ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Στατική Ανάλυση κώδικα με την βοήθεια μεγάλων γλωσσικών μοντέλων για ανίχνευση ευπαθειών σε web plugins»

«Εικόνα»

Του φοιτητή
Άγγελου Παπάζη
Αρ. Μητρώου: 2020130

Επιβλέπων
Ονοματεπώνυμο: Χαράλαμπος Μπράτσας

Ημερομηνία 23/05/2026

Τίτλος Δ.Ε.: Στατική Ανάλυση κώδικα με την βοήθεια μεγάλων γλωσσικών μοντέλων για ανίχνευση ευπαθειών σε web plugins

Κωδικός Δ.Ε.: 26174

Ονοματεπώνυμο φοιτητή: Άγγελος Παπάζης

Ονοματεπώνυμο εισηγητή: Χαράλαμπος Μπράτσας

Ημερομηνία ανάληψης Δ.Ε: 11/03/2025

Ημερομηνία περάτωσης Δ.Ε.: 23/05/2026

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Πρόλογος

Το θέμα της εργασίας προέκυψε από δύο παράλληλα ενδιαφέροντα, τα οποία συνδυάστηκαν κατά τη διάρκεια των προπτυχιακών μου σπουδών στο εν λόγω Τμήμα. Από τη μία, η ενασχόλησή μου με CTF challenges, μου κίνησε το ενδιαφέρον να εστιάσω την προσοχή μου στον τομέα της κυβερνοασφάλειας και παράλληλα διαμόρφωσε την επιθυμία μου να ασχοληθώ στο μέλλον επαγγελματικά με αυτόν. Από την άλλη, η ραγδαία ανάπτυξη των Μεγάλων Γλωσσικών Μοντέλων και των agentic συστημάτων με ώθησε να αναρωτηθώ πώς θα μπορούσε ενδεχομένως η τεχνολογία αυτή να αξιοποιηθεί για την ανίχνευση ευπαθειών λογισμικού.

Η εκπόνηση της εργασίας παρείχε ομολογουμένως την ευκαιρία για περαιτέρω εμβάθυνση τόσο στις αρχές της στατικής ανάλυσης ασφάλειας όσο και στη λειτουργία των agentic συστημάτων. Μέσα από τον σχεδιασμό και την εκτέλεση του πειράματος απέκτησα πρακτική εμπειρία στην αξιολόγηση γλωσσικών μοντέλων σε πραγματικές συνθήκες. Η διαδικασία αυτή με βοήθησε σημαντικά να κατανοήσω αφενός τις δυνατότητες και αφετέρου τα όρια της τεχνητής νοημοσύνης στον τομέα της κυβερνοασφάλειας, έναν τομέα που εξελίσσεται με ιλλιγιώδεις ρυθμούς και παρουσιάζει σημαντικές ερευνητικές προκλήσεις για το άμεσο μέλλον.

Περίληψη

Η αυτοματοποιημένη ανίχνευση ευπαθειών λογισμικού αποτελεί ένα κεντρικό ζήτημα της σύγχρονης ασφάλειας εφαρμογών. Είναι γεγονός ότι η ταχεία εξέλιξη των Μεγάλων Γλωσσικών Μοντέλων (LLMs) και των agentic αρχιτεκτονικών δημιουργεί νέες δυνατότητες σε αυτό τον τομέα, ωστόσο παραμένουν σε μεγάλο βαθμό ανεπαρκώς αξιολογημένες. Η παρούσα εργασία διερευνά τόσο τις δυνατότητες όσο και τους περιορισμούς LLM-based agentic συστημάτων στην ανίχνευση τεκμηριωμένων ευπαθειών ασφάλειας σε WordPress plugins μέσω στατικής ανάλυσης πηγαίου κώδικα. Το WordPress επιλέχθηκε λόγω της ευρείας χρήσης του και της διαθεσιμότητας τεκμηριωμένων ευπαθειών. Η μελέτη βασίζεται σε τέσσερα γλωσσικά μοντέλα - Qwen 3.6 Plus, GLM-5.1, Kimi K2.5 και MiniMax M2.7 – τα οποία αξιολογούνται μέσω του agentic πλαισίου OpenCode σε 10 plugins με γνωστές ευπάθειες. Κάθε plugin εξετάζεται αφενός στην ευάλωτη και αφετέρου στη διορθωμένη έκδοσή του. Κατόπιν, αποτιμάται η αποδοτικότητα των μοντέλων σύμφωνα με τις μετρικές Accuracy, Precision, Recall, F1-score και Hallucination rate, έχοντας ως σημείο αναφοράς τη βάση δεδομένων Wordfence Intelligence. Τα ευρήματα αναδεικνύουν ότι η δομική πολυπλοκότητα και η στατική ορατότητα της ευπάθειας αποτελούν τους σημαντικότερους παράγοντες απόδοσης. Ευπάθειες όπου η ροή δεδομένων περιορίζεται σε μεμονωμένες συναρτήσεις ή αρχεία, εντοπίστηκαν με υψηλότερα ποσοστά επιτυχίας από τα ισχυρότερα μοντέλα. Αντίθετα, ευπάθειες που εκτείνονται σε πολλαπλά αρχεία και απαιτούν την παρακολούθηση της απουσίας μηχανισμών ελέγχου, αποδείχθηκαν συστηματικά δυσκολότερες στην ανίχνευση. Το Kimi K2.5 παρουσίασε συνολικά τη καλύτερη απόδοση, ωστόσο κανένα μοντέλο δεν επέδειξε υψηλά επίπεδα ακρίβειας και αξιοπιστίας.

« Static Code Analysis Using Large Language Models for Vulnerability Detection in Web Plugins»

«Angelos Papazis»

Abstract

Automated vulnerability detection in software applications is a central challenge in modern application security. The rapid advancement of Large Language Models (LLMs) and agentic architectures introduces new possibilities in this field, yet these remain largely unevaluated under real-world conditions. This thesis investigates the capabilities and limitations of LLM-based agentic systems in detecting documented security vulnerabilities in WordPress plugins through static source code analysis. WordPress was selected due to its widespread adoption and the availability of well-documented vulnerabilities. The study evaluates four language models - Qwen 3.6 Plus, GLM-5.1, Kimi K2.5, and MiniMax M2.7 - through the OpenCode agentic framework across ten plugins with known vulnerabilities. Each plugin is examined in both its vulnerable and patched versions. Model performance is then assessed using Accuracy, Precision, Recall, F1-score and Hallucination rate metrics, with the Wordfence Intelligence database serving as the ground truth reference. The findings reveal that structural complexity and the static visibility of a vulnerability are the most significant factors influencing detection performance. Vulnerabilities where the data flow is confined to isolated functions or individual files were detected at higher success rates by the stronger models. In contrast, vulnerabilities that span multiple files and require tracking the absence of security control mechanisms proved systematically more difficult to identify. Kimi K2.5 achieved the best overall performance, yet no model demonstrated high levels of accuracy and reliability.

Ευχαριστίες

Ευχαριστώ τον επιβλέποντα καθηγητή μου, Χαράλαμπο Μπράτσα, για την βοήθεια και τις χρήσιμες συμβουλές του.

Περιεχόμενα

Κατάλογος εικόνων

Κατάλογος πινάκων

Κεφάλαιο 1ο: Εισαγωγή

1.1 Κίνητρο και πλαίσιο έρευνας

Οι διαδικτυακές εφαρμογές αποτελούν πλέον το κύριο μέσον παροχής υπηρεσιών σε τομείς όπως η υγεία, η εκπαίδευση, τα χρηματο-οικονομικά και το ηλεκτρονικό εμπόριο. Η ολοένα αυξανόμενη χρήση τους έχει καταστήσει την ασφάλειά τους εξαιρετικά σημαντική. Οι ευπάθειες που μπορεί να περιέχουν είναι πολύ πιθανό να επηρεάσουν, τόσο τους οργανισμούς που τις αναπτύσσουν, όσο και τους χρήστες που στηρίζονται σε αυτές. Στο πλαίσιο αυτό, η ερευνητική κοινότητα στρέφεται όλο και συχνότερα στην αυτοματοποίηση της ανάλυσης ασφάλειας, με στόχο την ανάπτυξη εργαλείων, τα οποία εντοπίζουν ευπάθειες με αξιοπιστία αλλά και με συστηματικότητα. Όταν επομένως πρόκειται αυτά τα εργαλεία να χρησιμοποιηθούν σε πραγματικές συνθήκες ανάπτυξης λογισμικού, είναι αναγκαία η συστηματική αξιολόγησή τους [1].

Η πλέον καθιερωμένη μέθοδος αυτοματοποιημένης ανάλυσης ασφάλειας είναι η στατική ανάλυση ασφάλειας πηγαίου κώδικα (SAST), η οποία εξετάζει τον κώδικα δίχως να απαιτεί και την παράλληλη εκτέλεσή του. Ωστόσο, τα παραδοσιακά εργαλεία SAST, εμφανίζουν ένα δομικό περιορισμό. Συγκεκριμένα, βασίζονται κυρίως σε προκαθορισμένους κανόνες και γνωστά μοτίβα αντιστοίχισης. Για τον λόγο αυτό, παρουσιάζουν δυσκολίες στον εντοπισμό ευπαθειών που απαιτούν βαθύτερη κατανόηση της λογικής του κώδικα ή την παρακολούθηση της ροής δεδομένων ανάμεσα σε πολλά αρχεία και συναρτήσεις [2].

Τα τελευταία χρόνια η ταχύτατη εξέλιξη των Μεγάλων Γλωσσικών Μοντέλων (LLMs) έχει δημιουργήσει καινούργιες δυνατότητες για την ανάλυση ασφάλειας λογισμικού. Σε αντίθεση με τα παραδοσιακά εργαλεία SAST , τα LLMs επεξεργάζονται τον κώδικα ως μορφή γλώσσας και εκπαιδεύονται επάνω σε πολύ μεγάλα σύνολα δεδομένων. Με τον τρόπο αυτό, είναι ικανά να αναγνωρίζουν σχέσεις και μοτίβα, που συχνά δεν είναι εφικτό να εντοπιστούν από προσεγγίσεις, οι οποίες στηρίζονται αποκλειστικά σε κανόνες [3]. Έντονο ενδιαφέρον εμφανίζει και η ενσωμάτωση των LLMs σε agentic αρχιτεκτονικές. Τα συστήματα αυτά έχουν τη δυνατότητα να κάνουν χρήση εργαλείων, να πλοηγούνται αυτόνομα σε αποθετήρια κώδικα και να εκτελούν ακολουθίες ενεργειών κατά την ανάλυση. Με αυτόν τον τρόπο, η διαδικασία ανάλυσης ασφάλειας γίνεται πιο δυναμική και προσαρμοστική [4].

Η παρούσα εργασία πραγματεύεται την προσέγγιση αυτή αξιοποιώντας ως πεδίο εφαρμογής το οικοσύστημα των WordPress plugins. Το WordPress συνιστά μία από τις πιο διαδεδομένες πλατφόρμες διαχείρισης ιστοσελίδων παγκοσμίως. Η λειτουργικότητά του επεκτείνεται μέσω plugins, δηλαδή μέσω πρόσθετων λογισμικών επεκτάσεων που αναπτύσσονται ως επί το πλείστον από τρίτους προγραμματιστές. Στις ημέρες μας είναι διαθέσιμα περισσότερα από 54.000 plugins, ενώ το WordPress χρησιμοποιείται από ένα εξαιρετικά μεγάλο ποσοστό ιστοσελίδων που βασίζονται σε συστήματα διαχείρισης περιεχομένου. Η ευρεία χρήση του, συνδυαστικά με τον ανοιχτό χαρακτήρα του οικοσυστήματος, το καθιστούν έναν από τους πιο διαδεδομένους στόχους επιθέσεων στο διαδίκτυο. Παράλληλα, πολλά plugins αναπτύσσονται

χωρίς τυποποιημένες διαδικασίες ελέγχου ασφάλειας, με αποτέλεσμα να αυξάνεται η πιθανότητα εμφάνισης ευπαθειών. Τα χαρακτηριστικά αυτά καθιστούν το WordPress κατάλληλο περιβάλλον για την αξιολόγηση της αποτελεσματικότητας LLM based agentic συστημάτων στην ανίχνευση ευπαθειών [5].

1.2 Στόχοι και διάρθρωση

Ο στόχος της μελέτης αυτής είναι η διερεύνηση της ικανότητας των LLM-based agentic συστημάτων να εντοπίζουν ευπάθειες σε WordPress plugins, κάτω από ελεγχόμενες πειραματικές συνθήκες. Προκειμένου να επιτευχθεί ο στόχος αυτός σχεδιάστηκε μια δομημένη μεθοδολογία αξιολόγησης, η οποία αξιοποιεί το agentic πλαίσιο OpenCode και τέσσερα LLMs προσανατολισμένα σε agentic εργασίες και ανάλυση κώδικα - Qwen 3.6 Plus, GLM-5.1, Kimi K2.5 και MiniMax M2.7 - τα οποία αναλύουν δέκα WordPress plugins με γνωστές, τεκμηριωμένες ευπάθειες. Κάθε plugin εξετάζεται σε δύο εκδόσεις: α) την ευάλωτη, με γενικό prompt, και β) τη διορθωμένη, με prompt στοχευμένο στο κατάλληλο CVE. Ως σημείο αναφοράς χρησιμοποιείται η εξειδικευμένη βάση δεδομένων Wordfence Intelligence και η αξιολόγηση γίνεται με βάση τις μετρικές Accuracy, Precision, Recall, F1-score και Hallucination rate.

Η εργασία διαρθρώνεται στα ακόλουθα κεφάλαια:

- Στο Κεφάλαιο 2 θεμελιώνεται η δομή της εργασίας με την παρουσίαση της επιστημονικής περιοχής της SAST. Εδώ αναλύονται οι κύριες τεχνικές που χρησιμοποιούν τα εργαλεία SAST, όπως και τα πλεονεκτήματα και οι περιορισμοί τους. Επιπρόσθετα, η SAST τοποθετείται στο ευρύτερο πλαίσιο των μεθόδων ελέγχου ασφάλειας, σε σύγκριση με τη δυναμική (DAST) και τη διαδραστική ανάλυση (IAST).

- Στο Κεφάλαιο 3 παρουσιάζονται οι βασικές κατηγορίες ευπαθειών σε διαδικτυακές εφαρμογές, οι οποίες εξετάζονται στο πείραμα και απαρτίζονται από τις: SQL Injection, Cross-Site Scripting, Remote Code Execution, Cross-Site Request Forgery. Επιπλέον, αναδεικνύεται το εννοιολογικό πλαίσιο ταξινόμησής τους μέσω OWASP, CWE και CVE, καθώς και οι βάσεις δεδομένων που χρησιμοποιούνται ως πηγές τεκμηρίωσης.

- Στο Κεφάλαιο 4 εισάγονται τα LLMs ως εργαλεία ανάλυσης κώδικα. Αρχικά, παρουσιάζεται η αρχιτεκτονική Transformer και ο μηχανισμός προσοχής. Ακολούθως, αναλύονται οι τεχνικοί περιορισμοί των LLMs, οι οποίοι σχετίζονται με το tokenization και το context window. Το κεφάλαιο αυτό ολοκληρώνεται με την ανάδειξη του φαινομένου των hallucinations.

- Στο Κεφάλαιο 5 αναπτύσσεται η έννοια των αυτόνομων πρακτόρων (AI agents) και οι αρχιτεκτονικές αρχές που τους διέπουν. Γίνεται ανάλυση του φάσματος της agentic συμπεριφοράς, καθώς και των βασικών δομικών στοιχείων ενός agent και το μοτίβο ReAct. Κατόπιν, γίνεται μια επισκόπηση της υπάρχουσας βιβλιογραφίας γύρω από τη χρήση LLM-based agents στην ανίχνευση ευπαθειών. Το κεφάλαιο ολοκληρώνεται με

την καταγραφή των περιορισμών των AI agents.

- Στο Κεφάλαιο 6 ασχολούμαστε αναλυτικά με τη μεθοδολογία που ακολουθήθηκε στο πείραμα. Στην αρχή περιγράφεται η διαδικασία επιλογής των WordPress plugins και η αιτιολόγηση της χρήσης τους στο πλαίσιο της έρευνας. Στη συνέχεια αναλύεται η διαδικασία συλλογής και επαλήθευσης των τεκμηριωμένων δεδομένων αναφοράς, όπως και ο σχεδιασμός των δύο πειραματικών συνθηκών που χρησιμοποιήθηκαν για την αξιολόγηση των μοντέλων. Καταληκτικά, παρουσιάζεται το πλαίσιο αξιολόγησης και οι μετρικές ταξινόμησης, οι οποίες αξιοποιήθηκαν για την αποτίμηση των αποτελεσμάτων.

- Στο Κεφάλαιο 7 αναφέρονται τα αποτελέσματα της πειραματικής αξιολόγησης για καθένα από τα 10 plugins, συμπεριλαμβανομένων των prompts που χρησιμοποιήθηκαν για τις διορθωμένες εκδόσεις και των πινάκων των αποτελεσμάτων ανά μοντέλο.

- Στο Κεφάλαιο 8 παρουσιάζεται η συνολική ανάλυση των αποτελεσμάτων του πειράματος. Στην αρχή πραγματοποιείται συγκριτική αξιολόγηση των μοντέλων, ενώ στη συνέχεια αναδεικνύονται τα κοινά μοτίβα και οι τάσεις που προέκυψαν από τα πειραματικά ευρήματα. Στο τέλος, τα αποτελέσματα συνδέονται με το θεωρητικό πλαίσιο της εργασίας και ερμηνεύονται σε σχέση με τις δυνατότητες και τους περιορισμούς των LLM based agentic συστημάτων.

- Στο Κεφάλαιο 9 συνοψίζονται τα συμπεράσματα της εργασίας και σκιαγραφούνται κατευθύνσεις για μελλοντική έρευνα.

Κεφάλαιο 2ο: Στατική ανάλυση ασφάλειας πηγαίου κώδικα (SAST)

2.1 Εισαγωγή

Μία από τις σημαντικότερες προκλήσεις της σύγχρονης μηχανικής λογισμικού συνιστά η ανάπτυξη ενός ασφαλούς λογισμικού. Επειδή οι εφαρμογές γίνονται ολοένα και περισσότερο σύνθετες και εξαρτώνται από εκτεταμένες βιβλιοθήκες και εξωτερικές διεπαφές, έχει μεγάλη σημασία ο εντοπισμός των ευπαθειών πριν από την ανάπτυξη σε παραγωγικό περιβάλλον. Με βάση την έκθεση της Verizon για το 2023, οι επιθέσεις σε διαδικτυακές εφαρμογές αντιστοιχούν σχεδόν στο 40% των παραβιάσεων δεδομένων, που έχουν καταγραφεί [6]. Η ανίχνευση των ευπαθειών εγκαίρως, δηλαδή προτού το λογισμικό διατεθεί στους χρήστες, μειώνει σημαντικά το κόστος της αποκατάστασης και τον κίνδυνο της εκμετάλλευσης.

Σε αυτό το πλαίσιο, η Στατική Ανάλυση Ασφάλειας Πηγαίου Κώδικα (Static Application Security Testing - SAST) συνιστά μία από τις θεμελιώδεις μεθόδους ανίχνευσης των ευπαθειών. Το κύριο χαρακτηριστικό στοιχείο της SAST είναι ότι διερευνά τον πηγαίο κώδικα δίχως να εκτελεί την εφαρμογή [7]. Η ιδιότητα αυτή την κατατάσσει στις μεθόδους white-box ανάλυσης. Κατά τη διαδικασία της white-box ανάλυσης, δίνεται πλήρης πρόσβαση στον κώδικα, γεγονός που παρέχει τη δυνατότητα μιας διερεύνησης σε βάθος, τόσο της δομής και της ροής των δεδομένων, όσο και της λογικής του προγράμματος. Η ορατότητα του κώδικα επιτρέπει στον αναλυτή να μελετά τον τρόπο με τον οποίο έχει πραγματοποιηθεί μια εφαρμογή και να συνεκτιμά αξιολογικά τις αδυναμίες που συνδέονται με τη λογική, τη δομή και τη χρήση επικίνδυνων προγραμματιστικών πρακτικών. Η μέθοδος της ανάλυσης πραγματοποιείται στον κώδικα σε κατάσταση ηρεμίας, χωρίς να απαιτούνται εκτελέσιμο ενεργό περιβάλλον, διακομιστής ή δεδομένα εισόδου από πραγματικούς χρήστες [8].

Ένα από τα πλεονεκτήματα αυτής της τεχνικής είναι η ένταξή της στα αρχικά στάδια του κύκλου ζωής και ανάπτυξης ενός λογισμικού. Η δυνατότητα αυτή έχει μεγάλη πρακτική αξία, καθώς ένα πρόβλημα που εντοπίζεται εγκαίρως, είναι συνήθως ευκολότερο, ταχύτερο και οικονομικότερο να διορθωθεί συγκριτικά με ένα πρόβλημα που ανακαλύπτεται αργότερα σε περιβάλλον δοκιμών ή παραγωγής. Για τον λόγο αυτό, τα εργαλεία SAST, ενσωματώνονται συνήθως σε αυτοματοποιημένες ροές ανάπτυξης, και με τον τρόπο αυτό επιτρέπουν τον συστηματικό έλεγχο του κώδικα σε κάθε στάδιο της διαδικασίας. Παράλληλα, η χρησιμότητά τους δεν εξαντλείται αποκλειστικά και μόνο στα πρώτα στάδια της ανάπτυξης, αφού υπάρχει η δυνατότητα να αξιοποιηθούν και σε ήδη υφιστάμενα αποθετήρια κώδικα για την επισήμανση των αδυναμιών, οι οποίες δεν είχαν νωρίτερα αναγνωριστεί.

Για να κατανοηθεί σαφέστερα η θέση της SAST στο ευρύτερο πλαίσιο του ελέγχου της ασφάλειας, θα μπορούσε να διερευνηθεί συγκριτικά με άλλες δύο ευρέως γνωστές μεθόδους: α) τη Δυναμική Ανάλυση Ασφάλειας (Dynamic Application Security Testing - DAST) και β) τη Διαδραστική Ανάλυση Ασφάλειας (Interactive Application Security Testing - IAST). Στην DAST εξετάζεται μια εφαρμογή όσο αυτή εκτελείται, αλληλεπιδρώντας με τις εξωτερικές διεπαφές της και προσομοιώνοντας τη συμπεριφορά ενός εισβολέα, ο οποίος δεν έχει

πρόσβαση στον πηγαίο κώδικα (black-box). Αυτό σημαίνει ότι η DAST έχει τη δυνατότητα να εντοπίσει μόνον ευπάθειες, οι οποίες εκδηλώνονται κατά την εκτέλεση και είναι ορατές δια των εξωτερικών διασυνδέσεων, αλλά δεν μπορεί να ανιχνεύσει τις αδυναμίες, οι οποίες κρύβονται στη λογική ή τη δομή του κώδικα [9]. Αντίθετα, στην περίπτωση της IAST, υπάρχει μια περισσότερο ενδιάμεση προσέγγιση, στην οποία συνδυάζονται η ορατότητα του κώδικα με την ανάλυση της πραγματικής συμπεριφοράς. Μολονότι η IAST συνδυάζει στοιχεία και από τις δύο μεθόδους, απαιτεί εκτελέσιμο περιβάλλον και πλήρη ενσωμάτωση στην εφαρμογή γεγονός που την καθιστά δυσκολότερη στην αξιοποίηση σε πρώιμα στάδια ανάπτυξης [10].

2.2 Οι τεχνικές ανάλυσης των εργαλείων της SAST

Η ανάλυση ενός πηγαίου κώδικα είναι ομολογουμένως μια απαιτητική διαδικασία, διότι προϋποθέτει εξειδικευμένες γνώσεις και είναι εξαιρετικά χρονοβόρα. Παραδοσιακά, ο εντοπισμός των ευπαθειών στηριζόταν ανέκαθεν σε χειροκίνητες δοκιμές και σε ελέγχους διείσδυσης, οι οποίοι, αν και αρκετά αποτελεσματικοί, δεν εφαρμόζονται εύκολα σε πολύπλοκα και εκτεταμένα έργα λογισμικού. Σε αυτό ακριβώς το πρόβλημα μπορούν να προσφέρουν τη λύση τα εργαλεία της SAST, τα οποία αυτοματοποιούν τη διαδικασία της ανάλυσης του πηγαίου κώδικα. Παράλληλα, παράγουν αναφορές με ευρύματα και πιθανούς κινδύνους που θα μπορούσαν να οδηγήσουν σε παραβιάσεις της ασφάλειας [11].

Τα εργαλεία της SAST στηρίζονται κατά κανόνα σε περισσότερο παραδοσιακές τεχνικές μεθόδους της στατικής ανάλυσης, όπως είναι η συντακτική και η σημασιολογική ανάλυση, αλλά και η σύγκριση με γνωστά πρότυπα ευπαθειών. Οι ανωτέρω τεχνικές παρέχουν στα εργαλεία τη δυνατότητα να διερευνούν τον πηγαίο κώδικα, δίχως να απαιτείται η εκτέλεση του προγράμματος, με στόχο τον εντοπισμό ενδεχόμενων αδυναμιών της ασφάλειας ήδη από τα αρχικά στάδια της ανάπτυξης. Η τεχνική μέθοδος της συντακτικής ανάλυσης (syntactic analysis) χρησιμοποιείται για την κατανόηση της δομής του πηγαίου κώδικα, με έμφαση στη γραμματική μορφή του προγράμματος, έτσι ώστε να εντοπίζονται πιθανά προβλήματα [12] [11]. Στο πλαίσιο αυτό, ελέγχεται η ορθή χρήση των μεταβλητών, των συναρτήσεων, των τελεστών και των δομών ελέγχου, ακολουθώντας πάντοτε τους κανόνες της εκάστοτε γλώσσας προγραμματισμού.

Εκτός από τη συντακτική ανάλυση, ιδιαίτερη αξία παρουσιάζει η σημασιολογική ανάλυση (semantic analysis), η οποία διερευνά τον τρόπο με τον οποίο αλληλεπιδρούν τα επιμέρους τμήματα του κώδικα αναμεταξύ τους. Ειδικότερα, εξετάζει τη διαχείριση των δεδομένων, των μεταβλητών και των συναρτήσεων στο συνολικό πλαίσιο της εφαρμογής [13]. Αυτού του είδους η μεθοδολογία παρέχει τη δυνατότητα να εντοπιστούν πρότυπα και ροές δεδομένων, που υποδηλώνουν την ύπαρξη ευπαθειών, όπως σημεία εισαγωγής κακόβουλου κώδικα ή εσφαλμένος χειρισμός μηχανισμών αυθεντικοποίησης και εξουσιοδότησης. Με παρόμοιο στόχο, το code scanning αναζητά στον στατικό κώδικα πρότυπα και πρακτικές που συσχετίζονται με ενδεχόμενες αδυναμίες ασφάλειας [14], όπως SQL injection, cross-site scripting (XSS), και με εσφαλμένη διαχείριση των διαδικασιών ταυτοποίησης και του ελέγχου πρόσβασης.

Επιπρόσθετα, κάποια εργαλεία αξιοποιούν τις τεχνικές code similarity, δια των οποίων

εντοπίζονται τμήματα του κώδικα με ομοιότητες ως προς τη δομή, τα οποία υπάρχει ενδεχόμενο να υποδεικνύουν προβληματικές περιοχές του συστήματος [15]. Μολονότι αυτή η ομοιότητα δεν συνεπάγεται πάντοτε την ύπαρξη κάποιας ευπάθειας, θα μπορούσε ίσως να λειτουργήσει σαν ένδειξη για επιπλέον διερεύνηση. Άλλη μια τεχνική που παρουσιάζει συνάφεια είναι και η τεχνική pattern matching, η οποία επικεντρώνεται στην αναζήτηση συγκεκριμένων προτύπων κώδικα, τα οποία ιστορικά έχουν συνδεθεί είτε με γνωστές ευπάθειες είτε με μη ασφαλείς προγραμματιστικές πρακτικές [16]. Έτσι, τα εργαλεία της SAST έχουν τη δυνατότητα να αξιοποιούν τη συσσωρευμένη γνώση από περιπτώσεις αδυναμιών που έχουν ήδη καταγραφεί και να τη μετατρέπουν σε αυτοματοποιημένους μηχανισμούς εντοπισμού.

Εξαιρετικά σημαντική θέση ανάμεσα σε αυτές τις τεχνικές έχει και η taint analysis, η οποία χρησιμοποιείται για τον εντοπισμό και την αξιολόγηση της διάδοσης μη έμπιστων ή δυνητικά κακόβουλων δεδομένων μέσα στο σύστημα [17], [18]. Η τεχνική αυτή ελέγχει τις εισόδους από την πηγή τους και διερευνά αν αυτές καταλήγουν σε καίρια σημεία του κώδικα, γνωστά ως sinks, δίχως να έχει προηγηθεί κατάλληλη απολύμανση ή έλεγχος. Στην περίπτωση αυτή, το σύστημα θεωρεί ότι υπάρχει πιθανή απειλή. Ωστόσο, η αποτελεσματικότητα της taint analysis επηρεάζεται από την ικανότητα της διερεύνησης όλων των ενδεχόμενων διαδρομών εκτέλεσης, γεγονός που μπορεί να οδηγήσει σε παραλείψεις [19].

Μια ακόμη θεμελιώδη τεχνική συνιστά η χρήση των γράφων ροής ελέγχου, γνωστών ως Control-Flow Graphs (CFGs). Μέσω αυτών, ο πηγαίος κώδικας μετατρέπεται σε μια αφηρημένη γραφική αναπαράσταση, στην οποία οι κόμβοι αντιστοιχούν σε βασικά τμήματα εντολών και οι ακμές αναπαριστούν τις πιθανές μεταβάσεις ελέγχου μεταξύ τους. Με τον τρόπο αυτό, αποβαίνει δυνατή η αποτύπωση όλων των ενδεχόμενων διαδρομών, τις οποίες μπορεί να ακολουθήσει το πρόγραμμα κατά την εκτέλεσή του. Η ανάκτηση του CFG πραγματοποιείται κυρίως μέσω αναδρομικών αλγορίθμων, οι οποίοι αποσυναρμολογούν και αναλύουν τα επιμέρους βασικά μπλοκ του κώδικα [20].

Στο σύνολό τους, οι τεχνικές αυτές συνιστούν τον πυρήνα της λειτουργίας των εργαλείων της SAST και έχουν τη δυνατότητα να εντοπίσουν ένα ευρύ φάσμα αδυναμιών, από απλά σφάλματα προγραμματισμού μέχρι τις πλέον σύνθετες αδυναμίες ενός σχεδιασμού. Η συνδυαστική αξιοποίησή τους επιτρέπει την πραγματοποίηση μιας περισσότερο ολοκληρωμένης και συστηματικής ανάλυσης του πηγαίου κώδικα, συνεισφέροντας κατ' ουσίαν στην έγκαιρη αναγνώριση, τη διόρθωση και τον μετριασμό των ευπαθειών, προτού διατεθεί ένα λογισμικό σε ένα παραγωγικό περιβάλλον.

2.3 Πλεονεκτήματα και περιορισμοί των εργαλείων της SAST

Αρκετά είναι τα πλεονεκτήματα που εμφανίζει η ενσωμάτωση των εργαλείων της SAST στη διαδικασία της ασφαλούς ανάπτυξης ενός λογισμικού. Πολλά σύγχρονα εργαλεία της SAST υποστηρίζουν πολλές γλώσσες προγραμματισμού, με αποτέλεσμα να αποβαίνουν κατάλληλα για ετερογενή περιβάλλοντα ανάπτυξης. Η διαρκής χρήση τους συμβάλλει στη διατήρηση ενός πιο καθαρού και περισσότερο ασφαλούς κώδικα και παράλληλα ενισχύει πιο ορθές πρακτικές ανάπτυξης μεταξύ των προγραμματιστών [21], [19]. Καταληκτικά, τα εργαλεία, όπως το Semgrep, παρέχουν τις δυνατότητες για παραμετροποίηση και για δημιουργία

προσαρμοσμένων κανόνων αλλά και για προσαρμογή της ανάλυσης στις ιδιαιτερότητες κάθε έργου [22].

Παρά τα ανωτέρω οφέλη, η χρήση των εργαλείων SAST έχει και σημαντικούς περιορισμούς. Ένα από τα πιο συχνά εμφανιζόμενα προβλήματα είναι η παραγωγή false positives, δηλαδή προειδοποιήσεων, οι οποίες τελικά δεν αντιστοιχούν σε πραγματικές ευπάθειες. Στην περίπτωση αυτή αυξάνεται ο φόρτος της εργασίας των προγραμματιστών, επειδή απαιτείται χειροκίνητη επιβεβαίωση κάθε ευρήματος, προκειμένου να αξιολογηθεί η εγκυρότητά του [23]. Επίσης, κανένα εργαλείο της SAST δεν έχει τη δυνατότητα να εντοπίσει μόνο του όλες τις ενδεχόμενες ευπάθειες. Αν και ο συνδυασμός πιο πολλών εργαλείων θα μπορούσε να βελτιώσει την κάλυψη, η διαχείριση πολλών λύσεων γίνεται συχνά περισσότερο σύνθετη και απαιτητική [24].

Ένας επιπλέον περιορισμός αφορά στην καμπύλη της μάθησης και στις απαιτήσεις της παραμετροποίησης. Η ορθή ρύθμιση των εργαλείων της SAST και η δημιουργία εξειδικευμένων κανόνων απαιτούν χρόνο, τεχνογνωσία και εμπειρία και μπορούν να αποβούν εμπόδιο για ορισμένες ομάδες ανάπτυξης. Συγχρόνως, η αποτελεσματικότητα ενός εργαλείου εξαρτάται σημαντικά τόσο από την ποιότητα όσο και από την πληρότητα του συνόλου των κανόνων που διαθέτει. Οι κανόνες, όταν διατυπώνονται ελιππώς ή ανεπαρκώς, μπορούν να οδηγήσουν στην παράλειψη σοβαρών ευπαθειών. Επιπρόσθετα, σε μεγαλύτερα έργα λογισμικού, η στατική ανάλυση μπορεί να είναι και χρονοβόρα και απαιτητική σε υπολογιστικούς πόρους, με συνέπεια ενδεχομένως να επηρεάζεται αρνητικά η απόδοση της διαδικασίας ανάπτυξης [15], [19], [21].

Καταληκτικά, τα εργαλεία της SAST εμφανίζουν περιορισμούς στον εντοπισμό ορισμένων κατηγοριών ευπαθειών, ιδίως εκείνων οι οποίες εξαρτώνται από συνθήκες ταυτόχρονης εκτέλεσης, όπως οι race conditions, ή από τη δυναμική ροή της εκτέλεσης του προγράμματος. Τέτοιου είδους αδυναμίες δεν είναι εύκολο συχνά να εντοπιστούν αποκλειστικά με τη στατική ανάλυση και ενδέχεται να απαιτούν τη συμπληρωματική χρήση δυναμικών τεχνικών ελέγχου, όπως η DAST [15]. Επομένως, η SAST αποτελεί ένα εξαιρετικά επωφελές εργαλείο για την ενίσχυση της ασφάλειας ενός λογισμικού, του οποίου η αποτελεσματικότητα εξαρτάται από την ορθή παραμετροποίηση και την ερμηνεία των αποτελεσμάτων αλλά και την αξιοποίησή του σε συνδυασμό με άλλες πρακτικές ασφαλούς ανάπτυξης.

Κεφάλαιο 3ο: Κατηγορίες ευπαθειών σε διαδικτυακές εφαρμογές

3.1 Εισαγωγή και πλαίσια ταξινόμησης ευπαθειών

Η αξιολόγηση της ικανότητας ενός γλωσσικού μοντέλου να εντοπίζει ευπάθειες στην ασφάλεια των εφαρμογών, είναι αδύνατον να γίνει χωρίς να υπάρχει ένα σαφές εννοιολογικό υπόβαθρο. Πρωτίστως, απαιτείται ένας κοινά αποδεκτός ορισμός για το τι ακριβώς συνιστά ευπάθεια, δευτερευόντως, απαιτείται ένα ταξινομικό πλαίσιο, το οποίο να επιτρέπει τη συστηματική κατηγοριοποίησή τους και, τέλος, απαιτείται ένα αξιόπιστο σύστημα αναγνώρισης που να συνδέει τις αφηρημένες κατηγορίες με τα πραγματικά περιστατικά. Το παρόν κεφάλαιο πραγματεύεται ακριβώς αυτήν τη λειτουργία, παρέχοντας στον αναγνώστη το θεωρητικό υπόβαθρο, το οποίο είναι αναγκαίο, προκειμένου να ερμηνευτούν ορθά τα αποτελέσματα του πειράματος.

Στο πεδίο της ασφάλειας διαδικτυακών εφαρμογών, η πιο ευρέως αναγνωρισμένη προσπάθεια συστηματικής ταξινόμησης ευπαθειών ανήκει στον οργανισμό OWASP (Open Worldwide Application Security Project). Πρόκειται για μια μη κερδοσκοπική διεθνή κοινότητα, η οποία ιδρύθηκε το 2001, με κεντρικό σκοπό τη βελτίωση της ασφάλειας λογισμικού μέσω εργαλείων ανοιχτής γνώσης, και προτύπων που διατίθενται ελεύθερα σε όλους [25]. Η κεντρική συνεισφορά του OWASP είναι η λίστα OWASP Top 10, η οποία αποτελεί μια περιοδικά ενημερωμένη κατάταξη των δέκα πιο κρίσιμων κατηγοριών κινδύνου που αντιμετωπίζουν οι σύγχρονες διαδικτυακές εφαρμογές. Η λίστα δεν αποτελεί απλώς μια στατιστική καταγραφή, αλλά ένα τεκμηριωμένο έγγραφο ευαισθητοποίησης που βασίζεται σε δεδομένα από εκατοντάδες οργανισμούς και εκατομμύρια εφαρμογές [26]. Η επίδρασή της επεκτείνεται και εκτός της ακαδημαϊκής κοινότητας, καθώς τόσο οι κυβερνητικοί φορείς όσο και τα διεθνή κανονιστικά πλαίσια αναφέρονται ρητά στο OWASP Top 10 ως τη βασική απαίτηση για συμμόρφωση.

Ένα καίριο χαρακτηριστικό της λίστας είναι η εξέλιξή της με την πάροδο του χρόνου. Στις εκδόσεις του 2013 και του 2017 κατατάσσονταν οι συγκεκριμένες τεχνικές ευπάθειες μεμονωμένα και παρέχονταν για κάθε μία ξεχωριστή θέση στη λίστα. Στην έκδοση του 2021 εισήχθη μια ουσιαστική αλλαγή στη μέθοδο της ταξινόμησης. Συγκεκριμένα αντί να αναφέρεται σε μεμονωμένες ευπάθειες, οργάνωσε το περιεχόμενό της σε ευρύτερες κατηγορίες κινδύνου, καθεμία από τις οποίες ομαδοποιεί πολλαπλούς συναφείς τύπους αδυναμιών [27]. Παραδείγματος χάριν, η κατηγορία A03 – Injection της έκδοσης 2021 καλύπτει SQL Injection, Remote Code Execution και Cross-Site Scripting. Με τον τρόπο αυτόν οι ανωτέρω ευπάθειες αντιμετωπίζονται ως εκφάνσεις του ίδιου θεμελιώδους προβλήματος, δηλαδή της απουσίας του διαχωρισμού μεταξύ δεδομένων και εκτελέσιμου κώδικα. Η πιο πρόσφατη έκδοση της λίστας, που κυκλοφόρησε τον Νοέμβριο του 2025, εισήγαγε δύο νέες κατηγορίες - Software Supply Chain Failures και Mishandling of Exceptional Conditions - αντικατοπτρίζοντας τη μετατόπιση των σύγχρονων απειλών προς την πολυπλοκότητα της αλυσίδας ανάπτυξης λογισμικού [28].

Η σύνδεση του OWASP με ένα περισσότερο τεχνικό επίπεδο ταξινόμησης πραγματοποιείται

μέσω του Common Weakness Enumeration (CWE), ενός καταλόγου τυποποιημένων κατηγοριών αδυναμιών λογισμικού που συντηρείται από το MITRE Corporation υπό την αιγίδα του Εθνικού Ινστιτούτου Προτύπων και Τεχνολογίας των ΗΠΑ (NIST) [29]. Κάθε κατηγορία OWASP αντιστοιχεί σε ένα σύνολο CWE αναγνωριστικών, και παρέχει τη δυνατότητα για την ακριβέστερη τεχνική περιγραφή παντός τύπου αδυναμίας. Ενδεικτικά, η SQL Injection αντιστοιχεί στο CWE-89, το Stored XSS στο CWE-79 και το CSRF στο CWE-352. Το CWE λειτουργεί ως κοινή γλώσσα μεταξύ ερευνητών, εργαλείων ανάλυσης και οργανισμών, επιτρέποντας τη συστηματική αντιστοίχιση ευρημάτων από διαφορετικά εργαλεία SAST σε μια ενιαία ταξινόμηση.

Στο σημείο αυτό οφείλουμε να καταστήσουμε σαφή τη θεμελιώδη διάκριση ανάμεσα στο CWE και στο CVE (Common Vulnerabilities and Exposures). Το CWE αφορά την ταξινόμηση των τύπων αδυναμιών και των ευπαθειών σε γενικό επίπεδο, ενώ το CVE χρησιμοποιείται για την καταγραφή πραγματικών ευπαθειών, οι οποίες εντοπίζονται σε συγκεκριμένες εκδόσεις λογισμικού [30]. Ένα CVE, δείγματος χάριν, μπορεί να αναφέρει ότι η έκδοση μιας εφαρμογής εμπεριέχει ευπάθεια τύπου CWE-89, αποδίδοντας ένα μοναδικό αναγνωριστικό σε ένα συγκεκριμένο περιστατικό. Αυτή η τριεπίπεδη σχέση - το OWASP ως πλαίσιο κατηγοριών κινδύνου, το CWE ως τεχνική αδυναμία, το CVE ως συγκεκριμένο περιστατικό - αποτελεί τον σκελετό επάνω στον οποίο στηρίζεται η βιβλιογραφία ανίχνευσης των ευπαθειών, συμπεριλαμβανομένης της παρούσας εργασίας.

3.2 Ανάλυση κύριων ευπαθειών διαδικτυακών εφαρμογών

3.2.1 SQL Injection (SQLi)

Η SQL Injection αποτελεί μία από τις σημαντικότερες και ταυτόχρονα πιο διαδεδομένες κατηγορίες ευπαθειών διαδικτυακών εφαρμογών. Εντάσσεται στην κατηγορία A05:2025 - Injection του OWASP Top 10, η οποία ομαδοποιεί όλες τις ευπάθειες που προκύπτουν από την απουσία διαχωρισμού μεταξύ δεδομένων και εκτελέσιμου κώδικα [31]. Όταν μια εφαρμογή αποτυγχάνει να επικυρώσει κατάλληλα τα δεδομένα που παρέχει ο χρήστης, προκύπτει η ευπάθεια, η οποία επιτρέπει την εισαγωγή κακόβουλων αποσπασμάτων SQL μέσω πεδίων φορμών, παραμέτρων URL ή κεφαλίδων HTTP. Εφόσον η εφαρμογή ενσωματώνει αυτά τα δεδομένα απευθείας σε δυναμικά δομημένα ερωτήματα, η βάση δεδομένων τα εκτελεί ως έγκυρες εντολές SQL, επιτρέποντας στον εισβολέα να αλλοιώσει την προβλεπόμενη λογική του ερωτήματος. Συνεπώς, η δυναμική κατασκευή SQL ερωτημάτων από δεδομένα χρήστη, χωρίς διαχωρισμό μεταξύ κώδικα και δεδομένων, αποτελεί τον πυρήνα του προβλήματος [32].

Το συνηθέστερο παράδειγμα απεικονίζει το πρόβλημα στην πλέον απλή του μορφή. Ας θεωρήσουμε το ακόλουθο ερώτημα SQL που κατασκευάζεται δυναμικά από την εφαρμογή:

*SELECT * FROM Employee WHERE Eid='CS9723';*

Εάν ο χρήστης εισάγει ως τιμή *CS9723 OR '1=1'*, το ερώτημα μετατρέπεται σε:

*SELECT * FROM Employee WHERE Eid='CS9723' OR '1=1';*

Κατά την εκτέλεση του ερωτήματος, η συνθήκη '1=1' αξιολογείται πάντοτε ως αληθής, με αποτέλεσμα το σύστημα διαχείρισης βάσεων δεδομένων (DBMS) να ερμηνεύει το ερώτημα ως *SELECT * FROM TABLE Employee*, επιστρέφοντας το σύνολο των εγγραφών του πίνακα αντί για τη συγκεκριμένη εγγραφή.

Οι επιθέσεις SQL Injection ομαδοποιούνται σε διάφορες κατηγορίες με βάση τον μηχανισμό της εκτέλεσης και τον τρόπο ανάκτησης των δεδομένων [33], [34]. Η περισσότερο συνηθισμένη και απλή μορφή είναι οι tautology-based επιθέσεις, κατά τις οποίες ο εισβολέας εισάγει συνθήκες που αξιολογούνται πάντοτε ως αληθείς. Ένα πολύ σημαντικό παράδειγμα της κατηγορίας αυτής είναι η έκφραση OR '1'='1', η οποία χρησιμοποιείται για την παράκαμψη των μηχανισμών αυθεντικοποίησης ή για την ανάκτηση του συνόλου των εγγραφών ενός πίνακα [35]. Η επόμενη κατηγορία είναι οι union-based επιθέσεις, οι οποίες προϋποθέτουν ότι ο εισβολέας γνωρίζει τη δομή του σχήματος και τον αριθμό των στηλών του πίνακα της βάσης δεδομένων. Συγκεκριμένα, αυτός που επιτίθεται αξιοποιεί τον τελεστή UNION για να συγχωνεύσει το αποτέλεσμα ενός κακόβουλου ερωτήματος με το αποτέλεσμα του νόμιμου, εξάγοντας δεδομένα από άλλους πίνακες της βάσης [34].

Όταν η εφαρμογή δεν επιστρέφει άμεσα δεδομένα ή μηνύματα σφάλματος, ο εισβολέας έχει τη δυνατότητα να χρησιμοποιήσει τεχνικές blind SQL injection, κατά τις οποίες η εξαγωγή των πληροφοριών πραγματοποιείται έμμεσα μέσω παρατήρησης της συμπεριφοράς της εφαρμογής. Στην κατηγορία αυτή διακρίνονται κυρίως δύο παραλλαγές. Στη boolean-based blind SQL injection, ο εισβολέας εισάγει λογικές συνθήκες και παρατηρεί διαφοροποιήσεις στην απόκριση της εφαρμογής ανάλογα με το αν η συνθήκη αξιολογείται ως αληθής ή ψευδής. Αντίστοιχα, στη time-based blind SQL injection χρησιμοποιούνται συναρτήσεις καθυστέρησης, όπως η SLEEP(), ώστε ο χρόνος απόκρισης του διακομιστή να αποκαλύπτει αν μια συνθήκη ισχύει ή όχι [36].

Μια ακόμη κατηγορία είναι οι piggy-backed queries. Σε αυτές ο επιτιθέμενος προσαρτά επιπλέον SQL εντολές μετά το αρχικό ερώτημα μέσω του χαρακτήρα τερματισμού (π.χ. *SELECT * FROM users WHERE id='1'; DELETE FROM users*), και έτσι επιτρέπει την εκτέλεση πολλαπλών ενεργειών στο ίδιο αίτημα. Αυτή η τεχνική είναι δυνατόν να οδηγήσει σε μη εξουσιοδοτημένες λειτουργίες, όπως η τροποποίηση ή η διαγραφή δεδομένων [33]. Αντίστοιχα, οι επιθέσεις μέσω stored procedures εκμεταλλεύονται τις αποθηκευμένες διαδικασίες της βάσης δεδομένων ώστε να εκτελούνται λειτουργίες πέρα από την προβλεπόμενη συμπεριφορά της εφαρμογής. Μάλιστα, σε ορισμένες περιπτώσεις μπορεί να επηρεαστεί ακόμη και το υποκείμενο λειτουργικό σύστημα [37]. Τέλος, οι second-order SQL injection επιθέσεις διαφοροποιούνται, επειδή το κακόβουλο φορτίο δεν εκτελείται άμεσα, αλλά αποθηκεύεται αρχικά στη βάση δεδομένων ως φαινομενικά έγκυρο δεδομένο. Το φορτίο ενεργοποιείται σε μεταγενέστερο στάδιο, όταν το αποθηκευμένο δεδομένο χρησιμοποιηθεί σε διαφορετικό ερώτημα της εφαρμογής, γεγονός που καθιστά τον εντοπισμό της επίθεσης ιδιαίτερα δύσκολο [34].

3.2.2 Cross-Site Scripting (XSS)

Το Cross-Site Scripting, γνωστό με το ακρωνύμιο XSS, αποτελεί μία ακόμη σημαντική ευπάθεια, η οποία εντάσσεται στην ίδια κατηγορία OWASP με την SQLi. Η ευπάθεια προκύπτει όταν μια διαδικτυακή εφαρμογή ενσωματώνει δεδομένα χρήστη σε μια ιστοσελίδα χωρίς την απαραίτητη επικύρωση και απολύμανση. Υπό τις συνθήκες αυτές, ο κακόβουλος κώδικας που υποβάλλει ο επιτιθέμενος δεν αντιμετωπίζεται από την εφαρμογή ως δεδομένο αλλά ως εκτελέσιμο περιεχόμενο, με αποτέλεσμα να εκτελείται όταν ο χρήστης επισκέπτεται τη σελίδα. Σε αντίθεση με το SQL Injection, το οποίο στοχεύει το επίπεδο της βάσης δεδομένων, το XSS εκμεταλλεύεται την εμπιστοσύνη που έχει ο χρήστης στην εφαρμογή. Ως αποτέλεσμα, ο κακόβουλος κώδικας εκτελείται μέσα στο πλαίσιο εμπιστοσύνης της νόμιμης εφαρμογής, επιτρέποντας στον επιτιθέμενο να αποκτήσει πρόσβαση σε ευαίσθητες πληροφορίες του χρήστη [38].

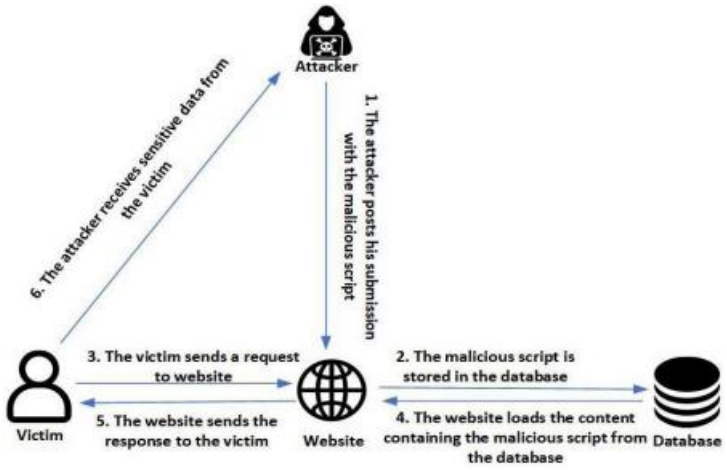
Το παράδειγμα που ακολουθεί παρουσιάζει τον πυρήνα της ευπάθειας. Ένας επιτιθέμενος μπορεί να υποβάλει, μέσω φόρμας εισαγωγής, το εξής script:

<script> window.location = 'http://attacker.com/?cookie='+ document.cookie </script>

Εάν η εφαρμογή αποθηκεύσει αυτή την τιμή αυτούσια στη βάση δεδομένων και στη συνέχεια την ενσωματώσει χωρίς απολύμανση στην HTML απόκριση, κάθε επισκέπτης που φορτώνει τη σελίδα θα ανακατευθυνθεί αυτόματα στον διακομιστή του επιτιθέμενου, μεταφέροντας το session cookie του ως παράμετρο URL [39].

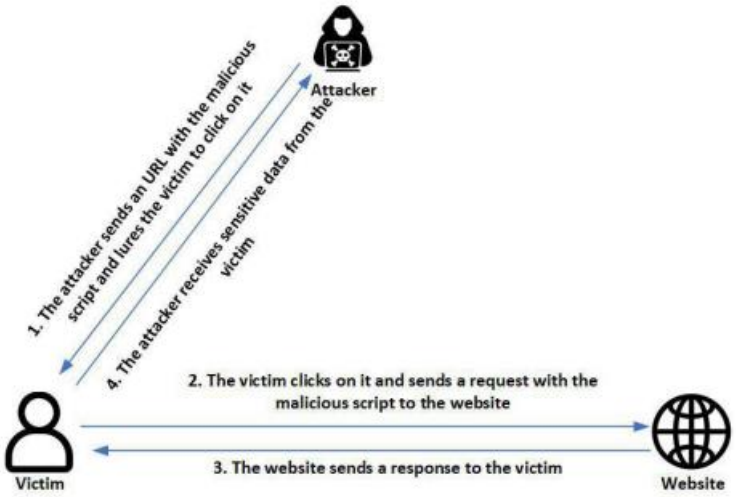
Με βάση τον μηχανισμό και τη διαδρομή της επίθεσης, το XSS ταξινομείται σε τρεις βασικές κατηγορίες [40].

Αρχικά, στο Stored ή Persistent XSS ο επιτιθέμενος υποβάλλει τον κακόβουλο κώδικα (script) μέσω φόρμας εισαγωγής (π.χ. πεδίου σχολίου, προφίλ χρήστη ή φόρουμ) και η εφαρμογή το αποθηκεύει στη βάση δεδομένων χωρίς απολύμανση. Κάθε φορά που οποιοσδήποτε επισκέπτης φορτώνει τη σχετική σελίδα, το αποθηκευμένο script επιστρέφεται μαζί με το περιεχόμενο και εκτελείται αυτόματα στον φυλλομετρητή (browser) του (Σχήμα 3.1). Η ιδιότητα αυτή, δεν απαιτεί καμία ενέργεια εκ μέρους του θύματος πέρα από την απλή πλοήγηση στη σελίδα, γεγονός που την καθιστά ιδιαίτερα επικίνδυνη [41].



Εικόνα 3.1: *Η διαδικασία ενός stored XSS [42]*

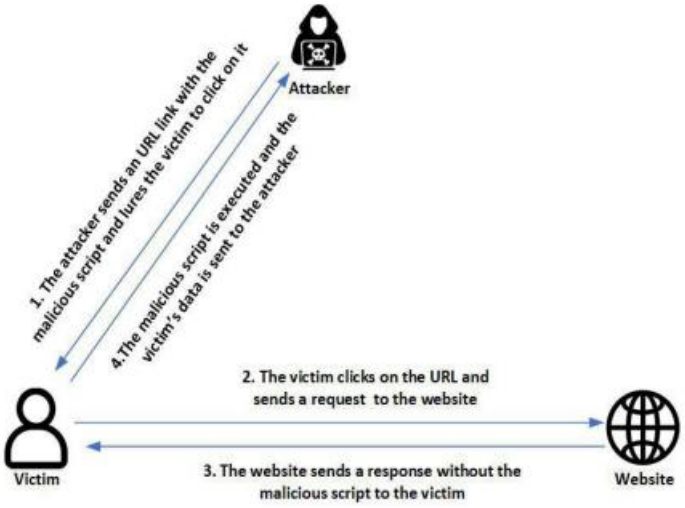
Στη συνέχεια, το Reflected ή Non-Persistent XSS διαφέρει ως προς τον τρόπο μεταφοράς του κακόβουλου φορτίου. Στην περίπτωση αυτή, ο επιτιθέμενος ενσωματώνει το κακόβουλο script σε μια διεύθυνση URL και αποστέλλει τον σύνδεσμο στο θύμα μέσω email ή κάποιου άλλου καναλιού επικοινωνίας. Όταν το θύμα ανοίξει τον σύνδεσμο, η εφαρμογή εμφανίζει στη σελίδα δεδομένα που προέρχονται από τη διεύθυνση URL χωρίς κατάλληλο έλεγχο, επιτρέποντας την εκτέλεση κακόβουλου κώδικα στον browser του χρήστη (Σχήμα 3.2). Ωστόσο, σε αντίθεση με τη Stored XSS, ο κακόβουλος κώδικας δεν αποθηκεύεται στον διακομιστή και η επιτυχία της επίθεσης εξαρτάται από το αν το θύμα ακολουθήσει ή όχι τον κακόβουλο σύνδεσμο [43].



Εικόνα 3.2: *Η διαδικασία ενός reflected XSS [42]*

Τέλος, το DOM-Based XSS αποτελεί κατηγορία ευπάθειας από την πλευρά του χρήστη και διαφοροποιείται ουσιαστικά από τις δύο προηγούμενες μορφές XSS. Στη περίπτωση αυτή, ο διακομιστής δεν συμμετέχει άμεσα στην εκτέλεση του κακόβουλου κώδικα, επειδή η ευπάθεια προκύπτει από κώδικα της ίδιας της ιστοσελίδας, η οποία διαχειρίζεται με μη ασφαλή τρόπο

τα δεδομένα του Document Object Model (DOM). Το DOM, αποτελεί τη δομή μέσω της οποίας ο browser αναπαριστά και διαχειρίζεται τα στοιχεία μιας ιστοσελίδας. Ως εκ τούτου, ο κακόβουλος κώδικας εκτελείται αποκλειστικά στον browser του χρήστη και δεν περιλαμβάνεται ποτέ στην απόκριση του διακομιστή, με συνέπεια να καθίσταται εξαιρετικά δυσχερής ο εντοπισμός της επίθεσης (Σχήμα 3.3) [44].



Εικόνα 3.3: *Η διαδικασία ενός DOM-Based XSS [42]*

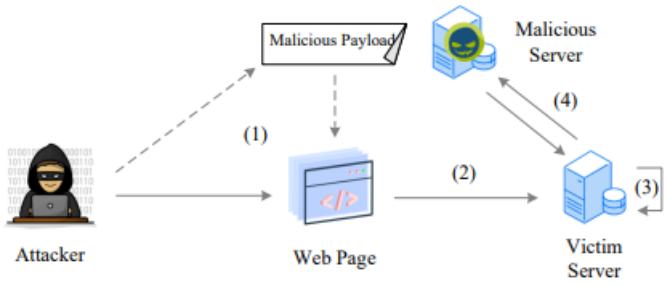
Ως προς τις συνέπειες της εκμετάλλευσης, το XSS επιτρέπει ένα ευρύ φάσμα επιθέσεων. Η κλοπή των cookies συνεδρίας και η εκ των υστέρων πλαστοπροσωπία χρήστη αποτελεί την πιο συνηθισμένη εκδοχή, ωστόσο οι δυνατότητες του επιτιθέμενου εκτείνονται πολύ πιο πέρα από αυτήν. Μέσω της χειραγωγήσεως του περιεχομένου της σελίδας, είναι δυνατόν να δημιουργηθούν ψεύτικες φόρμες σύνδεσης για την υποκλοπή διαπιστευτηρίων. Παράλληλα, με την αξιοποίηση των δυνατοτήτων του browser, μπορεί να καταγραφεί τί πληκτρολογεί το θύμα [45]. Ιδιαίτερα επικίνδυνη είναι η δυνατότητα αυτόματης εξάπλωσης. Συγκεκριμένα, ένα κακόβουλο script που έχει αποθηκευτεί σε ένα κοινωνικό δίκτυο, μπορεί αυτόματα να αντιγράψει τον εαυτό του στα προφίλ των επισκεπτών, δημιουργώντας XSS worm με εκθετική εξάπλωση. Χαρακτηριστικό παράδειγμα είναι το Samy worm στο MySpace το 2005, το οποίο μόλυνε περισσότερους από ένα εκατομμύριο λογαριασμούς σε χρονικό διάστημα μικρότερο από 24 ώρες [46].

3.2.3 Remote Code Execution (RCE)

Το Remote Code Execution (RCE) συνιστά ένα είδος ευπάθειας, το οποίο εντάσσεται όπως και οι δύο προηγούμενες, στην κατηγορία A05:2025 – Injection του OWASP Top 10. Ο επιτιθέμενος εδώ, πετυχαίνει να εισάγει και να εκτελέσει κακόβουλο κώδικα στον διακομιστή μιας εφαρμογής. Η επίθεση αυτή εκμεταλλεύεται αδυναμίες στον έλεγχο των δεδομένων εισόδου, με συνέπεια τα δεδομένα που στέλνει ο χρήστης να μετατρέπονται και να εκτελούνται ως εντολές ή scripts. Σύμφωνα με το OWASP, το RCE θεωρείται μία από τις σημαντικότερες απειλές για την ασφάλεια web εφαρμογών, ιδιαίτερα σε εφαρμογές PHP. Οι επιθέσεις RCE

είναι συνήθως πολύπλοκες, επειδή συχνά απαιτούν πολλαπλά αιτήματα προς τον server και κατάλληλη διαχείριση των συνεδριών (sessions). Παράλληλα, οι κακόβουλες είσοδοι είναι ειδικά διαμορφωμένες ώστε να παρακάμπτουν τους ελέγχους ασφαλείας της εφαρμογής [47].

Μία τυπική επίθεση RCE σε διαδικτυακή εφαρμογή ακολουθεί τέσσερα βήματα, όπως αυτά απεικονίζονται στο Σχήμα 3.4. Πρώτα, ο επιτιθέμενος εντοπίζει σημεία εισόδου στην εφαρμογή που δέχονται δεδομένα από τον χρήστη. Κατόπιν, δημιουργεί ένα κακόβουλο payload, δηλαδή ένα τμήμα κώδικα ή εντολής, και το υποβάλλει στην εφαρμογή. Η εφαρμογή το προωθεί στον διακομιστή, ο οποίος το εκτελεί ως νόμιμη εντολή. Εν τέλει, ο διακομιστής επικοινωνεί με τον κακόβουλο server του επιτιθέμενου, αποστέλλοντας δεδομένα ή δεχόμενος περαιτέρω εντολές [48].



Εικόνα 3.4*: Η διαδικασία μιας επίθεσης RCE σε web εφαρμογή [48].*

Με βάση τον τρόπο εκμετάλλευσης, το RCE διακρίνεται σε δύο κύριες κατηγορίες. Η πρώτη είναι το Web Based RCE, κατά το οποίο ο επιτιθέμενος εκμεταλλεύεται μια ευπάθεια της ίδιας της διαδικτυακής εφαρμογής προκειμένου να εκτελέσει εντολές στον διακομιστή. Αυτό επιτυγχάνεται με αιτήματα που αποστέλλονται στην εφαρμογή, είτε μέσω παραμέτρων URL είτε μέσω δεδομένων φόρμας, τα οποία η εφαρμογή επεξεργάζεται χωρίς τον απαιτούμενο έλεγχο. Χαρακτηριστικό παράδειγμα αποτελεί μια εφαρμογή που εκτελεί εντολή ping με διεύθυνση IP που παρέχει ο χρήστης: η είσοδος 8.8.8.8; cat /etc/passwd θα προκαλούσε την εκτέλεση δύο εντολών ταυτόχρονα, της νόμιμης εντολής ping και της εντολής ανάγνωσης του αρχείου κωδικών του συστήματος. Η δεύτερη κατηγορία είναι το System Based RCE, κατά το οποίο ο επιτιθέμενος εκμεταλλεύεται ευπάθεια σε υπηρεσίες ή μηχανισμούς του ίδιου του συστήματος, με στόχο την απόκτηση πρόσβασης και την εκτέλεση κακόβουλων εντολών [49].

Ανεξάρτητα από την κατηγορία στην οποία ανήκουν, οι συνέπειες μιας επιτυχημένης επίθεσης RCE είναι ιδιαίτερα σοβαρές. Σε αντίθεση με την SQL Injection και το XSS, όπου ο επιτιθέμενος στοχεύει στη βάση δεδομένων ή στον browser του θύματος αντίστοιχα, στο RCE τον στόχο αποτελεί ο ίδιος ο διακομιστής. Αυτό σημαίνει ότι αυτός που επιτίθεται μπορεί να υποκλέψει ευαίσθητα δεδομένα, να τροποποιήσει ή να διαγράψει αρχεία, να εγκαταστήσει κακόβουλο λογισμικό και εν τέλει να αποκτήσει τον πλήρη έλεγχο του συστήματος [50].

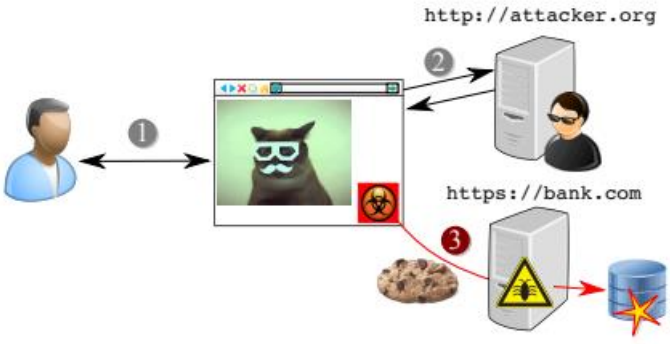
3.2.4 Cross-Site Request Forgery (CSRF)

Η ευπάθεια Cross-Site Request Forgery (CSRF) εντάσσεται στην κατηγορία A01:2025 -

Broken Access Control του OWASP Top 10. Σε αντίθεση με τις προηγούμενες ευπάθειες, το CSRF δεν εκμεταλλεύεται κάποιο ελάττωμα στον τρόπο που η εφαρμογή επεξεργάζεται τα δεδομένα εισόδου, αλλά αξιοποιεί έναν θεμελιώδη μηχανισμό του ίδιου του πρωτοκόλλου HTTP. Ειδικότερα, μπορεί να αξιοποιήσει την αυτόματη αποστολή cookies συνεδρίας σε κάθε αίτημα προς έναν διακομιστή, ανεξάρτητα από τον τομέα προέλευσης (domain) του αιτήματος [51].

Ο μηχανισμός λειτουργίας του CSRF βασίζεται στην εμπιστοσύνη που καταδεικνύει ο διακομιστής στον browser του χρήστη. Όταν ένας χρήστης αυθεντικοποιηθεί σε μια εφαρμογή, ο browser αποθηκεύει ένα cookie συνεδρίας, το οποίο επισυνάπτεται αυτόματα σε κάθε επόμενο αίτημα προς τον ίδιο διακομιστή. Ο επιτιθέμενος εκμεταλλεύεται αυτή τη συμπεριφορά δημιουργώντας μια κακόβουλη ιστοσελίδα, η οποία εμπεριέχει κώδικα αποστολής αιτήματος προς την εφαρμογή-στόχο. Όταν το θύμα επισκεφθεί τη σελίδα, το αίτημα αποστέλλεται αυτόματα δίχως τη γνώση ή τη συγκατάθεσή του, ενώ το cookie συνεδρίας επισυνάπτεται αυτόματα από τον browser. Το αποτέλεσμα είναι ότι ο διακομιστής θεωρεί το αίτημα έγκυρο και εκτελεί την ενέργεια που επιδιώκει ο επιτιθέμενος [52].

Το Σχήμα 3.5 απεικονίζει τη διαδικασία μιας επίθεσης authenticated CSRF, κατά την οποία το θύμα είναι ήδη αυθεντικοποιημένο στην εφαρμογή-στόχο. Αρχικά, ο χρήστης επισκέπτεται την κακόβουλη ιστοσελίδα που ελέγχεται από τον επιτιθέμενο (βήμα 1). Στη συνέχεια, η ιστοσελίδα αποστέλλει κακόβουλο περιεχόμενο στον browser του θύματος (βήμα 2). Το περιεχόμενο αυτό προκαλεί την αυτόματη αποστολή αιτήματος προς την εφαρμογή-στόχο μαζί με το cookie συνεδρίας του χρήστη. Εν τέλει, ο διακομιστής θεωρεί το αίτημα έγκυρο και εκτελεί την ενέργεια που επιδιώκει ο επιτιθέμενος (βήμα 3) [53].



Εικόνα 3.5: *Επίθεση authenticated CSRF [53].*

Οι συνέπειες μιας επιτυχημένης επίθεσης CSRF εξαρτώνται άμεσα από τα δικαιώματα του χρήστη που στοχοποιείται. Εάν ο στόχος είναι ένας απλός χρήστης, ο επιτιθέμενος μπορεί να πραγματοποιήσει ανεπιθύμητες ενέργειες στο όνομά του, όπως αλλαγή κωδικού πρόσβασης, μεταφορά χρημάτων ή διαγραφή δεδομένων. Εάν όμως ο στόχος είναι λογαριασμός διαχειριστή, ο επιτιθέμενος αποκτά τη δυνατότητα να θέσει σε κίνδυνο ολόκληρη την εφαρμογή. Παρά τη μεγάλη επικινδυνότητά της, η ευπάθεια CSRF εξακολουθεί να εμφανίζεται σε πολλές διαδικτυακές εφαρμογές λόγω ανεπαρκών μηχανισμών προστασίας [54].

3.3 Βάσεις δεδομένων ευπαθειών

Το εννοιολογικό πλαίσιο που παρουσιάστηκε στην ενότητα 3.1 - OWASP, CWE, CVE - συνιστά τη θεωρητική υποδομή για την κατανόηση και την ταξινόμηση των ευπαθειών. Ωστόσο, για να είναι εφικτή η πρακτική αξιολόγηση ενός εργαλείου ανίχνευσης, απαιτείται και μια αξιόπιστη πηγή τεκμηρίωσης έτσι ώστε να συνδεθούν αυτές οι κατηγορίες με συγκεκριμένες και πραγματικές ευπάθειες σε γνωστό λογισμικό. Ο ρόλος αυτός ανήκει στις βάσεις δεδομένων των ευπαθειών, που αποτελούν εξειδικευμένα συστήματα καταγραφής, τα οποία συγκεντρώνουν, επαληθεύουν και δημοσιεύουν πληροφορίες για γνωστά κενά ασφαλείας.

Μια διαφορετικού είδους προσέγγιση ακολουθεί το Exploit-DB, μια ανοιχτή βάση δεδομένων που συντηρείται από κοινοτικές συνεισφορές. Ειδικότερα, οποιοσδήποτε εντοπίσει μια ευπάθεια μπορεί να δημοσιεύσει το εύρημά του, χωρίς υποχρέωση αναφοράς στον κατασκευαστή ή στις αρμόδιες αρχές έκδοσης CVE. Κάθε καταχώριση συνοδεύεται από έτοιμο κώδικα απόδειξης της εκμετάλλευσης (exploits), με στόχο να διευκολύνει την κατανόηση της ευπάθειας και την ανάπτυξη διορθώσεων. Ωστόσο, η ίδια αυτή ιδιότητα ενέχει κινδύνους. Έρευνες αναδεικνύουν ότι το 73,5% των exploits που δημοσιεύονται στο Exploit-DB, ανακοινώνονται τουλάχιστον μια μέρα πριν από την έκδοση του αντίστοιχου CVE. Παράλληλα, περίπου το 28% δεν αποκτά ποτέ επίσημη CVE καταχώριση, γεγονός που αποδυναμώνει τις προσπάθειες έγκαιρης αντιμετώπισης [55].

Για την πραγματοποίηση του πειράματος της εργασίας, ως κύρια πηγή τεκμηρίωσης επιλέχτηκε η Wordfence Intelligence, η οποία αποτελεί μια εξειδικευμένη βάση δεδομένων ευπαθειών, που εστιάζει αποκλειστικά στο οικοσύστημα του WordPress [56]. Η βάση δεδομένων Wordfence Intelligence συντηρείται από ομάδα ερευνητών ασφάλειας και περιλαμβάνει ένα μεγάλο αριθμό καταχωρίσεων για ευπάθειες σε plugins, θέματα εμφάνισης και τον πυρήνα του WordPress. Για κάθε ευπάθεια παρέχει τον τύπο της (π.χ. SQL Injection, XSS, CSRF), την επηρεαζόμενη έκδοση, την έκδοση στην οποία εφαρμόστηκε η διόρθωση και το αντίστοιχο CVE ID. Η επιλογή της ως πηγή για το πείραμα στηρίχτηκε στον συνδυασμό τριών χαρακτηριστικών: την εξειδίκευσή της στο WordPress, τη δομημένη και επαληθευμένη μορφή των καταχωρίσεών της, καθώς και την ελεύθερη πρόσβαση τόσο στο διαδικτυακό περιβάλλον όσο και μέσω API [57].

Κεφάλαιο 4ο: Τεχνητή νοημοσύνη και LLMs στον κώδικα

4.1 Εισαγωγή

Η ταχύτατη εξέλιξη της Τεχνητής Νοημοσύνης και ιδίως των Μεγάλων Γλωσσικών Μοντέλων (Large Language Models - LLMs) έχει αλλάξει σημαντικά τον τρόπο με τον οποίο διερευνώνται τα προβλήματα της ανάλυσης, της παραγωγής και της κατανόησης κώδικα. Τα παραδοσιακά εργαλεία της στατικής ανάλυσης βασίζονται κυρίως σε ρητούς κανόνες, σε αναπαραστάσεις ροής ελέγχου και σε προκαθορισμένα μοτίβα ανίχνευσης. Αντίθετα, τα LLMs διαχειρίζονται τον πηγαίο κώδικα ως μορφή δομημένης γλώσσας, αξιοποιώντας τη στατιστική μάθηση πάνω σε πολύ μεγάλα σύνολα δεδομένων. Η ιδιότητα αυτή τα καθιστά ιδιαίτερα ενδιαφέροντα για εφαρμογές που απαιτούν όχι μόνο συντακτική, αλλά και εν μέρει σημασιολογική κατανόηση του κώδικα.

Η σημασία των LLMs στο πεδίο της ασφάλειας του λογισμικού είναι αυξανόμενη, επειδή τα μοντέλα αυτά μπορούν να αξιοποιηθούν για τον εντοπισμό πιθανών ευπαθειών, για τη δημιουργία επεξηγήσεων σχετικά με επικίνδυνα τμήματα κώδικα, για την παραγωγή διορθωτικών προτάσεων και για την υποστήριξη ημιαυτόνομων ή πλήρως αυτόνομων διαδικασιών ελέγχου. Παρά τα πλεονεκτήματα, που είναι ομολογουμένως σημαντικά, η αξιοποίηση των LLMs συνοδεύεται από βασικούς περιορισμούς. Δείγματος χάριν, μπορούμε να αναφέρουμε την εξάρτηση από το tokenization, τους περιορισμούς του πλαισίου συμφραζομένων (context window), την αβεβαιότητα στην ακολουθία λογικών βημάτων και το φαινόμενο των ψευδαισθήσεων (hallucinations). Για τον λόγο αυτό, η μελέτη της αρχιτεκτονικής και της λειτουργίας τους συνιστά απαραίτητη προϋπόθεση για την κατανόηση της συμπεριφοράς τους σε σενάρια ανάλυσης της ασφάλειας.

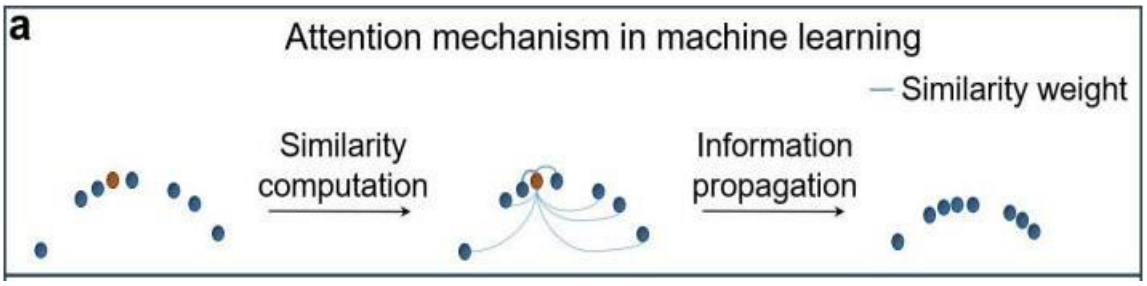
Στο παρόν κεφάλαιο εκτίθενται οι βασικές τεχνικές αρχές, οι οποίες διέπουν τα σύγχρονα LLMs. Ιδιαίτερη έμφαση δίνεται στα στοιχεία εκείνα που επηρεάζουν άμεσα την ικανότητά τους να αναλύουν πηγαίο κώδικα και να εντοπίζουν ευπάθειες σε εφαρμογές, σε περιβάλλον white-box.

4.2 Αρχιτεκτονική Transformer: Μηχανισμός προσοχής και κατανόηση κώδικα

Η θεμελιώδης αρχιτεκτονική, πάνω στην οποία στηρίζονται τα πιο πολλά σύγχρονα Μεγάλα Γλωσσικά Μοντέλα, είναι ο Transformer. Η αρχιτεκτονική αυτή προτάθηκε ως εναλλακτική των επαναληπτικών νευρωνικών δικτύων και επέφερε εξαιρετικά σημαντική πρόοδο στην επεξεργασία της φυσικής γλώσσας, κυρίως επειδή επιτρέπει την παράλληλη επεξεργασία ολόκληρων ακολουθιών εισόδου και την αποτύπωση μακρινών εξαρτήσεων ανάμεσα στα στοιχεία της ακολουθίας [58]. Στο πλαίσιο του πηγαίου κώδικα, το χαρακτηριστικό αυτό γνώρισμα είναι πολύ σημαντικό, καθώς η σημασία μιας εντολής συχνά εξαρτάται από μεταβλητές, από συναρτήσεις ή από ελέγχους που παρουσιάζονται σε διαφορετικά σημεία του αρχείου ή ακόμη και σε διαφορετικά αρχεία του ίδιου έργου.

Ο βασικός μηχανισμός που αυξάνει την απόδοση του Transformer είναι ο μηχανισμός

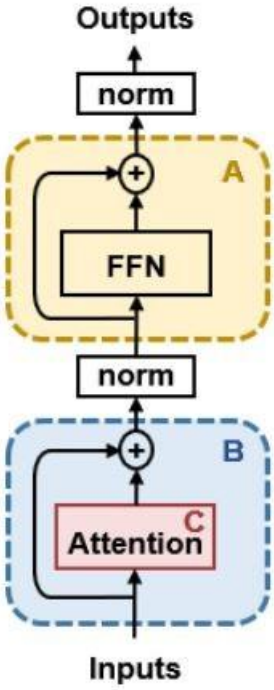
προσοχής (attention mechanism). Κατά την επεξεργασία μιας ακολουθίας tokens, το μοντέλο δεν αντιμετωπίζει όλα τα tokens ισότιμα. Αντιθέτως, για κάθε token υπολογίζει ένα βαθμό συνάφειας (attention score) ως προς κάθε άλλο token της ακολουθίας, και κατόπιν χρησιμοποιεί αυτούς τους βαθμούς για να διαμορφώσει μια πληρέστερη εικόνα της σημασίας του [59]. Ως αποτέλεσμα έρχεται ότι η τελική «κατανόηση» κάθε λέξης ή εντολής δεν είναι στατική. Εξαρτάται δε από τα συμφραζόμενα: το ίδιο token ερμηνεύεται κάθε φορά ανάλογα με το τι το περιβάλλει στην ακολουθία. Το Σχήμα 4.1 απεικονίζει διαισθητικά την ανωτέρω διαδικασία: ένα token υπολογίζει βαθμούς ομοιότητας με τα υπόλοιπα (similarity computation) και αναπροσαρμόζει την αναπαράστασή του βάσει των περισσότερο συναφών από αυτά (information propagation).



Εικόνα 4.1: *Οπτική αναπαράσταση του μηχανισμού προσοχής. Κάθε token (κόκκινο) υπολογίζει βαθμούς ομοιότητας με τα υπόλοιπα tokens της ακολουθίας και αναπροσαρμόζει την αναπαράστασή του ανάλογα με τη σχετικότητά τους [60].*

Στο πλαίσιο της στατικής ανάλυσης ενός κώδικα, η ιδιότητα αυτή μπορεί να αποβεί εξαιρετικά επωφελής. Όταν η είσοδος ενός χρήστη εισέρχεται σε ένα σύστημα και καταλήγει, μέσα από αλλεπάλληλες συναρτήσεις, σε ένα ερώτημα βάσης δεδομένων χωρίς προηγούμενη απολύμανση, τότε το μοντέλο μπορεί να εντοπίσει τη σύνδεση αυτή ακόμη και αν τα δύο σημεία απέχουν πολύ μέσα στον κώδικα. Αυτή η δυνατότητα συνιστά και το θεμελιώδες πλεονέκτημα των LLMs συγκρικά με τα παραδοσιακά εργαλεία SAST, τα οποία ως βάση τους έχουν αυστηρά προκαθορισμένα μοτίβα και δεν μπορούν να παρακολουθήσουν τέτοιου είδους διαδρομές δεδομένων χωρίς ρητή μοντελοποίησή τους [61].

Εκτός από τον μηχανισμό προσοχής, η αρχιτεκτονική Transformer εμπεριέχει και επιπρόσθετα δομικά στοιχεία που συνεισφέρουν στην αποτελεσματικότητά της. Όπως φαίνεται στο Σχήμα 4.2, ένα πλήρες Transformer block αποτελείται από δύο κύρια επίπεδα: α) το block προσοχής (Attention), το οποίο αναλύει τις σχέσεις μεταξύ tokens, και β) ένα Feed-Forward Network (FFN), το οποίο επεξεργάζεται κάθε token ανεξάρτητα για να εμπλουτίσει την αναπαράστασή του. Τα δύο αυτά επίπεδα περιλαμβάνουν κανονικοποίηση (norm) και παρακαμπτήριες συνδέσεις, οι οποίες οδηγούν στους αθροιστές (residual connections), και οι οποίες διατηρούν την αρχική πληροφορία και τη συναθροίζουν με την επεξεργασμένη. Αυτός ο συνδυασμός εξασφαλίζει τη σταθερή ροή των δεδομένων και την αποδοτική εκπαίδευση του μοντέλου, διασφαλίζοντας ότι δεν θα υπάρχει απώλεια σημαντικών στοιχείων κατά τη μετάβαση από το ένα επίπεδο στο άλλο [62].



Εικόνα 4.2: *Δομή ενός Transformer block Αποτελείται από το block προσοχής (C), που αναλύει τις σχέσεις μεταξύ tokens, και ένα Feed-Forward Network (A), το οποίο επεξεργάζεται κάθε token ανεξάρτητα. Και τα δύο επίπεδα περιλαμβάνουν residual connections (+) και κανονικοποίηση (norm) [60].*

Η κατανόηση του κώδικα μέσω της αρχιτεκτονικής Transformer διαφέρει σημαντικά από την τυπική ανάλυση, η οποία εκτελείται από τα παραδοσιακά εργαλεία της στατικής ανάλυσης. Αντί το μοντέλο να ακολουθεί αυστηρούς κανόνες, εντοπίζει στατιστικές συσχετίσεις και επαναλαμβανόμενα σχήματα, τα οποία έχει αναγνωρίσει κατά την εκπαίδευσή του σε εκτεταμένα σύνολα δεδομένων. Επιπρόσθετα, αξιολογεί τη δομή του κώδικα, όπως π.χ. α) τα ονόματα μεταβλητών, β) τις συνηθισμένες προγραμματιστικές πρακτικές και γ) τα επαναλαμβανόμενα πρότυπα στις κλήσεις συναρτήσεων. Με τον τρόπο αυτό, ένα μοντέλο μπορεί να βγάλει ως συμπέρασμα ότι μια σειρά ενεργειών είναι επικίνδυνη, ακόμη και αν δεν ταιριάζει ακριβώς σε κάποιο προκαθορισμένο κανόνα. Αυτή η παρεχόμενη ευελιξία είναι και ο λόγος που τα LLMs συνιστούν μια ομολογουμένως πολλά υποσχόμενη προσέγγιση για την ενίσχυση της ασφάλειας ενός λογισμικού.

4.3 Tokenization και Context Windows: Οι τεχνικοί περιορισμοί των μοντέλων

Προτού ένα Μεγάλο Γλωσσικό Μοντέλο καταφέρει να επεξεργαστεί μία φυσική γλώσσα ή ένα πηγαίο κώδικα, απαιτείται η μετατροπή της εισόδου σε μια ακολουθία από μικρότερες μονάδες, γνωστές ως tokens. Η διαδικασία αυτή, που ονομάζεται tokenization, αποτελεί βασικό στάδιο της λειτουργίας των LLMs. Αυτό συμβαίνει επειδή το μοντέλο δεν αναγνωρίζει άμεσα

χαρακτήρες, λέξεις ή γραμμές κώδικα, αλλά αριθμητικές αναπαραστάσεις των tokens που έχουν παραχθεί [63]. Για την περίπτωση του φυσικού λόγου, τα tokens μπορεί να αντιστοιχούν σε λέξεις, υπολέξεις ή σημεία στίξης· για την περίπτωση όμως του πηγαίου κώδικα, η διαδικασία είναι πιο σύνθετη: Ο κώδικας περιλαμβάνει ονόματα μεταβλητών, τελεστές, ειδικούς χαρακτήρες και δομικά στοιχεία, τα οποία δεν ακολουθούν πάντα τους κανόνες της φυσικής γλώσσας [64].

Η σημασία του tokenization στην ανάλυση του κώδικα είναι ιδιαίτερα μεγάλη, επειδή επηρεάζει άμεσα τον τρόπο με τον οποίο το μοντέλο τμηματοποιεί και ερμηνεύει τη δομή ενός προγράμματος. Ως παράδειγμα παραθέτουμε το εξής: αν ένα όνομα μεταβλητής ή μια εντολή σπάσει σε πολλά και ασύνδετα κομμάτια, το μοντέλο μπορεί να δυσκολευτεί να αντιληφθεί τη σημασία τους [64]. Επιπλέον, σύνθετα τμήματα κώδικα, όπως μεγάλα ερωτήματα SQL ή εντολές σε PHP, αρκετά συχνά μετατρέπονται σε έναν πολύ μεγάλο αριθμό από tokens. Αυτό συνεπάγεται να καταναλώνεται ταχύτερα η διαθέσιμη μνήμη του μοντέλου, περιορίζοντας τη δυνατότητα να μπορεί να δει ταυτόχρονα μεγάλο μέρος του υπόλοιπου κώδικα. Στην ανάλυση της ασφάλειας αυτό είναι εξαιρετικά κρίσιμο, διότι οι ευπάθειες συχνά κρύβονται πίσω από μικρές λεπτομέρειες, όπως ένας χαρακτήρας που λείπει ή μια λανθασμένη σειρά εντολών. Αν λοιπόν ο τεμαχισμός δεν γίνει σωστά, υπάρχει ο κίνδυνος το μοντέλο είτε να παραβλέψει το πρόβλημα είτε να μην μπορέσει να συνδέσει ορθά τα δεδομένα που προκαλούν την ευπάθεια [65], [66].

Ένας δεύτερος βασικός περιορισμός των LLMs είναι το μέγεθος του παραθύρου συμφραζομένων (context window), δηλαδή το μέγιστο πλήθος tokens, τα οποία μπορούν να ληφθούν υπόψη σε μία μόνο διεργασία ανάλυσης ή παραγωγής. Το context window λειτουργεί ως ένα προσωρινό πεδίο συμφραζομένων μέσα στο οποίο το μοντέλο επιχειρεί να εντοπίσει σχέσεις, εξαρτήσεις και λογικές ακολουθίες. Κάθε φορά που ο πηγαίος κώδικας μιας εφαρμογής ή ενός plugin υπερβαίνει αυτό το όριο, το μοντέλο δεν έχει τη δυνατότητα να επεξεργαστεί ταυτόχρονα ολόκληρο το έργο. Εάν όμως πραγματοποιηθεί ο τεμαχισμός του κώδικα σε μικρότερα αποσπάσματα ή διαδοχική πλοήγηση στα αρχεία, μπορεί να οδηγήσει σε απώλεια κρίσιμου πλαισίου [67].

Πρόκειται για έναν περιορισμό που επηρεάζει ουσιαστικά την ικανότητα των LLMs να εντοπίζουν ευπάθειες, οι οποίες δεν βρίσκονται σε ένα σημείο μόνο, αλλά απλώνονται σε μεγάλο μέρος του κώδικα. Πρακτικά, σε πολλές περιπτώσεις, ένα κενό ασφαλείας δεν προκύπτει από μία μόνο γραμμή κώδικα, αλλά από ολόκληρη τη διαδρομή που ακολουθούν τα δεδομένα. Ειδικότερα, το κενό προκύπτει, όταν ο χρήστης εισάγει δεδομένα, μέσα από τις μεταβολές που υφίστανται, μέχρι την οριστική χρήση τους στο σύστημα. Αν αυτά τα στάδια βρίσκονται σε διαφορετικές συναρτήσεις ή αρχεία, το μοντέλο θα πρέπει να μπορεί να παρακολουθεί ολόκληρη τη διαδρομή αυτή ταυτόχρονα για να εξάγει ορθό συμπέρασμα. Όμως, όταν το διαθέσιμο παράθυρο συμφραζομένων δεν επαρκεί, το μοντέλο μπορεί να χάσει τη σύνδεση μεταξύ των βημάτων [68]. Αυτό έχει ως αποτέλεσμα είτε να μην αντιληφθεί μια πραγματική ευπάθεια, είτε να καταλήξει σε λάθος συμπέρασμα, θεωρώντας ότι υπάρχει πρόβλημα, εκεί που στην πραγματικότητα δεν υπάρχει.

Ιδιαίτερη σημασία έχει ακόμη το γεγονός ότι ο πηγαίος κώδικας συνήθως περιλαμβάνει πιο πυκνή πληροφορία από το απλό κείμενο. Έτσι, μια μικρή αύξηση στο μέγεθος ενός αρχείου

μπορεί να σημαίνει πολύ περισσότερα tokens, ειδικά όταν υπάρχουν σύνθετες δομές, πολλές βιβλιοθήκες ή επαναλαμβανόμενα τμήματα κώδικα [69]. Για τον λόγο αυτό, ακόμη και μοντέλα με μεγάλα παράθυρα συμφραζομένων μπορεί να δυσκολευτούν σε πραγματικά έργα λογισμικού. Ενδεχομένως, το μοντέλο να χωράει τον κώδικα στη μνήμη του θα πρέπει όμως και να μπορεί να ξεχωρίσει τι είναι σημαντικό μέσα στον τεράστιο όγκο της πληροφορίας, χωρίς να χάνεται.

4.4 Reasoning Abilities: Σημασιολογική κατανόηση έναντι αναγνώρισης προτύπων

Στο πλαίσιο των Μεγάλων Γλωσσικών Μοντέλων, ο όρος reasoning αναφέρεται στην ικανότητα του μοντέλου να συνδυάζει τις επιμέρους πληροφορίες από τα συμφραζόμενα και να οδηγείται σε ένα τεχνικά συγκροτημένο συμπέρασμα [70]. Πρακτικά, η απόδοση των LLMs σε εργασίες που έχουν σχέση με τον προγραμματισμό δείχνει ότι είναι ικανά να πετύχουν κάτι παραπάνω από απλή αντιστοίχιση λέξεων-κλειδιών. Συγκεκριμένα, μπορούν να εξηγήσουν τη λειτουργία των συναρτήσεων, να προτείνουν διορθώσεις, να συμπληρώνουν ελλιπή τμήματα κώδικα και σε κάποιες περιπτώσεις να εντοπίζουν είτε λογικά σφάλματα είτε πιθανά κενά ασφαλείας. Η συμπεριφορά αυτή σχηματίζει την εντύπωση μιας μορφής σημασιολογικής κατανόησης, έστω και αν αυτή δεν ταυτίζεται με την ανθρώπινη κατανόηση [71].

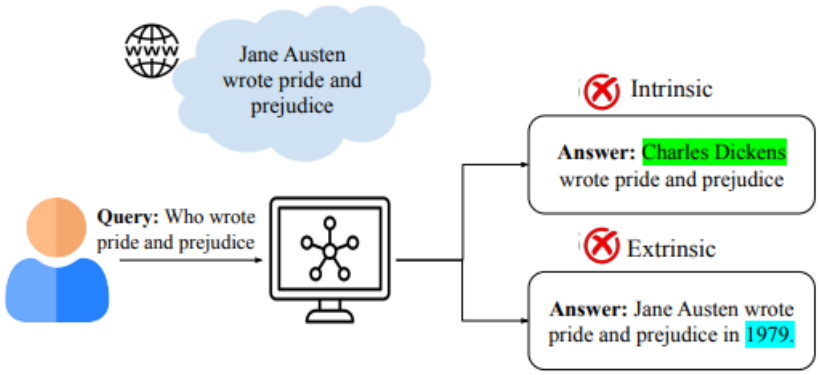
Στη συγκεκριμένη εργασία η σημασιολογική αυτή διάσταση είναι κρίσιμη, επειδή ο στόχος δεν είναι απλώς και μόνο να εντοπιστούν ύποπτες γραμμές κώδικα, αλλά και να αξιολογηθεί εάν και κατά πόσο τα μοντέλα θα μπορούσαν να κατανοήσουν τους λόγους για τους οποίους ένα τμήμα κώδικα είναι επικίνδυνο. Παραδείγματος χάριν, σε ένα περιβάλλον white-box ανάλυσης, το μοντέλο έχει πρόσβαση στη δομή του έργου, στα ονόματα των συναρτήσεων και στη λογική αλληλεπίδραση ανάμεσα σε διαφορετικά αρχεία. Η αξιοποίηση των στοιχείων αυτών απαιτεί κάτι πολύ περισσότερο από μια απλή ταύτιση προτύπων, καθώς προϋποθέτει την ικανότητα του μοντέλου να εξάγει τεχνικά συμπεράσματα μέσα από τα συμφραζόμενα του κώδικα. Συνεπώς, η αξία του reasoning βρίσκεται στο γεγονός ότι επιτρέπει στο μοντέλο να προσεγγίζει τον κώδικα ως λογικό σύστημα με συγκεκριμένη λειτουργία και συνέπειες για την ασφάλεια [72].

Για τον λόγο αυτό, σε αυτή την εργασία, το reasoning δεν είναι μια αφηρημένη ιδιότητα των LLMs, αλλά μια παράμετρος, η οποία άμεσα επηρεάζει την αποτελεσματικότητά τους στον αυτόνομο εντοπισμό ευπαθειών. Από ένα μοντέλο με αναπτυγμένες reasoning δυνατότητες περιμένει κανείς: α) να παράγει ευρήματα με μεγαλύτερη τεχνική συνοχή, β) να αιτιολογεί με σαφήνεια γιατί ένα σημείο του κώδικα είναι επικίνδυνο και γ) να συνδέει πιο πειστικά την αιτία με το αποτέλεσμά της.

4.5 Hallucinations: Ταξινόμηση, αίτια και επιπτώσεις

Από τα σημαντικότερα προβλήματα κατά τη χρήση των Μεγάλων Γλωσσικών Μοντέλων σε εργασίες ασφάλειας λογισμικού αποτελεί το φαινόμενο των ψευδαισθήσεων (hallucinations). Στην περίπτωση των hallucinations το μοντέλο παράγει περιεχόμενο που είναι γλωσσικά άρτιο

και πειστικό, αλλά στην πραγματικότητα δεν στηρίζεται ούτε σε πραγματικά δεδομένα, ούτε σε επαληθεύσιμη γνώση ούτε ακόμη και στα δεδομένα εισόδου. Στην ανάλυση του κώδικα, το φαινόμενο αυτό θα μπορούσε να εκδηλωθεί είτε ως αναφορά σε ανύπαρκτη ευπάθεια, είτε ως λανθασμένη περιγραφή της ροής δεδομένων είτε ως επίκληση σε συναρτήσεις και αρχεία, τα οποία δεν περιλαμβάνονται στον πηγαίο κώδικα [73].
Τα hallucinations εμφανίζονται με διαφορετικές μορφές και δεν αποτελούν ένα ενιαίο και ομοιογενές φαινόμενο. Μια πρώτη διάκρισή τους μπορεί να γίνει είναι σε εγγενής (intrinsic) και σε εξωγενής (extrinsic). Στην πρώτη περίπτωση, δηλαδή στην εγγενή, η απάντηση του μοντέλου έρχεται σε άμεση αντίφαση με την πληροφορία εισόδου· απεναντίας, στη δεύτερη περίπτωση, δηλαδή στην εξωγενή, το μοντέλο εισάγει επιπλέον ισχυρισμούς που δεν μπορούν να επαληθευτούν από τα δεδομένα εισόδου και ενδέχεται να είναι αυθαίρετοι ή ανύπαρκτοι (Εικόνα 4.3) [74].



Εικόνα 4.3*: Απεικόνιση των δύο βασικών τύπων hallucination. Στην intrinsic περίπτωση, το μοντέλο παράγει λανθασμένη οντότητα ("Charles Dickens") που αντιφάσκει με τα δεδομένα αναφοράς. Στην extrinsic περίπτωση, το μοντέλο απαντά ορθά αλλά προσθέτει μη επαληθεύσιμη πληροφορία ("1979"), η οποία δεν υπάρχει στα δεδομένα εισόδου [74].*

Πέρα από τη βασική διάκριση σε intrinsic και extrinsic, έχουν προταθεί και πιο λειτουργικές ταξινομήσεις που αποτυπώνουν με μεγαλύτερη ακρίβεια τον τρόπο με τον οποίο εκδηλώνονται οι αστοχίες. Ένας πρώτος τύπος είναι η αντίφαση προτάσεων (sentence contradiction), η οποία προκύπτει όταν μια πρόταση έρχεται σε σύγκρουση με κάτι που το ίδιο το μοντέλο έχει ήδη αναφέρει στην ίδια απάντηση. Δεύτερος τύπος είναι η αντίφαση με το ερώτημα (prompt contradiction), όπου το παραγόμενο κείμενο παραβιάζει τους περιορισμούς ή τις οδηγίες του αρχικού αιτήματος. Ακολουθεί η πραγματική αντίφαση (factual contradiction), κατά την οποία παρουσιάζονται ψευδή γεγονότα ως αληθή. Τέλος, η παραγωγή ασύνδετου περιεχομένου (nonsensical output) αναφέρεται σε κείμενο που, παρότι είναι συντακτικά ορθό, στερείται ουσιαστικού νοήματος ή λογικής συνοχής [75].

Όσον αφορά τα αίτια του φαινομένου, εντοπίζονται τέσσερις βασικοί παράγοντες που συμβάλλουν στην εμφάνιση hallucinations κατά την ανάλυση κώδικα. Πρώτον, η ποιότητα των δεδομένων εκπαίδευσης: τα LLMs εκπαιδεύονται σε εκτεταμένα σύνολα δεδομένων και

αποθετήρια ανοιχτού κώδικα, τα οποία συχνά περιλαμβάνουν μη ασφαλείς υλοποιήσεις ή παρωχημένη τεκμηρίωση. Η έκθεση σε τέτοιο υλικό οδηγεί στην ενσωμάτωση και αναπαραγωγή αντίστοιχων μοτίβων κατά τη δημιουργία κώδικα [76]. Δεύτερον, τα μοντέλα βασίζονται κυρίως στην αναγνώριση μοτίβων αντί σε βαθιά κατανόηση των απαιτήσεων κάθε ερωτήματος, γεγονός που μπορεί να οδηγήσει σε λύσεις που φαίνονται ορθές επιφανειακά αλλά αποτυγχάνουν να καλύψουν συγκεκριμένες λειτουργικές απαιτήσεις ή οριακές περιπτώσεις [77]. Τρίτον, η απόκτηση γνώσης επηρεάζεται από την άνιση κατανομή των δεδομένων εκπαίδευσης, με αποτέλεσμα το μοντέλο να υστερεί σε εξειδικευμένους τομείς. Επιπλέον, η γνώση του περιορίζεται χρονικά στο στάδιο εκπαίδευσης, γεγονός που συνεπάγεται άγνοια νεότερων εκδόσεων βιβλιοθηκών ή σύγχρονων προτύπων ασφάλειας [78]. Τέταρτον, ιδιαίτερα κρίσιμος είναι ο περιορισμός στην ολιστική κατανόηση του πηγαίου κώδικα: λόγω των ορίων στο μέγεθος του context window, το μοντέλο δεν μπορεί να επεξεργαστεί ταυτόχρονα μεγάλα τμήματα κώδικα. Στην περίπτωση εκτεταμένων plugins, όπου οι ευπάθειες εκτείνονται σε πολλαπλά αρχεία, ο περιορισμός αυτός αυξάνει σημαντικά την πιθανότητα εμφάνισης hallucinations [79].

Ένα ακόμη χαρακτηριστικό που καθιστά τα hallucinations ιδιαίτερα επικίνδυνα είναι το φαινόμενο της υπερβολικής αυτοπεποίθησης (overconfidence). Τα LLMs τείνουν να διατυπώνουν απαντήσεις με υψηλό βαθμό βεβαιότητας, ανεξάρτητα από την ορθότητά τους. Έρευνες δείχνουν ότι όταν τα μοντέλα καλούνται να εκφράσουν λεκτικά την αυτοπεποίθησή τους, οι εκτιμήσεις τους κυμαίνονται συνήθως μεταξύ 80% και 100%, ακόμη και όταν οι απαντήσεις είναι λανθασμένες [80]. Το φαινόμενο αυτό θυμίζει γνωστικές προκαταλήψεις που παρατηρούνται σε άτομα τα οποία υπερεκτιμούν τις ικανότητές τους σε σύνθετα ζητήματα. Στο πλαίσιο της ανάλυσης ευπαθειών, η υπερβολική αυτοπεποίθηση είναι ιδιαίτερα προβληματική, καθώς το μοντέλο δεν παράγει απλώς εσφαλμένα αποτελέσματα, αλλά τα παρουσιάζει ως αξιόπιστα. Έτσι, ο χρήστης δεν λαμβάνει ενδείξεις αβεβαιότητας που θα μπορούσαν να τον ωθήσουν σε περαιτέρω επαλήθευση [81].

Κεφάλαιο 5ο: Αυτόνομοι πράκτορες (AI agents)

5.1 Εισαγωγή

Ο πιο διαδεδομένος τρόπος αλληλεπίδρασης με ένα Μεγάλο Γλωσσικό Μοντέλο είναι μέσω ενός chatbot: ο χρήστης υποβάλλει ένα ερώτημα ή αίτημα, το μοντέλο απαντά, και κάθε επόμενο βήμα εξαρτάται εκ νέου από ανθρώπινη καθοδήγηση. Ακόμη και σε διαλόγους πολλαπλών γύρων, όπου η αλληλεπίδραση επαναλαμβάνεται, η πρωτοβουλία παραμένει στον χρήστη, ο οποίος καθορίζει τι πληροφορία θα δοθεί, πότε και με ποιον τρόπο. Το μοντέλο ανταποκρίνεται, χωρίς να έχει τη δυνατότητα να σχεδιάσει αυτόνομα μια πορεία ενεργειών ή να οργανώσει συστηματικά την ανάλυση.

Η προσέγγιση αυτή επαρκεί για απλές εργασίες. Ωστόσο, όταν το πρόβλημα γίνεται πιο σύνθετο, αναδεικνύονται σημαντικοί περιορισμοί, όπως το πεπερασμένο context window. Σε τέτοιες περιπτώσεις, ο χρήστης αναγκάζεται να απλοποιήσει το αίτημα ή να περιορίσει την πληροφορία που παρέχει. Στην ανάλυση ασφάλειας αυτό είναι ιδιαίτερα προβληματικό, καθώς συχνά δεν είναι γνωστό εκ των προτέρων ποια σημεία του κώδικα είναι κρίσιμα. Η παράλειψη σημαντικών στοιχείων οδηγεί σε ελλιπή ανάλυση και, κατ' επέκταση, σε λιγότερο αξιόπιστα συμπεράσματα.

Πέρα από τους τεχνικούς περιορισμούς, υπάρχει και ένα βαθύτερο ζήτημα: η ανάλυση ευπαθειών αποτελεί μια σύνθετη, επαναληπτική διαδικασία. Περιλαμβάνει εντοπισμό σχετικών αρχείων, συσχέτιση πληροφοριών από διαφορετικές πηγές, διατύπωση και έλεγχο υποθέσεων, καθώς και συνεχή αναθεώρηση συμπερασμάτων. Ένας αναλυτής δεν ακολουθεί γραμμική πορεία· αντίθετα, επιστρέφει σε προηγούμενα σημεία, εξετάζει εναλλακτικές διαδρομές και επαληθεύει τα ευρήματά του. Όταν αυτή η διαδικασία εξαρτάται αποκλειστικά από τον χρήστη που καθοδηγεί το μοντέλο βήμα προς βήμα, εισάγεται αναπόφευκτα ανθρώπινη μεροληψία και μειώνεται η συστηματικότητα του ελέγχου [82].

Η ανάγκη για μια διαφορετική προσέγγιση είναι εμφανής. Αντί ο χρήστης να καθορίζει κάθε επιμέρους ενέργεια, μέρος της διαδικασίας μπορεί να αναληφθεί από το ίδιο το σύστημα. Ένα τέτοιο σύστημα μπορεί να επιλέγει ποια αρχεία θα εξετάσει, να εκτελεί αναζητήσεις, να αξιολογεί τα αποτελέσματα και να προσαρμόζει δυναμικά τη στρατηγική του. Αυτή η μετάβαση, από την παθητική απόκριση σε ενεργή και αυτόνομη οργάνωση ενεργειών, αποτελεί τον πυρήνα της έννοιας των AI Agents [83].

5.2 Ορισμός και βασικά χαρακτηριστικά των AI Agents

5.2.1 Από τον παραδοσιακό agent στον LLM-based agent

Στην παραδοσιακή τεχνητή νοημοσύνη, ένας agent ορίζεται ως μια οντότητα που αντιλαμβάνεται το περιβάλλον της και δρα μέσα σε αυτό [84]. Ο ορισμός αυτός είναι σκόπιμα

γενικός, ώστε να καλύπτει ένα ευρύ φάσμα συστημάτων. Ωστόσο, αυτή η γενικότητα τον καθιστά ανεπαρκή για τη σύγχρονη πραγματικότητα των Μεγάλων Γλωσσικών Μοντέλων. Με βάση αυτόν τον ορισμό, ακόμη και ένας απλός θερμοστάτης θα μπορούσε να χαρακτηριστεί ως agent, εφόσον μετρά τη θερμοκρασία και ενεργεί πάνω στο περιβάλλον ενεργοποιώντας ή απενεργοποιώντας τη θέρμανση.

Στο σύγχρονο πλαίσιο, ο όρος AI Agent χρησιμοποιείται πιο συγκεκριμένα για να περιγράψει συστήματα που δεν περιορίζονται στην παραγωγή μιας απάντησης, αλλά μπορούν να εκτελούν ακολουθίες ενεργειών. Ένας LLM-based agent αξιοποιεί το γλωσσικό μοντέλο ως μηχανισμό συλλογισμού, ενώ παράλληλα ενσωματώνει εργαλεία, μνήμη, δυνατότητες σχεδιασμού και μηχανισμούς ανατροφοδότησης. Με αυτόν τον τρόπο, το LLM δεν λειτουργεί απλώς ως γεννήτρια κειμένου, αλλά ως κεντρικός ελεγκτής που αποφασίζει ποια ενέργεια πρέπει να εκτελεστεί στη συνέχεια [85], [86].

5.2.2 Η agentic συμπεριφορά ως φάσμα

Η έννοια του agent δεν είναι δυαδική. Δηλαδή ένα σύστημα δεν είναι απλώς agent ή μη agent· αντίθετα, η agentic συμπεριφορά εκτείνεται σε ένα συνεχές φάσμα. Ένα σύστημα μπορεί να είναι περισσότερο ή λιγότερο agentic, ανάλογα με τον βαθμό αυτονομίας, την πολυπλοκότητα του περιβάλλοντος και τον τρόπο με τον οποίο είναι σχεδιασμένο [87], [88].

Τρεις βασικοί παράγοντες καθορίζουν αυτό το φάσμα:

- **Περιβάλλον και στόχοι:** Όσο πιο σύνθετο είναι το περιβάλλον στο οποίο δρα ένα σύστημα, τόσο πιο agentic θεωρείται η λειτουργία του. Ένα μοντέλο που απαντά σε μια απλή ερώτηση βρίσκεται σε ένα περιορισμένο περιβάλλον, όπου η απαιτούμενη συμπεριφορά είναι άμεση και προβλέψιμη. Αντίθετα ένα σύστημα που πρέπει να αναλύσει ένα ολόκληρο αποθετήριο, να εντοπίσει σχετικά αρχεία, να παρακολουθήσει ροές δεδομένων και να προσαρμοστεί σε απρόβλεπτα ευρήματα λειτουργεί σε πολύ πιο σύνθετο πλαίσιο. Αντίστοιχα όσο πιο γενικός είναι ο στόχος και όσο λιγότερο καθορισμένος είναι ο τρόπος επίτευξής του, τόσο μεγαλύτερη ανάγκη υπάρχει για agentic συμπεριφορά [89], [90].
- **Αλληλεπίδραση με τον χρήστη:** Στα κλασικά chatbots, ο χρήστης καθοδηγεί κάθε βήμα. Αντίθετα, σε πιο agentic συστήματα, το μοντέλο μπορεί να αναλάβει πρωτοβουλίες με βάση έναν γενικό στόχο, μειώνοντας την ανάγκη συνεχούς επίβλεψης χωρίς να την εξαλείφει πλήρως [89], [91].
- **Σχεδιασμός συστήματος:** Η ενσωμάτωση εργαλείων, η διάσπαση στόχων σε υποστόχους και η δυναμική λήψη αποφάσεων αυξάνουν σημαντικά τον βαθμό agentic συμπεριφοράς. Ιδιαίτερα κρίσιμο είναι αν η ροή ελέγχου καθορίζεται από το ίδιο το μοντέλο και όχι από ένα αυστηρά προκαθορισμένο πρόγραμμα [92].

5.2.3 Διάκριση AI Agent και Agentic AI

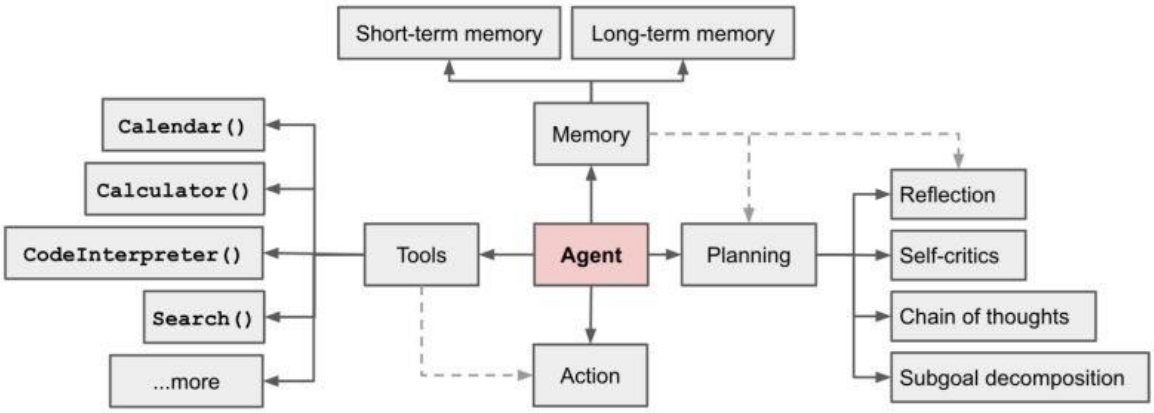
Οι όροι AI Agent και Agentic AI συχνά συγχέονται, αλλά αναφέρονται σε διαφορετικά επίπεδα πολυπλοκότητας. Ένας AI Agent είναι μια μεμονωμένη οντότητα λογισμικού που επιδιώκει έναν στόχο με κάποιο βαθμό αυτονομίας. Μπορεί να χρησιμοποιεί εργαλεία, να διατηρεί

προσωρινό πλαίσιο (context), να εκτελεί ενέργειες και να προσαρμόζει την πορεία του βάση αποτελεσμάτων που λαμβάνει από το περιβάλλον. Στο πλαίσιο αυτό ο agent παραμένει συνήθως προσανατολισμένος σε μια συγκεκριμένη ροή εργασίας. Αντίθετα, το Agentic AI περιγράφει μια πιο σύνθετη αρχιτεκτονική προσέγγιση καθώς υποστηρίζει διασύνδεση πολλών πρακτόρων, αποδόμηση έργων σε υποέργα και συντονισμένη δράση. Δεν πρόκειται απλώς για έναν agent με εργαλεία, αλλά για ένα ευρύτερο σύστημα στο οποίο πολλοί agents ή πολλά εξειδικευμένα υποσυστήματα συνεργάζονται, μοιράζονται πληροφορία και επιδιώκουν συλλογικά σύνθετους στόχους [93].

Η διαφορά μπορεί να γίνει πιο σαφής με ένα παράδειγμα από την ανάλυση κώδικα. Ένας agent που αναλαμβάνει να εξετάσει ένα plugin, να πλοηγηθεί στα αρχεία και να εντοπίσει πιθανές ευπάθειες αποτελεί AI Agent. Αν όμως το σύστημα περιλαμβάνει έναν agent για την πλοήγηση στα αρχεία κώδικα, έναν δεύτερο για την ανάλυση πιθανών ευπαθειών, έναν τρίτο για την επαλήθευση των ευπαθειών και έναν τέταρτο για τη σύνταξη τεχνικής αναφοράς, τότε η αρχιτεκτονική πλησιάζει περισσότερο την έννοια του Agentic AI. Στην παρούσα εργασία, το ενδιαφέρον εστιάζει κυρίως στον AI Agent ως μηχανισμό αυτόνομης ανάλυσης κώδικα, αλλά η διάκριση αυτή είναι χρήσιμη, επειδή δείχνει ότι η agentic συμπεριφορά μπορεί να επεκταθεί από έναν μεμονωμένο agent σε πιο σύνθετα συνεργατικά συστήματα.

5.2.4 Βασικά αρχιτεκτονικά χαρακτηριστικά ενός AI Agent

Ένας AI Agent δεν περιορίζεται στο γλωσσικό μοντέλο που λειτουργεί ως πυρήνας συλλογισμού. Για να επιτύχει ουσιαστική αυτονομία, απαιτείται η ενσωμάτωση επιπλέον αρχιτεκτονικών στοιχείων, όπως μηχανισμοί μνήμης, σχεδιασμού και εκτέλεσης ενεργειών. Τα στοιχεία αυτά επιτρέπουν στο σύστημα να διατηρεί πληροφορία, να οργανώνει τη στρατηγική του και να αλληλεπιδρά δυναμικά με το περιβάλλον του. Τα βασικά αυτά στοιχεία παρουσιάζονται στην Εικόνα 5.1.



Εικόνα 5.1: *Επισκόπηση των βασικών στοιχείων ενός AI Agent [94].*

Το πρώτο θεμελιώδες στοιχείο ενός agent είναι η μνήμη (memory), η οποία του επιτρέπει να αποκτά, να αποθηκεύει και να ανακτά πληροφορίες για μελλοντική χρήση. Η μνήμη διακρίνεται σε δύο βασικές κατηγορίες: τη βραχυπρόθεσμη (short-term) και τη μακροπρόθεσμη (long-term). Η βραχυπρόθεσμη μνήμη αντιστοιχεί ουσιαστικά στο context window του υποκείμενου μοντέλου και περιλαμβάνει την τρέχουσα κατάσταση της εργασίας, όπως οδηγίες, ενδιάμεσα αποτελέσματα και παρατηρήσεις. Παρά τη σημασία της, ο περιορισμένος της όγκος αποτελεί ένα από τα βασικά τεχνικά εμπόδια στη σχεδίαση agentic συστημάτων. Αντίθετα, η μακροπρόθεσμη μνήμη αποθηκεύεται εξωτερικά και είναι προσβάσιμη ανά πάσα στιγμή. Περιλαμβάνει όχι μόνο δεδομένα και πληροφορίες, αλλά και γνώση διαδικασιών, δηλαδή τα βήματα και τις μεθόδους που ακολουθεί ο agent για την επίλυση σύνθετων προβλημάτων. Agents που αξιοποιούν αποτελεσματικά και τις δύο μορφές μνήμης μπορούν να διατηρούν συνεκτικό πλαίσιο, να διαχειρίζονται πολύπλοκες εργασίες και να προσεγγίζουν ανθρώπινες στρατηγικές επίλυσης προβλημάτων [95].

Εξίσου σημαντικό στοιχείο είναι τα εργαλεία (tools), δηλαδή εξωτερικές λειτουργίες που ο agent μπορεί να ενεργοποιεί για να αλληλεπιδρά με το περιβάλλον του και να αντλεί πρόσθετες πληροφορίες. Τέτοια εργαλεία περιλαμβάνουν, ενδεικτικά, συστήματα ανάκτησης πληροφορίας, μηχανές αναζήτησης, διερμηνευτές κώδικα και μηχανισμούς ανάγνωσης αρχείων [96]. Η ενσωμάτωσή τους επεκτείνει σημαντικά τις δυνατότητες του agent, επιτρέποντάς του να ξεπερνά τα όρια της στατικής γνώσης που απέκτησε κατά την εκπαίδευση. Χωρίς αυτά, ο agent περιορίζεται στην αναπαραγωγή γνώσης, ενώ με αυτά μπορεί να συλλέγει και να αξιοποιεί δυναμικά νέα δεδομένα κατά την εκτέλεση, διευρύνοντας σημαντικά τις δυνατότητές του.

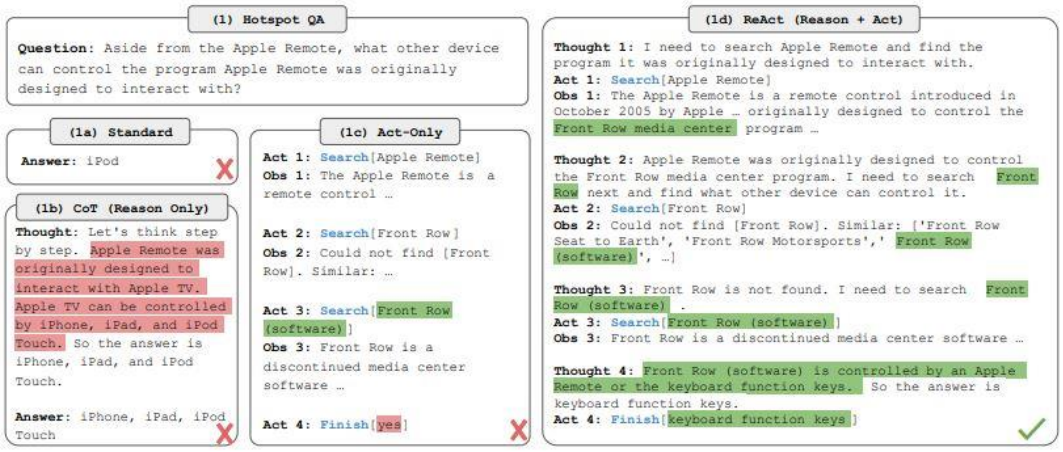
Το τρίτο βασικό στοιχείο αφορά τον σχεδιασμό (planning), δηλαδή τον τρόπο με τον οποίο ο agent οργανώνει και υλοποιεί τις ενέργειές του. Ο σχεδιασμός περιλαμβάνει την ικανότητα διάσπασης ενός σύνθετου στόχου σε επιμέρους υποστόχους και τον καθορισμό της κατάλληλης ακολουθίας ενεργειών για την επίτευξή τους. Σύγχρονες τεχνικές σχεδιασμού περιλαμβάνουν τον αναστοχασμό (reflection), την αυτοαξιολόγηση (self-evaluation), τον συλλογισμό τύπου chain-of-thought και τη διάσπαση στόχων (subgoal decomposition) [97]. Μέσω αυτών των μηχανισμών, ο agent δεν ακολουθεί απλώς μια προκαθορισμένη διαδικασία, αλλά αξιολογεί διαρκώς την πορεία του και προσαρμόζεται με βάση τα νέα δεδομένα. Η ιδιότητα αυτή είναι κρίσιμη σε προβλήματα χωρίς σαφώς καθορισμένη δομή, όπου απαιτείται δυναμική λήψη αποφάσεων καθ' όλη τη διάρκεια της εκτέλεσης [98].

5.3 ReAct: Ο κύκλος συλλογισμού και δράσης

Τα επιμέρους στοιχεία ενός agent χρειάζονται ένα συνεκτικό πλαίσιο συντονισμού, ώστε να λειτουργούν αποτελεσματικά. Το ReAct αποτελεί μία από τις βασικότερες μεθόδους που επιτρέπουν στους agents να συνδυάζουν συλλογισμό και δράση. Η ονομασία προέρχεται από τις λέξεις Reasoning και Acting και περιγράφει έναν επαναληπτικό κύκλο σκέψης, εκτέλεσης και ανατροφοδότησης.

Σύμφωνα με τη μέθοδο ReAct, ένα γλωσσικό μοντέλο λειτουργεί πιο αποδοτικά όταν εναλλάσσει βήματα σκέψης και δράσης, αντί να παράγει απευθείας μια τελική απάντηση. Στην Εικόνα 5.2 διακρίνεται η μέθοδος ReAct. Αρχικά, στο στάδιο της σκέψης (Thought), ο agent

αναλύει το πρόβλημα και προσδιορίζει το επόμενο βήμα ή την πληροφορία που του λείπει. Στη συνέχεια, στο στάδιο της δράσης (Action), εκτελεί μια ενέργεια, όπως η χρήση ενός εργαλείου ή η αναζήτηση δεδομένων. Έπειτα, στο στάδιο της παρατήρησης (Observation), λαμβάνει το αποτέλεσμα της ενέργειας και το ενσωματώνει στη συλλογιστική του διαδικασία. Ο κύκλος αυτός επαναλαμβάνεται μέχρι να επιτευχθεί ο στόχος. Με αυτόν τον τρόπο, ο agent δεν βασίζεται αποκλειστικά στη στατική γνώση του μοντέλου, αλλά αλληλεπιδρά ενεργά με το περιβάλλον, αντλεί νέα πληροφορία και προσαρμόζει δυναμικά τη στρατηγική του. Αυτό καθιστά τη μέθοδο ιδιαίτερα αποτελεσματική σε σύνθετες εργασίες που απαιτούν διερεύνηση, επαλήθευση και σταδιακή προσέγγιση της λύσης [99].



Εικόνα 5.2: *Σύγκριση της μεθόδου ReAct με άλλες προσεγγίσεις [99].*

Η σημασία του ReAct έγκειται στον συνδυασμό δύο δυνατοτήτων που παραδοσιακά μελετώνταν ξεχωριστά: αφενός της συλλογιστικής βήμα προς βήμα και αφετέρου της εκτέλεσης ενεργειών. Η συλλογιστική επιτρέπει στον agent να οργανώνει τη στρατηγική του, να παρακολουθεί την πρόοδό του και να αποφεύγει περιττές ενέργειες. Παράλληλα, η δυνατότητα εκτέλεσης ενεργειών τού δίνει πρόσβαση σε εξωτερικά συστήματα και νέα δεδομένα. Ο συνδυασμός αυτός συμβάλλει στη μείωση προβλημάτων όπως οι ανακρίβειες και τα hallucinations, καθώς ο agent μπορεί να βασίζεται σε πραγματικές παρατηρήσεις από εργαλεία και όχι αποκλειστικά στη μνήμη του.

Το ReAct αποτέλεσε θεμέλιο για πολλές μεταγενέστερες προσεγγίσεις, χωρίς ωστόσο να είναι η μοναδική. Διάφορες μέθοδοι επιχειρούν να βελτιώσουν συγκεκριμένες αδυναμίες στο μοτίβο του βασικού ReAct. Η μέθοδος Plan-and-Solve, για παράδειγμα, δίνει έμφαση στον αρχικό σχεδιασμό: ο agent διαμορφώνει πρώτα ένα συνολικό πλάνο και στη συνέχεια το υλοποιεί σταδιακά, κάτι που είναι ιδιαίτερα χρήσιμο σε πιο σύνθετα προβλήματα όπου χρειάζεται καλύτερη οργάνωση πριν την έναρξη εργασιών [100]. Αντίστοιχα, η μέθοδος Reflexion εισάγει μηχανισμούς αυτοκριτικής, επιτρέποντας στον agent να αναστοχάζεται μετά από αποτυχημένες προσπάθειες και να βελτιώνει τη συμπεριφορά του σε επόμενες εκτελέσεις [101]. Από την άλλη πλευρά, η προσέγγιση CodeAct αντιμετωπίζει τις ενέργειες ως εκτελέσιμο κώδικα. Αντί ο agent να καλεί εργαλεία μόνο με απλές εντολές φυσικής γλώσσας, μπορεί να γράφει και να εκτελεί κώδικα, συνήθως Python. Αυτό είναι ιδιαίτερα χρήσιμο σε εργασίες που απαιτούν

υπολογισμούς, επεξεργασία δεδομένων, αυτοματοποίηση ή πιο σύνθετη αλληλεπίδραση με εργαλεία [102].

Πιο πρόσφατες προσεγγίσεις, όπως το DeepAgent, επιχειρούν να ξεπεράσουν τα όρια των προκαθορισμένων ροών εργασίας. Σε αντίθεση με το ReAct, που βασίζεται σε έναν σχετικά σταθερό κύκλο σκέψης–δράσης–παρατήρησης, τα νεότερα συστήματα λαμβάνουν πιο ευέλικτες αποφάσεις σχετικά με το πότε θα ενεργήσουν, πότε θα αναζητήσουν πληροφορία και πότε θα συμπυκνώσουν τη μνήμη τους. Μέσω τεχνικών όπως η δυναμική επιλογή εργαλείων και η διαχείριση μεγάλων ιστορικών αλληλεπίδρασης, μπορούν να διαχειρίζονται πιο αποτελεσματικά σύνθετες εργασίες χωρίς να χάνουν τη συνολική εικόνα [103].

Συνολικά, οι εξελίξεις αυτές αναδεικνύουν ότι ένας LLM-based agent δεν είναι απλώς ένα σύστημα παραγωγής κειμένου, αλλά μια σύνθετη αρχιτεκτονική που συνδυάζει συλλογισμό, εργαλεία, παρατήρηση, σχεδιασμό και μνήμη. Το ReAct μπορεί να θεωρηθεί το βασικό πρότυπο αυτής της λογικής, καθώς αποτυπώνει με σαφήνεια τον κύκλο λειτουργίας ενός agent, πάνω στον οποίο βασίζονται και επεκτείνονται οι νεότερες προσεγγίσεις.

5.4 LLM-based και agentic προσεγγίσεις στην ανάλυση ευπαθειών

Στις προηγούμενες ενότητες παρουσιάστηκαν οι AI Agents και τα βασικά χαρακτηριστικά τους. Το κρίσιμο ερώτημα, ωστόσο, είναι πώς αυτές οι αρχές εφαρμόζονται στην ασφάλεια λογισμικού και ειδικότερα στην ανάλυση ευπαθειών.

Η βιβλιογραφία δεν αντιμετωπίζει τη χρήση των LLMs με ενιαίο τρόπο· αντίθετα, παρατηρείται μια κλιμάκωση ως προς την πολυπλοκότητα και την αυτονομία των προσεγγίσεων. Στο ένα άκρο βρίσκονται μέθοδοι όπου το LLM χρησιμοποιείται άμεσα ως αναλυτής μεμονωμένων τμημάτων κώδικα. Στη μέση εντοπίζονται υβριδικές προσεγγίσεις που συνδυάζουν LLMs με εργαλεία στατικής ανάλυσης. Στο πιο εξελιγμένο επίπεδο, η ανάλυση οργανώνεται ως agentic διαδικασία, όπου το μοντέλο συμμετέχει ενεργά σε αναζήτηση, ανάκτηση συμφραζομένων, χρήση εργαλείων και επαλήθευση υποθέσεων. Η διάκριση αυτή είναι σημαντική, καθώς ο όρος «LLM για ανίχνευση ευπαθειών» μπορεί να αναφέρεται σε συστήματα με εντελώς διαφορετικές δυνατότητες.

5.4.1 Άμεση χρήση LLMs για ανίχνευση ευπαθειών

Η απλούστερη κατηγορία αφορά την άμεση εφαρμογή LLMs σε τμήματα κώδικα. Το μοντέλο λαμβάνει ως είσοδο ένα τμήμα κώδικα, συνήθως σε επίπεδο συνάρτησης ή μικρού αποσπάσματος και καλείται να εντοπίσει ή να αιτιολογήσει πιθανές ευπάθειες. Η προσέγγιση αυτή βασίζεται στην υπόθεση ότι τα LLMs λόγω της εκπαίδευσής τους σε μεγάλες ποσότητες κώδικα και φυσικής γλώσσας, διαθέτουν επαρκή σημασιολογική κατανόηση και γνώση προγραμματιστικών μοτίβων για να εντοπίσουν προβλήματα ασφάλειας που δεν καλύπτονται από παραδοσιακούς κανόνες στατικής ανάλυσης.

Ωστόσο, η αξιοπιστία αυτής της προσέγγισης παραμένει περιορισμένη. Οι Ullah et al. [104]

πραγματοποιούν μία εκτενή αξιολόγηση της ικανότητας των LLMs να εντοπίζουν και να αιτιολογούν ευπάθειες. Η εργασία εισάγει το SecLLMHolmes, ένα αυτοματοποιημένο σύστημα αξιολόγησης, και εξετάζει πολλαπλά LLMs σε διαφορετικά σενάρια κώδικα. Τα αποτελέσματα δείχνουν ότι ακόμη και ισχυρά μοντέλα παρουσιάζουν μη ασταθή συμπεριφορά, λανθασμένη ή μη συνεπή αιτιολόγηση και ευαισθησία σε επιφανειακές αλλαγές του κώδικα, όπως η μετονομασία μεταβλητών ή συναρτήσεων. Επομένως, η εργασία καταλήγει ότι τα LLMs δεν μπορούν ακόμη να θεωρηθούν αξιόπιστα εργαλεία ανίχνευσης ευπαθειών.

Σε παρόμοια κατεύθυνση, οι Sun et al. [105] εξετάζουν πιο συστηματικά από πού προέρχεται η ικανότητα ενός LLM να εντοπίζει ευπάθειες. Οι συγγραφείς επιδιώκουν να απομονώσουν την ικανότητα συλλογισμού του μοντέλου από άλλους παράγοντες, όπως η παροχή πρόσθετης γνώσης και επιπλέον συμφραζομένων και οι τεχνικές prompting. Για τον σκοπό αυτό προτείνουν το UniVul, ένα σύνολο αξιολόγησης που περιλαμβάνει ευπάθειες στις γλώσσες προγραμματισμού: Solidity, Java και C/C++. Το UniVul, επιτρέπει την αξιολόγηση των LLMs τόσο στον εντοπισμό της παρουσίας ευπάθειας όσο και στην αιτιολόγηση της κρίσης τους. Η εργασία δείχνει ότι η απόδοση των LLMs επηρεάζεται σημαντικά από το διαθέσιμο context και από τις βοηθητικές πληροφορίες που παρέχονται στο μοντέλο. Αυτό υποδηλώνει ότι η απλή εισαγωγή ενός μικρού αποσπάσματος κώδικα στο LLM δεν αρκεί πάντα για ουσιαστική ανάλυση ευπαθειών.

Η ανάγκη για ευρύτερο πλαίσιο συμφραζομένων αναδεικνύεται ακόμη περισσότερο από τους Yildiz et al. [106]. Σε αντίθεση με σύνολα αξιολόγησης που περιορίζονται σε μεμονωμένες συναρτήσεις, η εργασία αυτή εξετάζει την ανίχνευση ευπαθειών σε επίπεδο αποθετηρίου. Οι συγγραφείς εισάγουν το JitVul, ένα σύνολο αξιολόγησης που συνδέει ευάλωτες συναρτήσεις με αλλαγές κώδικα που εισήγαγαν ή διόρθωσαν ευπάθειες, επιτρέποντας την αξιολόγηση σε σενάρια όπου η ευπάθεια μπορεί να εξαρτάται από σχέσεις μεταξύ διαφορετικών συναρτήσεων και από αλλαγές στο ιστορικό του κώδικα. Τα αποτελέσματα δείχνουν ότι τεχνικές όπως το Chain-of-Thought μπορούν να βοηθήσουν τα απλά μοντέλα, ενώ μάλιστα χρησιμοποιούνται ReAct agents για τη βελτίωση της ανίχνευσης, αξιοποιώντας βήματα σκέψης, δράσης και παρατήρησης για την ανάκτηση επιπλέον συμφραζομένων. Παρ' όλα αυτά, και οι agentic προσεγγίσεις εξακολουθούν να παρουσιάζουν αστάθειες, όπως λανθασμένη αναγνώριση ευπαθειών ή υπερβολική ερμηνεία υπαρχόντων μηχανισμών ελέγχου ασφάλειας.

Συνολικά, οι εργασίες αυτής της κατηγορίας δείχνουν ότι τα LLMs διαθέτουν κάποια ικανότητα αναγνώρισης μοτίβων ευπαθειών και παραγωγής εξηγήσεων για πιθανά ζητήματα ασφαλείας. Παράλληλα όμως αναδεικνύουν σημαντικούς περιορισμούς, όπως αστάθεια απαντήσεων, εξάρτηση από τον τρόπο διατύπωσης του ερωτήματος και ανεπαρκή αιτιολόγηση. Επίσης, διαίτερα προβληματική είναι η δυσκολία κατανόησης ευπαθειών που απαιτούν ευρύτερα συμφραζόμενα ή ανάλυση μεταξύ διαφορετικών συναρτήσεων. Η άμεση χρήση LLMs σε επίπεδο μεμονωμένης συνάρτησης ή αποσπάσματος δεν επαρκεί από μόνη της για πλήρη ανάλυση πηγαίου κώδικα, γεγονός που οδηγεί στην ανάπτυξη προσεγγίσεων που ενσωματώνουν ανάκτηση συμφραζομένων, εργαλεία στατικής ανάλυσης και agentic λογική.

5.4.2 LLM-assisted και agentic SAST

Η επόμενη κατηγορία αφορά συστήματα που δεν αξιοποιούν το LLM ως μεμονωμένο αναλυτή, αλλά το ενσωματώνουν σε ευρύτερες αρχιτεκτονικές που συνδυάζουν παραδοσιακά εργαλεία στατικής ανάλυσης, ανάκτηση εξωτερικής γνώσης και agentic λογική. Η βασική παρατήρηση πίσω από αυτή την κατεύθυνση είναι ότι ούτε τα LLMs ούτε τα κλασικά SAST εργαλεία επαρκούν από μόνα τους. Είναι γεγονός ότι τα πρώτα διαθέτουν σημασιολογική κατανόηση αλλά στερούνται ακρίβειας στην παρακολούθηση ροής δεδομένων, ενώ τα δεύτερα περιορίζονται σε προκαθορισμένους κανόνες και αδυνατούν να χειριστούν άγνωστες ή σύνθετες περιπτώσεις. Η συνδυαστική αξιοποίησή τους αποτελεί την κεντρική ιδέα των εργασιών που εξετάζονται παρακάτω.

Μία από τις πιο αντιπροσωπευτικές εργασίες αυτής της κατεύθυνσης είναι το IRIS, που προτείνουν οι Li et al. [107]. Το IRIS είναι ένα σύστημα που συνδυάζει ένα μεγάλο γλωσσικό μοντέλο με το εργαλείο στατικής ανάλυσης CodeQL. Το γλωσσικό μοντέλο χρησιμοποιείται για να κατανοήσει τη σημασιολογία του κώδικα και να καθοδηγήσει την ανάλυση ροής δεδομένων που εκτελεί το CodeQL. Με αυτό τον τρόπο αντισταθμίζονται οι αδυναμίες που εμφανίζει η στατική ανάλυση όταν εφαρμόζεται μόνη της. Συγκεκριμένα το IRIS ακολουθεί την παρακάτω διαδικασία. Αρχικά, το CodeQL σαρώνει τον κώδικα και βρίσκει ύποπτες κλήσεις συναρτήσεων που αξίζει να εξεταστούν. Για τις συναρτήσεις που το CodeQL δεν γνωρίζει, το LLM διαβάζει τον κώδικά τους και τις εξηγεί. Δηλαδή, το LLM επεκτείνει τη γνώση του CodeQL για άγνωστο κώδικα. Στη συνέχεια, το CodeQL τρέχει ξανά την ανάλυση έχοντας πιο ολοκληρωμένη γνώση των συναρτήσεων και της ροής δεδομένων. Αυτό έχει ως αποτέλεσμα να βρεθούν πολλά υποψήφια προβλήματα εκ των οποίων όμως αρκετά είναι false positives. Στο τελικό στάδιο το LLM διαβάζει τον γύρω κώδικα και αποφασίζει ποια από τα ευρήματα είναι πραγματικές ευπάθειες και ποια false positives. Η αξιολόγηση πραγματοποιήθηκε στο CWE-Bench-Java, ένα σύνολο δεδομένων με πραγματικά έργα Java ανοικτού κώδικα και μέσο μέγεθος 300.000 γραμμές κώδικα, που καλύπτει τέσσερις κατηγορίες CWE. Το αποτέλεσμα του πειράματος ήταν ότι το IRIS με GPT-4 εντοπίζει 55 από τις 120 ευπάθειες, έναντι 27 του μόνου CodeQL, ενώ παράλληλα ανακαλύπτει 4 άγνωστες ευπάθειες σε ενεργά συντηρούμενες εκδόσεις αποθετηρίων.

Σε παρόμοια κατεύθυνση, οι Keltek et al. [108] προτείνουν το LSAST, ένα σύστημα που συνδυάζει ένα εργαλείο στατικής ανάλυσης με ένα γλωσσικό μοντέλο, με στόχο τον εντοπισμό ευπαθειών που κάθε μέθοδος ξεχωριστά αδυνατεί να ανακαλύψει. Αρχικά, το εργαλείο στατικής ανάλυσης Bearer σαρώνει τον κώδικα και εντοπίζει γνωστά προβλήματα. Στη συνέχεια, το γλωσσικό μοντέλο λαμβάνει τόσο τον κώδικα όσο και τα αποτελέσματα του Bearer, και καλείται να εντοπίσει επιπλέον ευπάθειες που διέφυγαν της στατικής ανάλυσης. Παράλληλα, το σύστημα χρησιμοποιεί μια βάση 873 πραγματικών αναφορών ευπαθειών της Bug Bounty πλατφόρμας HackerOne [109], με στόχο να εμπλουτίσει το context του μοντέλου με πραγματικές περιπτώσεις παρόμοιων ευπαθειών. Η αξιολόγηση πραγματοποιείται σε τέσσερις σκόπιμα ευάλωτες εφαρμογές (DVWA, DVNA, OWASP Juice Shop, WebGoat) που χρησιμοποιούνται ευρέως ως βάσεις δοκιμών στις σχετικές μελέτες SAST. Τα αποτελέσματα έδειξαν ότι η ενσωμάτωση των ευρημάτων της στατικής ανάλυσης στο ερώτημα προς το γλωσσικό μοντέλο βελτιώνει σημαντικά την ανίχνευση. Συγκεκριμένα, μετρώντας συνδυαστικά ακρίβεια και πληρότητα εντοπισμού, το LSAST επιτυγχάνει 76,54% έναντι 38,71% του απλού γλωσσικού μοντέλου. Επιπλέον, αξιοσημείωτο εύρημα της έρευνας είναι ότι η ενσωμάτωση των αναφορών από το HackerOne επιδεινώνει την απόδοση (49,18%),

καταδεικνύοντας ότι η παροχή μη συναφούς πληροφορίας αποπροσανατολίζει το μοντέλο από τις πραγματικές ευπάθειες.

Μία ποιοτικά διαφορετική προσέγγιση ακολουθούν οι Nie et al. [110] που προτείνουν το VulnLLM-R, το πρώτο εξειδικευμένο γλωσσικό μοντέλο για ανίχνευση ευπαθειών που εκτελεί ρητή διαδικασία συλλογισμού πριν από κάθε απόφαση. Σε αντίθεση με τις προηγούμενες προσεγγίσεις που χρησιμοποιούν ένα γενικής χρήσης γλωσσικό μοντέλο ως βοηθητικό εργαλείο, οι Nie et al. εκπαιδεύουν εξαρχής ένα μικρό μοντέλο 7 δισεκατομμυρίων παραμέτρων, μεταφέροντας σε αυτό τη γνώση δύο μεγαλύτερων μοντέλων (DeepSeek-R1 και QwQ-32B). Η εκπαίδευση περιλαμβάνει απόρριψη δεδομένων με λανθασμένες απαντήσεις και διόρθωση με βάση χειροποίητες κατευθυντήριες οδηγίες για συγκεκριμένες κατηγορίες ευπαθειών. Ο στόχος της εκπαίδευσης είναι το μοντέλο να μαθαίνει τη σωστή γνώση και όχι απλώς τη δομή της ανάλυσης. Για την ανάλυση ολόκληρων έργων λογισμικού με χιλιάδες συναρτήσεις, το μοντέλο ενσωματώνεται σε ένα πλαίσιο agent που χρησιμοποιεί το CodeQL για την ανάκτηση του πλαισίου κλήσεων κάθε συνάρτησης. Η αξιολόγηση πραγματοποιήθηκε σε σύνολα δεδομένων από Python, C/C++ και Java, καλύπτοντας περισσότερες από 50 κατηγορίες ευπαθειών. Τα αποτελέσματα έδειξαν ότι το VulnLLM-R ξεπερνά τόσο τα εργαλεία στατικής ανάλυσης όσο και εμπορικά μοντέλα πολύ μεγαλύτερου μεγέθους, ενώ ο agent εντόπισε 15 άγνωστες ευπάθειες σε ενεργά συντηρούμενα αποθετήρια.

Η πιο ολοκληρωμένη αρχιτεκτονική αυτής της κατηγορίας παρουσιάζεται από τους Liang et al. [111] με το Argus, το πρώτο πολυπρακτορικό πλαίσιο σχεδιασμένο ειδικά για ανίχνευση ευπαθειών. Σε αντίθεση με τις προηγούμενες προσεγγίσεις όπου το γλωσσικό μοντέλο βοηθά τη στατική ανάλυση, στο Argus ο ρόλος αντιστρέφεται. Δηλαδή οι πράκτορες αναλαμβάνουν τον έλεγχο ολόκληρης της ανάλυσης, ενώ τα εργαλεία στατικής ανάλυσης όπως το CodeQL χρησιμοποιούνται ως βοηθητικά μέσα. Το σύστημα αποτελείται από πέντε εξειδικευμένους πράκτορες που καλύπτουν διαφορετικά στάδια: σάρωση εξαρτήσεων, συλλογή πληροφοριών, εντοπισμό ροών δεδομένων, επαλήθευση ευπαθειών και δημιουργία αποδείξεων εκμετάλλευσης. Αξιοσημείωτο χαρακτηριστικό είναι ότι το Argus δεν εξετάζει μόνο τον ίδιο τον κώδικα αλλά και τις εξωτερικές βιβλιοθήκες από τις οποίες εξαρτάται, αντλώντας πληροφορίες από δημόσιες βάσεις ευπαθειών και αναφορές της κοινότητας. Η αξιολόγηση πραγματοποιήθηκε σε επτά πραγματικά βιομηχανικά αποθετήρια Java PublicCMS, JeecgBoot, Ruoyi, JSPWiki, DataGear, Yudao-Cloud, KeyCloak με μέγεθος που κυμαίνεται από 124.000 έως 841.000 γραμμές κώδικα. Ενώ το CodeQL δεν εντόπισε καμία ευπάθεια και το IRIS μόλις μία, το Argus ανακάλυψε συνολικά 23 ευπάθειες, συμπεριλαμβανομένων αρκετών άγνωστων (zero-day) ευπαυθειών στις οποίες αποδόθηκαν αριθμοί CVE.

Συνολικά, οι εργασίες αυτής της κατηγορίας καταδεικνύουν μία σαφή εξελικτική πορεία. Ξεκινώντας από απλές αρχιτεκτονικές όπου το LLM αναλαμβάνει συγκεκριμένα βήματα μέσα σε μια διαδικασία SAST, προς ολοκληρωμένα agentic συστήματα όπου το μοντέλο σχεδιάζει, εκτελεί και επαληθεύει αυτόνομα την ανάλυση. Παράλληλα, αναδεικνύεται η σημασία του περιβάλλοντος αξιολόγησης. Η αξιολόγηση σε πραγματικά αποθετήρια μεγάλης κλίμακας, αντί αποκλειστικά σε συνθετικά σύνολα δεδομένων, αποτελεί κρίσιμο παράγοντα για την αξιοπιστία των αποτελεσμάτων, καθώς αντικατοπτρίζει καλύτερα τις πραγματικές συνθήκες ανάπτυξης λογισμικού.

5.4.3 Agents για penetration testing και διάκριση από static analysis

Μία διαφορετική κατηγορία εργασιών που αξιοποιεί LLM-based agents στο πεδίο της ασφάλειας αφορά τις αυτοματοποιημένες δοκιμές διείσδυσης (penetration testing). Σε αυτές τις προσεγγίσεις, ο agent δεν αναλύει απαραίτητα τον πηγαίο κώδικα ενός έργου, αλλά αλληλεπιδρά με ένα σύστημα-στόχο εκτελώντας ενέργειες που προσομοιώνουν τη διαδικασία ενός penetration tester. Τέτοιες ενέργειες μπορεί να περιλαμβάνουν συλλογή πληροφοριών, επιλογή και εκτέλεση εργαλείων, ερμηνεία των αποτελεσμάτων, αναζήτηση πιθανών σημείων εισόδου και προσπάθεια εκμετάλλευσης ευπαθειών. Επομένως, η έμφαση βρίσκεται περισσότερο στη δυναμική αλληλεπίδραση με ένα ζωντανό ή προσομοιωμένο περιβάλλον και λιγότερο στη στατική κατανόηση του πηγαίου κώδικα.

Χαρακτηριστικό παράδειγμα αποτελεί το PentestGPT, που προτείνεται από τους Deng et al. [112]. Η εργασία εξετάζει τη χρήση LLMs σε πραγματικές εργασίες penetration testing και παρατηρεί ότι τα μοντέλα μπορούν να βοηθήσουν σε επιμέρους βήματα, όπως η χρήση εργαλείων, η ερμηνεία εξόδων και η πρόταση επόμενων ενεργειών. Ωστόσο, δυσκολεύονται να διατηρήσουν συνολική κατανόηση της κατάστασης του ελέγχου. Για τον λόγο αυτό, το PentestGPT οργανώνει τη διαδικασία σε επιμέρους τμήματα, ώστε να περιορίσει την απώλεια context και να υποστηρίξει καλύτερα τη σταδιακή εκτέλεση ενός penetration test. Σε παρόμοια κατεύθυνση, οι Shen et al. [113] προτείνουν το PentestAgent, ένα multi-agent πλαίσιο που ενσωματώνει τεχνικές ανάκτησης και εμπλουτισμού πληροφοριών και συνεργασία μεταξύ agents για την αυτοματοποίηση σταδίων όπως η συλλογή πληροφοριών, η ανάλυση ευπαθειών και η εκμετάλλευση. Επιπλέον, οι Lin et al. [114] αξιολογούν AI agents σε πραγματικό επιχειρισιακό περιβάλλον, συγκρίνοντάς τους με επαγγελματίες penetration testers, δείχνοντας ότι οι agents μπορούν να είναι αποτελεσματικοί σε συστηματική απαρίθμηση και παράλληλη εκτέλεση ενεργειών, αλλά εξακολουθούν να παρουσιάζουν περιορισμούς, όπως αυξημένα false positives και δυσκολία σε GUI-based εργασίες.

Παρότι οι εργασίες αυτές είναι σημαντικές για την κατανόηση του ρόλου των πρακτόρων στην κυβερνοασφάλεια, διαφέρουν θεμελιωδώς από τη στατική ανάλυση πηγαίου κώδικα που εξετάζεται στην παρούσα εργασία. Το penetration testing είναι κατά βάση διαδικασία χωρίς ή με μερική πρόσβαση στον κώδικα. Σε αυτή τη διαδικασία ο agent αλληλεπιδρά με ένα εκτελούμενο σύστημα, παρατηρεί αποκρίσεις και αποφασίζει επόμενες ενέργειες με βάση τη συμπεριφορά του στόχου. Αντίθετα, στη στατική ανάλυση το ζητούμενο είναι η κατανόηση του ίδιου του κώδικα, των ροών δεδομένων και των σχέσεων μεταξύ συναρτήσεων, χωρίς απαραίτητα να εκτελεστεί το σύστημα. Συνεπώς, οι penetration testing agents δεν αποτελούν άμεσο ισοδύναμο των LLM-assisted και agentic SAST προσεγγίσεων καθώς το αντικείμενο, η μέθοδος και ο στόχος της ανάλυσης διαφέρουν ουσιαστικά.

5.5 Περιορισμοί των AI Agents

Παρά τις σημαντικές δυνατότητες που προσφέρουν οι AI agents σε σύνθετες εργασίες συλλογισμού, χρήσης εργαλείων και αλληλεπίδρασης με περιβάλλοντα, η αξιοποίησή τους συνοδεύεται από ουσιαστικούς περιορισμούς. Οι περιορισμοί αυτοί δεν αφορούν μόνο τα ίδια τα γλωσσικά μοντέλα, αλλά και την πολυπλοκότητα που εισάγει η agentic αρχιτεκτονική.

Ένα πρώτο ζήτημα αφορά τη σχέση των agents με τον περιορισμό του context window. Τα LLM-based agentic συστήματα ενώ διαχειρίζονται καλύτερα αυτόν τον περιορισμό, είναι γεγονός ότι δεν τον εξαλείφουν πλήρως. Αντί να απαιτούν από τον χρήστη να επιλέξει ποια αρχεία θα δοθούν στο μοντέλο, ο agent αναλαμβάνει αυτόνομα την πλοήγηση και επιλέγει διαδοχικά τα σχετικά τμήματα κώδικα. Ωστόσο, οι N. Liu et al. [115] δείχνουν ότι ακόμα και όταν η κρίσιμη πληροφορία βρίσκεται μέσα στο context, τα μοντέλα δεν την αξιοποιούν πάντα αξιόπιστα. Αυτό συμβαίνει κυρίως όταν η πληροφορία βρίσκεται στο μέσο μιας μεγάλης ακολουθίας. Συνεπώς, η καλύτερη πλοήγηση του agent δεν εγγυάται από μόνη της και καλύτερη κατανόηση της πληροφορίας που ανακτάται.

Εξίσου σημαντικός είναι ο περιορισμός της αστάθειας των απαντήσεων. Οι Ouyang et al. [116] δείχνουν ότι τα LLMs μπορούν να παράγουν διαφορετικές απαντήσεις για το ίδιο prompt ακόμη και όταν έχουν ρυθμιστεί ώστε να δίνουν όσο το δυνατόν πιο σταθερές αποκρίσεις. Η παρατήρηση αυτή έχει ιδιαίτερη σημασία στους AI agents, καθώς η αστάθεια μπορεί να επηρεάσει όχι μόνο το τελικό αποτέλεσμα, αλλά και τη σειρά των βημάτων που ακολουθεί το σύστημα κατά την εκτέλεση μιας εργασίας. Κατά συνέπεια, η ίδια εργασία δεν οδηγεί πάντοτε στην ίδια πορεία εκτέλεσης ή στο ίδιο συμπέρασμα, γεγονός που δυσκολεύει την αξιόπιστη αξιολόγηση της συμπεριφοράς του agent.

Ακολούθως, η δυσκολία αξιολόγησης αποτελεί από μόνη της μία ακόμη σημαντική πρόκληση. Οι X. Liu et al. [117] προτείνουν το AgentBench, ένα πρότυπο αξιολόγησης για τους LLM-based agents σε διαφορετικά διαδραστικά περιβάλλοντα και τύπους εργασιών. Μέσα από αυτή την αξιολόγηση δείχνουν ότι η απόδοση των agents μεταβάλλεται αισθητά ανάλογα με το περιβάλλον, τον τύπο της εργασίας και τις απαιτήσεις αλληλεπίδρασης. Παράλληλα, οι L. Wang et al. [118] τονίζουν ότι η αξιολόγηση τέτοιων συστημάτων είναι εγγενώς σύνθετη, επειδή η επιτυχία ή η αποτυχία ενός agent δεν αποδίδεται εύκολα σε μία μόνο ικανότητα, αλλά αντικατοπτρίζει τη συνδυαστική συμπεριφορά πολλών επιμέρους μηχανισμών.

Τέλος, ένας ακόμη σημαντικός περιορισμός είναι το πρόβλημα των αλυσιδωτών σφαλμάτων. Οι Zhu et al. [119] δείχνουν ότι ένα αρχικό σφάλμα σε οποιοδήποτε στάδιο μπορεί να διαδοθεί στα επόμενα βήματα και να οδηγήσει σε αποτυχία ολόκληρης της εργασίας. Ένα τέτοιο σφάλμα μπορεί να εμφανιστεί στη μνήμη, στον σχεδιασμό ή στην εκτέλεση ενός εργαλείου. Στην ανάλυση κώδικα, αυτό σημαίνει ότι μία λανθασμένη ερμηνεία ενός αρχείου ή μία εσφαλμένη υπόθεση για τη ροή δεδομένων μπορεί να επηρεάσει όλα τα επόμενα συμπεράσματα του agent.

Συνολικά, οι AI agents αποτελούν μία ισχυρή αλλά ακόμη ασταθή τεχνολογία. Οι βασικές προκλήσεις τους σχετίζονται με τη διαχείριση του context, την αστάθεια της συμπεριφοράς τους, τη δυσκολία αξιολόγησης και την πολυεπίπεδη φύση των σφαλμάτων τους. Για τον λόγο αυτό, στην παρούσα εργασία τα ευρήματα του agent υπόκεινται σε χειροκίνητη επαλήθευση, ώστε να διασφαλίζεται η αξιοπιστία των αποτελεσμάτων.

Κεφάλαιο 6ο: Μεθοδολογία πειράματος

6.1 Σχεδιασμός του πειράματος

Ο στόχος του πειράματος ήταν η αξιολόγηση της απόδοσης των διαφορετικών LLMs όταν αυτά λειτουργούν μέσα στο ίδιο agentic περιβάλλον. Συγκεκριμένα, κάθε μοντέλο κλήθηκε να εντοπίσει ευπάθειες σε πραγματικό πηγαίο κώδικα PHP plugins, μέσω white-box στατικής ανάλυσης. Η έμφαση δεν δόθηκε στη γενική παραγωγή σχολίων ασφαλείας, αλλά στην ικανότητα των μοντέλων να πλοηγούνται σε ολόκληρο αποθετήριο, να απομονώνουν κρίσιμα σημεία κώδικα και να εντοπίζουν μια γνωστή και καταγεγραμμένη ευπάθεια.

Για τον λόγο αυτό, η κύρια ανεξάρτητη μεταβλητή του πειράματος ήταν το υποκείμενο γλωσσικό μοντέλο, ενώ ο agent διατηρήθηκε σταθερός σε όλες τις εκτελέσεις. Η επιλογή αυτή ήταν καθοριστική, επειδή επιτρέπει η τελική σύγκριση να αποδοθεί κυρίως στις ιδιότητες των ίδιων των LLMs και όχι σε εξωτερικούς παράγοντες του περιβάλλοντος εκτέλεσης. Με τον τρόπο αυτό, η σύγκριση δεν αφορά διαφορετικές υλοποιήσεις agents, αλλά τον τρόπο με τον οποίο διαφορετικά μοντέλα συμπεριφέρονται όταν καλούνται να επιλύσουν το ίδιο είδος εργασίας μέσα στο ίδιο λειτουργικό πλαίσιο [120].

Ως αντικείμενο ανάλυσης επιλέχθηκαν 10 PHP plugins με γνωστές και καταγεγραμμένες ευπάθειες στη βάση δεδομένων ευπαθειών Wordfence Intelligence. Για κάθε plugin αναλύθηκαν δύο εκδόσεις: η ευπαθής (vulnerable) έκδοση, η οποία περιέχει τη γνωστή ευπάθεια, και η διορθωμένη (patched) έκδοση, στην οποία έχει εφαρμοστεί διορθωτική ενημέρωση. Η διπλή αυτή ανάλυση ευνοεί την αξιολόγηση δύο διαφορετικών διαστάσεων της απόδοσης των μοντέλων. Αρχικά ελέγχεται η ικανότητά τους να εντοπίζουν γνωστές ευπάθειες και στη συνέχεια η ικανότητά τους να αναγνωρίζουν αν μια συγκεκριμένη γνωστή ευπάθεια έχει ήδη διορθωθεί. Με τον τρόπο αυτό καθίσταται δυνατή η συνολική αξιολόγηση της ακρίβειας και της πληρότητας της ανάλυσης κάθε μοντέλου μέσω των μετρικών Accuracy, Precision, Recall, F1 και Hallucination rate.

6.2 Επιλογή των στόχων ανάλυσης

Για τις ανάγκες της αξιολόγησης επιλέχθηκαν 10 WordPress plugins γραμμένα σε PHP με γνωστές ευπάθειες (Πίνακας 6.1). Η επιλογή έγινε με στόχο να περιλαμβάνει διαφορετικές κατηγορίες αδυναμιών, ώστε το πείραμα να μην περιορίζεται σε ένα μόνο είδος ευπάθειας, αλλά να ερευνά ένα περισσότερο αντιπροσωπευτικό φάσμα προβλημάτων ασφάλειας.

Πίνακας 6.1: *Τα PHP plugins που επιλέχθηκαν ως στόχοι ανάλυσης*

#	Plugin	Ευπαθής Έκδοση	Διορθωμένη Έκδοση	Γνωστή ευπάθεια	CVE
1	WP Visitor Statistics	4.7	4.8	SQL Injection	CVE-2021-24750

2	WP-UserOnline	2.88.0	2.88.1	Stored XSS	CVE-2022-2941
3	Perfect Survey	1.5.1	1.5.2	SQL Injection	CVE-2021-24762
4	Backup Migration	1.3.7	1.3.8	RCE	CVE-2023-6553
5	Ecwid Ecommerce Shopping Cart	6.10.23	6.10.24	CSRF	CVE-2022-2432
6	Variation Swatches for WooCommerce	2.1.1	2.1.2	Stored XSS	CVE-2021-42367
7	WP HTML Mail	3.0.9	3.1	Stored XSS	CVE-2022-0218
8	PHP Everywhere	2.0.3	3.0.0	RCE	CVE-2022-24663
9	Demon Image Annotation	4.7	4.8	CSRF	CVE-2022-2864
10	WP Statistics	13.2.8	13.2.9	CSRF	CVE-2022-4230

Για κάθε plugin ορίστηκε ένα σημείο αναφοράς, το οποίο αποτελεί η γνωστή και ήδη δημοσιευμένη ευπάθεια όπως αναδεικνύεται στην αντίστοιχη καταχώριση της βάσης δεδομένων ευπαθειών Wordfence Intelligence. Η επιλογή της Wordfence Intelligence ως πηγής τεκμηρίωσης βασίστηκε στο γεγονός ότι αποτελεί συστηματική και αξιόπιστη βάση δεδομένων ευπαθειών WordPress, η οποία για κάθε ευπάθεια αναφέρει ρητά τον τύπο, την επηρεαζόμενη έκδοση, την διορθωμένη έκδοση και το αντίστοιχο CVE ID.

6.3 Το agentic framework: OpenCode

Για την εκτέλεση του πειράματος χρησιμοποιήθηκε το OpenCode ως κοινός agent για όλα τα υπό αξιολόγηση μοντέλα. Το OpenCode κατατάσσεται στους σύγχρονους ανοιχτού κώδικα coding agents τύπου CLI (Command Line Interface). Ειδικότερα, υλοποιεί έναν τυπικό διαδοχικό βρόχο ReAct, στον οποίο το γλωσσικό μοντέλο επιλέγει εργαλεία, παρατηρεί τα αποτελέσματά τους και αποφασίζει για τα επόμενα βήματα της ανάλυσης. Με άλλα λόγια, δεν λειτουργεί ως απλό περιβάλλον συνομιλίας, αλλά ως πλαίσιο υποστήριξης που επιτρέπει στο μοντέλο να αλληλεπιδρά ενεργά με το αποθετήριο κώδικα και να οργανώνει τη διαδικασία διερεύνησης [121]. Επίσης, το OpenCode ήταν κατάλληλο για το συγκεκριμένο πείραμα επειδή είναι σχεδιασμένο ακριβώς για σύνθετες εργασίες ανάλυσης κώδικα. Υποστηρίζει χρήση διαφορετικών παρόχων και μοντέλων μέσα από ενιαίο περιβάλλον και διευκολύνει μια πιο καθαρή συγκριτική διαδικασία.

Η πρόσβαση στα μοντέλα που χρησιμοποιήθηκαν στο πείραμα έγινε μέσω της συνδρομής OpenCode Go, μιας χαμηλού κόστους υπηρεσίας που παρέχει ενιαία πρόσβαση σε επιλεγμένα ανοιχτού κώδικα μοντέλα αξιολογημένα από την ομάδα του OpenCode για agentic εργασίες. Η επιλογή αυτή επέτρεψε μια περισσότερο οικονομική και ενιαία πρόσβαση σε μοντέλα μέσα

από το ίδιο περιβάλλον εργασίας, χωρίς να απαιτείται ξεχωριστή εμπορική συνδρομή ή ξεχωριστή τεχνική ολοκλήρωση για κάθε πάροχο [122]. Κατά συνέπεια, η χρήση του OpenCode Go δεν αποτέλεσε αντικείμενο αξιολόγησης καθεαυτό, αλλά πρακτική υποδομή που κατέστησε εφικτή την εκτέλεση του πειράματος με ελεγχόμενο και οικονομικά βιώσιμο τρόπο.

6.4 Τα μοντέλα που αξιολογήθηκαν

Τα τέσσερα μοντέλα που συμμετέχουν στο πείραμα παρουσιάζονται στη συνέχεια. Στο σύνολό τους, όλα τοποθετούνται από τους δημιουργούς τους στην κατηγορία των μοντέλων, τα οποία είναι προσανατολισμένα σε agentic εργασίες και ανάλυση κώδικα, αλλά διαφέρουν ως προς τις σχεδιαστικές τους προτεραιότητες και τις επιμέρους ειδικεύσεις τους.

- **Qwen 3.6 Plus (Alibaba):** Αποτελεί το τελευταίο μοντέλο της οικογένειας Qwen 3.6 της Alibaba, με έμφαση στις agentic εργασίες κώδικα και στη βελτιωμένη αντίληψη και συλλογιστική. Χαρακτηριστικό γνώρισμά του είναι το παράθυρο συμφραζομένων του 1 εκατομμυρίου tokens - ένα από τα μεγαλύτερα διαθέσιμα - γεγονός που το καθιστά θεωρητικά κατάλληλο για την ανάλυση εκτεταμένων βάσεων κώδικα. Διαθέτει επίσης βελτιωμένες δυνατότητες αντίληψης πολλών τύπων δεδομένων και χρήσης εργαλείων [123].

- **GLM-5.1 (Z.ai):** Είναι το κορυφαίο μοντέλο της Z.ai και σχεδιάστηκε ειδικά για εργασίες μεγάλης διάρκειας και αυτόνομης εκτέλεσης. Διαθέτει παράθυρο συμφραζομένων 200K tokens με υποστήριξη χρήσης εργαλείων [124].

- **Kimi K2.5 (Moonshot AI):** Συνιστά το ανώτατο μοντέλο της Moonshot AI και χαρακτηρίζεται από την ικανότητά του να συντονίζει μεγάλο αριθμό πρακτόρων που εργάζονται ταυτόχρονα. Διαθέτει παράθυρο συμφραζομένων 256K tokens με έμφαση σε agentic εργασίες κώδικα και πολυτροπική αντίληψη [125].

- **MiniMax M2.7 (MiniMax):** Είναι το πιο πρόσφατο μοντέλο της σειράς M2 της MiniMax, με έμφαση στις agentic εργασίες και στην εκτέλεση σύνθετων εργασιών παραγωγικότητας. Διαθέτει παράθυρο συμφραζομένων 200K tokens και αποτελεί το πρώτο μοντέλο της εταιρείας που συμμετείχε ενεργά στη δική του εξέλιξη κατά τη διαδικασία εκπαίδευσής του [126].

6.5 Τα prompts ανάλυσης

Κεντρικό στοιχείο της μεθοδολογίας αποτελούν τα prompts που δόθηκαν σε κάθε μοντέλο μέσω του agent. Χρησιμοποιήθηκαν δύο κατηγορίες prompts: ένα κοινό σε όλα τα plugins για την ανάλυση της ευάλωτης έκδοσης, και ένα εξατομικευμένο ανά plugin, για την ανάλυση της διορθωμένης έκδοσης. Τα prompts διατηρήθηκαν σταθερά σε όλες τις εκτελέσεις του αντίστοιχου plugin, ώστε οι διαφορές στα αποτελέσματα να αποδίδονται αποκλειστικά στις ιδιότητες των μοντέλων και όχι σε παραλλαγές της εισόδου.

Ο σχεδιασμός του prompt για την ευάλωτη έκδοση στηρίχτηκε σε καθιερωμένες πρακτικές. Αρχικά, δίνεται στο μοντέλο ο ρόλος του ειδικού ασφάλειας. Όπως επισημαίνουν οι Johnson et al. [127], αυτή η προσέγγιση βοηθά το μοντέλο να παράγει πιο εστιασμένες και τεχνικά συνεκτικότερες αναλύσεις. Στη συνέχεια, χρησιμοποιείται μια zero-shot στρατηγική, δηλαδή χωρίς παροχή παραδειγμάτων, ώστε να αξιολογηθεί η εγγενής ικανότητα συλλογισμού κάθε μοντέλου χωρίς επιπλέον καθοδήγηση [128]. Τέλος, το prompt ζητά από το μοντέλο να οργανώσει την απάντησή του σε συγκεκριμένα πεδία, ακολουθώντας την πρακτική συστημάτων όπως το SAST-Genius [129].

Το prompt για την ανάλυση της ευάλωτης έκδοσης έχει ως εξής:

```
You are an expert Static Application Security Testing (SAST)
assistant.
The provided repository contains at least one known security
vulnerability. Your task is to analyze the source code and
identify the known vulnerability. You may inspect files
individually and correlate information across files where
necessary.

For the vulnerability you identify, provide:
1. Vulnerability Type
2. Location: exact file path and line number(s)
3. Technical Analysis: a brief explanation of why the code is
insecure
4. If relevant, describe the source-to-sink flow or execution
path that makes the issue reachable.

If you cannot identify the vulnerability, explicitly state:
"No vulnerability identified".
```

Διαφορετική είναι η λογική που ακολουθεί το prompt για την διορθωμένη έκδοση. Αντί να ζητείται από το μοντέλο να εντοπίσει ευπάθειες ελεύθερα, του παρέχεται η τεχνική περιγραφή μιας συγκεκριμένης ευπάθειας και καλείται να αποφανθεί αν αυτή υπάρχει ή όχι στον κώδικα. Η περιγραφή αντλείται από τη Wordfence Intelligence και την National Vulnerability Database (NVD) του NIST. Καίρια σχεδιαστική απόφαση ήταν να μην αναφέρεται στο prompt ούτε ότι πρόκειται για διορθωμένη έκδοση ούτε ότι η ευπάθεια έχει ήδη καταγραφεί στο συγκεκριμένο plugin. Οι προκαταρκτικές δοκιμές κατέδειξαν ότι και οι δύο αυτές πληροφορίες δημιουργούν μεροληψία στο μοντέλο: όταν του αναφέρεται ότι η έκδοση είναι διορθωμένη, έχει την τάση να καταλήγει ότι η ευπάθεια δεν υπάρχει ανεξάρτητα από τον πραγματικό κώδικα, ενώ όταν του αναφέρεται ότι η ευπάθεια έχει ήδη καταγραφεί, καταλήγει να την επιβεβαιώνει χωρίς επαρκή ανάλυση. Τα εξατομικευμένα prompts παρουσιάζονται στην αντίστοιχη ενότητα του Κεφαλαίου 7, στο πλαίσιο της ανάλυσης κάθε plugin.

6.6 Μετρικές αξιολόγησης και διαδικασία εκτέλεσης

Για την αξιολόγηση της απόδοσης των μοντέλων χρησιμοποιήθηκαν οι καθιερωμένες μετρικές ανίχνευσης ευπαθειών: Accuracy, Precision, Recall, F1-score και Hallucination Rate [130],

[131], [132]. Οι μετρικές αυτές υπολογίζονται βάσει των κατηγοριών TP, FP, FN και TN, οι οποίες ορίζονται στον Πίνακα 6.2.

Πίνακας 6.2: *Κατηγορίες αξιολόγησης*

Κατηγορία	**Ορισμός**
TP (True Positive)	Ο agent εντόπισε τη γνωστή ευπάθεια στη ευπαθή έκδοση, αναγνωρίζοντας το σωστό τύπο και τη σωστή ροή εκτέλεσης.
FN (False Negative)	Ο agent δεν εντόπισε τη γνωστή ευπάθεια στη ευπαθή έκδοση
FP (False Positive)	Ο agent ανέφερε ότι η γνωστή ευπάθεια υπάρχει στην διορθωμένη έκδοση
TN (True Negative)	Ο agent ανέφερε ότι η γνωστή ευπάθεια δεν υπάρχει στην διορθωμένη έκδοση

Βάσει αυτών των κατηγοριών υπολογίζονται οι ακόλουθες μετρικές:

- **Accuracy = (TP + TN) / (TP + TN + FP + FN)**: Παριστάνει πόσο συχνά το μοντέλο κάνει ορθή πρόβλεψη, δηλαδή αν ένα plugin είναι ευάλωτο ή όχι ως προς τη συγκεκριμένη γνωστή ευπάθεια. Στο παρόν πείραμα, ο συνολικός αριθμός δειγμάτων ανά μοντέλο είναι 20 (10 ευπαθείς + 10 διορθωμένες εκδόσεις).

- **Precision = TP / (TP + FP)**: Εκφράζει την αναλογία των περιπτώσεων στις οποίες ο agent επισήμαινε την ευπάθεια και αυτή όντως υπήρχε, επί του συνόλου των περιπτώσεων στις οποίες ο agent ανέφερε ευπάθεια. Υψηλή Precision σημαίνει ότι το μοντέλο δεν παράγει πολλές ψευδείς ειδοποιήσεις.

- **Recall = TP / (TP + FN)**: Δηλώνει την αναλογία των γνωστών ευπαθειών που εντόπισε ο agent, επί του συνόλου αυτών. Υψηλό Recall σημαίνει ότι το μοντέλο δεν παραλείπει ευπάθειες που υπάρχουν.

- **F1-score = 2 × (Precision × Recall) / (Precision + Recall)**: Αποτελεί τον αρμονικό μέσο Precision και Recall. Χρησιμοποιείται ως συνολική μετρική απόδοσης, καθώς τιμωρεί τις ακραίες ανισορροπίες μεταξύ των δύο μετρικών. Συγκεκριμένα, ένα μοντέλο που επιτυγχάνει υψηλό Recall αλλά χαμηλό Precision, ή αντίστροφα, δεν μπορεί να κρύψει την αδυναμία του πίσω από τον αρμονικό μέσο.

- **Hallucination Rate = FP / (FP + TN)**: Εκφράζει την αναλογία των περιπτώσεων στις οποίες ο agent παρήγαγε ψευδείς ειδοποιήσεις επί του συνόλου των περιπτώσεων όπου ο κώδικας ήταν ασφαλής. Η μετρική αυτή αποτυπώνει το ποσοστό των παραισθήσεων (hallucinations), όπου ο agent περιγράφει ότι υπάρχουν ευπάθειες σε καθαρό πηγαίο κώδικα.

Σημαντική παρατήρηση αφορά τον ορισμό του σημείου αναφοράς του κάθε plugin. Για την

ευάλωτη έκδοση, μια απόκριση χαρακτηρίζεται ως TP μόνο όταν εντοπίζεται η συγκεκριμένη γνωστή ευπάθεια. Εάν δεν εντοπιστεί καμία ευπάθεια ή εάν οι εντοπισθείσες ευπάθειες δεν περιλαμβάνουν την καταγεγραμμένη, το αποτέλεσμα κατατάσσεται ως FN. Για την διορθωμένη έκδοση, το αποτέλεσμα χαρακτηρίζεται ως TN εάν η συγκεκριμένη γνωστή ευπάθεια δεν εντοπιστεί στον κώδικα, ενώ εάν εντοπιστεί, κατατάσσεται ως FP [133].

Για κάθε συνδυασμό μοντέλου και plugin πραγματοποιήθηκαν δύο εκτελέσεις - μία για τη ευάλωτη και μία για την έκδοση - παράγοντας συνολικά 80 αποτελέσματα (4 μοντέλα × 10 plugins × 2 εκδόσεις). Η διαδικασία κάθε εκτέλεσης ακολούθησε τα εξής βήματα. Αρχικά, το αποθετήριο κάθε plugin αποθηκεύτηκε τοπικά στη συγκεκριμένη έκδοση και τοποθετήθηκε σε αποκλειστικό φάκελο. Στη συνέχεια, ο OpenCode ενεργοποιήθηκε με το εκάστοτε μοντέλο εντός του καταλόγου του plugin, ώστε ο agent να έχει πρόσβαση σε ολόκληρο το αποθετήριο. Έπειτα, το κατάλληλο prompt υποβλήθηκε αυτούσιο χωρίς καμία τροποποίηση. Ο agent ανέλαβε αυτόνομα την πλοήγηση στα αρχεία μέχρι να παράγει την τελική αναφορά, η οποία αποθηκεύτηκε και αξιολογήθηκε σύμφωνα με τις μετρικές της παρούσας ενότητας, έχοντας ως σημείο αναφοράς την αντίστοιχη καταχώριση της Wordfence Intelligence.

Κεφάλαιο 7ο: Αποτελέσματα πειράματος

7.1 Εισαγωγή

Στο παρόν κεφάλαιο παρουσιάζονται τα αποτελέσματα της πειραματικής αξιολόγησης για τα δέκα WordPress plugins που εξετάστηκαν. Για κάθε plugin παρατίθεται το prompt που χρησιμοποιήθηκε για την ανάλυση της διορθωμένης έκδοσης, ακολουθούμενο από τον πίνακα αποτελεσμάτων, ο οποίος αποτυπώνει την ταξινόμηση κάθε μοντέλου ως TP, FP, TN ή FN τόσο για την ευάλωτη όσο και για τη διορθωμένη έκδοση. Ο σχολιασμός και η ερμηνεία των αποτελεσμάτων παρουσιάζονται στο Κεφάλαιο 8.

7.2 Prompting και αποτελέσματα ανά plugin

1. WP Visitor Statistics (SQL injection) - CVE-2021-24750

Για την ανάλυση της patched έκδοσης (4.8) χρησιμοποιήθηκε το επόμενο prompt:

```
You are an expert Static Application Security Testing (SAST)
assistant.

SQL Injection vulnerabilities can occur when the refUrl
parameter is not properly validated and escaped before being
used in a SQL statement in the refDetails AJAX action.

Your task is to analyze the source code and determine whether
this specific weakness exists in the codebase.

Report:
1. Whether the vulnerability is present or not
2. Location: exact file path and line number(s)
3. Technical Analysis: why the code is secure or vulnerable
4. If vulnerable, describe the source-to-sink flow
```

Τα αποτελέσματα της ανάλυσης για κάθε μοντέλο παρουσιάζονται στον Πίνακα 7.1:

Πίνακας 7.1: *Αποτελέσματα αξιολόγησης για το plugin WP Visitor Statistics*

Μοντέλο	Ευάλωτη έκδοση (4.7)	Διορθωμένη έκδοση (4.8)
Qwen 3.6 Plus	FN	FP
GLM-5.1	TP	FP
Kimi K2.5	TP	FP
MiniMax M2.7	FN	TN

2. WP-UserOnline (Stored XSS) - CVE-2022-2941

Για την ανάλυση της patched έκδοσης (2.88.1) χρησιμοποιήθηκε το prompt που ακολουθεί:

```
You are an expert Static Application Security Testing (SAST)
assistant.

Stored Cross-Site Scripting vulnerabilities can occur when
the fields in the "Naming Conventions" section do not
properly sanitize user input, nor escape it on output.

Your task is to analyze the source code and determine whether
this specific weakness exists in the codebase.

Report:
1. Whether the vulnerability is present or not
2. Location: exact file path and line number(s)
3. Technical Analysis: why the code is secure or vulnerable
4. If vulnerable, describe the source-to-sink flow
```

Τα αποτελέσματα της ανάλυσης για κάθε μοντέλο παρουσιάζονται στον Πίνακα 7.2:

Πίνακας 7.2: *Αποτελέσματα αξιολόγησης για το plugin WP-UserOnline*

Μοντέλο	**Ευάλωτη έκδοση (2.88.0)**	**Διορθωμένη έκδοση (2.88.1)**
Qwen 3.6 Plus	FN	TN
GLM-5.1	FN	FP
Kimi K2.5	FN	TN
MiniMax M2.7	FN	FP

3. Perfect Survey (SQL injection) - CVE-2021-24762

Για την ανάλυση της patched έκδοσης (1.5.2) χρησιμοποιήθηκε το ακόλουθο prompt:

```
You are an expert Static Application Security Testing (SAST)
assistant.

SQL Injection vulnerabilities can occur when the question_id
GET parameter is not properly validated and escaped before
being used in a SQL statement in the get_question AJAX
action.

Your task is to analyze the source code and determine whether
this specific weakness exists in the codebase.

Report:
1. Whether the vulnerability is present or not
2. Location: exact file path and line number(s)
3. Technical Analysis: why the code is secure or vulnerable
4. If vulnerable, describe the source-to-sink flow
```

Τα αποτελέσματα της ανάλυσης για κάθε μοντέλο παρουσιάζονται στον Πίνακα 7.3:

Πίνακας 7.3: *Αποτελέσματα αξιολόγησης για το plugin Perfect Survey*

Μοντέλο	**Ευάλωτη έκδοση (1.5.1)**	**Διορθωμένη έκδοση (1.5.2)**
Qwen 3.6 Plus	FN	FP
GLM-5.1	FN	FP
Kimi K2.5	TP	FP
MiniMax M2.7	FN	TN

4. Backup Migration (RCE) - CVE-2023-6553

Για την ανάλυση της patched έκδοσης (1.3.8) χρησιμοποιήθηκε το prompt που ακολουθεί:

```
You are an expert Static Application Security Testing (SAST)
assistant.

Remote Code Execution vulnerabilities can occur when an
attacker is able to control the values passed to an include
statement, and subsequently leverage that to achieve remote
code execution, via the /includes/backup-heart.php file.

Your task is to analyze the source code and determine whether
this specific weakness exists in the codebase.

Report:
1. Whether the vulnerability is present or not
2. Location: exact file path and line number(s)
3. Technical Analysis: why the code is secure or vulnerable
4. If vulnerable, describe the source-to-sink flow
```

Τα αποτελέσματα της ανάλυσης για κάθε μοντέλο παρουσιάζονται στον Πίνακα 7.4:

Πίνακας 7.4: *Αποτελέσματα αξιολόγησης για το plugin Backup Migration*

Μοντέλο	**Ευάλωτη έκδοση (1.3.7)**	**Διορθωμένη έκδοση (1.3.8)**
Qwen 3.6 Plus	FN	FP
GLM-5.1	FN	FP
Kimi K2.5	FN	FP
MiniMax M2.7	FN	FP

5. Ecwid Ecommerce Shopping Cart (CSRF) - CVE-2022-2432

Για την ανάλυση της patched έκδοσης (6.10.24) χρησιμοποιήθηκε το ακόλουθο prompt:

```
You are an expert Static Application Security Testing (SAST)
assistant.
```

```
Cross-Site Request Forgery vulnerabilities can occur when the
ecwid_update_plugin_params function does not properly verify
a nonce before processing and updating plugin settings.

Your task is to analyze the source code and determine whether
this specific weakness exists in the codebase.

Report:
1. Whether the vulnerability is present or not
2. Location: exact file path and line number(s)
3. Technical Analysis: why the code is secure or vulnerable
4. If vulnerable, describe the source-to-sink flow
```

Τα αποτελέσματα της ανάλυσης για κάθε μοντέλο παρουσιάζονται στον Πίνακα 7.5:

Πίνακας 7.5: *Αποτελέσματα αξιολόγησης για το plugin Ecwid Ecommerce Shopping Cart*

Μοντέλο	**Ευάλωτη έκδοση (6.10.23)**	**Διορθωμένη έκδοση (6.10.24)**
Qwen 3.6 Plus	FN	TN
GLM-5.1	FN	TN
Kimi K2.5	FN	TN
MiniMax M2.7	FN	TN

6. **Variation Swatches for WooCommerce (Stored XSS) - CVE-2021-42367**

Για την ανάλυση της patched έκδοσης (1.3.8) χρησιμοποιήθηκε το επόμενο prompt:

```
You are an expert Static Application Security Testing (SAST)
assistant.

Stored Cross-Site Scripting vulnerabilities can occur when
the parameters found in the ~/includes/class-menu-page.php
file are not properly sanitized on input nor escaped on
output, and the tawcvs_save_settings function does not verify
the caller's permissions before saving settings.

Your task is to analyze the source code and determine whether
this specific weakness exists in the codebase.

Report:
1. Whether the vulnerability is present or not
2. Location: exact file path and line number(s)
3. Technical Analysis: why the code is secure or vulnerable
4. If vulnerable, describe the source-to-sink flow
```

Τα αποτελέσματα της ανάλυσης για κάθε μοντέλο παρουσιάζονται στον Πίνακα 7.6:

Πίνακας 7.6: *Αποτελέσματα αξιολόγησης για το plugin Variation Swatches for WooCommerce*

Μοντέλο	**Ευάλωτη έκδοση (2.1.1)**	**Διορθωμένη έκδοση (2.1.2)**
Qwen 3.6 Plus	TP	TN
GLM-5.1	TP	FP
Kimi K2.5	TP	TN
MiniMax M2.7	FN	FP

7. **WP HTML Mail (Stored XSS - CVE-2022-0218)**

Για την ανάλυση της patched έκδοσης (3.1) χρησιμοποιήθηκε το ακόλουθο prompt:

```
You are an expert Static Application Security Testing (SAST)
assistant.

Stored Cross-Site Scripting vulnerabilities can occur when
the /themesettings REST-API endpoint found in the
~/includes/class-template-designer.php file does not perform
a capability check, allowing unauthenticated attackers to
retrieve and overwrite theme settings and inject arbitrary
JavaScript that is later rendered to administrators.

Your task is to analyze the source code and determine whether
this specific weakness exists in the codebase.

Report:
1. Whether the vulnerability is present or not
2. Location: exact file path and line number(s)
3. Technical Analysis: why the code is secure or vulnerable
4. If vulnerable, describe the source-to-sink flow
```

Τα αποτελέσματα της ανάλυσης για κάθε μοντέλο παρουσιάζονται στον Πίνακα 7.7:

Πίνακας 7.7: *Αποτελέσματα αξιολόγησης για το plugin WP HTML Mail*

Μοντέλο	**Ευάλωτη έκδοση (3.0.9)**	**Διορθωμένη έκδοση (3.1)**
Qwen 3.6 Plus	FN	TN
GLM-5.1	TP	TN
Kimi K2.5	TP	TN
MiniMax M2.7	FN	TN

8. **PHP Everywhere (RCE) - CVE-2022-24663**

Για την ανάλυση της patched έκδοσης (3.0.0) χρησιμοποιήθηκε το prompt που ακολουθεί:

```
You are an expert Static Application Security Testing (SAST)
assistant.

Remote Code Execution vulnerabilities can occur when the
```

```
plugin registers a shortcode that executes arbitrary PHP code
and exposes it via the parse-media-shortcode AJAX action
without restricting which user roles are permitted to invoke
it, allowing any authenticated user — including subscribers —
to execute arbitrary PHP on the server.

Your task is to analyze the source code and determine whether
this specific weakness exists in the codebase.

Report:
1. Whether the vulnerability is present or not
2. Location: exact file path and line number(s)
3. Technical Analysis: why the code is secure or vulnerable
4. If vulnerable, describe the source-to-sink flow
```

Τα αποτελέσματα της ανάλυσης για κάθε μοντέλο παρουσιάζονται στον Πίνακα 7.8:

Πίνακας 7.8: *Αποτελέσματα αξιολόγησης για το plugin PHP Everywhere*

Μοντέλο	**Ευάλωτη έκδοση (2.0.3)**	**Διορθωμένη έκδοση (3.0.0)**
Qwen 3.6 Plus	TP	FP
GLM-5.1	TP	TN
Kimi K2.5	TP	TN
MiniMax M2.7	TP	FP

9. **Demon Image Annotation (CSRF) - CVE-2022-2864**

Για την ανάλυση της patched έκδοσης (4.8) χρησιμοποιήθηκε το επόμενο prompt:

```
You are an expert Static Application Security Testing (SAST)
assistant.

Cross-Site Request Forgery vulnerabilities can occur when the
settings handler in the ~/includes/settings.php file does not
validate a nonce before processing and persisting plugin
settings, allowing unauthenticated attackers to modify plugin
settings and inject malicious web scripts via a forged
request.

Your task is to analyze the source code and determine whether
this specific weakness exists in the codebase.

Report:
1. Whether the vulnerability is present or not
2. Location: exact file path and line number(s)
3. Technical Analysis: why the code is secure or vulnerable
4. If vulnerable, describe the source-to-sink flow
```

Τα αποτελέσματα της ανάλυσης για κάθε μοντέλο παρουσιάζονται στον Πίνακα 7.9:

Πίνακας 7.9: *Αποτελέσματα αξιολόγησης για το plugin Demon Image Annotation*

Μοντέλο	**Ευάλωτη έκδοση (4.7)**	**Διορθωμένη έκδοση (4.8)**
Qwen 3.6 Plus	FN	TN
GLM-5.1	FN	TN
Kimi K2.5	FN	TN
MiniMax M2.7	FN	TN

10. **Login/Signup Popup (CSRF) - CVE-2022-0215**

Για την ανάλυση της patched έκδοσης (2.2.20.2) χρησιμοποιήθηκε το ακόλουθο prompt:

```
You are an expert Static Application Security Testing (SAST)
assistant.

Cross-Site Request Forgery vulnerabilities can occur when the
save_settings function found in the ~/includes/xoo
framework/admin/class-xoo-admin-settings.php file does not
verify a nonce before processing and updating plugin
settings, making it possible for unauthenticated attackers to
update arbitrary site options — including creating an
administrative user account — via a forged request.

Your task is to analyze the source code and determine whether
this specific weakness exists in the codebase.

Report:
1. Whether the vulnerability is present or not
2. Location: exact file path and line number(s)
3. Technical Analysis: why the code is secure or vulnerable
4. If vulnerable, describe the source-to-sink flow
```

Πίνακας 7.10: *Αποτελέσματα αξιολόγησης για το plugin Login/Signup Popup*

Μοντέλο	**Ευάλωτη έκδοση (2.2.20.1)**	**Διορθωμένη έκδοση (2.2.20.2)**
Qwen 3.6 Plus	TP	FP
GLM-5.1	FN	FP
Kimi K2.5	TP	FP
MiniMax M2.7	FN	TN

7.3 Συγκεντρωτικά αποτελέσματα και καταγραφή μετρικών αξιολόγησης

Πίνακας 7.11: *Ταξινόμηση TP/FP/TN/FN ανά plugin και μοντέλο (Vuln = ευάλωτη έκδοση, Patch = διορθωμένη έκδοση)*

Plugin	**CVE**	**Τύπος**	**Qwen 3.6**	**Qwen 3.6**	**GLM-5.1**	**GLM-5.1**	**Kimi K2.5**	**Kimi K2.5**	**MiniMax M2.7**	**MiniMax M2.7**
			Vuln	Patch	Vuln	Patch	Vuln	Patch	Vuln	Patch
WP Visitor Statistics	CVE-2021-24750	SQLi.	**FN**	**FP**	**TP**	**FP**	**TP**	**FP**	**FN**	**TN**
WP-UserOnline	CVE-2022-2941	XSS.	**FN**	**TN**	**FN**	**FP**	**FN**	**TN**	**FN**	**FP**
Perfect Survey	CVE-2021-24762	SQLi	**FN**	**FP**	**FN**	**FP**	**TP**	**FP**	**FN**	**TN**
Backup Migration	CVE-2023-6553	RCE	**FN**	**FP**	**FN**	**FP**	**FN**	**FP**	**FN**	**FP**
Ecwid Shopping Cart	CVE-2022-2432	CSRF	**FN**	**TN**	**FN**	**TN**	**FN**	**TN**	**FN**	**TN**
Variation Swatches	CVE-2021-42367	XSS.	**TP**	**TN**	**TP**	**FP**	**TP**	**TN**	**FN**	**FP**
WP HTML Mail	CVE-2022-0218	XSS.	**FN**	**TN**	**TP**	**TN**	**TP**	**TN**	**FN**	**TN**
PHP Everywhere	CVE-2022-24663	RCE.	**TP**	**FP**	**TP**	**TN**	**TP**	**TN**	**TP**	**FP**
Demon Image Annotation	CVE-2022-2864	CSRF	**FN**	**TN**	**FN**	**TN**	**FN**	**TN**	**FN**	**TN**
Login/Signup Popup	CVE-2022-0215	CSRF	**TP**	**FP**	**FN**	**FP**	**TP**	**FP**	**FN**	**TN**

Πίνακας 7.12*: Συγκεντρωτικές μετρικές ανά μοντέλο (σύνολο 20 αναλύσεων/μοντέλο - 10 plugins × 2 εκδόσεις)*

Μοντέλο	**TP**	**FP**	**TN**	**FN**	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Hallucination Rate**
Qwen 3.6 Plus	3	5	5	7	40.0%	37.5%	30.0%	33.3%	50.0%
GLM-5.1	4	6	4	6	40.0%	40.0%	40.0%	40.0%	60.0%
Kimi K2.5	6	4	6	4	60.0%	60.0%	60.0%	60.0%	40.0%
MiniMax M2.7	1	4	6	9	35.0%	20.0%	10.0%	13.3%	40.0%

Κεφάλαιο 8ο: Ανάλυση και ερμηνεία αποτελεσμάτων

8.1 Συγκριτική αξιολόγηση μοντέλων

Το Kimi K2.5 εμφανίζει συνολικά την καλύτερη απόδοση ανάμεσα στα τέσσερα μοντέλα, αφενός έχοντας εντοπίσει έξι TP στις ευάλωτες εκδόσεις και αφετέρου έχοντας καταφέρει τον υψηλότερο αριθμό πετυχημένων εντοπισμών. Συγκεκριμένα, αναγνώρισε τόσο ευπάθειες με άμεσα ανιχνεύσιμη ροή δεδομένων όσο και περισσότερο σύνθετα σενάρια. Στις διορθωμένες εκδόσεις εντόπισε ορθά έξι TN, ενώ τα FP, τα οποία παρήγαγε αφορούν κυρίως plugins όπου η ευπάθεια ήταν δυσχερής ακόμη και στον πρωταρχικό εντοπισμό. Αξιοσημείωτο είναι ότι στα plugins, όπου η ροή των δεδομένων ήταν πιο ξεκάθαρη, το Kimi παρήγαγε TP στην ευάλωτη έκδοση και TN στη διορθωμένη. Αυτό καταδεικνύει συνέπεια τόσο στον εντοπισμό των ευπαθειών όσο και στην αναγνώριση των ασφαλών εκδόσεων. Επιπρόσθετα, οι απαντήσεις του είναι σταθερά περισσότερο οργανωμένες και τεχνικά πιο συνεπείς, καθώς παρουσιάζουν με σαφήνεια τη διαδρομή των δεδομένων μέσα στον κώδικα. Ενδεικτικά σημειώνεται πως ακόμη και στις περιπτώσεις FP, η αιτιολόγηση παραμένει συνεπής.

Το GLM-5.1 εμφανίζει τάση να επεκτείνει το εύρος της ανάλυσής του πέρα από το ζητούμενο. Συνολικά κατέγραψε τέσσερα TP και έξι FN στις ευάλωτες εκδόσεις, γεγονός που δείχνει ότι η ικανότητά του να εντοπίζει τις γνωστές ευπάθειες ήταν περιορισμένη και όχι σταθερή. Παράλληλα, στις διορθωμένες εκδόσεις κατέγραψε τέσσερα TN και έξι FP, παρουσιάζοντας τον υψηλότερο αριθμό false positives ανάμεσα στα μοντέλα. Το εύρημα αυτό υποδηλώνει ότι το μοντέλο συχνά συνέχιζε να αναφέρει την ύπαρξη ευπάθειας ακόμη και όταν το ζητούμενο CVE είχε διορθωθεί. Σε αρκετές περιπτώσεις η ανάλυσή του ήταν εκτενής και τεχνικά τεκμηριωμένη, ωστόσο εστίαζε σε γενικότερες αδυναμίες ή πιθανά προβλήματα του κώδικα, τα οποία δεν αντιστοιχούσαν στη συγκεκριμένη υπό διερεύνηση ευπάθεια. Η συμπεριφορά αυτή ενδέχεται να είναι επωφελής σε συνθήκες πραγματικού ελέγχου ασφαλείας, ωστόσο δημιουργεί προβλήματα στο πλαίσιο της αξιολόγησης του πειράματος.

Το Qwen 3.6 Plus δεν εμφανίζει ομοιόμορφη συμπεριφορά. Αποδίδει ικανοποιητικά σε ευπάθειες με πιο εμφανή δομή, αλλά δυσκολεύεται σε περιπτώσεις κατά τις οποίες απαιτείται βαθύτερη κατανόηση της λογικής του κώδικα. Στις εν λόγω περιπτώσεις κατέγραψε μεγάλο αριθμό FN. Στις διορθωμένες εκδόσεις εντόπισε ορθά πέντε TN, υποδηλώνοντας ότι όταν η ευπάθεια έχει διορθωθεί με τρόπο σαφή και άμεσο, το μοντέλο αναγνωρίζει την αλλαγή. Αξίζει να σημειωθεί ότι σε ορισμένες περιπτώσεις το Qwen αναγνώρισε ορθά τη διορθωμένη έκδοση ως ασφαλή, ακόμη και όταν δεν εντόπισε την ορθή ευπάθεια στην ευάλωτη έκδοση. Αυτό καταδεικνύει ότι η επιτυχία στην ανάλυση της διορθωμένης έκδοσης δεν συνδέεται πάντοτε με την επιτυχία στην ανάλυση της ευπαθούς έκδοσης. Οι δύο διαδικασίες, όπως φαίνεται, απαιτούν διαφορετικές δυνατότητες από το μοντέλο.

Το MiniMax M2.7 παρουσιάζει τη χαμηλότερη συνολική απόδοση στις ευάλωτες εκδόσεις, καταγράφοντας μόλις ένα TP και εννιά FN συνολικά. Σε κάποιες περιπτώσεις ανέφερε εντελώς

διαφορετική ευπάθεια από αυτή που τεκμηριώνεται στο CVE. Το γεγονός αυτό υποδεικνύει ένα ουσιαστικό πρόβλημα, καθώς το μοντέλο φαίνεται πως βασίζεται σε γενικά μοτίβα αναγνώρισης και όχι στη συγκεκριμένη λογική του κώδικα. Παρά τη χαμηλή επίδοση στις ευπαθείς εκδόσεις, το MiniMax εμφάνισε συγκριτικά καλύτερη συμπεριφορά στις διορθωμένες εκδόσεις, καταγράφοντας έξι TN, αριθμό αντίστοιχο με εκείνον του Kimi K2.5, και τον υψηλότερο στο σύνολο του πειράματος. Η τάση αυτή, όπως φαίνεται, συνδέεται με μια περισσότερο επιφυλακτική στάση του μοντέλου κατά την αξιολόγηση πιθανών ευπαθειών. Το MiniMax αποφεύγει συχνά να καταλήξει σε θετικό εντοπισμό όταν τα διαθέσιμα στοιχεία στον κώδικα δεν είναι επαρκώς ισχυρά ή ξεκάθαρα [134]. Η εν λόγω συμπεριφορά ευνοεί την ορθή αναγνώριση των διορθωμένων εκδόσεων, αλλά παράλληλα οδηγεί συχνά σε αποτυχία εντοπισμού των πραγματικών ευπαθειών στις ευπαθείς εκδόσεις.

8.2 Συνολική ανάλυση πειραματικών ευρημάτων

Πέρα από την αξιολόγηση ανά μοντέλο, η συνολική ανάλυση αναδεικνύει ορισμένα μοτίβα, τα οποία αφορούν το σύνολο του πειράματος.

Ένας σημαντικός παράγοντας σχετικά με την απόδοση των μοντέλων φαίνεται πως είναι η στατική ορατότητα της ευπάθειας, δηλαδή το κατά πόσο η ευπάθεια εμφανίζεται μέσω σαφών και άμεσα αναγνωρίσιμων μοτίβων στον κώδικα. Ωστόσο, η σχέση αυτή δεν είναι απόλυτη, καθώς η απόδοση διαφοροποιείται τόσο ανά τύπο ευπάθειας όσο και ανά μοντέλο. Στις ευπάθειες XSS, το GLM-5.1 και το Kimi K2.5 πέτυχαν 2/3 TP, ενώ το Qwen 3.6 Plus πέτυχε 1/3 και το MiniMax M2.7 0/3. Στις RCE ευπάθειες, όλα τα μοντέλα εντόπισαν ορθά μία από τις δύο περιπτώσεις, καταγράφοντας 1/2 TP. Στις SQL Injection, το Kimi K2.5 ήταν το μόνο μοντέλο που αναγνώρισε ορθά και τις δύο περιπτώσεις, ενώ το GLM-5.1 εντόπισε μία από τις δύο και τα Qwen 3.6 Plus και MiniMax M2.7 απέτυχαν και στις δύο. Εν τέλει, στις CSRF κανένα μοντέλο δεν ξεπέρασε το 1/3 TP.

Αναφορικά με τις ευπάθειες CSRF, η συστηματική αδυναμία των μοντέλων στον εντοπισμό τους δεν είναι τυχαία. Η συγκεκριμένη κατηγορία ευπάθειας απαιτεί κατανόηση όχι μόνο του τι κάνει ο κώδικας, αλλά και του τι απουσιάζει από αυτόν. Ειδικότερα, το μοντέλο θα πρέπει να αναγνωρίσει αν η έλλειψη κάποιου μηχανισμού ελέγχου είναι δυνατό να οδηγήσει σε εκμεταλλεύσιμη ευπάθεια υπό συγκεκριμένες συνθήκες. Αυτό είναι ιδιαίτερα δύσκολο σε διαδικασίες της στατικής ανάλυσης [53].

Ένα δεύτερο εύρημα αφορά σε περιπτώσεις κατά τις οποίες τα μοντέλα αναγνώρισαν τον ορθό τύπο ευπάθειας, αλλά σε διαφορετικό σημείο του κώδικα από αυτό, το οποίο περιγράφεται στο CVE. Μολονότι οι εν λόγω αναλύσεις ήταν τεχνικά έγκυρες, αξιολογήθηκαν ως FN, επειδή δεν αντιστοιχούσαν στη συγκεκριμένη υπό διερεύνηση ευπάθεια. Το φαινόμενο αυτό αναδεικνύει έναν σημαντικό περιορισμό: τα μοντέλα εντοπίζουν τα πιο εμφανή σε στατικό επίπεδο σημεία αδυναμίας, τα οποία δεν αντιστοιχούν πάντοτε στην πραγματική τεκμηριωμένη ευπάθεια. Παράλληλα, αναδεικνύεται ένα μεθοδολογικό ζήτημα. Η αυστηρή αξιολόγηση, με βάση τη συγκεκριμένη ροή του CVE, συνιστά την ορθή επιλογή για λόγους επιστημονικής συνέπειας. Ωστόσο, η προσέγγιση αυτή μπορεί να υποτιμά μια πραγματική δυνατότητα των μοντέλων, δηλαδή τον εντοπισμό των υφιστάμενων ευπαθειών, οι οποίες δεν αντιστοιχούν στη

συγκεκριμένη καταγεγραμμένη περίπτωση. Το σημείο αυτό αναφέρεται ως περιορισμός της μεθοδολογίας.

Μια τρίτη παρατήρηση αφορά στη διαφοροποίηση ανάμεσα στην ανάλυση της ευάλωτης και της διορθωμένης έκδοσης. Τα αποτελέσματα καταδεικνύουν ότι η απόδοση ενός μοντέλου στην ευπαθή έκδοση δεν συνδέεται απαραίτητα με την απόδοσή του στη διορθωμένη έκδοση, γεγονός που υποδηλώνει ότι οι δύο διαδικασίες απαιτούν διαφορετικού τύπου δυνατότητες. Στην ανάλυση της ευπαθούς έκδοσης, το μοντέλο καλείται να εντοπίσει πιθανή διαδρομή εκμετάλλευσης μέσα στον κώδικα δίχως συγκεκριμένο σημείο αναφοράς. Αντίθετα, στη διορθωμένη έκδοση η διαδικασία επικεντρώνεται στην επιβεβαίωση της παρουσίας ή της απουσίας συγκεκριμένων μηχανισμών προστασίας. Η διάκριση αυτή αντανακλάται και στον πειραματικό σχεδιασμό, όπου χρησιμοποιήθηκε γενικό prompt για τις ευπαθείς εκδόσεις και CVE στοχευμένο prompt για τις διορθωμένες, επιτρέποντας την αξιολόγηση διαφορετικών πτυχών της ικανότητας των μοντέλων στην ανάλυση κώδικα.

Τέλος, η περίπτωση του plugin Backup Migration συνιστά την περισσότερο διαφωτιστική οριακή περίπτωση του πειράματος. Η ομόφωνη αποτυχία των τεσσάρων μοντέλων και στις δύο εκδόσεις αναδεικνύει τόσο τα όρια των μοντέλων όσο και την πολυπλοκότητα της αξιολόγησης. Στην ευάλωτη έκδοση, η ευπάθεια δεν είναι άμεσα εμφανής στον κώδικα και απαιτεί βαθύτερη κατανόηση του τρόπου με τον οποίο τα εξωτερικά δεδομένα επηρεάζουν τη διαδικασία της εκτέλεσης κώδικα. Αυτό δυσχεραίνει σημαντικά τα μοντέλα, επειδή δεν υφίστανται σαφή μοτίβα που να υποδεικνύουν εύκολα την παρουσία της ευπάθειας. Στη διορθωμένη έκδοση, τα μοντέλα εντόπισαν τεχνικά έγκυρες ατέλειες στη νέα υλοποίηση προστασίας. Ωστόσο, οι περιπτώσεις αυτές αξιολογούνται ως FP, καθώς η Wordfence χαρακτηρίζει τη συγκεκριμένη έκδοση ως διορθωμένη.

8.3 Ερμηνεία αποτελεσμάτων βάσει του θεωρητικού πλαισίου

Τα κεφάλαια 4 και 5 της παρούσας μελέτης οικοδομούν ένα θεωρητικό πλαίσιο στο οποίο τα LLMs, χάρη στον μηχανισμό προσοχής και τις reasoning δυνατότητές τους, μπορούν να κατανοούν τον κώδικα πέρα από την απλή αναγνώριση μοτίβων. Ειδικότερα, αποκτούν τη δυνατότητα να παρακολουθούν ροές δεδομένων, να συνδέουν σημεία εισόδου με σημεία εκμετάλλευσης, και να εξάγουν τεχνικά συμπεράσματα από τα συμφραζόμενα. Επιπλέον, η agentic αρχιτεκτονική του OpenCode παρέχει εργαλεία πλοήγησης, ανάγνωσης αρχείων και αναζήτησης, επιτρέποντας στο μοντέλο να έχει πιο ολοκληρωμένη εικόνα του συνολικού πηγαίου κώδικα κατά την ανάλυση. Τα αποτελέσματα του πειράματος επιβεβαιώνουν εν μέρει τις δυνατότητες αυτές, αλλά αναδεικνύουν και ένα σαφές χάσμα ανάμεσα στη θεωρητική δυνατότητα και την πρακτική απόδοση.

Όσον αφορά στο agentic πλαίσιο, τα μοντέλα χρησιμοποίησαν εκτενώς τα εργαλεία που τους παρείχε το OpenCode, καθώς διάβασαν πολλαπλά αρχεία, ακολούθησαν αλυσίδες εξαρτήσεων και αναζήτησαν ευπάθειες σε ολόκληρο τον πηγαίο κώδικα. Παρόλα αυτά, η απόδοση στις ευπάθειες, οι οποίες απαιτούν βαθύτερη ανάλυση, παρέμεινε χαμηλή. Το εύρημα αυτό καταδεικνύει ότι το κύριο πρόβλημα δεν είναι η πρόσβαση στον κώδικα, αλλά η ικανότητα των μοντέλων να κατανοούν τη συνολική ροή των δεδομένων μέσα σε πληθώρα αρχείων και

συναρτήσεων. Οι ευπάθειες που εντοπίστηκαν πιο εύκολα ήταν αυτές που εκδηλώνονται μέσα σε ένα αρχείο ή σε μια συνάρτηση, ενώ οι ευπάθειες που απαιτούσαν να ακολουθηθεί η ροή διαμέσου πολλών αρχείων ήταν συστηματικά πιο δύσκολες. Το εύρημα αυτό συμφωνεί με ένα γνωστό περιορισμό, ο οποίος αναφέρεται και στη βιβλιογραφία. Τα LLMs εξακολουθούν να δυσκολεύονται στην παρακολούθηση της ροής δεδομένων ανάμεσα σε πολλά αρχεία και συναρτήσεις, ακόμη και όταν υποστηρίζονται από πιο εξελιγμένα εργαλεία και πλαίσια λειτουργίας.

Ένας επιπλέον παράγοντας που εξηγεί μέρος των αποτυχιών είναι το φαινόμενο των hallucinations. Αρκετά FP στις διορθωμένες εκδόσεις δεν φαίνεται να οφείλονται σε εσφαλμένη ανάλυση υφιστάμενων στοιχείων αλλά σε αναφορά κώδικα που δεν υπάρχει ή σε παρανόηση της λειτουργίας υφιστάμενων συναρτήσεων. Το μοντέλο καταλήγει στην αναγνώριση της ευπάθειας επειδή στηρίζεται σε μοτίβα που έχει μάθει κατά την εκπαίδευσή του και τα εφαρμόζει ακόμη και σε περιπτώσεις, στις οποίες δεν αντιστοιχούν πλήρως στον συγκεκριμένο κώδικα. Αξίζει επίσης να επισημανθεί ότι τα LLMs δεν παράγουν πάντοτε ακριβώς την ίδια απάντηση για το ίδιο ερώτημα. Για τον λόγο αυτό, κάθε ανάλυση στο πείραμα εκτελέστηκε μία φορά και μια διαφορετική εκτέλεση του ίδιου μοντέλου στο ίδιο plugin θα μπορούσε ενδεχομένως να οδηγήσει σε διαφορετικό αποτέλεσμα. Οι μετρικές, οι οποίες παρουσιάζονται, αντιπροσωπεύουν επομένως μία εκτέλεση και όχι μία σταθερή συμπεριφορά.

Καταληκτικά, είναι σημαντικό να τοποθετηθούν τα ανωτέρο συμπεράσματα στο ορθό πλαίσιο. Τα τέσσερα μοντέλα που αξιολογήθηκαν - Qwen 3.6 Plus, GLM-5.1, Kimi K2.5 και MiniMax M2.7 - είναι ικανά μοντέλα γενικής χρήσης, αλλά δεν αντιπροσωπεύουν την αιχμή των δυνατοτήτων που είναι διαθέσιμες στις ημέρες μας. Τα μοντέλα με σημαντικά μεγαλύτερες reasoning δυνατότητες και με εξειδικευμένη εκπαίδευση σε ανάλυση κώδικα ενδέχεται να αποδίδουν διαφορετικά στο ίδιο σενάριο. Ως εκ τούτου, τα συμπεράσματα της παρούσας μελέτης δεν αποτελούν καθολική κρίση για τις δυνατότητες των LLMs στη SAST, αλλά αξιολόγηση τεσσάρων μοντέλων, ενταγμένων σε ένα agentic framework, σε ένα σύνολο πραγματικών WordPress ευπαθειών. Αυτή ακριβώς η συγκεκριμένη διαμόρφωση είναι και η συνεισφορά της εργασίας - και τα αποτελέσματά της ανοίγουν ένα ερευνητικό ερώτημα για το κατά πόσο μοντέλα με ισχυρότερες reasoning δυνατότητες ή με εξειδικευμένη εκπαίδευση θα μπορούσαν να γεφυρώσουν το χάσμα που αναδείχθηκε.

Κεφάλαιο 9ο: Μελλοντικές επεκτάσεις

Τα αποτελέσματα της παρούσας εργασίας αναδεικνύουν τόσο τις δυνατότητες όσο και τους περιορισμούς των LLM-based agentic συστημάτων στην ανίχνευση ευπαθειών. Οι περιορισμοί αυτοί, ωστόσο, υποδεικνύουν συγκεκριμένες κατευθύνσεις για μελλοντική έρευνα που θα μπορούσαν να εμβαθύνουν και να επεκτείνουν τα ευρήματα της εργασίας.

Για τη μελλοντική έρευνα μια πρώτη κατεύθυνση συνιστά η αξιολόγηση ενός μεγαλύτερου και περισσότερο διαφοροποιημένου συνόλου μοντέλων. Τα τέσσερα μοντέλα τα οποία χρησιμοποιήθηκαν στην παρούσα εργασία αποτελούν ισχυρά agentic LLMs, ωστόσο δεν αποτελούν τα κορυφαία μοντέλα της γενιάς μας. Συνεπώς, παραμένει το ερώτημα κατά πόσο μοντέλα όπως τα GPT, Claude ή DeepSeek, θα μπορούσαν να επιτύχουν ποιοτικότερη απόδοση, ιδιαίτερα σε ευπάθειες που απαιτούν βαθύτερη κατανόηση της λογικής του κώδικα και παρακολούθηση της ροής δεδομένων ανάμεσα σε πολλά αρχεία και συναρτήσεις. Η επανάληψη του ίδιου πειραματικού σχεδιασμού με μεγαλύτερο αριθμό και διαφορετικές κατηγορίες μοντέλων θα επέτρεπε ομολογουμένως μια πληρέστερη αξιολόγηση των πραγματικών δυνατοτήτων της προσέγγισης.

Επιπρόσθετα, εξίσου σημαντική θεωρείται η διεύρυνση του πειραματικού. Η παρούσα εργασία εξέτασε 10 plugins με τέσσερις τύπους ευπαθειών, αποτελώντας ένα δομημένο αλλά περιορισμένο σύνολο αξιολόγησης. Η επέκταση σε μεγαλύτερο αριθμό plugins και σε περισσότερους τύπους ευπαθειών, θα επέτρεπε την εξαγωγή περισσότερο γενικευμένων συμπερασμάτων για την αποτελεσματικότητα των LLM-based agentic συστημάτων στη SAST.

Καταληκτικά, μια ακόμη ενδιαφέρουσα κατεύθυνση αποτελεί η μελέτη της μεταβλητότητας στη συμπεριφορά των μοντέλων. Κατά την υλοποίηση του πειράματος, κάθε ανάλυση εκτελέστηκε μία φορά, με αποτέλεσμα οι μετρήσεις να βασίζονται σε μία μόνο εκτέλεση και όχι απαραίτητα σε σταθερή συμπεριφορά. Η μελλοντική έρευνα θα μπορούσε να περιλαμβάνει πολλαπλές επαναλήψεις ανά μοντέλο και plugin, έτσι ώστε να μετρηθεί η διακύμανση των αποτελεσμάτων και να αξιολογηθεί η συνέπεια της συμπεριφοράς κάθε μοντέλου. Η διάσταση αυτή είναι ιδιαίτερα σημαντική για την αξιοπιστία συστημάτων ανάλυσης ασφάλειας σε πραγματικές συνθήκες.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] M. Aydos, Ç. Aldan, E. Coşkun, and A. Soydan, “Security testing of web applications: A systematic mapping of the literature,” *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 9, pp. 6775–6792, 2022.

[2] B. Mweu and J. Ndia, “Static Analysis Techniques for Secure Software: A Systematic Review,” *J. Cyber Secur.*, vol. 7, no. 1, pp. 417–437, 2025, doi: 10.32604/jcs.2025.071765.

[3] C. Fang *et al.*, “Large Language Models for Code Analysis: Do LLMs Really Do Their Job?,” in *33rd USENIX Security Symposium (USENIX Security 24)*, Philadelphia, PA: USENIX Association, Aug. 2024, pp. 829–846. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/fang

[4] J. Liu *et al.*, “Large Language Model-Based Agents for Software Engineering: A Survey,” *ACM Trans Softw Eng Methodol*, Mar. 2026, doi: 10.1145/3796507.

[5] J. Lin, M. Sayagh, and A. E. Hassan, “The Co-evolution of the WordPress Platform and Its Plugins,” *ACM Trans Softw Eng Methodol*, vol. 32, no. 1, Feb. 2023, doi: 10.1145/3533700.

[6] S. Qadir, E. Waheed, A. Khanum, and S. Jehan, “Comparative evaluation of approaches & tools for effective security testing of Web applications,” *PeerJ Comput. Sci.*, vol. 11, p. e2821, Apr. 2025, doi: 10.7717/peerj-cs.2821.

[7] Z. D. Wadhams, C. Izurieta, and A. M. Reinhold, “Barriers to Using Static Application Security Testing (SAST) Tools: A Literature Review,” in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering Workshops*, Sacramento CA USA: ACM, Oct. 2024, pp. 161–166. doi: 10.1145/3691621.3694947.

[8] A. M. C. Balsa, “Scan Metrics for Static Application Security Testing (SAST)”.

[9] L. Dencheva, “Comparative analysis of Static application security testing (SAST) and Dynamic application security testing (DAST) by using open-source web application penetration testing tools”.

[10] F. Mateo Tudela, J.-R. Bermejo Higuera, J. Bermejo Higuera, J.-A. Sicilia Montalvo, and M. I. Argyros, “On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications,” *Appl. Sci.*, vol. 10, no. 24, p. 9119, Dec. 2020, doi: 10.3390/app10249119.

[11] J. D. Pereira, N. Ivaki, and M. Vieira, “Characterizing Buffer Overflow Vulnerabilities in Large C/C++ Projects,” *IEEE Access*, vol. 9, pp. 142879–142892, 2021, doi: 10.1109/ACCESS.2021.3120349.

[12] M. Ferreira *et al.*, “Efficient Static Vulnerability Analysis for JavaScript with Multiversion Dependency Graphs,” *Proc. ACM Program. Lang.*, vol. 8, no. PLDI, pp. 417–441, Jun. 2024, doi: 10.1145/3656394.

[13] A. Nguyen-Duc, M. V. Do, Q. L. Hong, and K. N. Khac, “On the combination of static analysis for software security assessment -- a case study of an open-source e-government project,” 2021, *arXiv*. doi: 10.48550/ARXIV.2103.08010.

[14] D. B. Cruz, J. R. Almeida, and J. L. Oliveira, “Open Source Solutions for Vulnerability Assessment: A Comparative Analysis,” *IEEE Access*, vol. 11, pp. 100234–100255, 2023, doi: 10.1109/ACCESS.2023.3315595.

[15] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, “Why don’t software developers use static analysis tools to find bugs?,” in *2013 35th International Conference on Software Engineering (ICSE)*, San Francisco, CA, USA: IEEE, May 2013, pp. 672–681. doi: 10.1109/ICSE.2013.6606613.

[16] Z. Li, D. Zou, J. Tang, Z. Zhang, M. Sun, and H. Jin, “A Comparative Study of Deep Learning-Based Vulnerability Detection System,” *IEEE Access*, vol. 7, pp. 103184–103197, 2019, doi: 10.1109/ACCESS.2019.2930578.

[17] A. Adhikari and P. Kulkarni, “Survey of techniques to detect common weaknesses in program binaries,” *Cyber Secur. Appl.*, vol. 3, p. 100061, Dec. 2025, doi: 10.1016/j.csa.2024.100061.

[18] P. Thomson, “Static Analysis: An Introduction: The fundamental challenge of

software engineering is one of complexity.," *Queue*, vol. 19, no. 4, pp. 29–41, Aug. 2021, doi: 10.1145/3487019.3487021.
[19] A. W. Marashdih, Z. F. Zaaba, and K. Suwais, "An Enhanced Static Taint Analysis Approach to Detect Input Validation Vulnerability," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 35, no. 2, pp. 682–701, Feb. 2023, doi: 10.1016/j.jksuci.2023.01.009.
[20] Y. Shoshitaishvili *et al.*, "SOK: (State of) The Art of War: Offensive Techniques in Binary Analysis," in *2016 IEEE Symposium on Security and Privacy (SP)*, San Jose, CA: IEEE, May 2016, pp. 138–157. doi: 10.1109/SP.2016.17.
[21] A. Kurniawan, B. S. Abbas, A. Trisetyarso, and S. M. Isa, "Static Taint Analysis Traversal with Object Oriented Component for Web File Injection Vulnerability Pattern Detection," *Procedia Comput. Sci.*, vol. 135, pp. 596–605, 2018, doi: 10.1016/j.procs.2018.08.227.
[22] G. Bennett, T. Hall, E. Winter, and S. Counsell, "Semgrep*: Improving the Limited Performance of Static Application Security Testing (SAST) Tools," in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, Salerno Italy: ACM, Jun. 2024, pp. 614–623. doi: 10.1145/3661167.3661262.
[23] J. Wang, Y. Huang, S. Wang, and Q. Wang, "Find bugs in static bug finders," in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, Virtual Event: ACM, May 2022, pp. 516–527. doi: 10.1145/3524610.3527899.
[24] T. Brito *et al.*, "Study of JavaScript Static Analysis Tools for Vulnerability Detection in Node.js Packages," *IEEE Trans. Reliab.*, vol. 72, no. 4, pp. 1324–1339, Dec. 2023, doi: 10.1109/TR.2023.3286301.
[25] D. U. Bhasutkar Mahesh and Nilesh R Ware, "Evolving Trends in Web Application Vulnerabilities: A Comparative Study of OWASP Top 10 2017 and OWASP Top 10 2021." International Journal of Engineering Technology and Management Sciences, Nov. 2023. doi: 10.46647/ijetms.2023.v07i06.038.
[26] J. Li and H. Li, "Evolution of Application Security based on OWASP Top 10 and CWE/SANS Top 25 with Predictions for the 2025 OWASP Top 10," in *2025 International Conference on Inventive Computation Technologies (ICICT)*, Kirtipur, Nepal: IEEE, Apr. 2025, pp. 1178–1183. doi: 10.1109/ICICT64420.2025.11004742.
[27] P. Sane, "Is the OWASP Top 10 List Comprehensive Enough for Writing Secure Code?," in *Proceedings of the 2020 International Conference on Big Data in Management*, Manchester United Kingdom: ACM, May 2020, pp. 58–61. doi: 10.1145/3437075.3437089.
[28] *https://owasp.org/Top10/2025/*.
[29] P. Mell and A. Gueye, "A Suite of Metrics for Calculating the Most Significant Security Relevant Software Flaw Types," Jun. 15, 2020, *arXiv*: arXiv:2006.08524. doi: 10.48550/arXiv.2006.08524.
[30] E. Aghaei, W. Shadid, and E. Al-Shaer, "ThreatZoom: CVE2CWE using Hierarchical Neural Network," vol. 335, 2020, pp. 23–41. doi: 10.1007/978-3-030-63086-7_2.
[31] Mohammed A M Oudah and Mohd Fadzli Marhusin, "SQL Injection Detection using Machine Learning: A Review," *Malays. J. Sci. Health Technol.*, vol. 10, no. 1, pp. 39–49, Apr. 2024, doi: 10.33102/mjosht.v10i1.368.
[32] D. Aryal and A. Shakya, *A Taxonomy of SQL Injection Defense Techniques*. 2011.
[33] A. Sadeghian, M. Zamani, and A. A. Manaf, "A taxonomy of SQL injection detection and prevention techniques," in *2013 international conference on informatics and creative multimedia*, IEEE, 2013, pp. 53–56.
[34] W. G. Halfond, J. Viegas, and A. Orso, "A classification of SQL injection attacks and countermeasures," 2006.
[35] M. Al Rubaiei, T. Al Yarubi, M. Al Saadi, and B. Kumar, "SQLIA Detection and Prevention Techniques," in *2020 9th International Conference System Modeling and Advancement in Research Trends (SMART)*, Moradabad, India: IEEE, Dec. 2020, pp. 115–121. doi: 10.1109/SMART50582.2020.9336795.
[36] T. Muhammad and H. Ghafory, "Sql injection attack detection using machine learning algorithm," *Mesopotamian J. Cybersecurity*, vol. 2022, pp. 5–17, 2022.
[37] J. Clarke-Salt, *SQL injection attacks and defense*. Elsevier, 2009.

[38] I. Hydara, A. B. M. Sultan, H. Zulzalil, and N. Admodisastro, “Current state of research on cross-site scripting (XSS)–A systematic literature review,” *Inf. Softw. Technol.*, vol. 58, pp. 170–186, 2015.
[39] A. Shrivastava, S. Choudhary, and A. Kumar, “XSS vulnerability assessment and prevention in web application,” in *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*, IEEE, 2016, pp. 850–853.
[40] M. Alsaffar *et al.*, “Detection of web cross-site scripting (xss) attacks,” *Electronics*, vol. 11, no. 14, p. 2212, 2022.
[41] S. Van Acker, N. Nikiforakis, L. Desmet, W. Joosen, and F. Piessens, “FlashOver: Automated discovery of cross-site scripting vulnerabilities in rich internet applications,” in *Proceedings of the 7th ACM symposium on information, computer and communications security*, 2012, pp. 12–13.
[42] M. Liu, B. Zhang, W. Chen, and X. Zhang, “A survey of exploitation and detection methods of XSS vulnerabilities,” *IEEE Access*, vol. 7, pp. 182004–182016, 2019.
[43] X. Tan, Y. Xu, T. Wu, and B. Li, “Detection of reflected XSS vulnerabilities based on paths-attention method,” *Appl. Sci.*, vol. 13, no. 13, p. 7895, 2023.
[44] S. Bensalim, D. Klein, T. Barber, and M. Johns, “Talking about my generation: Targeted dom-based xss exploit generation using dynamic data flow analysis,” in *Proceedings of the 14th European Workshop on Systems Security*, 2021, pp. 27–33.
[45] S. Gupta and B. B. Gupta, “Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art,” *Int. J. Syst. Assur. Eng. Manag.*, vol. 8, no. Suppl 1, pp. 512–530, 2017.
[46] B. B. Gupta, S. Gupta, S. Gangwar, M. Kumar, and P. K. Meena, “Cross-site scripting (XSS) abuse and defense: exploitation on several testing bed environments and its defense,” *J. Inf. Priv. Secur.*, vol. 11, no. 2, pp. 118–136, 2015.
[47] Y. Zheng and X. Zhang, “Path sensitive static analysis of web applications for remote code execution vulnerability detection,” in *2013 35th International Conference on Software Engineering (ICSE)*, IEEE, 2013, pp. 652–661.
[48] E. Sun, J. Han, Y. Li, and C. Huang, “A packet content-oriented remote code execution attack payload detection model,” *Future Internet*, vol. 16, no. 7, p. 235, 2024.
[49] S. Biswas, M. Sohel, M. M. Sajal, T. Afrin, T. Bhuiyan, and M. M. Hassan, “A study on remote code execution vulnerability in web applications,” in *International conference on cyber security and computer science (ICONCS 2018)*, 2018, pp. 50–57.
[50] S. Bier, B. Fajardo, O. Ezeadum, G. Guzman, K. Z. Sultana, and V. Anu, “Mitigating Remote Code Execution Vulnerabilities: A Study on Tomcat and Android Security Updates,” in *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, Toronto, ON, Canada: IEEE, Apr. 2021, pp. 1–6. doi: 10.1109/IEMTRONICS52119.2021.9422666.
[51] G. Beltrano, C. Greco, M. Ianni, and G. Fortino, “Deep learning-based detection of csrf vulnerabilities in web applications,” in *2023 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, IEEE, 2023, pp. 0916–0920.
[52] S. Calzavara, M. Conti, R. Focardi, A. Rabitti, and G. Tolomei, “Mitch: A Machine Learning Approach to the Black-Box Detection of CSRF Vulnerabilities,” in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, Stockholm, Sweden: IEEE, Jun. 2019, pp. 528–543. doi: 10.1109/EuroSP.2019.00045.
[53] G. Pellegrino, M. Johns, S. Koch, M. Backes, and C. Rossow, “Deemon: Detecting CSRF with Dynamic Analysis and Property Graphs,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, Dallas Texas USA: ACM, Oct. 2017, pp. 1757–1771. doi: 10.1145/3133956.3133959.
[54] R. D. Kombade and B. B. Meshram, “CSRF Vulnerabilities and Defensive Techniques,” *Int. J. Comput. Netw. Inf. Secur.*, vol. 4, no. 1, pp. 31–37, Feb. 2012.
[55] J. Sun *et al.*, “Generating Informative CVE Description From ExploitDB Posts by Extractive Summarization,” 2021, *arXiv*. doi: 10.48550/ARXIV.2101.01431.

[56] “the-wordfence-2020-wordpress-threat-report.” [Online]. Available: https://www.wordfence.com/blog/2021/01/the-wordfence-2020-wordpress-threat-report/
[57] [Online]. Available: https://www.wordfence.com/threat-intel/vulnerabilities
[58] A. Vaswani *et al.*, “Attention Is All You Need,” Aug. 02, 2023, *arXiv*: arXiv:1706.03762. doi: 10.48550/arXiv.1706.03762.
[59] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, “Efficient Streaming Language Models with Attention Sinks,” Apr. 07, 2024, *arXiv*: arXiv:2309.17453. doi: 10.48550/arXiv.2309.17453.
[60] T. Ruan and S. Zhang, “Towards understanding how attention mechanism works in deep learning,” Dec. 24, 2024, *arXiv*: arXiv:2412.18288. doi: 10.48550/arXiv.2412.18288.
[61] M. Adnan and C. C. N. Kuhn, “Adaptive Hierarchical Evaluation of LLMs and SAST tools for CWE Prediction in Python,” Jan. 04, 2026, *arXiv*: arXiv:2601.01320. doi: 10.48550/arXiv.2601.01320.
[62] A. Bruno, P. L. Mazzeo, A. Chetouani, M. Tliba, and M. A. Kerkouri, “Insights into Classifying and Mitigating LLMs’ Hallucinations,” Nov. 14, 2023, *arXiv*: arXiv:2311.08117. doi: 10.48550/arXiv.2311.08117.
[63] A. Mostafa, R. A. Nahid, and S. Mulder, “How Different Tokenization Algorithms Impact LLMs and Transformer Models for Binary Code Analysis,” in *Proceedings 2025 Workshop on Binary Analysis Research*, 2025. doi: 10.14722/bar.2025.23013.
[64] R.-M. Karampatsis, H. Babii, R. Robbes, C. Sutton, and A. Janes, “Big Code != Big Vocabulary: Open-Vocabulary Models for Source Code,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, Jun. 2020, pp. 1073–1085. doi: 10.1145/3377811.3380342.
[65] M. Chen *et al.*, “Evaluating Large Language Models Trained on Code,” Jul. 14, 2021, *arXiv*: arXiv:2107.03374. doi: 10.48550/arXiv.2107.03374.
[66] Z. Li *et al.*, “VulDeePecker: A Deep Learning-Based System for Vulnerability Detection,” in *Proceedings 2018 Network and Distributed System Security Symposium*, 2018. doi: 10.14722/ndss.2018.23158.
[67] N. F. Liu *et al.*, “Lost in the Middle: How Language Models Use Long Contexts,” Nov. 20, 2023, *arXiv*: arXiv:2307.03172. doi: 10.48550/arXiv.2307.03172.
[68] Z. Wang, G. Li, J. Li, Y. Dong, Y. Xiong, and Z. Jin, “Line-level Semantic Structure Learning for Code Vulnerability Detection,” Nov. 08, 2024, *arXiv*: arXiv:2407.18877. doi: 10.48550/arXiv.2407.18877.
[69] S. Rai, R. C. Belwal, and A. Gupta, “Effect of Identifier Tokenization on Automatic Source Code Documentation,” *Arab. J. Sci. Eng.*, vol. 47, no. 2, pp. 2141–2157, Feb. 2022, doi: 10.1007/s13369-021-06149-7.
[70] A. Patil and A. Jadon, “Advancing Reasoning in Large Language Models: Promising Methods and Approaches,” May 28, 2025, *arXiv*: arXiv:2502.03671. doi: 10.48550/arXiv.2502.03671.
[71] C. Laneve, A. Spanò, D. Ressi, S. Rossi, and M. Bugliesi, “Understanding code semantics: a benchmark study of LLMs,” *Int. J. Softw. Tools Technol. Transf.*, Mar. 2026, doi: 10.1007/s10009-026-00842-4.
[72] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, “Asleep at the Keyboard? Assessing the Security of GitHub Copilot’s Code Contributions,” *Commun. ACM*, vol. 68, no. 2, pp. 96–105, Feb. 2025, doi: 10.1145/3610721.
[73] A. Bruno, P. L. Mazzeo, A. Chetouani, M. Tliba, and M. A. Kerkouri, “Insights into Classifying and Mitigating LLMs’ Hallucinations,” Nov. 14, 2023, *arXiv*: arXiv:2311.08117. doi: 10.48550/arXiv.2311.08117.
[74] A. Alansari and H. Luqman, “Large Language Models Hallucination: A Comprehensive Survey,” Mar. 18, 2026, *arXiv*: arXiv:2510.06265. doi: 10.48550/arXiv.2510.06265.
[75] G. Perković, A. Drobnjak, and I. Botički, “Hallucinations in LLMs: Understanding and Addressing Challenges,” in *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*, Opatija, Croatia: IEEE, May 2024, pp. 2084–2088. doi: 10.1109/MIPRO60963.2024.10569238.

[76] J. H. Klemmer *et al.*, “Using AI Assistants in Software Development: A Qualitative Study on Security Practices and Concerns,” in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, Dec. 2024, pp. 2726–2740. doi: 10.1145/3658644.3690283.
[77] C. Spiess *et al.*, “Calibration and Correctness of Language Models for Code,” Aug. 21, 2024, *arXiv*: arXiv:2402.02047. doi: 10.48550/arXiv.2402.02047.
[78] X. Zhang *et al.*, “Self-Tuning: Instructing LLMs to Effectively Acquire New Knowledge through Self-Teaching,” 2024, *arXiv*. doi: 10.48550/ARXIV.2406.06326.
[79] Y.-S. Chuang, L. Qiu, C.-Y. Hsieh, R. Krishna, Y. Kim, and J. Glass, “Lookback Lens: Detecting and Mitigating Contextual Hallucinations in Large Language Models Using Only Attention Maps,” 2024, *arXiv*. doi: 10.48550/ARXIV.2407.07071.
[80] M. Xiong *et al.*, “Can LLMs Express Their Uncertainty? An Empirical Evaluation of Confidence Elicitation in LLMs,” 2023, *arXiv*. doi: 10.48550/ARXIV.2306.13063.
[81] S. J. Mielke, A. Szlam, E. Dinan, and Y.-L. Boureau, “Reducing conversational agents’ overconfidence through linguistic calibration,” 2020, *arXiv*. doi: 10.48550/ARXIV.2012.14983.
[82] G. Dong *et al.*, “Tool-Star: Empowering LLM-Brained Multi-Tool Reasoner via Reinforcement Learning,” 2025, *arXiv*. doi: 10.48550/ARXIV.2505.16410.
[83] L. Wang *et al.*, “A survey on large language model based autonomous agents,” *Front. Comput. Sci.*, vol. 18, no. 6, p. 186345, Dec. 2024, doi: 10.1007/s11704-024-40231-1.
[84] S. J. Russell, *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.
[85] P. Putta *et al.*, “Agent Q: Advanced Reasoning and Learning for Autonomous AI Agents,” Aug. 13, 2024, *arXiv*: arXiv:2408.07199. doi: 10.48550/arXiv.2408.07199.
[86] H. Jin, L. Huang, H. Cai, J. Yan, B. Li, and H. Chen, “From LLMs to LLM-based Agents for Software Engineering: A Survey of Current, Challenges and Future,” Apr. 13, 2025, *arXiv*: arXiv:2408.02479. doi: 10.48550/arXiv.2408.02479.
[87] C. Wissuchek and P. Zschech, “Exploring Agentic Artificial Intelligence Systems: Towards a Typological Framework,” Jul. 07, 2025, *arXiv*: arXiv:2508.00844. doi: 10.48550/arXiv.2508.00844.
[88] Y. Ren, Y. Liu, T. Ji, and X. Xu, “AI Agents and Agentic AI–navigating a plethora of concepts for future manufacturing,” *J. Manuf. Syst.*, vol. 83, pp. 126–133, Dec. 2025, doi: 10.1016/j.jmsy.2025.08.017.
[89] Y. Shavit *et al.*, “Practices for governing agentic AI systems,” *Res. Pap. OpenAI*, 2023.
[90] I. Gabriel *et al.*, “The Ethics of Advanced AI Assistants,” Apr. 28, 2024, *arXiv*: arXiv:2404.16244. doi: 10.48550/arXiv.2404.16244.
[91] A. Chan *et al.*, “Harms from Increasingly Agentic Algorithmic Systems,” in *2023 ACM Conference on Fairness Accountability and Transparency*, Chicago IL USA: ACM, Jun. 2023, pp. 651–666. doi: 10.1145/3593013.3594033.
[92] A. Ng, *Welcoming diverse approaches keeps machine learning strong*. Accessed, 2024.
[93] R. Sapkota, K. I. Roumeliotis, and M. Karkee, “AI Agents vs. Agentic AI: A Conceptual taxonomy, applications and challenges,” *Inf. Fusion*, vol. 126, p. 103599, Feb. 2026, doi: 10.1016/j.inffus.2025.103599.
[94] L. Weng, “LLM Powered Autonomous Agents,” Jun. 2023, [Online]. Available: https://lilianweng.github.io/posts/2023-06-23-agent/
[95] Y.-H. Jiang *et al.*, “AI Agent for Education: von Neumann Multi-Agent System Framework,” 2025, *arXiv*. doi: 10.48550/ARXIV.2501.00083.
[96] G. Mialon *et al.*, “Augmented Language Models: a Survey,” Feb. 15, 2023, *arXiv*: arXiv:2302.07842. doi: 10.48550/arXiv.2302.07842.
[97] J. Wei *et al.*, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,” 2022, *arXiv*. doi: 10.48550/ARXIV.2201.11903.
[98] N. Krishnan, “AI Agents: Evolution, Architecture, and Real-World Applications,” Mar. 16, 2025, *arXiv*: arXiv:2503.12687. doi: 10.48550/arXiv.2503.12687.

[99] S. Yao *et al.*, “ReAct: Synergizing Reasoning and Acting in Language Models,” Mar. 10, 2023, *arXiv*: arXiv:2210.03629. doi: 10.48550/arXiv.2210.03629.
[100] L. Wang *et al.*, “Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models,” in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, Canada: Association for Computational Linguistics, 2023, pp. 2609–2634. doi: 10.18653/v1/2023.acl-long.147.
[101] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, “Reflexion: Language Agents with Verbal Reinforcement Learning,” 2023, *arXiv*. doi: 10.48550/ARXIV.2303.11366.
[102] X. Wang *et al.*, “Executable Code Actions Elicit Better LLM Agents,” 2024, *arXiv*. doi: 10.48550/ARXIV.2402.01030.
[103] X. Li *et al.*, “DeepAgent: A General Reasoning Agent with Scalable Toolsets,” in *Proceedings of the ACM Web Conference 2026*, Dubai United Arab Emirates: ACM, Apr. 2026, pp. 2219–2230. doi: 10.1145/3774904.3792460.
[104] S. Ullah, M. Han, S. Pujar, H. Pearce, A. Coskun, and G. Stringhini, “LLMs Cannot Reliably Identify and Reason About Security Vulnerabilities (Yet?): A Comprehensive Evaluation, Framework, and Benchmarks,” 2023, *arXiv*. doi: 10.48550/ARXIV.2312.12575.
[105] Y. Sun *et al.*, “LLM4Vuln: A Unified Evaluation Framework for Decoupling and Enhancing LLMs’ Vulnerability Reasoning,” 2024, *arXiv*. doi: 10.48550/ARXIV.2401.16185.
[106] A. Yildiz, S. G. Teo, Y. Lou, Y. Feng, C. Wang, and D. M. Divakaran, “Benchmarking LLMs and LLM-based Agents in Practical Vulnerability Detection for Code Repositories,” 2025, *arXiv*. doi: 10.48550/ARXIV.2503.03586.
[107] Z. Li, S. Dutta, and M. Naik, “IRIS: LLM-Assisted Static Analysis for Detecting Security Vulnerabilities,” 2024, *arXiv*. doi: 10.48550/ARXIV.2405.17238.
[108] M. Keltek, R. Hu, M. F. Sani, and Z. Li, “Boosting Cybersecurity Vulnerability Scanning based on LLM-supported Static Application Security Testing,” Nov. 22, 2024, *arXiv*: arXiv:2409.15735. doi: 10.48550/arXiv.2409.15735.
[109] D. Luna, L. Allodi, and M. Cremonini, “Productivity and Patterns of Activity in Bug Bounty Programs: Analysis of HackerOne and Google Vulnerability Research,” in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, Canterbury CA United Kingdom: ACM, Aug. 2019, pp. 1–10. doi: 10.1145/3339252.3341495.
[110] Y. Nie *et al.*, “VulnLLM-R: Specialized Reasoning LLM with Agent Scaffold for Vulnerability Detection,” 2025, *arXiv*. doi: 10.48550/ARXIV.2512.07533.
[111] Z. Liang *et al.*, “Argus: Reorchestrating Static Analysis via a Multi-Agent Ensemble for Full-Chain Security Vulnerability Detection,” 2026, *arXiv*. doi: 10.48550/ARXIV.2604.06633.
[112] G. Deng *et al.*, “PentestGPT: An LLM-empowered Automatic Penetration Testing Tool,” 2023, *arXiv*. doi: 10.48550/ARXIV.2308.06782.
[113] X. Shen *et al.*, “PentestAgent: Incorporating LLM Agents to Automated Penetration Testing,” 2024, *arXiv*. doi: 10.48550/ARXIV.2411.05185.
[114] J. W. Lin *et al.*, “Comparing AI Agents to Cybersecurity Professionals in Real-World Penetration Testing,” 2025, *arXiv*. doi: 10.48550/ARXIV.2512.09882.
[115] N. F. Liu *et al.*, “Lost in the Middle: How Language Models Use Long Contexts,” 2023, *arXiv*. doi: 10.48550/ARXIV.2307.03172.
[116] S. Ouyang, J. M. Zhang, M. Harman, and M. Wang, “An Empirical Study of the Non-determinism of ChatGPT in Code Generation,” 2023, doi: 10.48550/ARXIV.2308.02828.
[117] X. Liu *et al.*, “AgentBench: Evaluating LLMs as Agents,” 2023, *arXiv*. doi: 10.48550/ARXIV.2308.03688.
[118] L. Wang *et al.*, “A Survey on Large Language Model based Autonomous Agents,” 2023, doi: 10.48550/ARXIV.2308.11432.
[119] K. Zhu *et al.*, “Where LLM Agents Fail and How They can Learn From Failures,” 2025, *arXiv*. doi: 10.48550/ARXIV.2509.25370.
[120] N. D. Q. Bui, “Building Effective AI Coding Agents for the Terminal: Scaffolding,

Harness, Context Engineering, and Lessons Learned," 2026, *arXiv*. doi: 10.48550/ARXIV.2603.05344.
[121] B. Rombaut, "Inside the Scaffold: A Source-Code Taxonomy of Coding Agent Architectures," 2026, *arXiv*. doi: 10.48550/ARXIV.2604.03515.
[122] *https://opencode.ai/go*.
[123] *https://modelstudio.alibabacloud.com/*.
[124] *https://huggingface.co/zai-org/GLM-5.1*.
[125] *https://github.com/MoonshotAI/Kimi-K2.5*.
[126] *https://huggingface.co/MiniMaxAI/MiniMax-M2.7*.
[127] O. Johnson, A. Fomina, R. Krishnamurthy, V. Chaudhari, R. K. Shanmuganathan, and E. Bodden, "Explaining Software Vulnerabilities with Large Language Models," 2025, *arXiv*. doi: 10.48550/ARXIV.2511.04179.
[128] S. Ullah, M. Han, S. Pujar, H. Pearce, A. Coskun, and G. Stringhini, "LLMs Cannot Reliably Identify and Reason About Security Vulnerabilities (Yet?): A Comprehensive Evaluation, Framework, and Benchmarks," 2023, *arXiv*. doi: 10.48550/ARXIV.2312.12575.
[129] V. Agrawal and K. Ahi, "LLM-Driven SAST-Genius: A Hybrid Static Analysis Framework for Comprehensive and Actionable Security," 2025, *arXiv*. doi: 10.48550/ARXIV.2509.15433.
[130] S. M. Taghavi Far and F. Feyzi, "Large language models for software vulnerability detection: a guide for researchers on models, methods, techniques, datasets, and metrics," *Int. J. Inf. Secur.*, vol. 24, no. 2, p. 78, Feb. 2025, doi: 10.1007/s10207-025-00992-7.
[131] A. Khare, S. Dutta, Z. Li, A. Solko-Breslin, R. Alur, and M. Naik, "Understanding the Effectiveness of Large Language Models in Detecting Security Vulnerabilities," 2023, *arXiv*. doi: 10.48550/ARXIV.2311.16169.
[132] D. Anh-Hoang, V. Tran, and L.-M. Nguyen, "Survey and analysis of hallucinations in large language models: attribution to prompting strategies or model behavior," *Front. Artif. Intell.*, vol. 8, p. 1622292, 2025.
[133] K. Tamberg and H. Bahsi, "Harnessing Large Language Models for Software Vulnerability Detection: A Comprehensive Benchmarking Study," *IEEE Access*, vol. 13, pp. 29698–29717, 2025, doi: 10.1109/ACCESS.2025.3541146.
[134] J. Song, S. Yu, and S. Yoon, "Large Language Models are Skeptics: False Negative Problem of Input-conflicting Hallucination," 2024, *arXiv*. doi: 10.48550/ARXIV.2406.13929.