



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Automated Vulnerability Detection in Android Applications

Μαλαματένια Βαλσαμάρα

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Κολομβάτσος Κωνσταντίνος

Λαμία Φθιώτιδα έτος 2026

ΠΕΡΙΛΗΨΗ

Η συνεχής ανάπτυξη των κινητών συσκευών και η εκτεταμένη χρήση των Android εφαρμογών σε ένα διαρκώς διευρυνόμενο φάσμα υπηρεσιών έχουν ενισχύσει τη σημασία της ασφάλειας στο επίπεδο των εφαρμογών. Διαχρονικά, το μοντέλο ασφάλειας του Android έχει εξελιχθεί σημαντικά, ενσωματώνοντας μηχανισμούς για την απομόνωση των εφαρμογών, τον έλεγχο της πρόσβασης σε πόρους του συστήματος και την προστασία των δεδομένων των χρηστών. Ωστόσο, σημαντικό μέρος της ασφάλειας μιας εφαρμογής εξακολουθεί να εξαρτάται από τις επιλογές των προγραμματιστών κατά τον σχεδιασμό και την υλοποίησή της. Ανεπαρκείς έλεγχοι και σφάλματα υλοποίησης μπορούν επομένως να οδηγήσουν στην εμφάνιση ευπαθειών που επηρεάζουν την εμπιστευτικότητα, την ακεραιότητα ή τη διαθεσιμότητα των δεδομένων και των λειτουργιών της εφαρμογής, παρά τους μηχανισμούς προστασίας που παρέχει το ίδιο το λειτουργικό σύστημα. Παράλληλα, η εξέλιξη των Μεγάλων Γλωσσικών Μοντέλων (*Large Language Models – LLMs*) δημιουργεί νέες δυνατότητες για την αυτοματοποίηση εργασιών στον χώρο της ασφάλειας λογισμικού. Εργασίες που βασίζονται σε σημαντικό βαθμό στην ανθρώπινη εμπειρία, όπως η αναγνώριση μοτίβων στον κώδικα, η κατανόηση της ροής μιας εφαρμογής, η συσχέτιση πληροφοριών από διαφορετικά σημεία του κώδικα και η διερεύνηση πιθανών ευπαθειών, μπορούν πλέον να υποστηριχθούν από συστήματα που αξιοποιούν τέτοια μοντέλα. Στην κατεύθυνση αυτή, η παρούσα πτυχιακή εργασία εξετάζει την αξιοποίηση των LLMs σε συνδυασμό με εργαλεία στατικής και δυναμικής ανάλυσης για την αυτοματοποίηση της έρευνας ευπαθειών σε εφαρμογές Android. Στο πλαίσιο της εργασίας μελετώνται οι βασικοί μηχανισμοί του Android που σχετίζονται με την επιφάνεια επίθεσης μιας εφαρμογής, καθώς και χαρακτηριστικές κατηγορίες ευπαθειών που μπορούν να προκύψουν από τη λανθασμένη χρήση τους. Παράλληλα, εξετάζεται ο τρόπος με τον οποίο τα ευρήματα της στατικής και της δυναμικής ανάλυσης μπορούν να συνδυαστούν στο πλαίσιο μιας ενιαίας διαδικασίας ελέγχου ασφάλειας, ώστε οι πληροφορίες που προκύπτουν σε κάθε στάδιο να καθοδηγούν τη συνέχεια της διερεύνησης. Η διαδικασία ολοκληρώνεται με την αξιολόγηση των υποψήφιων διαδρομών βάσει των τεχνικών τεκμηρίων που έχουν συγκεντρωθεί, τη διάκριση των επιβεβαιωμένων ευπαθειών από τις μη εκμεταλλεύσιμες περιπτώσεις και την παραγωγή τεκμηριωμένων αναφορών για τα τελικά findings.

Περιεχόμενο: Η παρούσα πτυχιακή εργασία πραγματεύεται την ανάπτυξη μιας ολοκληρωμένης προσέγγισης, η οποία δεν περιορίζεται μόνο στον εντοπισμό πιθανών ευπαθειών μέσω της ανάλυσης του κώδικα, αλλά προχωρά στην δυναμική διερεύνηση και επαληθευση τους.

ABSTRACT

The continuous development of mobile devices and the widespread use of Android applications across an ever-expanding range of services have increased the importance of application-level security. Over time, the Android security model has evolved significantly, incorporating mechanisms for application isolation, access control to system resources, and the protection of user data. However, a substantial part of an application's security still depends on the choices made by developers during its design and implementation. Inadequate checks and implementation flaws can therefore lead to vulnerabilities that affect the confidentiality, integrity, or availability of application data and functionality, despite the protection mechanisms provided by the operating system itself. At the same time, the evolution of Large Language Models (LLMs) has created new opportunities for the automation of tasks in the field of software security. Tasks that rely heavily on human expertise, such as identifying suspicious patterns in code, understanding application flow, correlating information across different parts of a codebase, and investigating potential vulnerabilities, can now be supported by systems that make use of such models. In this direction, this thesis investigates the use of LLMs in combination with static and dynamic analysis tools for the automation of vulnerability research in Android applications. In parallel, the thesis examines how findings from static and dynamic analysis can be combined within a unified security assessment process, allowing the information obtained at each stage to guide the subsequent investigation. The process concludes with the evaluation of candidate paths based on the technical evidence collected, the distinction between confirmed vulnerabilities and non-exploitable cases, and the production of documented reports for the final findings.

Περιεχόμενα

1.	ΕΙΣΑΓΩΓΗ	5
1.1	Κίνητρα και σύγχρονες εξελίξεις	5
1.2	Παρουσίαση του προβλήματος	8
1.3	Δομή της εργασίας	13
1.4	Σύνοψη	15
2.	ΑΣΦΑΛΕΙΑ ΣΤΟ ANDROID OS	15
2.1	Αρχιτεκτονική του Android	15
2.2	Εξέλιξη του μοντέλου ασφάλειας του Android	18
2.2.1	Μηχανισμός απομόνωσης εφαρμογών	18
2.2.2	Μοντέλο δικαιωμάτων του Android	22
2.2.3	Προστασία δεδομένων και αποθηκευτικού χώρου	23
2.3	Σύγχρονοι μηχανισμοί ασφάλειας σε επίπεδο εφαρμογών	24
2.4	Ευπάθειες σε επίπεδο εφαρμογών	30
2.4.1	Μη ασφαλής χρήση του WebView	30
2.4.2	Μη ασφαλής διαχείριση των Intents	43
2.4.3	Μη ασφαλής έκθεση των Services και Broadcast Receivers	49
2.4.4	Μη ασφαλής αποσειριοποίηση και ανακατασκευή αντικειμένων	55
2.4.5	Μη ασφαλής δυναμική φόρτωση κώδικα	57
2.4.6	Ευπάθειες στη χρήση JNI	58
2.4.7	Έκθεση ευαίσθητων δεδομένων και μυστικών	61
2.5	Σύνοψη	64
3.	ΠΡΟΤΕΙΝΟΜΕΝΗ ΠΡΟΣΕΓΓΙΣΗ	65
3.1	Στόχοι και σχεδιαστικές αρχές	65
3.2	Επιλογή μοντέλου	67
3.3	Επιλογή και ενοποίηση εργαλείων	70
3.3.1	Εργαλεία στατικής ανάλυσης	71
3.3.2	Εργαλεία δυναμικής ανάλυσης	74
3.3.3	Συντονισμός της διαδικασίας ανάλυσης	78
3.4	Αρχιτεκτονική	81
3.5	Σύνοψη	94

4.	ΑΠΟΤΕΛΕΣΜΑΤΑ ΚΑΙ ΣΥΜΠΕΡΑΣΜΑΤΑ	94
4.1	Εφαρμογή της μεθοδολογίας και αποτελέσματα	95
4.2	Συμβολή της παρούσας εργασίας	96
4.3	Σύνοψη	97
5.	ΒΙΒΛΙΟΓΡΑΦΙΑ	98

1. Εισαγωγή

“The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards - and even then I have my doubts. – Gene Spafford”

[Scientific American, Vol 260, No. 3, March 1989, pp. 110–113]

1.1 Κίνητρα και σύγχρονες εξελίξεις

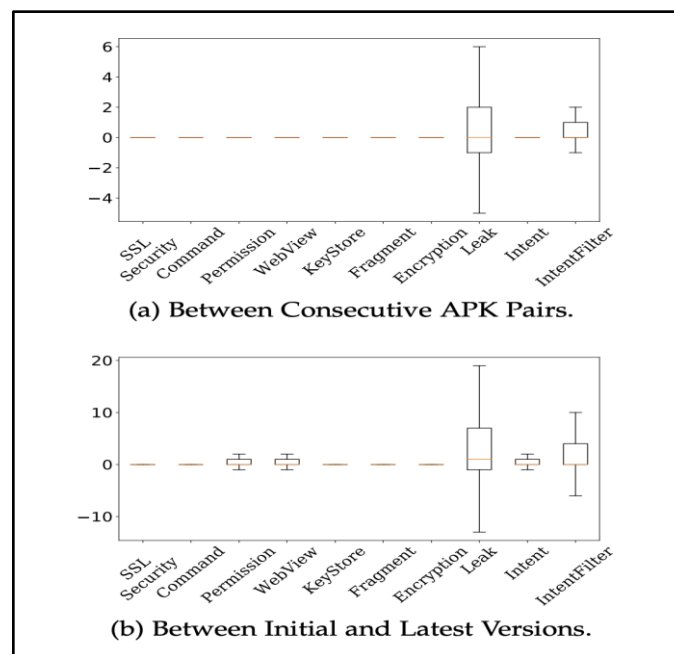
Η εξέλιξη των φορητών συσκευών έχει αλλάξει σημαντικά τον τρόπο αλληλεπίδρασης των χρηστών με τα σύγχρονα πληροφοριακά συστήματα. Από τις πρώτες φορητές συσκευές με περιορισμένες λειτουργίες έως τα σύγχρονα κινητά τηλέφωνα, έχουν εξελιχθεί σε πλήρη υπολογιστικά συστήματα, ικανά να εκτελούν ένα ευρύ φάσμα εργασιών και υπηρεσιών. Οι εφαρμογές επικοινωνίας, ηλεκτρονικών συναλλαγών, κοινωνικής δικτύωσης, ψυχαγωγίας και διαχείρισης προσωπικών δεδομένων αποτελούν αναπόσπαστο μέρος της καθημερινής τους χρήσης.

Το Android είναι ένα κεντρικό στοιχείο αυτής της ανάπτυξης, καθώς έχει μετατραπεί από ένα λειτουργικό σύστημα κινητών τηλεφώνων σε ένα ευρύτερο οικοσύστημα υπολογιστών. Ήδη από το 2022, η Google είχε αναφέρει περισσότερα από τρία δισεκατομμύρια ενεργές συσκευές Android σε μηνιαία βάση, ενώ το οικοσύστημα επεκτείνεται πέρα από τα κινητά τηλέφωνα, σε τηλεοράσεις, συστήματα αυτοκινήτων, φορητές συσκευές και άλλες κατηγορίες συνδεδεμένων συσκευών [1]. Η επέκταση αυτή έχει ως αποτέλεσμα την εκτέλεση εφαρμογών με διαφορετικά μοντέλα χρήσης, δυνατότητες και απαιτήσεις ασφαλείας πάνω στην ίδια βασική πλατφόρμα.

Παράλληλα, το μοντέλο ασφάλειας του Android έχει υποστεί σημαντικές εξελίξεις. Οι συνεχείς εκδόσεις της πλατφόρμας έχουν βελτιώσει την απομόνωση των εφαρμογών και έχουν σταδιακά περιορίσει την πρόσβασή τους σε πόρους και δεδομένα του συστήματος. Για παράδειγμα, η αρχική απομόνωση μέσω διαφορετικών Linux UIDs βελτιώθηκε με την υιοθέτηση του SELinux και την εφαρμογή του `seccomp-bpf` για τον περιορισμό των system calls, ενώ το permission model εξελίχθηκε από την εκχώρηση δικαιωμάτων κατά την εγκατάσταση, στην υιοθέτηση runtime permissions για ευαίσθητες λειτουργίες και δεδομένα. Σε μεταγενέστερες εκδόσεις εφαρμόστηκαν πιο αυστηροί περιορισμοί στην πρόσβαση σε κοινόχρηστο αποθηκευτικό χώρο και στα δεδομένα που είναι προσβάσιμα στις εφαρμογές [2], [30], [31], [33], [35].

Αυτοί οι μηχανισμοί περιορίζουν σε μεγάλο βαθμό τις ενέργειες που μπορεί να εκτελέσει μια εφαρμογή αυθαίρετα, χωρίς να εξαλείφουν τις ευπάθειες που προκύπτουν από τον τρόπο με τον οποίο η ίδια χρησιμοποιεί τις διεπαφές και τους μηχανισμούς της πλατφόρμας. Ένα χαρακτηριστικό παράδειγμα είναι η επικοινωνία μεταξύ εφαρμογών μέσω Intents και εκτεθειμένων application components. Οι Chin et al. [3] έδειξαν ότι ο μηχανισμός επικοινωνίας μεταξύ εφαρμογών του Android μπορεί να αποτελέσει επιφάνεια επίθεσης όταν τα μηνύματα ή τα components δεν προστατεύονται σωστά, επιτρέποντας επιθέσεις όπως interception of Intent, spoofing of Intent και μη εξουσιοδοτημένη ενεργοποίηση components.

Η ευρύτερη έκταση του προβλήματος και η τάση των ευπαθειών να διατηρούνται μέσα στον χρόνο αποτυπώνονται χαρακτηριστικά στη διαχρονική μελέτη των Gao et al. [4], στην οποία αναλύθηκαν 465.037 APKs οργανωμένα σε 28.564 οικογένειες διαδοχικών εκδόσεων εφαρμογών και εξετάστηκαν κατηγορίες ευπαθειών που σχετίζονταν, μεταξύ άλλων, με permissions, WebViews, Intents και Intent Filters. Τα αποτελέσματα της μελέτης έδειξαν ότι περισσότερες από τις μισές ευπάθειες παρέμεναν αμετάβλητες μετά από μια ενημέρωση της εφαρμογής, ενώ σημαντικό μέρος τους εξακολουθούσε να εμφανίζεται σε τρεις ή περισσότερες διαδοχικές εκδόσεις.



Εικόνα 1: Μεταβολές στον αριθμό των ευπαθειών μετά από ενημερώσεις εφαρμογών Android.

Πηγή : [4, fig 13]

Κατά συνέπεια, ακόμα και σε ένα πιο ασφαλές λειτουργικό περιβάλλον, τα σφάλματα στη λογική ή στην υλοποίηση μιας εφαρμογής μπορούν να οδηγήσουν σε εύλωτες καταστάσεις. Το Android παρουσιάζει

ιδιαίτερο ερευνητικό ενδιαφέρον λόγω της μεγάλης έκτασης του οικοσυστήματός και της σύνθετης αλληλεπίδρασης μεταξύ των μηχανισμών ασφάλειας που παρέχει το λειτουργικό σύστημα και του κώδικα των εφαρμογών που εκτελούνται σε αυτό.

Παράλληλα, η πρόοδος των Μεγάλων Γλωσσικών Μοντέλων (Large Language Models – LLMs) έχει διευρύνει την αξιοποίησή τους σε σύνθετες εργασίες που σχετίζονται με την κατανόηση και την τροποποίηση πηγαιού κώδικα. Χαρακτηριστικό παράδειγμα αποτελεί το SWE-bench των Jimenez et al. [5], το οποίο περιλαμβάνει 2.294 πραγματικά ζητήματα λογισμικού από αποθετήρια του GitHub και αξιολογεί την ικανότητα των μοντέλων να εντοπίζουν και να εφαρμόζουν τις απαιτούμενες αλλαγές στην αντίστοιχη βάση κώδικα. Η επίλυση τέτοιων προβλημάτων απαιτεί συχνά την εξέταση διαφορετικών σημείων ενός έργου λογισμικού και την κατανόηση των σχέσεων που αναπτύσσονται μεταξύ επιμέρους τμημάτων του κώδικα [5].

Σημαντική εξέλιξη στην ίδια κατεύθυνση αποτελεί η ανάπτυξη agentic συστημάτων, στα οποία το γλωσσικό μοντέλο εντάσσεται σε μια επαναληπτική διαδικασία αλληλεπίδρασης με εξωτερικά εργαλεία και περιβάλλοντα εκτέλεσης. Το μοντέλο μπορεί να αξιολογεί την τρέχουσα κατάσταση, να επιλέγει μια ενέργεια, να επεξεργάζεται το αποτέλεσμα της εκτέλεσής της και να προσαρμόζει ανάλογα τη συνέχεια της διαδικασίας. Η προσέγγιση αυτή έχει ήδη εφαρμοστεί σε εργασίες μηχανικής λογισμικού (software engineering), όπως στο SWE-agent των Yang et al. [6], όπου agents βασισμένοι σε γλωσσικά μοντέλα αλληλεπιδρούν με ένα υπολογιστικό περιβάλλον για την επίλυση πραγματικών προβλημάτων λογισμικού.

Οι εξελίξεις αυτές έχουν ενισχύσει το ερευνητικό ενδιαφέρον για την εφαρμογή των LLMs στην ασφάλεια λογισμικού, ιδιαίτερα σε εργασίες εντοπισμού και επιδιόρθωσης ευπαθειών [7], [8]. Η ανάλυση μιας πραγματικής ευπάθειας απαιτεί τη συσχέτιση πληροφοριών που βρίσκονται σε διαφορετικά σημεία της εφαρμογής, καθώς και τη διαδοχική αξιοποίηση εργαλείων που παρέχουν διαφορετικού τύπου πληροφορίες για τη συμπεριφορά του συστήματος. Η δυνατότητα των σύγχρονων γλωσσικών μοντέλων να επεξεργάζονται τον κώδικα στο ευρύτερο πλαίσιο μιας εφαρμογής και να αξιοποιούν τα αποτελέσματα εξωτερικών εργαλείων δημιουργεί τις προϋποθέσεις για την υποστήριξη πιο ολοκληρωμένων διαδικασιών ανάλυσης ασφάλειας.

Η ίδια λογική μπορεί να εφαρμοστεί σε διαδικασίες έρευνας ευπαθειών που εξελίσσονται σε διαδοχικά στάδια και προσαρμόζονται στα ευρήματα που προκύπτουν κατά την ανάλυση. Ένα ύποπτο σημείο που εντοπίζεται μέσω στατικής ανάλυσης μπορεί, για παράδειγμα, να οδηγήσει στη συλλογή πρόσθετων πληροφοριών, στην επιλογή κατάλληλου εργαλείου και στην εκτέλεση δυναμικού ελέγχου, του οποίου τα αποτελέσματα καθορίζουν τη συνέχεια της διερεύνησης. Η αξιοποίηση γλωσσικών μοντέλων σε τέτοιες

διαδικασίες έχει ήδη αποτελέσει αντικείμενο έρευνας, με χαρακτηριστικό παράδειγμα το PentestGPT των Deng et al. [9], το οποίο εξετάζει τη χρήση LLMs για την υποστήριξη διαφορετικών σταδίων ενός penetration test, από την αξιοποίηση εργαλείων και την ερμηνεία των αποτελεσμάτων τους έως τον σχεδιασμό των επόμενων ενεργειών.

Στο πλαίσιο αυτό εντάσσεται και η παρούσα εργασία, η οποία εξετάζει σε ποιο βαθμό μπορεί να αυτοματοποιηθεί η διαδικασία έρευνας ευπαθειών σε εφαρμογές Android, από τον αρχικό εντοπισμό ενός πιθανού προβλήματος έως τη διερεύνηση και την πρακτική επαλήθευσή του.

1.2 Παρουσίαση του προβλήματος

Η ανίχνευση ευπαθειών σε εφαρμογές Android αποτελεί σύνθετη διαδικασία, καθώς η ύπαρξη ενός προβλήματος ασφάλειας δεν προκύπτει πάντοτε από μια μεμονωμένη εντολή ή από ένα τοπικό μοτίβο κώδικα. Η αξιολόγηση ενός μηχανισμού προϋποθέτει την κατανόηση του τρόπου με τον οποίο αυτός εντάσσεται στη συνολική λειτουργία της εφαρμογής και αλληλεπιδρά με εξωτερικά δεδομένα ή άλλα components. Στην περίπτωση ενός WebView, για παράδειγμα, ο εντοπισμός μιας μεμονωμένης κλήσης φόρτωσης μιας διεύθυνσης URL δεν αρκεί για να χαρακτηριστεί ο αντίστοιχος κώδικας ως ευάλωτος, καθώς απαιτείται να εξεταστεί αν η τιμή μπορεί να επηρεαστεί από εξωτερική είσοδο και με ποιον τρόπο ελέγχεται πριν από τη φόρτωσή της [3], [4].

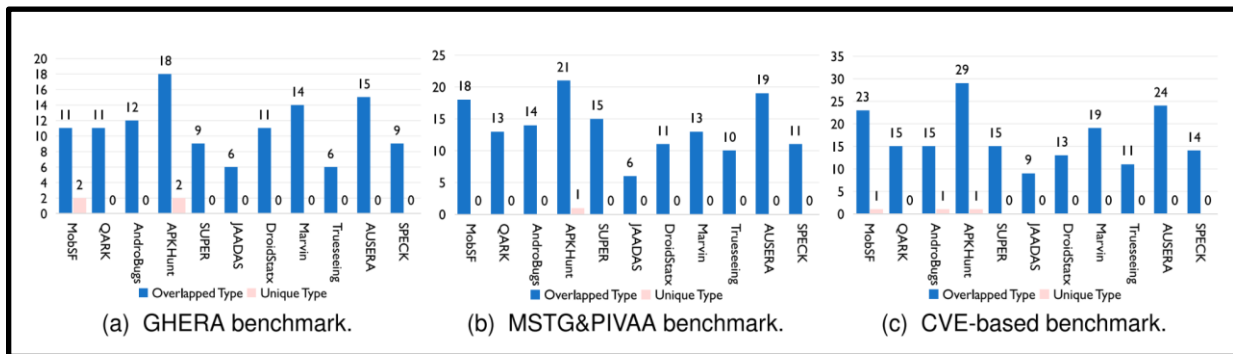
Η συγκεκριμένη εξάρτηση από το ευρύτερο πλαίσιο της εκάστοτε εφαρμογής, περιορίζει την αποτελεσματικότητα προσεγγίσεων που βασίζονται, κυρίως, στην αναγνώριση προκαθορισμένων μοτίβων. Εργαλεία στατικής ανάλυσης, μπορούν να εξετάσουν το AndroidManifest.xml, να εντοπίσουν χρήσεις ευαίσθητων APIs, μη ασφαλείς ρυθμίσεις, ενσωματωμένα ευαίσθητα δεδομένα (hard-coded secrets) ή γνωστά μη ασφαλή μοτίβα κώδικα [10], [11]. Αντίστοιχη λογική χρησιμοποιείται και από γενικότερα εργαλεία στατικής ανάλυσης ασφάλειας (Static Application Security Testing – SAST), τα οποία εξετάζουν τον κώδικα με βάση προκαθορισμένους κανόνες και μοντέλα ανάλυσης για τον εντοπισμό πιθανών προβλημάτων. Σε πιο σύνθετες περιπτώσεις, η ανάλυση μπορεί να παρακολουθεί τη ροή δεδομένων μέσα στην εφαρμογή μέσω τεχνικών taint analysis, εξετάζοντας αν δεδομένα που προέρχονται από μη έμπιστες πηγές μπορούν να καταλήξουν σε ευαίσθητα ή επικίνδυνα σημεία χρήσης [12].

Οι Zhu et al. [10] αξιολόγησαν έντεκα εργαλεία SAST για Android και, έπειτα από εξέταση των μηχανισμών ανίχνευσής τους, ενοποίησαν τις δυνατότητές τους σε 67 γενικές κατηγορίες ευπαθειών. Στη μελέτη συμπεριλήφθηκαν τα MobSF, QARK, AndroBugs, APKHunt, SUPER, JAADAS, DroidStatx,

Marvin, Trueseeing, AUSERA και SPECK, τα οποία ακολουθούν διαφορετικές προσεγγίσεις κατά την ανάλυση του κώδικα. Ενδεικτικά, τα SUPER, APKHunt και SPECK βασίζονται κυρίως σε αντιστοίχιση μοτίβων πάνω σε decompiled κώδικα, τα MobSF και QARK χρησιμοποιούν AST-based pattern matching, ενώ σε άλλα εργαλεία συναντώνται τεχνικές σημασιολογικής ανάλυσης, control-flow και data-flow analysis [10].

Παρά το εύρος των τεχνικών αυτών, η μελέτη αναγνώρισε πέντε κύριες τεχνικές αιτίες που μπορούν να μειώσουν την αποτελεσματικότητα των εργαλείων [10]:

- **Η λεπτομέρεια των κανόνων ανίχνευσης** : Οι πολύ γενικοί κανόνες μπορούν να παράγουν ένα μεγάλο πλήθος ψευδώς θετικών αποτελεσμάτων, ενώ οι υπερβολικά αυστηροί κανόνες μπορούν να αγνοήσουν πραγματικά περιστατικά.
- **Προβλήματα στην λογική των ανιχνευτών** : Σε ορισμένες περιπτώσεις, η υλοποίηση του ελέγχου δεν αντικατοπτρίζει απόλυτα την δηλωμένη ικανότητα ανίχνευσης του εργαλείου.
- **Έλλειψη επαρκούς πλαισίου κώδικα (code context)** : Η αναγνώριση ενός API ή μιας σύνταξης χωρίς να εξεταστεί ο τρόπος χρήσης της μπορεί να μην είναι επαρκής για την αξιολόγηση μιας ευπάθειας.
- **Περιορισμοί των βασικών πλαισίων (frameworks) και βιβλιοθηκών** : Οι αστοχίες στα πλαίσια στατικής ανάλυσης από τα οποία εξαρτώνται τα εργαλεία μπορούν να επηρεάσουν τα αποτελέσματά τους.
- **Προβλήματα στην διαδικασία του decompilation** : Η ατελής ανακατασκευή του κώδικα μπορεί να οδηγήσει σε απώλεια τμημάτων του προγράμματος και, κατά συνέπεια, σε μη εντοπισμό ευπαθειών [10].



Εικόνα 3: Κάλυψη τύπων ευπαθειών από εργαλεία SAST για Android σε διαφορετικά benchmarks.

Πηγή: [10, Fig. 3].

Ένας βασικός περιορισμός της στατικής ανάλυσης είναι ότι ο εντοπισμός μιας δυνητικά επικίνδυνης λειτουργίας δεν αρκεί για να προσδιοριστεί αν η συγκεκριμένη χρήση της συνιστά πραγματική ευπάθεια. Στη μελέτη των Zhu et al., για παράδειγμα, εργαλεία όπως τα MobSF και APKHunt μπορούσαν να αναφέρουν πιθανές διαρροές πληροφοριών όταν εντόπιζαν κλήσεις σε APIs καταγραφής, όπως η **Log.e()**, ωστόσο, η παρουσία μιας τέτοιας κλήσης δεν σημαίνει αυτόματα την διαρροή των δεδομένων. Η ορθή αξιολόγηση ενός τέτοιου ευρήματος προϋποθέτει την εξέταση της πληροφορίας που φτάνει στο σημείο καταγραφής και του πλαισίου μέσα στο οποίο πραγματοποιείται η συγκεκριμένη λειτουργία, διαφορετικά ο στατικός έλεγχος μπορεί να καταλήξει σε μεγάλο αριθμό false positives [10].

Εργαλεία όπως το AUSERA και το Trueseeing επιχειρούν να περιορίσουν αυτή την αβεβαιότητα αξιοποιώντας ανάλυση της ροής δεδομένων, ώστε το εύρημα να αξιολογείται σε μεγαλύτερο πλαίσιο και όχι μόνο με βάση την παρουσία μιας συγκεκριμένης κλήσης [10]. Παρ' όλα αυτά, τα αποτελέσματα της στατικής ανάλυσης εξακολουθούν να απαιτούν περαιτέρω διερεύνηση πριν χαρακτηριστούν ως πραγματικές ευπάθειες, ιδιαίτερα όταν οι κανόνες ανίχνευσης είναι αρκετά γενικοί. Η επίδραση αυτού του περιορισμού φαίνεται και στα αποτελέσματα της ίδιας μελέτης, όπου για ένα πραγματικό APK το SUPER ανέφερε 597 περιπτώσεις *Logging Data Exposure*, ενώ το APKHunt 1.210, δημιουργώντας έναν μεγάλο όγκο ευρημάτων που έπρεπε να αξιολογηθούν χειροκίνητα [10]. Πιο προηγμένα εργαλεία στατικής ανάλυσης επεκτείνουν αυτή την προσέγγιση με ακριβέστερη μοντελοποίηση της ροής και της εκτέλεσης του προγράμματος. Το FlowDroid, για παράδειγμα, εφαρμόζει *context-, flow-, field-, object-sensitive* και *lifecycle-aware taint analysis* ειδικά για εφαρμογές Android, επιτρέποντας την παρακολούθηση της διάδοσης δεδομένων από καθορισμένες πηγές προς ευαίσθητα σημεία χρήσης [12]. Αντίστοιχες δυνατότητες παρέχει και το CodeQL για Java και Kotlin, μέσω μηχανισμών τοπικής και καθολικής ανάλυσης data-flow και taint tracking, με δυνατότητα ορισμού *sources, sinks, barriers* και πρόσθετων βημάτων ροής [16].

Ωστόσο, η αύξηση της λεπτομέρειας στην ανάλυση δημιουργεί νέα προβλήματα. Ενδεικτικά:

- Η *global data-flow analysis* εξετάζει τη ροή των δεδομένων μεταξύ διαφορετικών συναρτήσεων και τμημάτων του προγράμματος και είναι πιο χρονοβόρα και απαιτητική σε υπολογιστικούς πόρους από την τοπική ανάλυση [16],
- Η ακρίβεια της ανάλυσης εξαρτάται από τη σωστή μοντελοποίηση των *sources, sinks, barriers* και των πρόσθετων βημάτων διάδοσης των δεδομένων [16],
- Οι εξωτερικές βιβλιοθήκες και τα frameworks μπορεί να απαιτούν πρόσθετα models ή summaries, ώστε η ανάλυση να γνωρίζει πώς διαδίδονται τα δεδομένα μέσω κώδικα που δεν είναι διαθέσιμος ή δεν μοντελοποιείται άμεσα [16]

- Στο Android, η ανάλυση περιπλέκεται περαιτέρω από το ιδιαίτερο lifecycle των components και τη χρήση callbacks, τα οποία απαιτούν ειδική μοντελοποίηση για την ανακατασκευή των πιθανών ροών εκτέλεσης [12]

Σε ένα σενάριο ανάλυσης πραγματικών εφαρμογών, ο αρχικός πηγαίος κώδικας δεν είναι πάντα διαθέσιμος. Η ανάλυση μπορεί να ξεκινά μόνο από το APK, το οποίο πρέπει να γίνει decompiled ή να μετατραπεί σε μια ενδιάμεση αναπαράσταση. Αυτή η διαδικασία δεν ανακατασκευάζει απαραίτητα τον αρχικό κώδικα με ακρίβεια και μπορεί να επηρεάζεται περαιτέρω από τεχνικές όπως *obfuscation*, *reflection* ή *dynamic code loading* [17]. Η αξιολόγηση Android SAST εργαλείων των Zhu et al. έδειξε ότι προβλήματα κατά το *decompilation* μπορούν να οδηγήσουν ακόμη και σε απώλεια τμημάτων κώδικα ή μεταβολή παραμέτρων, προκαλώντας *false negatives* σε εργαλεία που πραγματοποιούν την ανάλυσή τους πάνω στον ανακατασκευασμένο κώδικα [10].

Οι περιορισμοί της στατικής ανάλυσης καθιστούν τη δυναμική εξέταση ιδιαίτερα σημαντική, καθώς η εκτέλεση της εφαρμογής επιτρέπει την παρατήρηση της πραγματικής συμπεριφοράς της σε συγκεκριμένες συνθήκες και με πραγματικά δεδομένα εισόδου [18]. Μέσα από αυτή τη διαδικασία μπορούν να εξεταστούν οι αλληλεπιδράσεις της εφαρμογής με το λειτουργικό σύστημα, το δίκτυο και τα δεδομένα που διαχειρίζεται κατά την εκτέλεση. Εργαλεία όπως το Frida παρέχουν δυνατότητες δυναμικού instrumentation και hooking συναρτήσεων μέσα στην εκτελούμενη διεργασία [19], ενώ το objection αξιοποιεί το Frida για την υποστήριξη runtime exploration σε εφαρμογές Android, επιτρέποντας την επιθεώρηση και την τροποποίηση της κατάστασης της εφαρμογής κατά την εκτέλεση [20]. Μια πιο ολοκληρωμένη προσέγγιση προσφέρεται από το MobSF, το οποίο συνδυάζει στατική και δυναμική ανάλυση, υποστηρίζοντας την εξέταση APKs και την παρακολούθηση της συμπεριφοράς τους σε ελεγχόμενο περιβάλλον εκτέλεσης [21].

Η δυναμική ανάλυση περιορίζεται στις συμπεριφορές που ενεργοποιούνται κατά τη συγκεκριμένη εκτέλεση της εφαρμογής, γεγονός που καθιστά κρίσιμη την ικανότητα του συστήματος να προσεγγίσει την κατάλληλη κατάσταση και να ενεργοποιήσει το σχετικό execution path. Η εκδήλωση μιας ευπάθειας μπορεί να εξαρτάται από προηγούμενες ενέργειες του χρήστη, από συγκεκριμένες συνθήκες εκτέλεσης ή από δεδομένα που παρέχονται μέσω μηχανισμών όπως τα Intents, με αποτέλεσμα η απλή εκκίνηση και παρακολούθηση της εφαρμογής να μην επαρκεί για την κάλυψη όλων των πιθανών περιπτώσεων.

Το πρόβλημα της αποτελεσματικής εξερεύνησης του state space έχει μελετηθεί εκτενώς και στο Android GUI testing. Οι Su et al. δημιούργησαν το Themis, ένα benchmark που περιλαμβάνει 52 πραγματικά και αναπαραγώγιμα Android crash bugs, και αξιολόγησαν έξι σύγχρονα εργαλεία αυτοματοποιημένου GUI

testing σε περισσότερες από 10.920 ώρες εκτέλεσης [13]. Παρά την εκτεταμένη αυτή αξιολόγηση, το 34,6% των σφαλμάτων δεν εντοπίστηκε από κανένα από τα εξεταζόμενα εργαλεία, γεγονός που αναδεικνύει τη δυσκολία κάλυψης όλων των καταστάσεων και διαδρομών εκτέλεσης μιας εφαρμογής.

Η ανάλυση των bugs που δεν εντοπίστηκαν ανέδειξε ορισμένες επαναλαμβανόμενες δυσκολίες, οι οποίες επηρέασαν σημαντικά την αποτελεσματικότητα των εργαλείων [13]:

- Η πρόσβαση σε βαθιές καταστάσεις της εφαρμογής που απαιτούν μεγάλες ακολουθίες ενεργειών.
- Η ανάγκη για συγκεκριμένες τιμές εισόδου.
- Οι μεταβολές σε ρυθμίσεις της εφαρμογής ή του συστήματος.
- Η ανάγκη για συγκεκριμένα μοτίβα αλληλεπίδρασης του χρήστη.
- Οι αλληλεπιδράσεις με εξωτερικές εφαρμογές.

Παρότι η συγκεκριμένη μελέτη επικεντρώνεται σε crash bugs και όχι σε ευπάθειες ασφάλειας, τα αποτελέσματά της αναδεικνύουν έναν γενικότερο περιορισμό της δυναμικής ανάλυσης: η υψηλή κάλυψη κώδικα δεν συνεπάγεται απαραίτητα την ενεργοποίηση της ακολουθίας γεγονότων που απαιτείται για την εμφάνιση ενός προβλήματος. Οι ίδιοι οι συγγραφείς επισημαίνουν ότι μετρικές όπως το line coverage ή το activity coverage δεν αποτυπώνουν επαρκώς σύνθετες αλληλεπιδράσεις μεταξύ callbacks, οι οποίες μπορεί να είναι καθοριστικές για την αναπαραγωγή συγκεκριμένων συμπεριφορών [13].

Οι παραπάνω προσεγγίσεις επομένως, παρέχουν διαφορετικού τύπου πληροφορίες και ενδείξεις, χωρίς όμως καθεμία από αυτές να επαρκεί μεμονωμένα για την πλήρη διερεύνηση μιας πιθανής ευπάθειας [10],[12],[17]. Ένα SAST εργαλείο μπορεί να αναγνωρίσει μια επικίνδυνη κλήση API, ένα *taint-analysis* εργαλείο να εντοπίσει μια πιθανή διαδρομή δεδομένων και ένα εργαλείο δυναμικού *instrumentation* να επιτρέψει την παρατήρηση της σχετικής συμπεριφοράς κατά την εκτέλεση. Ωστόσο, παραμένει το πρόβλημα της συσχέτισης αυτών των πληροφοριών.

Η αξιοποίηση των LLMs μπορεί να υποστηρίξει τη συσχέτιση πληροφοριών από διαφορετικά σημεία της εφαρμογής και την επιλογή επόμενων ενεργειών στο πλαίσιο μιας *agentic* διαδικασίας [6], [9]. Η χρήση τους, ωστόσο, απαιτεί μηχανισμούς επαλήθευσης, καθώς οι εκτιμήσεις τους είναι πιθανοτικές και μπορεί να βασίζονται σε ελλειπείς ή μη επαρκώς τεκμηριωμένες πληροφορίες [7], [8]. Η δυσκολία αυτή γίνεται εντονότερη σε μεγάλες εφαρμογές, όπου τα στοιχεία που σχετίζονται με ένα πιθανό πρόβλημα κατανέμονται σε διαφορετικά αρχεία, *components* και επίπεδα εκτέλεσης, αυξάνοντας τις απαιτήσεις σε *context* και καθιστώντας κρίσιμη την επιλογή των πληροφοριών που θα ληφθούν υπόψη [5], [8].

Η παρούσα εργασία εξετάζει μια ευρύτερη διαδικασία από την απλή αυτόματη επισημάνση ύποπτου κώδικα. Στόχος είναι η μετάβαση από ένα αρχικό εύρημα σε μια τεκμηριωμένη διερεύνηση, κατά την οποία το σύστημα αξιοποιεί πληροφορίες από στατική και δυναμική ανάλυση, διαμορφώνει ελέγξιμες υποθέσεις και χρησιμοποιεί τα αποτελέσματα των επιμέρους ελέγχων για να επανεκτιμήσει, να επιβεβαιώσει ή να απορρίψει τα αρχικά ευρήματα.

1.3 Δομή της εργασίας

Η παρούσα εργασία απαρτίζεται από τέσσερα κεφάλαια και οργανώνεται εννοιολογικά σε τρία κύρια μέρη. Στο πρώτο μέρος, παρουσιάζεται το απαραίτητο θεωρητικό υπόβαθρο σχετικά με την ασφάλεια του Android, την εξέλιξη των μηχανισμών προστασίας της πλατφόρμας και τις ευπάθειες που μπορούν να εμφανιστούν στις εφαρμογές. Στο δεύτερο μέρος, αναλύονται οι τεχνικές και τα εργαλεία που χρησιμοποιούνται για την στατική και δυναμική ανάλυση εφαρμογών, καθώς και οι δυνατότητες αξιοποίησης Μεγάλων Γλωσσικών Μοντέλων για την αυτοματοποίηση της ανάλυσης ασφάλειας. Στο τρίτο μέρος, παρουσιάζεται η προτεινόμενη προσέγγιση, δηλαδή ο τρόπος με τον οποίο οι παραπάνω τεχνικές και εργαλεία συνδυάζονται σε μια ολοκληρωμένη διαδικασία έρευνας ευπαθειών.

Τα τέσσερα κεφάλαια της εργασίας έχουν την ακόλουθη δομή :

- Στο **πρώτο κεφάλαιο**, με τίτλο «*Εισαγωγή*», αναφέρονται τα κίνητρα και οι σύγχρονες εξελίξεις που οδήγησαν στην εκπόνηση της παρούσας εργασίας. Επίσης, παρουσιάζεται το πρόβλημα της αυτοματοποίησης της έρευνας ευπαθειών σε εφαρμογές Android και προσδιορίζονται οι βασικές προκλήσεις που προκύπτουν από την αξιοποίηση Μεγάλων Γλωσσικών Μοντέλων για τον σκοπό αυτό. Τέλος, παρουσιάζεται η δομή και η οργάνωση της εργασίας.
- Στο **δεύτερο κεφάλαιο**, με τίτλο «*Ασφάλεια στο Android OS*», παρουσιάζεται αρχικά η αρχιτεκτονική της πλατφόρμας Android και εξετάζεται η εξέλιξη του μοντέλου ασφάλειάς της μέσα από σημαντικούς μηχανισμούς και περιορισμούς που εισήχθησαν σε διαδοχικές εκδόσεις του λειτουργικού συστήματος. Στη συνέχεια, παρουσιάζονται τα βασικά application components και τα σημεία εισόδου που συνθέτουν την επιφάνεια επίθεσης μιας εφαρμογής Android. Τέλος, αναλύονται επιλεγμένες κατηγορίες ευπαθειών που παρουσιάζουν ιδιαίτερο ενδιαφέρον στο πλαίσιο της παρούσας εργασίας, με έμφαση τόσο στον τρόπο με τον οποίο μπορούν να προκύψουν όσο και στους ελέγχους ασφαλείας που παραμένουν ευθύνη του developer.
- Στο **τρίτο κεφάλαιο**, με τίτλο «*Προτεινόμενη προσέγγιση*», παρουσιάζεται ο σχεδιασμός της μεθοδολογίας που αναπτύχθηκε για την αξιολόγηση ασφάλειας Android εφαρμογών με τη χρήση LLM-based agents. Εξετάζονται οι βασικές σχεδιαστικές επιλογές, η μετάβαση από την αρχική

ιδέα ενός εξειδικευμένου classifier προς ένα agentic περιβάλλον και τα εργαλεία στατικής και δυναμικής ανάλυσης που ενσωματώθηκαν στο σύστημα. Στη συνέχεια περιγράφεται ο τρόπος με τον οποίο ο agent συντονίζει τη διερεύνηση, διατηρεί την κατάσταση της αξιολόγησης και χρησιμοποιεί τα αποτελέσματα των επιμέρους εργαλείων για τη διαμόρφωση και επαλήθευση υποθέσεων. Το κεφάλαιο ολοκληρώνεται με την παρουσίαση της συνολικής αρχιτεκτονικής, όπου αναλύεται η επαναληπτική σχέση μεταξύ στατικής και δυναμικής ανάλυσης, η λειτουργία του Android Client-Side Assessment Skill και τα κριτήρια με τα οποία μια υπό εξέταση διαδρομή μπορεί να οδηγήσει σε τεκμηριωμένο finding.

- Στο **τέταρτο κεφάλαιο**, με τίτλο *«Αποτελέσματα και Συμπεράσματα»*, παρουσιάζονται τα αποτελέσματα από την εφαρμογή της προτεινόμενης προσέγγισης και αξιολογείται η πρακτική της λειτουργία. Η OVAA χρησιμοποιείται ως το κύριο δημόσια αναπαραγώγιμο περιβάλλον αξιολόγησης, καθώς επιτρέπει την παρουσίαση των findings και των αντίστοιχων τεχνικών τεκμηρίων χωρίς περιορισμούς δημοσιοποίησης. Παράλληλα παρουσιάζονται συγκεντρωτικά αποτελέσματα από την εφαρμογή της ίδιας προσέγγισης σε πραγματικές Android εφαρμογές στο πλαίσιο vulnerability disclosure και bug bounty programs, για τις οποίες οι επιμέρους τεχνικές λεπτομέρειες δεν μπορούν να δημοσιοποιηθούν. Στο τέλος του κεφαλαίου συνοψίζεται η συνεισφορά της εργασίας.

1.4 Σύνοψη

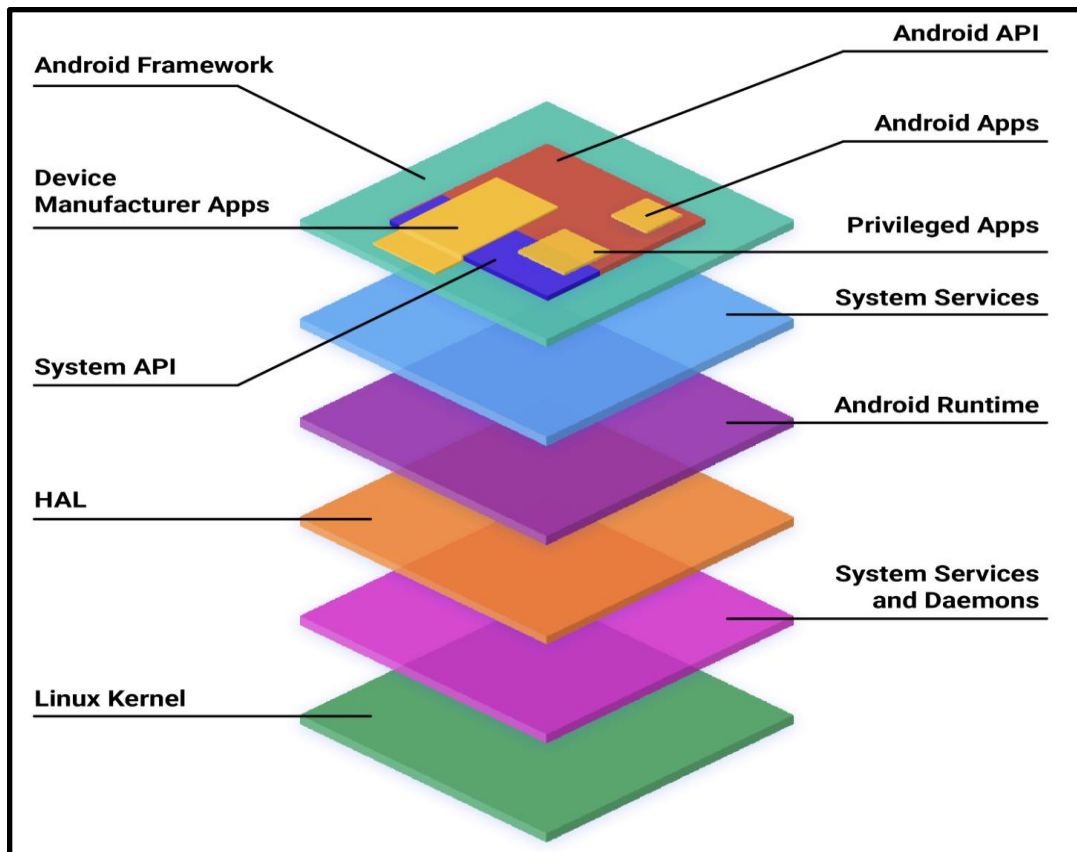
Στο κεφάλαιο αυτό διαμορφώθηκε το γενικό πλαίσιο της παρούσας εργασίας, με έμφαση στην ανάγκη αυτοματοποίησης της έρευνας ευπαθειών σε εφαρμογές Android και στις δυσκολίες που προκύπτουν κατά την ανάλυση πραγματικών εφαρμογών. Παράλληλα, εξετάστηκαν οι δυνατότητες και οι περιορισμοί της αξιοποίησης Μεγάλων Γλωσσικών Μοντέλων σε μια τέτοια διαδικασία, ιδιαίτερα ως προς τη διαχείριση των ευρημάτων και την ανάγκη επαλήθευσής τους. Τέλος, προσδιορίστηκε ο στόχος της παρούσας εργασίας και παρουσιάστηκε η οργάνωση των κεφαλαίων που ακολουθούν.

2. Ασφάλεια στο Android OS

Στο παρόν κεφάλαιο παρουσιάζεται η αρχιτεκτονική του Android και οι βασικοί μηχανισμοί πάνω στους οποίους στηρίζεται το μοντέλο ασφαλείας του λειτουργικού συστήματος. Αρχικά αναλύεται η πολυεπίπεδη αρχιτεκτονική της πλατφόρμας, η οποία εκτείνεται από το επίπεδο των εφαρμογών και το Android Framework έως τα system services, το Android Runtime, το Hardware Abstraction Layer και τον Linux kernel. Στη συνέχεια εξετάζονται οι κύριοι μηχανισμοί ασφαλείας που χρησιμοποιούνται για την απομόνωση των εφαρμογών, τον έλεγχο της πρόσβασης σε πόρους και την προστασία του συστήματος από μη εξουσιοδοτημένες ενέργειες. Τέλος, παρουσιάζονται τα σημεία στα οποία η ασφάλεια εξαρτάται και από την ορθή χρήση των παρεχόμενων μηχανισμών από τον developer, καθώς οι λανθασμένες ρυθμίσεις ή επιλογές υλοποίησης μπορούν να περιορίσουν την αποτελεσματικότητα των προστατευτικών μηχανισμών του Android.

2.1 Αρχιτεκτονική του Android

Το Android έχει σχεδιαστεί ως μια πολυεπίπεδη αρχιτεκτονική, στην οποία κάθε επίπεδο επιτελεί διακριτό ρόλο και επικοινωνεί με τα υπόλοιπα μέσω καθορισμένων διεπαφών. Η δομή αυτή επιτρέπει στις εφαρμογές να αξιοποιούν λειτουργίες της συσκευής χωρίς να εξαρτώνται άμεσα από τις λεπτομέρειες της υποκείμενης υλοποίησης. Έτσι, μια εφαρμογή μπορεί να ζητήσει πρόσβαση στην κάμερα μέσω του Android Framework, ενώ η επικοινωνία με το αντίστοιχο hardware πραγματοποιείται μέσω των system services και του Hardware Abstraction Layer, χωρίς να απαιτείται γνώση του συγκεκριμένου driver ή της υλοποίησης της κάμερας στη συσκευή [22].



Εικόνα 3: Αρχιτεκτονική της στοίβας λογισμικού του Android Open Source Project.

Πηγή: [22, Fig.1].

- **Linux kernel :** Στη βάση της πλατφόρμας βρίσκεται ο *Linux kernel*, ο οποίος παρέχει τις θεμελιώδεις λειτουργίες του λειτουργικού συστήματος, όπως την διαχείριση των διεργασιών και νημάτων, την εικονική μνήμη, την δικτύωση, το filesystem και τους device drivers. Το Android αξιοποιεί υπάρχοντες μηχανισμούς του Linux, ενώ παράλληλα προσθέτει νέους μηχανισμούς και τροποποιήσεις που απαιτούνται από την πλατφόρμα. Ιδιαίτερα σημαντικός είναι επίσης ο μηχανισμός Binder, ο οποίος χρησιμοποιείται ως βασικός μηχανισμός επικοινωνίας μεταξύ διεργασιών (*Inter-Process Communication – IPC*) στο Android [22].
- **Hardware Abstraction Layer (HAL) :** Πάνω από τον kernel βρίσκεται το *Hardware Abstraction Layer (HAL)*, το οποίο παρέχει τυποποιημένες διεπαφές προς τις δυνατότητες του hardware της συσκευής. Μέσω του HAL, υψηλότερα επίπεδα της πλατφόρμας μπορούν να χρησιμοποιούν λειτουργίες όπως η κάμερα, το Bluetooth ή τον ήχο χωρίς να εξαρτώνται άμεσα από την υλοποίηση του αντίστοιχου device driver. Οι constructors υλοποιούν τις αντίστοιχες HAL interfaces για το

hardware της συσκευής τους, ενώ το Android Framework μπορεί να χρησιμοποιεί τις ίδιες αφαιρέσεις ανεξάρτητα από τη συγκεκριμένη συσκευή [22].

- **Native libraries & Android Runtime** : Στο επόμενο επίπεδο βρίσκονται βασικές native βιβλιοθήκες, μεγάλο μέρος των οποίων είναι υλοποιημένο σε C και C++. Σε αυτές περιλαμβάνονται βιβλιοθήκες για γραφικά, πολυμέσα, δικτύωση, αποθήκευση δεδομένων και άλλες λειτουργίες της πλατφόρμας. Οι εφαρμογές μπορούν επίσης να περιλαμβάνουν δικό τους native κώδικα, ο οποίος μεταγλωττίζεται μέσω του Android Native Development Kit (NDK) σε native shared libraries (.so) και μπορεί να κληθεί από Java ή Kotlin κώδικα μέσω του Java Native Interface (JNI). Η εκτέλεση του managed κώδικα των εφαρμογών πραγματοποιείται από το Android Runtime (ART). Ο κώδικας Java ή Kotlin μετατρέπεται κατά τη διαδικασία build στη μορφή Dalvik Executable (DEX), η οποία περιλαμβάνεται στο APK της εφαρμογής. Στις πρώτες εκδόσεις του Android, η εκτέλεση του DEX bytecode πραγματοποιούνταν από τη virtual machine *Dalvik*, ενώ από το Android 5.0 το Dalvik αντικαταστάθηκε από το *ART* ως κύριο runtime της πλατφόρμας. Το ART υποστηρίζει διαφορετικές μορφές εκτέλεσης και μεταγλώττισης, μεταξύ των οποίων interpretation, Just-In-Time (JIT) και Ahead-Of-Time (AOT) compilation [23], [24]. Επειδή το APK περιέχει DEX bytecode και όχι τον αρχικό Java ή Kotlin κώδικα, κατά την αντίστροφη ανάλυση ο κώδικας αυτός μπορεί είτε να αποσυμπληθεί σε Java-like αναπαράσταση είτε να αποσυναρμολογηθεί στη χαμηλότερου επιπέδου αναπαράσταση *Smali*.
- **Application Framework** : Το Application Framework παρέχει τις υψηλότερου επιπέδου υπηρεσίες και APIs που χρησιμοποιούνται από τις εφαρμογές. Μέσω αυτού εκτίθενται λειτουργίες για τη διαχείριση Activities, παραθύρων, packages, notifications, τοποθεσίας, τηλεφωνίας και πολλών άλλων πόρων της πλατφόρμας. Σε αρκετές περιπτώσεις, η κλήση ενός framework API δεν εκτελεί ολόκληρη τη λειτουργία μέσα στη διεργασία της εφαρμογής, αλλά οδηγεί μέσω *Binder IPC* σε ένα αντίστοιχο *system service*, το οποίο εκτελείται με διαφορετικά δικαιώματα και αναλαμβάνει την πραγματική λειτουργία [22]. Οι εφαρμογές τρίτων αναπτύσσονται μέσω του *Android SDK*, το οποίο παρέχει τις δημόσιες API definitions και τα εργαλεία που απαιτούνται για τη μεταγλώττιση και συσκευασία της εφαρμογής. Το API που είναι διαθέσιμο κατά το build καθορίζεται από το αντίστοιχο Android API level. Η ύπαρξη μιας κλάσης ή μεθόδου στον πηγαίο κώδικα του AOSP δεν σημαίνει απαραίτητα ότι αυτή αποτελεί δημόσιο API διαθέσιμο στις εφαρμογές τρίτων.
- **Applications** : Στο υψηλότερο επίπεδο της αρχιτεκτονικής βρίσκονται οι εφαρμογές, στις οποίες περιλαμβάνονται τόσο εκείνες που εγκαθίστανται από τον χρήστη όσο και όσες παρέχονται εξ αρχής ως μέρος του system image. Οι εφαρμογές τρίτων βασίζονται κυρίως στο δημόσιο Android API, ενώ ορισμένες προεγκατεστημένες ή προνομιούχες εφαρμογές μπορούν, υπό συγκεκριμένες προϋποθέσεις, να χρησιμοποιούν πρόσθετα System APIs ή να διαθέτουν privileged

permissions. Η διαφοροποίηση αυτή αντανακλά το μοντέλο εμπιστοσύνης και ελέγχου πρόσβασης που εφαρμόζει το Android [22].

Κατά την εκκίνηση των εφαρμογών, σημαντικό ρόλο διαδραματίζει η διεργασία *Zygote*, η οποία εκκινείται από το `init` κατά την αρχικοποίηση του λειτουργικού συστήματος και λειτουργεί ως βάση για τη δημιουργία των περισσότερων διεργασιών εφαρμογών για το αντίστοιχο ABI. Όταν απαιτείται η εκκίνηση μιας νέας διεργασίας, το `system_server` επικοινωνεί με το *Zygote*, το οποίο δημιουργεί τη νέα διεργασία μέσω `fork()`. Στη συνέχεια εφαρμόζονται τα χαρακτηριστικά που αντιστοιχούν στη συγκεκριμένη εφαρμογή, όπως το UID και το κατάλληλο `runtime` περιβάλλον. Η αρχιτεκτονική αυτή επιτρέπει την επαναχρησιμοποίηση προφορτωμένων κλάσεων και πόρων μεταξύ των διεργασιών, μειώνοντας το κόστος δημιουργίας τους και επιταχύνοντας την εκκίνηση των εφαρμογών [25].

2.2 Εξέλιξη του μοντέλου ασφάλειας του Android

Το μοντέλο ασφάλειας του Android δεν διαμορφώθηκε εξ αρχής στη σημερινή του μορφή, αλλά εξελίχθηκε σταδιακά μέσα από διαδοχικές ενισχύσεις της πλατφόρμας. Οι πρώτες εκδόσεις βασίστηκαν κυρίως στους μηχανισμούς απομόνωσης του Linux, με διαφορετικό αναγνωριστικό χρήστη (UID) για κάθε εφαρμογή, στο `permission model` του συστήματος και στην κρυπτογραφική υπογραφή των εφαρμογών, μέσω της οποίας διατηρείται η σχέση εμπιστοσύνης μεταξύ διαδοχικών εκδόσεων του ίδιου πακέτου. Με την πάροδο του χρόνου προστέθηκαν επιπλέον μηχανισμοί προστασίας σε χαμηλότερα επίπεδα του συστήματος, όπως ο υποχρεωτικός έλεγχος πρόσβασης, ο περιορισμός των `system calls`, η ενίσχυση της προστασίας απέναντι σε `memory-corruption exploits` και η επαλήθευση της ακεραιότητας του λειτουργικού συστήματος [26].

Παράλληλα, εξελίχθηκαν και οι μηχανισμοί που επηρεάζουν άμεσα τη συμπεριφορά των εφαρμογών, με χαρακτηριστικά παραδείγματα την εισαγωγή των `runtime permissions`, τους αυστηρότερους περιορισμούς στην κοινόχρηστη αποθήκευση και τη σταδιακή ενίσχυση των κανόνων που διέπουν την έκθεση και την επικοινωνία των `application components` [26], [27]. Οι σημαντικότερες από αυτές τις εξελίξεις παρουσιάζονται στη συνέχεια του κεφαλαίου.

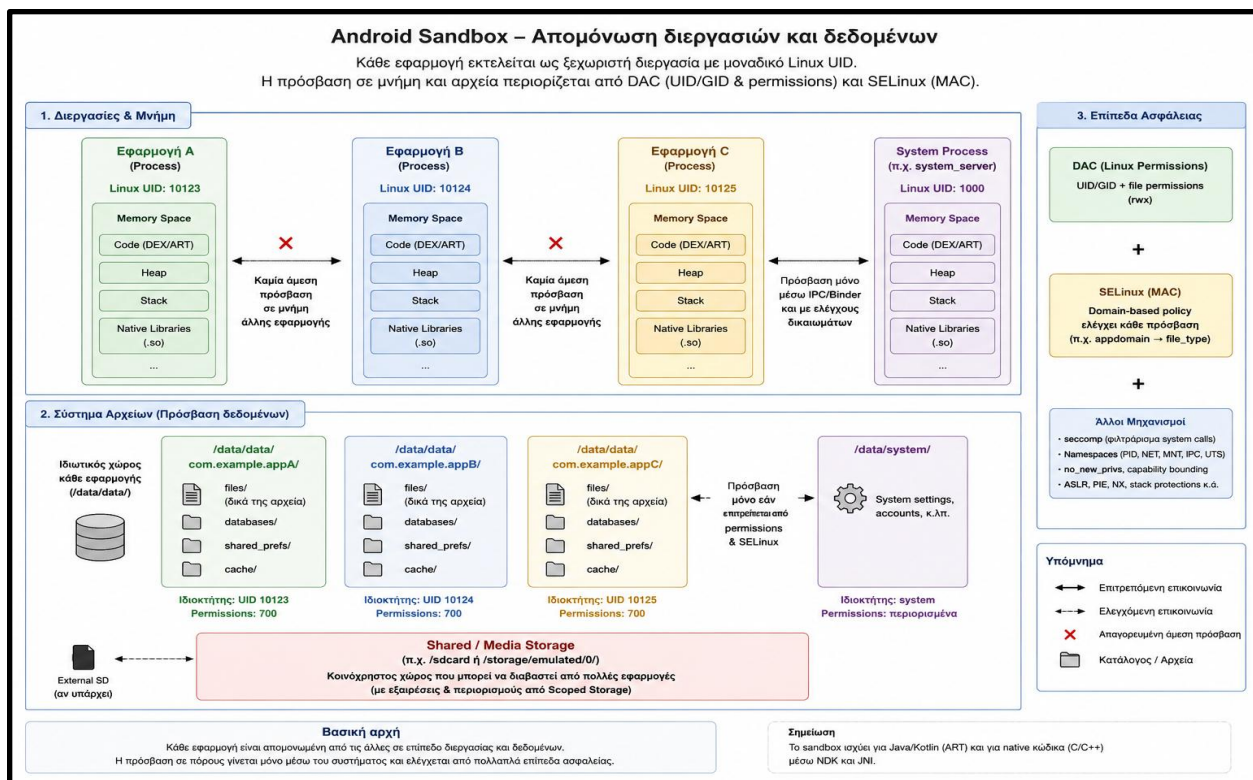
2.2.1 Μηχανισμός απομόνωσης εφαρμογών

Στις πρώτες εκδόσεις του Android, η απομόνωση των εφαρμογών βασίστηκε σε μεγάλο βαθμό στο μοντέλο χρηστών του Linux. Κατά την εγκατάσταση, κάθε εφαρμογή λαμβάνει ένα μοναδικό Linux User ID (UID), το οποίο χρησιμοποιείται από τον `kernel` για την αναγνώριση και τον διαχωρισμό της από τις υπόλοιπες εφαρμογές του συστήματος. Τα `components` της εκτελούνται, κατά κανόνα, σε ξεχωριστή διεργασία με το αντίστοιχο UID, με αποτέλεσμα οι περιορισμοί πρόσβασης να επιβάλλονται σε επίπεδο λειτουργικού

συστήματος [2]. Το AOSP περιγράφει τον μηχανισμό αυτό ως *kernel-level Application Sandbox*, καθώς η απομόνωση δεν εξαρτάται από το Android Runtime ή από τη γλώσσα υλοποίησης της εφαρμογής, αλλά εφαρμόζεται απευθείας από τον Linux kernel [2].

Η απομόνωση μεταξύ των εφαρμογών επεκτείνεται στη διαχείριση των διεργασιών, της μνήμης και των δεδομένων τους. Κάθε διεργασία διαθέτει ξεχωριστό εικονικό χώρο διευθύνσεων, μέσα στον οποίο χαρτογραφούνται ο κώδικας και οι περιοχές μνήμης που χρησιμοποιούνται κατά την εκτέλεσή της. Έτσι, μια εφαρμογή δεν μπορεί να προσπελάσει άμεσα τη μνήμη άλλης διεργασίας απλώς γνωρίζοντας μια διεύθυνση, καθώς η πρόσβαση ελέγχεται από τον kernel και κάθε διεργασία λειτουργεί μέσα στον δικό της εικονικό χώρο. Από προεπιλογή, τα components μιας εφαρμογής εκτελούνται στην ίδια διεργασία, αν και το Android επιτρέπει τη μεταφορά συγκεκριμένων components σε ξεχωριστές διεργασίες μέσω αντίστοιχης δήλωσης στο Manifest [28].

Αντίστοιχα, το UID χρησιμοποιείται για τον έλεγχο της πρόσβασης στα ιδιωτικά δεδομένα της εφαρμογής μέσω των μηχανισμών Discretionary Access Control (DAC) του Linux. Κάθε εφαρμογή διαθέτει app-specific χώρο στην εσωτερική αποθήκευση, στον οποίο μπορεί να αποθηκεύει αρχεία, βάσεις δεδομένων, προτιμήσεις και κρυφή μνήμη (cache). Σε μια τυπική εγκατάσταση ο χώρος αυτός αντιστοιχεί σε διαδρομή της μορφής `/data/data/<user_id>/<package_name>/`. Τα αρχεία αυτά ανήκουν στην ταυτότητα της αντίστοιχης εφαρμογής και το λειτουργικό σύστημα εμποδίζει άλλες εφαρμογές να τα προσπελάσουν απευθείας [2], [29]. Ωστόσο, η ίδια η εφαρμογή, δεν χρειάζεται να ζητήσει κάποια άδεια αποθήκευσης (runtime storage permission) για να διαβάσει ή να γράψει στους δικούς της εσωτερικούς καταλόγους της εφαρμογής (internal app-specific directories). Όταν απαιτείται επικοινωνία ή ανταλλαγή δεδομένων μεταξύ εφαρμογών, αυτή πρέπει να πραγματοποιηθεί μέσω μηχανισμών που παρέχει το λειτουργικό σύστημα, όπως Binder IPC, Intents, Content Providers ή ελεγχόμενη κοινή αποθήκευση.



Εικόνα 4 : Το sandbox στο Android.

Ο αρχικός μηχανισμός απομόνωσης του Android βασίστηκε σε μεγάλο βαθμό στο Discretionary Access Control (DAC) του Linux, σε συνδυασμό με τη χρήση διαφορετικών UIDs, ξεχωριστών διεργασιών και δικαιωμάτων πρόσβασης στο filesystem. Στο συγκεκριμένο μοντέλο, η πρόσβαση σε έναν πόρο καθορίζεται από την ταυτότητα του χρήστη ή της ομάδας και από τα permissions που έχουν οριστεί για το αντίστοιχο αντικείμενο. Η προσέγγιση αυτή παρουσιάζει περιορισμούς σε περιπτώσεις όπου μια διεργασία αποκτά αυξημένα privileges ή όταν παραβιάζεται κάποιος privileged system service, καθώς οι υπάρχοντες μηχανισμοί δεν επαρκούν πάντα για τον περιορισμό της περαιτέρω πρόσβασης. Από το Android 4.3, η πλατφόρμα ενισχύθηκε με το Security-Enhanced Linux (SELinux), το οποίο εισάγει πολιτικές Mandatory Access Control (MAC) και επιβάλλει πρόσθετους περιορισμούς ανεξάρτητα από τα δικαιώματα που απορρέουν από το παραδοσιακό μοντέλο DAC [30].

Στο SELinux, η απόφαση πρόσβασης δεν βασίζεται αποκλειστικά στο UID της διεργασίας ή στα Unix permissions ενός αρχείου, αλλά στην πολιτική που έχει οριστεί για τα security contexts των επιμέρους οντοτήτων του συστήματος. Οι διεργασίες εκτελούνται μέσα σε συγκεκριμένα SELinux domains και οι πόροι χαρακτηρίζονται από αντίστοιχα security types, ώστε η πολιτική να καθορίζει ποιες ενέργειες επιτρέπονται μεταξύ τους [30]. Για παράδειγμα, μια εφαρμογή τρίτου μπορεί να εκτελείται στο domain

untrusted_app, ενώ τα δεδομένα της εφαρμογής να χαρακτηρίζονται με type app_data_file. Όταν η διεργασία επιχειρεί να διαβάσει ένα τέτοιο αρχείο, το SELinux ελέγχει αν η ενεργή policy περιλαμβάνει κανόνα που επιτρέπει την ενέργεια read για τον συγκεκριμένο συνδυασμό domain και type. Σε απλοποιημένη μορφή, ένας τέτοιος κανόνας μπορεί να γραφτεί ως:

```
allow untrusted_app app_data_file:file { read };
```

Ο κανόνας δηλώνει ότι διεργασίες του domain untrusted_app επιτρέπεται να εκτελούν την ενέργεια read σε αρχεία με type app_data_file, εφόσον η συγκεκριμένη άδεια περιλαμβάνεται στην ενεργή SELinux policy [30].

Βασικό χαρακτηριστικό του SELinux είναι ότι οι κανόνες πρόσβασης επιβάλλονται κεντρικά από την πολιτική του συστήματος και ακολουθούν τη λογική του *default deny*, σύμφωνα με την οποία μια ενέργεια επιτρέπεται μόνο όταν υπάρχει αντίστοιχος κανόνας που την προβλέπει. Με τον τρόπο αυτό, μια πρόσβαση μπορεί να απορριφθεί ακόμη και όταν θα ήταν επιτρεπτή από τα συμβατικά Linux permissions, ενώ οι ίδιοι περιορισμοί εξακολουθούν να εφαρμόζονται και σε διεργασίες με αυξημένα privileges. Η προσέγγιση αυτή συμβάλλει στον περιορισμό της πρόσβασης που μπορεί να αποκτήσει ένα compromised system component και μειώνει την έκταση των ενεργειών που μπορεί να εκτελέσει μετά την παραβίασή του [30].

Η ενσωμάτωση του SELinux στο Android πραγματοποιήθηκε σταδιακά. Στο Android 4.3 χρησιμοποιήθηκε αρχικά σε *permissive mode*, ώστε οι παραβιάσεις της πολιτικής να καταγράφονται χωρίς να εμποδίζεται η εκτέλεση των αντίστοιχων ενεργειών, ενώ στο Android 4.4 ενεργοποιήθηκε επιλεκτικά το *enforcing mode* για κρίσιμα system domains, όπως τα *installd*, *netd*, *vold* και *zygote*. Από το Android 5.0 η επιβολή της SELinux policy επεκτάθηκε σε ολόκληρο το σύστημα και στις επόμενες εκδόσεις ενισχύθηκε με πρόσθετους περιορισμούς στην πρόσβαση σε ευαίσθητες διεπαφές και system resources [30]. Η απομόνωση επεκτάθηκε σταδιακά και στην προστασία των ιδιωτικών δεδομένων των εφαρμογών. Οι πολιτικές του SELinux προσαρμόστηκαν ώστε να λαμβάνουν υπόψη τον διαχωρισμό μεταξύ διαφορετικών χρηστών της συσκευής, ενώ για εφαρμογές με *targetSdkVersion* ≥ 24 τα προεπιλεγμένα DAC permissions του private home directory έγιναν αυστηρότερα, από **751** σε **700**. Στο Android 9 εισήχθη επιπλέον per-app SELinux isolation για μη προνομιούχες εφαρμογές με *targetSdkVersion* ≥ 28 , ενισχύοντας την προστασία των private data directories απέναντι σε πρόσβαση από άλλες εφαρμογές [2].

Παρότι το SELinux περιορίζει ποιος πόρος μπορεί να προσπελαστεί και με ποιον τρόπο, μια εφαρμογή εξακολουθεί να επικοινωνεί με τον Linux kernel μέσω system calls. Τα system calls αποτελούν τη βασική διεπαφή μέσω της οποίας μια userspace διεργασία ζητά λειτουργίες από τον kernel, όπως άνοιγμα αρχείων, δημιουργία mappings μνήμης ή επικοινωνία μέσω Binder. Καθώς κάθε επιπλέον syscall που είναι

διαθέσιμο σε μια μη έμπιστη εφαρμογή αυξάνει δυνητικά την επιφάνεια επίθεσης του kernel, στο Android 8.0 προστέθηκε ένα επιπλέον επίπεδο περιορισμού μέσω *seccomp-bpf* [2].

Το *seccomp* (*secure computing*) χρησιμοποιείται για τον περιορισμό των Linux system calls που μπορεί να εκτελέσει μια διεργασία. Στο Android 8.0, το σχετικό φίλτρο εγκαθίσταται στο Zygote και κληρονομείται από τις διεργασίες εφαρμογών που δημιουργούνται από αυτό, επιτρέποντας στον kernel να απορρίπτει system calls που δεν θεωρούνται απαραίτητα για τη λειτουργία τους [2], [31]. Όταν μια διεργασία επιχειρεί να εκτελέσει system call που δεν επιτρέπεται από την ενεργή seccomp policy, ο kernel εφαρμόζει την ενέργεια που έχει οριστεί για τη συγκεκριμένη περίπτωση, η οποία μπορεί να περιλαμβάνει και την αποστολή σήματος SIGSYS [31].

2.2.2 Μοντέλο δικαιωμάτων του Android

Ένας ακόμη βασικός άξονας εξέλιξης του μοντέλου ασφάλειας του Android αφορά τον τρόπο με τον οποίο οι εφαρμογές αποκτούν πρόσβαση σε προστατευμένα δεδομένα και λειτουργίες του συστήματος. Η πρόσβαση αυτή ελέγχεται μέσω του συστήματος permissions, το οποίο καθορίζει ποιες δυνατότητες βρίσκονται έξω από το βασικό sandbox της εφαρμογής και υπό ποιες προϋποθέσεις μπορούν να χρησιμοποιηθούν. Οι απαιτούμενες άδειες δηλώνονται στο AndroidManifest.xml μέσω στοιχείων όπως το `<uses-permission>`, ενώ η τελική απόδοσή τους εξαρτάται από τον τύπο της άδειας και τους κανόνες που εφαρμόζει η αντίστοιχη έκδοση του Android [32].

Τα permissions διακρίνονται ανάλογα με το επίπεδο προστασίας που παρέχουν. Τα normal permissions αφορούν λειτουργίες χαμηλότερου κινδύνου και αποδίδονται αυτόματα από το σύστημα, ενώ τα dangerous permissions σχετίζονται με πρόσβαση σε πιο ευαίσθητα δεδομένα ή δυνατότητες της συσκευής και απαιτούν αυστηρότερο έλεγχο από το λειτουργικό σύστημα. Ξεχωριστή κατηγορία αποτελούν τα signature permissions, τα οποία αποδίδονται όταν η εφαρμογή που ζητά την άδεια έχει υπογραφεί με certificate που ικανοποιεί τις απαιτήσεις της άδειας, συνήθως το ίδιο με εκείνο της εφαρμογής ή του συστήματος που την έχει ορίσει [32].

Στις πρώτες εκδόσεις του Android, τα permissions που ζητούσε μια εφαρμογή εγκρίνονταν κατά την εγκατάστασή της, το οποίο απαιτούσε από τον χρήστη να αποδεχθεί εκ των προτέρων το σύνολο των δικαιωμάτων που είχαν δηλωθεί. Με το Android 6.0 (API level 23) εισήχθη το μοντέλο των runtime permissions, με βάση το οποίο τα permissions με protection level dangerous αποδίδονται κατά την εκτέλεση της εφαρμογής και έπειτα από ρητή έγκριση του χρήστη μέσω system-controlled permission dialog [26], [33]. Στο Android 11 προστέθηκαν τα *one-time permissions* και η αυτόματη επαναφορά δικαιωμάτων για εφαρμογές που παραμένουν ανενεργές για μεγάλο χρονικό διάστημα [62]. Οι αλλαγές

αυτές δείχνουν τη σταδιακή μετατόπιση του Android προς ένα μοντέλο πρόσβασης στο οποίο τα permissions παραχωρούνται με μεγαλύτερο έλεγχο ως προς τη διάρκεια και το εύρος τους.

2.2.3 Προστασία δεδομένων και αποθηκευτικού χώρου

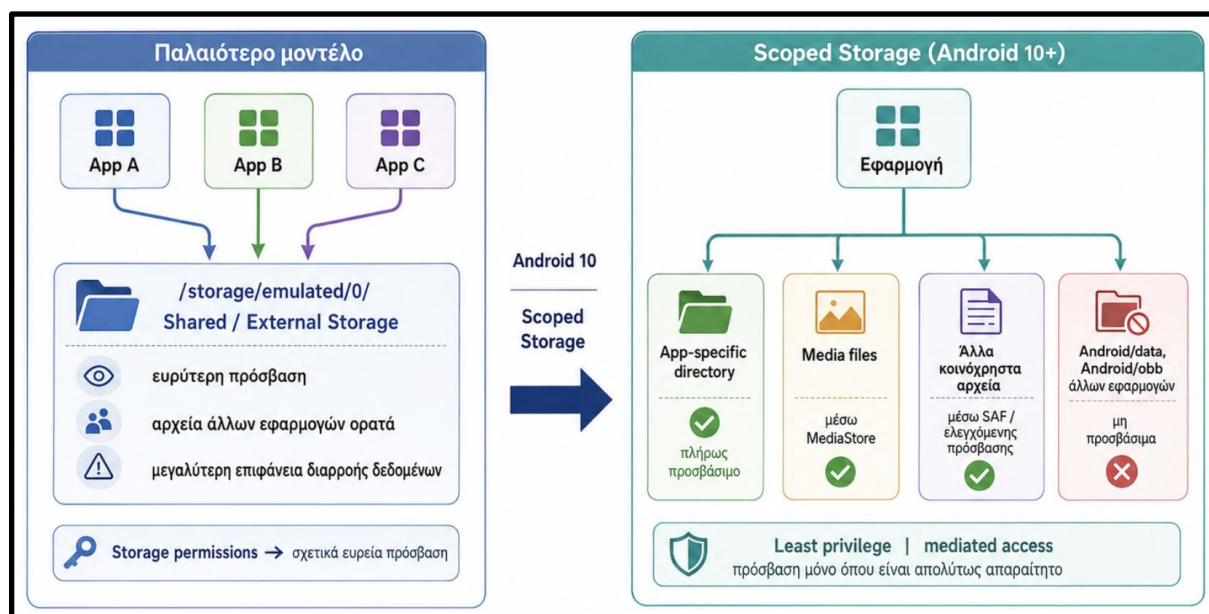
Πέρα από την προστασία των ιδιωτικών καταλόγων μιας εφαρμογής, σημαντικό μέρος της εξέλιξης του Android αφορά τον τρόπο με τον οποίο τα δεδομένα αποθηκεύονται και διαμοιράζονται εκτός του app-specific χώρου. Ιδιαίτερη σημασία είχε η εξωτερική αποθήκευση (*shared/external storage*), η οποία χρησιμοποιείται για δεδομένα που μπορούν να είναι προσβάσιμα από περισσότερες εφαρμογές. Ο όρος *external storage* δεν αναφέρεται απαραίτητα σε αφαιρούμενη SD card, καθώς στις σύγχρονες συσκευές μπορεί να υλοποιείται ως *emulated storage* πάνω στην εσωτερική μνήμη της συσκευής [29].

Ανάλογη εξέλιξη παρατηρήθηκε και στον μηχανισμό διαμοιρασμού αρχείων μεταξύ εφαρμογών. Σε παλαιότερες εκδόσεις, μια εφαρμογή μπορούσε να κοινοποιήσει την πραγματική διαδρομή ενός αρχείου μέσω URI της μορφής `file://`, με αποτέλεσμα η πρόσβαση να εξαρτάται άμεσα από τα permissions του filesystem. Από το Android 7.0 (API level 24), η έκθεση τέτοιων URIs εκτός της εφαρμογής περιορίστηκε για εφαρμογές που στοχεύουν API level 24 ή νεότερο, και η προσπάθεια κοινοποίησής τους μπορεί να οδηγήσει σε `FileUriExposedException` [34].

Η διαμοίραση αρχείων στο Android βασίζεται πλέον σε ένα περισσότερο ελεγχόμενο μοντέλο, το οποίο αξιοποιεί `content://` URIs και τους μηχανισμούς των Content Providers. Χαρακτηριστικό παράδειγμα αποτελεί ο `FileProvider`, ο οποίος είναι ένας εξειδικευμένος `ContentProvider` μέσω του οποίου η εφαρμογή μπορεί να εκθέσει μόνο προκαθορισμένες περιοχές του αποθηκευτικού της χώρου, χωρίς να αποκαλύπτει την πραγματική διαδρομή του αρχείου στο filesystem. Η πρόσβαση στο περιεχόμενο παραχωρείται προσωρινά μέσω του αντίστοιχου URI και μπορεί να περιοριστεί σε δικαιώματα ανάγνωσης ή εγγραφής με flags όπως τα `FLAG_GRANT_READ_URI_PERMISSION` και `FLAG_GRANT_WRITE_URI_PERMISSION` [127]. Με αυτόν τον τρόπο, ο διαμοιρασμός πραγματοποιείται χωρίς άμεση έκθεση της πραγματικής θέσης του αρχείου και με σαφέστερο έλεγχο της πρόσβασης που παρέχεται στην εφαρμογή-παραλήπτη.

Σε ευρύτερο επίπεδο, η πρόσβαση στο *shared/external storage* αποτέλεσε διαχρονικά σημαντικό ζήτημα ιδιωτικότητας και ασφάλειας, καθώς το παλαιότερο μοντέλο επέτρεπε σε εφαρμογές με τα κατάλληλα storage permissions αρκετά εκτεταμένη πρόσβαση σε κοινόχρηστα αρχεία. Για τον περιορισμό αυτής της συμπεριφοράς, το Android 10 (API level 29) εισήγαγε το *Scoped Storage*, περιορίζοντας την πρόσβαση κάθε εφαρμογής σε σαφέστερα καθορισμένες περιοχές του αποθηκευτικού χώρου [26], [29].

Η πρόσβαση σε κοινόχρηστα δεδομένα πραγματοποιείται πλέον μέσω ελεγχόμενων μηχανισμών της πλατφόρμας, όπως το MediaStore και το Storage Access Framework, ενώ ο app-specific χώρος της εφαρμογής παραμένει άμεσα προσβάσιμος από την ίδια [29]. Το Android 11 ενίσχυσε περαιτέρω το μοντέλο απομόνωσης του shared storage, περιορίζοντας την πρόσβαση των εφαρμογών σε app-specific directories που ανήκουν σε άλλες εφαρμογές και αποκλείοντας μέσω του Storage Access Framework την επιλογή προστατευμένων περιοχών όπως τα Android/data/ και Android/obb/. Για εφαρμογές που απαιτούν εκτεταμένη διαχείριση αρχείων προβλέπεται η ειδική permission `MANAGE_EXTERNAL_STORAGE`, ώστε η ευρεία πρόσβαση στο filesystem να παρέχεται μόνο σε περιπτώσεις όπου δικαιολογείται από τη λειτουργία της εφαρμογής και να μην αποτελεί την προεπιλεγμένη συμπεριφορά [35] , [128].



Εικόνα 5 : Εξέλιξη της πρόσβασης στην κοινόχρηστη αποθήκευση με την εισαγωγή του Scoped Storage.

2.3 Σύγχρονοι μηχανισμοί ασφάλειας σε επίπεδο εφαρμογών

Πέρα από τους βασικούς μηχανισμούς απομόνωσης, δικαιωμάτων και προστασίας δεδομένων που περιγράφηκαν προηγουμένως, η ασφάλεια του Android εξαρτάται σε μεγάλο βαθμό από τον τρόπο με τον οποίο μια εφαρμογή αλληλεπιδρά με το λειτουργικό σύστημα, άλλες εφαρμογές και εξωτερικές πηγές δεδομένων. Οι Android εφαρμογές οργανώνονται γύρω από τέσσερις βασικούς τύπους application components [129] :

- **Activities** : οι οποίες αντιπροσωπεύουν συνήθως ένα σημείο αλληλεπίδρασης του χρήστη με την εφαρμογή και μπορούν να ενεργοποιηθούν για την παρουσίαση μιας συγκεκριμένης λειτουργίας ή οθόνης.
- **Services** : τα οποία χρησιμοποιούνται για την εκτέλεση λειτουργιών χωρίς άμεσο γραφικό περιβάλλον.
- **Broadcast Receivers** : οι οποίοι επιτρέπουν σε μια εφαρμογή να λαμβάνει και να επεξεργάζεται broadcast events που προέρχονται από το Android system ή από άλλες εφαρμογές.
- **Content Providers** : οι οποίοι παρέχουν έναν δομημένο μηχανισμό πρόσβασης και διαμοιρασμού δεδομένων μεταξύ εφαρμογών.

Η επικοινωνία μεταξύ πολλών application components πραγματοποιείται μέσω Intents, τα οποία μπορούν να μεταφέρουν πληροφορίες όπως actions, URIs και extras [59]. Ανάλογα με τον τρόπο που ένα component δηλώνεται στο AndroidManifest.xml και χρησιμοποιείται από την εφαρμογή, μπορεί να είναι προσβάσιμο και από άλλες εφαρμογές, δημιουργώντας ένα σημείο μέσω του οποίου εξωτερικά ελεγχόμενα δεδομένα εισέρχονται στη λογική της εφαρμογής [36]. Η επιφάνεια αυτή δεν περιορίζεται στην άμεση επικοινωνία μεταξύ εφαρμογών, καθώς τα δεδομένα μπορούν να φτάσουν και μέσω μηχανισμών πλοήγησης, διαμοιρασμού περιεχομένου ή άλλων διεπαφών που εκθέτει η πλατφόρμα. Βασικά σημεία εισόδου αποτελούν :

- **Inter-application communication:** μέσω exported application components και Intents, τα οποία επιτρέπουν την ενεργοποίηση λειτουργιών της εφαρμογής από εξωτερικούς callers και τη μεταφορά δεδομένων μέσω actions, extras και URIs [36], [59].
- **URI-based entry points:** μέσω Deep Links και App Links, τα οποία μπορούν να ενεργοποιήσουν συγκεκριμένα components και να μεταφέρουν δεδομένα από εξωτερικά URIs προς την εφαρμογή [48], [51].
- **Content URIs και URI grants:** μέσω Content Providers και μηχανισμών προσωρινής παραχώρησης πρόσβασης σε συγκεκριμένα δεδομένα ή αρχεία [47], [85].
- **WebViews και εξωτερικό web περιεχόμενο:** όταν δεδομένα που προέρχονται από εξωτερικά entry points επηρεάζουν τα URLs ή το περιεχόμενο που φορτώνεται σε WebView, δημιουργώντας πρόσθετη επιφάνεια αλληλεπίδρασης μεταξύ web περιεχομένου και εφαρμογής [63], [64].

Τα παραπάνω σημεία αποτελούν ένα μεγάλο μέρος της επιφάνειας επίθεσης (*attack surface*) μιας εφαρμογής Android, καθώς επιτρέπουν την είσοδο δεδομένων ή αιτημάτων από εξωτερικές πηγές. Κάθε τέτοιο σημείο δημιουργεί ένα όριο εμπιστοσύνης, στο οποίο τα εισερχόμενα δεδομένα πρέπει να

θεωρούνται μη έμπιστα και να υποβάλλονται στους κατάλληλους ελέγχους πριν επηρεάσουν τη λειτουργία της εφαρμογής [3] , [51], [63].

Επειδή οι μηχανισμοί αυτοί είναι απαραίτητοι για τη διαλειτουργικότητα και την κανονική λειτουργία των εφαρμογών, η εξέλιξη του Android επικεντρώθηκε κυρίως στη μείωση της μη αναγκαίας έκθεσης και στην καθιέρωση ασφαλέστερων προεπιλογών. Χαρακτηριστικό παράδειγμα αποτελεί το `android:exported` στο `AndroidManifest.xml`, το οποίο καθορίζει αν ένα component μπορεί να ενεργοποιηθεί από άλλες εφαρμογές. Όταν η τιμή του είναι `true`, το component καθίσταται προσβάσιμο εκτός της εφαρμογής και απαιτείται προσοχή στην επεξεργασία των `Intents`, `URIs` και `extras` που λαμβάνει, ενώ με `exported="false"` η πρόσβαση περιορίζεται στην ίδια την εφαρμογή και σε ορισμένες ειδικές περιπτώσεις του συστήματος [36].

Ιστορικά, η τιμή του `android:exported` δεν απαιτούνταν πάντοτε να δηλώνεται ρητά, με την προεπιλεγμένη συμπεριφορά να εξαρτάται από τον τύπο του component και την ύπαρξη `intent-filter`. Από το Android 12 (API level 31), οι εφαρμογές που στοχεύουν API level 31 ή νεότερο υποχρεούνται να δηλώνουν ρητά το `android:exported` για κάθε `Activity`, `Service` ή `Broadcast Receiver` που περιλαμβάνει `intent-filter`, διαφορετικά η εγκατάσταση της εφαρμογής αποτυγχάνει [27]. Στο Android 13 (API level 33) ενισχύθηκε επιπλέον ο έλεγχος των `Intents` που προέρχονται από εξωτερικές εφαρμογές. Για εφαρμογές που στοχεύουν στο API level 33 ή νεότερο, ένα `exported component` που διαθέτει `intent-filter` δέχεται εξωτερικά `Intents` μόνο όταν τα `actions` και `categories` τους αντιστοιχούν σε κάποιο από τα δηλωμένα φίλτρα [59]. Η αλλαγή αυτή περιορίζει περιπτώσεις όπου ένα component μπορούσε προηγουμένως να ενεργοποιηθεί μέσω `explicit Intent` χωρίς να ακολουθεί το `interface` που είχε δηλωθεί από τον developer [59].

Η ασφάλεια της επικοινωνίας μέσω `Intents` ενισχύθηκε περαιτέρω στο Android 14 (API level 34), όπου εισήχθησαν αυστηρότεροι περιορισμοί στη χρήση `implicit Intents` για εσωτερικά components μιας εφαρμογής. Για εφαρμογές που στοχεύουν API level 34 ή νεότερο, ένα `implicit Intent` παραδίδεται μόνο σε `exported components`, ενώ η ενεργοποίηση ενός `non-exported component` της ίδιας εφαρμογής απαιτεί κάποιο `explicit Intent` που προσδιορίζει τον προορισμό. Η αλλαγή αυτή μειώνει τη δυνατότητα παρεμβολής τρίτων εφαρμογών σε επικοινωνία που προορίζεται για εσωτερική χρήση [42].

Το `Intent redirection` αποτελεί ένα ξεχωριστό ζήτημα, το οποίο προκύπτει όταν μια εφαρμογή λαμβάνει ένα `Intent` από εξωτερική πηγή και χρησιμοποιεί ένα `nested Intent` ή στοιχεία που προέρχονται από αυτό για την ενεργοποίηση άλλου component. Από το Android 12, το `StrictMode` παρέχει μηχανισμό εντοπισμού ορισμένων `unsafe intent launches` κατά την ανάπτυξη, επιτρέποντας στον developer να αναγνωρίσει περιπτώσεις όπου ένα μη έμπιστο `nested Intent` προωθείται απευθείας σε επόμενη λειτουργία [38]. Στο

Android 16 (API level 36) προστέθηκε προεπιλεγμένο hardening απέναντι σε επιθέσεις Intent redirection, ώστε το framework να εφαρμόζει πρόσθετους ελέγχους όταν ένα nested Intent που προέρχεται από εξωτερική είσοδο χρησιμοποιείται για την εκκίνηση κάποιου άλλου component. Η αλλαγή αυτή στοχεύει στον περιορισμό περιπτώσεων όπου ένας επιτιθέμενος μπορεί να επηρεάσει τον τελικό προορισμό της εκτέλεσης ή να αποκτήσει πρόσβαση σε προστατευμένους πόρους μέσω ενός ευάλωτου ενδιάμεσου component [39].

Έναν ιδιαίτερο μηχανισμό επικοινωνίας μέσω Intents αποτελούν οι Broadcast Receivers, οι οποίοι επιτρέπουν σε μια εφαρμογή να λαμβάνει μηνύματα για γεγονότα που προέρχονται από το σύστημα ή από άλλες εφαρμογές. Ένας receiver μπορεί να δηλωθεί στατικά στο AndroidManifest.xml ή να καταχωριστεί δυναμικά μέσω της registerReceiver(). Από το Android 13 (API level 33) παρέχεται η δυνατότητα καθορισμού της εξωτερικής προσβασιμότητας ενός dynamically registered receiver μέσω των flags RECEIVER_EXPORTED και RECEIVER_NOT_EXPORTED [40], [41], ενώ στο Android 14 η σχετική δήλωση έγινε υποχρεωτική στις αντίστοιχες περιπτώσεις για εφαρμογές που στοχεύουν νεότερα API levels [42].

Αντίστοιχο ρόλο στη διαμοίραση δεδομένων έχουν οι Content Providers, οι οποίοι παρέχουν δομημένη πρόσβαση σε δεδομένα μέσω content:// URIs. Η δυνατότητα πρόσβασης από άλλες εφαρμογές καθορίζεται από ρυθμίσεις όπως τα android:exported, android:permission, android:readPermission και android:writePermission, ενώ ο μηχανισμός android:grantUriPermissions επιτρέπει την προσωρινή παραχώρηση πρόσβασης σε συγκεκριμένα URIs χωρίς να απαιτείται η συνολική έκθεση του provider [46]. Για πιο περιορισμένο έλεγχο, μπορεί να χρησιμοποιηθεί το <grant-uri-permission> ώστε να οριστούν συγκεκριμένα URI paths στα οποία επιτρέπεται να εφαρμοστούν τέτοια grants [46], [84].

Πραγματοποιήθηκαν βελτιώσεις και στον τρόπο με τον οποίο το Android Framework χειρίζεται αντικείμενα που μεταφέρονται μέσω Intents και Binder. Από το Android 13 (API level 33) εισήχθησαν type-safe εκδόσεις μεθόδων όπως οι getParcelableExtra(String, Class) και getParcelableArrayListExtra(String, Class), στις οποίες ο caller δηλώνει τον αναμενόμενο τύπο του αντικειμένου. Οι παλαιότερες untyped εκδόσεις χαρακτηρίστηκαν ως deprecated, με αποτέλεσμα το framework να πραγματοποιεί αυστηρότερο type checking κατά την ανάκτηση αντικειμένων από ένα Parcel [54].

Η επιφάνεια εισόδου μιας εφαρμογής επεκτείνεται και στους μηχανισμούς πλοήγησης που επιτρέπουν σε εξωτερικά URLs να ενεργοποιούν λειτουργίες της μέσω Deep Links. Τα links αυτά δρομολογούνται μέσω Intents και μπορούν να μεταφέρουν δεδομένα από πηγές που βρίσκονται εκτός του άμεσου ελέγχου της

εφαρμογής. Οι πρώτες υλοποιήσεις βασίζονταν συχνά σε custom URI schemes, όπως myapp://login ή bank://callback, τα οποία δηλώνονται μέσω intent-filter στο AndroidManifest.xml. Επειδή ένα custom scheme δεν δεσμεύεται αποκλειστικά από μία εφαρμογή, διαφορετικές εφαρμογές μπορούν να δηλώσουν ότι το χειρίζονται, δημιουργώντας τη δυνατότητα ένα URI να δρομολογηθεί σε διαφορετικό handler από εκείνον που είχε αρχικά προβλεφθεί [49]. Για την αντιμετώπιση αυτού του περιορισμού, το Android 6.0 (API level 23) εισήγαγε τα Android App Links, τα οποία βασίζονται σε HTTP ή HTTPS URLs και επαληθεύουν τη σχέση μεταξύ μιας εφαρμογής και του αντίστοιχου web domain. Η εφαρμογή δηλώνει το σχετικό intent-filter με android:autoVerify="true", ενώ το domain δημοσιεύει το αρχείο /.well-known/assetlinks.json, μέσω του οποίου το Android μπορεί να επαληθεύσει ότι το συγκεκριμένο package είναι εξουσιοδοτημένο να χειρίζεται links του domain [48].

Η αποτελεσματικότητα του μηχανισμού εξαρτάται όμως από την ορθή διαμόρφωση της εφαρμογής και της αντίστοιχης web υποδομής. Οι Liu et al. εξέτασαν περισσότερες από 160.000 Android εφαρμογές και διαπίστωσαν ότι, στο dataset της μελέτης τους, μόνο 194 από τις 8.878 εφαρμογές που χρησιμοποιούσαν App Links περνούσαν επιτυχώς τη διαδικασία verification [49]. Συνεπώς η ύπαρξη ενός μηχανισμού ασφαλείας σε επίπεδο πλατφόρμας δεν αρκεί από μόνη της, η σωστή λειτουργία του εξαρτάται από τη διαμόρφωση που πραγματοποιείται από τον developer.

Date	Apps w/ App Links	Apps Verif. Turned On	Apps Verified	Apps with Failed Verifications*					
				App Misconfig.	Host w/o Assoc. F.	Host w/ HTTP	Wrong Path	Host Invalid F.	Host Assoc. Other apps
Jan.17	8,878	415	194	26	177	11	0	10	60
May.17	8,878	415	192	26	171	8	0	18	57

Εικόνα 6 : Στατιστικά επαλήθευσης Android App Links και συχνότερα σφάλματα διαμόρφωσης στο dataset App 2016. **Πηγή:** [49, Table 3].

Η προστασία ενισχύθηκε στο Android 12 (API level 31), όπου ένα generic web intent μπορεί να οδηγήσει απευθείας σε Activity μιας εφαρμογής μόνο όταν υπάρχει έγκριση για τον αντίστοιχο domain, είτε μέσω επιτυχούς App Link verification είτε μέσω επιλογής του χρήστη. Σε διαφορετική περίπτωση, το intent δρομολογείται στον προεπιλεγμένο browser, περιορίζοντας τη δυνατότητα μιας εφαρμογής να αναλαμβάνει URLs απλώς μέσω ενός κατάλληλου intent-filter [50]. Στο Android 15 (API level 35) εισήχθησαν τα Dynamic App Links, τα οποία επιτρέπουν τον καθορισμό πιο λεπτομερών κανόνων δρομολόγησης μέσω του assetlinks.json, όπως περιορισμούς σε paths και άλλες ιδιότητες του URL [52].

Η αλληλεπίδραση με web περιεχόμενο επεκτείνεται και στα WebViews, μέσω των οποίων μια εφαρμογή μπορεί να ενσωματώνει απομακρυσμένο ή τοπικό περιεχόμενο στο περιβάλλον της. Η χρήση τους

δημιουργεί ένα πρόσθετο όριο εμπιστοσύνης μεταξύ του web περιεχομένου και των λειτουργιών της εφαρμογής, γεγονός που οδήγησε σταδιακά σε αυστηρότερες προεπιλεγμένες ρυθμίσεις πρόσβασης σε τοπικούς πόρους [53]. Χαρακτηριστικό παράδειγμα αποτελεί η πρόσβαση σε file:// URLs, για την οποία επιλογές όπως οι `setAllowFileAccessFromFileURLs()` και `setAllowUniversalAccessFromFileURLs()` περιορίστηκαν και αργότερα χαρακτηρίστηκαν deprecated λόγω των κινδύνων που δημιουργούσαν. Για τη φόρτωση τοπικού περιεχομένου, το Android προτείνει πλέον μηχανισμούς όπως το `WebViewAssetLoader` αντί της άμεσης χρήσης file:// URLs [53], [135].

Μία από τις σημαντικές αλλαγές στην ασφάλεια των WebViews αφορά τον μηχανισμό `addJavascriptInterface()`, μέσω του οποίου ένα Java object μπορεί να εκτεθεί στο JavaScript περιβάλλον του WebView. Σε εφαρμογές που στόχευαν API level 16 ή χαμηλότερο, η JavaScript μπορούσε να καλέσει public methods του injected object, συμπεριλαμβανομένων inherited methods, δημιουργώντας αυξημένο κίνδυνο όταν το WebView επεξεργαζόταν μη έμπιστο περιεχόμενο. Από το Android 4.2 (API level 17), για εφαρμογές που στοχεύουν API level 17 ή νεότερο, η πρόσβαση περιορίστηκε στις public methods που έχουν επισημανθεί με `@JavascriptInterface` [60].

Η ενίσχυση της ασφάλειας σε επίπεδο εφαρμογών δεν περιορίστηκε στους μηχανισμούς ανταλλαγής δεδομένων, αλλά επεκτάθηκε και στην προστασία απέναντι σε μορφές κατάχρησης του γραφικού περιβάλλοντος, όπως τα UI overlays. Μέσω τέτοιων μηχανισμών, μια εφαρμογή μπορεί να επιχειρήσει να επηρεάσει την αλληλεπίδραση του χρήστη με το interface μιας άλλης εφαρμογής. Από το Android 6.0 (API level 23), η εμφάνιση παραθύρων πάνω από άλλες εφαρμογές μέσω `SYSTEM_ALERT_WINDOW` απαιτεί ειδική έγκριση του χρήστη, ενώ στο Android 8.0 (API level 26) οι παλαιότεροι τύποι system alert windows αντικαταστάθηκαν από το περισσότερο περιορισμένο `TYPE_APPLICATION_OVERLAY` [56], [57]. Στο Android 12 (API level 31) προστέθηκαν πρόσθετοι περιορισμοί για μη έμπιστα touch events όταν ένα παράθυρο διαφορετικού UID επικαλύπτει την underlying εφαρμογή, ενώ παρέχεται και η δυνατότητα απόκρυψης third-party overlays από ευαίσθητα παράθυρα μέσω της `HIDE_OVERLAY_WINDOWS` permission και της `Window.setHideOverlayWindows(true)` [50], [58].

Παρά τους μηχανισμούς που εφαρμόζονται σε επίπεδο πλατφόρμας, η αξιολόγηση της ασφάλειας μιας συγκεκριμένης ροής εξακολουθεί να απαιτεί γνώση της λογικής της εφαρμογής και του ρόλου των δεδομένων που επεξεργάζεται. Το Android μπορεί να περιορίσει την πρόσβαση σε components, να επαληθεύσει σχέσεις εμπιστοσύνης με web domains και να επιβάλει αυστηρότερους κανόνες στη χρήση Intents ή WebViews, χωρίς όμως να μπορεί να προσδιορίσει από μόνο του αν ένα συγκεκριμένο extra, URI, URL ή destination είναι έγκυρο και αναμενόμενο στο πλαίσιο της συγκεκριμένης λειτουργίας.

2.4 Ευπάθειες σε επίπεδο εφαρμογών

Οι παραπάνω εξελίξεις αποτυπώνουν τη σταδιακή ενίσχυση της ασφάλειας του Android μέσω αυστηρότερων περιορισμών, ασφαλέστερων προεπιλογών και σαφέστερου ελέγχου της εξωτερικής πρόσβασης. Οι μηχανισμοί αυτοί περιορίζουν σημαντικό μέρος της επιφάνειας επίθεσης, χωρίς όμως να μπορούν να αξιολογήσουν τη σημασιολογία των δεδομένων που επεξεργάζεται μια εφαρμογή ή τον τρόπο με τον οποίο αυτά επηρεάζουν τη λειτουργία της.

Σε αυτό το πλαίσιο, οι ευπάθειες σε επίπεδο εφαρμογής εμφανίζονται συχνά όταν εξωτερικά ελεγχόμενα δεδομένα φτάνουν σε ευαίσθητες λειτουργίες χωρίς τους απαραίτητους ελέγχους ή όταν μηχανισμοί της πλατφόρμας χρησιμοποιούνται με τρόπο που διευρύνει την επιφάνεια επίθεσης. Στην ενότητα που ακολουθεί εξετάζονται χαρακτηριστικές κατηγορίες τέτοιων ευπαθειών, με έμφαση στα WebViews, στα Intents και στα externally reachable application components, καθώς και σε ζητήματα αποσειριοποίησης, δυναμικής φόρτωσης κώδικα, JNI και διαχείρισης ευαίσθητων δεδομένων.

2.4.1 Μη ασφαλής χρήση του Webview

Το WebView επιτρέπει σε μια Android εφαρμογή να εμφανίζει web περιεχόμενο μέσα στο δικό της γραφικό περιβάλλον και δημιουργεί ένα σημείο μετάβασης μεταξύ του Web security model και της native εφαρμογής. Ανάλογα με τη διαμόρφωσή του, το περιεχόμενο που φορτώνεται μπορεί να αποκτήσει πρόσβαση σε λειτουργίες ή δεδομένα που δεν θα ήταν διαθέσιμα σε μια συνηθισμένη σελίδα εξωτερικού browser, ιδιαίτερα όταν χρησιμοποιούνται native interfaces, authentication state ή άλλες application-specific δυνατότητες. Για τον λόγο αυτό, η δυνατότητα φόρτωσης μη έμπιστου ή εξωτερικά ελεγχόμενου περιεχομένου σε ένα WebView μπορεί να δημιουργήσει σημαντικά προβλήματα ασφάλειας [63], [65].

Μια συνηθισμένη κατηγορία τέτοιων προβλημάτων είναι η μη ασφαλής φόρτωση URI (*unsafe URI loading*), όπου η εφαρμογή χρησιμοποιεί URI που προέρχεται από εξωτερικά ελεγχόμενη πηγή για τη φόρτωση περιεχομένου χωρίς να έχει προηγηθεί επαρκής επικύρωση του προορισμού. Η τιμή μπορεί να εισέλθει μέσω Intent, Deep Link ή άλλης παραμέτρου που διασχίζει το application boundary και να φτάσει τελικά σε λειτουργία όπως η WebView.loadUrl(). Η επίσημη καθοδήγηση του Android συνιστά την ανάλυση του URI και την επαλήθευση τόσο του scheme όσο και του host πριν χρησιμοποιηθεί από το WebView, καθώς κανένα από τα δύο στοιχεία δεν επαρκεί από μόνο του για να προσδιορίσει με ασφάλεια τον τελικό προορισμό [63].

Ιδιαίτερο ενδιαφέρον παρουσιάζουν τα Deep Links, επειδή μπορούν να μεταφέρουν δεδομένα από το εξωτερικό περιβάλλον προς την εφαρμογή και αυτά να χρησιμοποιηθούν αργότερα ως ξεχωριστές τιμές

πλοήγησης. Για παράδειγμα, στο URI `myapp://open/web?url=https://example.com/` το πρώτο URI χρησιμοποιείται από το Android για τη δρομολόγηση προς την εφαρμογή, ενώ η τιμή της παραμέτρου `url` επεξεργάζεται από την ίδια την εφαρμογή. Η εγκυρότητα του Deep Link δεν καθιστά αυτομάτως έμπιστες τις παραμέτρους που μεταφέρει, επομένως τα δεδομένα αυτά πρέπει να επικυρώνονται πριν χρησιμοποιηθούν για λειτουργίες που επηρεάζουν την ασφάλεια [51].

```
// The vulnerable source code
String url = getIntent().getStringExtra("url");

webView.getSettings().setJavaScriptEnabled(true);
webView.loadUrl(url);
```

Εικόνα 7: Παράδειγμα μη ασφαλούς φόρτωσης εξωτερικά ελεγχόμενης URL σε WebView.

Στο παράδειγμα της Εικόνας 7, η τιμή `url` λαμβάνεται από το εισερχόμενο Intent και χρησιμοποιείται απευθείας στη `WebView.loadUrl()`. Εφόσον το αντίστοιχο component είναι προσβάσιμο από εξωτερικό caller, η επιλογή του περιεχομένου που θα φορτωθεί στο WebView επηρεάζεται από δεδομένα εκτός της εφαρμογής [65]. Η σοβαρότητα της συμπεριφοράς εξαρτάται στη συνέχεια από τις δυνατότητες του συγκεκριμένου WebView και από το περιβάλλον στο οποίο εκτελείται το περιεχόμενο, όπως η παρουσία native bridges, authentication state ή πρόσβασης σε τοπικούς πόρους.

Η ασφαλής επικύρωση απαιτεί η απόφαση εμπιστοσύνης να βασίζεται στα επιμέρους στοιχεία του parsed URI και όχι σε απλές συγκρίσεις πάνω στην αρχική συμβολοσειρά. Το scheme καθορίζει τον τρόπο με τον οποίο θα ερμηνευθεί το URI, ενώ ο host προσδιορίζει τον προορισμό για network-based schemes. Η επαλήθευση μόνο του host αφήνει ασαφή τη σημασιολογία του URI και μπορεί να επιτρέψει διαφορετικά schemes από εκείνα που αναμένει η εφαρμογή [63], [65].

```
// Host only validation
private boolean isTrusted(String value) {
    Uri uri = Uri.parse(value);
    return "example.com".equals(uri.getHost());
}
```

Εικόνα 8 : Παράδειγμα ανεπαρκούς επικύρωσης URI με έλεγχο μόνο του host.

Αν μια εφαρμογή επαληθεύει αποκλειστικά ότι το `getHost()` επιστρέφει `example.com`, δεν έχει ακόμη καθορίσει ποια schemes θεωρούνται αποδεκτά. Για web περιεχόμενο, η εφαρμογή θα πρέπει να ελέγχει

ρητά και το αναμενόμενο scheme μαζί με τον host, ώστε η απόφαση εμπιστοσύνης να αντιστοιχεί στη μορφή URI που σχεδιάστηκε να υποστηρίζει [63], [65].

```
private boolean isTrusted(String value) {  
    Uri uri = Uri.parse(value);  
  
    return "https".equals(uri.getScheme())  
        && uri.getHost().endsWith("example.com");  
}
```

Εικόνα 9 : Παράδειγμα λανθασμένου ελέγχου domain μέσω string comparison.

Προβλήματα μπορούν επίσης να προκύψουν όταν η επικύρωση ενός domain βασίζεται σε μεθόδους string comparison. Ένας έλεγχος όπως `host.endsWith("example.com")` δεν λαμβάνει υπόψη τα όρια των DNS labels και μπορεί να δεχθεί τιμές όπως `notexample.com`, οι οποίες δεν αποτελούν subdomain του `example.com`. Αν η εφαρμογή επιτρέπει το βασικό domain και τα subdomains του, η σύγκριση πρέπει να διατηρεί το domain boundary, για παράδειγμα ελέγχοντας είτε ακριβή ισότητα με `example.com` είτε κατάληξη σε `.example.com` [63].

Έστω ότι μια εφαρμογή επιθυμεί να επιτρέπει στο WebView τη φόρτωση περιεχομένου αποκλειστικά από το `legitimate.com` και πραγματοποιεί τον έλεγχο που παρουσιάζεται στην Εικόνα 10.

```
String url = getIntent().getStringExtra("url");  
  
if (url.startsWith("https://legitimate.com")) {  
    webView.loadUrl(url);  
}
```

Εικόνα 10 : Παράδειγμα λανθασμένης επικύρωσης URL μέσω string comparison.

Η χρήση του `startsWith()` μπορεί αρχικά να φαίνεται επαρκής, επειδή ελέγχει την παρουσία του αναμενόμενου scheme και domain στην αρχή της τιμής, χωρίς όμως να λαμβάνει υπόψη τη δομή του URI. Για παράδειγμα, η διεύθυνση `https://legitimate.com@attacker.com/` ικανοποιεί έναν έλεγχο `startsWith("https://legitimate.com")`, παρότι ο πραγματικός host είναι το `attacker.com`. Στα hierarchical URIs, το τμήμα πριν από το `@` μπορεί να αποτελεί user information και επομένως η θέση στην οποία εμφανίζεται ένα domain μέσα στην αρχική συμβολοσειρά δεν προσδιορίζει από μόνη της τον τελικό προορισμό [66]. Η απόφαση εμπιστοσύνης πρέπει για τον λόγο αυτό να βασίζεται στα πεδία που προκύπτουν μετά το parsing του URI.

Η επικύρωση γίνεται πιο σύνθετη όταν η ίδια τιμή περνά από περισσότερα στάδια επεξεργασίας πριν χρησιμοποιηθεί από το WebView. Decoding, normalization ή ανακατασκευή του URI μπορούν να αλλάξουν την αναπαράσταση που τελικά καταναλώνεται, με αποτέλεσμα ο έλεγχος ασφάλειας και η επόμενη λειτουργία να μην αξιολογούν απαραίτητα ακριβώς την ίδια μορφή της εισόδου. Τέτοιες διαφορές αποκτούν σημασία όταν διαφορετικοί parsers χειρίζονται μη κανονικές αναπαραστάσεις με διαφορετικό τρόπο, δημιουργώντας αυτό που μπορεί να περιγραφεί ως parser ή representation differential [65].

Για τον λόγο αυτό είναι χρήσιμο να διακρίνονται το parsing, το normalization και το validation ως διαφορετικά στάδια της επεξεργασίας. Το parsing αναγνωρίζει τα δομικά στοιχεία του URI, ενώ το normalization μπορεί να μετατρέψει διαφορετικές αλλά ισοδύναμες αναπαραστάσεις σε κοινή μορφή. Η τελική απόφαση για το αν μια τιμή επιτρέπεται στη συγκεκριμένη λειτουργία αποτελεί ευθύνη του application-level validation και πρέπει να εφαρμόζεται στην ίδια σημασιολογική αναπαράσταση που θα χρησιμοποιηθεί στη συνέχεια. Η android.net.Uri δεν πρέπει επομένως να αντιμετωπίζεται ως μηχανισμός ασφάλειας από μόνη της, καθώς η επιτυχής Uri.parse() ή η ανάκτηση ενός αναμενόμενου host δεν αποδεικνύουν ότι ολόκληρη η είσοδος ικανοποιεί την πολιτική που θέλει να εφαρμόσει η εφαρμογή [66].

Τεχνικές αναλύσεις της Oversecured έχουν παρουσιάσει πραγματικές περιπτώσεις στις οποίες διαφορετικές αναπαραστάσεις του ίδιου URI οδήγησαν σε διαφορετική ερμηνεία μεταξύ του validation logic και του WebView. Χαρακτηριστικό ιστορικό παράδειγμα αποτελεί ο διαφορετικός χειρισμός των backslashes από το android.net.Uri και το WebView σε παλαιότερα Android API levels, όπου η απόκλιση αυτή μπορούσε να επηρεάσει την ερμηνεία του τελικού host [65]. Τα παραδείγματα αυτά δείχνουν ότι η ασφάλεια του URI validation εξαρτάται από την αναπαράσταση που τελικά χρησιμοποιείται για τη φόρτωση και από τη συνέπεια μεταξύ του parser που εφαρμόζει τον έλεγχο και εκείνου που καταναλώνει την τιμή.

Η επικύρωση του αρχικού URI δεν αρκεί για να εξασφαλίσει ότι το WebView θα παραμείνει στον ίδιο έμπιστο προορισμό σε όλη τη διάρκεια της πλοήγησης. Ένα URL που περνά επιτυχώς τον αρχικό έλεγχο μπορεί στη συνέχεια να ανακατευθυνθεί σε διαφορετικό origin, είτε μέσω HTTP redirect είτε μέσω πλοήγησης που πραγματοποιείται από το ίδιο το περιεχόμενο. Για τον λόγο αυτό, η πολιτική εμπιστοσύνης της εφαρμογής πρέπει να λαμβάνει υπόψη την τελική κατάσταση της πλοήγησης και να εφαρμόζεται ξανά όταν το URL μεταβάλλεται ή ανακατασκευάζεται πριν χρησιμοποιηθεί από το WebView [63], [65].

Ο έλεγχος της πλοήγησης αποτελεί μία μόνο πλευρά των προβλημάτων που μπορούν να εμφανιστούν σε ένα WebView. Ιδιαίτερο ενδιαφέρον παρουσιάζουν και οι περιπτώσεις όπου εξωτερικά ελεγχόμενα δεδομένα ενσωματώνονται σε HTML ή JavaScript που εκτελείται μέσα στο WebView, καθώς η είσοδος

επηρεάζει πλέον άμεσα τον κώδικα και το execution context της σελίδας. APIs όπως τα `evaluateJavascript()`, `loadUrl("javascript:...")`, `loadData()` και `loadDataWithBaseURL()` μπορούν να χρησιμοποιηθούν για δυναμική δημιουργία ή εκτέλεση περιεχομένου και απαιτούν προσεκτικό χειρισμό όταν τα δεδομένα προέρχονται από μη έμπιστη πηγή. Η λανθασμένη χρήση τους μπορεί να οδηγήσει σε προβλήματα όπως JavaScript injection ή Cross-Site Scripting, ενώ η JavaScript στο WebView θα πρέπει να ενεργοποιείται μόνο όταν απαιτείται από τη λειτουργικότητα της εφαρμογής [60].

```
webView.getSettings().setJavaScriptEnabled(true);
webView.loadUrl("https://legitimate.com/");

String page =
    getIntent().getData().getQueryParameter("page");

webView.evaluateJavascript(
    "loadPage('" + page + "')",
    null
);
```

Εικόνα 11 : Παράδειγμα επικίνδυνης χρήσης του `evaluateJavascript()` API.

Στο παράδειγμα της Εικόνας 11, μια εξωτερικά ελεγχόμενη τιμή ενσωματώνεται απευθείας σε JavaScript source που εκτελείται μέσω `evaluateJavascript()`. Η τιμή `page` αντιμετωπίζεται από την εφαρμογή ως argument της `loadPage()`, όμως η άμεση εισαγωγή της μέσα στο script σημαίνει ότι ειδικοί χαρακτήρες μπορούν να μεταβάλουν τη σύνταξη της JavaScript έκφρασης και να επηρεάσουν τον κώδικα που τελικά εκτελείται. Η συμπεριφορά αποκτά ιδιαίτερη σημασία επειδή το script εκτελείται στο context της σελίδας που είναι ενεργή στο WebView εκείνη τη στιγμή, οπότε οι δυνατότητες που προκύπτουν εξαρτώνται και από το συγκεκριμένο origin [65].

Αντίστοιχο πρόβλημα εμφανίζεται όταν δημιουργούνται δυναμικά javascript: URLs μέσω της `loadUrl()`. Η ασφαλέστερη πρακτική είναι τα εξωτερικά δεδομένα να μετατρέπονται σε κατάλληλη JavaScript ή JSON representation πριν ενσωματωθούν σε script, αποφεύγοντας την άμεση string concatenation που μπορεί να αλλάξει τη σημασιολογία του κώδικα [65].

```
String safeValue = JSONObject.quote(page);

webView.evaluateJavascript(
    "loadPage(" + safeValue + ")",
    null
);
```

Εικόνα 12 : Παράδειγμα ασφαλούς χρήσης του evaluateJavascript API.

Στην ασφαλέστερη εκδοχή της Εικόνας 12, η εξωτερική τιμή μετατρέπεται πρώτα σε JSON string representation πριν ενσωματωθεί στον JavaScript κώδικα. Με αυτόν τον τρόπο, χαρακτήρες που διαφορετικά θα μπορούσαν να επηρεάσουν τη σύνταξη της έκφρασης αντιμετωπίζονται ως μέρος της τιμής και διατηρείται σαφέστερος διαχωρισμός ανάμεσα στον κώδικα που δημιουργεί η εφαρμογή και στα δεδομένα που προέρχονται από εξωτερική πηγή. Η πρακτική αυτή αποφεύγει την άμεση string concatenation μη έμπιστων δεδομένων μέσα σε executable JavaScript και μειώνει τον κίνδυνο JavaScript injection [65].

```
public class WebActivity extends Activity {  
    private WebView webView;  
  
    @Override  
    protected void onCreate(Bundle state) {  
        super.onCreate(state);  
  
        webView = findViewById(R.id.webView);  
        webView.getSettings().setJavaScriptEnabled(true);  
  
        webView.loadUrl(getIntent().getDataString());  
    }  
  
    @Override  
    protected void onNewIntent(Intent intent) {  
        super.onNewIntent(intent);  
  
        webView.loadUrl(intent.getDataString());  
    }  
}
```

Εικόνα 13 : Παράδειγμα της κατάστασης του webview μετά από συγκεκριμένα API calls

Η ασφάλεια ενός WebView επηρεάζεται και από το state που διατηρεί μεταξύ διαδοχικών navigations. Όταν η ίδια WebView instance επαναχρησιμοποιείται, μια νέα πλοήγηση μπορεί να εκτελεστεί πάνω στο context του document που έχει ήδη φορτωθεί. Αν, για παράδειγμα, έχει προηγηθεί φόρτωση έμπιστου περιεχομένου και στη συνέχεια μια εξωτερικά ελεγχόμενη είσοδος επιτρέπει javascript: navigation, ο JavaScript κώδικας μπορεί να εκτελεστεί μέσα στο ήδη ενεργό document και να κληρονομήσει το αντίστοιχο origin context [65].

Η συμπεριφορά αυτή μπορεί να οδηγήσει σε Universal Cross-Site Scripting (UXSS), όταν attacker-controlled JavaScript εκτελείται με τις δυνατότητες ενός άλλου origin. Στο πλαίσιο των Android WebViews, το πρόβλημα μπορεί να προκύψει από τη λογική της ίδιας της εφαρμογής και από τον τρόπο με τον οποίο επαναχρησιμοποιεί το WebView state, χωρίς να απαιτείται απαραίτητα ευπάθεια στον Chromium engine [65].

Η αντιμετώπιση τέτοιων περιπτώσεων βασίζεται κυρίως στον περιορισμό των δυνατοτήτων του WebView σε όσες απαιτούνται πραγματικά από τη λειτουργία της εφαρμογής. Η JavaScript θα πρέπει να ενεργοποιείται μόνο όταν είναι απαραίτητη, ενώ εξωτερικά ελεγχόμενα δεδομένα δεν πρέπει να ενσωματώνονται απευθείας σε JavaScript ή HTML που πρόκειται να εκτελεστεί. Αντίστοιχα, APIs όπως το `loadDataWithBaseURL()` χρειάζονται ιδιαίτερη προσοχή όταν το περιεχόμενο δεν ελέγχεται πλήρως από την εφαρμογή, καθώς το `baseUrl` επηρεάζει το `origin` του `document` και επομένως τις δυνατότητες που αποκτά το φορτωμένο περιεχόμενο [68].

Η εκτέλεση JavaScript μέσα σε ένα WebView αποκτά μεγαλύτερη σημασία όταν η εφαρμογή έχει εκθέσει μηχανισμό επικοινωνίας με τον native Android κώδικα. Οι μηχανισμοί αυτοί, είναι γνωστοί ως JavaScript ή native bridges, και επιτρέπουν στο web περιεχόμενο να καλεί λειτουργίες της εφαρμογής ή να ανταλλάσσει δεδομένα με αυτή, επεκτείνοντας ουσιαστικά τις δυνατότητες που είναι διαθέσιμες μέσα από το WebView.

Χαρακτηριστικό παράδειγμα αποτελεί η `addJavascriptInterface()`, μέσω της οποίας ένα Java object εκτίθεται στη JavaScript του WebView με συγκεκριμένο όνομα. Σε εφαρμογές που στοχεύουν API level 17 ή νεότερο, οι methods που μπορούν να κληθούν από JavaScript πρέπει να είναι `public` και να έχουν επισημανθεί με `@JavascriptInterface` [64].

```
public class AppBridge {  
  
    @JavascriptInterface  
    public String getUsername() {  
        return currentUser.getName();  
    }  
  
    @JavascriptInterface  
    public void openSettings() {  
        startActivity(  
            new Intent(Settings.ACTION_SETTINGS)  
        );  
    }  
}  
  
webView.getSettings().setJavaScriptEnabled(true);  
webView.addJavascriptInterface(  
    new AppBridge(),  
    "AndroidBridge"  
);  
  
webView.loadUrl("https://app.example.com/");
```

Εικόνα 14 : Παράδειγμα χρήσης του `addJavaScriptInterface()`.

Η παρουσία ενός τέτοιου bridge επηρεάζει άμεσα την επίπτωση μιας WebView ευπάθειας. Αν attacker-controlled JavaScript αποκτήσει δυνατότητα εκτέλεσης μέσα στο ίδιο WebView, για παράδειγμα μέσω

unsafe URL loading ή JavaScript injection, μπορεί να επιχειρήσει να χρησιμοποιήσει και τα native interfaces που έχουν εκτεθεί από την εφαρμογή [64], [65]. Η addJavascriptInterface() παρουσιάζει επίσης περιορισμούς ως προς το origin isolation, καθώς το εκτεθειμένο object είναι διαθέσιμο στα frames του WebView χωρίς η εφαρμογή να μπορεί να προσδιορίσει με αξιόπιστο τρόπο το origin του frame που πραγματοποιεί την κλήση μέσω της συγκεκριμένης API [64]. Για τον λόγο αυτό, η παρουσία έμπιστου top-level content δεν αρκεί από μόνη της για την προστασία του interface όταν στο ίδιο WebView μπορεί να εκτελεστεί περιεχόμενο διαφορετικής προέλευσης.

Η χρήση JavaScript interfaces θα πρέπει επομένως να περιορίζεται στις περιπτώσεις όπου είναι απαραίτητη για τη λειτουργία της εφαρμογής και οι εκτεθειμένες methods να παρέχουν μόνο τις απολύτως αναγκαίες δυνατότητες. Όταν το WebView πρόκειται να φορτώσει μη έμπιστο περιεχόμενο, το interface μπορεί να αφαιρείται μέσω removeJavascriptInterface(), ενώ η JavaScript θα πρέπει να παραμένει απενεργοποιημένη όταν δεν απαιτείται [64].

Μια ακόμη σημαντική πλευρά της ασφάλειας ενός WebView αφορά την πρόσβαση σε τοπικούς πόρους της συσκευής. Ανάλογα με τη διαμόρφωσή του, το WebView μπορεί να φορτώνει περιεχόμενο και μέσω schemes όπως file:// και content://, γεγονός που αποκτά ιδιαίτερη σημασία όταν στο ίδιο περιβάλλον εκτελείται μη έμπιστο ή εξωτερικά ελεγχόμενο περιεχόμενο. Σε τέτοιες περιπτώσεις, λανθασμένες ρυθμίσεις μπορούν να επιτρέψουν την πρόσβαση σε αρχεία ή δεδομένα που βρίσκονται μέσα στο security context της εφαρμογής και να οδηγήσουν σε προβλήματα της κατηγορίας *Unsafe File Inclusion* [136], [65].

Μία από τις βασικές ρυθμίσεις που επηρεάζουν αυτή τη συμπεριφορά είναι η WebSettings.setAllowFileAccess(), η οποία καθορίζει αν το WebView επιτρέπεται να προσπελαίνει το filesystem μέσω file:// URLs. Η προεπιλεγμένη τιμή της εξαρτάται από το target SDK της εφαρμογής: για εφαρμογές που στοχεύουν Android 10 (API level 29) ή παλαιότερο είναι ενεργοποιημένη, ενώ από το Android 11 (API level 30) και μετά η προεπιλογή είναι απενεργοποιημένη [53]. Σε εφαρμογές που μπορούν να λάβουν URLs από εξωτερικές πηγές, η δυνατότητα αυτή θα πρέπει να παραμένει απενεργοποιημένη εκτός αν είναι απαραίτητη για τη λειτουργία τους.

```
WebSettings settings = webView.getSettings();  
settings.setJavaScriptEnabled(true);  
settings.setAllowFileAccess(true);  
  
webView.loadUrl(attackerControlledUrl);
```

Εικόνα 15 : Παράδειγμα χρήσης του setAllowFileAccess API.

Η επίπτωση της πρόσβασης σε file:// περιεχόμενο εξαρτάται και από το origin του document που εκτελείται μέσα στο WebView, καθώς και από τις υπόλοιπες σχετικές ρυθμίσεις. Ιδιαίτερη σημασία έχουν οι `setAllowFileAccessFromFileURLs()` και `setAllowUniversalAccessFromFileURLs()`, οι οποίες ιστορικά επέτρεπαν σε JavaScript που εκτελείται από file:// origin να πραγματοποιεί ευρύτερη πρόσβαση σε τοπικούς ή άλλους πόρους.

Η `setAllowFileAccessFromFileURLs()` επιτρέπει σε JavaScript που εκτελείται μέσα από τοπικό document να πραγματοποιεί requests προς άλλα file:// resources. Αν το περιεχόμενο του αρχικού document μπορεί να επηρεαστεί από έναν επιτιθέμενο, η ρύθμιση αυτή μπορεί να επεκτείνει σημαντικά τις δυνατότητές του, επιτρέποντας την ανάγνωση πρόσθετων αρχείων στα οποία έχει πρόσβαση το WebView. Για εφαρμογές που στοχεύουν API level 16 ή νεότερο η επιλογή αυτή είναι απενεργοποιημένη από προεπιλογή, ενώ από το API level 30 έχει χαρακτηριστεί deprecated λόγω των κινδύνων ασφαλείας που δημιουργεί [53].

Ακόμη ευρύτερη πρόσβαση παρέχει η `setAllowUniversalAccessFromFileURLs()`, καθώς επιτρέπει σε JavaScript που εκτελείται μέσα σε file:// context να πραγματοποιεί cross-origin requests προς τοπικούς πόρους αλλά και προς απομακρυσμένα origins. Η ενεργοποίησή της περιορίζει σημαντικά την προστασία που διαφορετικά παρέχει το origin isolation και μπορεί να αυξήσει τις δυνατότητες ενός local document που περιέχει μη έμπιστο κώδικα. Η ρύθμιση είναι απενεργοποιημένη από προεπιλογή για εφαρμογές που στοχεύουν API level 16 ή νεότερο και έχει χαρακτηριστεί deprecated από το API level 30 [53].

Ο κίνδυνος γίνεται ουσιαστικός όταν οι παραπάνω ρυθμίσεις συνδυάζονται με εξωτερικά ελεγχόμενη πλοήγηση ή JavaScript execution. Μια εφαρμογή που επιτρέπει σε attacker-controlled δεδομένα να επηρεάσουν το URL του WebView και ταυτόχρονα διαθέτει permissive file-origin access μπορεί να δημιουργήσει διαδρομή προς τοπικούς πόρους που διαφορετικά προστατεύονται από το application sandbox. Η Oversecured έχει παρουσιάσει αντίστοιχες αλυσίδες, όπου ο συνδυασμός ελεγχόμενης WebView navigation, JavaScript και file-origin access οδηγεί σε πρόσβαση σε ευαίσθητα αρχεία της εφαρμογής [65].

Παρόμοια προσοχή απαιτείται και για content:// URIs, τα οποία επιτρέπουν στο WebView να αλληλεπιδρά με δεδομένα που παρέχονται μέσω Android Content Providers. Η δυνατότητα αυτή ελέγχεται από τη `setAllowContentAccess()` και είναι ενεργοποιημένη από προεπιλογή [53]. Όταν το URI που φορτώνεται μπορεί να επηρεαστεί από εξωτερική πηγή, η συμπεριφορά του WebView πρέπει να αξιολογείται σε σχέση με τα permissions και τους providers στους οποίους έχει πρόσβαση η ίδια η εφαρμογή, καθώς το WebView λειτουργεί μέσα στο δικό της security context.

Η φόρτωση τοπικού ενεργού περιεχομένου μπορεί επίσης να οδηγήσει σε file-based XSS όταν ο attacker έχει δυνατότητα να δημιουργήσει ή να τροποποιήσει HTML που θα ανοιχτεί από το WebView ενώ η JavaScript είναι ενεργοποιημένη. Για τον περιορισμό αυτών των περιπτώσεων, οι δυνατότητες πρόσβασης σε file:// και content:// resources θα πρέπει να ενεργοποιούνται μόνο όταν απαιτούνται από τη λειτουργία της εφαρμογής, ενώ οι `setAllowFileAccessFromFileURLs()` και `setAllowUniversalAccessFromFileURLs()` δεν θα πρέπει να χρησιμοποιούνται σε σύγχρονες εφαρμογές λόγω των γνωστών κινδύνων που εισάγουν [136].

Για τη φόρτωση τοπικού application content, το Android προτείνει τη χρήση του `WebViewAssetLoader`, ο οποίος επιτρέπει σε assets και resources της εφαρμογής να διατίθενται στο WebView μέσω HTTP(S)-style URLs, όπως `https://appassets.androidplatform.net/assets/...`, αντί να απαιτείται άμεση πρόσβαση μέσω file://. Με αυτή την προσέγγιση, το τοπικό περιεχόμενο εντάσσεται στο κανονικό origin model του Web και μπορεί να χρησιμοποιήσει resources όπως JavaScript, CSS, εικόνες και iframes χωρίς να χρειάζεται η ενεργοποίηση των permissive file-origin settings που περιγράφηκαν προηγουμένως [135].

```
WebViewAssetLoader assetLoader =  
    new WebViewAssetLoader.Builder()  
        .addPathHandler(  
            "/assets/",  
            new WebViewAssetLoader.AssetsPathHandler(this))  
        .build();  
  
webView.loadUrl(  
    "https://appassets.androidplatform.net/assets/index.html"  
);
```

Εικόνα 16 : Παράδειγμα χρήσης `WebViewAssetLoader`.

Η βασική διαφορά είναι ότι η εφαρμογή εκθέτει στο WebView μόνο συγκεκριμένα application resources μέσα από ελεγχόμενα URL paths, περιορίζοντας την ανάγκη για γενική πρόσβαση στο filesystem. Η χρήση του `WebViewAssetLoader` αποτελεί επομένως προτιμότερη επιλογή όταν η εφαρμογή χρειάζεται να εμφανίσει bundled web content και θέλει να διατηρήσει συμβατότητα με το Same-Origin Policy [135].

Μια διαφορετική μορφή πρόσβασης σε τοπικό περιεχόμενο εμφανίζεται κατά την επιλογή αρχείων από web σελίδες που εκτελούνται μέσα στο WebView. Στοιχεία όπως το `<input type="file">` μπορούν να ενεργοποιήσουν τον μηχανισμό file selection του Android μέσω της `WebChromeClient.onShowFileChooser()`, επιτρέποντας στην εφαρμογή να ανοίξει κατάλληλο picker και να επιστρέψει στο WebView τα Uri των αρχείων που επέλεξε ο χρήστης [53], [67]. Η λειτουργία αυτή

δημιουργεί διαφορετικό security boundary από το άμεσο file:// access, καθώς η πρόσβαση στο αρχείο προκύπτει μέσα από τη διαδικασία επιλογής και από τα permissions που συνοδεύουν το επιστρεφόμενο Uri.

```
private ValueCallback<Uri[]> fileCallback;

@Override
public boolean onShowFileChooser(
    WebView webView,
    ValueCallback<Uri[]> callback,
    FileChooserParams params) {

    fileCallback = callback;

    Intent intent = params.createIntent();
    startActivityForResult(intent, FILE_CHOOSER_REQUEST);

    return true;
}

// Όταν επιστρέφει το αποτέλεσμα
Uri[] result =
    WebChromeClient.FileChooserParams.parseResult(
        resultCode, data);

fileCallback.onReceiveValue(result);
```

Εικόνα 17 : Παράδειγμα χρήσης του file chooser σε WebView.

Η χρήση file chooser δημιουργεί ένα επιπλέον σύνορο εμπιστοσύνης, καθώς το Uri που επιστρέφεται στο WebView μπορεί να προέρχεται από διαφορετική εφαρμογή της συσκευής και πρέπει επομένως να αντιμετωπίζεται ως εξωτερικά ελεγχόμενο input. Η τεκμηρίωση της onShowFileChooser() επισημαίνει ότι το WebView δεν επιβάλλει από μόνο του περιορισμούς στα αρχεία που επιστρέφονται και ότι το αποτέλεσμα μπορεί να αναφέρεται σε resources στα οποία έχει πρόσβαση η host εφαρμογή [67].

Η συμπεριφορά αυτή αποκτά μεγαλύτερη σημασία όταν το WebView εμφανίζει περιεχόμενο που δεν ελέγχεται πλήρως από την εφαρμογή. Αν μια μη έμπιστη σελίδα μπορεί να ενεργοποιήσει file-selection request και η εφαρμογή επιστρέψει χωρίς επαρκή έλεγχο Uri προς ευαίσθητο resource, το αντίστοιχο περιεχόμενο μπορεί να καταστεί διαθέσιμο στο web layer μέσω των δικαιωμάτων που διαθέτει η ίδια η εφαρμογή [53], [67]. Το κρίσιμο σημείο επομένως βρίσκεται στον τρόπο με τον οποίο η εφαρμογή χειρίζεται το αποτέλεσμα του picker και στις δυνατότητες που μεταφέρονται μαζί με το επιστρεφόμενο Uri.

Για τον λόγο αυτό, τα αποτελέσματα του file-selection flow πρέπει να επικυρώνονται πριν παραδοθούν στο WebView. Η εφαρμογή μπορεί να λαμβάνει υπόψη τις παραμέτρους του FileChooserParams, όπως τον επιτρεπόμενο τύπο περιεχομένου και το mode επιλογής, και να ελέγχει τα Uri που επιστρέφονται ώστε να συμφωνούν με τη λειτουργία που αναμένεται. Η προέλευση του αποτελέσματος από το Android file picker

δεν αποτελεί από μόνη της εγγύηση ότι το resource είναι κατάλληλο ή ασφαλές για το web content που το ζήτησε [67].

Συνολικά, οι WebView file-access ευπάθειες βασίζονται στην ίδια γενική συνθήκη: το WebView λειτουργεί μέσα στο security context της εφαρμογής και μπορεί, αν έχει διαμορφωθεί με υπερβολικά permissive ρυθμίσεις, να αποτελέσει σημείο πρόσβασης από web content προς τοπικούς πόρους. Η πρόσβαση αυτή μπορεί να προκύψει μέσω file:// ή content:// URLs, JavaScript που εκτελείται σε local-file context ή μη ασφαλούς χειρισμού file-selection flows. Για τον περιορισμό της συγκεκριμένης επιφάνειας επίθεσης, το Android έχει μεταβάλει αρκετές από τις σχετικές προεπιλογές, έχει χαρακτηρίσει τις setAllowFileAccessFromFileURLs() και setAllowUniversalAccessFromFileURLs() ως deprecated και προτείνει μηχανισμούς όπως το WebViewAssetLoader για την ελεγχόμενη διάθεση τοπικών application resources [135], [136].

Η δυνατότητα εξωτερικού ελέγχου του URL αποκτά διαφορετική βαρύτητα όταν η εφαρμογή συνδέει τη φόρτωση του WebView με authentication material. Η WebView.loadUrl(String, Map<String,String>) επιτρέπει την αποστολή πρόσθετων HTTP headers μαζί με το request και μπορεί να χρησιμοποιηθεί για τη μεταφορά session tokens ή άλλων credentials προς το web layer. Σε μια τέτοια υλοποίηση, η εμπιστοσύνη στον προορισμό πρέπει να έχει καθοριστεί πριν το URL συνδυαστεί με ευαίσθητα headers, καθώς οι δύο τιμές παρέχονται ανεξάρτητα στην API [60].

```
String url = getIntent()  
    .getData()  
    .getQueryParameter("url");  
  
Map<String, String> headers = new HashMap<>();  
headers.put("Authorization", "Bearer " + getUserToken());  
  
webView.loadUrl(url, headers);
```

Εικόνα 18 : Παράδειγμα διαδρομής από εξωτερικά ελεγχόμενη παράμετρο Deep Link προς τη WebView.loadUrl(), όπου η εφαρμογή προσθέτει παράλληλα Authorization header.

Αν ο προορισμός προέρχεται από εξωτερικά ελεγχόμενη είσοδο, η εφαρμογή μπορεί να καταλήξει να αποστείλει authentication material σε host που έχει επιλέξει ο attacker. Η Oversecured παρουσιάζει αντίστοιχη περίπτωση όπου URL που μεταφέρεται μέσω Deep Link χρησιμοποιείται σε loadUrl() μαζί με Authorization header, μετατρέποντας το αρχικό arbitrary URL loading primitive σε διαρροή credentials

[65]. Η αξιολόγηση μιας τέτοιας διαδρομής πρέπει επομένως να λαμβάνει υπόψη και τα δεδομένα που συνοδεύουν το request, καθώς η επίπτωση μπορεί να είναι σημαντικά μεγαλύτερη από την απλή φόρτωση μη έμπιστου περιεχομένου.

Παρόμοια συμπεριφορά μπορεί να εμφανιστεί όταν το authentication state μεταφέρεται μέσω cookies. Η `CookieManager.setCookie()` συσχετίζει την τιμή του cookie με το URL που παρέχεται από την εφαρμογή, επομένως ο συγκεκριμένος προορισμός επηρεάζει το domain για το οποίο το cookie θα είναι διαθέσιμο [68].

```
String url = getIntent().getDataString();

CookieManager cookies = CookieManager.getInstance();
cookies.setCookie(url, "token=" + getUserToken());

webView.loadUrl(url);
```

Εικόνα 19 : Παράδειγμα συσχέτισης ευαίσθητου cookie με εξωτερικά ελεγχόμενο URL πριν από τη φόρτωσή του στο WebView.

Όταν το URL προέρχεται από μη έμπιστη πηγή, ένα sensitive cookie μπορεί να συνδεθεί με domain που βρίσκεται εκτός του ελέγχου της εφαρμογής και να παραμείνει διαθέσιμο στο WebView cookie store για επόμενες πλοηγήσεις. Η Oversecured επισημαίνει ότι το URL πρέπει να επικυρώνεται πριν από την εγκατάσταση authentication cookies και προτείνει τη χρήση προκαθορισμένου συνόλου trusted domains για τις περιπτώσεις όπου η εφαρμογή χρειάζεται να μεταφέρει credentials στο web layer [65].

2.4.2 Μη ασφαλής διαχείριση των Intents

Η επικοινωνία μεταξύ application components αποτελεί βασικό μέρος του μοντέλου λειτουργίας του Android και πραγματοποιείται σε μεγάλο βαθμό μέσω Intents. Ένα Intent μπορεί να ενεργοποιήσει Activities, Services ή Broadcast Receivers και να μεταφέρει πληροφορίες δρομολόγησης, όπως action και data, μαζί με πρόσθετα δεδομένα μέσω extras. Από πλευράς ασφάλειας, η ροή αυτή δημιουργεί ένα όριο εμπιστοσύνης μεταξύ διαφορετικών components και, σε αρκετές περιπτώσεις, μεταξύ διαφορετικών εφαρμογών, καθώς η ασφάλεια της επικοινωνίας εξαρτάται από το ποιος μπορεί να δημιουργήσει το Intent, ποιος μπορεί να το παραλάβει και ποια λειτουργία ενεργοποιείται ως αποτέλεσμα [59].

Ιδιαίτερη σημασία έχει η διάκριση μεταξύ explicit και implicit Intents. Στην πρώτη περίπτωση, το Intent προσδιορίζει άμεσα τον component ή το package που θα το παραλάβει, επομένως ο προορισμός είναι

γνωστός από τη στιγμή που δημιουργείται η επικοινωνία. Στα implicit Intents περιγράφεται η επιθυμητή ενέργεια και τα σχετικά δεδομένα χωρίς να ορίζεται συγκεκριμένος παραλήπτης, και το Android χρησιμοποιεί το Intent Resolution για να εντοπίσει components των εγκατεστημένων εφαρμογών των οποίων τα intent-filters αντιστοιχούν στο συγκεκριμένο request [59]. Η επιλογή του τελικού παραλήπτη εξαρτάται έτσι και από τα components που είναι διαθέσιμα στη συσκευή, γεγονός που πρέπει να λαμβάνεται υπόψη όταν η επικοινωνία μεταφέρει ευαίσθητα δεδομένα ή αναμένει αξιόπιστο αποτέλεσμα.

```
// Explicit Intent
Intent explicitIntent =
    new Intent(this, PaymentActivity.class);
startActivity(explicitIntent);

// Implicit Intent
Intent implicitIntent =
    new Intent(Intent.ACTION_VIEW);
implicitIntent.setData(
    Uri.parse("https://example.com/account")
);
startActivity(implicitIntent);
```

Εικόνα 20 : Παράδειγμα χρήσης explicit και implicit Intent.

Στο explicit παράδειγμα, το Intent κατευθύνεται απευθείας προς την PaymentActivity, ενώ στην implicit εκδοχή η εφαρμογή καθορίζει μόνο την ενέργεια και το URI και αφήνει το Android να επιλέξει τον κατάλληλο handler. Η ευελιξία αυτή είναι απαραίτητη για πολλές λειτουργίες της πλατφόρμας, εισάγει όμως ένα επιπλέον security consideration, καθώς άλλη εφαρμογή μπορεί να δηλώσει συμβατό intent-filter και να συμμετάσχει στη διαδικασία του resolution.

Όταν ένα implicit Intent χρησιμοποιείται σε security-sensitive ροή χωρίς να περιορίζεται ο τελικός παραλήπτης, μπορεί να προκύψει **Implicit Intent Hijacking**. Μια κακόβουλη εφαρμογή μπορεί να δηλώσει συμβατό intent-filter και να συμμετάσχει στη διαδικασία Intent Resolution, αποκτώντας πρόσβαση σε δεδομένα που δεν προορίζονταν για αυτήν ή επηρεάζοντας τη συνέχεια της ροής [69]. Η σοβαρότητα της περίπτωσης εξαρτάται από το περιεχόμενο του Intent και από τις λειτουργίες που εκτελούνται μετά την παράδοσή του.

Ιδιαίτερη προσοχή χρειάζονται και τα implicit flows που αναμένουν αποτέλεσμα από εξωτερικό component, όπως file ή content pickers. Σε αυτές τις περιπτώσεις, το component που επιλέγεται από το

Android μπορεί να επιστρέψει δεδομένα ή Uri που βρίσκονται υπό τον έλεγχό του, επομένως το αποτέλεσμα πρέπει να αντιμετωπίζεται ως μη έμπιστη εξωτερική είσοδος. Αν η εφαρμογή χρησιμοποιήσει το επιστρεφόμενο Uri χωρίς επαρκή επικύρωση, μπορεί να δημιουργηθεί νέα attacker-controlled διαδρομή προς λειτουργίες που προσπελούν αρχεία, Content Providers ή άλλους ευαίσθητους πόρους [79].

Αντίστοιχο ζήτημα εμφανίζεται όταν ένα exported Activity επιστρέφει δεδομένα προς τον caller μέσω setResult(). Όταν το επιστρεφόμενο Intent περιέχει ευαίσθητες πληροφορίες ή content:// URIs μαζί με flags παραχώρησης πρόσβασης, η εφαρμογή μπορεί να μεταφέρει δεδομένα ή δικαιώματα σε εξωτερικό caller που δεν θα είχε διαφορετικά πρόσβαση στους ίδιους πόρους [137].

Η ασφαλής χρήση implicit Intents απαιτεί επομένως σαφή έλεγχο τόσο του προορισμού όσο και των δεδομένων που επιστρέφονται από externally resolved components. Όταν η επικοινωνία προορίζεται για γνωστό παραλήπτη, είναι προτιμότερο να χρησιμοποιείται explicit Intent ή να περιορίζεται κατάλληλα το σύνολο των handlers, ενώ τα αποτελέσματα από implicit flows πρέπει να επικυρώνονται πριν χρησιμοποιηθούν σε security-sensitive λειτουργίες [69], [79], [137].

Η προηγούμενη κατηγορία αφορούσε κυρίως περιπτώσεις όπου η εφαρμογή δεν ελέγχει επαρκώς τον παραλήπτη ενός εξερχόμενου Intent ή την προέλευση του αποτελέσματος που επιστρέφεται. Στο **Intent Redirection**, η εξωτερική επιρροή ακολουθεί διαφορετική πορεία, στην οποία ένα component της εφαρμογής δέχεται δεδομένα από εξωτερικό caller και τα χρησιμοποιεί στη συνέχεια για την ενεργοποίηση κάποιας άλλης λειτουργίας. Αν ο caller μπορεί να επηρεάσει το Intent που τελικά εκτελείται, η εφαρμογή μπορεί να λειτουργήσει ως *confused deputy*, χρησιμοποιώντας τα δικά της δικαιώματα για την πραγματοποίηση ενέργειας που έχει καθοριστεί από μη έμπιστη πηγή [38].

```
Intent nestedIntent =  
    getIntent().getParcelableExtra("redirect_intent");  
  
if (nestedIntent != null) {  
    startActivity(nestedIntent);  
}
```

Εικόνα 21 : Παράδειγμα ενός intent redirection.

Χαρακτηριστική περίπτωση αποτελεί η μεταφορά ενός δεύτερου Intent ως nested object μέσα στα extras του αρχικού Intent. Το externally reachable component παραλαμβάνει την εξωτερική είσοδο και στη συνέχεια χρησιμοποιεί το nested Intent για να ενεργοποιήσει άλλο component, μετατρέποντας έτσι δεδομένα που προήλθαν από τον caller σε νέα εντολή εκτέλεσης. Η δυνατότητα επηρεασμού του

προορισμού ή άλλων security-relevant χαρακτηριστικών του δεύτερου Intent μπορεί να επιτρέψει πρόσβαση σε functionality που δεν είχε σχεδιαστεί ώστε να είναι άμεσα διαθέσιμη σε τρίτες εφαρμογές [38].

Ιδιαίτερη σημασία έχουν και οι δυνατότητες πρόσβασης που μπορεί να μεταφέρει ένα Intent μαζί με ένα content:// URI. Flags όπως τα FLAG_GRANT_READ_URI_PERMISSION και FLAG_GRANT_WRITE_URI_PERMISSION μπορούν να συνοδεύσουν το Intent με προσωρινή εξουσιοδότηση πάνω στον αντίστοιχο πόρο. Αν ένας redirected Intent επηρεάζεται από εξωτερικό caller, η αξιολόγηση πρέπει επομένως να εξετάζει και τα permissions που μεταφέρονται μαζί του, καθώς η εφαρμογή μπορεί να καταλήξει να παραχωρεί πρόσβαση σε δεδομένα που διαφορετικά προστατεύονται από το Android permission model [38], [85].

Η αντιμετώπιση του Intent Redirection βασίζεται στον περιορισμό της εξωτερικής επιρροής πριν το εισερχόμενο δεδομένο χρησιμοποιηθεί για νέα component activation. Όπου είναι δυνατό, η εφαρμογή μπορεί να δημιουργεί νέο Intent από ήδη επικυρωμένες τιμές και να ορίζει ρητά τον επιτρεπόμενο προορισμό. Για περιπτώσεις όπου απαιτείται η περαιτέρω χρήση εισερχόμενου Intent, το AndroidX παρέχει τον IntentSanitizer, μέσω του οποίου μπορούν να καθοριστούν allowlists για τα χαρακτηριστικά που επιτρέπεται να διατηρηθούν πριν από την εκτέλεσή του [70].

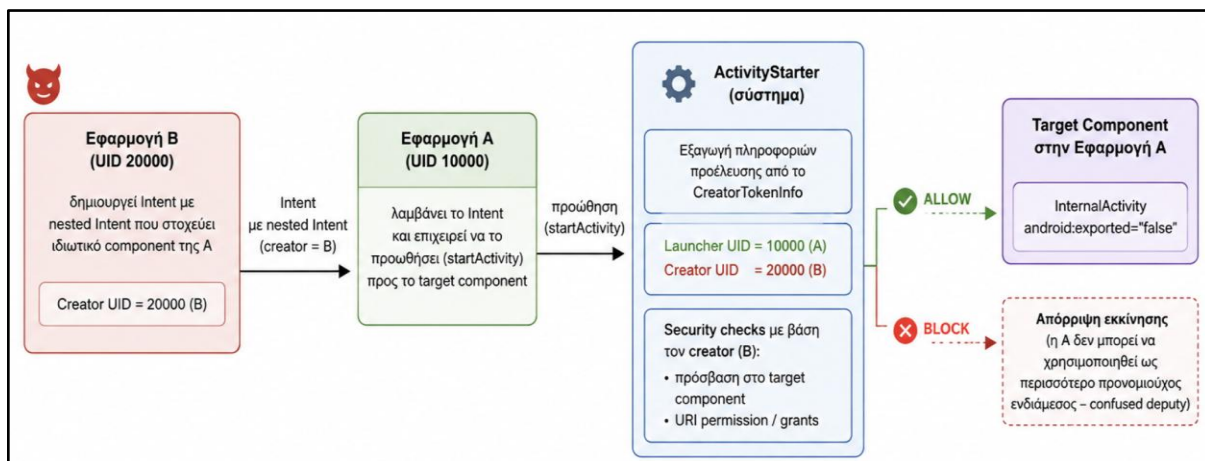
```
Intent safeIntent =
    new IntentSanitizer.Builder()
        .allowComponent(
            new ComponentName(
                "com.example.app",
                "com.example.app.SafeActivity"))
        .allowAction("com.example.app.ACTION_OPEN")
        .build()
        .sanitizeByThrowing(nestedIntent);

startActivity(safeIntent);
```

Εικόνα 22 : Παράδειγμα περιορισμού nested Intent μέσω IntentSanitizer, με ρητή allowlist για τα χαρακτηριστικά που επιτρέπεται να χρησιμοποιηθούν.

Η πλατφόρμα Android έχει εισαγάγει μηχανισμούς που βοηθούν στον εντοπισμό και τον περιορισμό μη ασφαλών Intent forwarding patterns. Από το Android 12, το StrictMode.detectUnsafeIntentLaunch() μπορεί να επισημάνει περιπτώσεις όπου ένα Intent που προέρχεται από εξωτερικά δεδομένα χρησιμοποιείται για την ενεργοποίηση άλλου component [38], ενώ στο Android 16 προστέθηκαν επιπλέον περιορισμοί για ορισμένες μορφές Intent Redirection που μπορούν να οδηγήσουν σε private components ή σε μεταφορά URI access [39]. Οι προσαυσίες αυτές μειώνουν την πιθανότητα εκμετάλλευσης γνωστών

patterns, ενώ η εφαρμογή εξακολουθεί να χρειάζεται να καθορίζει ποια destinations και δεδομένα θεωρούνται έγκυρα για τη δική της λειτουργία.



Εικόνα 23 : Παράδειγμα προστασίας απέναντι σε Intent Redirection στο Android 16.

Συναφής μηχανισμός μεταβίβασης εξουσιοδότησης είναι το `PendingIntent`, μέσω του οποίου μια εφαρμογή επιτρέπει σε άλλο component να εκτελέσει αργότερα μια ενέργεια με τα δικά της δικαιώματα. Από πλευράς ασφάλειας, ιδιαίτερη σημασία έχει το κατά πόσο το εσωτερικό Intent παραμένει τροποποιήσιμο και αν η ίδια εξουσιοδότηση μπορεί να χρησιμοποιηθεί περισσότερες από μία φορές. Ένα mutable `PendingIntent` μπορεί, υπό ορισμένες συνθήκες, να επιτρέψει την τροποποίηση πεδίων που δεν είχαν καθοριστεί κατά τη δημιουργία του, ενώ η επαναχρησιμοποίησή του μπορεί να οδηγήσει σε επανάληψη ενέργειας που είχε σχεδιαστεί για μία μόνο εκτέλεση [37]. Για τον περιορισμό αυτών των περιπτώσεων προτιμάται immutable `PendingIntent` όταν δεν απαιτείται μεταβολή του περιεχομένου του και `FLAG_ONE_SHOT` όταν η αντίστοιχη ενέργεια πρέπει να εκτελεστεί μόνο μία φορά.

Ένα ακόμη σημείο που χρειάζεται προσοχή είναι η επεξεργασία νέων Intents από ήδη υπάρχον Activity instance. Ανάλογα με το launch mode και τα flags που χρησιμοποιούνται, ένα Activity μπορεί να λάβει νέο Intent μέσω της `onNewIntent()` αντί να δημιουργηθεί εκ νέου και να περάσει από την `onCreate()`. Το νέο Intent αποτελεί ανεξάρτητη είσοδο και χρειάζεται να υποβάλλεται στους ίδιους ελέγχους validation και authorization που εφαρμόζονται κατά την αρχική ενεργοποίηση του component. Αν οι έλεγχοι πραγματοποιούνται μόνο στην `onCreate()`, μπορεί να παραμείνει εναλλακτική διαδρομή μέσω της οποίας εξωτερικά ελεγχόμενα δεδομένα φτάνουν στην ίδια λειτουργικότητα χωρίς την αναμενόμενη επαλήθευση [71].

Μια ιδιαίτερα σημαντική πηγή externally controlled δεδομένων αποτελούν τα Deep Links, μέσω των οποίων ένα εξωτερικό URI μπορεί να ενεργοποιήσει Activity της εφαρμογής και να μεταφέρει πληροφορίες στην εσωτερική της λογική. Από πλευράς vulnerability analysis, το Deep Link λειτουργεί κυρίως ως entry point, καθώς path segments, query parameters και άλλα στοιχεία του URI μπορούν να επηρεάζονται από τον caller και να χρησιμοποιηθούν στη συνέχεια για navigation, επιλογή λειτουργιών ή πρόσβαση σε δεδομένα. [51]

Ειδική περίπτωση αποτελούν τα Android App Links, τα οποία βασίζονται σε HTTP(S) URLs και χρησιμοποιούν domain verification ώστε το Android να μπορεί να τα συσχετίσει με τη σωστή εφαρμογή. Η επαλήθευση αυτή αφορά κυρίως τη σχέση ανάμεσα στο domain και στην εφαρμογή που θα χειριστεί το link· τα δεδομένα που περιέχει το ίδιο το URI εξακολουθούν να προέρχονται από εξωτερική πηγή. Για τον λόγο αυτό, οι παράμετροι ενός έγκυρου App Link χρειάζονται application-level validation και authorization πριν χρησιμοποιηθούν σε security-sensitive λειτουργίες [51].

Η σημασία αυτής της διάκρισης γίνεται πιο εμφανής σε εφαρμογές που χρησιμοποιούν το Jetpack Navigation Component, όπου οι εσωτερικοί προορισμοί και οι διαδρομές μεταξύ τους περιγράφονται σε navigation graph. Το graph μπορεί να περιλαμβάνει destinations και arguments που χρησιμοποιούνται κατά την πλοήγηση, επιτρέποντας στην εφαρμογή να μεταφέρει δεδομένα από το αρχικό entry point προς εσωτερικά Fragments ή Activities [72].

```
<navigation
  android:id="@+id/main_graph"
  app:startDestination="@id/homeFragment">

  <fragment
    android:id="@+id/homeFragment"
    android:name="com.example.HomeFragment" />

  <fragment
    android:id="@+id/privateFragment"
    android:name="com.example.PrivateFragment" />

</navigation>
```

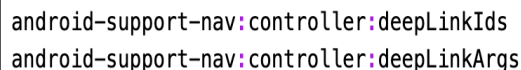
Εικόνα 24 : Παράδειγμα ορισμού ενός Navigation Graph με χρήση του Jetpack Navigation.

Η παρουσία ενός Fragment στο navigation graph δεν το καθιστά από μόνη της externally reachable, καθώς τα Fragments δεν αποτελούν ανεξάρτητα Android components που ενεργοποιούνται απευθείας μέσω IPC. Η εξωτερική είσοδος πρέπει πρώτα να φτάσει σε προσβάσιμο Activity και στη συνέχεια να επηρεάσει την εσωτερική διαδικασία πλοήγησης που επιλέγει τον τελικό destination. Η ανάλυση μιας τέτοιας διαδρομής χρειάζεται επομένως να ακολουθεί τόσο το αρχικό component boundary όσο και την εσωτερική navigation

logic, ώστε να διαπιστωθεί αν attacker-controlled δεδομένα μπορούν πράγματι να οδηγήσουν σε λειτουργικότητα που δεν είχε σχεδιαστεί για άμεση εξωτερική πρόσβαση [72].

Η υποστήριξη Deep Links στο Jetpack Navigation ενσωματώνεται στην εσωτερική διαδικασία πλοήγησης της εφαρμογής μέσω του NavController. Η μέθοδος `handleDeepLink(Intent)` επεξεργάζεται το εισερχόμενο Intent και, όταν αυτό περιέχει αναγνωρίσιμες πληροφορίες πλοήγησης, μπορεί να κατασκευάσει την κατάλληλη διαδρομή προς destination του navigation graph. Η επεξεργασία πραγματοποιείται συνήθως αυτόματα κατά την αρχική ενεργοποίηση του Activity, ενώ σε περιπτώσεις επαναχρησιμοποίησης υπάρχοντος instance το νέο Intent μπορεί να προωθηθεί στο Navigation Component μέσω της `onNewIntent()` [72].

Ιδιαίτερο ενδιαφέρον από πλευράς ασφάλειας παρουσιάζει το γεγονός ότι η εσωτερική αναπαράσταση ενός Navigation deep link μπορεί να περιλαμβάνει πληροφορίες πλοήγησης και μέσα στα extras του Intent. Μέσω αυτών μπορούν να μεταφερθούν αναγνωριστικά destinations του navigation graph και arguments που προορίζονται για τους αντίστοιχους προορισμούς, δυνατότητα που χρησιμοποιείται και από την επίσημη `NavDeepLinkBuilder` για τη δημιουργία deep links προς συγκεκριμένα σημεία της εφαρμογής [72].



```
android-support-nav:controller:deeplinkIds
android-support-nav:controller:deeplinkArgs
```

Εικόνα 25 : Παράδειγμα των εσωτερικών πληροφοριών που χρησιμοποιούνται από το Jetpack Navigation για την επιλογή destination και τη μεταφορά arguments.

Η επεξεργασία αυτών των δεδομένων δημιούργησε σε επηρεαζόμενες εκδόσεις του Navigation Component μια πρόσθετη επιφάνεια εισόδου. Τεχνική ανάλυση της Positive Technologies έδειξε ότι ειδικά διαμορφωμένο Intent προς exported Activity μπορούσε να επηρεάσει την εσωτερική πλοήγηση μέσω destination identifiers που μεταφέρονταν στα extras. Η `handleDeepLink()` χρησιμοποιούσε τις πληροφορίες αυτές για την αναζήτηση κόμβων του navigation graph και μπορούσε, υπό τις συνθήκες της συγκεκριμένης ροής, να οδηγήσει σε Fragment που δεν είχε σχεδιαστεί από τον developer ως externally reachable destination [73].

Η επίδραση μιας τέτοιας διαδρομής δεν περιορίζεται στην επιλογή του τελικού Fragment, καθώς μαζί με την πλοήγηση μπορούν να μεταφερθούν και arguments που επηρεάζουν τη λειτουργία του destination. Αν το Fragment χρησιμοποιεί τις παραμέτρους αυτές για την επιλογή δεδομένων ή την ενεργοποίηση κάποιας λειτουργίας, η εξωτερική επιρροή μπορεί να συνεχιστεί και μετά την ολοκλήρωση της πλοήγησης, γεγονός

που καθιστά αναγκαία την αξιολόγηση τόσο της reachability του destination όσο και του τρόπου με τον οποίο επεξεργάζεται τα εισερχόμενα arguments [73].

Στην έκδοση AndroidX Navigation 2.10.0 η συμπεριφορά της `handleDeepLink()` ενισχύθηκε ώστε η επεξεργασία των explicit deep-link δεδομένων να περιορίζεται σε αναγνωρισμένες περιπτώσεις, μειώνοντας τη δυνατότητα χρήσης πλαστογραφημένων Intent-based navigation δεδομένων για την επίτευξη αντίστοιχων διαδρομών [74]. Το παράδειγμα αυτό δείχνει ότι frameworks εσωτερικής πλοήγησης μπορούν να προσθέσουν επιπλέον επίπεδα δρομολόγησης μετά το αρχικό Android component boundary, επομένως η ασφάλεια μιας Deep Link ροής χρειάζεται να εξετάζει και τον τελικό destination μαζί με τα arguments που φτάνουν σε αυτόν [73], [74].

2.4.3 Μη ασφαλής έκθεση των Services και Broadcast Receivers

Οι προηγούμενες περιπτώσεις επικεντρώθηκαν κυρίως στον τρόπο με τον οποίο τα Intents μεταφέρονται και επηρεάζουν την εσωτερική ροή της εφαρμογής. Η ασφάλεια της επικοινωνίας μεταξύ εφαρμογών εξαρτάται όμως και από το ίδιο το component που εκτίθεται προς το εξωτερικό περιβάλλον, με ιδιαίτερο ενδιαφέρον να παρουσιάζουν τα Services και τα Broadcast Receivers, καθώς μπορούν να αποτελέσουν σημεία εισόδου χωρίς να διαθέτουν άμεση γραφική διεπαφή. Όταν ένα τέτοιο component είναι externally reachable, τα Intents, οι IPC κλήσεις και τα δεδομένα που δέχεται πρέπει να αντιμετωπίζονται ως μη έμπιστα και να ελέγχονται πριν επηρεάσουν ευαίσθητη λειτουργικότητα [76], [40].

Τα Services χρησιμοποιούνται για την εκτέλεση εργασιών στο παρασκήνιο και μπορούν να λειτουργούν είτε ως started είτε ως bound services. Στην πρώτη περίπτωση, η επικοινωνία πραγματοποιείται μέσω ενός Intent που παραδίδεται στο `onStartCommand()`, ενώ στα bound services ο client συνδέεται μέσω `bindService()` και αλληλεπιδρά με το interface που παρέχεται από το IBinder [75]. Η διάκριση αυτή επηρεάζει άμεσα και την επιφάνεια επίθεσης, καθώς ένα started Service δέχεται κυρίως δεδομένα μέσω του εισερχόμενου Intent, ενώ σε ένα bound Service πρέπει να εξετάζονται και οι λειτουργίες που εκτίθενται μέσω του Binder interface.

Όταν ένα Service προορίζεται για εσωτερική χρήση, η περιορισμένη εξωτερική προσβασιμότητα μειώνει σημαντικά τον κίνδυνο μη εξουσιοδοτημένων κλήσεων. Για Services που πρέπει να είναι διαθέσιμα σε άλλες εφαρμογές, το IPC boundary χρειάζεται να προστατεύεται με κατάλληλα permissions και, όπου απαιτείται, με πρόσθετο έλεγχο του caller πριν από την εκτέλεση ευαίσθητων ενεργειών [76].

```

public int onStartCommand(Intent intent, int flags, int startId) {
    String account = intent.getStringExtra("account");
    deleteAccountData(account);

    return START_NOT_STICKY;
}

```

Εικόνα 26 : Παράδειγμα Service που λαμβάνει externally controlled τιμή μέσω Intent extra.

Αν το Service της Εικόνας 26 είναι προσβάσιμο από τρίτες εφαρμογές και χρησιμοποιεί το εισερχόμενο extra για την ενεργοποίηση μιας λειτουργίας όπως η deleteAccountData() χωρίς προηγούμενο authorization check, ένας εξωτερικός caller μπορεί να προκαλέσει την εκτέλεση της ενέργειας μέσα στο security context της εφαρμογής. Η κρίσιμη διαδρομή στην περίπτωση αυτή είναι η σύνδεση ενός μη έμπιστου IPC request με λειτουργία που διαθέτει αυξημένες δυνατότητες, γεγονός που μπορεί να μετατρέψει το Service σε ενδιάμεσο μέσω του οποίου εκτελείται ενέργεια που ο caller δεν θα μπορούσε να πραγματοποιήσει απευθείας [81].

Στα bound Services, η επιφάνεια επικοινωνίας δεν περιορίζεται στο αρχικό Intent που χρησιμοποιείται για το bindService(), καθώς το Service μπορεί να εκθέτει λειτουργίες μέσω του IBinder που επιστρέφεται από την onBind(). Σε επικοινωνία μεταξύ διαφορετικών processes, ο Binder αποτελεί τον βασικό IPC μηχανισμό του Android, ενώ για πιο σύνθετες διεπαφές μπορεί να χρησιμοποιηθεί το AIDL ώστε μια εφαρμογή να ορίσει methods που μπορούν να κληθούν από εξωτερικούς clients [83]. Αντίστοιχα, μηχανισμοί όπως το Messenger βασίζονται επίσης στον Binder και μεταφέρουν μηνύματα προς το Service, επομένως τα δεδομένα που φτάνουν μέσω αυτών των διεπαφών πρέπει να αντιμετωπίζονται ως εξωτερικό IPC input [76], [77].

Από πλευράς ασφάλειας, κάθε λειτουργία που εκτίθεται μέσω Binder, AIDL ή Messenger αποτελεί ξεχωριστό σημείο εισόδου και χρειάζεται να αξιολογείται ανάλογα με την ενέργεια που εκτελεί και τα δεδομένα που δέχεται. Η ύπαρξη ελέγχου κατά το αρχικό binding δεν καλύπτει απαραίτητα όλες τις επόμενες IPC κλήσεις, γι' αυτό σε λειτουργίες που χειρίζονται ευαίσθητα δεδομένα ή προνόμια μπορεί να απαιτείται έλεγχος της ταυτότητας του caller και των αντίστοιχων permissions σε επίπεδο method [76].

Ιδιαίτερη προσοχή απαιτείται στη χρήση της Binder.clearCallingIdentity(), η οποία αφαιρεί προσωρινά την ταυτότητα του αρχικού Binder caller από το τρέχον thread και επιτρέπει στις επόμενες ενέργειες να εκτελεστούν με την ταυτότητα του process της εφαρμογής [77]. Η χρήση της μπορεί να είναι απαραίτητη σε ορισμένες περιπτώσεις, όμως οι έλεγχοι authorization πρέπει να έχουν ολοκληρωθεί πριν από αυτή τη μετάβαση, καθώς διαφορετικά ένας μη έμπιστος caller μπορεί να προκαλέσει την εφαρμογή να εκτελέσει λειτουργία με δικαιώματα που ο ίδιος δεν διαθέτει. Το pattern αυτό αποτελεί χαρακτηριστική μορφή

confused deputy και δείχνει γιατί η διαχείριση της caller identity είναι κρίσιμο μέρος της ασφάλειας των Binder-based Services [76], [77].

Η επιλογή του Service που θα δεχθεί μια κλήση αποτελεί και αυτή μέρος του security model της επικοινωνίας. Το Android συνιστά τη χρήση explicit Intents για την εκκίνηση Services, καθώς ένα implicit Intent αφήνει την επιλογή του τελικού recipient στη διαδικασία resolution και μπορεί έτσι να επιτρέψει σε διαφορετική εφαρμογή να δηλώσει συμβατό Service και να συμμετάσχει στη ροή. Από το Android 5.0 και μετά, η χρήση implicit Intent με bindService() απορρίπτεται από το σύστημα, περιορίζοντας αυτού του είδους την ασάφεια στην επιλογή του IPC endpoint [75].

Στα bound Services, η αξιολόγηση πρέπει να επεκτείνεται και στο interface που εκτίθεται μέσω Binder ή AIDL. Ένα AIDL interface μπορεί να προσφέρει απευθείας πρόσβαση σε methods που επιστρέφουν δεδομένα ή εκτελούν λειτουργίες με τα δικαιώματα της εφαρμογής, επομένως η ύπαρξη της διεπαφής δεν παρέχει από μόνη της καμία εγγύηση εξουσιοδότησης. Η πρόσβαση στις εκτεθειμένες methods χρειάζεται να περιορίζεται με permissions και κατάλληλους caller checks, ιδιαίτερα όταν η λειτουργία τους σχετίζεται με ευαίσθητα δεδομένα ή privileged actions [76], [83].

```
interface IAccountService {
    String getAuthToken();
    void deleteBackup(String name);
}

@Override
public String getAuthToken() {
    return accountManager.getToken();
}
```

Εικόνα 27 : Παράδειγμα Binder/AIDL interface που εκθέτει security-sensitive λειτουργία.

Αν οποιαδήποτε εγκατεστημένη εφαρμογή μπορεί να αποκτήσει το αντίστοιχο Binder και να καλέσει μια method όπως η getAuthToken(), το πρόβλημα βρίσκεται στην απουσία authorization πάνω στην IPC κλήση και όχι στο ίδιο το AIDL mechanism. Το AIDL καθορίζει τη μορφή της επικοινωνίας μεταξύ processes, ενώ η εφαρμογή παραμένει υπεύθυνη για τον έλεγχο του caller πριν εκτελεστεί η ευαίσθητη λειτουργία [76].

Το ίδιο trust boundary μπορεί να αποτύχει και προς την αντίθετη κατεύθυνση, όταν η εφαρμογή αποστέλλει δεδομένα σε Binder ή callback implementation που ελέγχεται από τρίτη εφαρμογή. Σε αυτή την περίπτωση, η ευάλωτη ροή αφορά την εμπιστοσύνη που αποδίδεται στον απομακρυσμένο IPC peer και μπορεί να οδηγήσει σε διαρροή δεδομένων προς μη εξουσιοδοτημένο παραλήπτη. Η Oversecured διαχωρίζει

αντίστοιχα αυτές τις περιπτώσεις από τα exposed AIDL interfaces, όπου ο attacker καλεί απευθείας τις methods που εκθέτει η ίδια η εφαρμογή [82].

Για Services που προορίζονται να χρησιμοποιούνται μόνο από συγκεκριμένες συνεργαζόμενες εφαρμογές, η ταυτότητα του caller χρειάζεται να προκύπτει από στοιχεία που συνδέονται με την πραγματική IPC κλήση και όχι από πληροφορίες που μπορούν να δηλωθούν αυθαίρετα από τον client. Στις Binder επικοινωνίες μπορεί να χρησιμοποιηθεί το Binder.getCallingUid() για την αναγνώριση του UID του caller και, όταν η σχέση εμπιστοσύνης βασίζεται στην εφαρμογή ή στον developer που την έχει υπογράψει, η πληροφορία αυτή μπορεί να συνδυαστεί με ελέγχους μέσω του PackageManager και των signing certificates. Για εφαρμογές που ανήκουν στον ίδιο οργανισμό, τα signature-level permissions παρέχουν πιο άμεσο τρόπο περιορισμού της πρόσβασης σε γνωστούς και υπογεγραμμένους clients [76], [77].

Παρόμοιο trust boundary εμφανίζεται και στους Broadcast Receivers, οι οποίοι δέχονται broadcast Intents και ενεργοποιούν την onReceive() όταν ικανοποιούνται οι αντίστοιχες συνθήκες. Ένας receiver μπορεί να δηλωθεί στο Manifest ή να καταχωριστεί δυναμικά κατά την εκτέλεση και, όταν είναι προσβάσιμος από άλλες εφαρμογές, κάθε εισερχόμενο broadcast πρέπει να θεωρείται εξωτερική είσοδος. Η ασφάλεια της ροής εξαρτάται επομένως από τους περιορισμούς που εφαρμόζονται στον αποστολέα και από τη λειτουργία που ενεργοποιείται όταν το Intent φτάσει στον receiver [40].

Η πιο προβληματική περίπτωση εμφανίζεται όταν externally reachable receiver χρησιμοποιεί action ή extras ενός broadcast για την εκτέλεση security-sensitive λειτουργίας χωρίς επαρκή authorization. Η γνώση ενός action string καθορίζει ποια λειτουργία ζητείται, δεν αποδεικνύει όμως την ταυτότητα της εφαρμογής που έστειλε το Intent, επομένως ένας τρίτος caller μπορεί να κατασκευάσει αντίστοιχο broadcast όταν δεν υπάρχει πρόσθετος μηχανισμός προστασίας [78].

```
public class AccountReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if ("com.example.DELETE_BACKUP".equals(intent.getAction())) {
            String name = intent.getStringExtra("backup");
            deleteInternalBackup(name);
        }
    }
}

// Στο manifest
<receiver
    android:name=".AccountReceiver"
    android:exported="true">
    <intent-filter>
        <action android:name="com.example.DELETE_BACKUP" />
    </intent-filter>
</receiver>
```

Εικόνα 28 : Παράδειγμα exported BroadcastReceiver που επεξεργάζεται δεδομένα από εξωτερικό Intent.

Στο παράδειγμα της Εικόνας 28, η τιμή που μεταφέρεται στο broadcast επηρεάζει την εκτέλεση της `deleteInternalBackup()`. Αν ο receiver είναι προσβάσιμος από τρίτες εφαρμογές και δεν προστατεύεται με κατάλληλο permission ή άλλο authorization check, ένας εξωτερικός caller μπορεί να προκαλέσει την εκτέλεση της λειτουργίας μέσα στο security context της εφαρμογής. Η αξιολόγηση της περίπτωσης εξαρτάται επομένως από τη downstream ενέργεια που ενεργοποιείται και από τα δικαιώματα που διαθέτει η εφαρμογή κατά την εκτέλεσή της [78], [81].

Η εξωτερική προσβασιμότητα ενός BroadcastReceiver καθορίζεται ανάλογα με τον τρόπο με τον οποίο έχει καταχωριστεί. Στους receivers που δηλώνονται στο Manifest χρησιμοποιείται το `android:exported`, ενώ στους dynamically registered receivers οι σύγχρονες εκδόσεις του Android παρέχουν τα flags `RECEIVER_EXPORTED` και `RECEIVER_NOT_EXPORTED`, μέσω των οποίων ο developer δηλώνει αν ο receiver προορίζεται να δέχεται broadcasts και από άλλες εφαρμογές [40]. Από το Android 13 (API level 33) η επιλογή αυτή μπορεί να δηλώνεται ρητά κατά το runtime registration και, για εφαρμογές που στοχεύουν Android 14 (API level 34) ή νεότερο, η χρήση ενός από τα δύο flags είναι υποχρεωτική για context-registered receivers που δεν περιορίζονται αποκλειστικά σε system broadcasts [42].

Όταν ένας receiver χρειάζεται να είναι προσβάσιμος από άλλες εφαρμογές, η δυνατότητα αποστολής broadcasts μπορεί να περιοριστεί μέσω permissions. Σε επικοινωνία μεταξύ εφαρμογών του ίδιου developer, τα signature-level permissions επιτρέπουν την πρόσβαση μόνο σε εφαρμογές που έχουν υπογραφεί με έμπιστο certificate, ενώ για ορισμένα system events το Android χρησιμοποιεί protected broadcasts ώστε να περιορίζει ποιοι senders μπορούν να εκτέλεσουν συγκεκριμένα actions [78]. Με τον τρόπο αυτό, η έκθεση του receiver μπορεί να διατηρηθεί χωρίς να θεωρείται κάθε εφαρμογή της συσκευής εξουσιοδοτημένος αποστολέας.

Η ταυτοποίηση του sender δεν ακολουθεί όμως το ίδιο μοντέλο με μια άμεση Binder IPC κλήση. Η χρήση του `Binder.getCallingUid()` μέσα στην `BroadcastReceiver.onReceive()` δεν παρέχει αξιόπιστη ταυτότητα της εφαρμογής που δημιούργησε το broadcast, επειδή η εκτέλεση του receiver δεν διατηρεί τις ίδιες caller-identity semantics που υπάρχουν σε μια Binder method call [78]. Για τον λόγο αυτό, οι μηχανισμοί authorization των Broadcast Receivers πρέπει να βασίζονται κυρίως στους περιορισμούς που παρέχει η ίδια η broadcast επικοινωνία, όπως permissions και ελεγχόμενη έκθεση του receiver, αντί να μεταφέρονται αυτούσιοι οι caller checks που χρησιμοποιούνται σε bound Services.

Το trust boundary των Broadcast Receivers αφορά και τα δεδομένα που εκπέμπει η ίδια η εφαρμογή. Ένα implicit broadcast που περιέχει ευαίσθητες πληροφορίες μπορεί να παραληφθεί από οποιαδήποτε εφαρμογή διαθέτει συμβατό receiver, εφόσον ο αποστολέας δεν έχει περιορίσει κατάλληλα τους

επιτρεπόμενους παραλήπτες. Η `sendBroadcast()` επιτρέπει την απαίτηση συγκεκριμένου `permission`, ενώ η αποστολή μπορεί επίσης να περιοριστεί σε γνωστό `package` ή `component` όταν ο προορισμός είναι προκαθορισμένος. Οι μηχανισμοί αυτοί είναι ιδιαίτερα σημαντικοί όταν το `Intent` μεταφέρει `credentials`, `tokens` ή άλλα δεδομένα που δεν προορίζονται για γενική διανομή [40].

Ιδιαίτερη κατηγορία αποτελούν τα `sticky broadcasts`, στα οποία το `Intent` διατηρείται από το σύστημα ώστε να μπορεί να παραδοθεί και σε `receivers` που καταχωρίζονται μετά την αρχική εκπομπή. Η συμπεριφορά αυτή δεν παρέχει κατάλληλες εγγυήσεις για τη μεταφορά ευαίσθητων δεδομένων και μπορεί να εκθέσει το περιεχόμενο του `broadcast` σε εφαρμογές που δεν αποτελούσαν τον αρχικό προορισμό της επικοινωνίας. Για τον λόγο αυτό, οι σχετικές APIs έχουν χαρακτηριστεί `deprecated` και δεν συνιστώνται για `security-sensitive` πληροφορίες [80].

Η αξιολόγηση `Services` και `Broadcast Receivers` χρειάζεται επομένως να ακολουθεί τη συνολική διαδρομή που δημιουργείται από το εξωτερικό `IPC entry point` μέχρι τη λειτουργία ή τα δεδομένα που τελικά επηρεάζονται. Η εξωτερική προσβασιμότητα ενός `component` αποκτά σημασία όταν συνδυάζεται με ανεπαρκείς ελέγχους πρόσβασης και επιτρέπει σε μη έμπιστο `caller` να ενεργοποιήσει λειτουργικότητα ή να αποκτήσει πληροφορία πέρα από όσα προβλέπει η σχεδίαση της εφαρμογής [78], [81].

2.4.4 Μη ασφαλής αποσειριοποίηση και ανακατασκευή αντικειμένων

Η μεταφορά δεδομένων μεταξύ `components` ή `processes` δεν περιορίζεται σε απλούς τύπους, καθώς πολλές εφαρμογές χρειάζεται να ανταλλάσσουν πιο σύνθετα αντικείμενα μαζί με την εσωτερική τους κατάσταση. Για τον σκοπό αυτό χρησιμοποιούνται μηχανισμοί `serialization` και `deserialization`, μέσω των οποίων ένα αντικείμενο μετατρέπεται σε κατάλληλη αναπαράσταση και στη συνέχεια ανακατασκευάζεται όταν χρειαστεί. Στο Android χρησιμοποιούνται τόσο ο γενικός μηχανισμός `Serializable` της Java όσο και οι `Parcel` και `Parcelable`, οι οποίοι είναι σχεδιασμένοι κυρίως για αποδοτική μεταφορά δεδομένων μέσω `IPC` [54], [87]. Από πλευράς ασφάλειας, το κρίσιμο σημείο εμφανίζεται όταν η `serialized` αναπαράσταση μπορεί να επηρεαστεί από μη έμπιστη πηγή, καθώς η διαδικασία ανακατασκευής μπορεί να επηρεάσει άμεσα την κατάσταση και τη συμπεριφορά των αντικειμένων που δημιουργούνται [54], [86].

Στον μηχανισμό της Java, μια κλάση που υλοποιεί το `Serializable` μπορεί να αποθηκευτεί και αργότερα να ανακατασκευαστεί μέσω της `ObjectInputStream`. Κατά τη `deserialization` αποκαθίσταται η κατάσταση του αντικειμένου και μπορούν να ενεργοποιηθούν ειδικοί μηχανισμοί που έχει ορίσει η ίδια η κλάση, όπως η `readObject()`. Αν τα δεδομένα που εισέρχονται στη διαδικασία ελέγχονται από `attacker` και η λογική της ανακατασκευής χρησιμοποιεί τις αποκατεστημένες τιμές σε `security-sensitive` ενέργειες, η `deserialization` μπορεί να προκαλέσει ανεπιθύμητη συμπεριφορά ήδη κατά τη δημιουργία του αντικειμένου [54], [86].

Στο Android, η inter-component και inter-process επικοινωνία βασίζεται σε μεγάλο βαθμό στον μηχανισμό Parcel, ο οποίος έχει σχεδιαστεί για τη μεταφορά δεδομένων μέσω Binder IPC. Application-defined αντικείμενα μπορούν να υλοποιούν το Parcelable, γράφοντας την κατάστασή τους σε ένα Parcel και ανακατασκευάζοντάς την στην πλευρά του παραλήπτη [87], [88]. Επειδή τα δεδομένα αυτά μπορούν να διασχίζουν application boundaries, η αξιολόγηση πρέπει να εξετάζει τόσο την προέλευση του Parcel όσο και τον τρόπο με τον οποίο οι τιμές που περιέχει χρησιμοποιούνται κατά την ανακατασκευή του αντικειμένου.

Κάθε κλάση που υλοποιεί το Parcelable παρέχει ένα CREATOR, μέσω του οποίου το Android μπορεί να ανακατασκευάσει νέα instances από τα δεδομένα ενός Parcel. Η `createFromParcel()` διαβάζει τις serialized τιμές και δημιουργεί το αντίστοιχο αντικείμενο, επομένως η ασφάλεια της διαδικασίας εξαρτάται από τις υποθέσεις που κάνει ο συγκεκριμένος κώδικας σχετικά με τον τύπο, τη μορφή και το περιεχόμενο των δεδομένων που λαμβάνει [88], [89]. Όταν το Parcel προέρχεται από εξωτερικό IPC boundary, οι τιμές αυτές πρέπει να θεωρούνται μη έμπιστες.

Η σημασία του ζητήματος γίνεται μεγαλύτερη επειδή Parcelable και Serializable αντικείμενα μεταφέρονται συχνά μέσα σε Intent extras και Bundle. Ένα εισερχόμενο Bundle μπορεί να περιέχει περισσότερα αντικείμενα από εκείνα που η application logic αναμένει να χρησιμοποιήσει, ενώ η επεξεργασία του μπορεί να οδηγήσει στην αποσειριοποίηση δεδομένων που προστέθηκαν από τον εξωτερικό caller. Το Android επισημαίνει για τον λόγο αυτό ότι ένα κακόβουλο extra μπορεί να επηρεάσει τη διαδικασία deserialization ακόμη και όταν το αντίστοιχο key δεν αποτελεί μέρος της κανονικής επιχειρησιακής ροής του component [54]. Σε externally reachable Activities, Services ή Broadcast Receivers, ολόκληρη η serialized είσοδος χρειάζεται επομένως να αντιμετωπίζεται ως μέρος της επιφάνειας επίθεσης.

Ιστορικά, ευπάθειες αυτής της κατηγορίας έχουν δείξει ότι η ανακατασκευή μη αναμενόμενων Parcelable ή Serializable αντικειμένων μπορεί να έχει σοβαρότερες συνέπειες όταν στο process υπάρχει κλάση της οποίας η deserialization logic χειρίζεται μη ασφαλώς τα ελεγχόμενα δεδομένα [54], [89]. Για τον περιορισμό του κινδύνου, οι νεότερες Android APIs παρέχουν type-safe παραλλαγές για την ανάκτηση Parcelable αντικειμένων από Intent και Bundle, στις οποίες δηλώνεται ρητά η αναμενόμενη κλάση και απορρίπτονται μη συμβατοί τύποι πριν επιστραφούν στην application logic [54].

Μια διαφορετική κατηγορία προβλημάτων εμφανίζεται όταν η serialized αναπαράσταση δεν συμφωνεί με τον τρόπο με τον οποίο διαβάζεται. Στα Parcelable αντικείμενα, η `writeToParcel()` καθορίζει τη σειρά και τη μορφή των τιμών που γράφονται, ενώ η `createFromParcel()` πρέπει να τις ανακτήσει με συμβατή σειρά και τύπο. Ασυμφωνίες μεταξύ των δύο μπορούν να οδηγήσουν σε λανθασμένη ερμηνεία των επόμενων

δεδομένων του Parcel, προκαλώντας από exceptions και denial of service μέχρι σοβαρότερες επιπτώσεις όταν η διαφορετική ερμηνεία επηρεάζει security-sensitive πεδία [54]. Η χρήση type-safe APIs μειώνει μέρος αυτής της επιφάνειας, καθώς επιτρέπει στο framework να επαληθεύει τον αναμενόμενο runtime type και να απορρίπτει ασύμβατα αντικείμενα [54], [87].

Η επαλήθευση του τύπου αποτελεί όμως μόνο το πρώτο επίπεδο ελέγχου. Ένα αντικείμενο μπορεί να είναι απολύτως έγκυρο ως instance της αναμενόμενης κλάσης και ταυτόχρονα να περιέχει attacker-controlled τιμές που επηρεάζουν επόμενες λειτουργίες, όπως file paths, URIs ή άλλα security-sensitive πεδία. Η ασφαλής επεξεργασία απαιτεί επομένως έλεγχο τόσο του αντικειμένου που ανακατασκευάζεται όσο και της εσωτερικής του κατάστασης πριν οι τιμές του χρησιμοποιηθούν σε λειτουργίες με επίπτωση στην ασφάλεια [54].

Για να αποκτήσει μια ευπάθεια deserialization ουσιαστικό security impact, η ανακατασκευή του αντικειμένου πρέπει να καταλήγει σε συμπεριφορά που επηρεάζεται από attacker-controlled δεδομένα. Μια κλάση που υπάρχει ήδη στο runtime και μπορεί να προκαλέσει τέτοια συμπεριφορά λειτουργεί ως *deserialization gadget*, καθώς τα πεδία που αποκαθίστανται κατά τη deserialization μπορούν να χρησιμοποιηθούν άμεσα από methods όπως η readObject() ή η createFromParcel(), ή να επηρεάσουν μεταγενέστερη application logic όταν το αντικείμενο χρησιμοποιηθεί από την εφαρμογή [54], [89].

Ιδιαίτερα σοβαρές περιπτώσεις εμφανίζονται όταν η serialized κατάσταση συνδέεται με native resources. Μια Serializable ή Parcelable κλάση μπορεί, για παράδειγμα, να διατηρεί αριθμητικό πεδίο που χρησιμοποιείται αργότερα ως native pointer ή handle. Αν η τιμή αυτή προέρχεται από μη έμπιστα serialized δεδομένα και περάσει χωρίς επαρκή έλεγχο σε native function, η εξωτερική είσοδος μπορεί να επηρεάσει άμεσα το native memory handling της εφαρμογής, οδηγώντας από process crash μέχρι memory corruption [89], [90].

Χαρακτηριστικό παράδειγμα παρουσιάζεται στην OVAA μέσω της MemoryCorruptionSerializable, όπου ένα πεδίο τύπου long αποτελεί μέρος της serialized κατάστασης και χρησιμοποιείται στη συνέχεια από native κώδικα ως pointer. Η δυνατότητα ελέγχου της συγκεκριμένης τιμής από τον αποστολέα μετατρέπει ένα αντικείμενο που εισέρχεται μέσω IPC σε διαδρομή προς το native επίπεδο της εφαρμογής, δείχνοντας πώς μια ευπάθεια deserialization μπορεί να αποκτήσει σημαντικά μεγαλύτερη επίπτωση από την απλή λανθασμένη ανακατασκευή ενός object [90].

Η ασφαλής επεξεργασία serialized δεδομένων απαιτεί επομένως έλεγχο της προέλευσης και του αναμενόμενου τύπου του αντικειμένου, καθώς και των τιμών που αποκαθίστανται πριν αυτές χρησιμοποιηθούν από security-sensitive ή native λειτουργίες. Οι type-safe APIs περιορίζουν μέρος της

επιφάνειας επίθεσης, ενώ η τελική ασφάλεια εξαρτάται από το αν η εσωτερική κατάσταση του αντικειμένου παραμένει έγκυρη για το context στο οποίο πρόκειται να χρησιμοποιηθεί [54].

2.4.5 Μη ασφαλής δυναμική φόρτωση κώδικα

Η δυναμική φόρτωση κώδικα επιτρέπει σε μια Android εφαρμογή να εισάγει εκτελέσιμη λογική κατά τη διάρκεια της λειτουργίας της, χρησιμοποιώντας μηχανισμούς όπως οι `DexClassLoader` και `InMemoryDexClassLoader` για τη φόρτωση classes από DEX, JAR, APK ή δεδομένα που βρίσκονται στη μνήμη [91], [91]. Από πλευράς ασφάλειας, η διαδικασία αυτή δημιουργεί ένα σαφές όριο εμπιστοσύνης, καθώς η ασφάλεια της εφαρμογής εξαρτάται πλέον και από την προέλευση και την ακεραιότητα του artifact που πρόκειται να εκτελεστεί.

Ο κώδικας που φορτώνεται δυναμικά εκτελείται μέσα στη διεργασία της εφαρμογής και αποκτά τις δυνατότητες που διαθέτει αυτή στο συγκεκριμένο περιβάλλον. Αν ένας attacker μπορεί να αντικαταστήσει ή να επηρεάσει το executable artifact πριν από τη φόρτωσή του, η εξωτερική επιρροή μετατρέπεται ουσιαστικά σε εκτέλεση κώδικα μέσα στο application context [91], [76]. Για τον λόγο αυτό, dynamically loaded code δεν πρέπει να αποθηκεύεται σε external storage ή σε άλλες writable τοποθεσίες όπου η ακεραιότητά του δεν μπορεί να εξασφαλιστεί [91], [76].

Παρόμοιο όριο εμπιστοσύνης εμφανίζεται όταν μια εφαρμογή φορτώνει κώδικα από άλλο εγκατεστημένο package. Η `createPackageContext()` μπορεί, με τη χρήση του `CONTEXT_INCLUDE_CODE`, να διαθέσει και τον κώδικα του συγκεκριμένου package στο νέο Context, ενώ η επακόλουθη χρήση του `ClassLoader` επιτρέπει την εκτέλεσή του μέσα στη διεργασία της εφαρμογής. Σε τέτοιες περιπτώσεις η ταυτότητα του package πρέπει να επαληθεύεται αξιόπιστα, ιδιαίτερα όταν χρησιμοποιούνται επιλογές όπως το `CONTEXT_IGNORE_SECURITY`, καθώς λανθασμένη εμπιστοσύνη σε εξωτερικό package μπορεί να οδηγήσει στη φόρτωση μη έμπιστου κώδικα [93], [94].

Οι νεότερες εκδόσεις του Android έχουν προσθέσει επιπλέον περιορισμούς γύρω από τη δυναμική φόρτωση. Για εφαρμογές που στοχεύουν Android 14 (API level 34) ή νεότερο, αρχεία όπως DEX, JAR και APK που πρόκειται να φορτωθούν δυναμικά πρέπει να έχουν χαρακτηριστεί read-only πριν από τη χρήση τους, μειώνοντας τη δυνατότητα μεταβολής ενός executable artifact μεταξύ της δημιουργίας και της φόρτωσής του [42].

Η ίδια βασική αρχή ισχύει και για native libraries που φορτώνονται κατά το runtime, για παράδειγμα μέσω της `System.load()`. Αν η διαδρομή ή το native artifact μπορεί να επηρεαστεί από μη έμπιστη πηγή, η εφαρμογή μπορεί να καταλήξει να εκτελεί native code μέσα στη δική της διεργασία. Η προστασία της

πλατφόρμας επεκτείνεται στις νεότερες εκδόσεις και προς αυτή την κατεύθυνση, ενώ τα ζητήματα που εμφανίζονται μετά τη φόρτωση native κώδικα εξετάζονται ξεχωριστά μέσω του JNI boundary στην επόμενη ενότητα [95].

2.4.6 Ευπάθειες στη χρήση JNI

Η μετάβαση από managed σε native κώδικα αποτελεί σημαντικό όριο ανάλυσης στις Android εφαρμογές. Μέσω του Java Native Interface (JNI), κώδικας Java ή Kotlin μπορεί να καλεί συναρτήσεις υλοποιημένες σε C ή C++, επιτρέποντας στην εφαρμογή να αξιοποιεί native βιβλιοθήκες και χαμηλότερου επιπέδου λειτουργικότητα. Από πλευράς ασφάλειας, το JNI boundary δεν αποτελεί το τέλος μιας data flow, οι τιμές που έχουν προέλθει από Intents, Binder IPC, αρχεία, URI ή το δίκτυο μπορούν να περάσουν στο native επίπεδο και να συνεχίσουν να επηρεάζουν την εκτέλεση εκεί [96], [76].

Στην περίπτωση αυτή θεωρείται ότι η native βιβλιοθήκη αποτελεί ήδη μέρος της εφαρμογής και το ενδιαφέρον επικεντρώνεται στον τρόπο με τον οποίο χρησιμοποιεί μη έμπιστα δεδομένα. Η επεξεργασία τέτοιων τιμών σε C ή C++ μπορεί να εκθέσει την εφαρμογή σε κατηγορίες σφαλμάτων που δεν εμφανίζονται με τον ίδιο τρόπο στο managed περιβάλλον, όπως buffer overflows, use-after-free και άλλες μορφές memory corruption [76]. Για τον λόγο αυτό, η ανάλυση μιας externally controlled τιμής πρέπει να συνεχίζεται και μετά τη Java ή Kotlin δήλωση της native method, μέχρι να προσδιοριστεί η πραγματική χρήση της μέσα στη βιβλιοθήκη.

Χαρακτηριστικό παράδειγμα αποτελεί η MemoryCorruptionSerializable της OVAA, το οποίο αναφέρθηκε και παραπάνω, όπου μια τιμή τύπου long αποκαθίσταται από serialized input και χρησιμοποιείται στη συνέχεια από native κώδικα ως pointer. Η δυνατότητα επηρεασμού της συγκεκριμένης τιμής από εξωτερικό caller συνδέει ένα Android IPC entry point με native memory handling και δείχνει πώς μια διαδρομή που αρχίζει στο managed επίπεδο μπορεί να καταλήξει σε memory corruption [90]. Παρόμοιες περιπτώσεις έχουν παρουσιαστεί και σε αναλύσεις πραγματικών Android εφαρμογών, όπου externally reachable functionality κατέληγε σε native state μέσω object reconstruction [90].

```

//Java κώδικας
public class NativeActivity extends Activity {

    static {
        System.loadLibrary("native-lib");
    }

    private native void nativeProcess(byte[] data);

    @Override
    protected void onCreate(Bundle state) {
        super.onCreate(state);

        // Attacker-controlled source
        byte[] data =
            getIntent().getByteArrayExtra("data");

        if (data != null) {
            nativeProcess(data); // JNI boundary
        }
    }
}

// native C/C++ κώδικας
extern "C"
JNIEXPORT void JNICALL
Java_com_example_NativeActivity_nativeProcess(
    JNIEnv* env,
    jobject thiz,
    jbyteArray input) {

    jsize len = env->GetArrayLength(input);
    jbyte* data =
        env->GetByteArrayElements(input, nullptr);

    char buffer[64];

    // Native security-sensitive operation
    memcpy(buffer, data, len);

    env->ReleaseByteArrayElements(
        input, data, JNI_ABORT);
}

```

Εικόνα 29 : Παράδειγμα μεταφοράς μη έμπιστων δεδομένων από Android Intent μέσω JNI προς native λειτουργία.

Στο παράδειγμα της Εικόνας 29, δεδομένα που προέρχονται από εισερχόμενο Intent μεταφέρονται σε native method, όπου χρησιμοποιούνται σε λειτουργία αντιγραφής προς buffer σταθερού μεγέθους. Αν το Activity είναι προσβάσιμο από μη έμπιστο caller και το μέγεθος της εισόδου δεν ελεγχθεί πριν από την αντιγραφή, η Android-level πηγή δεδομένων μπορεί να καταλήξει σε native memory-safety πρόβλημα. Η περίπτωση αυτή αποτυπώνει γιατί η αξιολόγηση μιας διαδρομής πρέπει να ακολουθεί το input πέρα από το managed/native boundary και να εξετάζει την τελική λειτουργία στην οποία χρησιμοποιείται [76].

Η μετάβαση από το managed στο native επίπεδο δεν αλλάζει την προέλευση ή τη σημασιολογία των δεδομένων που μεταφέρονται. Μια τιμή που προήλθε από μη έμπιστο Intent, αρχείο ή άλλο εξωτερικό input παραμένει attacker-controlled και μετά τη μετατροπή της σε JNI representation, επομένως οι απαραίτητοι έλεγχοι μήκους, ορίων και εγκυρότητας πρέπει να έχουν εφαρμοστεί πριν η τιμή χρησιμοποιηθεί σε native buffers, pointers ή άλλες security-sensitive δομές [96], [76].

Η αντιστοίχιση μεταξύ managed και native methods μπορεί να πραγματοποιείται είτε μέσω των συμβατικών JNI naming rules είτε μέσω δυναμικής καταχώρισης με RegisterNatives(), συχνά κατά την εκτέλεση της JNI_OnLoad() [96]. Στη δεύτερη περίπτωση, η Java ή Kotlin declaration δεν συνδέεται απαραίτητα με εμφανές exported symbol που ακολουθεί το αναμενόμενο JNI όνομα, γεγονός που

δυσκολεύει τη στατική χαρτογράφηση της διαδρομής. Η ανάλυση της native βιβλιοθήκης χρειάζεται τότε να εξετάσει τις registrations που συνδέουν τις managed method signatures με τις αντίστοιχες C ή C++ implementations.

Η πρακτική αυτή έχει ιδιαίτερη σημασία και για την αυτοματοποιημένη ανάλυση, καθώς η απουσία αναμενόμενου exported JNI symbol δεν σημαίνει ότι η σχετική native implementation απουσιάζει. Η επίσημη καθοδήγηση του Android προτείνει μάλιστα τη χρήση RegisterNatives() μέσα στην JNI_OnLoad() και τον περιορισμό των exported symbols, επομένως η ανακατασκευή του managed/native interface μπορεί να απαιτεί την εξέταση των registration tables και των function pointers που χρησιμοποιούνται κατά την αρχικοποίηση της βιβλιοθήκης [96].

Για την αξιολόγηση μιας πιθανής ευπάθειας, το κρίσιμο σημείο είναι η διαδρομή που ακολουθεί η externally controlled τιμή από το managed επίπεδο μέχρι την τελική native operation. Η JNI declaration λειτουργεί επομένως ως ενδιάμεσο σημείο της ροής, καθώς η πραγματική security-sensitive χρήση μπορεί να βρίσκεται αρκετά βαθύτερα στη native implementation. Η ανάλυση συνεχίζει μέχρι να προσδιοριστεί πώς χρησιμοποιείται τελικά η τιμή, διατηρώντας την ίδια source-to-sink λογική που εφαρμόζεται και στις υπόλοιπες κατηγορίες ευπαθειών.

2.4.7 Έκθεση ευαίσθητων δεδομένων και μυστικών

Η ασφάλεια μιας Android εφαρμογής επηρεάζεται και από τον τρόπο με τον οποίο διαχειρίζεται ευαίσθητες πληροφορίες που βρίσκονται ήδη στο εσωτερικό της. Authentication tokens, passwords, προσωπικά δεδομένα, cryptographic keys και credentials προς απομακρυσμένες υπηρεσίες μπορούν να αποτελέσουν στόχο όταν αποθηκεύονται ή καταγράφονται σε σημεία από τα οποία είναι δυνατό να ανακτηθούν αργότερα. Η σχετική επιφάνεια περιλαμβάνει runtime διαρροές, όπως η εμφάνιση ευαίσθητων τιμών στα logs, καθώς και static secrets που έχουν ενσωματωθεί στο APK, στα resources ή σε bundled configuration files [55], [97].

Χαρακτηριστική περίπτωση runtime leakage αποτελεί η χρήση των Android logging APIs για την καταγραφή security-sensitive δεδομένων. Κλήσεις όπως Log.d(), Log.i() και Log.e() χρησιμοποιούνται συχνά κατά την ανάπτυξη για την παρακολούθηση της κατάστασης της εφαρμογής ή της επικοινωνίας με απομακρυσμένες υπηρεσίες, όμως η καταγραφή passwords, session tokens ή άλλων credentials σε production builds δημιουργεί πρόσθετο κανάλι έκθεσης [55]. Οι σύγχρονες εκδόσεις του Android περιορίζουν σημαντικά την άμεση πρόσβαση τρίτων εφαρμογών στο logcat, χωρίς αυτό να καθιστά ασφαλή την αποθήκευση secrets σε logs, καθώς τα δεδομένα αυτά μπορούν να εμφανιστούν σε debugging ή diagnostic περιβάλλοντα και σε άλλους μηχανισμούς συλλογής τεχνικών πληροφοριών.

Αντίστοιχο παράδειγμα εμφανίζεται στην OVAA, όπου authentication δεδομένα καταγράφονται μέσω Log.d() κατά τη διάρκεια του login flow. Το security issue προκύπτει από τη μεταφορά μιας ευαίσθητης τιμής σε κανάλι που δεν προορίζεται για ασφαλή αποθήκευση credentials και μπορεί να διατηρήσει την πληροφορία πέρα από το σημείο στο οποίο χρησιμοποιήθηκε αρχικά. Η Android security guidance συνιστά την αποφυγή καταγραφής ευαίσθητων δεδομένων και, όταν κάποιο logging είναι απαραίτητο, την εφαρμογή κατάλληλου redaction ή masking πριν η πληροφορία αποθηκευτεί ή μεταφερθεί εκτός της κανονικής ροής της εφαρμογής [55], [134].

Μια δεύτερη σημαντική κατηγορία αφορά credentials, tokens και άλλα secrets που ενσωματώνονται στατικά στην εφαρμογή. Επειδή το APK διανέμεται τελικά στη συσκευή του χρήστη, ο κώδικας, τα resources και τα bundled assets του πρέπει να θεωρούνται προσβάσιμα σε όποιον μπορεί να αποκτήσει και να αναλύσει το πακέτο. Ένα σταθερό password ή authentication token που βρίσκεται σε Java ή Kotlin code, configuration files ή resources μπορεί επομένως να ανακτηθεί μέσω στατικής ανάλυσης και reverse engineering, γεγονός που καθιστά προβληματική τη χρήση της μυστικότητας του APK ως μηχανισμού προστασίας [97], [98].

Η αξιολόγηση ενός τέτοιου candidate χρειάζεται να λαμβάνει υπόψη τον πραγματικό ρόλο της τιμής. Ορισμένα API keys ή client identifiers προορίζονται να υπάρχουν στον client και προστατεύονται μέσω server-side restrictions, ενώ άλλα secrets παρέχουν άμεσα πρόσβαση σε backend resources ή privileged functionality. Στη δεύτερη περίπτωση, η δυνατότητα εξαγωγής και επαναχρησιμοποίησης της τιμής εκτός της εφαρμογής δημιουργεί το ουσιαστικό security impact. Η ανάλυση πρέπει επομένως να εξετάζει την υπηρεσία στην οποία χρησιμοποιείται το secret, τα δικαιώματα που παρέχει και τους πρόσθετους περιορισμούς που εφαρμόζονται από την πλευρά του server [98].

```
public class ApiClient {  
    private static final String API_TOKEN =  
        "secret_backend_token_123";  
  
    public void sendRequest() {  
        Request request = new Request.Builder()  
            .url("https://api.example.com/account")  
            .addHeader(  
                "Authorization",  
                "Bearer " + API_TOKEN  
            )  
            .build();  
  
        client.newCall(request).enqueue(callback);  
    }  
}
```

Εικόνα 30 : Παράδειγμα hardcoded authentication token που χρησιμοποιείται για την αυθεντικοποίηση αιτήματος προς απομακρυσμένη υπηρεσία.

Στο παράδειγμα αυτό, το token αποτελεί σταθερή τιμή που διανέμεται μαζί με το APK και μπορεί να εντοπιστεί κατά τη στατική ανάλυση. Η περαιτέρω αξιολόγηση επικεντρώνεται στο αν η τιμή παραμένει ενεργή, αν μπορεί να χρησιμοποιηθεί ανεξάρτητα από την εφαρμογή και ποια πρόσβαση παρέχει στην απομακρυσμένη υπηρεσία. Τα στοιχεία αυτά καθορίζουν αν το hardcoded secret αποτελεί πραγματικά εκμεταλλεύσιμη ευπάθεια και ποια είναι η αντίστοιχη επίπτωσή της [98].

Ιδιαίτερη περίπτωση αποτελούν τα hardcoded cryptographic keys, καθώς ένα ισχυρό cryptographic primitive δεν παρέχει την αναμενόμενη προστασία όταν το ίδιο το secret key διανέμεται μαζί με τα δεδομένα και τον κώδικα που το χρησιμοποιεί. Στην OVAA, για παράδειγμα, η λειτουργία WeakCrypto χρησιμοποιεί σταθερό AES key που αποτελεί μέρος της εφαρμογής, επιτρέποντας σε όποιον ανακτήσει το APK να αποκτήσει και το key material που απαιτείται για την επεξεργασία των προστατευμένων δεδομένων [99]. Για τέτοιες περιπτώσεις, το Android παρέχει μηχανισμούς όπως το Android Keystore, μέσω του οποίου cryptographic keys μπορούν να δημιουργούνται και να χρησιμοποιούνται χωρίς να αποθηκεύονται ως απλές σταθερές στον application code ή στα resources [97].

Παρόμοια προσοχή απαιτείται και για authentication material που αποθηκεύεται τοπικά κατά τη λειτουργία της εφαρμογής, όπως session ή refresh tokens. Η αποθήκευσή τους στον ιδιωτικό χώρο της εφαρμογής επωφελείται από την προστασία του Android application sandbox, όμως η διάρκεια ζωής και η μορφή στην οποία διατηρούνται επηρεάζουν την επίπτωση άλλων ευπαθειών που μπορούν να δώσουν πρόσβαση σε application files ή runtime state. Η ελαχιστοποίηση των αποθηκευμένων credentials, ο περιορισμός της διάρκειας ζωής τους και η χρήση κατάλληλων μηχανισμών προστασίας για cryptographic material μειώνουν την έκταση της πληροφορίας που μπορεί να εκτεθεί σε μια τέτοια περίπτωση [29], [76], [97].

Η τοποθεσία στην οποία αποθηκεύονται ευαίσθητα δεδομένα αποτελεί σημαντικό μέρος της συνολικής ασφάλειας της εφαρμογής. Το internal storage προστατεύεται από το Android application sandbox και περιορίζει κανονικά την πρόσβαση στα δεδομένα στο UID της εφαρμογής, ενώ το external ή shared storage έχει ιστορικά πιο permissive μοντέλο πρόσβασης. Για τον λόγο αυτό, το Android συνιστά τα ευαίσθητα δεδομένα να αποθηκεύονται κατά προτίμηση στον ιδιωτικό χώρο της εφαρμογής, ακόμη και στις νεότερες εκδόσεις όπου το Scoped Storage έχει περιορίσει σημαντικά την πρόσβαση άλλων εφαρμογών σε κοινόχρηστα αρχεία [100].

Η χρήση externally writable storage δημιουργεί και ζήτημα ακεραιότητας όταν η εφαρμογή επαναχρησιμοποιεί αργότερα τα δεδομένα που έχουν αποθηκευτεί εκεί. Αν μια εξωτερική εφαρμογή μπορεί να τροποποιήσει ένα configuration file, serialized object ή άλλο artifact πριν αυτό διαβαστεί ξανά, η αλλαγή μπορεί να επηρεάσει τη μελλοντική συμπεριφορά της εφαρμογής. Σε περιπτώσεις όπου η χρήση

external storage είναι απαραίτητη, η Android guidance συνιστά κατάλληλους μηχανισμούς ελέγχου της ακεραιότητας των δεδομένων πριν αυτά θεωρηθούν έμπιστα [100].

Ένα διαφορετικό σημείο έκθεσης προκύπτει από τους μηχανισμούς backup και μεταφοράς δεδομένων μεταξύ συσκευών. Ανάλογα με τη διαμόρφωση της εφαρμογής, αρχεία από τον ιδιωτικό της χώρο μπορούν να συμπεριληφθούν σε Auto Backup ή device-to-device migration, επομένως credentials, tokens και άλλο ευαίσθητο state χρειάζεται να αξιολογούνται και ως προς τη συμμετοχή τους σε αυτές τις διαδικασίες. Το Android επιτρέπει τον αποκλεισμό συγκεκριμένων δεδομένων μέσω backup rules όταν η μεταφορά τους δεν είναι επιθυμητή [101].

Η προστασία πρέπει να συνεχίζεται και κατά τη μεταφορά των δεδομένων εκτός της συσκευής. Cleartext επικοινωνία που περιλαμβάνει passwords, authentication tokens ή προσωπικά δεδομένα μπορεί να επιτρέψει την παρακολούθηση ή αλλοίωσή τους από actor με πρόσβαση στο δίκτυο, γι' αυτό η πλατφόρμα συνιστά τη χρήση κρυπτογραφημένων καναλιών για security-sensitive traffic [61]. Η κατηγορία αυτή αφορά την προστασία του ίδιου του καναλιού επικοινωνίας και διαφέρει από περιπτώσεις όπου ένα κανονικά αυθεντικοποιημένο request κατευθύνεται προς μη έμπιστο προορισμό.

Από πλευράς ανάλυσης, ο εντοπισμός μιας τιμής που μοιάζει με secret αποτελεί μόνο την αρχή της διερεύνησης. Η αξιολόγηση πρέπει να εξετάζει τον τρόπο με τον οποίο η τιμή χρησιμοποιείται, το σημείο στο οποίο αποθηκεύεται ή μεταδίδεται και το αν ένας μη εξουσιοδοτημένος actor μπορεί πράγματι να την αποκτήσει και να την αξιοποιήσει. Η ίδια προσέγγιση εφαρμόζεται σε hardcoded values, logs και αποθηκευμένο authentication material, ώστε να διαχωρίζονται οι πραγματικές διαρροές από δεδομένα που είναι ορατά στον client χωρίς να παρέχουν ουσιαστική security-sensitive δυνατότητα.

2.5 Σύνοψη

Οι κατηγορίες ευπαθειών που παρουσιάστηκαν βασίζονται σε διαφορετικούς μηχανισμούς του Android, όμως πολλές από αυτές συνδέονται με το ίδιο βασικό πρόβλημα: τη μεταφορά μη έμπιστης επιρροής από ένα εξωτερικό σημείο εισόδου προς λειτουργίες ή δεδομένα με μεγαλύτερη σημασία για την ασφάλεια. Η πραγματική επίπτωση κάθε περίπτωσης εξαρτάται από το context στο οποίο καταλήγει η σχετική διαδρομή και από τους ελέγχους validation και authorization που εφαρμόζονται κατά μήκος της.

Η εξέταση των παραπάνω κατηγοριών δείχνει επίσης ότι η παρουσία ενός exported component, ενός security-sensitive API ή μιας ύποπτης τιμής αποτελεί κυρίως αφετηρία για περαιτέρω διερεύνηση. Η αξιόπιστη αξιολόγηση απαιτεί τη συσχέτιση της εξωτερικής εισόδου με την τελική λειτουργία και την

εξέταση της πραγματικής επίπτωσης που προκύπτει. Η ανάγκη αυτή αποτελεί βασικό σημείο αναφοράς για τη μεθοδολογία αυτοματοποιημένης ανάλυσης που παρουσιάζεται στο επόμενο κεφάλαιο.

3. ΠΡΟΤΕΙΝΟΜΕΝΗ ΠΡΟΣΕΓΓΙΣΗ

Στο παρόν κεφάλαιο παρουσιάζεται η προτεινόμενη προσέγγιση για την αυτοματοποίηση της ανάλυσης ασφάλειας εφαρμογών Android και οι σχεδιαστικές επιλογές που διαμόρφωσαν την υλοποίησή της. Αρχικά καθορίζονται οι στόχοι της μεθοδολογίας και ο τρόπος με τον οποίο η στατική και η δυναμική ανάλυση εντάσσονται σε μια κοινή διαδικασία διερεύνησης. Στη συνέχεια παρουσιάζονται τα κριτήρια που οδήγησαν στην επιλογή του γλωσσικού μοντέλου και των εργαλείων που ενσωματώθηκαν στο σύστημα, καθώς και ο τρόπος με τον οποίο συντονίζονται στο πλαίσιο της *agentic* διαδικασίας. Το κεφάλαιο ολοκληρώνεται με την παρουσίαση της συνολικής αρχιτεκτονικής και της ροής της ανάλυσης, από τη διαμόρφωση μιας αρχικής υπόθεσης έως τη συλλογή των απαραίτητων τεκμηρίων και την τελική αξιολόγηση του ευρήματος.

3.1 Στόχοι και σχεδιαστικές αρχές

Βασικός στόχος της παρούσας εργασίας είναι να εξεταστεί σε ποιο βαθμό μπορεί να αυτοματοποιηθεί η διαδικασία εντοπισμού και αξιολόγησης ευπαθειών σε εφαρμογές Android με τη χρήση Μεγάλων Γλωσσικών Μοντέλων. Όπως αναφέρθηκε στο πρώτο κεφάλαιο, η ανάλυση μιας πραγματικής εφαρμογής απαιτεί συνήθως περισσότερα από την αναγνώριση ενός επικίνδυνου API ή ενός γνωστού μοτίβου κώδικα, καθώς η σημασία ενός ευρήματος εξαρτάται από το πλαίσιο στο οποίο εμφανίζεται, την εξωτερική προσβασιμότητα της σχετικής λειτουργίας και την προέλευση των δεδομένων που φτάνουν σε αυτή.

Οι παραδοσιακές τεχνικές αυτοματοποιημένης ανάλυσης αντιμετωπίζουν μέρος του προβλήματος, χωρίς να καλύπτουν πάντοτε το σύνολο της απαιτούμενης πληροφορίας. Οι κανόνες στατικής ανάλυσης μπορούν να εντοπίσουν ύποπτες χρήσεις APIs ή γνωστά μοτίβα, ενώ τεχνικές *data-flow* και *taint analysis* εξετάζουν τη διαδρομή που ακολουθούν τα δεδομένα μέσα στο πρόγραμμα. Εργαλεία όπως το FlowDroid και το CodeQL επιτρέπουν έτσι τη συσχέτιση *sources* και *sinks* σε μεγαλύτερο τμήμα της εφαρμογής, με την ακρίβεια της ανάλυσης να επηρεάζεται από τη μοντελοποίηση του προγράμματος και των ιδιαίτερων μηχανισμών του Android [10], [12], [16].

Η δυναμική ανάλυση παρέχει συμπληρωματικές πληροφορίες μέσα από την παρατήρηση της πραγματικής συμπεριφοράς της εφαρμογής κατά την εκτέλεση [18], [19]. Η δυνατότητα αυτή εξαρτάται από την ενεργοποίηση της διαδρομής που σχετίζεται με το υπό εξέταση πρόβλημα, η οποία μπορεί να προϋποθέτει συγκεκριμένη κατάσταση της εφαρμογής ή προηγούμενη ακολουθία ενεργειών. Η δυσκολία προσέγγισης τέτοιων καταστάσεων έχει παρατηρηθεί και σε εργαλεία αυτοματοποιημένου GUI exploration, όπου μέρος

της συμπεριφοράς της εφαρμογής μπορεί να παραμένει αδιερεύνητο ακόμη και μετά από εκτεταμένη εξερεύνηση [13].

Η προτεινόμενη προσέγγιση οργανώνει τις δύο μορφές ανάλυσης σε μια κοινή διαδικασία διερεύνησης. Η στατική ανάλυση χρησιμοποιείται για τη διαμόρφωση αρχικών υποθέσεων και τον εντοπισμό διαδρομών που παρουσιάζουν ενδιαφέρον, ενώ η δυναμική εξέταση κατευθύνεται στα αντίστοιχα σημεία για τη συλλογή πρόσθετων στοιχείων και την παρατήρηση της συμπεριφοράς τους κατά την εκτέλεση. Η δυναμική ανάλυση λειτουργεί επομένως ως στοχευμένο στάδιο ελέγχου, το οποίο καθορίζεται από τα ευρήματα που έχουν προηγηθεί και όχι ως γενική εξερεύνηση του state space της εφαρμογής.

Κατά την εξέλιξη της διερεύνησης, οι πληροφορίες που συλλέγονται διατηρούνται στο ίδιο πλαίσιο ώστε ένα αρχικό candidate να μπορεί να εξετάζεται ως μέρος μιας συνολικής διαδρομής. Η αξιολόγηση ακολουθεί την εξωτερικά ελεγχόμενη είσοδο μέσα στη ροή της εφαρμογής και εξετάζει τη συμπεριφορά που προκύπτει όταν η σχετική διαδρομή ενεργοποιείται στην πράξη. Με αυτόν τον τρόπο, η απόφαση για την επιβεβαίωση ή την απόρριψη ενός candidate βασίζεται στα τεκμήρια που συγκεντρώνονται κατά τη διάρκεια της ανάλυσης.

Η ίδια οργάνωση επιδιώκει να περιορίσει και τη χειροκίνητη εργασία που απαιτείται για την αξιολόγηση μεγάλου αριθμού αρχικών ευρημάτων. Η αυτοματοποιημένη διαδικασία προχωρά τη διερεύνηση όσο επιτρέπουν τα διαθέσιμα στοιχεία, ώστε η παρέμβαση του αναλυτή να επικεντρώνεται σε μικρότερο αριθμό περιπτώσεων για τις οποίες έχει ήδη συγκεντρωθεί ουσιαστική τεκμηρίωση.

Παράλληλα, η διαδικασία πρέπει να παραμένει ελέγξιμη και αναπαραγώγιμη. Επομένως κάθε τελικό εύρημα χρειάζεται να συνδέεται με τα αντίστοιχα σημεία του κώδικα, τις ενέργειες που πραγματοποιήθηκαν κατά τη διερεύνηση και τα αποτελέσματα που παρατηρήθηκαν κατά την εκτέλεση. Η απαίτηση αυτή έχει ιδιαίτερη σημασία σε ένα LLM-based σύστημα, καθώς οι εκτιμήσεις του μοντέλου μπορεί να επηρεάζονται από ελλιπές context ή να περιλαμβάνουν μη επαρκώς τεκμηριωμένες υποθέσεις [7], [8].

Η διαχείριση του context αποτελεί επίσης βασική σχεδιαστική απαίτηση, ιδιαίτερα σε εφαρμογές μεγάλου μεγέθους. Η εισαγωγή ολόκληρης της βάσης κώδικα μπορεί να προσθέσει μεγάλο όγκο άσχετης πληροφορίας, ενώ ένα υπερβολικά περιορισμένο context μπορεί να αποκρύψει διαδρομές που εκτείνονται σε διαφορετικές συναρτήσεις ή components. Η επιλογή των πληροφοριών που διατηρούνται κατά την ανάλυση επηρεάζει επομένως την ικανότητα του μοντέλου να συσχετίσει τα σχετικά τμήματα της εφαρμογής [118]. Σε μια πολυσταδιακή διαδικασία χρειάζεται επίσης να διατηρείται η κατάσταση της

διερεύνησης, ώστε τα νέα στοιχεία να ενσωματώνονται στις ήδη διαμορφωμένες υποθέσεις και να επηρεάζουν τα επόμενα βήματα [116].

Στην παρούσα εργασία, το LLM εντάσσεται επομένως στη διαδικασία ως μηχανισμός συσχέτισης των διαθέσιμων πληροφοριών και καθοδήγησης των επόμενων ενεργειών. Η αξιολόγηση ενός ευρήματος εξακολουθεί να βασίζεται σε στοιχεία που προκύπτουν από τον κώδικα και την πραγματική εκτέλεση της εφαρμογής, ενώ το μοντέλο χρησιμοποιείται για να οργανώνει και να αξιοποιεί τις πληροφορίες που παράγονται κατά τα διαφορετικά στάδια της ανάλυσης. Οι παραπάνω απαιτήσεις αποτέλεσαν τη βάση για την επιλογή του γλωσσικού μοντέλου, των εργαλείων ανάλυσης και της συνολικής αρχιτεκτονικής που παρουσιάζονται στις επόμενες ενότητες.

3.2 Επιλογή μοντέλου

Σε αρχικό στάδιο της εργασίας εξετάστηκε η ανάπτυξη ενός εξειδικευμένου μοντέλου μηχανικής μάθησης για την αναγνώριση ευπαθειών σε εφαρμογές Android. Η προσέγγιση βασιζόταν στην εκπαίδευση ενός classifier πάνω σε επισημασμένα παραδείγματα ευάλωτου και μη ευάλωτου κώδικα, με σκοπό την εφαρμογή του σε νέες περιπτώσεις. Η κατεύθυνση αυτή απαιτούσε όμως τη δημιουργία ενός αρκετά μεγάλου και αξιόπιστου dataset, με επαρκή κάλυψη διαφορετικών κατηγοριών ευπαθειών και κατάλληλα αρνητικά παραδείγματα. Η ποιότητα των δεδομένων θα επηρέαζε επομένως άμεσα την αξιοπιστία του μοντέλου, ενώ η συλλογή και η επαλήθευσή τους αποτελούσαν από μόνες τους σημαντικό μέρος της απαιτούμενης ερευνητικής εργασίας.

Η έκταση αυτής της απαίτησης γίνεται περισσότερο εμφανής σε αντίστοιχες προσεγγίσεις της βιβλιογραφίας με χαρακτηριστικό παράδειγμα το Devign, το οποίο αντιμετωπίζει το vulnerability detection ως πρόβλημα binary classification σε επίπεδο function και χρησιμοποιεί γραφικές αναπαραστάσεις του κώδικα που περιλαμβάνουν πληροφορίες από AST, control flow, data flow και τη σειρά των tokens. Η αναπαράσταση αυτή επεξεργάζεται από Graph Neural Network, το οποίο προβλέπει αν η εξεταζόμενη function ανήκει στην κατηγορία vulnerable ή non-vulnerable [120].

Η ανάπτυξη και αξιολόγηση του Devign είναι παράλληλα ενδεικτική των πόρων που απαιτούνται για τη δημιουργία ενός τέτοιου συστήματος. Οι συγγραφείς κατασκεύασαν χειροκίνητα επισημασμένα datasets από τα Linux Kernel, QEMU, Wireshark και FFmpeg, με τη διαδικασία επισημάνσης και διασταύρωσης των δεδομένων να απαιτεί περίπου 600 ανθρώπινες ώρες από ομάδα τεσσάρων επαγγελματιών security researchers. Το τελικό σύνολο περιλάμβανε 58.965 functions, από τις οποίες 27.652 χαρακτηρίστηκαν vulnerable και 31.313 non-vulnerable. Το μοντέλο παρουσίασε κατά μέσο όρο βελτίωση 10,51% στην

accuracy και 8,68% στο F1 score σε σχέση με τις baseline learning-based approaches, ενώ σε ξεχωριστή αξιολόγηση 40 νεότερων CVEs των ίδιων projects πέτυχε accuracy 74,11% [120].

Η προσέγγιση του Devign δείχνει ότι η εφαρμογή machine learning στη vulnerability detection είναι εφικτή, παρουσιάζει όμως ορισμένες διαφορές σε σχέση με το πρόβλημα που εξετάζεται στην παρούσα εργασία. Το dataset της μελέτης προέρχεται από έργα γραμμένα σε C και η ταξινόμηση πραγματοποιείται σε επίπεδο μεμονωμένης function [120]. Ο περιορισμός αυτός δεν αφορά αποκλειστικά τις εφαρμογές Android, καθώς πολλές ευπάθειες απαιτούν τη συσχέτιση κώδικα που βρίσκεται σε διαφορετικές functions. Στο Android, όμως, η ανάλυση μπορεί να χρειάζεται επιπλέον να ακολουθήσει ροές που διασχίζουν application components, framework callbacks και μηχανισμούς επικοινωνίας της πλατφόρμας, γεγονός που καθιστά μια αποκλειστικά function-level αναπαράσταση λιγότερο κατάλληλη για το αντικείμενο της παρούσας εργασίας.

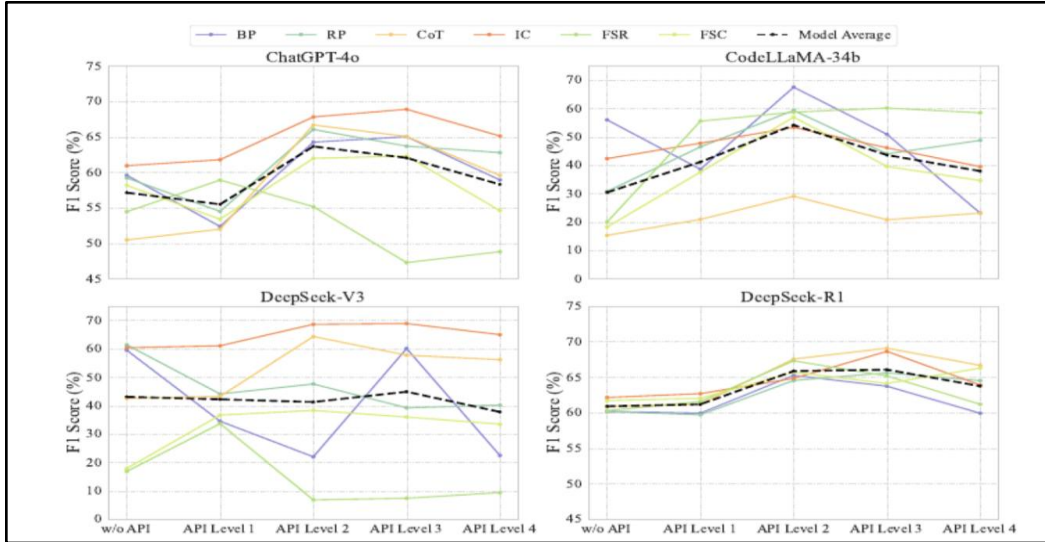
Ο περιορισμός της function-level ανάλυσης έχει εξεταστεί ειδικότερα από τους Li et al. [121], οι οποίοι μελέτησαν inter-procedural vulnerabilities όπου τα statements που σχετίζονται με την ευπάθεια και εκείνα που οδηγούν στην εκδήλωσή της βρίσκονται σε διαφορετικές functions. Στο dataset της μελέτης, οι συγκεκριμένες ευπάθειες εκτείνονταν κατά μέσο όρο σε 2,8 inter-procedural επίπεδα και οι function-level detectors παρουσίασαν χαμηλότερη αποτελεσματικότητα σε σχέση με τις αντίστοιχες intra-procedural περιπτώσεις [121]. Το ζήτημα αυτό αποκτά ιδιαίτερη σημασία στο Android, όπου μια security-relevant ροή μπορεί να επηρεάζεται από framework callbacks, application components και διαφορετικά επίπεδα υλοποίησης.

Οι παραπάνω παράγοντες οδήγησαν στην εγκατάλειψη της ανάπτυξης ενός νέου εξειδικευμένου classifier και στην αξιοποίηση ενός ήδη προεκπαιδευμένου Large Language Model. Η επιλογή αυτή αποφεύγει την ανάγκη δημιουργίας ενός Android-specific dataset για την εκπαίδευση ενός νέου μοντέλου και επιτρέπει την εξέταση του κώδικα μαζί με τις πληροφορίες που σχετίζονται με την εκάστοτε διερεύνηση.

Η επιλογή ενός LLM δεν επιλύει από μόνη της το πρόβλημα του context. Η εξέταση μόνο της function στην οποία εμφανίζεται ένα ύποπτο σημείο μπορεί να αποκρύψει πληροφορίες που βρίσκονται σε callers, callbacks ή άλλα components, ενώ η εισαγωγή ολόκληρης της βάσης κώδικα αυξάνει σημαντικά τον όγκο μη σχετικής πληροφορίας. Οι Yang et al. εξετάζουν το ζήτημα αυτό μέσω του PacVD, μιας προσέγγισης που συνδυάζει program analysis και LLMs ώστε να επιλέγεται context σχετικό με control flow, data flow και σχέσεις μεταξύ functions πριν από την αξιολόγηση του κώδικα από το μοντέλο [118].

Τα αποτελέσματα της μελέτης έδειξαν ότι η επιλογή του κατάλληλου context επηρεάζει μετρήσιμα την αποτελεσματικότητα της vulnerability detection. Στην καλύτερη configuration που εξετάστηκε, με

DeepSeek-R1 και Chain-of-Thought prompting, αναφέρθηκε βελτίωση έως 12,77% στην accuracy, 10,05% στην precision και 9,25% στο F1 score σε σχέση με τις baseline approaches. Παράλληλα, το καταλληλότερο επίπεδο abstraction διαφοροποιήθηκε μεταξύ των μοντέλων, δείχνοντας ότι η ποσότητα και η μορφή του context δεν μπορούν να αντιμετωπίζονται ως σταθερές παράμετροι ανεξάρτητα από το χρησιμοποιούμενο LLM [118].



Εικόνα 31: Επίδραση διαφορετικών επιπέδων contextual abstraction και prompting strategies στο F1 score διαφορετικών LLMs για vulnerability detection. **Πηγή:** Yang et al., *Context-Enhanced Vulnerability Detection Based on Large Language Model* [118].

Η παροχή κατάλληλου context βελτιώνει την ικανότητα ενός LLM να αξιολογήσει ένα συγκεκριμένο τμήμα κώδικα, δεν καλύπτει όμως από μόνη της τον τρόπο με τον οποίο εξελίσσεται μια πραγματική διαδικασία διερεύνησης. Η ανάλυση μιας πιθανής ευπάθειας μπορεί να απαιτήσει την αναζήτηση πρόσθετου κώδικα, την εξέταση διαφορετικού component ή την εκτέλεση ενός ελέγχου που θα προσφέρει νέα στοιχεία για την αρχική υπόθεση. Η πληροφορία που προκύπτει σε κάθε στάδιο επηρεάζει έτσι την κατεύθυνση της επόμενης ενέργειας, γεγονός που διαφοροποιεί τη διαδικασία από μια μεμονωμένη αλληλεπίδραση τύπου prompt-response.

Για τον λόγο αυτό, η χρήση του γλωσσικού μοντέλου εξετάστηκε στο πλαίσιο ενός agentic συστήματος, στο οποίο το μοντέλο συμμετέχει ενεργά στην εξέλιξη της ανάλυσης και μπορεί να αλληλεπιδρά με εξωτερικά εργαλεία. Η σχετική βιβλιογραφία για LLM-based agents στη μηχανική λογισμικού περιγράφει ως βασικά στοιχεία τέτοιων συστημάτων τον σχεδιασμό των επόμενων ενεργειών, τη διατήρηση της κατάστασης της εργασίας και την αξιοποίηση εργαλείων για την εκτέλεση επιμέρους βημάτων [116]. Η

μορφή αυτή ταιριάζει περισσότερο στη φύση του vulnerability research, όπου η επόμενη ενέργεια δεν είναι γνωστή εξ αρχής αλλά καθορίζεται από τα αποτελέσματα που έχουν ήδη συγκεντρωθεί.

Η χρησιμότητα μιας τέτοιας οργάνωσης έχει εξεταστεί και σε security tasks. Οι Zhu et al. πρότειναν το HPTSA (Hierarchical Planning and Task-Specific Agents), στο οποίο ένας planning agent συντονίζει εξειδικευμένους agents κατά την εκτέλεση διαφορετικών σταδίων μιας διαδικασίας exploitation. Σε benchmark 14 πραγματικών vulnerabilities, η προσέγγιση με GPT-4 πέτυχε 42% pass@5 και 18% pass@1, παρουσιάζοντας υψηλότερη επίδοση από έναν single-agent configuration του ίδιου μοντέλου [122]. Τα ablation experiments της μελέτης έδειξαν επίσης αισθητή μείωση της επίδοσης όταν αφαιρούνταν οι task-specific agents ή η hierarchical οργάνωση, παρέχοντας ένδειξη ότι ο τρόπος με τον οποίο διαχωρίζεται και συντονίζεται μια σύνθετη εργασία επηρεάζει ουσιαστικά το τελικό αποτέλεσμα [122].

Παρότι το HPTSA εξετάζει exploitation ευπαθειών σε web εφαρμογές και όχι Android vulnerability analysis, παρουσιάζει μια δομή που είναι σχετική με το πρόβλημα της παρούσας εργασίας: ένα σύνθετο security task αναλύεται σε διαδοχικά βήματα και η συνέχεια της διαδικασίας καθορίζεται από τα αποτελέσματα που προκύπτουν κατά την εκτέλεσή τους. Αντίστοιχα, η συστηματική μελέτη των Liu et al. για LLM-based agents στη μηχανική λογισμικού, η οποία εξετάζει 124 σχετικές εργασίες, αναγνωρίζει την αλληλεπίδραση με εξωτερικά εργαλεία και πόρους ως βασικό χαρακτηριστικό των σύγχρονων agentic προσεγγίσεων [123].

Η παρούσα εργασία δεν επιχειρεί να συγκρίνει διαφορετικά LLMs ή να απομονώσει την επίδοση μιας συγκεκριμένης έκδοσης μοντέλου, αλλά να αξιολογήσει τη μεθοδολογία με την οποία ένα σύγχρονο γλωσσικό μοντέλο μπορεί να χρησιμοποιηθεί για τον συντονισμό μιας διαδικασίας έρευνας ευπαθειών. Η επιλογή αυτή εισάγει έναν περιορισμό ως προς την πλήρη αναπαραγωγικότητα των αποτελεσμάτων, καθώς ένα εμπορικό LLM-based περιβάλλον μπορεί να μεταβάλλεται ως προς το μοντέλο και τη συμπεριφορά του. Για τον λόγο αυτό, τα αποτελέσματα της εργασίας αξιολογούνται κυρίως ως προς τη λειτουργία της προτεινόμενης μεθοδολογίας και της συνολικής αρχιτεκτονικής και όχι ως μέτρηση της επίδοσης μιας σταθερής έκδοσης γλωσσικού μοντέλου.

3.3 Επιλογή και ενοποίηση εργαλείων

Η ανάλυση μιας πραγματικής εφαρμογής Android απαιτεί την επεξεργασία πληροφοριών που προέρχονται από διαφορετικά μέρη της εφαρμογής και γίνονται διαθέσιμες με διαφορετικούς τρόπους. Ο κώδικας μπορεί να περιλαμβάνει Dalvik bytecode, native βιβλιοθήκες ή JavaScript/Hermes bundles, ενώ σημαντικό μέρος της συμπεριφοράς της εφαρμογής μπορεί να παρατηρηθεί μόνο κατά την εκτέλεσή της. Η

προτεινόμενη προσέγγιση βασίζεται επομένως στη χρήση διαφορετικών εργαλείων στατικής και δυναμικής ανάλυσης, ανάλογα με το είδος της πληροφορίας που χρειάζεται σε κάθε στάδιο της διερεύνησης.

Κατά την επιλογή των εργαλείων δόθηκε έμφαση στη δυνατότητά τους να καλύπτουν τις τεχνολογίες που συναντώνται στις εφαρμογές Android και να παρέχουν πληροφορίες που μπορούν να αξιοποιηθούν στα επόμενα στάδια της ανάλυσης. Παράλληλα, τα εργαλεία έπρεπε να μπορούν να χρησιμοποιηθούν προγραμματιστικά ή μέσω κατάλληλων διεπαφών, ώστε η εκτέλεσή τους και η αξιοποίηση των αποτελεσμάτων τους να εντάσσονται στην ίδια LLM-based διαδικασία.

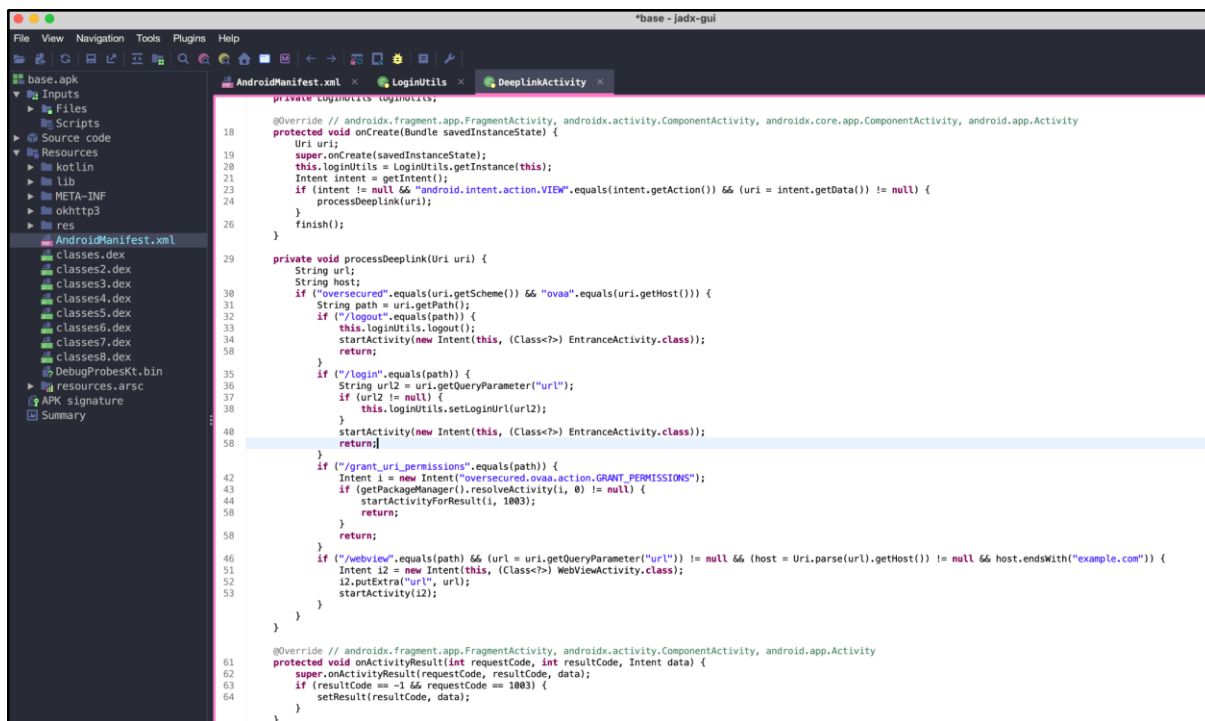
3.3.1 Εργαλεία στατικής ανάλυσης

Η στατική ανάλυση αποτελεί το αρχικό στάδιο της διαδικασίας αξιολόγησης και πραγματοποιείται χωρίς να απαιτείται η εκτέλεση της εφαρμογής. Στόχος της είναι να αποτυπώσει τη δομή του APK και να εντοπίσει τα σημεία από τα οποία μπορεί να εισέλθει εξωτερικά ελεγχόμενη πληροφορία, ώστε να διερευνηθούν πιθανές διαδρομές προς security-sensitive λειτουργίες. Η μορφή του κώδικα που συναντάται σε κάθε εφαρμογή καθορίζει και τα εργαλεία που απαιτούνται για την ανάκτησή του και την περαιτέρω ανάλυση. Κατά την επιλογή τους δόθηκε έμφαση στην ποιότητα της πληροφορίας που μπορούν να παρέχουν για τη δομή και τις σχέσεις του κώδικα, καθώς και στη δυνατότητα αξιοποίησής τους μέσα στην αυτοματοποιημένη διαδικασία που περιγράφεται στις επόμενες ενότητες.

JADX: Το JADX αποτελεί το κύριο εργαλείο στατικής ανάλυσης Android εφαρμογών που χρησιμοποιείται στην παρούσα εργασία. Πρόκειται για decompiler Dalvik bytecode, ο οποίος μπορεί να επεξεργαστεί αρχεία APK και DEX και να ανακατασκευάσει Java-like source code. Παράλληλα αποκωδικοποιεί το AndroidManifest.xml και τα resources της εφαρμογής, ενώ παρέχει δυνατότητες πλοήγησης και αναζήτησης στον κώδικα, όπως full-text search, μετάβαση σε declarations, αναζήτηση usages και cross-references [102]. Η χρησιμότητά του ενισχύεται από τη δυνατότητα προγραμματιστικής πρόσβασης στις λειτουργίες του, γεγονός που επιτρέπει την ενσωμάτωσή του στην αυτοματοποιημένη διαδικασία της παρούσας εργασίας. Μέσω του JADX μπορούν να εξεταστούν οι σχέσεις μεταξύ methods και classes, να εντοπιστούν references σε APIs και strings και να συσχετιστούν στοιχεία του Manifest με τις αντίστοιχες implementations στον decompiled κώδικα.

Το obfuscation, το reflection ή το dynamically loaded code είναι παράγοντες που μπορούν να περιορίσουν την πληροφορία που είναι διαθέσιμη μέσω της στατικής ανάλυσης. Για τον λόγο αυτό, το JADX χρησιμοποιείται ως κύρια πηγή στατικής πληροφορίας, χωρίς τα αποτελέσματά του να θεωρούνται από μόνα τους επαρκή για την επιβεβαίωση μιας ευπάθειας [17], [139].

Υπάρχουν και άλλα εργαλεία ανάλυσης APK, όπως το Arktool και το MobSF. Το Arktool είναι ιδιαίτερα χρήσιμο για resource decoding και Smali-level examination [130], ενώ το MobSF παρέχει μεγαλύτερο βαθμό αυτοματοποιημένου scanning [21]. Στην παρούσα εργασία προτιμήθηκε όμως το JADX επειδή προσφέρει πιο άμεση πρόσβαση στη δομή και στις σχέσεις του managed code, οι οποίες είναι απαραίτητες για την περαιτέρω διερεύνηση των ευρημάτων.



Εικόνα 32: Παράδειγμα στατικής ανάλυσης Deep Link μέσω JADX.

Hermes-dec: Το hermes-dec χρησιμοποιείται συμπληρωματικά σε εφαρμογές React Native όπου μέρος της application logic έχει μεταγλωττιστεί σε Hermes bytecode. Το Hermes είναι JavaScript engine σχεδιασμένο για React Native και υποστηρίζει ahead-of-time compilation του JavaScript source σε bytecode, ενώ από το React Native 0.70 αποτελεί το προεπιλεγμένο JavaScript engine για νέα React Native projects [103].

Σημαντικό μέρος της λογικής ορισμένων εφαρμογών μπορεί να βρίσκεται εκτός των Java ή Kotlin classes που ανακατασκευάζονται από το JADX. Το hermes-dec, το οποίο αναπτύσσεται από την P1 Security, παρέχει δυνατότητες parsing, disassembly και decompilation του Hermes bytecode [104], επιτρέποντας την εξέταση της JavaScript-side λογικής σε υψηλότερο επίπεδο από μια απλή bytecode representation. Η έξοδος μπορεί να χρησιμοποιηθεί για την αναζήτηση routes, URLs, API endpoints, strings και άλλων στοιχείων που είναι χρήσιμα κατά τη στατική ανάλυση.

Η ανακατασκευασμένη έξοδος δεν αποτελεί απαραίτητα έγκυρο JavaScript source και ενδέχεται να μην επαναφέρει πλήρως τις αρχικές δομές του προγράμματος. Παρ' όλα αυτά, η δυνατότητα παραγωγής υψηλότερου επιπέδου pseudo-code αποτέλεσε βασικό λόγο επιλογής του έναντι εργαλείων που περιορίζονται κυρίως σε disassembly του Hermes bytecode [104].

Το hermes-dec χρησιμοποιείται συμπληρωματικά προς το JADX και καλύπτει το τμήμα της εφαρμογής που βρίσκεται στο React Native/Hermes επίπεδο, ενώ το JADX παραμένει το κύριο εργαλείο για το Android-specific managed code και τη δομή του APK.

IDA Pro / Ghidra: Η παρουσία native βιβλιοθηκών σε μια εφαρμογή Android απαιτεί διαφορετική προσέγγιση από εκείνη που χρησιμοποιείται για τον managed κώδικα, καθώς τα αρχεία .so περιέχουν μεταγλωττισμένο machine code και δεν μπορούν να αναλυθούν μέσω Java decompilers όπως το JADX. Για την εξέταση αυτού του τμήματος της εφαρμογής χρησιμοποιούνται εργαλεία reverse engineering όπως το IDA Pro και το Ghidra, τα οποία παρέχουν disassembly και decompilation του native binary σε αναπαράσταση υψηλότερου επιπέδου. Μέσα από τις πληροφορίες που ανακτώνται για functions, symbols, imports και cross-references είναι δυνατή η σταδιακή κατανόηση της δομής μιας βιβλιοθήκης και των σχέσεων μεταξύ των επιμέρους τμημάτων του κώδικα [105], [106], [107].

Στην τελική υλοποίηση, ωστόσο, επιλέχθηκε το Ghidra ως κύριο περιβάλλον native analysis. Το Ghidra παρέχει πλήρες σύνολο δυνατοτήτων για disassembly και decompilation, καθώς και εργαλεία όπως το Function Graph, το Function Call Tree και τα cross-references, τα οποία επιτρέπουν την εξέταση της δομής του binary και των σχέσεων μεταξύ των επιμέρους functions [107]. Η ελεύθερη διάθεσή του και η δυνατότητα χρήσης του ως ολοκληρωμένου reverse-engineering environment το κατέστησαν κατάλληλο για τις ανάγκες της παρούσας εργασίας.

Η σύνδεση του managed κώδικα με τις native βιβλιοθήκες πραγματοποιείται μέσω του Java Native Interface (JNI), επιτρέποντας τη συσχέτιση declarations που εντοπίζονται στο Java ή Kotlin επίπεδο με τις αντίστοιχες native implementations. Η αντιστοίχιση μπορεί να προκύπτει από exported JNI symbols ή από dynamic registration μέσω RegisterNatives(), συχνά κατά την εκτέλεση της JNI_OnLoad() [96]. Η πληροφορία αυτή επιτρέπει στην ανάλυση να συνεχίζεται από το managed επίπεδο προς τον native κώδικα όταν μέρος της λειτουργικότητας μιας εφαρμογής υλοποιείται σε βιβλιοθήκες .so.

Στην παρούσα εργασία το ADB χρησιμοποιείται κυρίως για την ελεγχόμενη ενεργοποίηση σημείων εισόδου που έχουν προηγουμένως εντοπιστεί κατά τη στατική ανάλυση. Μέσω του Activity Manager μπορούν να αποσταλούν Intents με συγκεκριμένα actions, URIs και extras, επιτρέποντας τη μετατροπή μιας πιθανής διαδρομής που έχει εντοπιστεί στον κώδικα σε επαναλήψιμο runtime test [108].

Σε περιπτώσεις όπου μια λειτουργία είναι προσβάσιμη μόνο μέσα από το UI, το ADB μπορεί να χρησιμοποιηθεί σε συνδυασμό με εργαλεία του Android όπως το uiautomator και τα input commands, ώστε να αναπαραχθούν οι απαραίτητες αλληλεπιδράσεις με την εφαρμογή [108], [109]. Με τη δυνατότητα αυτή συμπληρώνεται η άμεση ενεργοποίηση μέσω Intents και επιτρέπεται η προσέγγιση διαδρομών που εξαρτώνται από κάποια προηγούμενη κατάσταση ή ενέργεια του χρήστη. Στην τελική υλοποίηση χρησιμοποιήθηκε κυρίως σε πραγματική συσκευή, αλλά η δυναμική ανάλυση μπορεί να πραγματοποιηθεί αντίστοιχα και σε κάποιο emulator.

```
malamateria@Mac Downloads % adb shell
redfin:/ $ am start -W -a android.intent.action.VIEW -d oversecured://ovaa/webview?url=http://evilexample.com -f 0x8000
Starting: Intent { act=android.intent.action.VIEW dat=oversecured://ovaa/... flg=0x8000 }
Status: ok
LaunchState: WARM
Activity: oversecured.ovaa/.activities.WebViewActivity
TotalTime: 146
WaitTime: 148
Complete
redfin:/ $
```

Εικόνα 34: Ενεργοποίηση Deep Link της εφαρμογής OVAA μέσω του Android Activity Manager.

Frida: Το Frida αποτελεί framework δυναμικού instrumentation και χρησιμοποιείται για την παρατήρηση της συμπεριφοράς μιας εφαρμογής κατά την εκτέλεσή της. Στο Android μπορεί να συνδεθεί με μια ήδη εκτελούμενη διεργασία ή να εκκινήσει την εφαρμογή υπό instrumentation, επιτρέποντας την παρακολούθηση methods του Android Runtime και native functions που βρίσκονται σε φορτωμένες βιβλιοθήκες [110]. Η δυνατότητα αυτή είναι ιδιαίτερα χρήσιμη όταν ένα εύρημα της στατικής ανάλυσης πρέπει να εξεταστεί με τις πραγματικές τιμές και τις συνθήκες που εμφανίζονται κατά την εκτέλεση.

Η λειτουργία του βασίζεται στον instrumentation core Gum και στο GumJS, μέσω του οποίου JavaScript scripts εκτελούνται μέσα στη διεργασία-στόχο και αλληλεπιδρούν με τις APIs του Frida [111]. Στο managed επίπεδο, οι διαθέσιμες Java APIs επιτρέπουν την πρόσβαση σε φορτωμένες classes και την προσωρινή παρεμβολή στην εκτέλεση συγκεκριμένων methods, ώστε να παρατηρούνται οι παράμετροι και οι επιστρεφόμενες τιμές τους. Η ίδια λογική εφαρμόζεται στον native κώδικα μέσω μηχανισμών όπως το Interceptor, ο οποίος επιτρέπει την τοποθέτηση hooks σε functions που έχουν φορτωθεί στη διεργασία και την εξέταση της συμπεριφοράς τους κατά την είσοδο και την έξοδο από αυτές [111].

Στην παρούσα εργασία το Frida χρησιμοποιείται κυρίως για την επαλήθευση διαδρομών που έχουν προηγουμένως εντοπιστεί κατά τη στατική ανάλυση. Για παράδειγμα, όταν μια τιμή φαίνεται να προέρχεται από εξωτερικά ελεγχόμενη είσοδο και να καταλήγει σε μια security-sensitive λειτουργία, το instrumentation μπορεί να χρησιμοποιηθεί για να παρατηρηθεί αν η ίδια τιμή εμφανίζεται πράγματι στα αντίστοιχα σημεία κατά την εκτέλεση. Η δυνατότητα παρακολούθησης managed και native κώδικα επιτρέπει επίσης τη συνέχεια της διερεύνησης σε περιπτώσεις όπου μια ροή περνά από το Java ή Kotlin επίπεδο προς μια native βιβλιοθήκη μέσω JNI.

Το Frida παρέχει και δυνατότητες προσωρινής τροποποίησης της συμπεριφοράς μιας function, όπως αλλαγή ενός argument ή μιας return value. Στην παρούσα εργασία οι δυνατότητες αυτές χρησιμοποιούνται μόνο όταν είναι απαραίτητες για την κατανόηση μιας διαδρομής, καθώς ο κύριος ρόλος του instrumentation είναι η παρατήρηση και η επιβεβαίωση της πραγματικής συμπεριφοράς της εφαρμογής και όχι η δημιουργία συνθηκών που δεν θα εμφανιζόταν κανονικά κατά την εκτέλεση [150].

Medusa: Στην παρούσα εργασία η κύρια αξιοποίηση του Frida πραγματοποιείται μέσω του Medusa, ενός modular framework δυναμικής ανάλυσης για Android που βασίζεται στο Frida. Το Medusa οργανώνει το instrumentation σε επαναχρησιμοποιήσιμα modules, τα οποία μπορούν να συνδυαστούν ανάλογα με τη συμπεριφορά που εξετάζεται και να εκτελεστούν μέσα στην ίδια Frida session [112]. Με τον τρόπο αυτό αποφεύγεται η ανάγκη δημιουργίας ανεξάρτητων instrumentation scripts για κάθε έλεγχο και διατηρείται πιο οργανωμένη η παρακολούθηση διαφορετικών τμημάτων της εφαρμογής.

Η επιλογή του Medusa συνδέεται κυρίως με αυτή τη δυνατότητα επαναχρησιμοποίησης και σύνθεσης των hooks. Η βιβλιοθήκη των modules καλύπτει διαφορετικές κατηγορίες Android APIs, μεταξύ των οποίων Intents και WebViews, ενώ το framework επιτρέπει την προσθήκη νέου instrumentation όταν η εφαρμογή χρησιμοποιεί δικές της methods ή συμπεριφορές που δεν καλύπτονται από τα υπάρχοντα modules. Τα events που παράγονται κατά την εκτέλεση μπορούν να παρακολουθούνται και να καταγράφονται, ώστε να συσχετίζονται αργότερα με τα ευρήματα της στατικής ανάλυσης [112].

Τα runtime observations μπορούν παράλληλα να οδηγήσουν σε νέα σημεία ενδιαφέροντος που δεν είχαν εντοπιστεί κατά την αρχική στατική ανάλυση. Μια method call ή ένα argument που εμφανίζεται στα logs μπορεί να οδηγήσει σε νέα εξέταση του decompiled κώδικα και στη διαμόρφωση πρόσθετης υπόθεσης προς έλεγχο. Το Medusa λειτουργεί επομένως ως ενδιάμεσο επίπεδο μεταξύ της στατικής ανάλυσης και της άμεσης χρήσης του Frida, διατηρώντας την ευελιξία του instrumentation μέσα σε πιο οργανωμένη και επαναχρησιμοποιήσιμη μορφή.

```

[+] INCOMING INTENT: Intent { act=com.google.android.gms.safetynet.service.START pkg=com.google.android.gms }
    \_ intent package origin:com.google.android.gms
[+] INCOMING INTENT: Intent { act=com.google.android.gms.usagereporting.service.START pkg=com.google.android.gms }
    \_ intent package origin:com.google.android.gms
[i] exiting Webview.$init
[i] exiting Webview.$init
[i] exiting Webview.$init
[i] exiting Webview.$init
[+] INCOMING INTENT: Intent { cmp=oversecured.ovaa/.activities.WebViewActivity (has extras) }
    \_ String Extra, name: url , http://evilexample.com
[i] ----- Dumping webview settings -----:
====> Class Name: android.webkit.WebView
====> WebView Client: android.webkit.WebViewClient@16fb83a
    Allows Content Access: true
    Allows Javascript execution: true
    Allows File Access: false
    Allows File Access From File URLs: true
    Allows Universal Access from File URLs: false
[i] ----- Dumping webview settings EOF -----].
[i] Loading URL:http://evilexample.com

```

Εικόνα 35: Runtime παρακολούθηση της WebViewActivity μέσω Medusa μετά την ενεργοποίηση ελεγχόμενου Deep Link.

Burp Suite: Το Burp Suite χρησιμοποιείται στην παρούσα εργασία για την παρατήρηση της δικτυακής συμπεριφοράς της εφαρμογής κατά την εκτέλεση. Η Android συσκευή ή ο emulator ρυθμίζεται ώστε η HTTP/S κίνηση να διέρχεται μέσω του Burp Proxy, δίνοντας τη δυνατότητα καταγραφής και εξέτασης των requests και responses που παράγονται κατά τη λειτουργία της εφαρμογής [113]. Με αυτόν τον τρόπο, πληροφορίες που έχουν ήδη προκύψει από τη στατική ή τη δυναμική ανάλυση μπορούν να συσχετιστούν με την πραγματική δικτυακή επικοινωνία που ακολουθεί.

Η χρήση του Burp αποκτά ιδιαίτερη σημασία όταν μια πιθανή διαδρομή καταλήγει σε network operation, καθώς επιτρέπει να εξεταστεί το request που δημιουργείται στην πράξη και να προσδιοριστεί ο τελικός προορισμός της επικοινωνίας. Η πληροφορία αυτή μπορεί να συμπληρώσει τα runtime observations που έχουν ήδη συλλεχθεί από εργαλεία όπως το Medusa, επιτρέποντας τη σύνδεση μιας ελεγχόμενης εισόδου με το αντίστοιχο network αποτέλεσμα. Το HTTP history παρέχει παράλληλα εικόνα για τα endpoints με τα οποία επικοινωνεί η εφαρμογή και για τη μορφή των requests που παράγονται κατά την εκτέλεση, ενώ το Repeater μπορεί να χρησιμοποιηθεί όταν απαιτείται επανάληψη ενός ήδη καταγεγραμμένου request υπό διαφορετικές συνθήκες [131].

Chrome DevTools Protocol (CDP): Τα Android WebViews για τους οποίους έχει ενεργοποιηθεί το debugging μπορούν να εξεταστούν μέσω των Chrome DevTools και του Chrome DevTools Protocol (CDP), παρέχοντας άμεση εικόνα του web περιβάλλοντος που εκτελείται μέσα στην εφαρμογή. Μέσω αυτής της διεπαφής είναι δυνατή η επιθεώρηση του ενεργού document και της JavaScript εκτέλεσης, καθώς και η εξέταση των δεδομένων και των resources που είναι διαθέσιμα στο WebView κατά το runtime [114], [115].

Στην παρούσα εργασία το CDP χρησιμοποιείται κυρίως σε WebView-related διαδρομές, αφού προηγουμένως έχει επιβεβαιωθεί ότι το υπό εξέταση περιεχόμενο μπορεί πράγματι να φτάσει στο WebView. Το runtime instrumentation μπορεί να καταγράψει την τιμή που παρέχεται σε μια λειτουργία όπως η `loadUrl()`, ενώ η επιθεώρηση μέσω CDP επιτρέπει να εξεταστεί η κατάσταση που προκύπτει μετά τη φόρτωση μέσα στο web περιβάλλον. Με αυτόν τον τρόπο μπορούν να ελεγχθούν το τελικό document, η εκτέλεση JavaScript και οι διεπαφές ή τα δεδομένα που είναι διαθέσιμα στο φορτωμένο περιεχόμενο, προσθέτοντας ένα ακόμη επίπεδο runtime παρατήρησης στις διαδρομές που διασχίζουν το όριο μεταξύ της Android εφαρμογής και του WebView [114], [115].

Η δυναμική ανάλυση δεν αποτελεί αποκλειστικά το τελικό στάδιο μιας ήδη γνωστής διαδρομής. Οι παρατηρήσεις που προκύπτουν κατά την εκτέλεση μπορεί να αποκαλύψουν στοιχεία τα οποία δεν είχαν εντοπιστεί κατά την πρώτη εξέταση του κώδικα και να οδηγήσουν σε νέα στατική διερεύνηση. Η διαδικασία αποκτά επομένως επαναληπτικό χαρακτήρα, καθώς τα νέα runtime evidence τροφοδοτούν την επόμενη φάση της ανάλυσης και επιτρέπουν την αναθεώρηση ή περαιτέρω ανάπτυξη των αρχικών υποθέσεων.

3.3.3 Συντονισμός της διαδικασίας ανάλυσης

Η χρήση Large Language Models στην ανάλυση ασφάλειας λογισμικού δεν περιορίζεται πλέον στην παραγωγή μίας απάντησης ή στην ταξινόμηση ενός αποσπάσματος κώδικα. Στα σύγχρονα agentic systems, το μοντέλο μπορεί να συμμετέχει σε μια διαδοχική διαδικασία κατά την οποία αξιολογεί τις διαθέσιμες πληροφορίες, αλληλεπιδρά με εξωτερικά εργαλεία και προσαρμόζει την πορεία της ανάλυσης με βάση τα αποτελέσματα που προκύπτουν [116], [123]. Η μορφή αυτή ταιριάζει ιδιαίτερα στο vulnerability research, όπου η διερεύνηση μιας αρχικής υπόθεσης εξελίσσεται σταδιακά και μπορεί να απαιτεί μετάβαση από την εξέταση του κώδικα στην εκτέλεση της εφαρμογής και στην αξιολόγηση των runtime observations.

Η δυνατότητα ενός γλωσσικού μοντέλου να προσαρμόζει την πορεία της εργασίας με βάση πληροφορίες που αποκτά κατά την εκτέλεση έχει εξεταστεί ευρύτερα στη βιβλιογραφία των LLM agents. Οι Yao et al. παρουσίασαν το ReAct, μια προσέγγιση στην οποία το reasoning συνδέεται με ενέργειες προς το περιβάλλον και οι παρατηρήσεις που επιστρέφονται αξιοποιούνται για τη συνέχεια της διαδικασίας. Στα interactive benchmarks ALFWorld και WebShop, το ReAct παρουσίασε αύξηση του absolute success rate κατά 34% και 10% αντίστοιχα σε σχέση με τις αντίστοιχες baseline προσεγγίσεις [124]. Στην ίδια γενικότερη κατεύθυνση, το Toolformer εξετάζει τη χρήση εξωτερικών APIs από ένα language model, ενσωματώνοντας τις κλήσεις και τα αποτελέσματά τους στην επεξεργασία που ακολουθεί [125]. Παρότι οι εργασίες αυτές δεν αφορούν την ασφάλεια εφαρμογών Android, παρέχουν τη βάση για μια διαδικασία στην

οποία το μοντέλο μπορεί να αποκτά νέα πληροφορία κατά τη διάρκεια της ανάλυσης και να αξιοποιεί τα αποτελέσματά της στα επόμενα βήματα.

Για τον λόγο αυτό, στην παρούσα εργασία το LLM χρησιμοποιείται ως agent και orchestration layer πάνω από τα εργαλεία που παρουσιάστηκαν στις προηγούμενες ενότητες. Ο ρόλος του δεν περιορίζεται στην επεξεργασία των πληροφοριών που έχουν ήδη δοθεί, αλλά επεκτείνεται στην αναζήτηση πρόσθετων στοιχείων όταν η εξέλιξη της διερεύνησης το απαιτεί. Μια αρχική υπόθεση μπορεί έτσι να οδηγήσει σε νέα εξέταση του κώδικα ή σε δυναμικό έλεγχο, ενώ τα αποτελέσματα που προκύπτουν ενσωματώνονται στην κατάσταση της ανάλυσης και επηρεάζουν τις επόμενες αποφάσεις του agent.

Η λειτουργία αυτή υποστηρίζεται στην υλοποίηση από ένα σύνολο μηχανισμών που καλύπτουν διαφορετικές ανάγκες της διαδικασίας. Το SKILL.md καθορίζει τις σταθερές οδηγίες και τη μεθοδολογία που ακολουθεί ο agent, ενώ το Model Context Protocol παρέχει την πρόσβαση στα εξωτερικά εργαλεία που χρησιμοποιούνται κατά την ανάλυση. Παράλληλα, η ιδιωτική knowledge base διατηρεί τεχνική γνώση που μπορεί να αξιοποιηθεί σε επόμενες διερευνήσεις και η κατάσταση της αξιολόγησης καταγράφεται εξωτερικά, ώστε η πληροφορία που έχει ήδη συλλεχθεί να παραμένει διαθέσιμη καθώς η ανάλυση εξελίσσεται σε περισσότερα στάδια.

SKILL.md και καθοδήγηση του agent: Το SKILL.md περιγράφει τη μεθοδολογία που ακολουθεί ο agent κατά τη διαδικασία αξιολόγησης και καθορίζει τους βασικούς κανόνες με τους οποίους οργανώνεται η διερεύνηση. Οι οδηγίες αυτές παρέχονται εξωτερικά στο μοντέλο και δεν αποτελούν μορφή fine-tuning ή πρόσθετης εκπαίδευσης. Με αυτή τη σχεδίαση, το LLM λειτουργεί μέσα σε ένα σταθερό πλαίσιο αξιολόγησης, όπου έχουν ήδη οριστεί οι προϋποθέσεις για την εξέταση ενός candidate και τα κριτήρια που πρέπει να ικανοποιούνται πριν αυτό χαρακτηριστεί ως finding. Η μεθοδολογία μπορεί έτσι να τροποποιείται ή να επεκτείνεται ανεξάρτητα από το υποκείμενο μοντέλο, χωρίς να απαιτείται νέα διαδικασία εκπαίδευσης. Η δομή του SKILL.md και ο τρόπος με τον οποίο χρησιμοποιείται κατά την ανάλυση παρουσιάζονται αναλυτικότερα στην επόμενη ενότητα.

Model Context Protocol και εξωτερικά εργαλεία: Η πρόσβαση στα εργαλεία ανάλυσης πραγματοποιείται, όπου αυτό υποστηρίζεται, μέσω του Model Context Protocol (MCP). Το MCP παρέχει έναν κοινό τρόπο με τον οποίο εξωτερικά συστήματα μπορούν να διαθέτουν λειτουργίες και πληροφορίες σε έναν AI client μέσω tools και resources [117]. Με αυτή τη σχεδίαση, ο agent μπορεί να αλληλεπιδρά με διαφορετικά εργαλεία χωρίς να εξαρτάται από τον τρόπο με τον οποίο έχει υλοποιηθεί εσωτερικά το καθένα. Στην πράξη, η διερεύνηση μπορεί να απαιτεί την ανάκτηση κώδικα ή cross-references από ένα εργαλείο στατικής ανάλυσης και στη συνέχεια τη συλλογή runtime ή δικτυακών παρατηρήσεων από

διαφορετικό περιβάλλον. Όταν υπάρχει κατάλληλη MCP διεπαφή, οι ενέργειες αυτές μπορούν να πραγματοποιούνται απευθείας μέσα στην ίδια διαδικασία, χωρίς να απαιτείται χειροκίνητη μεταφορά των αποτελεσμάτων προς το μοντέλο.

Knowledge base και προηγούμενη τεχνική γνώση: Ο agent μπορεί επίσης να αξιοποιεί μια ιδιωτική knowledge base, στην οποία διατηρούνται τεχνικές πληροφορίες από ευπάθειες που έχουν εξεταστεί σε προηγούμενες αξιολογήσεις. Η πληροφορία αυτή λειτουργεί συμπληρωματικά προς τη γενική μεθοδολογία και παρέχει παραδείγματα πραγματικών διαδρομών εκμετάλλευσης μαζί με τα στοιχεία που χρησιμοποιήθηκαν για τη διερεύνηση και την επιβεβαίωσή τους. Όταν κατά την ανάλυση εμφανίζεται μια παρόμοια περίπτωση, ο agent μπορεί να ανακτήσει σχετική προηγούμενη γνώση και να τη χρησιμοποιήσει ως αφετηρία για τη διαμόρφωση νέων υποθέσεων.

Η χρήση εξωτερικής γνώσης είναι ιδιαίτερα χρήσιμη σε περιπτώσεις όπου το μοντέλο δεν διαθέτει επαρκή πληροφορία για το πρόβλημα που εξετάζεται, καθώς τα LLMs μπορούν να παράγουν απαντήσεις που εμφανίζονται ως λογικές χωρίς να στηρίζονται σε επαρκή τεκμηρίωση [133]. Η ανάκτηση σχετικού υλικού περιορίζει την ανάγκη να βασιστεί η ανάλυση αποκλειστικά στην παραμετρική γνώση του μοντέλου και παρέχει συγκεκριμένα στοιχεία πάνω στα οποία μπορεί να στηριχθεί η επόμενη υπόθεση. Η λογική αυτή συνδέεται με τις retrieval-augmented προσεγγίσεις, όπου η εξωτερική πληροφορία χρησιμοποιείται για την ενίσχυση της γνώσης που είναι διαθέσιμη κατά την παραγωγή της απάντησης [126]. Οι Shuster et al. έδειξαν ότι η ενσωμάτωση retrieval σε knowledge-grounded dialogue μπορεί να μειώσει σημαντικά το φαινόμενο της knowledge hallucination [132].

Εξωτερική κατάσταση αξιολόγησης: Μέρος της πληροφορίας που παράγεται κατά τη διερεύνηση διατηρείται εκτός του άμεσου conversational context του LLM, ώστε η πορεία της ανάλυσης να μην εξαρτάται αποκλειστικά από το ιστορικό που παραμένει διαθέσιμο στο ενεργό context window. Στην εξωτερική αυτή κατάσταση καταγράφονται οι υποθέσεις που έχουν διαμορφωθεί, τα αποτελέσματα των ελέγχων που έχουν ήδη πραγματοποιηθεί και τα σημεία που παραμένουν προς διερεύνηση, επιτρέποντας στον agent να συνεχίζει την αξιολόγηση έχοντας διαθέσιμη την προηγούμενη πρόοδο της διαδικασίας. Η διατήρηση της κατάστασης μιας πολυσταδιακής εργασίας αποτελεί σημαντικό στοιχείο των LLM-based software-engineering agents, όταν η εκτέλεση εκτείνεται σε μεγάλο αριθμό διαδοχικών ενεργειών [156], [165].

Συνολικά, τα εργαλεία και οι μηχανισμοί που επιλέχθηκαν εντάσσονται σε μια κοινή διαδικασία αξιολόγησης, στην οποία η πληροφορία που προκύπτει σε κάθε στάδιο αξιοποιείται στη συνέχεια της διερεύνησης. Τα ευρήματα της στατικής ανάλυσης μπορούν να κατευθύνουν την εκτέλεση συγκεκριμένων

δυναμικών ελέγχων, ενώ οι παρατηρήσεις που προκύπτουν κατά το runtime επιστρέφουν στη διαδικασία και συμβάλλουν στην περαιτέρω αξιολόγηση της αρχικής υπόθεσης. Ο LLM-based agent συντονίζει αυτή τη ροή και διατηρεί τη σύνδεση μεταξύ των πληροφοριών που συλλέγονται από τα διαφορετικά εργαλεία. Η συνολική οργάνωση των επιμέρους στοιχείων και ο τρόπος με τον οποίο συνεργάζονται παρουσιάζονται στην επόμενη ενότητα.

3.4 Αρχιτεκτονική

Η αρχιτεκτονική της προτεινόμενης προσέγγισης σχεδιάστηκε ώστε η ανάλυση ασφάλειας μιας Android εφαρμογής να πραγματοποιείται ως ενιαία και επαναληπτική διαδικασία, στην οποία τα στοιχεία που προκύπτουν σε κάθε στάδιο αξιοποιούνται στη συνέχεια της διερεύνησης. Όπως παρουσιάζεται στην Εικόνα 36, κεντρικό ρόλο έχει ο LLM agent, ο οποίος συντονίζει την αλληλεπίδραση με την υπό εξέταση εφαρμογή και τα εργαλεία στατικής και δυναμικής ανάλυσης, ακολουθώντας τη μεθοδολογία αξιολόγησης που έχει οριστεί για το σύστημα. Τα εργαλεία reverse engineering, instrumentation και network inspection παραμένουν υπεύθυνα για την παραγωγή της τεχνικής πληροφορίας, ενώ ο agent αξιοποιεί τα αποτελέσματά τους για να καθορίσει πώς θα συνεχιστεί η ανάλυση.

Η διαδικασία ξεκινά από το APK της εφαρμογής και από μια εγκατεστημένη έκδοσή της σε Android συσκευή. Το APK χρησιμοποιείται για την ανάκτηση της πληροφορίας που απαιτείται στη στατική ανάλυση, ενώ η συσκευή αποτελεί το περιβάλλον στο οποίο εκτελούνται οι δυναμικοί έλεγχοι. Ο agent συνδυάζει τις πληροφορίες που προκύπτουν από τα δύο επίπεδα, ακολουθώντας τη μεθοδολογία που ορίζεται από το Android Client-Side Assessment Skill, και χρησιμοποιεί τα εργαλεία της Ενότητας 3.3 μέσω MCP ή, όπου αυτό είναι πιο κατάλληλο, μέσω command-line interfaces. Παράλληλα, η κατάσταση της αξιολόγησης διατηρείται εξωτερικά, ώστε οι υποθέσεις που παραμένουν ανοιχτές και τα στοιχεία που έχουν ήδη συγκεντρωθεί να παραμένουν διαθέσιμα καθώς η διερεύνηση εξελίσσεται, ανεξάρτητα από το ενεργό context του γλωσσικού μοντέλου.



Εικόνα 36: Συνολική αρχιτεκτονική της προτεινόμενης προσέγγισης.

Η πρώτη ουσιαστική φάση αφορά τη στατική χαρτογράφηση της εφαρμογής. Ο agent εξετάζει αρχικά τα σημεία από τα οποία μπορεί να εισέλθει πληροφορία ή επιρροή από το εξωτερικό περιβάλλον, όπως externally reachable components, Deep Links ή δεδομένα που λαμβάνονται μέσω Intents και παράλληλα αναζητά λειτουργίες των οποίων η χρήση μπορεί να έχει σημασία για την ασφάλεια, όπως WebView operations, πρόσβαση σε αρχεία ή providers, network operations και μεταβάσεις προς native code. Σκοπός σε αυτό το στάδιο δεν είναι να χαρακτηριστούν τα σημεία αυτά ως ευάλωτα, αλλά να δημιουργηθεί μια αρχική εικόνα των πιθανών sources και sinks της εφαρμογής.

Αφού εντοπιστούν τα βασικά σημεία ενδιαφέροντος, η ανάλυση προχωρά στη διερεύνηση των σχέσεων μεταξύ τους. Μια externally controlled είσοδος μπορεί να ακολουθηθεί μέσα από διαδοχικές method calls, μετασχηματισμούς και μεταβάσεις μεταξύ Android components μέχρι να προσδιοριστεί η τελική χρήση της, ενώ η αντίστροφη πορεία ξεκινά από μια security-sensitive λειτουργία και εξετάζει τους callers και τις συνθήκες που μπορούν να οδηγήσουν στην εκτέλεσή της. Η χρήση και των δύο κατευθύνσεων βοηθά ιδιαίτερα σε εφαρμογές όπου η σχετική λογική είναι καταναμημένη σε περισσότερα components ή επίπεδα υλοποίησης και η σύνδεση ανάμεσα στην εξωτερική είσοδο και την τελική ενέργεια δεν είναι εμφανής από μία μόνο method. Στην ίδια διαδρομή εξετάζονται οι μηχανισμοί προστασίας που συναντώνται στον

κώδικα, όπως validation, permission checks, caller restrictions ή άλλοι έλεγχοι που μπορεί να περιορίζουν την επιρροή του εξωτερικού actor πριν αυτή φτάσει στη λειτουργία που εξετάζεται.

Μια διαδρομή που παρουσιάζει ενδιαφέρον μετατρέπεται στη συνέχεια σε συγκεκριμένη υπόθεση προς επαλήθευση, αφού πρώτα προσδιοριστεί ποιο μέρος της έχει ήδη τεκμηριωθεί στατικά και ποια πληροφορία εξακολουθεί να λείπει. Αν, για παράδειγμα, ο κώδικας δείχνει ότι μια τιμή που προέρχεται από Deep Link μεταφέρεται μέσω της εφαρμογής και χρησιμοποιείται αργότερα από ένα WebView, η στατική ανάλυση έχει ήδη προσδιορίσει μια πιθανή σχέση μεταξύ του entry point και του sink, ενώ το πραγματικό runtime behavior της διαδρομής παραμένει το επόμενο ερώτημα της διερεύνησης. Ο δυναμικός έλεγχος που ακολουθεί επιλέγεται με βάση αυτή την εκκρεμή πληροφορία, ώστε κάθε test να απαντά σε ένα σαφές ερώτημα που προέκυψε από την προηγούμενη εξέταση του κώδικα.

Η δυναμική φάση εκτελεί τις υποθέσεις αυτές στο πραγματικό περιβάλλον της εφαρμογής, χρησιμοποιώντας τον τρόπο ενεργοποίησης που ταιριάζει στο αντίστοιχο threat model και παρακολουθώντας τα σημεία της διαδρομής που έχουν ήδη αναγνωριστεί στατικά. Όταν το ζητούμενο είναι να επιβεβαιωθεί η μεταφορά μιας externally controlled τιμής προς συγκεκριμένο sink, το runtime instrumentation επικεντρώνεται στην πραγματική τιμή που εμφανίζεται κατά την εκτέλεση και στις συνθήκες κάτω από τις οποίες πραγματοποιείται η σχετική κλήση. Σε διαδρομές που συνεχίζονται προς WebView ή δικτυακή επικοινωνία, η παρατήρηση μπορεί να επεκταθεί στα αντίστοιχα εργαλεία, ώστε η ανάλυση να ακολουθήσει τη συμπεριφορά και πέρα από το αρχικό Java ή Kotlin επίπεδο και να καταγράψει το αποτέλεσμα που παράγεται στο επόμενο στάδιο.

Η επιλογή του runtime test λαμβάνει υπόψη και τις ιδιότητες του actor που υποτίθεται ότι ενεργοποιεί τη διαδρομή. Ένα entry point που είναι προσβάσιμο μέσω ενός συνηθισμένου Deep Link μπορεί να εξεταστεί με άμεση ενεργοποίηση, ενώ περιπτώσεις όπου η συμπεριφορά εξαρτάται από caller identity, permissions ή άλλες ιδιότητες του Android security model χρειάζονται συνθήκες που προσεγγίζουν περισσότερο εκείνες ενός πραγματικού εξωτερικού caller. Το instrumentation χρησιμοποιείται κυρίως για την παρατήρηση αυτής της συμπεριφοράς και των δεδομένων που μεταφέρονται κατά την εκτέλεση, ενώ τυχόν προσωρινές παρεμβάσεις σε arguments ή return values αξιοποιούνται μόνο για την καλύτερη κατανόηση μιας διαδρομής και αξιολογούνται χωριστά από τις συνθήκες που μπορούν να δημιουργηθούν κανονικά από τον επιτιθέμενο.

Κατά την εκτέλεση των δυναμικών ελέγχων μπορεί επίσης να εμφανιστούν πληροφορίες που δεν είχαν εντοπιστεί κατά την αρχική στατική εξέταση, όπως μια άγνωστη method call, μια τιμή που προέρχεται από διαφορετικό component ή μια διαδρομή εκτέλεσης που αποκλίνει από εκείνη που είχε αρχικά υποτεθεί. Ο

agent επιστρέφει τότε στον αντίστοιχο κώδικα και εξετάζει πώς συνδέεται η νέα παρατήρηση με όσα έχουν ήδη συγκεντρωθεί, αναθεωρώντας την υπάρχουσα υπόθεση ή δημιουργώντας νέο σημείο προς διερεύνηση όταν αυτό δικαιολογείται από τα δεδομένα. Η διαδικασία αποκτά έτσι επαναληπτικό χαρακτήρα, καθώς η στατική ανάλυση καθοδηγεί τα runtime tests και τα αποτελέσματα της εκτέλεσης επηρεάζουν με τη σειρά τους την επόμενη εξέταση του κώδικα.

Η συνολική πορεία της αξιολόγησης οργανώνεται από το **Android Client-Side Assessment Skill**, το οποίο περιλαμβάνει τις βασικές απαιτήσεις της μεθοδολογίας και τις ειδικότερες οδηγίες που χρειάζονται ανάλογα με το attack surface που συναντάται στην εφαρμογή. Οι γενικοί κανόνες παραμένουν διαθέσιμοι σε όλη τη διάρκεια της διερεύνησης, ενώ η πληροφορία που αφορά συγκεκριμένες vulnerability classes εισάγεται όταν εντοπιστεί σχετικό τμήμα της εφαρμογής, όπως WebView, Content Provider ή κάποιος άλλος μηχανισμός που απαιτεί διαφορετική διαδικασία εξέτασης. Η modular αυτή οργάνωση περιορίζει την ποσότητα άσχετης πληροφορίας που μεταφέρεται στο context και επιτρέπει στη μεθοδολογία να επεκτείνεται με νέες κατηγορίες χωρίς να αυξάνεται αντίστοιχα το σύνολο των οδηγιών που χρειάζεται να επεξεργάζεται ο agent σε κάθε βήμα.

Συμπληρωματικά προς τη μεθοδολογία, η ιδιωτική knowledge base παρέχει προηγούμενη τεχνική γνώση που μπορεί να αξιοποιηθεί κατά τη διαμόρφωση των hypotheses. Μια γνωστή μορφή source-to-sink διαδρομής ή ένα παλαιότερο finding μπορεί να υποδείξει ποιο σημείο του νέου στόχου αξίζει να εξεταστεί περισσότερο και να κατευθύνει την αναζήτηση προς σχέσεις που διαφορετικά θα ήταν δυσκολότερο να αναγνωριστούν. Η πληροφορία που ανακτάται χρησιμοποιείται επομένως ως επιπλέον context για τη διερεύνηση, ενώ η τεκμηρίωση της νέας περίπτωσης προκύπτει από το APK, το runtime περιβάλλον και τα evidence που συλλέγονται ειδικά για την εφαρμογή που αξιολογείται.

Η συνέχεια της διαδικασίας υποστηρίζεται από την εξωτερική κατάσταση της αξιολόγησης, όπου διατηρούνται οι υποθέσεις που έχουν σχηματιστεί, οι έλεγχοι που έχουν πραγματοποιηθεί, τα αποτελέσματά τους και τα σημεία που χρειάζονται περαιτέρω διερεύνηση. Η καταγραφή αυτή επιτρέπει στον agent να επιστρέφει σε προηγούμενες διαδρομές και να συσχετίζει observations που εμφανίστηκαν σε διαφορετικά στάδια της ανάλυσης, ενώ παράλληλα παρέχει μια πιο σταθερή εικόνα της προόδου από εκείνη που θα μπορούσε να διατηρηθεί αποκλειστικά μέσα στο ενεργό conversational context. Η ίδια κατάσταση χρησιμοποιείται για την παρακολούθηση της κάλυψης της αξιολόγησης, ώστε να παραμένει εμφανές ποια τμήματα του attack surface έχουν εξεταστεί επαρκώς και σε ποια εξακολουθούν να υπάρχουν ανοιχτά ερωτήματα.

Σε μεγαλύτερες εφαρμογές, ο κύριος agent μπορεί να αναθέσει περιορισμένα και σαφώς οριοθετημένα tasks σε επιμέρους agents, όταν ένα μέρος της διερεύνησης μπορεί να εξεταστεί ανεξάρτητα χωρίς να απαιτείται το σύνολο του διαθέσιμου context. Μια τέτοια εργασία μπορεί να αφορά την ανάλυση μιας συγκεκριμένης διαδρομής, ενός component ή ενός περιορισμένου τμήματος του attack surface, με τα αποτελέσματα να επιστρέφουν στη συνέχεια στον κύριο agent και να ενσωματώνονται στην κοινή κατάσταση της αξιολόγησης. Η αποσύνθεση αυτή βοηθά στη διαχείριση πιο σύνθετων targets, ενώ η σύνθεση των observations και η συνολική αξιολόγηση παραμένουν συγκεντρωμένες στο επίπεδο του κύριου agent.

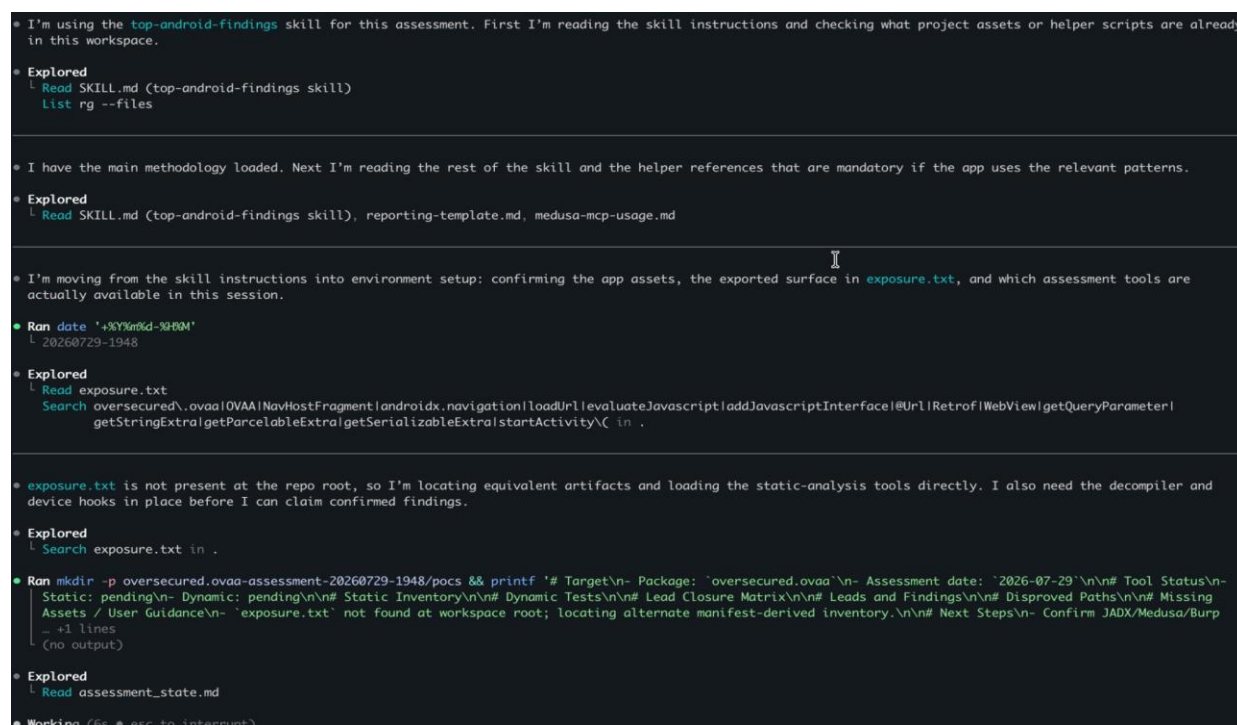
Η επιβεβαίωση ενός ευρήματος πραγματοποιείται όταν τα evidence που έχουν συγκεντρωθεί σχηματίζουν μια πλήρη security-relevant διαδρομή, στην οποία υπάρχει εξωτερικά ελεγχόμενη πηγή, η επιρροή αυτή φτάνει σε security-sensitive λειτουργία, η σύνδεση έχει παρατηρηθεί κατά την πραγματική εκτέλεση και από τη συγκεκριμένη συμπεριφορά προκύπτει σαφής επίπτωση στην ασφάλεια. Η απαίτηση αυτής της συνολικής εικόνας επιτρέπει να διατηρούνται ως leads περιπτώσεις που έχουν τεχνικό ενδιαφέρον χωρίς ακόμη να παρέχουν επαρκή στοιχεία για reportability, όπως ένα exported component ή μια ελεγχόμενη τιμή που καταλήγει σε κάποιο σημαντικό API χωρίς να έχει προσδιοριστεί η πρακτική συνέπεια της χρήσης της. Το finding gate συνδέει επομένως την τελική αξιολόγηση με την πλήρη διαδρομή και με τη δυνατότητα που αποκτά ο επιτιθέμενος μέσα από αυτή.

Ανάλογη διαδικασία ακολουθείται όταν τα αποτελέσματα ενός test δεν επιβεβαιώνουν την αρχική υπόθεση. Η απουσία του αναμενόμενου runtime event εξετάζεται μαζί με τις συνθήκες υπό τις οποίες πραγματοποιήθηκε ο έλεγχος και με το statically reconstructed path, ώστε να εντοπιστεί το σημείο στο οποίο η διαδρομή σταματά ή η εξωτερική επιρροή παύει να είναι ουσιαστική. Ένας αποτελεσματικός έλεγχος πρόσβασης, ένα component που δεν μπορεί να προσεγγιστεί από τον modeled attacker ή ένας validation μηχανισμός που αφαιρεί τον έλεγχο της κρίσιμης τιμής μπορούν να αποτελέσουν τέτοιο όριο και να δώσουν τεχνική βάση για το κλείσιμο της συγκεκριμένης υπόθεσης.

Με την ολοκλήρωση της διερεύνησης, τα τελικά συμπεράσματα επανασυσχετίζονται με τα artifacts που έχουν συγκεντρωθεί κατά τα διαφορετικά στάδια της ανάλυσης, ώστε κάθε reportable finding να συνδέεται με τον σχετικό κώδικα, το runtime test και τα observations που τεκμηριώνουν τη συμπεριφορά. Η αναφορά που παράγεται περιγράφει τη διαδρομή και την επίπτωσή της με βάση αυτά τα στοιχεία, διατηρώντας παράλληλα τη δυνατότητα επανεξέτασης του evidence ανεξάρτητα από την προηγούμενη reasoning διαδικασία του agent. Πριν κλείσει η αξιολόγηση πραγματοποιείται επίσης ντετερμινιστικός έλεγχος της καταγεγραμμένης κατάστασης, ο οποίος εξετάζει αν οι ανοιχτές διαδρομές, τα tests και τα findings έχουν

καταχωριστεί με συνεπή τρόπο και αν η δηλωμένη κάλυψη συμφωνεί με τα artifacts που έχουν παραχθεί κατά τη διάρκεια της διαδικασίας.

Η εφαρμογή **OVAA (Oversecured Vulnerable Android App)** [134] χρησιμοποιείται στην παρούσα εργασία ως ενδεικτικό παράδειγμα για την παρουσίαση του τρόπου με τον οποίο η γενική μεθοδολογία εφαρμόζεται σε μια συγκεκριμένη διαδικασία αξιολόγησης. Στόχος του παραδείγματος είναι να αποτυπωθεί η πορεία της ανάλυσης από την αρχική χαρτογράφηση της εφαρμογής και τη διαμόρφωση των πρώτων υποθέσεων μέχρι τη δυναμική επαλήθευση και τη συλλογή των τεκμηρίων που χρησιμοποιούνται στην τελική αξιολόγηση. Η OVAA προσφέρει κατάλληλο περιβάλλον για τον σκοπό αυτό, καθώς πρόκειται για εφαρμογή σχεδιασμένη με γνωστές Android security vulnerabilities και επιτρέπει την επανάληψη των ίδιων ελέγχων σε ελεγχόμενες συνθήκες.



```
* I'm using the top-android-findings skill for this assessment. First I'm reading the skill instructions and checking what project assets or helper scripts are already in this workspace.

* Explored
  ↳ Read SKILL.md (top-android-findings skill)
  ↳ List rg --files

* I have the main methodology loaded. Next I'm reading the rest of the skill and the helper references that are mandatory if the app uses the relevant patterns.

* Explored
  ↳ Read SKILL.md (top-android-findings skill), reporting-template.md, medusa-mcp-usage.md

* I'm moving from the skill instructions into environment setup: confirming the app assets, the exported surface in exposure.txt, and which assessment tools are actually available in this session.

* Ran date '+%Y%m%d-%H%M'
  ↳ 20260729-1948

* Explored
  ↳ Read exposure.txt
  ↳ Search oversecured\,ovaa\OVAA\NavHostFragment\androidx.navigation\loadUrl\evaluateJavascript\addJavascriptInterface\@Url\|Retrofit\WebView\getQueryParameter\getStringExtra\getParcelableExtra\getSerializableExtra\startActivity\(\ in .

* exposure.txt is not present at the repo root, so I'm locating equivalent artifacts and loading the static-analysis tools directly. I also need the decompiler and device hooks in place before I can claim confirmed findings.

* Explored
  ↳ Search exposure.txt in .

* Ran mkdir -p oversecured.ovaa-assessment-20260729-1948/pocs && printf '# Target\n- Package: `oversecured.ovaa`\n- Assessment date: `2026-07-29`\n\n# Tool Status\n- Static: pending\n- Dynamic: pending\n\n# Static Inventory\n\n# Dynamic Tests\n\n# Lead Closure Matrix\n\n# Leads and Findings\n\n# Disproved Paths\n\n# Missing Assets / User Guidance\n- `exposure.txt` not found at workspace root; locating alternate manifest-derived inventory.\n\n# Next Steps\n- Confirm JADX/Medusa/Burp'
  ↳ - +1 lines
  ↳ (no output)

* Explored
  ↳ Read assessment_state.md

* Working (6s • esc to interrupt)
```

Εικόνα 37: Αρχικοποίηση της διαδικασίας αξιολόγησης από τον agent.

Η αξιολόγηση ξεκινά με τη φόρτωση του κεντρικού SKILL.md και των βασικών οδηγιών της μεθοδολογίας, ενώ παράλληλα εξετάζονται τα artifacts που είναι διαθέσιμα για το συγκεκριμένο target. Στο ίδιο στάδιο δημιουργείται η αρχική κατάσταση της αξιολόγησης, στην οποία θα καταγράφονται η κάλυψη, οι υποθέσεις που προκύπτουν και τα τεκμήρια που συγκεντρώνονται καθώς εξελίσσεται η διερεύνηση. Αφού ολοκληρωθεί αυτή η προετοιμασία, ο agent μπορεί να περάσει στην πρώτη τεχνική απογραφή της εφαρμογής.

```

• exposure.txt is not present at the repo root, so I'm locating equivalent artifacts and loading the static-analysis tools directly. I also need the decompiler and device hooks in place before I can claim confirmed findings.

• Explored
  ↳ Search exposure.txt in .

• Ran mkdir -p oversecured.ovaa-assessment-20260729-1948/pocs && printf '# Target\n- Package: `oversecured.ovaa`\n- Assessment date: `2026-07-29`\n\n# Tool Status\n- Static: pending\n- Dynamic: pending\n\n# Static Inventory\n\n# Dynamic Tests\n\n# Lead Closure Matrix\n\n# Leads and Findings\n\n# Disproved Paths\n\n# Missing Assets / User Guidance\n- `exposure.txt` not found at workspace root; locating alternate manifest-derived inventory.\n\n# Next Steps\n- Confirm JADX/Medusa/Burp
  ↳ ... +1 lines
  ↳ (no output)

• Explored
  ↳ Read assessment_state.md

• The decompiler and traffic tools are available. I'm now building the sink inventory directly against the loaded APK, starting with WebViews, deeplink handlers, and exported components for oversecured.ovaa.

• Called jadx.search_classes_by_keyword({"search_term":"oversecured.ovaa","search_in":"code,class","count":50})
  ↳ {"type": "class-list", "items": ["androidx.appcompat.R", "androidx.constraintlayout.widget.R", "androidx.core.R", "androidx.customview.R", "androidx.drawerlayout.R", "androidx.loader.R", "androidx.vectordrawable.R", "androidx.vectordrawable.animated.R", "androidx.viewpager.R", "oversecured.ovaa.OversecuredApplication", "oversecured.ovaa.R", "oversecured.ovaa.activities.DeepLinkActivity", "oversecured.ovaa.activities.EntranceActivity", "oversecured.ovaa.activities.LoginActivity", "oversecured.ovaa.activities.MainActivity", "oversecured.ovaa.activities.WebViewActivity", "oversecured.ovaa.network.LoginService", "oversecured.ovaa.network.RetrofitInstance", "oversecured.ovaa.objects.DeleteFilesSerializable", "oversecured.ovaa.objects.LoginData", "oversecured.ovaa.objects.MemoryCorruptionParcelable", "overs...

• Called jadx.search_classes_by_keyword({"search_term":"WebView","search_in":"code,class","count":50})
  ↳ {"type": "class-list", "items": ["oversecured.ovaa.activities.WebViewActivity", "androidx.core.text.util.LinkifyCompat", "oversecured.ovaa.R", "oversecured.ovaa.activities.DeepLinkActivity", "oversecured.ovaa.activities.MainActivity"], "pagination": {"total": 5, "offset": 0, "limit": 5, "count": 5, "has_more": false, "current_page": 1, "total_pages": 1, "page_size": 5}}

• Called jadx.search_classes_by_keyword({"search_term":"loadUrl","search_in":"code,method","count":50})
  ↳ {"type": "class-list", "items": ["oversecured.ovaa.activities.WebViewActivity"], "pagination": {"total": 1, "offset": 0, "limit": 1, "count": 1, "has_more": false, "current_page": 1, "total_pages": 1, "page_size": 1}}

• Called jadx.search_classes_by_keyword({"search_term":"evaluateJavascript","search_in":"code,method","count":50})
  ↳ {"type": "class-list", "items": [], "pagination": {"total": 0, "offset": 0, "limit": 0, "count": 0, "has_more": false}}

• Called jadx.search_classes_by_keyword({"search_term":"getQueryParameter","search_in":"code,method","count":50})
  ↳ {"type": "class-list", "items": ["androidx.core.content.FileProvider", "oversecured.ovaa.activities.DeepLinkActivity"], "pagination": {"total": 2, "offset": 0, "limit": 2, "count": 2, "has_more": false, "current_page": 1, "total_pages": 1, "page_size": 2}}

• Called jadx.search_classes_by_keyword({"search_term":"getStringExtra","search_in":"code,method","count":50})
  ↳ {"type": "class-list", "items": ["androidx.core.app.ShareCompat", "oversecured.ovaa.activities.WebViewActivity"], "pagination": {"total": 2, "offset": 0, "limit": 2, "count": 2, "has_more": false, "current_page": 1, "total_pages": 1, "page_size": 2}}

• Working (9s • esc to interrupt)

```

Εικόνα 38 : Αρχική χαρτογράφηση της επιφάνειας επίθεσης μέσω του JADX

Στην περίπτωση της OVAA δεν υπήρχε διαθέσιμη έτοιμη απογραφή της επιφάνειας επίθεσης, επομένως ο agent χρησιμοποίησε το **JADX** για να σχηματίσει την αρχική εικόνα του target μέσα από το Manifest και τον decompiled κώδικα. Η εξέταση επικεντρώθηκε στα externally reachable components και στις διαδρομές μέσω των οποίων δεδομένα μπορούν να εισέλθουν στην εφαρμογή, ενώ ταυτόχρονα εντοπίστηκαν λειτουργίες με πιθανή security relevance, όπως WebView και network-related code. Η χαρτογράφηση αυτή καθόρισε τα σημεία που άξιζαν περαιτέρω εξέταση και έδωσε στον agent τα πρώτα ζεύγη πιθανών sources και security-sensitive λειτουργιών.

```

• Called jadx.search_classes_by_keyword({"search_term":"getStringExtra","search_in":"code,method","count":50})
  ↳ {"type": "class-list", "items": [{"androidx.core.app.ShareCompat", "oversecured.ovaa.activities.WebViewActivity"}, {"pagination": {"total": 2, "offset": 0, "limit": 2, "count": 2, "has_more": false, "current_page": 1, "total_pages": 1, "page_size": 2}}]

• I have the target classes. I'm pulling full sources for the app-owned activities, providers, services, and network helpers so I can build the sink-to-caller matrix instead of guessing from symbol names.

• Called jadx.get_package_tree({})
  ↳ {"total_packages": 1144, "total_classes": 6060, "packages": [{"is_likely_library": true, "name": "kotlinx.coroutines.flow", "class_count": 217}, {"is_likely_library": true, "name": "kotlinx.coroutines", "class_count": 148}, {"is_likely_library": true, "name": "kotlin.collections", "class_count": 124}, {"is_likely_library": true, "name": "org.apache.commons.io.input", "class_count": 115}, {"is_likely_library": true, "name": "org.apache.commons.io.function", "class_count": 110}, {"is_likely_library": true, "name": "kotlinx.coroutines.channels", "class_count": 107}, {"is_likely_library": false, "name": "kotlin", "class_count": 97}, {"is_likely_library": true, "name": "androidx.core.view", "class_count": 91}, {"is_likely_library": true, "name": "androidx.lifecycle", "class_count": 91}, {"is_likely_library": ...

• Called jadx.get_class_source({"class_name":"oversecured.ovaa.activities.DeepLinkActivity"})
  ↳ {"response": "package oversecured.ovaa.activities;\n\nimport android.content.Intent;\nimport android.net.Uri;\nimport android.os.Bundle;\nimport androidx.appcompat.app.AppCompatActivity;\nimport oversecured.ovaa.utils.LoginUtils;\n\n/* loaded from: classes8.dex */\npublic class DeepLinkActivity extends AppCompatActivity {\n    private static final int URI_GRANT_CODE = 1003;\n    private LoginUtils loginUtils;\n\n    @Override // androidx.fragment.app.FragmentActivity, androidx.activity.ComponentActivity, androidx.core.app.ComponentActivity, android.app.Activity\n    onCreate(Bundle savedInstanceState) {\n        Uri uri;\n        super.onCreate(savedInstanceState);\n        this.loginUtils = LoginUtils.getInstance(this);\n        Intent intent = getIntent();\n        if (intent != null ...

• Called jadx.get_class_source({"class_name":"oversecured.ovaa.activities.WebViewActivity"})
  ↳ {"response": "package oversecured.ovaa.activities;\n\nimport android.os.Bundle;\nimport android.webkit.WebChromeClient;\nimport android.webkit.WebView;\nimport android.webkit.WebViewClient;\nimport androidx.appcompat.app.AppCompatActivity;\nimport oversecured.ovaa.R;\n\n/* loaded from: classes8.dex */\npublic class WebViewActivity extends AppCompatActivity {\n    @Override // androidx.fragment.app.FragmentActivity, androidx.activity.ComponentActivity, androidx.core.app.ComponentActivity, android.app.Activity\n    onCreate(Bundle savedInstanceState) {\n        super.onCreate(savedInstanceState);\n        setContentView(R.layout.activity_web_view);\n        WebView webView = (WebView) findViewById(R.id.webview);\n        webView.loadUrl(getIntent().getStringExtra("url"));

• Called jadx.get_class_source({"class_name":"oversecured.ovaa.activities.MainActivity"})
  ↳ {"response": "package oversecured.ovaa.activities;\n\nimport android.content.Intent;\nimport android.os.Bundle;\nimport android.view.View;\nimport androidx.appcompat.app.AppCompatActivity;\nimport androidx.core.app.ActivityCompat;\nimport androidx.core.content.ContextCompat;\nimport oversecured.ovaa.R;\nimport oversecured.ovaa.objects.LoginData;\nimport oversecured.ovaa.utils.FileUtils;\nimport oversecured.ovaa.utils.IntentUtils;\nimport oversecured.ovaa.utils.LoginUtils;\nimport oversecured.ovaa.utils.WeakCrypto;\n\n/* loaded from: classes8.dex */\npublic class MainActivity extends AppCompatActivity {\n    private static final int PERMISSIONS_CODE = 1002;\n    private static final int PICK_CODE = 1001;\n    private LoginUtils loginUtils;\n\n    @Override // androidx.fragment.app.FragmentActivity, and...

• Called jadx.get_class_source({"class_name":"oversecured.ovaa.network.RetrofitInstance"})
  ↳ {"response": "package oversecured.ovaa.network;\n\nimport retrofit2.Retrofit;\nimport retrofit2.converter.gson.GsonConverterFactory;\n\n/* loaded from: classes4.dex */\npublic class RetrofitInstance {\n    private static final String BASE_URL = "http://example.com/api/v1/";\n    private static Retrofit retrofit;\n\n    public static Retrofit getInstance() {\n        if (retrofit == null) {\n            retrofit = new Retrofit.Builder().baseUrl(BASE_URL).addConverterFactory(GsonConverterFactory.create()).build();\n        }\n        return retrofit;\n    }\n}"

• Called jadx.get_class_source({"class_name":"oversecured.ovaa.network.LoginService"})
  ↳ {"response": "package oversecured.ovaa.network;\n\nimport oversecured.ovaa.objects.LoginData;\nimport retrofit2.Call;\nimport retrofit2.http.Body;\nimport retrofit2.http.POST;\nimport retrofit2.http.Url;\n\n/* loaded from: classes4.dex */\npublic interface LoginService {\n    @POST\n    Call<Void> login(@Url String url, @Body LoginData loginData);\n}"

• Working (Ⓢ = esc to interrupt)

```

Εικόνα 39: Ανάκτηση των implementations των components και network helpers που προέκυψαν από την αρχική απογραφή.

Τα σημεία που προέκυψαν από την απογραφή εξετάστηκαν στη συνέχεια μέσα στο πραγματικό application-owned code, ώστε να αποσαφηνιστεί ο ρόλος τους και ο τρόπος με τον οποίο συνδέονται μεταξύ τους. Στην ΟΥΑΑ, η εξέταση classes όπως οι DeepLinkActivity και WebViewActivity, μαζί με τον σχετικό network code, επέτρεψε στον agent να ακολουθήσει την επεξεργασία externally controlled δεδομένων και να ανακατασκευάσει τις πρώτες πιθανές διαδρομές προς WebView και authenticated network functionality. Με αυτό το βήμα, τα αρχικά API και component observations άρχισαν να μετατρέπονται σε συγκεκριμένες αλυσίδες εκτέλεσης που μπορούσαν πλέον να αξιολογηθούν ως σύνολο.

- The first critical paths are already visible: the deeplink handler, the WebView sink, and an authenticated @Url login flow. I'm pulling the remaining app-owned classes now so I can validate reachability and collect at least five end-to-end issues rather than stopping at the obvious ones.
- Called `jadx.get_class_source({"class_name": "oversecured.ovaa.activities.EntranceActivity"})`

```
{ "response": "package oversecured.ovaa.activities;\n\nimport android.content.Intent;\nimport android.os.Bundle;\nimport androidx.appcompat.app.AppCompatActivity;\nimport androidx.fragment.app.FragmentActivity;\nimport androidx.activity.ComponentActivity;\nimport androidx.core.app.ComponentActivity;\nimport androidx.appcompat.app.AppCompatActivity;\nimport androidx.core.app.Activity;\n\nprotected void onCreate(Bundle savedInstanceState) {\n    super.onCreate(savedInstanceState);\n    if (LoginUtils.getInstance(this).isLoggedIn()) {\n        startActivity(new Intent(\"oversecured.ovaa.action.ACTIVITY_MAIN\"));\n    } else {\n        startActivity(new Intent(\"oversecured.ovaa.action.LOGIN\"));\n    }\n    fin...
}
```
- Called `jadx.get_class_source({"class_name": "oversecured.ovaa.activities.LoginActivity"})`

```
{ "response": "package oversecured.ovaa.activities;\n\nimport android.content.Intent;\nimport android.os.Bundle;\nimport android.text.TextUtils;\nimport android.util.Log;\nimport android.view.View;\nimport android.widget.TextView;\nimport android.widget.Toast;\nimport androidx.appcompat.app.AppCompatActivity;\nimport oversecured.ovaa.R;\nimport oversecured.ovaa.network.LoginService;\nimport oversecured.ovaa.network.RetrofitInstance;\nimport oversecured.ovaa.objects.LoginData;\nimport oversecured.ovaa.utils.LoginUtils;\nimport retrofit2.Call;\nimport retrofit2.Callback;\nimport retrofit2.Response;\n\n/* loaded from: classes8.dex */\npublic class LoginActivity extends AppCompatActivity {\n    public static final String INTENT_REDIRECT_KEY = \"redirect_intent\";\n    private LoginUtils loginUtils;\n    ...
}
```
- Called `jadx.get_class_source({"class_name": "oversecured.ovaa.providers.CredentialsProvider"})`

```
{ "response": "package oversecured.ovaa.providers;\n\nimport android.content.ContentValues;\nimport android.database.Cursor;\nimport android.database.MatrixCursor;\nimport android.net.Uri;\nimport androidx.core.app.NotificationCompat;\nimport oversecured.ovaa.objects.LoginData;\nimport oversecured.ovaa.utils.LoginUtils;\n\n/* loaded from: classes3.dex */\npublic class CredentialsProvider extends ContentProvider {\n    @Override // android.content.ContentProvider\n    public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {\n        LoginData loginData = LoginUtils.getInstance(getContext()).getLoginData();\n        MatrixCursor cursor = new MatrixCursor(new String[] {NotificationCompat.CATEGORY_EMAIL, \"passwo...
}
```
- Called `jadx.get_class_source({"class_name": "oversecured.ovaa.providers.TheftOverwriteProvider"})`

```
{ "response": "package oversecured.ovaa.providers;\n\nimport android.content.ContentProvider;\nimport android.content.ContentValues;\nimport android.database.Cursor;\nimport android.net.Uri;\nimport android.os.Environment;\nimport android.os.ParcelFileDescriptor;\nimport java.io.File;\nimport java.io.FileNotFoundException;\n\n/* loaded from: classes3.dex */\npublic class TheftOverwriteProvider extends ContentProvider {\n    @Override // android.content.ContentProvider\n    public int delete(Uri uri, String selection, String[] selectionArgs) {\n        return 0;\n    }\n    @Override // android.content.ContentProvider\n    public String getType(Uri uri) {\n        return null;\n    }\n    @Override // android.content.ContentProvider\n    public Uri insert(Uri uri, ContentValues values) {\n        re...
}
```
- Calling `jadx.get_class_source({"class_name": "oversecured.ovaa.services.InsecureLoggerService"})`

Εικόνα 40: Συσχέτιση των αρχικών static observations σε συγκεκριμένες διαδρομές προς διερεύνηση.

Από τη συσχέτιση αυτή διαμορφώθηκαν οι πρώτες ουσιαστικές στατικές υποθέσεις, κυρίως γύρω από WebView και authenticated-network διαδρομές. Η εξέταση συνεχίστηκε και στα υπόλοιπα application-owned components, ώστε οι candidates που είχαν ήδη εμφανιστεί να τοποθετηθούν μέσα στη συνολική εικόνα του attack surface και να μην καθορίσουν πρόωρα την κατεύθυνση ολόκληρης της αξιολόγησης. Όταν ολοκληρώθηκε αυτή η αρχική κάλυψη, οι διαδρομές για τις οποίες παρέμεναν ουσιαστικά ερωτήματα πέρασαν στη δυναμική φάση.

Μία από τις διαδρομές που εξετάστηκαν αφορούσε *authenticated network functionality*, όπου η στατική ανάλυση είχε δείξει ότι μια εξωτερικά επηρεαζόμενη τιμή μπορούσε να χρησιμοποιηθεί ως δυναμικό `Retrofit @Url`. Το σημείο που χρειαζόταν περαιτέρω διερεύνηση αφορούσε τον τελικό προορισμό της επικοινωνίας και τα δεδομένα που μεταδίδονταν κατά την πραγματική εκτέλεση, γι' αυτό ο agent δημιούργησε ένα στοχευμένο **OkHttp hook** για την καταγραφή του αντίστοιχου request. Το instrumentation προέκυψε έτσι απευθείας από το ερώτημα που είχε αφήσει ανοιχτό η στατική ανάλυση και παρείχε το runtime evidence που χρειαζόταν για την επόμενη αξιολόγηση της διαδρομής.

Εικόνα 43: Δυναμική επαλήθευση της WebView διαδρομής.

Η δεύτερη αντιπροσωπευτική διαδρομή αφορούσε το WebView, για το οποίο είχε ήδη εντοπιστεί στατικά σύνδεση με δεδομένα που προέρχονταν από Deep Link. Κατά τη δυναμική δοκιμή, μια ελεγχόμενη URL παραδόθηκε μέσω του πραγματικού Android entry point και παρακολουθήθηκε μέσω του **Medusa** μέχρι να εμφανιστεί ως προορισμός φόρτωσης στο WebView. Η παρατήρηση αυτή επιβεβαίωσε την εκτέλεση της διαδρομής που είχε προηγουμένως ανακατασκευαστεί στο JADX και παρείχε τη βάση για την περαιτέρω εξέταση της επίπτωσης της συγκεκριμένης συμπεριφοράς.

```
• Ran adb shell content query --uri content://oversecured.ovaa.creds_provider/
  ↳ Error while accessing provider:oversecured.ovaa.creds_provider
    java.lang.SecurityException: Permission Denial: opening provider oversecured.ovaa.providers.CredentialsProvider from (null) (pid=14436, uid=2000) that is not
    ... +15 lines (ctrl + t to view transcript)
      at com.android.server.am.ActivityManagerService.getContentProviderExternal(ActivityManagerService.java:6708)

• The credential provider path is disproved on this installed build because the provider is not exported. I'm treating that as a closed lead and shifting to the next
  reachable exported sink instead of overstating it.

• Called medusa-android.tail_output({"since_line":194,"lines":200})
  ↳ {"output_path": "/Users/malamatenia/Desktop/android_findings/top-android-findings/oversecured.ovaa-assessment-20260729-1948/medusa.log", "content": "",
    "line_count": 0, "exists": true, "source": "active-session-log", "total_lines": 193, "start_line": 193, "next_line": 193, "cursor": null, "next_cursor":
    {"path": "/Users/malamatenia/Desktop/android_findings/top-android-findings/oversecured.ovaa-assessment-20260729-1948/medusa.log", "line_offset": 193,
    "total_lines": 193, "size": 10243, "mtime": 1785343978.6144466, "mtime_iso": "2026-07-29T19:52:58+0300"}, "cursor_reset": false}

• Running adb shell dumpsys activity top | rg -n 'ACTIVITY oversecured\\.ovaa\\LoginActivity|MainActivity|WebViewActivity'

✓ You approved codex to run adb shell dumpsys activity top | rg -n 'ACTIVITY oversecured\\.ovaa\\LoginActiv... this time

✓ You approved codex to run adb exec-out uiautomator dump /dev/tty | rg 'emailView|passwordView|loginButt... this time

• Ran adb shell dumpsys activity top | rg -n 'ACTIVITY oversecured\\.ovaa\\LoginActivity|MainActivity|WebViewActivity'
  ↳ 628: WorkspaceItemInfo{id=6 type=APP container=# com.android.launcher3.logger.LauncherAtom$ContainerInfo@21a259b
    targetComponent=ComponentInfo{com.phonope.app/com.phonope.app.launch.core.main.ui.MainActivity} screen=1 cell(0,0) span(1,1) minSpan(1,1) rank=0 user=UserHandle{0}
    ... +7 lines (ctrl + t to view transcript)
      852: DecorView@a17878[LoginActivity]
      886: context: oversecured.ovaa.activities.LoginActivity@ac9b205

• Ran adb exec-out uiautomator dump /dev/tty | rg 'emailView|passwordView|loginButton|text="LOGIN"|MainActivity'
  ↳ <?xml version='1.0' encoding='UTF-8' standalone='yes' ?><hierarchy rotation="0"><node index="0" text="" resource-id="" class="android.widget.FrameLayout"
    package="oversecured.ovaa" content-desc="" checkable="false" checked="false" clickable="false" enabled="true" focusable="false" focused="false" scrollable="false"
    ... +35 lines (ctrl + t to view transcript)
      content-desc="" checkable="false" checked="false" clickable="false" enabled="true" focusable="false" focused="false" scrollable="false" long-clickable="false"
      password="false" selected="false" bounds="[0,2274][1080,2340]" /></node></hierarchy>UI hierarchy dumped to: /dev/tty

• The app is still on LoginActivity, so the previous text injection did not submit cleanly. I'm re-reading the live UI and will use a more reliable form-entry sequence
  before I continue the credential-exfiltration proof.

• Working (ls • esc to interrupt)
```

Εικόνα 44 : Παράδειγμα τεκμηριωμένης απόρριψης ενός lead.

Η ίδια διαδικασία επέτρεψε και το κλείσιμο υποθέσεων που δεν μπορούσαν να οδηγήσουν σε εκμεταλλεύσιμη διαδρομή. Στην περίπτωση του CredentialsProvider, η στατική ανάλυση είχε εντοπίσει ιδιαίτερα ευαίσθητη λειτουργικότητα, καθώς ο provider μπορούσε να επιστρέφει αποθηκευμένα credentials. Η εξέταση της πραγματικής προσβασιμότητας έδειξε όμως ότι ο provider δεν ήταν exported στο συγκεκριμένο build και ότι η εξωτερική πρόσβαση σταματούσε στο Android framework. Η υπόθεση καταγράφηκε επομένως ως **disproved path** με συγκεκριμένο platform-enforced boundary, παρέχοντας σαφή τεχνική αιτιολόγηση για το κλείσιμό της.

```
• I have the five confirmed paths now, plus one disproved lead. I'm extracting the exact evidence lines from medusa.log and the ADB observations so the final answer can
  stay concise without becoming hand-wavy.

• Explored
  ↳ Read medusa.log
    Search OVAA Hijack|PickerActivity|ResolverActivity|LoginUtils.getLoginUri|OVAA_HTTP|redirect=Intent|Loading URL:|InsecureLoggerService|getDumpFile|Data
    URI:oversecured://ovaa/(webview|login|logout) in medusa.log

• I'm writing the assessment artifacts now: the state log with confirmed/disproved paths, and a concise report under the dedicated assessment directory. After that I'll
  give you the five findings directly with the evidence anchors.

• Working (28s • esc to interrupt)
```

Εικόνα 45: Τελική επανασυσχέτιση των runtime artifacts με τις διαδρομές που πρόκειται να αναφερθούν ή να κλείσουν.

Μετά την ολοκλήρωση των επιμέρους ελέγχων, ο agent επέστρεψε στα artifacts που είχαν παραχθεί κατά τη δυναμική ανάλυση και συνέδεσε κάθε επιβεβαιωμένη ή απορριφθείσα διαδρομή με τα observations που την υποστήριζαν. Τα runtime logs και τα υπόλοιπα τεχνικά τεκμήρια οργανώθηκαν έτσι γύρω από τις αντίστοιχες υποθέσεις, διατηρώντας σαφή σύνδεση ανάμεσα στον αρχικό candidate, στον έλεγχο που πραγματοποιήθηκε και στο τελικό συμπέρασμα. Η επανασυσχέτιση αυτή αποτέλεσε το τελευταίο βήμα πριν από τη σύνθεση των αποτελεσμάτων σε αναφορά.

```
oversecured.ovaa-assessment-20260729-1948/oversecured.ovaa.md (+50 -0)
1 # Security Assessment Report for 'oversecured.ovaa'
2 +
3 +- **Assessment date:** 2026-07-29
4 +- **Application version:** 1.0
5 +- **Assessment scope:** Static and dynamic assessment of app-exported surfaces, deeplinks, WebView behavior, intent handling, and service misuse on the installed debug build.
6 +
7 +## Summary of Findings
8 +
9 +- **Finding 1: Deeplinked WebView takeover via weak host validation**
10 + - **Attack vector:** Single-click deeplink
11 + - **Impact:** Attacker-controlled URL loads inside the app WebView with JavaScript enabled
12 + - **Description:** 'DeepLinkActivity' accepts '/webview?url=' when 'Uri.parse(url).getHost().endsWith("example.com")'. 'https://attackerexample.com' passed the check and was loaded by 'WebViewActivity'.
13 +
14 +- **Finding 2: Deeplink-controlled credential exfiltration through Retrofit '@Url' **
15 + - **Attack vector:** Single-click deeplink followed by victim login
16 + - **Impact:** Victim credentials are POSTed to an attacker-controlled endpoint
17 + - **Description:** '/login?url=' overwrites the stored login URL. The next login sent JSON credentials to 'http://127.0.0.1:31337/pwn', confirmed in Medusa.
18 +
19 +- **Finding 3: Exported 'LoginActivity' intent redirection**
20 + - **Attack vector:** Malicious app on device
21 + - **Impact:** Attacker-controlled nested intent is launched in the victim app context after login state check
22 + - **Description:** 'LoginActivity.onLoginFinished()' trusts the Parcelable extra 'redirect_intent'. The installed helper app supplied a nested intent that opened 'WebViewActivity' with an attacker URL.
23 +
24 +- **Finding 4: Exported logger service arbitrary file write**
25 + - **Attack vector:** Malicious app on device
26 + - **Impact:** External caller can dump app/device logcat into attacker-chosen file paths writable by the app
27 + - **Description:** 'InsecureLoggerService' accepts a file path in 'oversecured.ovaa.extra.file' and writes 'logcat -d' output there without validation.
28 +
29 +- **Finding 5: Sensitive picker flow hijackable by malicious 'image/*' handler**
30 + - **Attack vector:** Malicious app on device
31 + - **Impact:** Attacker-controlled picker can intercept a sensitive file-selection flow and return attacker-controlled content
32 + - **Description:** 'MainActivity' launches implicit 'ACTION_PICK image/*' for the file-theft workflow. The runtime resolver offered the malicious app 'OVAA Hijack', and activity history confirmed 'poc.ovaa.hijack.PickerActivity' launched.
33 +
34 +## Summary of Leads
35 +
36 +- **Lead 1: 'CredentialsProvider' leak**
37 + - **Description:** The provider code returns stored email/password.
38 + - **Evidence:** Dynamic access from shell failed with 'SecurityException' because the provider is not exported in this build.
39 +
40 +- **Lead 2: Implicit 'oversecured.ovaa.action.WEBVIEW' hijack from 'MainActivity' **
41 + - **Description:** A malicious app is registered for the action.
42 + - **Evidence:** On this device, the tested launch path resolved to the app's own 'WebViewActivity', so no confirmed external hijack was observed.
43 +
44 +## Appendix: Key Runtime Evidence
45 +
46 +- 'medusa.log' lines 126-171: '/webview' deeplink -> attacker URL loaded
47 +- 'medusa.log' lines 183-199: '/login' deeplink -> attacker POST destination and credential body
48 +- 'medusa.log' lines 209-250: exported 'LoginActivity' redirect -> attacker-supplied 'WebViewActivity' intent
49 +- 'adb shell run-as oversecured.ovaa head -n 5 files/ovaa-dump.txt': confirms exported logger service wrote log output to attacker-chosen path
50 +- 'adb shell dumpsys activity activities | rg 'poc\\ovaa\\.hijack|PickerActivity': confirms malicious picker activity launch

• Working (SIs • esc to interrupt)
```

Εικόνα 46: Παράδειγμα της τελικής αναφοράς αξιολόγησης.

Στο τελικό στάδιο, τα αποτελέσματα οργανώθηκαν σε δομημένες αναφορές που διατηρούν τη σύνδεση κάθε περίπτωσης με τα τεκμήρια της διερεύνησης. Για τα επιβεβαιωμένα findings της OVAA

καταγράφηκαν η διαδρομή ενεργοποίησης, η συμπεριφορά που παρατηρήθηκε κατά την εκτέλεση, η επίπτωση στην ασφάλεια και τα runtime evidence που υποστήριζαν το συμπέρασμα, ενώ οι υπόλοιπες υποθέσεις διατηρήθηκαν με την τελική τους κατάσταση ως leads ή disproved paths. Η μορφή αυτή επιτρέπει στο τελικό αποτέλεσμα να αποτυπώνει τόσο τις ευπάθειες που επιβεβαιώθηκαν όσο και την έκταση της ανάλυσης που πραγματοποιήθηκε για να καταλήξει σε αυτές.

3.5 Σύνοψη

Στο παρόν κεφάλαιο παρουσιάστηκε η προτεινόμενη προσέγγιση για την αυτοματοποίηση της αξιολόγησης ασφάλειας Android εφαρμογών με τη χρήση LLM-based agents και αναλύθηκαν οι βασικές επιλογές που διαμόρφωσαν τον σχεδιασμό της. Η μεθοδολογία συνδυάζει στατική και δυναμική ανάλυση μέσα σε μια κοινή διαδικασία διερεύνησης, ενώ η επιλογή ενός προεκπαιδευμένου LLM σε agentic περιβάλλον επιτρέπει τη συσχέτιση πληροφοριών που προέρχονται από διαφορετικά εργαλεία και στάδια της αξιολόγησης.

Κεντρική αρχή της προσέγγισης είναι ότι τα συμπεράσματα του μοντέλου πρέπει να στηρίζονται σε τεχνικά τεκμήρια που προκύπτουν από τον κώδικα και την πραγματική εκτέλεση της εφαρμογής. Η αρχιτεκτονική οργανώθηκε με βάση αυτή την απαίτηση, διατηρώντας παράλληλα την κατάσταση και την πρόοδο της διερεύνησης εκτός του άμεσου context του LLM. Η εφαρμογή της μεθοδολογίας στην OVAA έδειξε τον τρόπο με τον οποίο τα επιμέρους στοιχεία του συστήματος συνεργάζονται σε μια ολοκληρωμένη αξιολόγηση και οδήγησε στη μετάβαση από τη θεωρητική περιγραφή της προσέγγισης στην πρακτική εξέτασή της, η οποία παρουσιάζεται στα επόμενα κεφάλαια.

4. ΑΠΟΤΕΛΕΣΜΑΤΑ ΚΑΙ ΣΥΜΠΕΡΑΣΜΑΤΑ

4.1 Εφαρμογή της μεθοδολογίας και αποτελέσματα

Η αξιολόγηση της προτεινόμενης προσέγγισης πραγματοποιήθηκε αρχικά στην OVAA (Oversecured Vulnerable Android App) [179], η οποία χρησιμοποιήθηκε ως δημόσια αναπαραγώγιμο περιβάλλον αναφοράς. Η εφαρμογή περιλαμβάνει σκόπιμα ευάλωτες υλοποιήσεις και επέτρεψε την εξέταση της συμπεριφοράς του συστήματος σε περιπτώσεις όπου ο κώδικας, οι διαδρομές εκτέλεσης και τα παραγόμενα runtime artifacts μπορούσαν να παρουσιαστούν χωρίς περιορισμούς δημοσιοποίησης.

Κατά την αξιολόγηση της OVAA προέκυψαν τόσο διαδρομές που επιβεβαιώθηκαν δυναμικά όσο και περιπτώσεις που έκλεισαν μετά την αναγνώριση συγκεκριμένου Android security boundary. Στις επιβεβαιωμένες περιπτώσεις, τα findings μπορούσαν να συνδεθούν με την αντίστοιχη εξωτερικά ελεγχόμενη είσοδο, τη security-sensitive λειτουργία και τα runtime evidence που τεκμηριώναν την πραγματική συμπεριφορά της εφαρμογής. Παράλληλα, η εξέταση candidates όπως ο CredentialsProvider έδειξε ότι στατικά ενδιαφέρουσες περιπτώσεις μπορούσαν να απορριφθούν όταν η πραγματική προσβασιμότητα της διαδρομής περιοριζόταν από το Android framework. Τα αποτελέσματα αυτά έδωσαν ένα πλήρως παρατηρήσιμο παράδειγμα της λειτουργίας της προτεινόμενης προσέγγισης, από τον αρχικό εντοπισμό ενός candidate μέχρι την τελική του κατάσταση.

Πέρα από την αξιολόγηση στην OVAA, η προτεινόμενη προσέγγιση εφαρμόστηκε και σε πραγματικές Android εφαρμογές στο πλαίσιο ιδιωτικών vulnerability disclosure και bug bounty programs. Συνολικά εξετάστηκαν 12 εφαρμογές, από τις οποίες προέκυψαν 10 security findings που ικανοποίησαν τα κριτήρια τεκμηρίωσης της μεθοδολογίας. Αυτά υποβλήθηκαν στα αντίστοιχα προγράμματα και 8 είχαν γίνει triaged ή accepted κατά τον χρόνο συγγραφής της εργασίας. Οι αριθμοί αυτοί παρουσιάζονται συγκεντρωτικά, καθώς οι όροι των αντίστοιχων προγραμμάτων δεν επιτρέπουν τη δημοσιοποίηση της ταυτότητας των εφαρμογών, των Proofs of Concept και των τεχνικών artifacts που συνδέονται με τις συγκεκριμένες περιπτώσεις.

Η εφαρμογή της μεθοδολογίας σε πραγματικά targets παρείχε επίσης διαφορετικές συνθήκες από εκείνες της OVAA, καθώς οι εφαρμογές αυτές περιείχαν μεγαλύτερα codebases, third-party dependencies και λειτουργικές διαδρομές που δεν είχαν σχεδιαστεί για εκπαιδευτικούς ή ερευνητικούς σκοπούς. Τα αποτελέσματα αυτών των αξιολογήσεων χρησιμοποιούνται στην παρούσα εργασία ως συμπληρωματική ένδειξη ότι η αρχιτεκτονική μπορεί να εφαρμοστεί και εκτός ελεγχόμενου benchmark, ενώ η OVAA

παραμένει η περίπτωση για την οποία μπορεί να παρουσιαστεί πλήρως το evidence και να αναπαραχθεί ανεξάρτητα η διαδικασία.

Με βάση τα διαθέσιμα αποτελέσματα, η αξιολόγηση υποστηρίζει κυρίως τη λειτουργικότητα της προτεινόμενης αρχιτεκτονικής ως διαδικασίας vulnerability research. Η εργασία δεν επιχειρεί να υπολογίσει detection rate ή να συγκρίνει ποσοτικά την απόδοση του agent με άλλα συστήματα, καθώς κάτι τέτοιο θα απαιτούσε μεγαλύτερο σύνολο εφαρμογών με γνωστό και πλήρως διαθέσιμο ground truth. Τα αποτελέσματα εξετάζονται επομένως ως προς την ικανότητα της προσέγγισης να οδηγεί από αρχικά candidates σε τεκμηριωμένα τελικά συμπεράσματα και να λειτουργεί τόσο σε ελεγχόμενο όσο και σε πραγματικό περιβάλλον αξιολόγησης.

4.2 Συμβολή της παρούσας εργασίας

Η κύρια συνεισφορά της παρούσας εργασίας αφορά τον σχεδιασμό και την υλοποίηση μιας agentic προσέγγισης για την αξιολόγηση ασφάλειας Android εφαρμογών, στην οποία το γλωσσικό μοντέλο χρησιμοποιείται για τον συντονισμό μιας πολυσταδιακής διαδικασίας διερεύνησης. Η προσέγγιση αυτή επιτρέπει τη συσχέτιση πληροφοριών που προέρχονται από διαφορετικά σημεία της εφαρμογής και από διαφορετικές φάσεις της ανάλυσης, ώστε η αξιολόγηση να μπορεί να ακολουθεί διαδρομές που εκτείνονται σε περισσότερα components ή επίπεδα υλοποίησης.

Στο πλαίσιο αυτό αναπτύχθηκε το Android Client-Side Assessment Skill, το οποίο οργανώνει τη μεθοδολογία σε γενικούς κανόνες αξιολόγησης και σε εξειδικευμένες διαδικασίες για διαφορετικές κατηγορίες ευπαθειών. Η modular αυτή δομή επιτρέπει την προσθήκη νέων vulnerability classes και την προσαρμογή της ανάλυσης στο attack surface κάθε εφαρμογής, χωρίς να απαιτείται αντίστοιχη αλλαγή στον βασικό μηχανισμό συντονισμού του agent.

Ένα δεύτερο στοιχείο της συνεισφοράς είναι η σύνδεση της στατικής και της δυναμικής ανάλυσης μέσα στην ίδια ροή αξιολόγησης. Τα ευρήματα από τον κώδικα χρησιμοποιούνται για τη διαμόρφωση συγκεκριμένων hypotheses, ενώ τα runtime observations μπορούν να μεταβάλουν την πορεία της διερεύνησης και να οδηγήσουν σε νέα εξέταση του implementation. Η επιλογή των εργαλείων και του instrumentation γίνεται με βάση το ερώτημα που παραμένει ανοιχτό σε κάθε περίπτωση, επιτρέποντας περισσότερο στοχευμένους ελέγχους και περιορίζοντας την ανάγκη για γενική εξερεύνηση της εφαρμογής.

Για τη διατήρηση της συνέχειας μιας τέτοιας πολυσταδιακής διαδικασίας, η κατάσταση της αξιολόγησης αποθηκεύεται εξωτερικά και χρησιμοποιείται για την παρακολούθηση των hypotheses, των tests που έχουν πραγματοποιηθεί, των διαθέσιμων evidence και των διαδρομών που παραμένουν ανοιχτές. Η επιλογή αυτή

επιτρέπει στον agent να διατηρεί μια σταθερή εικόνα της προόδου της ανάλυσης ακόμη και όταν η διερεύνηση εκτείνεται σε μεγάλο αριθμό ενεργειών ή απαιτεί επιστροφή σε προηγούμενα σημεία.

Ιδιαίτερη έμφαση δόθηκε επίσης στα κριτήρια με τα οποία μια τεχνικά ενδιαφέρουσα διαδρομή μετατρέπεται σε reportable finding. Η μεθοδολογία απαιτεί τη σύνδεση μιας externally controlled πηγής με security-sensitive λειτουργία, την επιβεβαίωση της διαδρομής κατά την πραγματική εκτέλεση και την ύπαρξη συγκεκριμένης επίπτωσης στην ασφάλεια. Η απαίτηση αυτή περιορίζει την εξαγωγή συμπερασμάτων αποκλειστικά από API matches ή από στατικά patterns και παρέχει κοινό σημείο αναφοράς για την αξιολόγηση διαφορετικών κατηγοριών ευπαθειών.

Η υλοποίηση συμπληρώνεται από έναν ντετερμινιστικό έλεγχο της καταγεγραμμένης κατάστασης, ο οποίος εξετάζει τη συνέπεια μεταξύ των υποθέσεων, των tests, των evidence και των τελικών reports πριν ολοκληρωθεί η αξιολόγηση. Ο validator λειτουργεί ως μηχανισμός ελέγχου της διαδικαστικής συνοχής και βοηθά στον εντοπισμό περιπτώσεων που έχουν παραμείνει ανοιχτές ή δεν διαθέτουν την απαιτούμενη τεκμηρίωση. Η συνεισφορά της εργασίας εντοπίζεται επομένως στον τρόπο με τον οποίο ένα σύγχρονο LLM, τα εργαλεία ανάλυσης και μια σαφώς ορισμένη μεθοδολογία μπορούν να οργανωθούν σε ένα ενιαίο σύστημα για Android vulnerability research.

4.3 Σύνοψη

Στο παρόν κεφάλαιο παρουσιάστηκαν τα αποτελέσματα από την εφαρμογή της προτεινόμενης προσέγγισης και αποτιμήθηκε η βασική συνεισφορά της εργασίας. Συνολικά, η εργασία δείχνει ότι ένα LLM μπορεί να χρησιμοποιηθεί αποτελεσματικά ως επίπεδο συντονισμού σε διαδικασίες Android vulnerability research, όταν η λειτουργία του εντάσσεται σε σαφώς ορισμένη μεθοδολογία και τα τελικά συμπεράσματα στηρίζονται σε επαληθεύσιμα τεχνικά στοιχεία.

5. ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1] S. Samat, “Living in a multi-device world with Android,” Google, May 11, 2022. [Online]. Available: <https://blog.google/products-and-platforms/platforms/android/io22-multideviceworld/>. [Accessed: Aug. 22, 2026].
- [2] Android Open Source Project, “Application Sandbox,” Android Open Source Project, Jul. 13, 2026. [Online]. Available: <https://source.android.com/docs/security/app-sandbox>. [Accessed: Aug. 22, 2026].
- [3] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, “Analyzing inter-application communication in Android,” in Proc. 9th Int. Conf. Mobile Systems, Applications, and Services (MobiSys ’11), Bethesda, MD, USA, 2011, pp. 239–252, doi: 10.1145/1999995.2000018.
- [4] J. Gao, L. Li, P. Kong, T. F. Bissyandé, and J. Klein, “Understanding the evolution of Android app vulnerabilities,” IEEE Trans. Rel., vol. 70, no. 1, pp. 212–230, Mar. 2021, doi: 10.1109/TR.2019.2956690.
- [5] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, “SWE-bench: Can language models resolve real-world GitHub issues?,” in Proc. Int. Conf. Learn. Representations (ICLR), 2024.
- [6] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, “SWE-agent: Agent-computer interfaces enable automated software engineering,” in Advances in Neural Information Processing Systems, vol. 37, 2024, doi: 10.52202/079017-1601.
- [7] Z. Sheng, Z. Chen, S. Gu, H. Huang, G. Gu, and J. Huang, “LLMs in software security: A survey of vulnerability detection techniques and insights,” ACM Comput. Surv., vol. 58, no. 5, Art. no. 134, pp. 1–35, 2025, doi: 10.1145/3769082.
- [8] X. Zhou, S. Cao, X. Sun, and D. Lo, “Large language model for vulnerability detection and repair: Literature review and the road ahead,” ACM Trans. Softw. Eng. Methodol., vol. 34, no. 5, Art. no. 145, pp. 1–31, 2025, doi: 10.1145/3708522.
- [9] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, “PentestGPT: Evaluating and harnessing large language models for automated penetration testing,” in Proc. 33rd USENIX Security Symp. (USENIX Security ’24), Philadelphia, PA, USA, 2024, pp. 847–864.

- [10] J. Zhu, K. Li, S. Chen, L. Fan, J. Wang, and X. Xie, “A comprehensive study on static application security testing (SAST) tools for Android,” *IEEE Trans. Softw. Eng.*, vol. 50, no. 12, pp. 3385–3402, Dec. 2024, doi: 10.1109/TSE.2024.3488041.
- [11] V.-P. Ranganath and J. Mitra, “Are free Android app security analysis tools effective in detecting known vulnerabilities?,” *Empir. Softw. Eng.*, vol. 25, no. 1, pp. 178–219, 2020, doi: 10.1007/s10664-019-09749-y.
- [12] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, “FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps,” in *Proc. 35th ACM SIGPLAN Conf. Programming Language Design and Implementation (PLDI ’14)*, 2014, pp. 259–269, doi: 10.1145/2594291.2594299.
- [13] T. Su, J. Wang, and Z. Su, “Benchmarking automated GUI testing for Android against real-world bugs,” in *Proc. 29th ACM Joint Eur. Softw. Eng. Conf. Symp. Foundations Softw. Eng. (ESEC/FSE ’21)*, Athens, Greece, 2021, pp. 119–130, doi: 10.1145/3468264.3468620.
- [14] OWASP Foundation, “Testing Tools,” OWASP Mobile Application Security Testing Guide (MASTG). [Online]. Available: <https://mas.owasp.org/MASTG/tools/>. [Accessed: Aug. 22, 2026].
- [15] T. Sutter, “FirmwareDroid: Security analysis of the Android firmware ecosystem,” arXiv:2112.08520, 2021.
- [16] GitHub, “Analyzing data flow in Java and Kotlin,” CodeQL Documentation. [Online]. Available: <https://codeql.github.com/docs/codeql-language-guides/analyzing-data-flow-in-java/>. [Accessed: Aug. 22, 2026].
- [17] L. Li, T. F. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, and Y. Le Traon, “Static analysis of Android apps: A systematic literature review,” *Inf. Softw. Technol.*, vol. 88, pp. 67–95, Aug. 2017, doi: 10.1016/j.infsof.2017.04.001.
- [18] OWASP Foundation, “Mobile Application Security Testing,” OWASP Mobile Application Security Testing Guide (MASTG). [Online]. Available: <https://mas.owasp.org/MASTG/0x04b-Mobile-App-Security-Testing/>. [Accessed: Aug. 22, 2026].
- [19] Frida, “Modes of Operation,” Frida Documentation. [Online]. Available: <https://frida.re/docs/modes/>. [Accessed: Aug. 22, 2026].

- [20] SensePost, “objection – Runtime Mobile Exploration,” GitHub. [Online]. Available: <https://github.com/sensepost/objection>. [Accessed: Aug. 22, 2026].
- [21] MobSF, “Mobile Security Framework (MobSF),” GitHub. [Online]. Available: <https://github.com/MobSF/Mobile-Security-Framework-MobSF>. [Accessed: Aug. 22, 2026].
- [22] Android Open Source Project, “Architecture overview,” Android Open Source Project, Jun. 17, 2026. [Online]. Available: <https://source.android.com/docs/core/architecture>. [Accessed: Aug. 23, 2026].
- [23] Android Open Source Project, “Android runtime and Dalvik,” Android Open Source Project, Jul. 14, 2026. [Online]. Available: <https://source.android.com/docs/core/runtime>. [Accessed: Aug. 23, 2026].
- [24] Android Developers, “Concepts,” Android NDK. [Online]. Available: <https://developer.android.com/ndk/guides/concepts>. [Accessed: Aug. 23, 2026].
- [25] Android Open Source Project, “About the Zygote processes,” Android Open Source Project, Jun. 17, 2026. [Online]. Available: <https://source.android.com/docs/core/runtime/zygote>. [Accessed: Aug. 23, 2026].
- [26] Android Open Source Project, “Security enhancements,” Android Open Source Project, Jul. 13, 2026. [Online]. Available: <https://source.android.com/docs/security/enhancements>. [Accessed: Aug. 23, 2026].
- [27] Android Developers, “Behavior changes: Apps targeting Android 12,” Android Developers, Aug. 7, 2026. [Online]. Available: <https://developer.android.com/about/versions/12/behavior-changes-12>. [Accessed: Aug. 23, 2026].
- [28] Android Developers, “Processes and threads overview,” Android Developers, Aug. 7, 2026. [Online]. Available: <https://developer.android.com/guide/components/processes-and-threads>. [Accessed: Aug. 23, 2026].
- [29] Android Developers, “Data and file storage overview,” Android Developers, Aug. 7, 2026. [Online]. Available: <https://developer.android.com/training/data-storage>. [Accessed: Aug. 23, 2026].
- [30] Android Open Source Project, “Security-Enhanced Linux in Android,” Android Open Source Project, Aug. 26, 2024. [Online]. Available: <https://source.android.com/docs/security/features/selinux>. [Accessed: Aug. 23, 2026].

- [31] P. Lawrence, “Seccomp filter in Android O,” Android Developers Blog, Jul. 20, 2017. [Online]. Available: <https://android-developers.googleblog.com/2017/07/seccomp-filter-in-android-o.html>. [Accessed: Aug. 23, 2026].
- [32] Android Developers, “Permissions on Android,” Android Developers. [Online]. Available: <https://developer.android.com/guide/topics/permissions/overview>. [Accessed: Aug. 23, 2026].
- [33] Android Developers, “Android 6.0 Changes,” Android Developers. [Online]. Available: <https://developer.android.com/about/versions/marshmallow/android-6.0-changes>. [Accessed: Aug. 23, 2026].
- [34] Android Developers, “Android 7.0 Behavior Changes,” Android Developers. [Online]. Available: <https://developer.android.com/about/versions/nougat/android-7.0-changes>. [Accessed: Aug. 23, 2026].
- [35] Android Developers, “Storage updates in Android 11,” Android Developers. [Online]. Available: <https://developer.android.com/about/versions/11/privacy/storage>. [Accessed: Aug. 23, 2026].
- [36] Android Developers, “android,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/android-exported>. [Accessed: Aug. 23, 2026].
- [37] Android Developers, “Pending intents,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/pending-intent>. [Accessed: Aug. 23, 2026].
- [38] Android Developers, “Intent redirection,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/intent-redirection>. [Accessed: Aug. 23, 2026].
- [39] Android Developers, “Behavior changes: all apps – Android 16,” Android Developers. [Online]. Available: <https://developer.android.com/about/versions/16/behavior-changes-all>. [Accessed: Aug. 23, 2026].
- [40] Android Developers, “Broadcasts overview,” Android Developers. [Online]. Available: <https://developer.android.com/develop/background-work/background-tasks/broadcasts>. [Accessed: Aug. 23, 2026].
- [41] Android Developers, “Features and APIs overview – Android 13,” Android Developers. [Online]. Available: <https://developer.android.com/about/versions/13/features>. [Accessed: Aug. 23, 2026].

- [42] Android Developers, “Behavior changes: Apps targeting Android 14 or higher,” Android Developers. [Online]. Available: <https://developer.android.com/about/versions/14/behavior-changes-14>. [Accessed: Aug. 23, 2026].
- [43] Android Developers, “Background Execution Limits,” Android Developers. [Online]. Available: <https://developer.android.com/about/versions/oreo/background>. [Accessed: Aug. 23, 2026].
- [44] Android Open Source Project, “Intent.java,” Android Open Source Project – platform/frameworks/base. [Online]. Available: <https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/java/android/content/Intent.java>. [Accessed: Aug. 23, 2026].
- [45] Android Open Source Project, “ActivityStarter.java,” Android Open Source Project – platform/frameworks/base. [Online]. Available: <https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/services/core/java/com/android/server/wm/ActivityStarter.java>. [Accessed: Aug. 23, 2026].
- [46] Android Developers, “<provider>,” Android Developers – App architecture. [Online]. Available: <https://developer.android.com/guide/topics/manifest/provider-element>. [Accessed: Aug. 24, 2026].
- [47] Android Developers, “Content provider basics,” Android Developers. [Online]. Available: <https://developer.android.com/guide/topics/providers/content-provider-basics>. [Accessed: Aug. 24, 2026].
- [48] Android Developers, “About App Links,” Android Developers. [Online]. Available: <https://developer.android.com/training/app-links/about>. [Accessed: Aug. 24, 2026].
- [49] F. Liu, C. Wang, A. Pico, D. Yao, and G. Wang, “Measuring the insecurity of mobile deep links of Android,” in Proc. 26th USENIX Security Symp. (USENIX Security ’17), Vancouver, BC, Canada, 2017, pp. 953–969.
- [50] Android Developers, “Behavior changes: all apps – Android 12,” Android Developers. [Online]. Available: <https://developer.android.com/about/versions/12/behavior-changes-all>. [Accessed: Aug. 24, 2026].
- [51] Android Developers, “Unsafe use of deep links,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/unsafe-use-of-deeplinks>. [Accessed: Aug. 24, 2026].

- [52] Android Developers, “Configure website associations and dynamic rules,” Android Developers. [Online]. Available: <https://developer.android.com/training/app-links/configure-assetlinks>. [Accessed: Aug. 24, 2026].
- [53] Android Developers, “WebSettings,” Android Developers – API Reference. [Online]. Available: <https://developer.android.com/reference/android/webkit/WebSettings>. [Accessed: Aug. 24, 2026].
- [54] Android Developers, “Unsafe Deserialization,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/unsafe-deserialization>. [Accessed: Aug. 24, 2026].
- [55] Android Developers, “Log Info Disclosure,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/log-info-disclosure>. [Accessed: Aug. 24, 2026].
- [56] Android Developers, “Tapjacking,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/tapjacking>. [Accessed: Aug. 24, 2026].
- [57] Android Developers, “Android 8.0 Behavior Changes,” Android Developers. [Online]. Available: <https://developer.android.com/about/versions/oreo/android-8.0-changes>. [Accessed: Aug. 24, 2026].
- [58] Android Developers, “Secure sensitive activities,” Android Developers – Fraud prevention. [Online]. Available: <https://developer.android.com/security/fraud-prevention/activities>. [Accessed: Aug. 24, 2026].
- [59] Android Developers, “Intents and intent filters,” Android Developers – App architecture. [Online]. Available: <https://developer.android.com/guide/components/intents-filters>. [Accessed: Aug. 24, 2026].
- [60] Android Developers, “WebView,” Android Developers – API Reference. [Online]. Available: <https://developer.android.com/reference/android/webkit/WebView>. [Accessed: Aug. 24, 2026].
- [61] Android Developers, “Network security configuration,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/security-config>. [Accessed: Aug. 24, 2026].
- [62] Android Developers, “Permissions updates in Android 11,” Android Developers. [Online]. Available: <https://developer.android.com/about/versions/11/privacy/permissions>. [Accessed: Sep. 17, 2026].
- [63] Android Developers, “Webviews – Unsafe URI Loading,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/unsafe-uri-loading>. [Accessed: Sep. 16, 2026].

- [64] Android Developers, “WebView – Native bridges,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/insecure-webview-native-bridges>. [Accessed: Sep. 16, 2026].
- [65] Oversecured, “Android security checklist: WebView,” Oversecured Blog, Oct. 29, 2021. [Online]. Available: <https://oversecured.com/blog/android-security-checklist-webview>. [Accessed: Sep. 16, 2026].
- [66] Android Developers, “Uri,” Android API Reference, android.net.Uri. [Online]. Available: <https://developer.android.com/reference/android/net/Uri>. [Accessed: Sep. 16, 2026].
- [67] Android Developers, “WebChromeClient,” Android API Reference, android.webkit.WebChromeClient. [Online]. Available: <https://developer.android.com/reference/android/webkit/WebChromeClient>. [Accessed: Sep. 16, 2026].
- [68] Android Developers, “CookieManager,” Android API Reference, android.webkit.CookieManager. [Online]. Available: <https://developer.android.com/reference/android/webkit/CookieManager>. [Accessed: Sep. 17, 2026].
- [69] Android Developers, “Implicit intent hijacking,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/implicit-intent-hijacking>. [Accessed: Sep. 17, 2026].
- [70] Android Developers, “IntentSanitizer,” AndroidX Core API Reference. [Online]. Available: <https://developer.android.com/reference/androidx/core/content/IntentSanitizer>. [Accessed: Sep. 17, 2026].
- [71] Android Developers, “Android intent and component security best practices,” Android Developers – Security. [Online]. Available: <https://developer.android.com/agents/skills/security/android-intent-security/SKILL>. [Accessed: Sep. 17, 2026].
- [72] Android Developers, “Create a deep link for a destination,” Android Developers – Navigation. [Online]. Available: <https://developer.android.com/guide/navigation/design/deep-link>. [Accessed: Sep. 17, 2026].
- [73] A. Pesnyak, “Android Jetpack Navigation: Deep Links Handling Exploitation,” PT SWARM, Positive Technologies, Mar. 20, 2024. [Online]. Available: <https://swarm.ptsecurity.com/android-jetpack-navigation-deep-links-handling-exploitation/>. [Accessed: Sep. 17, 2026].

- [74] Android Developers, “Navigation – Version 2.10.0,” AndroidX Release Notes. [Online]. Available: <https://developer.android.com/jetpack/androidx/releases/navigation>. [Accessed: Sep. 17, 2026].
- [75] Android Developers, “Services overview,” Android Developers – Background work. [Online]. Available: <https://developer.android.com/develop/background-work/services>. [Accessed: Sep. 17, 2026].
- [76] Android Developers, “Security checklist,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/security-tips>. [Accessed: Sep. 17, 2026].
- [77] Android Developers, “Binder,” Android API Reference. [Online]. Available: <https://developer.android.com/reference/android/os/Binder>. [Accessed: Sep. 17, 2026].
- [78] Android Developers, “Insecure broadcast receivers,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/insecure-broadcast-receiver>. [Accessed: Sep. 17, 2026].
- [79] Oversecured, “Interception of Android implicit intents,” Oversecured Blog, Oct. 22, 2020. [Online]. Available: <https://oversecured.com/blog/interception-of-android-implicit-intents>. [Accessed: Sep. 17, 2026].
- [80] Android Developers, “Sticky Broadcasts,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/sticky-broadcast>. [Accessed: Sep. 17, 2026].
- [81] Android Developers, “Permission-based access control to exported components,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/access-control-to-exported-components>. [Accessed: Sep. 17, 2026].
- [82] Oversecured, “Vulnerabilities,” Oversecured. [Online]. Available: <https://app.oversecured.com/vulnerabilities>. [Accessed: Sep. 17, 2026].
- [83] Android Developers, “Android Interface Definition Language (AIDL),” Android Developers – Background work. [Online]. Available: <https://developer.android.com/develop/background-work/services/aidl>. [Accessed: Sep. 17, 2026].
- [84] Android Developers, “<path-permission>,” Android Developers – App architecture. [Online]. Available: <https://developer.android.com/guide/topics/manifest/path-permission-element>. [Accessed: Sep. 13, 2026].

- [85] Android Developers, “Intent,” Android Developers – API Reference. [Online]. Available: <https://developer.android.com/reference/android/content/Intent>. [Accessed: Sep. 13, 2026].
- [86] Android Developers, “ObjectInputStream,” Android API Reference. [Online]. Available: <https://developer.android.com/reference/java/io/ObjectInputStream>. [Accessed: Sep. 17, 2026].
- [87] Android Developers, “Parcel,” Android API Reference. [Online]. Available: <https://developer.android.com/reference/android/os/Parcel>. [Accessed: Sep. 17, 2026].
- [88] Android Developers, “Parcelable,” Android API Reference. [Online]. Available: <https://developer.android.com/reference/android/os/Parcelable>. [Accessed: Sep. 17, 2026].
- [89] M. Y. Mo, “Android Deserialization Vulnerabilities: A Brief history,” GitHub Security Lab, Feb. 2, 2018. [Online]. Available: <https://securitylab.github.com/resources/android-deserialization-vulnerabilities/>. [Accessed: Sep. 17, 2026].
- [90] Oversecured, “Exploiting memory corruption vulnerabilities on Android,” Oversecured Blog, Apr. 30, 2021. [Online]. Available: <https://oversecured.com/blog/exploiting-memory-corruption-vulnerabilities-on-android>. [Accessed: Sep. 17, 2026].
- [91] Android Developers, “Dynamic Code Loading,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/dynamic-code-loading>. [Accessed: Sep. 17, 2026].
- [92] Android Developers, “DexClassLoader,” Android API Reference. [Online]. Available: <https://developer.android.com/reference/dalvik/system/DexClassLoader>. [Accessed: Sep. 17, 2026].
- [93] Android Developers, “Context,” Android API Reference. [Online]. Available: <https://developer.android.com/reference/android/content/Context>. [Accessed: Sep. 17, 2026].
- [94] Android Developers, “createPackageContext,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/create-package-context>. [Accessed: Sep. 17, 2026].
- [95] Android Developers, “Behavior changes: Apps targeting Android 17 or higher,” Android Developers. [Online]. Available: <https://developer.android.com/about/versions/17/behavior-changes-17>. [Accessed: Sep. 17, 2026].

- [96] Android Developers, “JNI tips,” Android NDK. [Online]. Available: <https://developer.android.com/ndk/guides/jni-tips>. [Accessed: Sep. 17, 2026].
- [97] Android Developers, “Hardcoded Cryptographic Secrets,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/hardcoded-cryptographic-secrets>. [Accessed: Sep. 17, 2026].
- [98] Android Developers, “Insecure API usage,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/insecure-api-usage>. [Accessed: Sep. 17, 2026].
- [99] Oversecured, “Use cryptography in mobile apps the right way,” Oversecured Blog, Dec. 15, 2021. [Online]. Available: <https://oversecured.com/blog/use-cryptography-in-mobile-apps-the-right-way>. [Accessed: Sep. 17, 2026].
- [100] Android Developers, “Sensitive Data Stored in External Storage,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/sensitive-data-external-storage>. [Accessed: Sep. 17, 2026].
- [101] Android Developers, “Security recommendations for backups,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/backup-best-practices>. [Accessed: Sep. 17, 2026].
- [102] Skylot, “JADX – Dex to Java Decompiler,” GitHub. [Online]. Available: <https://github.com/skylot/jadx>. [Accessed: Sep. 17, 2026].
- [103] M. Leon, “Hermes as the Default,” React Native, Jul. 8, 2022. [Online]. Available: <https://reactnative.dev/blog/2022/07/08/hermes-as-the-default>. [Accessed: Sep. 17, 2026].
- [104] P1 Security, “hermes-dec: A reverse engineering tool for decompiling and disassembling the React Native Hermes bytecode,” GitHub. [Online]. Available: <https://github.com/P1sec/hermes-dec>. [Accessed: Sep. 17, 2026].
- [105] Hex-Rays, “Decompiler,” IDA Documentation. [Online]. Available: <https://docs.hex-rays.com/user-guide/decompiler>. [Accessed: Sep. 17, 2026].
- [106] Hex-Rays, “Cross-references,” IDA Documentation. [Online]. Available: <https://docs.hex-rays.com/user-guide/user-interface/menu-bar/jump/cross-references>. [Accessed: Sep. 17, 2026].

- [107] National Security Agency, “Introduction to Ghidra,” Ghidra Documentation. [Online]. Available: https://ghidra.re/ghidra_docs/GhidraClass/Beginner/Introduction_to_Ghidra_Student_Guide.html. [Accessed: Sep. 17, 2026].
- [108] Android Developers, “Android Debug Bridge (adb),” Android Developers. [Online]. Available: <https://developer.android.com/tools/adb>. [Accessed: Sep. 17, 2026].
- [109] Android Developers, “Write automated tests with UI Automator,” Android Developers. [Online]. Available: <https://developer.android.com/training/testing/other-components/ui-automator>. [Accessed: Sep. 17, 2026].
- [110] Frida, “Android,” Frida Documentation. [Online]. Available: <https://frida.re/docs/android/>. [Accessed: Sep. 17, 2026].
- [111] Frida, “JavaScript API,” Frida Documentation. [Online]. Available: <https://frida.re/docs/javascript-api/>. [Accessed: Sep. 17, 2026].
- [112] Ch0pin, “Medusa: Mobile Edge-Dynamic Unified Security Analysis,” GitHub. [Online]. Available: <https://github.com/Ch0pin/medusa>. [Accessed: Sep. 17, 2026].
- [113] PortSwigger, “Mobile testing,” Burp Suite Documentation. [Online]. Available: <https://portswigger.net/burp/documentation/desktop/mobile>. [Accessed: Sep. 17, 2026].
- [114] Chrome Developers, “Remote debug Android devices,” Chrome DevTools. [Online]. Available: <https://developer.chrome.com/docs/devtools/remote-debugging/>. [Accessed: Sep. 17, 2026].
- [115] Chrome DevTools, “Chrome DevTools Protocol – Runtime domain,” Chrome DevTools Protocol. [Online]. Available: <https://chromedevtools.github.io/devtools-protocol/tot/Runtime/>. [Accessed: Sep. 17, 2026].
- [116] H. Wang, J. Gong, H. Zhang, and Z. Wang, “AI Agentic Programming: A Survey of Techniques, Challenges, and Opportunities,” arXiv preprint arXiv:2508.11126, 2025.
- [117] Model Context Protocol, “Model Context Protocol Specification – 2026-07-28,” Model Context Protocol Documentation, Jul. 28, 2026. [Online]. Available: <https://modelcontextprotocol.io/specification/2026-07-28>. [Accessed: Sep. 17, 2026].

- [118] Y. Yang, B. Xu, X. Gao, and H. Sun, “Context-Enhanced Vulnerability Detection Based on Large Language Models,” *ACM Trans. Softw. Eng. Methodol.*, vol. 35, no. 8, Art. no. 234, pp. 1–47, 2026, doi: 10.1145/3779222.
- [119] M. Stone, “Reverse Engineering Android Apps – Native Libraries,” *Android App Reverse Engineering* 101. [Online]. Available: https://www.ragingrock.com/AndroidAppRE/reversing_native_libs.html. [Accessed: Sep. 17, 2026].
- [120] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, “Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks,” in *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 2019.
- [121] Z. Li, N. Wang, D. Zou, Y. Li, R. Zhang, S. Xu, C. Zhang, and H. Jin, “On the Effectiveness of Function-Level Vulnerability Detectors for Inter-Procedural Vulnerabilities,” in *Proc. 46th IEEE/ACM Int. Conf. Software Engineering (ICSE ’24)*, Lisbon, Portugal, 2024, pp. 157:1–157:12, doi: 10.1145/3597503.3639218.
- [122] Y. Zhu, A. Kellermann, A. Gupta, P. Li, R. Fang, R. Bindu, and D. Kang, “Teams of LLM Agents can Exploit Zero-Day Vulnerabilities,” in *Proc. 19th Conf. European Chapter of the Association for Computational Linguistics (EACL 2026)*, Volume 1: Long Papers, Rabat, Morocco, 2026, pp. 23–35, doi: 10.18653/v1/2026.eacl-long.2.
- [123] J. Liu, K. Wang, Y. Chen, X. Peng, Z. Chen, L. Zhang, and Y. Lou, “Large Language Model-Based Agents for Software Engineering: A Survey,” *ACM Transactions on Software Engineering and Methodology*, 2026, doi: 10.1145/3796507.
- [124] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, “ReAct: Synergizing Reasoning and Acting in Language Models,” in *Proc. International Conference on Learning Representations (ICLR)*, 2023.
- [125] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, “Toolformer: Language Models Can Teach Themselves to Use Tools,” in *Advances in Neural Information Processing Systems*, vol. 36, 2023, doi: 10.52202/075280-2997.
- [126] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” in *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

- [127] Android Developers, “Improve your app's security,” Android Developers. [Online]. Available: <https://developer.android.com/privacy-and-security/security-best-practices>. [Accessed: Sep. 17, 2026].
- [128] Android Developers, “Manage all files on a storage device,” Android Developers. [Online]. Available: <https://developer.android.com/training/data-storage/manage-all-files>. [Accessed: Sep. 17, 2026].
- [129] Android Developers, “Application fundamentals,” Android Developers – App architecture. [Online]. Available: <https://developer.android.com/guide/components/fundamentals>. [Accessed: Sep. 17, 2026].
- [130] iBotPeaches, “Apktool,” GitHub. [Online]. Available: <https://github.com/iBotPeaches/Apktool>. [Accessed: Sep. 17, 2026].
- [131] PortSwigger, “Burp Repeater,” Burp Suite Documentation. [Online]. Available: <https://portswigger.net/burp/documentation/desktop/tools/repeater>. [Accessed: Sep. 17, 2026].
- [132] K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston, “Retrieval Augmentation Reduces Hallucination in Conversation,” in Findings of the Association for Computational Linguistics: EMNLP 2021, Punta Cana, Dominican Republic, 2021, pp. 3784–3803, doi: 10.18653/v1/2021.findings-emnlp.320.
- [133] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, and T. Liu, “A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions,” ACM Transactions on Information Systems, vol. 43, no. 2, Art. no. 42, pp. 1–55, 2025, doi: 10.1145/3703155.
- [134] Oversecured Inc., “OVAA – Oversecured Vulnerable Android App,” GitHub. [Online]. Available: <https://github.com/oversecured/ovaa>. [Accessed: Sep. 17, 2026].
- [135] Android Developers, “Load in-app content,” Android Developers. [Online]. Available: <https://developer.android.com/develop/ui/views/layout/webapps/load-local-content>. [Accessed: Sep. 17, 2026].
- [136] Android Developers, “WebViews – Unsafe File Inclusion,” Android Developers – Security. [Online]. Available: <https://developer.android.com/privacy-and-security/risks/webview-unsafe-file-inclusion>. [Accessed: Sep. 17, 2026].
- [137] Android Developers, “Activity,” Android API Reference, android.app.Activity. [Online]. Available: <https://developer.android.com/reference/android/app/Activity>. [Accessed: Sep. 17, 2026].