SQL injection patterns detected in the traffic.",
  },
  {
    "case_id": "detect_010",
    "pcap_identifier": "capture-010.pcap",
    "user_input": """Analyze capture-010.pcap. Return exactly three fields:
                Detection (benign or sqli),
                Attack type (or null if benign),
                and justify your classification decision of each possible attack type based on the retrieved
evidence.""",
    "reference_label": "benign",
    "reference_attack_type": None,
    "reference_justification": "No SQL injection patterns detected in the traffic.",
  },
  {
    "case_id": "detect_011",
    "pcap_identifier": "capture-011.pcap",
    "user_input": """Analyze capture-011.pcap. Return exactly three fields:
                Detection (benign or sqli),
                Attack type (or null if benign),
                and justify your classification decision of each possible attack type based on the retrieved
evidence.""",
    "reference_label": "benign",
    "reference_attack_type": None,
    "reference_justification": "No SQL injection patterns detected in the traffic.",
  },
  {
    "case_id": "detect_012",
    "pcap_identifier": "capture-012.pcap",
    "user_input": """Analyze capture-012.pcap. Return exactly three fields:
                Detection (benign or sqli),
```

```
                Attack type (or null if benign),
                and justify your classification decision of each possible attack type based on the retrieved
evidence.""",
    "reference_label": "benign",
    "reference_attack_type": None,
    "reference_justification": "No SQL injection patterns detected in the traffic.",
  },
  {
    "case_id": "detect_013",
    "pcap_identifier": "capture-013.pcap",
    "user_input": """Analyze capture-013.pcap. Return exactly three fields:
                Detection (benign or sqli),
                Attack type (or null if benign),
                and justify your classification decision of each possible attack type based on the retrieved
evidence.""",
    "reference_label": "sqli",
    "reference_attack_type": ["boolean-based or multi-technique"],
    "reference_justification": """Boolean-based blind SQLi detected in 'id' GET parameter using OR NOT
technique (payload: id=1' OR NOT 3966=3966#). Because
                the traffic also contains manual testing of other techniques, it can be classified as multi-
technique SQLi.""",
  },
  {
    "case_id": "detect_014",
    "pcap_identifier": "capture-014.pcap",
    "user_input": """Analyze capture-014.pcap. Return exactly three fields:
                Detection (benign or sqli),
                Attack type (or null if benign),
                and justify your classification decision of each possible attack type based on the retrieved
evidence.""",
    "reference_label": "sqli",
    "reference_attack_type": ["error-based or multi-technique"],
    "reference_justification": """Error-based SQLi detected in 'id' GET parameter using EXTRACTVALUE
function (payload: id=1' AND EXTRACTVALUE(7267,CONCAT(0x5c,0x7170626a71,(SELECT
(ELT(7267=7267,1))),0x7178627871))-- ZxGr).
                Because the traffic also contains manual testing of other techniques, it can be classified
as multi-technique SQLi.""",
  },
```

```
  {
    "case_id": "detect_015",
    "pcap_identifier": "capture-015.pcap",
    "user_input": """Analyze capture-015.pcap. Return exactly three fields:
                Detection (benign or sqli),
                Attack type (or null if benign),
                and justify your classification decision of each possible attack type based on the retrieved evidence.""",
    "reference_label": "sqli",
    "reference_attack_type": ["union-based or multi-technique"],
    "reference_justification": """UNION-based SQLi detected in 'id' GET parameter using UNION ALL to append a second SELECT (payload: id=1' UNION ALL SELECT NULL,CONCAT(0x7162767871,0x627343454b7a7061716c495642474c6467514f7079456a6d56436579426c4b5664764e7566705252,0x717a706271)-- -).
                            Because the traffic also contains manual testing of other techniques, it can be classified as multi-technique SQLi.""",
  },
  {
    "case_id": "detect_016",
    "pcap_identifier": "capture-016.pcap",
    "user_input": """Analyze capture-016.pcap. Return exactly three fields:
                Detection (benign or sqli),
                Attack type (or null if benign),
                and justify your classification decision of each possible attack type based on the retrieved evidence.""",
    "reference_label": "sqli",
    "reference_attack_type": ["time-based blind or multi-technique"],
    "reference_justification": """Time-based blind SQLi detected in 'id' GET parameter using SLEEP function (payload: id=1' AND (SELECT 5466 FROM (SELECT(SLEEP(5)))sdYx)-- OvgA).
                            Because the traffic also contains manual testing of other techniques, it can be classified as multi-technique SQLi.""",
  },
  {
    "case_id": "detect_017",
    "pcap_identifier": "capture-017.pcap",
    "user_input": """Analyze capture-017.pcap. Return exactly three fields:
                Detection (benign or sqli),
                Attack type (or null if benign),
```

```
                and justify your classification decision of each possible attack type based on the retrieved
evidence.""",
    "reference_label": "sqli",
    "reference_attack_type": "multi-technique",
    "reference_justification": "Multi-technique SQLi detected in 'id' GET parameter using boolean-based blind,
error-based (EXTRACTVALUE), time-based blind (SLEEP), and UNION query techniques (146 HTTP
requests).",
  },
  {
    "case_id": "detect_018",
    "pcap_identifier": "capture-018.pcap",
    "user_input": """Analyze capture-018.pcap. Return exactly three fields:
                Detection (benign or sqli),
                Attack type (or null if benign),
                and justify your classification decision of each possible attack type based on the retrieved
evidence.""",
    "reference_label": "sqli",
    "reference_attack_type": "multi-technique",
    "reference_justification": "Multi-technique SQLi detected in 'id' GET parameter using boolean-based blind,
error-based (EXTRACTVALUE), time-based blind (SLEEP), and UNION query techniques (146 HTTP
requests).",
  }
]
```

Υπεύθυνη Δήλωση Συγγραφέα:

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης.