ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Penetration Test σε Android»

Του φοιτητή
Ευθύμιου Κελέσμητου
Αρ. Μητρώου: 154465

Επιβλέπων
Χρήστος Ηλιούδης
Καθηγητής

3 Φεβρουαρίου 2022

Τίτλος Δ.Ε.: Penetration Test σε Android
Κωδικός Δ.Ε.: 21253
Φοιτητής: Ευθύμιος Κελέσμητος
Εισηγητής: Χρήστος Ηλιούδης
Ημερομηνία ανάληψης Δ.Ε.: 03-04-2021
Ημερομηνία περάτωσης Δ.Ε. 03-02-2022

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

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Περίληψη

Το λειτουργικό σύστημα Android είναι το πιο δημοφιλές λειτουργικό σύστημα για κινητές συσκευές στον κόσμο. Απόρροια της μεγάλης χρήσης του λογισμικού και της ανοδικής πορείας του, είναι η δημιουργία και χρήση εφαρμογών του. Για τον λόγο αυτό οι ανησυχίες για την ανάγκη ασφάλισης των εφαρμογών, με σκοπό την διασφάλιση των δεδομένων των χρηστών και των επιχειρήσεων, είναι αυξημένες. Μια τέτοια μεθοδολογία ασφάλισης των εφαρμογών είναι το penetration testing. Βάσει των παραπάνω, αρχικά, στην παρούσα διπλωματική γίνεται μια λεπτομερής αναφορά στη δομή του λογισμικού Android και μελετάται η αρχιτεκτονική και ο τρόπος λειτουργίας των εφαρμογών του. Έπειτα , δίνεται η έννοια του penetration testing και παρουσιάζονται οι διάφοροι τύποι και μέθοδοι του. Στη συνέχεια εξετάζεται η σημαντικότητα του Mobile Application Penetration Testing (MAPT). Για την κατανόηση αυτού του τύπου αναλύεται η μεθοδολογία του και προβάλλονται τα εργαλεία που βοηθούν στην επίτευξή του. Ακόμα, αναφέρονται και επεξηγούνται οι συνήθεις κίνδυνοι και ευπάθειες που αντιμετωπίζουν οι εφαρμογές όπως τους ορίζει ο OWASP. Για την παρουσίαση και τον εντοπισμό μιας τέτοιας ευπάθειας και συγκεκριμένα του reverse engineering, στήνεται το σωστό pentesting lab. Ακολούθως, διεξάγεται ένα MAPT σε 2 Android εφαρμογές του OWASP, ενώ παράλληλα εξηγούνται τα βήματα που ακολουθήθηκαν για τον εντοπισμό της ευπάθειας. Τέλος, παραθέτονται συμπεράσματα γύρω από τη διεξαγωγή του test, καθώς και οι μελλοντικές επεκτάσεις της παρούσας εργασίας.

«Penetration Test on Android»

«Efthymios Kelesmitos»

Abstract

Android is the most popular mobile operating system in the world. As a result, of its huge global market share, developers are creating more and more apps. Security concerns inherent in mobile device use, increase the need for consultation in mobile security. One methodology to identify weaknesses and test the security of a mobile app is through a penetration test. The purpose of this thesis is to demonstrate the meaning of the penetration testing process and its results. At first, we analyze the Android Operating System in order to acknowledge the basics of Android applications. Then, the concept of penetration testing is given and the importance of Mobile Application Penetration Testing (MAPT) is examined. To understand this type of testing, its methodology is analyzed and some tools the help to achieve it are debriefed. Furthermore, the top 10 threats mobiles face more frequently as OWASP announces are presented. Afterwards, we prepare a penetration testing lab with some of the most important tools included and perform a penetration test in 2 Android app challenges from OWASP by describing every step of the process. Finally, we present the conclusions about the penetration test and the future extensions of the present work.

Ευχαριστίες

Με την περάτωση της παρούσας διπλωματικής εργασίας, θα ήθελα να ευχαριστήσω θερμά τον Καθηγητή του Τμήματος Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙΠΑΕ κ. Χρήστο Ηλιούδη, καθώς η παρακολούθηση του μαθήματός του, «Ασφάλεια Πληροφοριακών Συστημάτων» αποτέλεσε το έναυσμα για την ενασχόλησή μου με τον τομέα της κυβερνοασφάλειας. Επιπλέον, τον ευχαριστώ για τις χρήσιμες υποδείξεις και συμβουλές του στην εκπόνηση της παρούσας εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω και την οικογένειά μου για την αμέριστη ηθική συμπαράστασή της κατά τη διάρκεια των σπουδών μου.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

Δ.Ε Διπλωματική Εργασία

ΔΙΠΑΕ Διεθνές Πανεπιστήμιο Ελλάδος

API Application Programming Interface

APK Android Application Package

MAPT Mobile Application Penetration Testing

MAPTM Mobile Application Penetration Testing Methodology

MSTG Mobile Security Testing Guide

OWASP Open Web Application Security Project

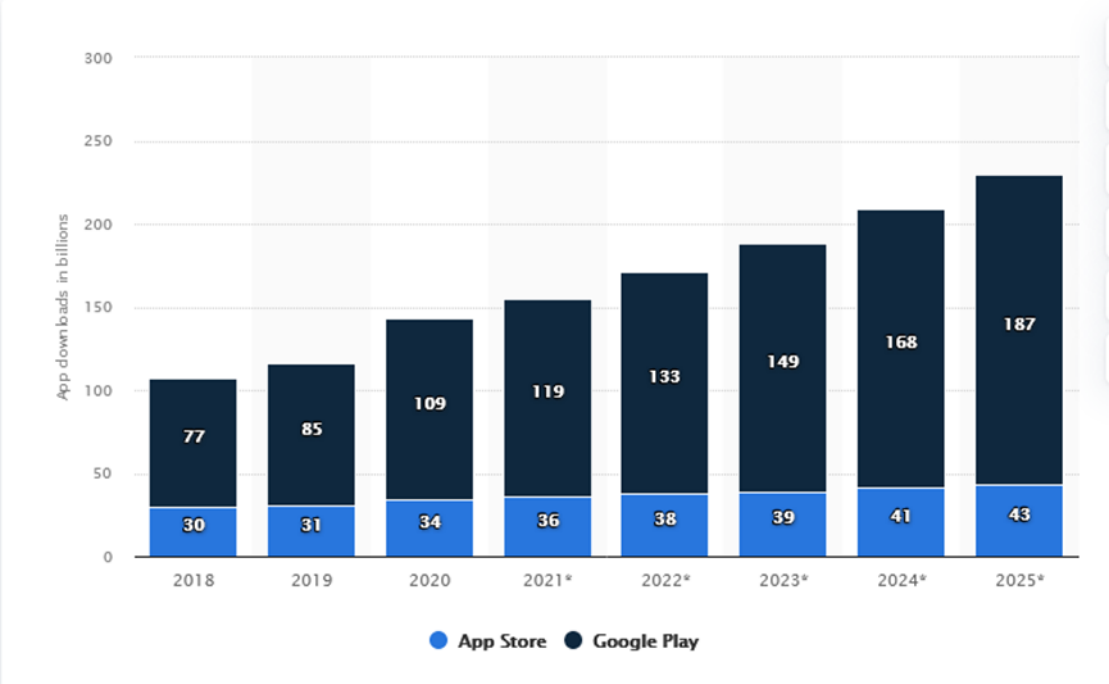
PII Personally Identifiable Information

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Η επικοινωνία είναι η διαδικασία ανταλλαγής πληροφοριών ανάμεσα στους ανθρώπους. Στις μέρες μας, οι άνθρωποι έχουν την ικανότητα να επικοινωνούν με πολλά μέσα. Οι κινητές συσκευές είναι το πιο γνωστό μέσο επικοινωνίας και μερικές φορές αποθηκεύουν σημαντικές πληροφορίες. Οι άνθρωποι χρησιμοποιούν τα smartphones σε σχεδόν όλες τις πτυχές της ζωής τους, όπως είναι η κοινωνική δικτύωση, η αναζήτηση των εσωτερικών σκέψεων και προτιμήσεων τους στο διαδίκτυο, η εύρεση μιας νέας επαγγελματικής θέσης εργασίας, η αναζήτηση θεμάτων που τους αφορούν όπως είναι η υγεία, τα οικονομικά κτλ. Ο αριθμός των χρηστών smartphones ξεπερνάει τα 6 δισεκατομμύρια και η προβλέψεις υπολογίζουν ότι αυτός ο αριθμός θα αυξάνεται κατά εκατοντάδες εκατομμύρια κάθε χρόνο[1]. Οι κατασκευαστές κινητών ξέρουν πόσο σημαντική ανάγκη είναι τα smartphones, για αυτό και ενσωματώνουν συνέχεια καινούργιες εφαρμογές ώστε να ικανοποιήσουν μέχρι και τον πιο απαιτητικό χρήστη.

Τα πιο δημοφιλή λειτουργικά συστήματα είναι το Android και το iOS, με το πρώτο να κυριαρχεί στην παγκόσμια αγορά με ποσοστό πάνω από 70% [2]. Η κύριες διαφορές τους είναι ότι το Android είναι ένα λογισμικό ανοιχτού κώδικα(open source) και είναι πιο ευέλικτο από το iOS. Αυτοί είναι και οι λόγοι που υπάρχουν πολλοί κατασκευαστές κινητών που χρησιμοποιούν το Android σε αντίθεση με το iOS που είναι διαθέσιμο μόνο από την Apple. Σύμφωνα με την πλατφόρμα Statista, έχει παρατηρηθεί αυξανόμενη τάση στο κατέβασμα εφαρμογών μέσω του Google Play Store χωρίς καμία ένδειξη για μείωση του ρυθμού αυτού (**Σχήμα 1.1**). Η τάση αυτή, προσελκύει όλο και περισσότερους κακόβουλους χρήστες (hackers) ,γεγονός που αυξάνει την πιθανότητα για επίθεση στις εφαρμογές των smartphones αλλά και του δικτύου που βρίσκεται η συσκευή. Ένα σημαντικό μέτρο στην ασφάλεια των εφαρμογών είναι ο έλεγχος διείσδυσης (penetration testing) , που σύμφωνα με αυτή τη μεθοδολογία , εντοπίζονται οι ευπάθειες και εκμεταλλεύονται με σκοπό να παρουσιαστεί το μέγεθος της ζημιάς που μπορεί να προκληθεί.



Σχήμα 1.1 : Statista - αριθμός εφαρμογών που έχουν κατέβει σε παγκόσμιο επίπεδο από το 2018 μέχρι το 2025 (σε δισεκατομμύρια)

Πηγή[3]

Γενικότερα, ως «penetration testing» ή αλλιώς «ethical hacking» ορίζεται η διαδικασία κατά την οποία πραγματοποιείται εξουσιοδοτημένη κυβερνοεπίθεση σε κάποιο υπολογιστικό σύστημα προκειμένου να εκτιμηθεί το επίπεδο ασφάλειας. Με λίγα λόγια εντοπίζονται ταυτόχρονα οι αδυναμίες αλλά και τα δυνατά σημεία του συστήματος που εξετάζεται.

Οργανισμοί όπως είναι ο OWASP και ο NIST δημοσιεύουν λίστες με τους πιο συνήθεις κινδύνους που αντιμετωπίζουν οι εφαρμογές στα κινητά με σκοπό να προστατέψουν τους χρήστες. Ένας τέτοιος κίνδυνος είναι το reverse engineering, η οποία τεχνική ανήκει στη μέθοδο του mobile application penetration testing.

Όπως συμβαίνει σε κάθε λειτουργικό σύστημα ή λογισμικό, κατά το στάδιο της ανάπτυξής του, οι προγραμματιστές άθελά τους αφήνουν κενά ασφαλείας ή αγνοούν τις καλές πρακτικές ασφάλισης του συστήματος. Τέτοια λάθη θα πρέπει να εντοπίζονται πολύ πριν την επίσημη κυκλοφορία της εφαρμογής. Μια εφαρμογή για να θεωρείται ότι έχει ένα σημαντικό επίπεδο ασφάλειας , θα πρέπει να πραγματοποιείται ανά τακτά χρονικά διαστήματα το penetration testing προκειμένου να μειωθούν οι ευπάθειες και οι επιτυχείς προσπάθειες επίθεσης. Το penetration testing είναι μια γνωστή μέθοδος ασφάλισης στο τομέα των εταιριών και των οργανισμών αλλά όχι και τόσο γνωστή για τις κινητές συσκευές. Σκοπός αυτής της εργασίας είναι η ανάδειξη της σπουδαιότητας: α) της ασφάλισης των εφαρμογών , προκειμένου τα δεδομένα του χρήστη και του πηγαίου κώδικα να μένουν ασφαλή και β) του mobile application penetration testing κατά τη γραμμή παραγωγής για τον εντοπισμό τρωτών σημείων στις εφαρμογές.

1.2 Στόχος - Κίνητρο

Τα τελευταία χρόνια, οι ζωές των ανθρώπων έχουν αλλάξει υπερβολικά με την ταχεία ανάπτυξη των εφαρμογών για τα κινητά, καθώς η χρήση τους έχει γίνει αναπόσπαστο κομμάτι της καθημερινότητάς τους. Με το πάτημα ενός κουμπιού πετυχαίνουν ενέργειες όπως: η επικοινωνία με συνάδελφους, η πληρωμή λογαριασμών, η παρακολούθηση του σπιτιού τους κ.ά. Πολλές γνωστές εφαρμογές όπως Whatsapp [4], Tinder, Mediatek και οι χρήστες τους έχουν πέσει θύματα των κενών ασφάλειας. Επίσης, το λογισμικό κατασκοπίας(spyware) Pegasus που στοχεύει στην παρακολούθηση των δραστηριοτήτων των κυβερνήσεων , τονίζει τη σημασία των ασφαλών εφαρμογών και κινητών συσκευών[5].

Ο στόχος της παρούσας διπλωματικής εργασίας είναι διπλός. Το πρώτο μέρος αφορά την κατανόηση του λειτουργικού συστήματος Android και της μεθοδολογίας του mobile penetration test. Βασικό κριτήριο για την πραγματοποίηση ενός επιτυχημένου penetration test είναι η άριστη γνώση του εξεταζόμενου λειτουργικού συστήματος. Το δεύτερο μέρος αφορά την πρακτική παρουσίαση του penetration testing σε δύο ευάλωτες εφαρμογές του OWASP, με λεπτομερή περιγραφή των εργαλείων που χρησιμοποιήσαμε αλλά και των ενεργειών που ακολουθήσαμε για την επίτευξη του στόχου μας.

1.3 Οργάνωση της εργασίας

Στο πρώτο κεφάλαιο της εργασίας έγινε αναφορά στο λογισμικό των κινητών συσκευών, το penetration testing και την ανάγκη ασφάλισης των εφαρμογών για κινητές συσκευές. Επίσης, προσδιορίστηκαν οι λόγοι και οι στόχοι που οδήγησαν και τέθηκαν στην εκπόνηση της συγκεκριμένης εργασίας. Στο δεύτερο κεφάλαιο θα γίνει ανάλυση του λογισμικού Android και της αρχιτεκτονικής των εφαρμογών του. Στο τρίτο κεφάλαιο θα αναφερθεί η έννοια του penetration test, οι τύποι και οι μέθοδοι που υπάρχουν, η σημαντικότητα του mobile penetration test, τα στάδια της μεθοδολογίας, καθώς και τα εργαλεία που μπορούν να βοηθήσουν. Στο τέταρτο κεφάλαιο θα παρουσιαστεί το πρότζεκτ του OWASP και οι κίνδυνοι που αντιμετωπίζουν σε μεγάλο βαθμό οι εφαρμογές στις κινητές συσκευές. Στο πέμπτο κεφάλαιο, θα προετοιμαστεί το pentesting lab περιβάλλον. Στη συνέχεια, στο έκτο κεφάλαιο θα διεξαχθεί ένα penetration test σε δύο ευάλωτες εφαρμογές με τη χρήση reverse engineering. Στο έβδομο κεφάλαιο θα εξαχθούν τα συμπεράσματα γύρω από την ασφάλεια των εφαρμογών και την αναγκαιότητα των penetration tests και θα αναφερθούν οι μελλοντικές επεκτάσεις της διπλωματικής. Τέλος, στο όγδοο κεφάλαιο θα δοθούν όλοι οι σύνδεσμοι-πηγές που βοήθησαν στην εκπόνηση της εργασίας.

Κεφάλαιο 2ο: Λειτουργικό σύστημα Android

2.1 Τι είναι το Android;

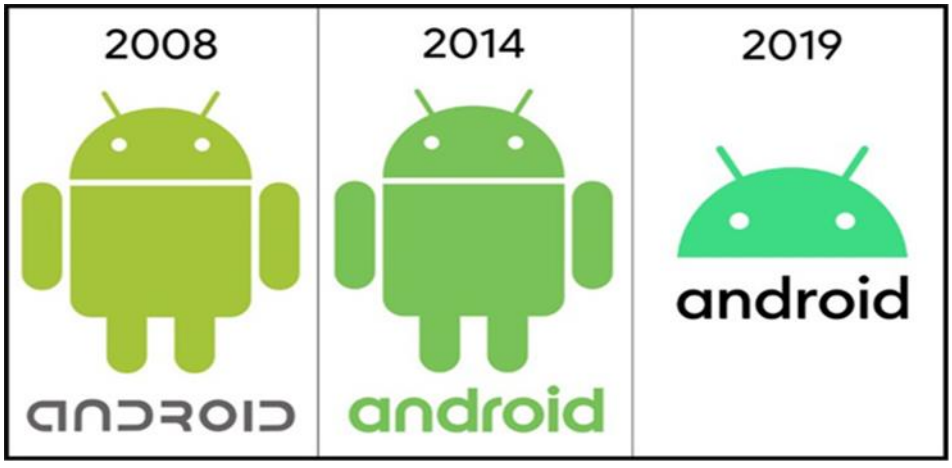
Το Android είναι ένα λειτουργικό σύστημα το οποίο κυκλοφορεί εδώ και 15 περίπου χρόνια και είναι βασισμένο σε μια τροποποιημένη έκδοση του Linux kernel. Αρχικά, σχεδιάστηκε για κινητές συσκευές με οθόνη αφής, όπως είναι τα smartphones και τα tablets αλλά πλέον οι Android εφαρμογές υποστηρίζονται και από άλλα λειτουργικά συστήματα, όπως είναι το Chrome Os και τα Windows 11. Η ανάπτυξη του λογισμικού πραγματοποιήθηκε από την Open Handset Alliance (OHA), ενώ η προώθησή του από την Google. Στόχος του πρότζεκτ Android είναι να δημιουργήσει ένα επιτυχημένο προϊόν στον πραγματικό κόσμο με σκοπό τη βελτίωση της κινητής τηλεφωνικής εμπειρίας.

Το Android είναι με μεγάλη διαφορά στο κόσμο το πιο δημοφιλές λειτουργικό σύστημα και είναι open source σε σχέση με άλλα λειτουργικά, όπως το Apple's iOS, macOS και Microsoft's Windows. Η πλέον κύρια γλώσσα για να προγραμματίσει κανείς σε Android είναι η Kotlin αλλά μπορεί να χρησιμοποιήσει και Java και C++[6]. Υπάρχουν πάνω από 3 εκατομμύρια εφαρμογές που λειτουργούν σε Android και οι περισσότερες από αυτές μπορούν να βρεθούν στο επίσημο Google Play Store, αλλά και να κατεβούν ανεπίσημα από το διαδίκτυο.

2.2 Η ιστορία του Android

Η Android Inc. δημιουργήθηκε στο Πάλο Άλτο της Καλιφόρνιας από τους Andy Rubin, Rich Miner, Nick Sears και Chris White τον Οκτώβρη του 2003. Το Android ξεκίνησε σα λογισμικό για κάμερες. Η ιδέα ήταν να δημιουργηθεί ένα λειτουργικό σύστημα με σκοπό να το χρησιμοποιούν όλες οι εταιρείες με κάμερες. Καθώς αναπτυσσόταν το λογισμικό , οι προαναφερθέντες συντελεστές συνειδητοποίησαν πως τα smartphones θα ήταν το μέλλον και αποφασίστηκε πως το Android θα χρησιμοποιούνταν σα λειτουργικό σύστημα για smartphones. Τον Ιούλιο του 2005 η Google εξαγόρασε την Android Inc. μαζί με τους δημιουργούς της για 50 εκατομμύρια δολάρια.

Η πρώτη αναφορά στο λογισμικό Android καταγράφηκε το 2007, ενώ η κυκλοφορία της πρώτης κινητής συσκευής με λειτουργικό Android έγινε το Σεπτέμβριο του 2008.



Σχήμα 2.1: Google Android logo με τη πάροδο των χρόνων.

Πηγή [7]

2.3 Εκδόσεις Android

Το ιστορικό των εκδόσεων του Android λειτουργικού συστήματος για κινητές συσκευές ξεκίνησε το Νοέμβριο του 2007 αλλά σε beta μορφή. Η πρώτη επίσημη και εμπορική έκδοση Android 1.0 κυκλοφόρησε στις 23 Σεπτεμβρίου του 2008[8]. Κάθε έκδοση που κυκλοφορούσε πήρε το όνομά της από κάποιο γλυκό.

Στον πίνακα **2.1** που ακολουθεί από τα αριστερά προς τα δεξιά φαίνεται το όνομα του Android, η κωδική του ονομασία, η έκδοσή του, η επίσημη ημερομηνία έκδοσης και το API level για την έκδοση.

Τα κελιά με κόκκινο χρώμα αναφέρουν ποιες εκδόσεις έχει σταματήσει να παρέχεται τεχνική υποστήριξη, με το κίτρινο ποιες εκδόσεις υποστηρίζονται και με το μπλε ποιες εκδόσεις είναι οι τελευταίες.

Πίνακας 2.1: Ιστορικό εκδόσεων Android

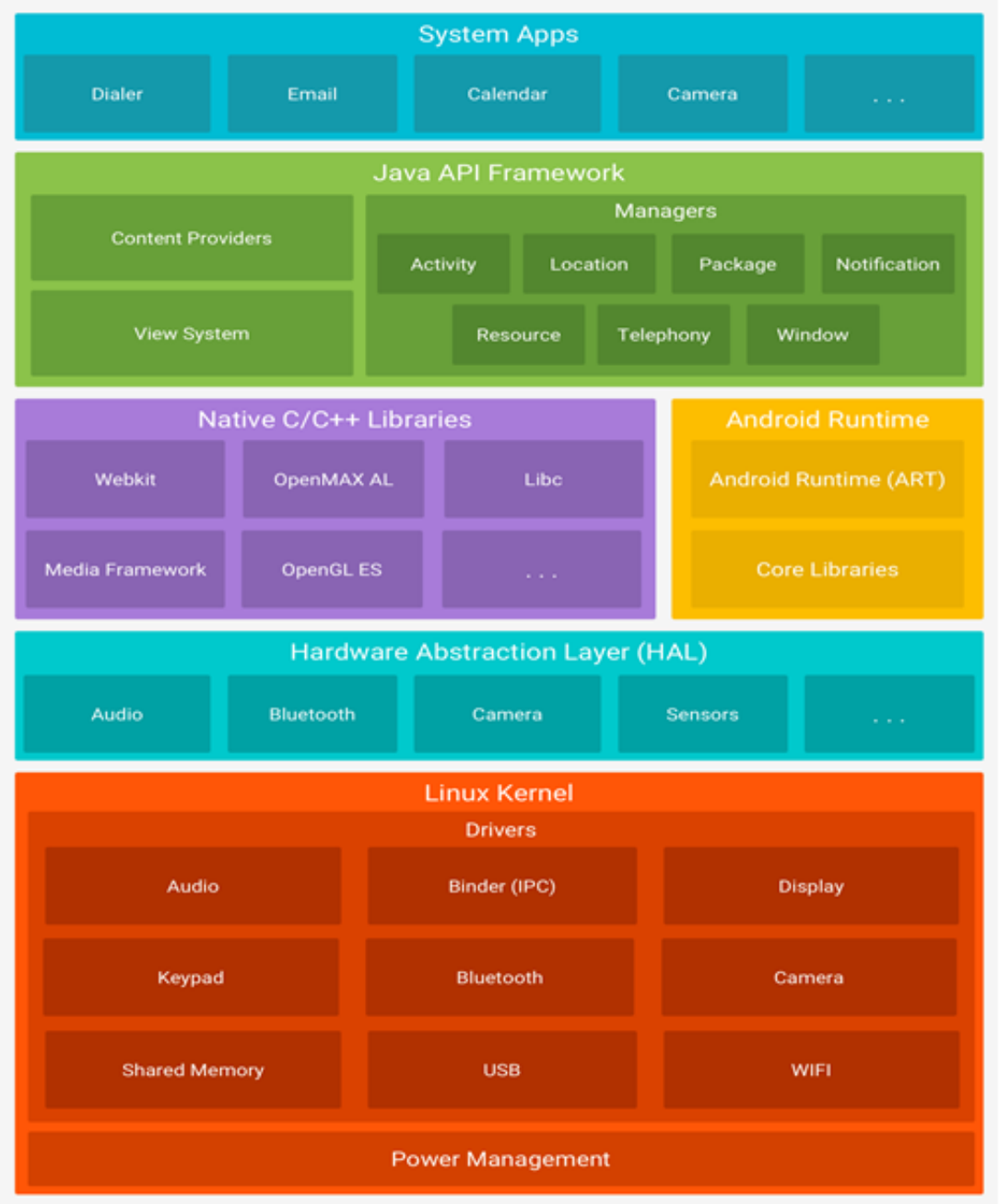
Όνομα	**Κωδική Ονομασία**	**Έκδοση**	**Επίσημη ημερομηνία έκδοσης**	**API Level**
Android 1.0	Android 1	1.0	23 Σεπτεμβριου 2008	1
Android 1.1	Petit Four	1.1	9 Φεβρουαρίου 2008	2
Android Cupcake	Cupcake	1.5	27 Απριλίου 2009	3
Android Donut	Donut	1.6	15 Σεπτεμβρίου 2009	4
Android Éclair	Éclair	2.0	27 Οκτωβρίου 2009	5
		2.0.1	3 Δεκεμβρίου 2009	6
Android Froyo	Froyo	2.2 – 2.2.3	20 Μαΐου 2010	8
Android Gingerbread	Gingerbread	2.3 – 2.3.2	6 Δεκεμβρίου 2010	9
		2.3.3 – 2.3.7	9 Φεβρουαρίου 2011	10
Android Honeycomb	Honeycomb	3.0	22 Φεβρουαρίου 2011	11
		3.1	10 Μαΐου 2011	12
		3.2 – 3.2.6	15 Ιουλίου 2011	13
Android Ice Cream Sandwich	Ice cream Sandwich	4.0 – 4.0.2	18 Οκτωβρίου 2011	14
		4.0.3 – 4.0.4	16 Δεκεμβρίου 2011	15
Android Jelly Bean	Jelly Bean	4.1 – 4.1.2	9 Ιουλίου 2012	16
		4.2 – 4.2.2	13 Νοεμβρίου 2012	17
		4.3 – 4.3.1	24 Ιουλίου 2013	18
Android KitKat	Key Lime Pie	4.4 – 4.4.4	31 Οκτωβρίου 2013	19
		4.4W – 4.4W.2	25 Ιουνίου 2014	20

Android Lollipop	Lemon Meringue Pie	5.0 - 5.0.2	4 Νοεμβρίου 2014	21
		5.1 – 5.1.1	2 Μαρτίου 2015	22
Android Marshmallow	Macadamia Nut Cookie	6.0 – 6.0.1	2 Οκτωβρίου 2015	23
Android Nougat	New York Cheesecake	7.0	22 Αυγούστου 2016	24
		7.1 – 7.1.2	4 Οκτωβρίου 2016	25
Android Oreo	Oatmeal Cookie	8.0	21 Αυγούστου 2017	26
		8.1	5 Δεκεμβρίου 2017	27
Android Pie	Android Pie	9	6 Αυγούστου 2018	28
Android 10	Quince Tart	10	3 Σεπτεμβρίου 2019	29
Android 11	Red Velvet Cake	11	8 Σεπτεμβρίου 2020	30
Android 12	Snow Cone	12	4 Οκτωβρίου 2021	31
Android 12L	-	-	-	32

2.4 Αρχιτεκτονική Android

Για να υπάρξει μια καλύτερη κατανόηση γύρω από τον τρόπο λειτουργίας του λογισμικού θα πρέπει να εξετασθεί η αρχιτεκτονική του. Το Android αποτελείται από πέντε διαφορετικά επίπεδα, όπως φαίνονται στο **Σχήμα 2.2** τα οποία θα αναλυθούν στη συνέχεια. Τα επίπεδα είναι:

- Linux Kernel
- Hardware Abstraction Layer (HAL)
- Native Libraries – Android Runtime (ART)
- Java API Framework
- System Applications



Σχήμα 2.2: Αρχιτεκτονική πλατφόρμας Android

Πηγή[9]

2.4.1 Linux Kernel

Στο χαμηλότερο επίπεδο της αρχιτεκτονικής του Android βρίσκεται το Linux Kernel το οποίο βασίζεται στην έκδοση 4.14 των Linux. Θεωρείται η «καρδιά» της αρχιτεκτονικής, καθώς διαχειρίζεται όλους τους διαθέσιμους drivers, όπως είναι της οθόνης, της κάμερας, του Bluetooth, του ήχου, της μνήμης κτλ. Επίσης, είναι υπεύθυνο για τη διαχείριση μνήμης, το multitasking, τη διαχείριση ενέργειας κ.ά. Να σημειωθεί πως σε αυτό το επίπεδο δεν υπάρχει άμεση αλληλεπίδραση χρήστη – συστήματος.

2.4.2 Hardware Abstraction Layer (HAL)

Στο επόμενο επίπεδο βρίσκεται το Hardware Abstraction Layer (HAL) το οποίο λειτουργεί ως διεπαφή μεταξύ του hardware με το υπόλοιπο Java API Framework και διαθέτει πολλά διαφορετικά library modules. Κάθε ένα από αυτά τα modules δημιουργεί μια διεπαφή για κάθε ξεχωριστό κομμάτι hardware. (π.χ. Κάμερα, Bluetooth κ.α.). Με άλλα λόγια, κάθε φορά που το API ζητά πρόσβαση σε κάποιο hardware κομμάτι του συστήματος, το Android καλεί το αντίστοιχο library module.

2.4.3 Native Libraries – Android Runtime

Στο τρίτο επίπεδο βρίσκονται οι Native Libraries οι οποίες περιλαμβάνουν C/C++ core libraries αλλά και libraries υλοποιημένες σε Java. Μερικές από αυτές είναι:

- **Media**: μια βιβλιοθήκη η οποία παρέχει υποστήριξη για την αναπαραγωγή και την καταγραφή ήχου και βίντεο.
- **Surface Manager**: μια βιβλιοθήκη υπεύθυνη για τη διαχείριση πρόσβασης στο υποσύστημα οθονών.
- **SGL** και **OpenGL**: και τα δύο είναι cross-platform, cross-language APIs που χρησιμοποιούνται για 2D και 3D γραφικά.
- **SQLite**: σύστημα για βάσεις δεδομένων
- **Web-Kit**: ένας browser ανοιχτού κώδικα
- **SSL**: μία τεχνολογία υπεύθυνη για την κρυπτογραφημένη σύνδεση μεταξύ ενός web browser και ενός web server.

Παράλληλα, σε αυτό το επίπεδο αυτό βρίσκεται και το Android Runtime που είναι το runtime environment του Android. Οι πρώτες εκδόσεις Android έτρεχαν τις εφαρμογές τους σε Virtual Machines στο εσωτερικό τους. Η πρώτη έκδοση ήταν το Dalvik Virtual Machine (DVM) το οποίο είχε την τεχνολογία «Just In Time» (JIT) που μετέτρεπε τον κώδικα της εφαρμογής σε γλώσσα μηχανής κάθε φορά που ο χρήστης εκκινούσε την εφαρμογή. Από την έκδοση 4.4 KitKat κι έπειτα, εμφανίστηκε σε πειραματικό στάδιο το ART (Android Runtime) που σε σχέση με την προηγούμενη τεχνολογία μετέτρεπε τον κώδικα της εφαρμογής σε γλώσσα μηχανής κατά τη διαδικασία της εγκατάστασης.

Πλέον η τεχνολογία που χρησιμοποιείται ονομάζεται «Ahead of Time» compilation (AOT) και λειτουργεί ως προκαθορισμένος τρόπος για τη μεταγλώττιση(compile) των εφαρμογών.[**10**]

2.4.4 Java API Framework

Στο συγκεκριμένο επίπεδο βρίσκεται το Java API Framework από το οποίο οι εφαρμογές αντλούν πολλές διαφορετικές υπηρεσίες με την μορφή κλάσεων. Οι προγραμματιστές εφαρμογών μπορούν να κάνουν χρήση από τις υπηρεσίες που ακολουθούν:

- **Activity Manager**: διαχειρίζεται από κάθε πλευρά τον κύκλο ζωής της εφαρμογής.
- **Content Providers**: επιτρέπει τις εφαρμογές να δημοσιεύουν και να μοιράζουν δεδομένα με άλλες εφαρμογές.
- **Resource Manager**: παρέχει πρόσβαση σε πηγές όπως είναι τα strings, color settings και user interface layouts.
- **Notifications Manager**: Επιτρέπει τις εφαρμογές να εμφανίζουν ειδοποιήσεις στη συσκευή του χρήστη.
- **Telephony Manager**: παρέχει πληροφορίες σχετικά με το κινητό πχ. Sim serial number, σήμα κινητού κ.α.
- **Location Manager**: παρέχει την δυνατότητα σε μια εφαρμογή να ενημερώνεται για την τρέχουσα τοποθεσία.

2.4.5 System Applications

Στο τελευταίο επίπεδο βρίσκονται οι εφαρμογές του συστήματος με τις οποίες έχει άμεση επαφή ένας απλός χρήστης. Μερικά παραδείγματα είναι η εφαρμογή των επαφών , του περιηγητή διαδικτύου, τα παιχνίδια, τα μηνύματα, τα email και άλλες οι οποίες είναι υλοποιημένες σε Java.

Σε αυτό το επίπεδο οι εφαρμογές έχουν την δυνατότητα να τρέχουν με το μηχανισμό του sandboxing, λεπτομέρειες του οποίου θα αναφερθούν σε επόμενη ενότητα της εργασίας.

2.5 Application Components

Τα Application Components είναι τα βασικά στοιχεία που χρειάζεται για να λειτουργήσει μία εφαρμογή. Συνδέονται μεταξύ τους και αλληλοεπιδρούν σύμφωνα με το αρχείο AndroidManifest.xml που βρίσκεται στο root φάκελο του συστήματος. Τα βασικά components είναι τέσσερα [11]:

- Activities
- Services
- Broadcast Receivers
- Content Providers

2.5.1 Activities

Ο κύριος τρόπος αλληλεπίδρασης του χρήστη με την εφαρμογή είναι μέσω των activities. Ως activity μπορεί να θεωρηθεί η κάθε διαφορετική οθόνη που έχει ένα user interface σε μια εφαρμογή. Για

παράδειγμα, όταν ανοίξει κανείς μια εφαρμογή με email, θα υπάρχει μια διαφορετική οθόνη – activity για τη λίστα των νέων email και μια διαφορετική οθόνη για την σύνταξη ενός καινούργιου email. Παρόλο που συνεργάζονται όλα τα activities μεταξύ τους για να παρέχουν μια ωραία εμπειρία στον χρήστη , το καθένα είναι ανεξάρτητο από το άλλο. Μια εφαρμογή μπορεί να έχει πολλά activities ή και μόνο ένα. Έτσι, μια διαφορετική εφαρμογή μπορεί να ξεκινήσει οποιοδήποτε άλλο activity από την εφαρμογή των email, εάν αυτή το επιτρέπει. Για παράδειγμα , η εφαρμογή της κάμερας μπορεί να ξεκινήσει το activity της εφαρμογής των email με σκοπό τη σύνθεση ενός νέου μηνύματος αλληλογραφίας επιτρέποντας το χρήστη να μοιραστεί μια φωτογραφία.

2.5.2 Services

Τα services είναι σημαντικά για την λειτουργία μιας εφαρμογής στο παρασκήνιο. Συνήθως είναι διεργασίες και εκτελούνται στο παρασκήνιο για μεγάλα χρονικά διαστήματα. Ένα service δεν παρέχει User Interface. Για παράδειγμα , ένα service μπορεί να παίζει μουσική στο παρασκήνιο, ενώ ο χρήστης να χρησιμοποιεί μια άλλη εφαρμογή ή ακόμα μπορεί να του έρθει μια ειδοποίηση από μια εφαρμογή που δεν την άνοιξε. Ένα service μπορεί να εκτελεστεί από ένα activity.

Υπάρχουν δύο κατηγορίες services: started services και bound services.

Started: τα started services λένε στο σύστημα ότι θα τρέχουν στο παρασκήνιο επ’ αόριστον μέχρι να τελειώσει η δουλειά τους.

Bound: τα bound services εκτελούνται, επειδή κάποια άλλη εφαρμογή (ή το σύστημα) χρειάστηκε να εκτελέσει αυτό το service. Ουσιαστικά το service παρέχει ένα API σε μια άλλη διαδικασία. Όταν ολοκληρωθεί αυτή η διαδικασία, τότε τερματίζεται το service σε αντίθεση με τα started.

2.5.3 Broadcast Receivers

Τα broadcast receivers είναι components που απαντούν σε broadcast messages που στέλνουν άλλες εφαρμογές ή ακόμα και το σύστημα. Το σύστημα μπορεί να στέλνει broadcasts ακόμα και σε εφαρμογές που είναι ανενεργές. Πολλά broadcasts δημιουργούνται από το σύστημα. Χαρακτηριστικά παραδείγματα είναι όταν ένα broadcast ανακοινώνει ότι έσβησε η οθόνη, ότι η μπαταρία είναι χαμηλή ή ότι βγήκε μια φωτογραφία.

2.5.4 Content Providers

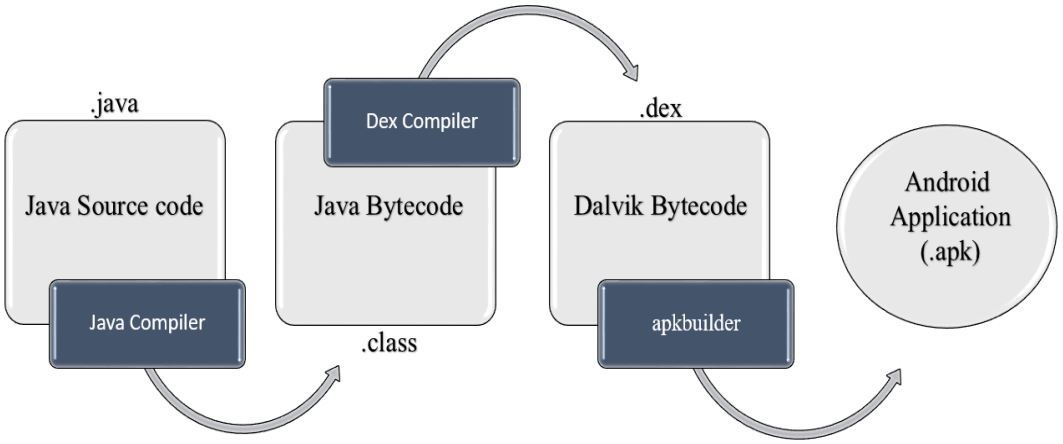
Οι εφαρμογές μπορεί να αποθηκεύουν τα δεδομένα τους στο file system, σε μια SQLite βάση δεδομένων κτλ. Οι content providers διαχειρίζονται τα δεδομένα αυτά και μπορούν μέσω άλλων εφαρμογών να τα αναζητήσουν ή ακόμα και να τα τροποποιήσουν αν υπάρχει η αντίστοιχη πρόσβαση(permission). Για παράδειγμα, το σύστημα Android παρέχει έναν content provider που διαχειρίζεται τις πληροφορίες από τις επαφές του χρήστη. Έτσι, κάθε εφαρμογή με το κατάλληλο permission μπορεί να κάνει ένα ερώτημα στον provider με σκοπό να διαβάσει ή να επεξεργαστεί τα στοιχεία για μια συγκεκριμένη επαφή.

2.6 Android Application Package (APK)

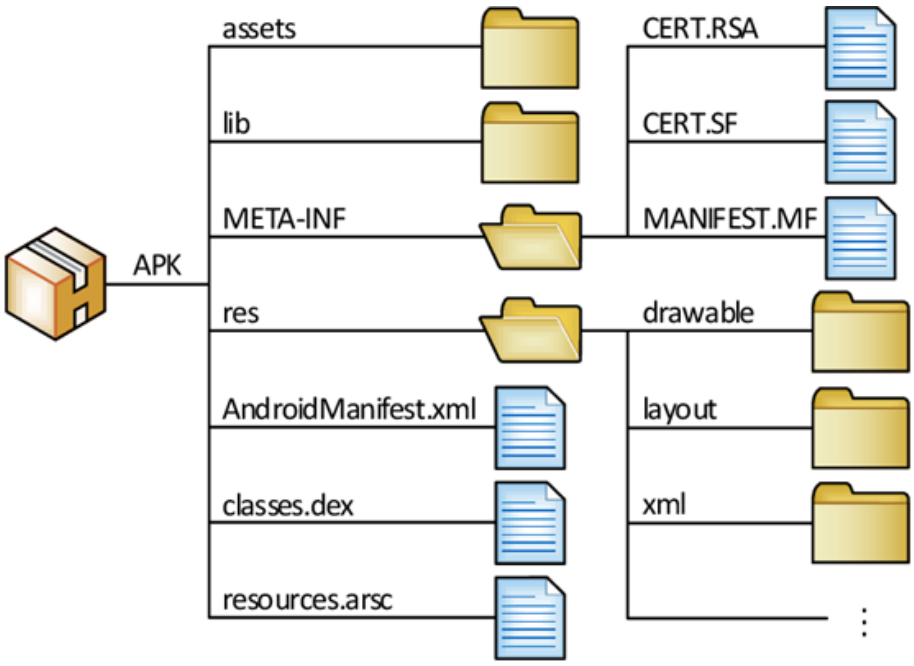
Το Android Package Kit (APK) είναι μια μορφή αρχείου που χρησιμοποιεί το λειτουργικό σύστημα Android για να μπορεί να διανέμει και να κάνει εγκατάσταση εφαρμογές στα κινητά. Το APK λειτουργεί ακριβώς όπως τα αρχεία «.exe» στα Windows. Σε ένα αρχείο APK περιλαμβάνονται όλα τα δεδομένα μιας εφαρμογής: τα resources της, το manifest, ο compiled πηγαίος κώδικά της κ.ά. (όπως βλέπετε στο **Σχήμα 2.4**) , τα οποία θα αναλυθούν στην συνέχεια. Τα APKs είναι μια παραλλαγή της μορφής αρχείου JAR(Java Archive) , καθώς μεγάλο τμήμα του Android έχει χτιστεί σε Java. Τα APKs θεωρούνται συμπιεσμένα αρχεία ZIP μορφής. Για να δει κανείς τα περιεχόμενα ενός APK, πολύ απλά μπορεί να χρησιμοποιήσει ένα εργαλείο για να το αποσυμπιέσει.

Η απλοποιημένη διαδικασία σύνθεσης ενός APK αρχείου είναι η εξής:

- Αρχικά, ο πηγαίος κώδικας γραμμένος σε Java γίνεται compile σε Java Bytecode και η κατάληξή του αλλάζει από «.java» σε «.class».
- Στη συνέχεια ο κώδικας σε Java Bytecode με τη βοήθεια του Dex compiler μετατρέπεται σε Dalvik Bytecode. Κάθε ένα ξεχωριστό αρχείο «.dex» μπορεί να έχει μέχρι 65536 μεθόδους, συμπεριλαμβάνοντας το Android framework, 3rd parties βιβλιοθήκες και οποιεσδήποτε άλλες μεθόδους έχει γράψει ο προγραμματιστής.
- Τέλος, όλα τα παραπάνω συμπιέζονται μαζί με τα resources και άλλα αρχεία και δημιουργούν ένα APK.



Σχήμα 2.3: Διαδικασία δημιουργίας APK αρχείου



Σχήμα 2.4: Δομή ενός Apk αρχείου

Πηγή[12]

- **Assets**: ο φάκελος περιλαμβάνει διάφορα αρχεία π.χ. ήχου, εικόνας κ.ά. που είναι χρήσιμα για την εφαρμογή.
- **Lib**: ο φάκελος περιλαμβάνει native βιβλιοθήκες.
- **META-INF**: ο φάκελος περιλαμβάνει τα απαραίτητα πιστοποιητικά και υπογραφές καθώς και το αρχείο manifest.
- **Res**: ο φάκελος αυτό περιλαμβάνει ό,τι στοιχεία δεν γίνονται compiled στο resources.arsc και είναι σημαντικά για την εμφάνιση της εφαρμογής πχ. εικόνες.
- **AndroidManifest.xml**: το αρχείο αυτό παρέχει μια λεπτομερή περιγραφή του περιεχομένου, της έκδοσης, των permissions και του ονόματος του αρχείου APK.
- **Classes.dex**: είναι οι μεταγλωττισμένες classes Java οι οποίες θα τρέξουν στη συσκευή.
- **Resources.arsc**: είναι τα compiled resources της εφαρμογής πχ τα strings.

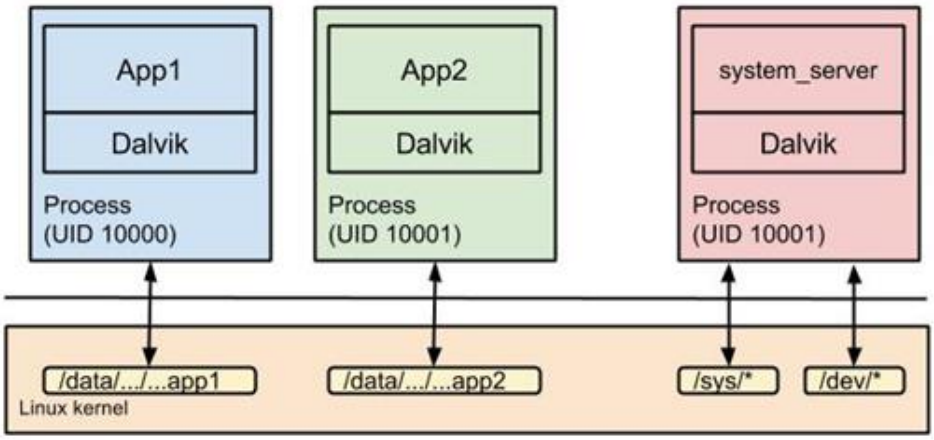
2.7 Android Security

2.7.1 Application Sandboxing

Το λειτουργικό σύστημα Android εκμεταλλεύεται την ικανότητα των συστημάτων Linux να αναγνωρίζουν και να απομονώνουν τις λειτουργίες κάθε εφαρμογής. Με αυτήν την απομόνωση προστατεύονται οι εφαρμογές, αλλά και το σύστημα από άλλες κακόβουλες εφαρμογές. Για να συμβεί αυτό, το Android αναθέτει έναν ξεχωριστό user ID(UID) σε κάθε Android εφαρμογή και αυτή τρέχει

σαν ξεχωριστή διεργασία. Η κάθε εφαρμογή έχει πρόσβαση μόνο στα δικά της αρχεία. (όπως φαίνεται στο **Σχήμα 2.5**).

Το UID μπορεί να πάρει ακέραιες τιμές τυχαία από το 10000 έως το 99999. Βάσει του κωδικού που λαμβάνει η κάθε εφαρμογή , ονομάζεται έτσι και η αντίστοιχη διαδικασία. Για παράδειγμα, αν ανατεθεί ο κωδικός UID 10163, τότε η αντίστοιχη διαδικασία θα πάρει το όνομα «uo_a163»



Σχήμα 2.5: Διαδικασία Application Sandboxing

Πηγή[13]

Το Android Application Sandbox επιδρά θετικά στον τομέα της ασφάλειας, καθώς ξεχωρίζει τα δεδομένα της εφαρμογής και του κώδικα εκτέλεσης από άλλες εφαρμογές που βρίσκονται στην ίδια συσκευή. [14]

2.7.2 Application Signing

Η υπογραφή εφαρμογών κάνει την εφαρμογή έμπιστη και τη διαδικασία της αναβάθμισής της λιγότερο περίπλοκη. Κάθε εφαρμογή που τρέχει στο λειτουργικό σύστημα Android θα πρέπει να είναι υπογεγραμμένη από τους προγραμματιστές της. Οι εφαρμογές που προσπαθούν να εγκατασταθούν χωρίς να είναι υπογεγραμμένες απορρίπτονται είτε από το Google Play είτε από τον package installer της Android συσκευής[14].

Η μέθοδος της υπογραφής του κώδικα είναι βασισμένη στην υπογραφή των JAR αρχείων, καθώς τα APK αρχεία είναι μια προέκταση της μορφής πακέτου JAR. Το Android χρησιμοποιεί την υπογραφή του APK με σκοπό να βεβαιώσει ότι οι αναβαθμίσεις προέρχονται από τον ίδιο τον δημιουργό και να εγκαθιδρύσει μια «σχέση εμπιστοσύνης» με τις άλλες εφαρμογές του συστήματος[15].

Κεφάλαιο 3ο: Penetration testing

3.1 Τι είναι το penetration testing;

Το Penetration testing, γνωστό ως και pen test, security pen testing ή security testing είναι μια μορφή του «ethical hacking». Pen test ονομάζεται η σκόπιμη πραγματοποίηση προσομοιωμένων ηλεκτρονικών επιθέσεων (cyberattacks) από «white hat» penetration testers χρησιμοποιώντας συγκεκριμένες στρατηγικές και ειδικά εργαλεία σχεδιασμένα να αποκτούν πρόσβαση και να εκμεταλλεύονται ηλεκτρονικά συστήματα, δίκτυα, εφαρμογές και ιστοσελίδες [16][17][18]. Το penetration testing μπορεί να διεξαχθεί είτε χειροκίνητα με τη χρήση ειδικών εργαλείων είτε μέσω αυτόματων εργαλείων [19]. Ο κύριος στόχος του pen testing είναι να αξιολογήσει την ασφάλεια ενός πληροφοριακού συστήματος ή μιας εφαρμογής ,προσπαθώντας να εκμεταλλευτεί τα αδύνατα σημεία που μπορεί να υπάρχουν. Οι ευπάθειες αυτές μπορεί να υπάρχουν σε λειτουργικά συστήματα, σε εφαρμογές, σε λανθασμένους παραμέτρους στο σύστημα(misconfigurations), σε λάθος γραμμές κώδικα οι οποίες δεν ανακαλύφθηκαν ποτέ κατά την φάση του testing ή σε ριψοκίνδυνες συμπεριφορές χρήστη. Ακόμα το pen testing μπορεί να γίνει με σκοπό τον έλεγχο των πολιτικών ασφαλείας σε έναν οργανισμό και ένα συμμορφώνεται απέναντι σε κανόνες και νομοθεσίες (π.χ. GDPR)[20].

Οι ethical hackers, γνωστοί ως και «white hats» είναι εξειδικευμένοι επαγγελματίες στον τομέα της κυβερνοασφάλειας οι οποίοι έχουν άριστες τεχνικές γνώσεις και ικανότητες να μπορούν να αναγνωρίζουν και να εκμεταλλεύονται ευπάθειες στο στόχο τους. Ένας ethical hacker λειτουργεί με την άδεια των ιδιοκτητών και θα πρέπει να συμμορφώνεται με τους κανόνες και τις πολιτικές του οργανισμού-στόχου. [21]

Το penetration test θεωρείται ένα σημαντικό μέτρο κυβερνοασφάλειας καθώς τα ευρήματά του συγκεντρώνονται σε μια τελική αναφορά και αποστέλλονται στο υπεύθυνο τμήμα (π.χ. IT) με σκοπό την αναβάθμιση του συστήματος και την δημιουργία ή την τροποποίηση των ήδη υπαρχουσών στρατηγικών με στόχο την μέγιστη ασφάλεια.

3.2 Τύποι penetration testing

Οι τύποι των penetration tests είναι 4:

3.2.1 Infrastructure Penetration Testing

Αποτελεί το πιο δημοφιλές είδος test. Στόχος του είναι ο εντοπισμός ευπαθειών στην υποδομή του δικτύου του οργανισμού. Οι 3 υποκατηγορίες του είναι: **external**, **internal** και **wireless**.

- Στο **external** test αναγνωρίζονται οι ευπάθειες στην ασφάλεια σαν να γίνεται μια κανονική επίθεση έξω από το δίκτυο.
- Το **internal** test εστιάζεται στο τι θα μπορούσε να είχε πετύχει ένας επιτιθέμενος, εάν είχε αποκτήσει πρόσβαση στο δίκτυο.

- Το **wireless** test εφαρμόζεται, όταν χρησιμοποιούνται ασύρματες τεχνολογίες όπως είναι το Wi-Fi. Μερικές από τις ενέργειες του είναι:
    - Εντοπισμός Wi-Fi δικτύων και ασύρματων δακτυλικών αποτυπωμάτων(wireless fingerprinting),καθώς και η διαρροή πληροφοριών και σήματος(information and signal leakage).
    - Προσδιορισμός αδυναμίας στην κρυπτογράφηση, wireless sniffing και session hijacking.
    - Εύρεση τρόπων για διείσδυση στο δίκτυο.

Το Network Penetration testing περιλαμβάνει:

- Παράκαμψη των firewalls.
- FTP client/server tests.
- Router testing.
- Αποφυγή IPS/IDS.
- DNS footprinting.
- Σκανάρισμα ανοιχτών Port.
- Επιθέσεις SSH.
- Proxy Servers.
- Wireless encryption protocols.
- MAC address spoofing.
- DoS attacks.

3.2.2 Application Penetration Testing

Το test αυτό είναι λεπτομερές και στοχευμένο στο να προσπαθεί να βρει ευπάθειες και αδυναμίες τόσο στον κώδικα της εφαρμογής όσο και στη σχεδίαση. Εφαρμόζεται κυρίως σε φυλλομετρητές(browsers), διαδικτυακές εφαρμογές και στα επί μέρη τους στοιχεία όπως είναι τα Plug-ins, ActiveX και Applets[22].
Το πιο γνωστό είναι το Web Application Penetration Test(WAPT) και οι τεχνικές που εφαρμόζονται είναι οι εξής:

- SQL Injection.
- Cross-Site Scripting(XSS).
- XML External Entities Injection(XXE).
- Broken authentication και session management.
- Password Cracking

Οι εφαρμογές μπορεί να είναι:

- Διαδικτυακές εφαρμογές(Java, PHP, .NET).
- APIs.
- Συνδέσεις(XML, MySQL, Oracle).
- Frameworks.
- Συστήματα(SAP, CRM, Logistics, Financial, HR).
- Εφαρμογές για κινητά.

3.2.3 Social Engineering Penetration Testing

Ο πιο αδύναμος κρίκος συνήθως σε έναν οργανισμό είναι οι υπάλληλοί του , οι οποίοι μπορεί να κάνουν συχνά λάθη. Με αυτό τον τύπο test εξετάζεται πόσο επιρρεπές είναι το προσωπικό στη διαρροή εμπιστευτικών πληροφοριών. Το test μπορεί να γίνει είτε εξ αποστάσεως(remote) είτε δια ζώσης(**physical**)[23].

1. Στην πρώτη περίπτωση (**remote** test) ο tester προσπαθεί να ξεγελάσει τον υπάλληλο με σκοπό να αποκτήσει ευαίσθητες πληροφορίες. Η πιο γνωστή τεχνική είναι η χρήση email με σκοπό το «ηλεκτρονικό ψάρεμα»(phishing).
2. Στη δεύτερη περίπτωση (**physical** test) απαιτείται άμεση επαφή με το στόχο με σκοπό την απόκτηση των σημαντικών πληροφορίων. Παρουσιάζεται πόσο εύκολα θα μπορέσει να εισέλθει στις εγκαταστάσεις του οργανισμού ένας κακόβουλος χρήστης. Μερικές από τις τεχνικές που χρησιμοποιούνται είναι οι εξής:

- Tailgating
- Name-dropping
- Pre-texting
- Dumpster Diving
- Eavesdropping
- RFID και lock-picking

Πριν γίνει ένα social engineering penetration test ο οργανισμός θα πρέπει να ενημερώσει τους υπαλλήλους του. Εάν το test αποτύχει, η διοίκηση θα πρέπει να εκπαιδεύσει καλύτερα το προσωπικό της και να το ευαισθητοποιήσει για την ασφάλεια στον κυβερνοχώρο.

3.2.4 Cloud Penetration Testing

Στις μέρες μας , η χρήση δημόσιων cloud υπηρεσιών για τη χρήση υπολογισμών και αποθήκευσης είναι πολύ γνωστή. Για αυτό το λόγο προσελκύει πληθώρα επιθέσεων κακόβουλων χρηστών. Το cloud pen testing περιέχει:

- Πρόσβαση σε εφαρμογές και API.
- Πρόσβαση σε Βάσεις Δεδομένων και αποθηκευτικών χώρων.
- Έλεγχος κακών χρησιμοποιημένων firewalls.
- Έλεγχος των VMs και Operating Systems.
- Σύνδεση διαχειριστή από απόσταση με SSH και RDP

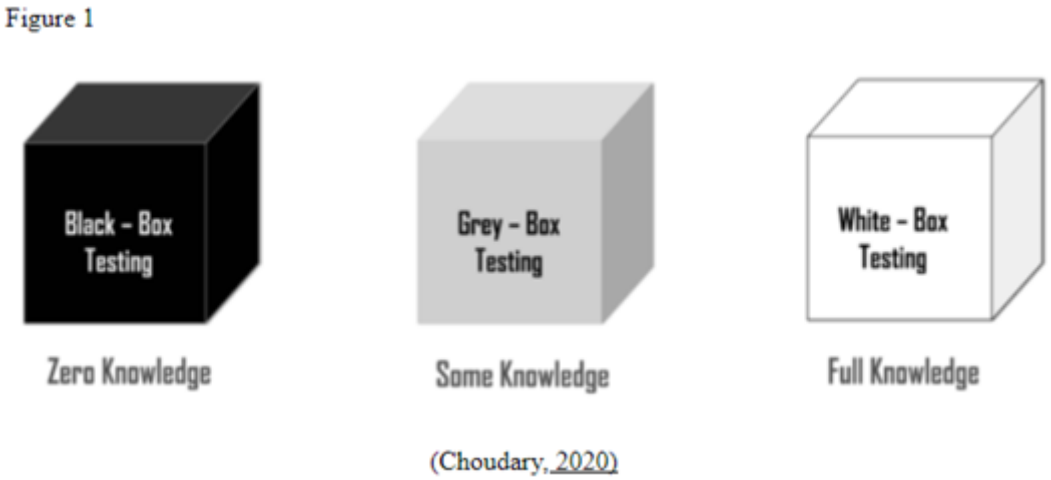
Καλό θα ήταν πριν ξεκινήσει το pen test να ενημερωθεί ο πάροχος της cloud υπηρεσίας για το test και να διατυπωθούν τα όρια που μπορεί να φτάσει.

Το cloud penetration testing μπορεί να είναι πολύ δύσκολο, γι’ αυτό και το test θα πρέπει να γίνει με τη μέθοδο του white-box, που θα αναλυθεί στη συνέχεια.[24]

Εάν ο πελάτης χρησιμοποιεί το Microsoft Azure, τότε ο tester θα πρέπει να συμμορφωθεί με τους Microsoft Cloud Unified Penetration Testing Rules of Engagement. Εάν χρησιμοποιεί Amazon Web

Services(AWS), τότε θα πρέπει να συμπληρωθεί η AWS Vulnerability / Penetration Testing Request Form.

3.3 Μέθοδοι Penetration Testing



Σχήμα 3.1: Μέθοδοι Penetration testing

Πηγή[25]

Οι μέθοδοι των penetration tests συνήθως εξαρτώνται από το εύρος των πληροφοριών και από τον βαθμό πρόσβασης που παρέχει ο κάθε οργανισμός για το σύστημά του ή την εφαρμογή του. Έτσι, προκύπτουν 3 διαφορετικοί μέθοδοι pen tests:

3.3.1 Black-Box

Το Black-Box testing , γνωστό ως και closed test ή εξωτερικό(external) penetration testing το οποίο προσομοιώνει μια επίθεση που γίνεται έξω από τον οργανισμό. Ο pen tester ξεκινάει με λίγες έως καθόλου πληροφορίες για την υποδομή και τους μηχανισμούς ασφάλειας του οργανισμού, όπως δηλαδή και ένας κακόβουλος χρήστης. Δεν έχει γνώση για:

i. Τις διαδικτυακές εφαρμογές.
ii. Την αρχιτεκτονική του λογισμικού.
iii. Τον πηγαίο κώδικα.

Αυτή η μέθοδος testing δίνει διευκρινίσεις όσο αφορά τις τεχνικές και τα εργαλεία που χρειάζεται ένας επιτιθέμενος προκειμένου να παραβιάσει τους τρόπους άμυνας. Επίσης, παρουσιάζει το μέγεθος της ζημιάς που θα μπορούσε να προκαλέσει, εάν αποκτούσε πρόσβαση στο σύστημα. Σύμφωνα με το Ινστιτούτο Infosec[26]:

Το black-box penetration testing βασίζεται στη δυναμική ανάλυση των προγραμμάτων και των συστημάτων που τρέχουν εντός του δικτύου. Ο tester θα πρέπει να είναι εξοικειωμένος με εργαλεία που «σκανάρουν» αυτόματα και μεθοδολογίες που γίνονται χειροκίνητες. Επίσης , θα πρέπει να είναι ικανός να δημιουργεί το δικό του διάγραμμα του δικτύου με βάση τις παρατηρήσεις του, καθώς δε του δίνετε καμία πληροφορία.

Το Black-box test μπορεί να διαρκέσει μέχρι και 6 εβδομάδες και αυτό εξαρτάται από την ικανότητα του tester να βρίσκει εύκολα τις ευπάθειες.

Θετικά:

- Παρέχει ένα ρεαλιστικό σενάριο και αναφέρεται στον τρόπο με τον οποίο γίνεται μια επίθεση από κάποιον που δεν έχει πληροφορίες για τον οργανισμό.
- Ο οργανισμός αναγνωρίζει ποια είναι τα κρίσιμα του σημεία, καθώς προσομοιώνεται μια αληθινή επίθεση.

Αρνητικά:

- Ο χρόνος, καθώς διαρκεί εβδομάδες.
- Είναι η πιο ακριβή μέθοδος test.[27]

3.3.2 White box

Το White-box penetration testing ονομάζεται κι αλλιώς ως structural, clear box, glass box, open box ή internal penetration testing[28]. Είναι ακριβώς το αντίθετο του black-box καθώς ο tester έχει πλήρη γνώση των συστημάτων, των εφαρμογών και του δικτύου του οργανισμού δηλαδή: το διάγραμμα του δικτύου, τον Πηγαίο Κώδικα, λεπτομέρειες για το λειτουργικό σύστημα, IP διευθύνσεις κ.ά. Θεωρείται σα μια προσομοίωση επίθεσης που γίνεται από κάποιον υπάλληλο ή ενός κακόβουλου χρήστη που έχει ήδη αποκτήσει πρόσβαση στο σύστημα.

Οι pen testers ξεκινάνε με τα ίδια δικαιώματα που έχει ένας εξουσιοδοτημένος χρήστης και από εκεί προσπαθούν να εντοπίσουν κενά ασφάλειας και να βρουν αδυναμίες στον στόχο τους.

Θετικά:

- Το test είναι διεξοδικό.
- Μπορεί να διαρκέσει 2-3 εβδομάδες.
- Καλύπτει περιοχές που δεν φτάνει το black box όπως: ποιότητα του κώδικα, έλεγχος τυπογραφικών λαθών, σχεδίαση των εφαρμογών.

Αρνητικά:

- Δεν παρουσιάζει ένα αληθινό σενάριο επίθεσης, καθώς παρέχονται όλες οι πληροφορίες.
- Καθώς ο tester έχει πλήρης γνώση, μπορεί να αργήσει να αποφασίσει που θα εστιάσει την προσοχή του, για αυτό και απαιτείται αρκετή εμπειρία και ένα καλό πλάνο από την αρχή.
- Για την επίτευξη αυτού του είδους test , χρειάζεται εξελιγμένα και ακριβά εργαλεία όπως: software code analyzers και debuggers.

3.3.3 Gray box

Το Gray–Box βρίσκεται ανάμεσα στο white και το black. Κατά τη διάρκεια του gray-box test, ο pen tester έχει μερική γνώση και πρόσβαση στο εσωτερικό δίκτυο ή την εφαρμογή του οργανισμού. Τα πιο συνήθη σενάρια είναι:

- Ο tester έχει τους κωδικούς από έναν απλό χρήστη ή του admin.
- Ο tester έχει πλήρης γνώση για την αρχιτεκτονική και τη λειτουργία της εφαρμογής.
- Ο tester έχει πρόσβαση σε μερικά κομμάτια του κώδικα.

Αυτό το είδος test χρησιμοποιεί μεθοδολογίες και από τα δύο προηγούμενα είδη test, δηλαδή το white-box και το black-box.[28]

Θετικά:

- Δεν είναι απαραίτητη η παροχή πληροφοριών για τη λειτουργία των εφαρμογών.
- Παρέχει μια αποτελεσματική αξιολόγηση της ασφάλειας του οργανισμού.

Αρνητικά:

- Τα test δεν είναι εύκολα να σχεδιαστούν.
- Δεν είναι κατάλληλος τρόπος για κατανεμημένα συστήματα.

3.4 Ομάδες στο penetration testing

Ο αριθμός των pen testers που θα εμπλακούν σε ένα πρότζεκτ εξαρτάται από τρεις παράγοντες:

1. Ο τύπος του penetration testing που θα εκτελεστεί.
2. Το μέγεθος της εταιρίας ή του οργανισμού.
3. Η πολυπλοκότητα της υποδομής που θα γίνει το pen test.

Για παράδειγμα, η υποδομή ενός οργανισμού που επιθυμεί να του γίνει ένα pen test και έχει λιγότερους από 20 υπαλλήλους θεωρείται σχετικά απλή. Σε αυτό το παράδειγμα, δε χρειάζεται μια ολοκληρωμένη ομάδα pentesters, καθώς δυο-τρεις είναι αρκετοί για να εκτελέσουν τα tests και να συντάξουν την τελική αναφορά με τα αποτελέσματα και τις προτάσεις τους. Εάν, όμως ο οργανισμός που χρειάζεται penetration testing είναι μεγάλος (για παράδειγμα πάνω από 10.000 υπαλλήλους) , η υποδομή του είναι πολύπλοκη. Συνεπώς, θα πρέπει να εφαρμοστεί μια δομημένη penetration testing ομάδα. Σε τέτοιες περιπτώσεις, συνήθως δημιουργούνται τρεις ομάδες: η **red** team , η **blue** team και μπορεί η **purple** team[29]. Πιο αναλυτικά:

1. Red Team

Κύριος ρόλος της red team είναι να εκτελέσει μια κυβερνοεπίθεση στα πλαίσια της ηθικής εναντίον μιας επιχείρησης ή ενός οργανισμού με σκοπό να ανακαλύψει άγνωστες ευπάθειες ή αδυναμίες. Είναι σημαντικό να σημειωθεί πως την red team δεν την ενδιαφέρει σε ποιον επιτίθεται αλλά να βρει τρόπους να εισέλθει στους στόχους της. Τα τεστ που κάνει τυπικά περιλαμβάνει τόσο επίθεση στο επίπεδο του software (βάση δεδομένων, διαδικτυακή εφαρμογή κτλ.) όσο και στο επίπεδο του hardware(servers, δίκτυο της επιχείρησης κτλ.).

Η red team θα χρησιμοποιήσει πολύ δημιουργικότητα στις τεχνικές της, καθώς ο στόχος της είναι η πρόσβαση με κάθε αναγκαίο μέσο.

2. Blue Team

Αυτή η ομάδα του penetration testing μιμείται τον ρόλο του προσωπικού της πληροφορικής σε μια εταιρία ή οργανισμό. Η blue team θα είναι αυτή που θα παρακολουθεί όλες τις ειδοποιήσεις, τις ανωμαλίες και οποιαδήποτε άλλη ύποπτη κίνηση που μπορεί να συμβεί μέσα στο δίκτυο. Ουσιαστικά το αποτέλεσμα του pentesting της είναι διπλός: να αποτρέψει την κυβερνοεπίθεση της red team και να δώσει μια ιδέα της απαιτούμενης επαγρύπνησης και διορατικότητας στο πραγματικό προσωπικό της εταιρίας. Πιο συγκεκριμένα, η blue team πρέπει:

- **Να είναι σε ετοιμότητα**: Περιλαμβάνει τη δοκιμή όλων των μηχανισμών ασφάλειας που υπάρχουν για να βεβαιωθούν ότι είναι βελτιστοποιημένοι για τον εντοπισμό κάθε είδους

πιθανής ύποπτης συμπεριφοράς. Δηλαδή διασφαλίζει ότι όλα τα τείχη προστασίας (firewalls), οι συσκευές ανίχνευσης ή αποτροπής δικτύου (IDS ή IPS) , οι δρομολογητές, οι συσκευές που φιλτράρουν τα πακέτα και ούτω καθεξής λειτουργούν σε βέλτιστη κατάσταση.

- **Να αναγνωρίζει και να περιορίζει τις απειλές**: Καταβάλλεται κάθε προσπάθεια για να εντοπιστεί επιτυχημένα οποιαδήποτε πιθανή κυβερνοεπίθεση εναντίον της επιχείρησης ή της εταιρείας. Ακόμα, περιλαμβάνει τον περιορισμό της ζημιάς που προκλήθηκε από την κυβερνοεπίθεση, εκτελώντας σωστά το σχέδιο αντιμετώπισης περιστατικών(incident response plan).
- **Να ανακτά το σύστημα**: Αυτό είναι το σημείο όπου οι κρίσιμες λειτουργίες και διαδικασίες της εταιρίας ή του οργανισμού επιστρέφουν στο διαδίκτυο, συνήθως σε χρονικό διάστημα μία-δυο ημερών.
- **Να θωρακίζει τα λειτουργικά συστήματα**: Ο στόχος εδώ είναι η ελαχιστοποίηση των ευπαθειών στα λειτουργικά συστήματα που χρησιμοποιούνται.

3. Purple team

Σκοπός της purple team είναι η διασφάλιση και η μεγιστοποίηση των προσπαθειών τόσο της red όσο και της blue team. Ουσιαστικά αποτελείται από τον συνδυασμό μελών και των δύο ομάδων. Μερικοί από τους στόχους της είναι:

- **Να δουλεύει σε αρμονικό επίπεδο με την red και blue team**. Εδώ περιλαμβάνεται οι παρατηρήσεις και οι σημειώσεις του τρόπου με τον οποίο συνεργάζονται οι δύο ομάδες, καθώς και οι συστάσεις για αλλαγή της σύνθεσης των ομάδων.
- **Να κατανοεί τον στόχο και των δύο ομάδων.** Αυτό σημαίνει πως θα πρέπει να υιοθετήσει τη νοοτροπία, τις σκέψεις και τις ευθύνες τόσο της red όσο και της blue team.
- **Να παρέχει το βέλτιστο αποτέλεσμα στον πελάτη**. Συλλέγοντας πληροφορίες και δεδομένα και από τις δυο ομάδες, η Purple team μπορεί να παραδώσει μια τελική αναφορά με πολλές λεπτομέρειες. Με αυτήν την ενέργεια, οι μηχανισμοί ασφάλειας ενισχύονται με τον βέλτιστο τρόπο.

3.5 Τι είναι το Mobile Application Penetration Testing και γιατί είναι σημαντικό;

Το Mobile Application Penetration Testing (MAPT) είναι ένα μέτρο ασφάλειας που χρησιμοποιούν οι οργανισμοί με σκοπό την αξιολόγηση του επιπέδου ασφάλειας των κινητών εφαρμογών τους. Το MAPT βασίζεται στον OWASP Mobile Application Security Verification Standard (MASVS), ο οποίος είναι ένας οδηγός για τη χρήση καλών τεχνικών ασφάλειας στις εφαρμογές για τα κινητά.

Ουσιαστικά, πρόκειται για μία ρεαλιστική πιθανή επίθεση που εκτελείται από ειδικούς σε θέματα ασφάλειας σε μια εφαρμογή με σκοπό την εύρεση ευπαθειών σε 3 μέρη: στην ίδια την εφαρμογή, στο API της και στον server που φιλοξενεί το API.

Τα στοιχεία που «τεστάρονται» σε ένα MAPT είναι:

- Οι χώροι αποθήκευσης.
- Η επικοινωνία μεταξύ εφαρμογής-API.
- Ο πηγαίος κώδικας.
- Οι ρυθμίσεις παραμετροποίησης του συστήματος (π.χ. signature, debug).

- Ο server και οι υπηρεσίες του (web, mail, FTP, SSH κ.ά.).
- Οι πολιτικές της εταιρίας (π.χ. πώς συνεργάζονται και διανέμουν τον κώδικα μεταξύ τους οι υπάλληλοι.)

Τα τελικά αποτελέσματα των tests συνοψίζονται σε μία τελική αναφορά, η οποία περιέχει αναλυτικά ποιες ευπάθειες εντοπίστηκαν, πού εντοπίστηκαν, τι είναι, πώς μπορούν να εκμεταλλευτούν και ποιες λύσεις υπάρχουν για να εξαλειφθούν[30].

Διεξάγοντας ένα MAPT ο οργανισμός μπορεί να αναγνωρίσει διάφορες ευπάθειες, loopholes και άλλες αδυναμίες στην εφαρμογή του, πριν γίνει διαθέσιμη στο κοινό. Σαν αποτέλεσμα, ο οργανισμός θα έχει συνειδητοποιήσει πώς θα πρέπει να τροποποιήσει με το βέλτιστο τρόπο το design, τον κώδικα και την αρχιτεκτονική της εφαρμογής, πριν την επίσημή της κυκλοφορία. Το κόστος που απαιτείται για να διορθωθούν οι αδυναμίες που πιθανώς να εντοπιστούν κατά τη διάρκεια ενός MAPT είναι πολύ μικρότερο σε σχέση με την οικονομική ζημιά που θα είχε υποστεί μια επιχείρηση, εάν είχαν διαρρεύσει εμπιστευτικές πληροφορίες μετά την εκμετάλλευση των ευπαθειών της εφαρμογής από κακόβουλους χρήστες. Οι εταιρίες που αγνοούν τη σημαντικότητα του MAPT , όχι μόνο ρισκάρουν να αντιμετωπίσουν οικονομική απώλεια, αλλά να έρθουν αντιμέτωποι και με νομικά ζητήματα και δυσφήμιση. Χαρακτηριστικό παράδειγμα είναι η υπόθεση της Under Armour, όταν από την εφαρμογή της – My Fitness Pal – διέρρευσαν 150 εκατομμύρια προσωπικά στοιχεία στο Dark Web [31].Αυτός είναι ο κύριος λόγος που το MAPT πρέπει να θεωρείται βασικό στάδιο στη γραμμή παραγωγής μιας εφαρμογής.

3.6 Μεθοδολογία Mobile Application Penetration Testing

Υπάρχουν πολλά πρότυπα μεθοδολογίας για το MAPT, αλλά η πιο γνωστή Mobile Application Penetration Testing Methodology(MAPTM) αποτελείται από 4 στάδια τα οποία θα αναπτυχθούν στην συνέχεια:

- Προετοιμασία
- Ανάλυση/Αξιολόγηση
- Εκμετάλλευση
- Αναφορά

1. Προετοιμασία

Συχνά αποκαλείται και στάδιο της Αναγνώρισης, καθώς σε αυτό το στάδιο ο pentester πρέπει να συλλέξει όλες τις κρίσιμες πληροφορίες που χρειάζονται για να εκμεταλλευτεί με επιτυχία την εφαρμογή του κινητού. Είναι ένα σημαντικό στάδιο, διότι αυτή η ικανότητα αναγνώρισης κρυφών στοιχείων και φαινομενικά ασήμαντων ευπαθειών μπορεί να είναι το κλειδί για την επίτευξη ενός επιτυχημένου pentest. Αυτό το στάδιο περιλαμβάνει τις εξής ενέργειες:

- Open Source Intelligence (OSINT): ο pentester αναζητά πληροφορίες από τα κοινωνικά μέσα δικτύωσης και τις μηχανές αναζήτησης, τυχόν «διαρροές» του πηγαίου κώδικα της εφαρμογής σε φόρουμ προγραμματιστών και ψάχνει τα repositories του πηγαίου κώδικα. Η αναζήτηση πληροφοριών μπορεί να γίνει ακόμα και στο Dark Web.
- Κατανόηση της Αρχιτεκτονικής: είναι σημαντικό για τον pentester να κατανοήσει την αρχιτεκτονική και μετέπειτα να σχηματίσει ένα threat model το οποίο θα το χρησιμοποιήσει σε αυτήν την αρχιτεκτονική/πλατφόρμα. Σε ένα ιδανικό test, ο tester θα πρέπει να λάβει υπόψη

την επιχείρηση που βρίσκεται πίσω από αυτήν την εφαρμογή, το επιχειρηματικό σχέδιο και σχετικούς κατόχους μετοχών.

- Client-side vs server-side σενάρια: ο pentester πρέπει να είναι σε θέση να αντιληφθεί τον τύπο της εφαρμογής (native, hybrid ή web), πριν ξεκινήσει τις δοκιμές. Επίσης, θα πρέπει να λάβει υπόψη τις διεπαφές δικτύου της εφαρμογής, τα δεδομένα του χρήστη, την επικοινωνία με άλλες εφαρμογές και τη συμπεριφορά σε jailbreak κινητά.[32]

2. Ανάλυση/Αξιολόγηση

Η αξιολόγηση των εφαρμογών για τις κινητές συσκευές είναι ξεχωριστή γιατί απαιτείται από τον pentester να ελέγξει τις εφαρμογές πριν και μετά την εγκατάστασή τους. Μερικές από τις τεχνικές αξιολόγησης της MAPTM είναι:

- Ανάλυση τοπικών αρχείων: μόλις εγκατασταθεί η εφαρμογή στο κινητό, δημιουργείται και ένας δικός της φάκελος στο σύστημα. Όταν χρησιμοποιείται η εφαρμογή, τα δεδομένα διαβάζονται και γράφονται σε αυτόν το φάκελο.
- Ανάλυση πακέτων: ο tester εξάγει τα APK και εξετάζει διεξοδικά τις παραμετροποιήσεις της εφαρμογής για τυχόν μετατροπές.
- Στατική ανάλυση: Κατά τη διάρκεια της στατικής ανάλυσης, ο tester δεν εκτελεί την εφαρμογή. Η ανάλυση γίνεται στα παρεχόμενα αρχεία ή ακόμα και στον απομεταγλωττισμένο(decompiled) πηγαίο κώδικα.
- Δυναμική ανάλυση: αυτού του είδους ανάλυση γίνεται, όταν η εφαρμογή εκτελείται. Περιλαμβάνει ενέργειες, όπως είναι η εγκληματολογική ανάλυση των αρχείων του συστήματος και η ανάλυση της κίνησης του δικτύου ανάμεσα στην εφαρμογή και το server.
- Reverse Engineering: είναι η μέθοδος κατά την οποία ο πηγαίος μεταγλωττισμένος κώδικας της εφαρμογής μετατρέπεται σε μορφή ικανή να την διαβάσει ο άνθρωπος. Έπειτα ο tester μελετάει τον κώδικα με σκοπό να καταλάβει τον τρόπο που λειτουργούν οι μέθοδοι της εφαρμογής και ψάχνει για ευπάθειες. Ο πηγαίος κώδικας της Android εφαρμογής μπορεί να μετατραπεί και να μεταγλωττιστεί ξανά.
- Παρακολούθηση της κίνησης δικτύου και διαδικτύου: σε αυτήν την τεχνική ο tester στήνει ένα proxy server και κάθε κίνηση που αποστέλλεται από την συσκευή προς τον server μπορεί να διακοπτεί, να προβληθεί και να μετατροποιηθεί. Επίσης, μπορεί να καταγραφθούν και να αναλυθούν τα UDP και TCP πακέτα.
- Inter-process communication (IPC) endpoint ανάλυση: Το λειτουργικό Android χρησιμοποιεί IPC μηχανισμούς προκειμένου να επιτυγχάνεται η επικοινωνία μεταξύ των διεργασιών. Σε αυτήν την ανάλυση εξετάζονται τα components της εφαρμογής (όπως είδαμε στην προηγούμενη ενότητα/κεφάλαιο) δηλαδή τα Activities, Content Providers, Services και Broadcast receivers.[33]

3. Εκμετάλλευση

Σε αυτό το στάδιο εκμεταλλεύονται οι τυχόν ευπάθειες που ανακάλυψε ο tester μαζεύοντας πληροφορίες από τα προηγούμενα δύο στάδια. Η διεξοδική συλλογή πληροφοριών εγγυάται μία επιτυχημένη εκμετάλλευση. Στη συνέχεια ο tester προσπαθεί να συλλέξει σημαντικές πληροφορίες εκτελώντας κακόβουλες δραστηριότητες και τελικά να κάνει privilege escalation. Με αυτήν τη

διαδικασία αποκτά ίδια αξία και δικαιώματα με τον πιο προνομιούχο χρήστη (δηλαδή τον χρήστη root), ώστε να μην υπάρχουν καθόλου περιορισμοί στις δραστηριότητες της εφαρμογής που εκτελείται.

Τέλος, ο tester δημιουργεί ένα backdoor στη συσκευή που του επιτρέπει την ικανότητα να συνδεθεί στο μέλλον όποτε ξανά επιθυμήσει.

4. Αναφορά

Το τελευταίο στάδιο της μεθοδολογίας του MAPT αποτελείται από τη δημιουργία μίας τελικής αναφοράς και της παρουσίασή της στον πελάτη. Αυτή η αναφορά θα πρέπει να περιλαμβάνει το σύνολο των ευπαθειών που ανακαλύφθηκαν και εκμεταλλεύτηκαν με επιτυχία. Ακόμα, θα πρέπει να είναι λεπτομερής και σε κατανοητή μορφή, περιλαμβάνοντας πληθώρα φωτογραφιών από τις ενέργειες του προηγούμενου σταδίου. Στην παρουσίασή της θα αναλυθούν τα ευρήματα που ανακάλυψαν οι testers και θα απαντηθούν οι τυχόν ερωτήσεις από τον πελάτη. Τέλος, οι testers θα παρέχουν λύσεις για τις ευπάθειες που εντόπισαν και καλύτερα μέτρα για την ενίσχυση της ασφάλειας της εφαρμογής.

Όπως αναφέρθηκε και πιο πριν, υπάρχουν πολλά πρότυπα MAPTM με τα ίδια ή περισσότερα στάδια στη διαδικασία. Έτσι, σε κάποιες περιπτώσεις, ανάμεσα στο πρώτο και δεύτερο στάδιο υπάρχει ένα ακόμα που συνήθως παραλείπεται, γιατί θεωρείται πολύ βασικό, το στάδιο της **προετοιμασίας του pentesting lab**. Η επιλογή του κατάλληλου pentesting lab περιβάλλοντος γίνεται με βάση τη συλλογή πληροφορίων που έγινε στο 1ο στάδιο. Ακόμα, σε άλλες περιπτώσεις μπορεί να υπάρχει ακόμα ένα στάδιο μετά το τελευταίο, το **Retest**. Αφού ο οργανισμός, χρησιμοποιήσει τις λύσεις που του παρείχε το 1ο test, ο καλύτερος τρόπος για να βεβαιωθούν πως οι νέες αλλαγές στην άμυνα είναι αποτελεσματικές είναι να γίνει το test ξανά.

3.7 Εργαλεία για Android Penetration Test

Πλατφόρμες ιδανικές για Mobile Application Security:

- **Appie**: είναι ένα φορητό πακέτο λογισμικού που έχει διαμορφωθεί για να λειτουργεί ως ένα Android Pentesting περιβάλλον σε οποιοδήποτε μηχάνημα διαθέτει Windows χωρίς την ανάγκη ύπαρξης Virtual Machine (VM) ή dualboot.
- **Android Tamer**: είναι μία Virtual / Live πλατφόρμα που επιτρέπει τους επαγγελματίες στην ασφάλεια Android να εκτελούν διάφορες ενέργειες, όπως είναι η ανάλυση κακόβουλου λογισμικού (Malware Analysis), penetration testing και reverse engineering.
- **AppUse**: είναι ένα VM που αναπτύχθηκε από τα AppSec Labs.
- **Androl4b**: είναι ένα VM βασισμένο στα Ubuntu.
- **Santoku**: είναι ένα λειτουργικό σύστημα το οποίο μπορεί να λειτουργεί και πέρα από ένα VM.
- **Vezir**: είναι ένα Linux VM ιδανικό για mobile application pentesting και mobile malware analysis.

All-in-One Mobile Security Frameworks:

- **Mobile Security Framework (MobSF)**: είναι ένα έξυπνο, ανοιχτού κώδικα (open source) αυτοματοποιημένο pen-testing framework ιδανικό για να εκτελεί στατική και δυναμική ανάλυση.
- **Objection**: είναι μία συλλογή εργαλείων που χρησιμοποιούνται κατά την εκτέλεση της εφαρμογής και υποστηρίζεται από τη Frida. Δημιουργήθηκε με στόχο τη βοήθεια της

αξιολόγησης της ασφάλειας των εφαρμογών για κινητά και τον έλεγχο του επιπέδου της ασφάλειας που διαθέτει η συσκευή χωρίς να είναι jailbroken ή rooted.[34]

Εργαλεία για Reverse engineering και Στατική Ανάλυση:

- **APKTool**: είναι ένα εργαλείο για reverse engineering. Μπορεί να αποδικωποιήσει τα τελικά αρχεία στη σχεδόν τους αρχική μορφή και να δημιουργήσει ξανά την εφαρμογή με μερικές μετατροπές.
- **Dex2jar**: είναι ένα εργαλείο που μετατρέπει τα αρχεία .dex σε classes.jar ή σε αρχεία .smali.
- **JD-GUI**: είναι ένα πρόγραμμα που συνήθως συνδυάζεται με το Dex2jar και επιτρέπει την ανάγνωση των αρχείων .class.
- **Sign**: το Sign.jar υπογράφει αυτόματα ένα αρχείο apk με ένα Android test πιστοποιητικό.
- **Bytecode Viewer**: είναι ένα ελαφρύ και open source εργαλείο γραμμένο σε Java και παρουσιάζει τα αρχεία Java σε ευανάγνωστη μορφή.
- **Jadx**: Απομεταγλωττίζει (decompiles) τα Dex αρχεία σε Java. Είναι ένα command line εργαλείο αλλά και διαθέτει GUI και παράγει Java αρχεία από αρχεία Apk και DEX.
- **Oat2dex**: μετατρέπει τα OAT αρχεία σε DEX.
- **Qark**: αυτό το εργαλείο σχεδιάστηκε με σκοπό να βρίσκει διάφορες ευπάθειες στην ασφάλεια των εφαρμογών Android, είτε βρίσκονται στον πηγαίο κώδικα είτε στο πακέτο
- **APK.AndroBugs**: το AndroBugs Framework είναι ένας αποτελεσματικός ανιχνευτής ευπαθειών που βοηθάει τους προγραμματιστές ή τους χάκερς να βρουν πιθανά κενά ασφάλειας στις Android εφαρμογές.
- **Simplify**: το simplify, αφού εκτελέσει εικονικά και προσπαθήσει να καταλάβει τη συμπεριφορά μιας εφαρμογής, βελτιστοποιεί τον κώδικα της, ώστε να λειτουργεί με τον ίδιο τρόπο και να γίνει κατανοητός από τον άνθρωπο. Δεν έχει σημασία ποιος τύπος obfuscation χρησιμοποιήθηκε.
- **Android backup extractor**: είναι μια υπηρεσία που εξάγει και δημιουργεί ξανά Android backups. Είναι βασισμένο σε μεγάλο βαθμό στη κλάση BackupManagerService.java από το Android Open Source Project(AOSP).

Εργαλεία για δυναμική ανάλυση:

- **Xposed Framework**: επιτρέπει την παραμετροποίηση του συστήματος ή της εφαρμογής κατά τη διάρκεια της εκτέλεσής της, χωρίς να χρειάζεται να παραμετροποιηθεί το APK της Android εφαρμογής.
- **Frida**: είναι μία συλλογή εργαλείων που χρησιμοποιούν το μοντέλο client-server και επιτρέπει την εισαγωγή κώδικα στις διαδικασίες που εκτελούνται όχι μόνο σε Android αλλά και σε iOS, Windows και Mac.
- **Diff-GUI**: είναι ένα διαδικτυακό Framework που χρησιμοποιεί την hook μέθοδο εισάγοντας JavaScript κώδικα με το Frida.
- **AndBug**: είναι ένας debugger που στοχεύει στην Dalvik Virtual machine (DVK) του Android, ιδανικός για προγραμματιστές και reverse engineers.
- **Drozer**: το Drozer αποτελεί μία από τις πιο σημαντικές εφαρμογές για την αναζήτηση ευπαθειών σε εφαρμογές. Διαθέτει μεγάλη ποικιλία εντολών που επιτρέπουν την ανακάλυψη ευπαθειών αλλά και την εκμετάλλευσή τους.

Εργαλεία για Server-Side testing:

- **Android tcpdump**: είναι μία command line υπηρεσία καταγραφής πακέτων δικτύου. Μπορεί να καταγράψει τα πακέτα σχεδόν από όλες τις συνδέσεις σε ένα Android κινητό.
- **Wireshark**: είναι ο πιο γνωστός και open-source αναλυτής πακέτων.
- **Burp Suite**: είναι μία ολοκληρωμένη πλατφόρμα που εκτελεί τεστ ασφάλειας στις εφαρμογές.
- **Proxydroid**: είναι μία παγκόσμια εφαρμογή Proxy για συστήματα Android.

Εργαλεία για την παράκαμψη root και για SSL pinning:

- **Xposed Module - Just Trust Me**: είναι ένα εργαλείο που χρησιμοποιείται για την παράκαμψη SSL πιστοποιητικού.
- **Xposed Module - SSL Unpinning**: είναι ένα Android module που παρακάμπτει την επαλήθευση του SSL πιστοποιητικού (Certificate Pinning).
- **Cydia Substrate Module - Android SSL Trust Killer**: είναι ένα blackbox εργαλείο που παρακάμπτει το SSL πιστοποιητικό στις περισσότερες εφαρμογές που τρέχουν σε μια συσκευή.
- **Cydia Substrate Module - RootCoak Plus**: αποτρέπει τις εφαρμογές από τον εντοπισμό του root.[35]

Security Libraries:

- **Java AES Crypto**: είναι μία απλή Android κλάση που κρυπτογραφεί και αποκρυπτογραφεί strings, με σκοπό την αποφυγή των κλασσικών λαθών που κάνουν οι κλάσεις.
- **Proguard**: είναι ένα δωρεάν πρόγραμμα που συρρικνώνει, βελτιστοποιεί, obfuscate αρχεία Java κλάσεων. Εντοπίζει και αφαιρεί τις κλάσεις, τα πεδία, τις μεθόδους και τις μεταβλητές που δεν χρησιμοποιούνται.
- **SQL Cipher**: είναι μία προέκταση της open source SQLite βιβλιοθήκης η οποία παρέχει ένα AES κλειδί μήκους 256-bit με σκοπό την κρυπτογράφηση των αρχείων της βάσης δεδομένων.

Κεφάλαιο 4ο: OWASP

4.1 Τι είναι ο OWASP;

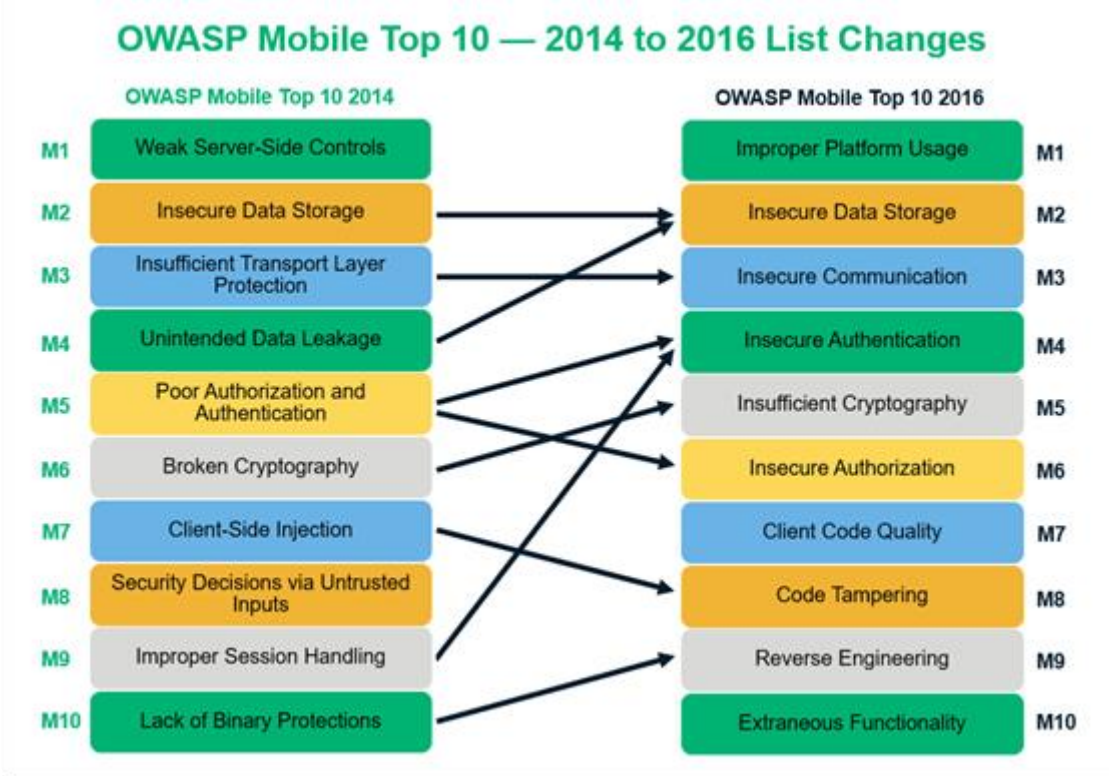
Ο OWASP (Open Web Application Security Project) είναι μία μη κερδοσκοπική παγκόσμια οργάνωση που ξεκίνησε το 2001 με σκοπό την παροχή βοήθειας στους δημιουργούς ιστοσελίδων και την προσφορά συμβουλών και μεθοδολογιών για τη βελτίωση της ασφάλειας των λειτουργικών συστημάτων και των εφαρμογών[36]. Καθώς είναι μη-κερδοσκοπικός οργανισμός, όλοι οι πόροι του (συμπεριλαμβανομένου άρθρα , μεθοδολογίες, εργαλεία και τεχνολογίες) είναι διαθέσιμοι δωρεάν και εύκολα προσβάσιμοι σε οποιονδήποτε θέλει να έχει ένα ικανοποιητικό επίπεδο ασφάλειας στην εφαρμογή του.

Προκειμένου να αυξηθεί η ασφάλεια και η ετοιμότητα των προγραμματιστών ο OWASP δημιουργεί τις “top 10 lists” οι οποίες περιγράφουν τους κορυφαίους κινδύνους σε web-mobile εφαρμογές και ανανεώνονται ανά τακτά χρονικά διαστήματα. Πέρα από τις δημοφιλείς λίστες του ο OWASP είναι γνωστός και για: το OWASP SAMM (Software Assurance Maturity Model) το οποίο παρέχει στρατηγικές για την αντιμετώπιση κινδύνων σε επιχειρήσεις, τον OWASP Development Guide που αφορά την ανάπτυξη κώδικα, τον OWASP Testing Guide, ένας οδηγός σωστών πρακτικών για penetration test και τον OWASP Code Review Guide στον οποίο περιγράφονται οδηγίες για τον έλεγχο του κώδικα.

4.2 OWASP mobile project

Το OWASP mobile project είναι ένα προτζεκτ που αναφέρεται στους προγραμματιστές εφαρμογών και στις ομάδες ασφάλειας, με σκοπό να δημιουργήσουν και να διατηρήσουν μια ασφαλής κινητή εφαρμογή. Το πρότζεκτ ανανεώνεται συχνά με τα πιο πρόσφατα είδη επιθέσεων που υπάρχουν και παρέχει λύσεις για να μειωθεί η πιθανότητα εμφανίσεων τους στις κινητές συσκευές. Περιέχει έναν testing guide , την OWASP mobile top 10 list και διάφορες σημειώσεις για την ασφαλή ανάπτυξη και θωράκιση μιας εφαρμογής[37].

Η OWASP mobile top 10 list περιλαμβάνει τα 10 πιο συχνά λάθη στην ασφάλεια των κινητών συσκευών. Όπως βλέπετε και στο **Σχήμα 4.1** , η πρώτη λίστα δημοσιεύθηκε το 2014 και ανανεώθηκε το 2016.



Σχήμα 4.1: Σύγκριση λιστών OWASP 2014 και 2016

Πηγή[38]

4.3 TOP 10 risk list

Παρακάτω θα παρουσιαστούν και θα αναλυθούν οι κορυφαίοι δέκα κίνδυνοι όπως ορίζει ο OWASP για τις κινητές συσκευές[39]:

- M1: Improper Platform Usage
- M2: Insecure Data Storage
- M3: Insecure Communication
- M4: Insecure Authentication
- M5: Insufficient Cryptography
- M6: Insecure Authorization
- M7: Client Code Quality
- M8: Code Tampering
- M9: Reverse Engineering
- M10: Extraneous Functionality

- M1: Improper Platform Usage

  Τα λειτουργικά συστήματα και οι πλατφόρμες τους, όπως είναι το iOS και το Android, προσφέρουν μία τεράστια ποικιλία λειτουργιών. Η μη ασφαλής χρήση τους και η κακή ανάπτυξη κώδικα μπορεί να οδηγήσει σε διάφορες ευπάθειες, όπως είναι η είναι η λάθος χρήση των Android Intents , Permissions κτλ.

Για το λειτουργικό σύστημα του Android, η συγκεκριμένη ευπάθεια προκύπτει συνήθως από τα Android Intents. Τα Intents είναι μηνύματα που αποστέλλονται μεταξύ των components(πχ. Activities με Services κ.ά.). Η εκμετάλλευση των Android Intents μπορεί να οδηγήσει σε διαρροή εμπιστευτικών δεδομένων του χρήστη ή να επιτρέψει τη μη εξουσιοδοτημένη πρόσβαση. Πιο συγκεκριμένα θα πρέπει να δοθεί μεγάλη προσοχή αν κάποια από τα tags όπως: *exported:true, android:allowBackup, android:debuggable* κ.ά. που βρίσκονται στο AndroidManifest.xml αρχείο έχουν την τιμή true.

Μερικές από τις λύσεις που προτείνει ο OWASP για αυτήν την ευπάθεια είναι οι προγραμματιστές να:

- Ακολουθούν τις οδηγίες των διάφορων πλατφόρμων για την ανάπτυξη εφαρμογών.
- Χρησιμοποιούν τις βέλτιστες τεχνικές για τη χρήση λειτουργιών όπως, είναι τα Android Intents, Touch ID, iOS keychain κτλ.
- Περιορίζουν τις εφαρμογές από το ανταλλάζουν τα δεδομένα του χρήστη μεταξύ τους.
- Περιορίζουν τα δικαιώματα και τα αρχεία που ζητάει πρόσβαση να έχει η εφαρμογή.
- Κρυπτογραφούν τα δεδομένα στον αποθηκευτικό χώρο.

- M2: Insecure Data Storage

Το Insecure Data Storage είναι ο πιο απειλητικός κίνδυνος που μπορεί να συμβεί στις εφαρμογές για τα κινητά, στις διαδικτυακές εφαρμογές, στις IoT συσκευές κτλ. Σχεδόν κάθε εφαρμογή αποθηκεύει τα δεδομένα του χρήστη γνωστά ως και PII(Personally Identifiable Information). Εάν αυτά τα δεδομένα πέσουν στα λάθος χέρια , οι επιπτώσεις μπορεί να είναι πολλές. Αυτή η πρόσβαση στα δεδομένα μπορεί να γίνει με 2 τρόπους: είτε μέσω φυσικής πρόσβασης(π.χ. κλοπή συσκευής) είτε μέσω malware. Ο κύριος λόγος για την ύπαρξη αυτής της ευπάθειας είναι η άγνοια των προγραμματιστών στο που θα πρέπει να αποθηκεύουν σωστά τα δεδομένα του χρήστη.

Ένα κλασσικό παράδειγμα αυτής της ευπάθειας είναι όταν μια εφαρμογή αποθηκεύει στη βάση δεδομένων της, τους κωδικούς και τα username των χρηστών της σε απλό κείμενο. Κάποιος κακόβουλος χρήστης μπορεί εύκολα να παρατηρήσει τα logs ή να ψάξει τους αποθηκευτικούς χώρους της κινητής συσκευής (/data/data/«όνομα εφαρμογής», /sdcard/ κτλ.) για την εύρεση τέτοιων αρχείων.

Μερικές από τις λύσεις που προτείνει ο OWASP για αυτήν την ευπάθεια είναι οι προγραμματιστές να:

- Κρυπτογραφούν τα δεδομένα.
- Εφαρμόζουν στις εφαρμογές για κινητά μηχανισμούς αυθεντικοποίησης και εξουσιοδότησης.
- Περιορίζουν την πρόσβαση των εφαρμογών στον αποθευτικό χώρο.
- Αποφεύγουν να στέλνουν στατιστικές αναλύσεις σε τρίτους.
- Εφαρμόζουν καλές τεχνικές ασφάλειας σε περιπτώσεις buffer overflow, καταγραφής και αποθήκευσης προσωρινών δεδομένων(data logging και chaching), χρήση cookies κτλ.

- M3: Insecure Communication

  Πολλές εφαρμογές ανταλλάσσουν δεδομένα με βάση το μοντέλο client-server. Όταν τα δεδομένα αυτά «ταξιδεύουν» στο δίκτυο μη κρυπτογραφημένα σε απλό κείμενο, οποιοσδήποτε παρακολουθεί την κίνηση του δικτύου μπορεί να καταγράψει και να διαβάσει όλες τις πληροφορίες που αποστέλλονται. Οι προγραμματιστές σπανίως δίνουν προσοχή στην κρυπτογράφηση της μεταφοράς των δεδομένων, με αποτέλεσμα να μπορεί κανείς να υποκλέψει τα δεδομένα επικοινωνίας.

  Για παράδειγμα, μια εφαρμογή μεταδίδει τα PII ενός χρήστη σε ένα endpoint μέσω ενός μη-ασφαλούς channel και όχι με SSL. Οποιοσδήποτε βρίσκεται μέσα στο ίδιο Wi-Fi ή έχει εγκαταστήσει κακόβουλο λογισμικό στη συσκευή του θύματος ή εκτελεί Man-In-The-Middle επίθεση μπορεί εύκολα να εντοπίσει και να διαβάσει αυτά τα δεδομένα.

  Μερικές από τις λύσεις που προτείνει ο OWASP για αυτήν την ευπάθεια είναι οι προγραμματιστές να:

  - Εφαρμόζουν SSL/TLS πιστοποιητικά για ασφαλή επικοινωνία.
  - Χρησιμοποιούν εμπιστευτικά και υπογεγραμμένα CA (Certificate Authority) πιστοποιητικά.
  - Αποφεύγουν να στέλνουν το session user ID με SSL session token.
  - Κρυπτογραφούν τα δεδομένα εάν μπορούν πριν μεταφερθούν με SSL channel.
  - Χρησιμοποιούν standard πρωτόκολλα κρυπτογράφησης.
  - Εφαρμόζουν SSL/TLS στη μεταφορά ευαίσθητων πληροφοριών, session tokens ή άλλα σημαντικά δεδομένα μεταξύ εφαρμογών και backend API ή διαδικτυακής υπηρεσίας.

- M4: Insecure Authentication

  Αυτή η περίπτωση προκύπτει όταν η εφαρμογή αποτυγχάνει να επαληθεύσει την ταυτότητα του χρήστη. Για να επιτύχει ένας κακόβουλος χρήστης τη μη εξουσιοδοτημένη πρόσβασή του, εκμεταλλεύεται τις ευπάθειες που υπάρχουν στα αιτήματα που γίνονται στη μεριά του server χρησιμοποιώντας ειδικά εργαλεία που κάνουν bypass την αυθεντικοποίηση. Οι εφαρμογές για κινητά πρέπει να επαληθεύουν και να διατηρούν την ταυτότητα του χρήστη, ειδικά όταν μεταδίδουν εμπιστευτικά δεδομένα όπως είναι συναλλαγές στην τράπεζα.

  Ένα σενάριο είναι μια εφαρμογή να απαιτεί από τον χρήστη να δημιουργήσει ένα 4-ψήφιο PIN για να έχει πρόσβαση σε αυτήν. Ο κώδικας από τη μεριά του server αποθηκεύει την hashed μορφή αυτού του PIN. Ωστόσο, λόγω του εξαιρετικά μικρού μήκους του PIN , ένας κακόβουλος χρήστης μπορεί εύκολα να μαντέψει αυτό το PIN με τη βοήθεια των rainbow hash tables.

  Μερικές από τις λύσεις που προτείνει ο OWASP για αυτήν την ευπάθεια είναι οι προγραμματιστές να:

  - Μην αποθηκεύουν τους κωδικούς τοπικά στις κινητές συσκευές.
  - Εφαρμόζουν μηχανισμούς αυθεντικοποίησης από την πλευρά του server.
  - Εξασφαλίζουν τη διαθεσιμότητα των δεδομένων μετά από την επιτυχή αυθεντικοποίηση.
  - Προστατεύουν την εφαρμογή από binary attack.
  - Προειδοποιούν τον χρήστη όταν διαλέγει την επιλογή «Remember me».

- Μην επιτρέπουν τον χρήστη να δημιουργεί ένα PIN με 4 ψηφία σαν κωδικό αυθεντικοποίησης.

- M5: Insufficient Cryptography

  Σε αυτήν την περίπτωση ο επιτιθέμενος εκμεταλλεύεται την αδυναμία των μηχανισμών κρυπτογράφησης ή γενικά τους αδύναμους αλγόριθμους κρυπτογράφησης. Αυτό έχει σαν αποτέλεσμα ένας κακόβουλος χρήστης να έχει πρόσβαση σε ευαίσθητες και προσωπικές πληροφορίες όπως είναι το όνομα χρήστη, η τοποθεσία, λεπτομέρειες της πιστωτικής κάρτας, PII, PHI κ.ά. Οι ξεπερασμένοι αλγόριθμοι κρυπτογράφησης , όπως είναι ο RC2, MD4, MD5 και ο SHA1 θα πρέπει να σταματήσουν να εφαρμόζονται.

  Ένας κακόβουλος χρήστης μπορεί μέσα σε λίγα δευτερόλεπτα να αποκρυπτογραφήσει τα δεδομένα καθώς μεταφέρονται στο δίκτυο ή να εγκαταστήσει ένα κακόβουλο λογισμικό στη συσκευή έχοντας πρόσβαση στα κρυπτογραφημένα δεδομένα.

  Οι λύσεις που προτείνει ο OWASP για αυτήν την ευπάθεια είναι, οι προγραμματιστές να:

  - Αποφεύγουν να χρησιμοποιούν σημαντικά δεδομένα στις κινητές συσκευές.
  - Αποφεύγουν να χρησιμοποιούν ξεπερασμένους αλγόριθμους κρυπτογράφησης όπως, είναι ο RC2, MD4, MD5 και SHA1.
  - Ακολουθούν τις οδηγίες του NIST σχετικά με προτεινόμενους αλγόριθμους [40].

- M6: Insecure Authorization

  Η διαδικασία της εξουσιοδότησης (authorization) του χρήστη γίνεται συνήθως παράλληλα με τη διαδικασία της αυθεντικοποίησης. Η περίπτωση του Insecure Authorization προκύπτει όταν ένας κακόβουλος χρήστης εκμεταλλεύεται τις ευπάθειες και τους αδύναμους ελέγχους που πραγματοποιούνται στην εφαρμογή, αλλά από τη μεριά του server. Αφού αποκτήσουν πρόσβαση στην εφαρμογή οι hackers και συνδεθούν ως απλοί χρήστες, μπορούν εύκολα να εκτελέσουν μια επίθεση κλιμάκωσης προνομίων (privilege escalation attack).

  Για παράδειγμα, ένας χρήστης δημιουργεί ένα αίτημα API endpoint με σκοπό να το στείλει στο backend REST API, συμπεριλαμβάνοντας το userID του και το OAuth token του. Και το userID και το OAuth token είναι μέρος του URL που αποστέλλεται και λαμβάνεται από τον χρήστη. Το backend επαληθεύει την εγκυρότητα του token, αλλά αποτυγχάνει να το επικυρώσει μαζί με το userID. Ως αποτέλεσμα, ο χρήστης μπορεί να τροποποιήσει το userID και να αποκτήσει πληροφορίες για άλλους χρήστες χρησιμοποιώντας το δικό του αρχικό REST API αίτημα.

  Μερικές από τις λύσεις που προτείνει ο OWASP για αυτήν την ευπάθεια είναι:

  - Η έγκριση ρόλων και δικαιωμάτων ενός αυθεντικοποιημένου χρήστη θα πρέπει να γίνεται στα backend συστήματα της εφαρμογής και όχι στην κινητή συσκευή.
  - Ο backend κώδικας θα πρέπει να επαληθεύει σωστά κάθε ένα αίτημα που δέχεται.

- M7: Client Code Quality

Η περίπτωση αυτή προκύπτει λόγω της χρήσης κακών πρακτικών στον προγραμματισμό κατά την ανάπτυξη των εφαρμογών για τα κινητά. Αυτές οι αδυναμίες στον κώδικα μπορούν να εμφανιστούν τόσο από μεριάς client αλλά και από του server. Οι πιο γνωστές ευπάθειες που μπορεί να εμφανιστούν είναι τα memory leaks, buffer overflows, remote code execution(RCE) κ.ά.

Ένα πιθανό σενάριο τέτοιας ευπάθειας είναι ο παρακάτω κώδικας τον οποίο παρέχει ο OWASP [41]:

```c
include <stdio.h>

int main(int argc, char **argv)
{
   char buf[8]; // buffer για 8 χαρακτήρες
   gets(buf); // διαβάζει από το stdio (επικίνδυνη συνάρτηση!)
   printf("%s\n", buf); // τύπωσε τα δεδομένα που είναι αποθηκευμένα στο buf
   return 0; // επιστρέφει την τιμή 0
}
```

Σχήμα 4.2: Παράδειγμα κώδικα ευπάθειας buffer overflow σε C

Το παραπάνω απλό πρόγραμμα είναι χαρακτηριστικό παράδειγμα ευπάθειας buffer overflow. Το πρόγραμμα καλεί μια μέθοδο που συσχετίζεται με το buffer ενός char τύπου και δεν ελέγχει την υπερχείλιση του μεγέθους που έχει εκχωρηθεί σε αυτό το buffer. Ως αποτέλεσμα , είναι δυνατή η ακούσια αποθήκευση περισσότερων δεδομένων στο buffer, προκαλώντας σφάλμα στην οργάνωση της μνήμης. Οι χαρακτήρες που θα περισσέψουν θα αντικαταστήσουν την τιμή σε έναν από τους άλλους καταχωρητές.

Μερικές από τις λύσεις που προτείνει ο OWASP για αυτήν την ευπάθεια είναι οι προγραμματιστές να:

- Εφαρμόζουν ασφαλείς πρακτικές στην ανάπτυξη κώδικα.
- Χρησιμοποιούν εργαλεία στατικής ανάλυσης κατά τη διαδικασία ανάπτυξης της εφαρμογής.
- Γράφουν κατανοητό και ευανάγνωστο κώδικα.
- Χρησιμοποιούν εμπιστευτικές και ασφαλείς 3rd-party βιβλιοθήκες.
- Ελέγχουν για περιπτώσεις buffer overflow, memory leak και remote code execution (RCE) με τη χρήση αυτόματου εργαλείου.

- M8: Code Tampering

Εκμεταλλεύοντας την ευπάθεια Code Tampering , ο επιτιθέμενος μετατρέπει τον κώδικα της εφαρμογής και δημιουργεί μία νέα ψευδής έκδοσή της με σκοπό να ξεγελάσει τον απλό χρήστη και να την κατεβάσει χρησιμοποιώντας διάφορες τεχνικές του social engineering. Ίσως η πιο σημαντική τροποποίηση που μπορεί να γίνει είναι η απευθείας προσθήκη malware στα binaries ή στα resources.

Οι τραπεζικές εφαρμογές είναι ένας δημοφιλής στόχος για μια τέτοια επίθεση. Αυτές οι εφαρμογές τυπικά επεξεργάζονται σημαντικές πληροφορίες από διάφορους χρήστες. Ένας

κακόβουλος χρήστης μπορεί να δημιουργήσει μια ψεύτικη έκδοση της εφαρμογής η οποία στέλνει το username και το password ενός χρήστη στον δικό του ιστότοπο. Το συγκεκριμένο παράδειγμα έχει πολλές ομοιότητες με το κακόβουλο λογισμικό Zeus [42].

Οι λύσεις που προτείνει ο OWASP για τις συσκευές Android αφορούν κυρίως στον εντοπισμό αν η συσκευή είναι root ή jailbroken [43]. Έτσι, προτείνει στους προγραμματιστές να ελέγχουν:

- Τα test-keys.
- Τα OTA πιστοποιητικά.
- Εάν το build.prop περιλαμβάνει τη γραμμή ro.build.tags=test-keys.
- Εάν υπάρχει ο φάκελος /etc/security/otacerts.zip
- Για SU binaries όπως είναι τα /system/bin/su, /system/xbin/su, /sbin/su, /system/su, /system/bin/.ext/.su κ.ά.
- Για πολλά γνωστά rooted APK's π.χ. com.noshufou.android.su, com.thirdparty.superuser, eu.chainfire.supersu κ.ά.

- M9: Reverse Engineering

Η περίπτωση του Reverse Engineering αναφέρεται στην διαδικασία αποδόμησης του αρχείου της εφαρμογής με σκοπό να αποκαλυφθεί η αρχιτεκτονική της, ο κώδικας κ.ά. Ο επιτιθέμενος, αφού μελετήσει τον κώδικα της εφαρμογής και κατανοήσει το πώς λειτουργεί , μπορεί με ειδικά εργαλεία να αλλάξει τον κώδικα αφαιρώντας ή προσθέτοντας δικές του γραμμές κώδικα. Ωστόσο, κάτι τέτοιο μπορεί να πραγματοποιηθεί και από έναν μη καλόβουλο χρήστη για ερευνητικούς σκοπούς ή για penetration testing (ethical hacking).

Για παράδειγμα, ένα APK αρχείο μπορεί εύκολα να αποσυμπιεστεί με κάποιο εργαλείο όπως είναι το apktool, 7zip, Winrar κ.ά. Μόλις αποσυμπιεστεί, ένας κακόβουλος χρήστης έχει πρόσβαση στο manifest αρχείο, στα assets, στα resources και κυρίως στο classes.dex αρχείο. Έπειτα, μπορεί εύκολα να μετατρέψει το DEX αρχείο σε JAR και στη συνέχεια με τη χρήση ενός Java Decompiler να έχει πρόσβαση στον πηγαίο κώδικα.

Μερικές από τις λύσεις που προτείνει ο OWASP για αυτήν την ευπάθεια είναι οι προγραμματιστές να:

- Εφαρμόζουν καλές τεχνικές obfuscation ακόμα και στα metadata.
- Χρησιμοποιούν C ή C++ βιβλιοθήκες για να προστατεύουν τις λειτουργίες κατά την εκτέλεση της εφαρμογής.
- Εφαρμόζουν το binary packaging με σκοπό να αποτρέψουν έναν κακόβουλο χρήστη να μελετήσει τον κώδικα απομεταγλωττίζοντάς τον εύκολα.
- Εφαρμόζουν κατάλληλους μηχανισμούς με σκοπό την αποφυγή debugging από εργαλεία όπως είναι το IDA Pro και το Hopper.

- M10: Extraneous Functionality

Η περίπτωση του Extraneous Functionality προκύπτει από τις πρόσθετες λειτουργίες που αρχικά δημιουργούνται για λόγους ευκολίας χρήσης της εφαρμογής και δεν αφαιρούνται από τους developers.

Ένα σχετικό παράδειγμα είναι η διατήρηση «developer account» το οποίο έχει πρόσβαση σε όλες τις λειτουργίες και ουσιαστικά μπορεί να χρησιμοποιηθεί ως backdoor για έναν κακόβουλο χρήστη.

Ένα πιθανό σενάριο είναι όταν ένας κακόβουλος χρήστης προσθέτει την εντολή "debug=true" σε ένα «.properties» αρχείο σε μια τοπική εφαρμογή. Κατά την εκκίνηση της εφαρμογής, εμφανίζονται αναλυτικές πληροφορίες που βοηθούν τον επιτιθέμενο να κατανοήσει πλήρως το backend σύστημα. Ο επιτιθέμενος στη συνέχεια ανακαλύπτει τις πιθανές ευπάθειες ως αποτέλεσμα της ανάγνωσης του log αρχείου. Σε αυτήν την περίπτωση οι προγραμματιστές θα πρέπει να είχαν απενεργοποιήσει το "debug mode" πριν την επίσημη κυκλοφορία της εφαρμογής.

Ο καλύτερος τρόπος που προτείνει ο OWASP για αυτήν την ευπάθεια είναι να εκτελεστεί μια επαγγελματική ανάλυση κώδικα ελέγχοντας:

- Τις ρυθμίσεις παραμετροποίησης(configurations) της εφαρμογής.
- Τον test κώδικα ο οποίος δεν συμπεριλαμβάνεται στην τελική έκδοση.
- Όλα τα API endpoints που έχει πρόσβαση η εφαρμογή και διαθέσιμα στο κοινό.
- Τα logs της εφαρμογής μήπως παρέχονται πληροφορίες για το backend σύστημα.

Κεφάλαιο 5ο: Δημιουργία Penetration Test Περιβάλλοντος

5.1 Εγκατάσταση Kali Linux

Για τη διεξαγωγή penetration test αναφερόμαστε σε Android, iOS ή web εφαρμογές. Σημαντικό στοιχείο είναι η δημιουργία περιβάλλοντος που θα διαθέτει όλα τα κατάλληλα εργαλεία για τις απαιτούμενες ενέργειες του test.

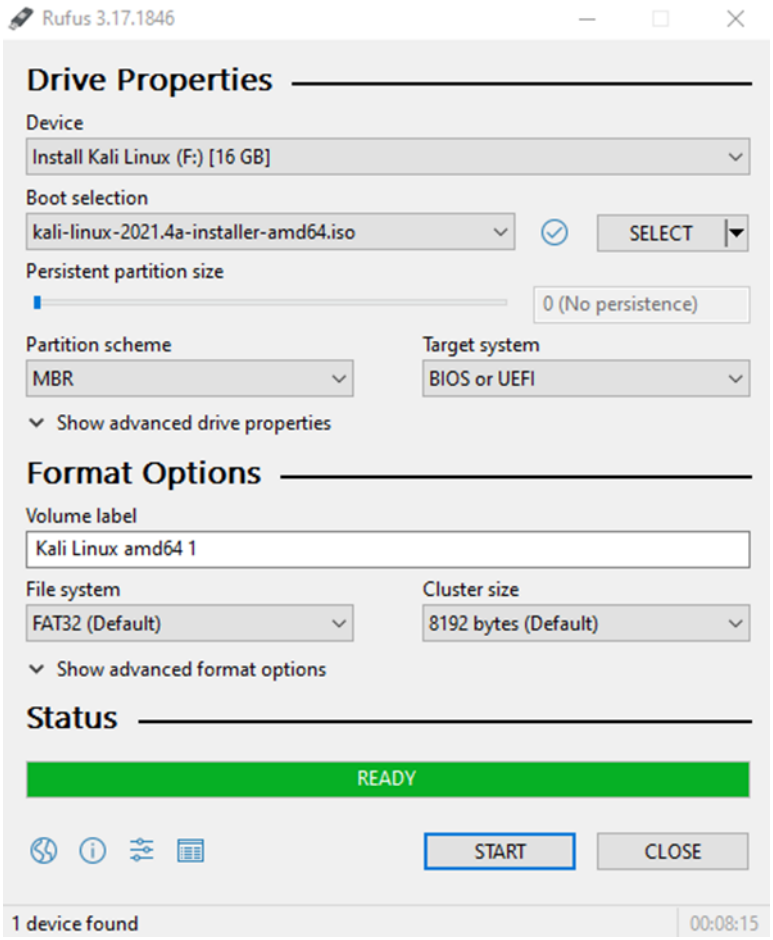
Όπως αναφέρθηκε και σε προηγούμενη ενότητα υπάρχουν πολλές πλατφόρμες ιδανικές για mobile penetration testing που έχουν ήδη προεγκατεστημένα αρκετά εργαλεία για mobile pentesting. Αλλά σε αυτήν τη διπλωματική εργασία επιλέχθηκε ως λειτουργικό σύστημα το Kali Linux, καθώς θα δείξουμε τα βήματα εγκατάστασης των εργαλείων που θα χρησιμοποιήσουμε.

Η έκδοση που χρησιμοποιήσαμε ήταν η Kali Linux 2021.4a λόγω του ότι αυτή ήταν η τελευταία έκδοση του Kali για τη χρονική στιγμή της εκπόνησης της εργασίας.

Για την ομαλή εγκατάσταση του Kali Linux τα προτεινόμενα χαρακτηριστικά που πρέπει να έχει ένα μηχάνημα είναι:

- 2GB of RAM
- 20 GB Hard Drive
- Πρόσβαση στο διαδίκτυο
- USB Port για την εγκατάσταση

Αρχικά, πλοηγηθήκαμε στην επίσημη σελίδα του Kali Linux (https://www.kali.org/get-kali/#kali-bare-metal) ώστε να κατεβάσουμε το κατάλληλο λογισμικό για 64-bit και σε μορφή «.iso». Έπειτα, μέσω του λογισμικού «Rufus» (https://rufus.ie/en/) χρησιμοποιήσαμε ένα κενό USB ώστε να το μετατρέψουμε σε bootable.



Σχήμα 5.1: Χρήση λογισμικού Rufus

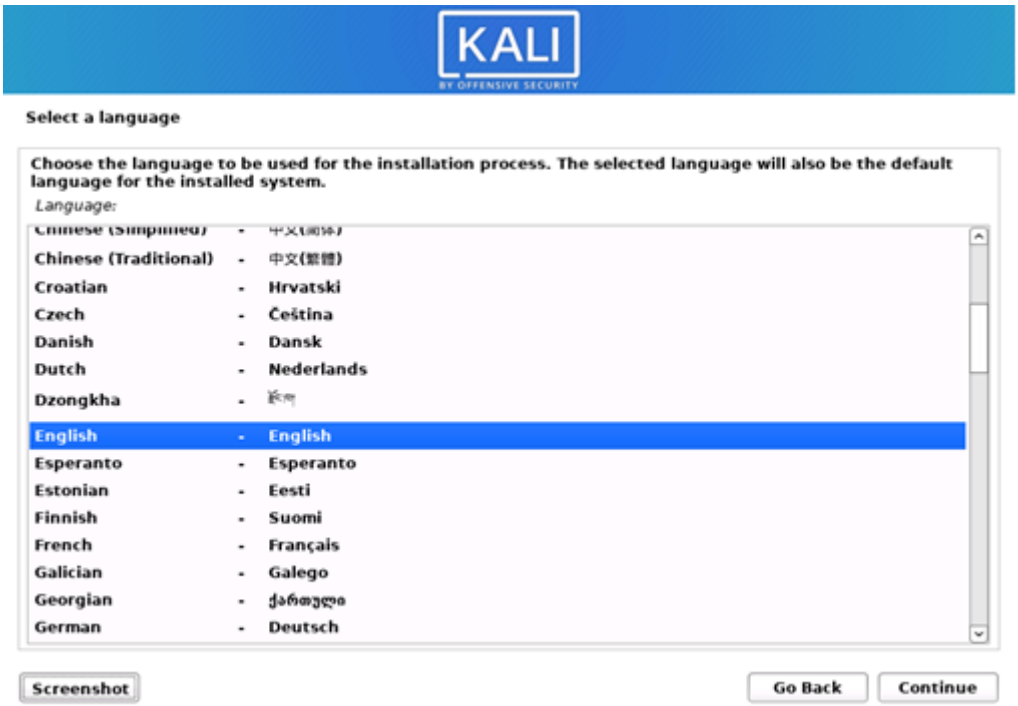
Μετά την ολοκλήρωση της παραπάνω διαδικασίας, ενεργοποιήσαμε το μηχάνημα στο οποίο επιθυμούσαμε να πραγματοποιήσουμε την εγκατάσταση του Kali. Κατά την είσοδό μας στο BIOS επιλέξαμε να κάνουμε boot από το USB ώστε να ξεκινήσει η διαδικασία εγκατάστασης του λογισμικού.

Παρακάτω, παραθέτονται τα στιγμιότυπα από κάθε βήμα εγκατάστασης του λογισμικού:

1) Διαλέγουμε την επιλογή “Graphical install”.

Σχήμα 5.2: Εγκατάσταση Kali Linux – Βήμα 1ο

2) Επιλέγουμε τη γλώσσα που επιθυμούμε να έχει η εγκατάσταση και το σύστημα.

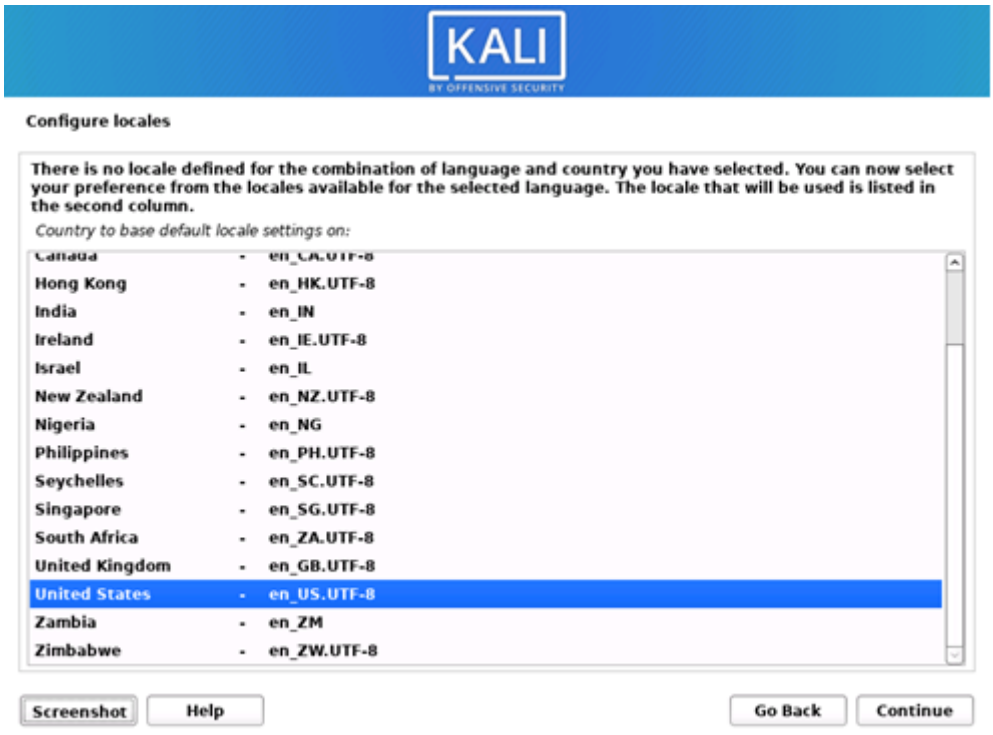


Σχήμα 5.3: Εγκατάσταση Kali Linux – Βήμα 2ο

3) Ακολούθως επιλέγουμε την τοποθεσία μας.

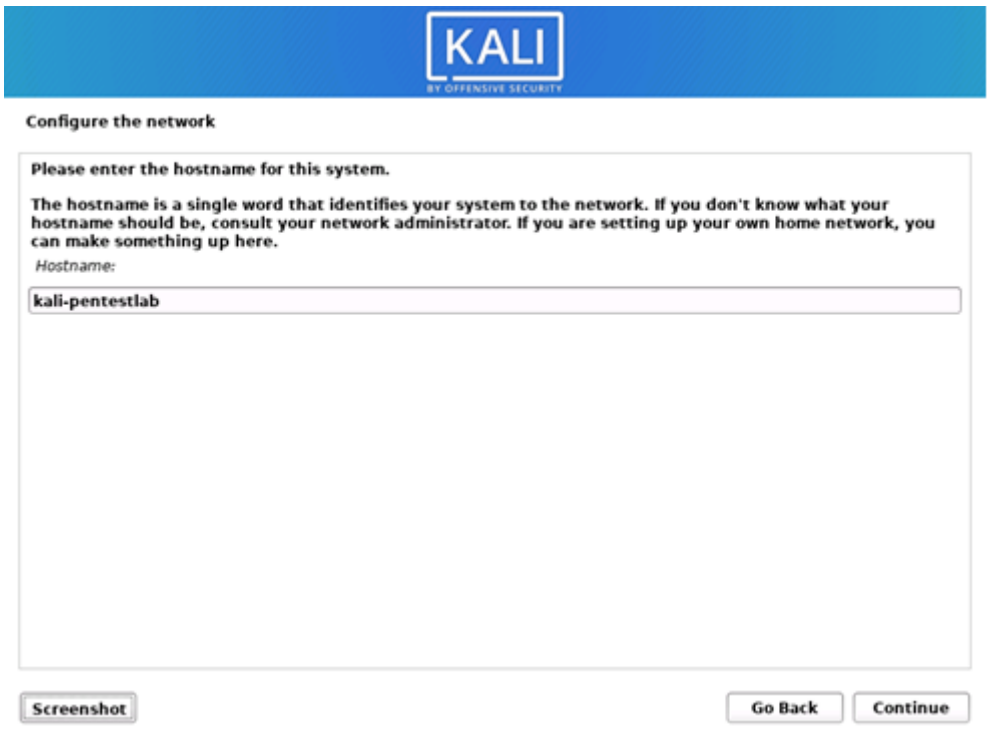
Σχήμα 5.4: Εγκατάσταση Kali Linux – Βήμα 3ο

4) Επιλέγουμε το layout που θα έχει το πληκτρολόγιο και τη γλώσσα.



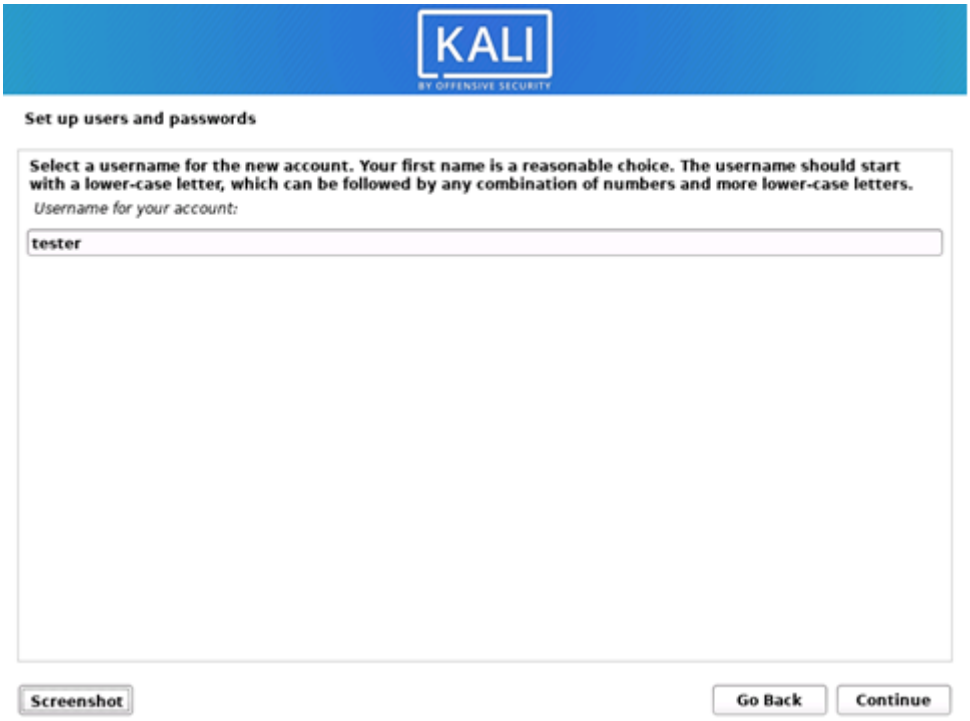
Σχήμα 5.5: Εγκατάσταση Kali Linux – Βήμα 4ο

5) Εισάγουμε το hostname που θα έχει το σύστημα και στο επόμενο βήμα στο domain name βάζουμε κενό.

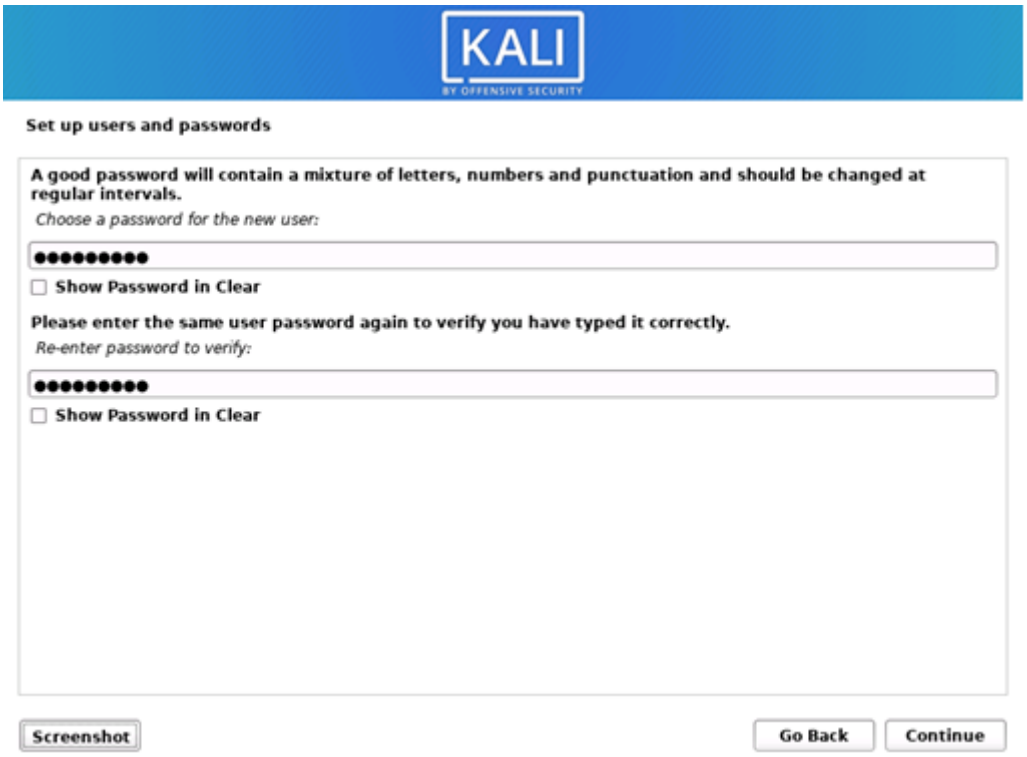


Σχήμα 5.6: Εγκατάσταση Kali Linux – Βήμα 5ο

6) Στη συνέχεια δημιουργούμε ένα λογαριασμό βάζοντας username και έναν δυνατό κωδικό.

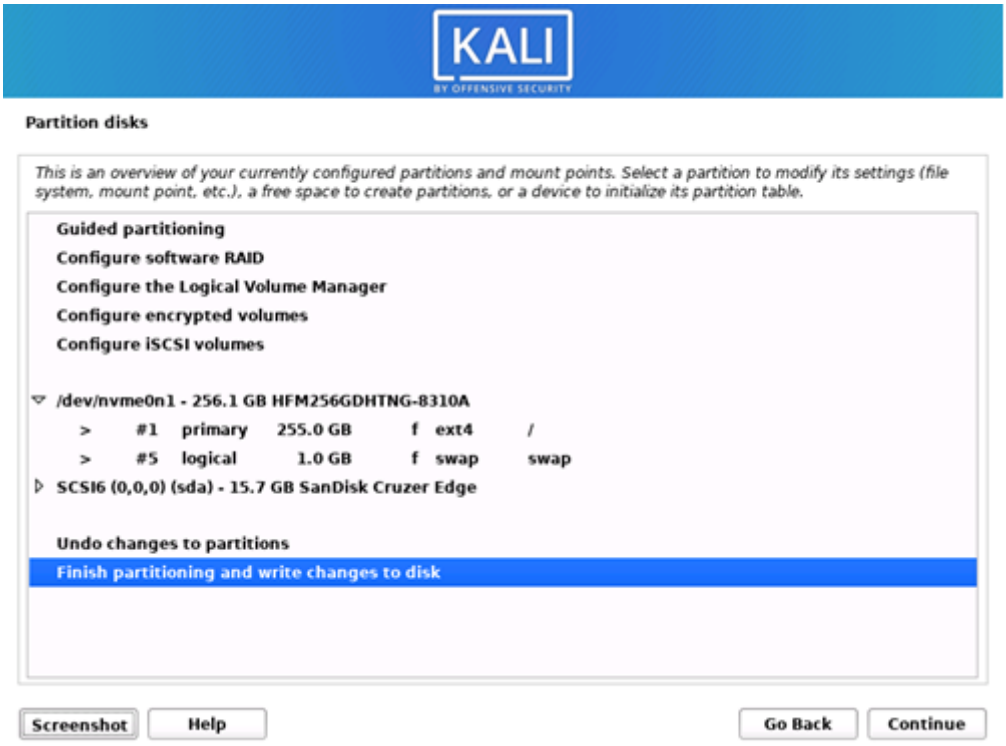


Σχήμα 5.7: Εγκατάσταση Kali Linux – Βήμα 6ο



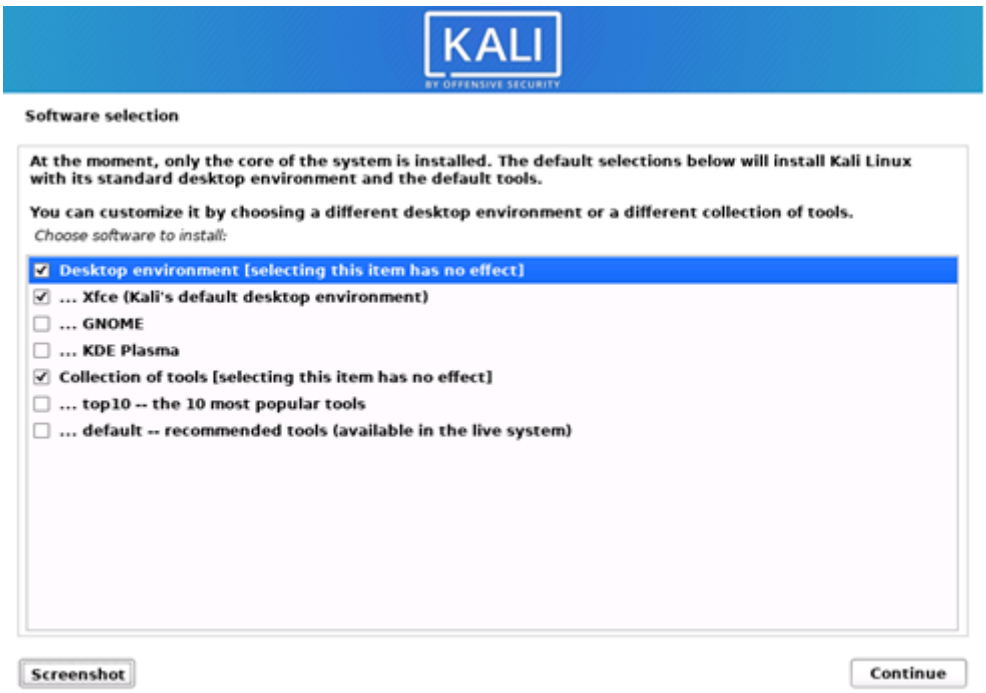
Σχήμα 5.8: Εγκατάσταση Kali Linux – Βήμα 6ο

7) Διαμορφώνουμε κατάλληλα τον δίσκο.



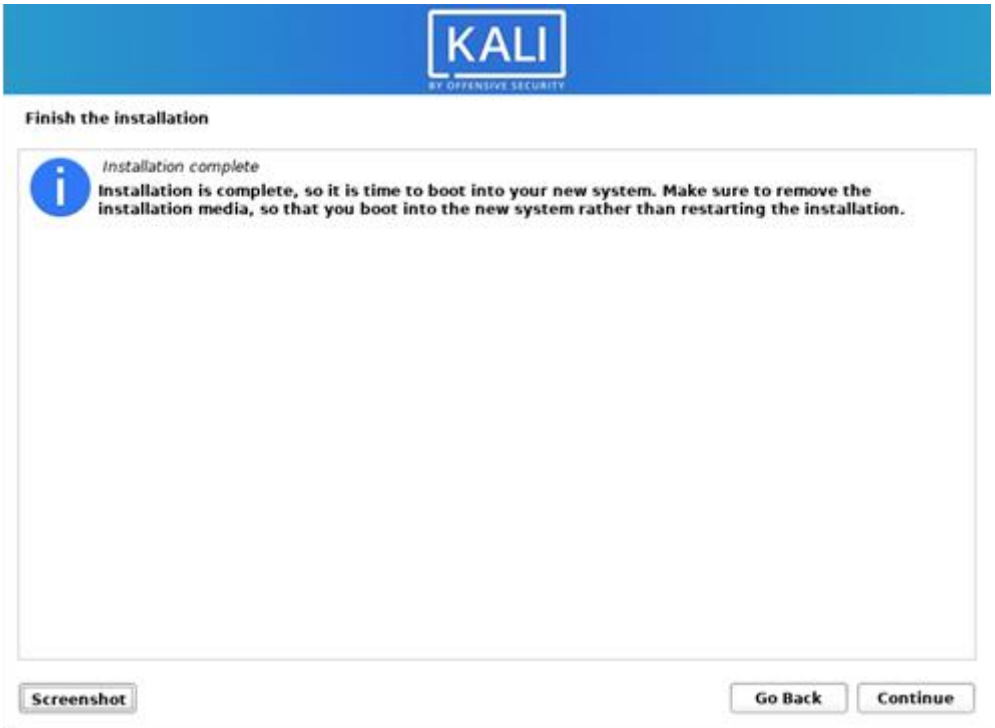
Σχήμα 5.9: Εγκατάσταση Kali Linux – Βήμα 7ο

8) Σε αυτό το βήμα επιλέγουμε το desktop environment που θέλουμε να έχουν τα Linux. Ακόμα, παρέχεται η δυνατότητα επιλογής εγκατάστασης των πιο γνωστών εργαλείων για penetration testing.



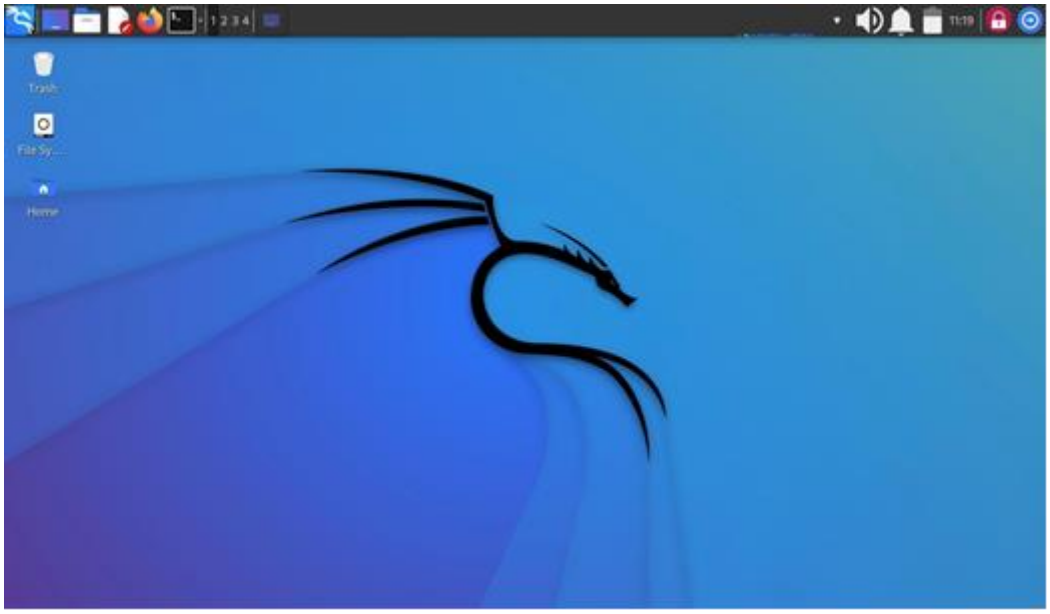
Σχήμα 5.10: Εγκατάσταση Kali Linux – Βήμα 8ο

9) Τέλος, η παρακάτω ένδειξη σημαίνει την ολοκλήρωση της διαδικασίας της εγκατάστασης.



Σχήμα 5.11: Εγκατάσταση Kali Linux – Βήμα 9ο

10) Στιγμιότυπο από την επιφάνεια εργασίας του Kali Linux ύστερα από την ολοκλήρωση της εγκατάστασης.



Σχήμα 5.12: Εγκατάσταση Kali Linux – Βήμα 10ο

5.2 Εγκατάσταση Προσομοιωτή κινητού - Genymotion

Η επιλογή ανάμεσα σε φυσική συσκευή ή σε emulator είναι βασική προϋπόθεση για τον έλεγχος κάποιας εφαρμογής. Για τις ανάγκες της εργασίας ακολουθήσαμε την επιλογή του emulator. Χρησιμοποιήσαμε ένα δωρεάν εργαλείο , το Genymotion (https://www.genymotion.com/download/). Σημαντικό είναι να αναφερθεί πως για να χρησιμοποιηθεί ο emulator , απαιτείται το VirtualBox, το οποίο εγκαταστάθηκε με ευκολία χρησιμοποιώντας τις εντολές από την επίσημη σελίδα του Kali (https://www.kali.org/docs/virtualization/install-virtualbox-host/).

Για την ομαλή εγκατάσταση του Genymotion τα προτεινόμενα χαρακτηριστικά που πρέπει να έχει το μηχάνημα είναι:

- Debian 9+ , μόνο 64 bit
- x86_64 CPU
- Hardware accelerated GPU
- 400Mb ελεύθερα στο δίσκο
- 4GM RAM
- Virtualbox

Αφού κατεβάσαμε το αρχείο από το παραπάνω σύνδεσμο, ακολουθούμε τα παρακάτω βήματα:

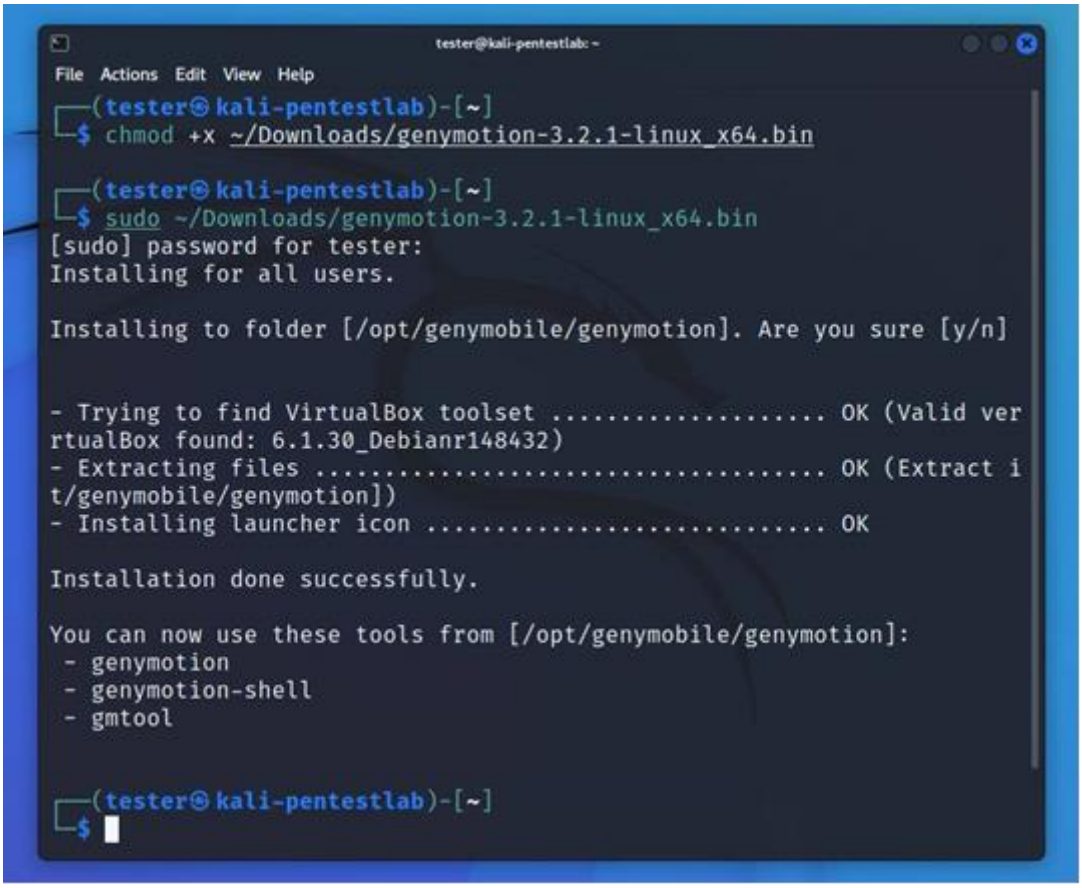
1. Εντοπίζουμε το αρχείο στον φάκελο Downloads που κατεβάσαμε.
2. Δίνουμε την παρακάτω εντολή ώστε το αρχείο να μετατραπεί σε εκτελέσιμο:

   chmod +x ~/Downloads/genymotion-3.2.1-linux_x64.bin

3. Στη συνέχεια εκτελούμε την παρακάτω εντολή με τη χρήση σα root:

   sudo ~/Downloads/Genymotion-3.2.1-linux_x64.bin

Μετά την εκτέλεση των παραπάνω εντολών θα εμφανιστούν τα παρακάτω στο terminal:

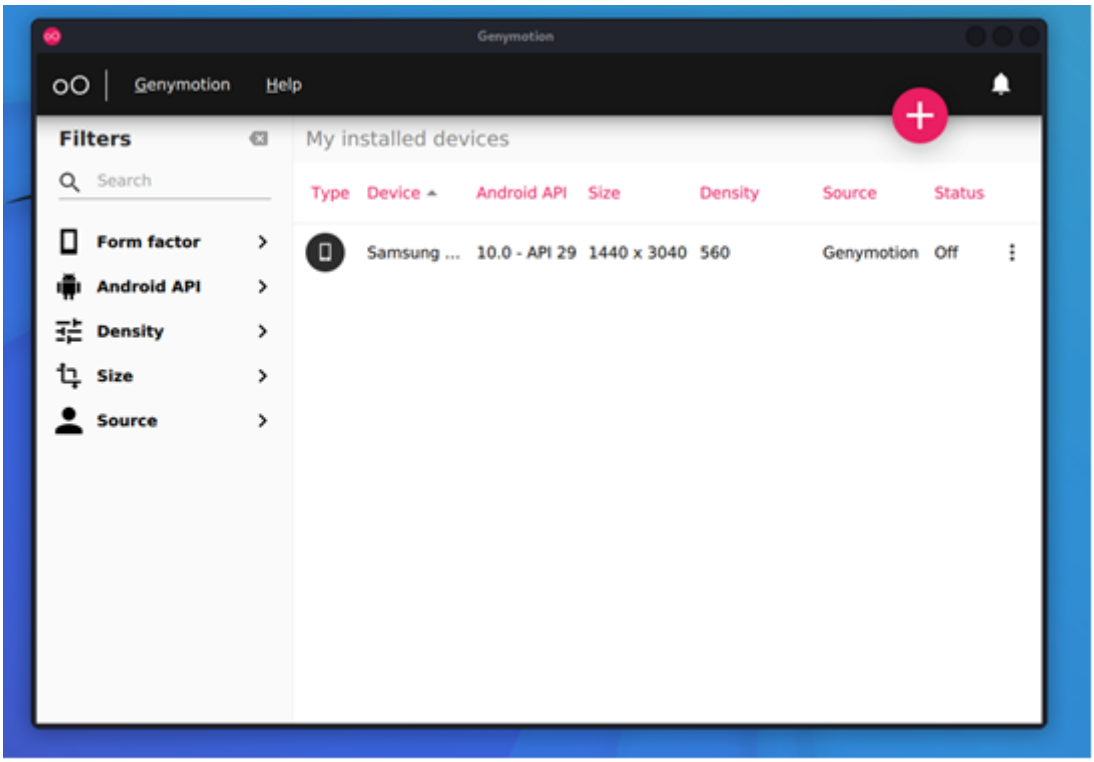


Σχήμα 5.13: Εγκατάσταση Genymotion(1/3)

Κατόπιν εκτελούμε την εφαρμογή του Genymotion. Σε περίπτωση που δεν έχουμε λογαριασμό θα πρέπει να δημιουργήσουμε. Έπειτα κάνοντας log in, εμφανίζεται το παρακάτω παράθυρο που είναι και το βασικό πλαίσιο του Genymotion(**Σχήμα 5.14**). Η επιλογή εγκατάστασης κάποιας συσκευής εξαρτάται από τις προτιμήσεις μας (π.χ. Android Version, API κ.ά.). Για να εκκινήσει ο emulator πατάμε διπλό κλικ.

Για τη συγκεκριμένη εργασία, επιλέξαμε τη συσκευή Samsung Galaxy S10 με Android Version 10.0 και API 29 επειδή διέθετε τα πιο πρόσφατα χαρακτηριστικά.

Τέλος, πατώντας πάνω στη συσκευή που δημιουργήσαμε, ξεκινάει ο emulator(**Σχήμα 5.15**).



Σχήμα 5.14: Εγκατάσταση Genymotion (2/3)

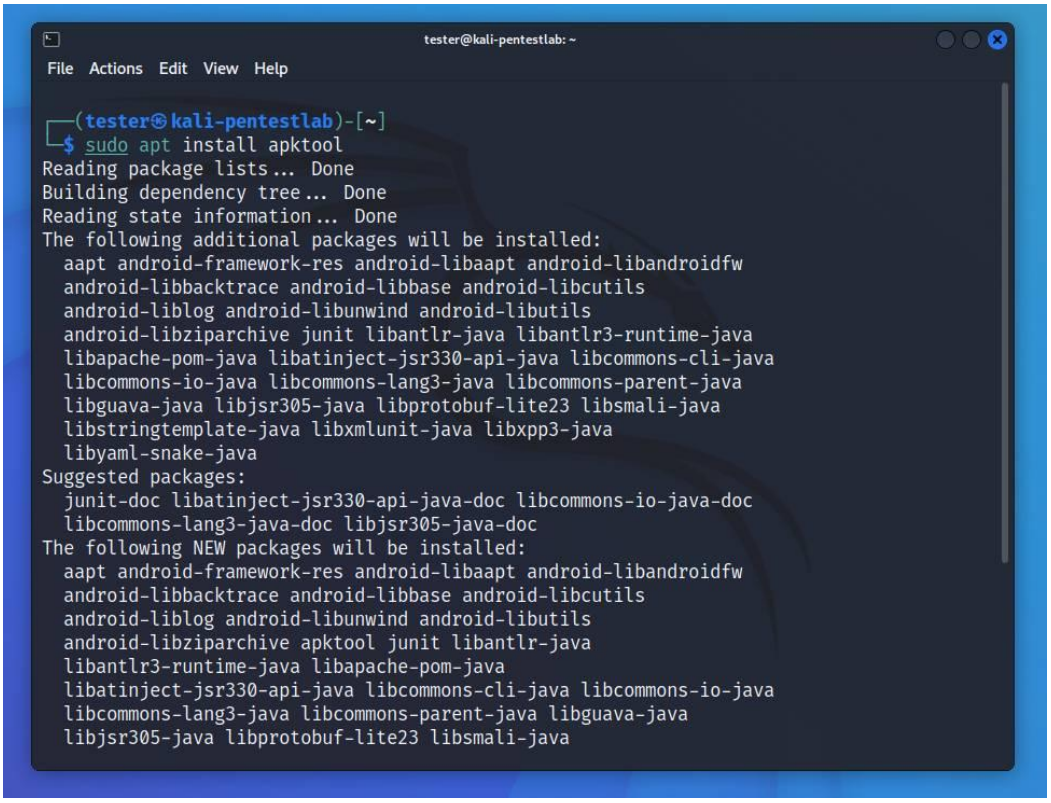
Σχήμα 5.15: Εγκατάσταση Genymotion (3/3)

5.3 Εγκατάσταση Apktool

Το Apktool είναι ένα 3rd party εργαλείο κατάλληλο για APK reverse engineering. Έχει τη δυνατότητα να αποκωδικοποιεί τα resources του APK σε μορφή σχεδόν ίδια με την αρχική, αλλά και να δημιουργεί ξανά το APK έχοντας υποστεί μετατροπές ο κώδικας της εφαρμογής. Ένα πλεονέκτημα του Apktool είναι πως μπορεί να αποδικωποιήσει compiled java αρχεία σε αρχεία «.smali». Αυτά τα αρχεία είναι υπεύθυνα για τη λειτουργικότητα της εφαρμογής. Τα αρχεία «.smali» μπορούν να προβληθούν και να επεξεργαστούν από έναν απλό editor.

Για την εγκατάσταση του Apktool, πληκτρολογήσαμε την εντολή στο terminal:

sudo apt install apktool

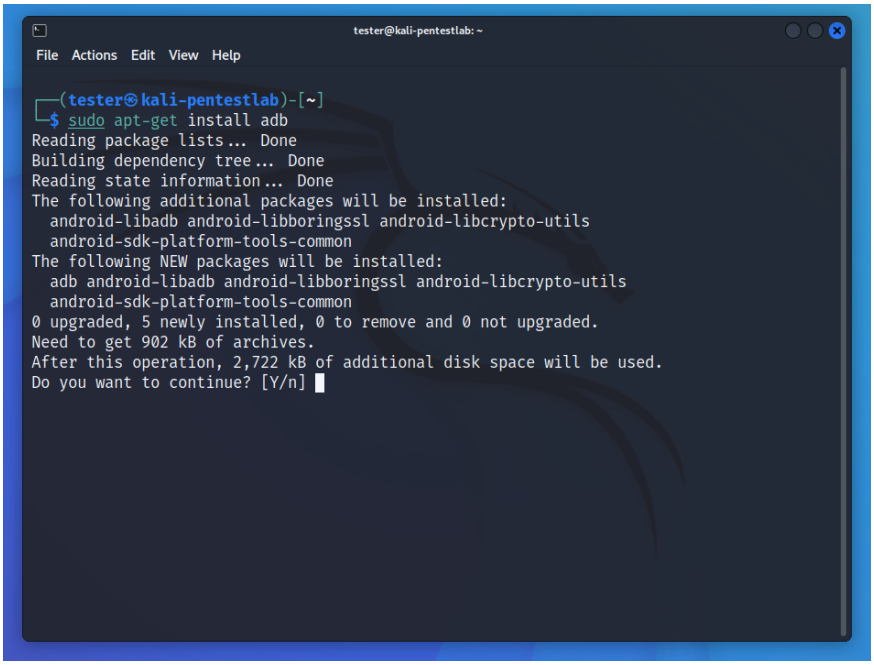


Σχήμα 5.16: Εγκατάσταση εργαλείου Apktool

5.4 Εγκατάσταση Adb

Το Android Debug Bridge (adb) είναι πρωτόκολλο επικοινωνίας που χρησιμοποιείται για τη σύνδεση ενός υπολογιστή με μια Android συσκευή. Στην παρούσα εργασία θα χρειαστεί καθώς θα μας βοηθήσει να εγκαταστήσουμε τις εφαρμογές στο emulator του Genymotion.

Για την εγκατάστασή του πληκτρολογήσαμε την εντολή **sudo apt-get install adb** στο terminal και στη συνέχεια απαντήσαμε με **Y** στην ερώτηση.

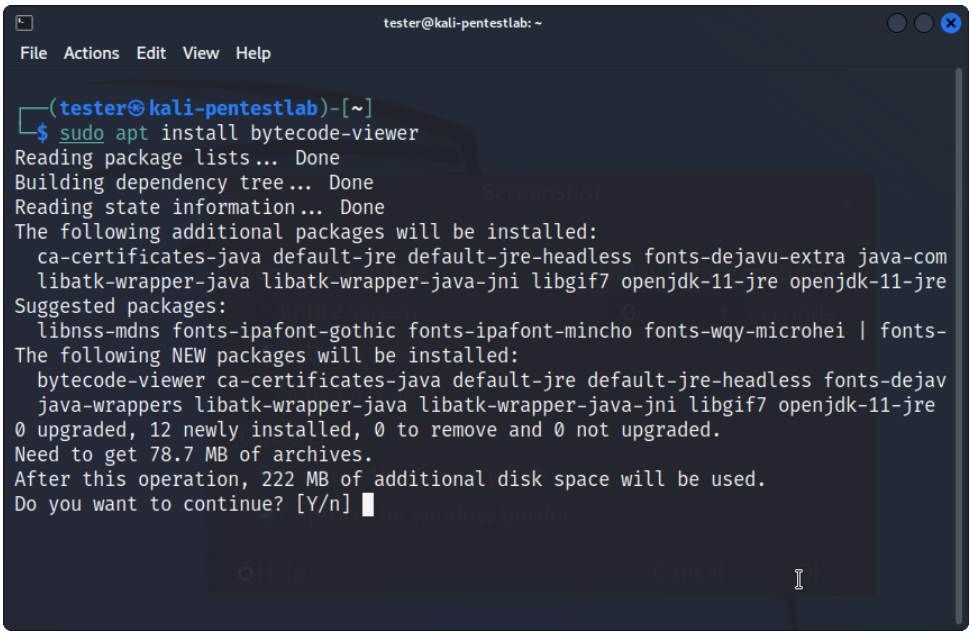


Σχήμα 5.17: Εγκατάσταση adb

5.5 Εγκατάσταση Bytecode-viewer

Το Bytecode-viewer είναι ένα πολυεργαλείο ιδανικό για Android/Java Reverse Engineering. Περιέχει 6 decompilers, 2 disassemblers, 2 assemblers, 2 APK converters κ.ά. Το Bytecode-viewer μπορεί να μετατρέψει ένα APK σε μια κατανοητή μορφή γραμμένη σε κώδικα Java, χωρίς να χρειάζονται τα εργαλεία apktool, dex2jar και Jd-Gui.

Για την εγκατάστασή του, απλώς πληκτρολογήσαμε την εντολή **sudo apt install bytecode-viewer** στο terminal και στο ερώτημα άμα θέλουμε να συνεχιστεί η εγκατάστασή του πατήσαμε **Y**.

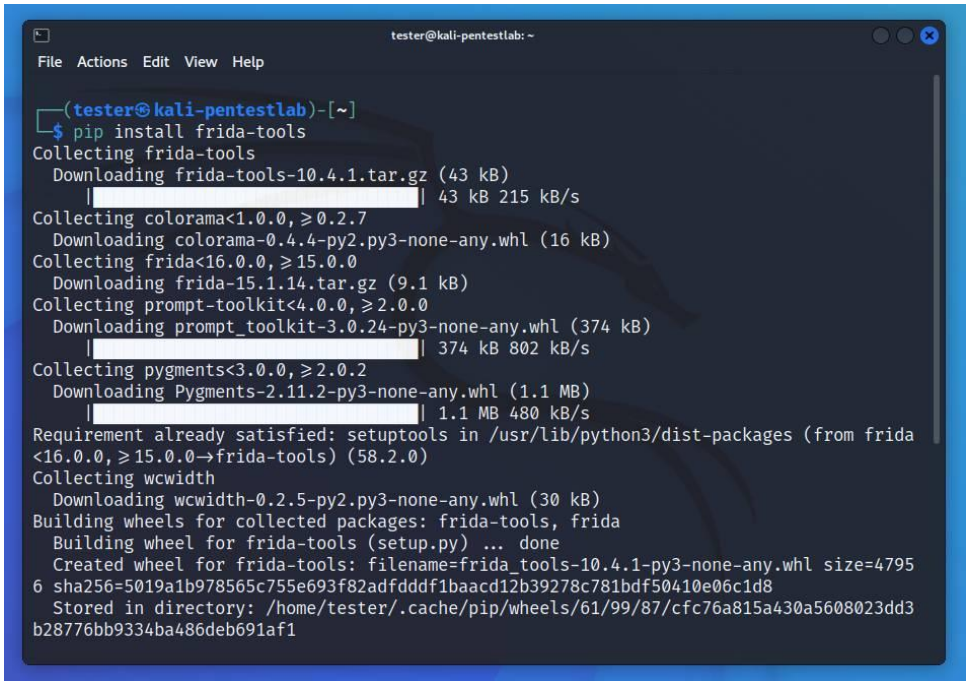


Σχήμα 5.18: Εγκατάσταση Bytecode-viewer

5.6 Εγκατάσταση Frida

Το Frida είναι ένα εργαλείο που χρησιμοπείται κατά τη δυναμική ανάλυση. Επιτρέπει την διακοπή των δεδομένων που λαμβάνονται και αποστέλλονται από κάποια εφαρμογή, καθώς και την εισαγωγή τμημάτων κώδικα κατά τη διαδικασία του compilation.

Για την εγκατάστασή του πληκτρολογήσαμε την εντολή **pip install frida-tools** στο terminal.

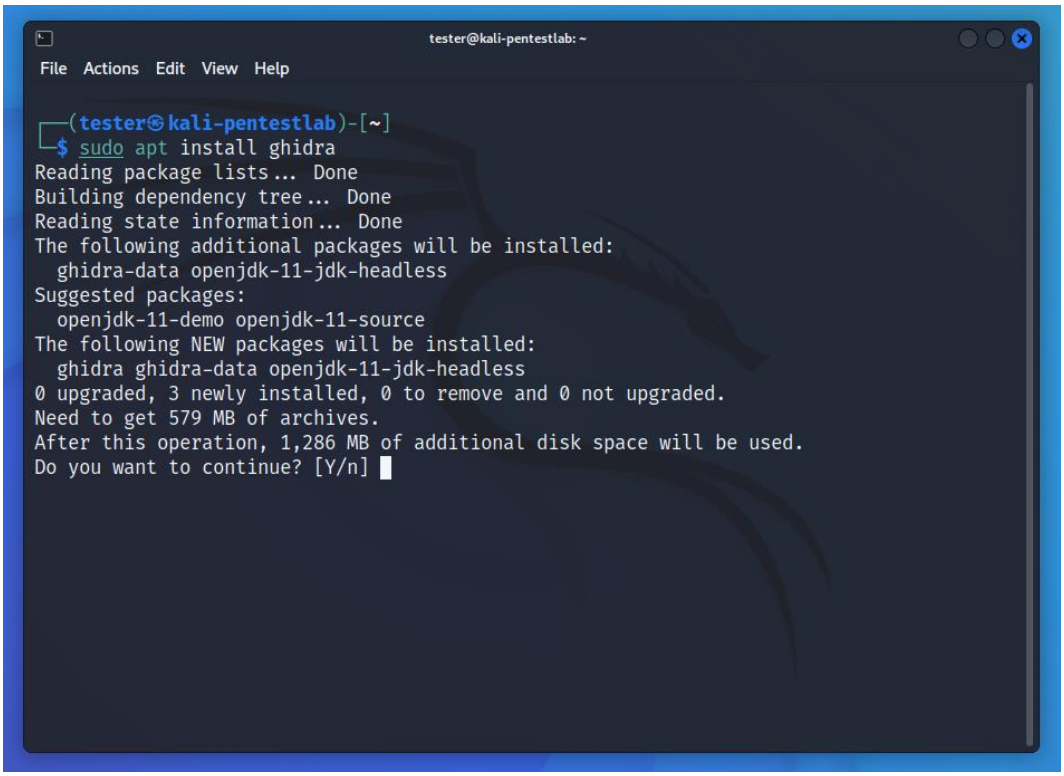


Σχήμα 5.19: Εγκατάσταση Frida

5.7 Εγκατάσταση Ghidra

Το Ghidra είναι ένα framework ιδανικό για Software Reverse Engineering (SRE). Δημιουργήθηκε και συντηρείτε από το τμήμα έρευνας της Υπηρεσία Εθνικής Ασφάλειας (NSA) των ΗΠΑ. Παρέχει πολλές δυνατότητες όπως είναι: disassembly, assembly, decompilation, scripting κ.ά.

Για την εγκατάστασή του πληκτρολογήσαμε την εντολή **sudo apt install ghidra** στο terminal.



Σχήμα 5.20: Εγκατάσταση προγράμματος Ghidra

Κεφάλαιο 6ο: Android Penetration Test

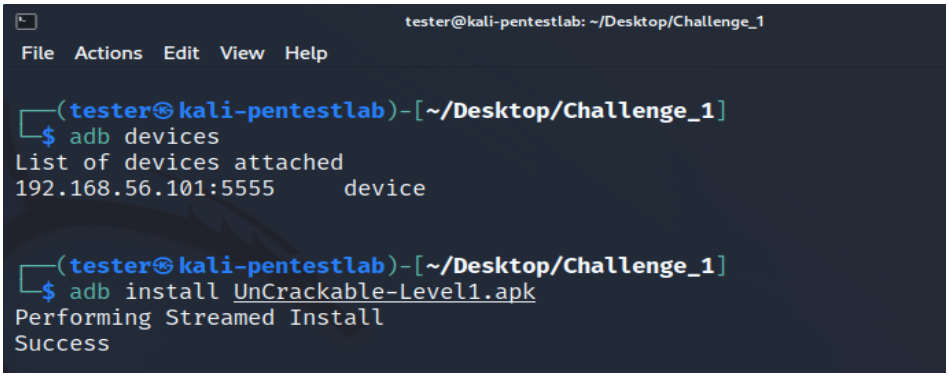
Ο Mobile Security Testing Guide (MSTG) ανήκει στο OWASP mobile project και είναι ένα ολοκληρωμένο εγχειρίδιο για την ανάπτυξη, τη δοκιμή και τη χρήση reverse engineering σε εφαρμογές για κινητά. Οι συντάκτες του MSTG έχουν δημιουργήσει κάποια challenges για τις Android και iOS πλατφόρμες με σκοπό την κατανόηση της ασφάλειας των εφαρμογών. Σε αυτήν την εργασία επιλέχθηκαν για λύση οι πρώτες δύο σε βαθμό δυσκολίας εφαρμογές για Android: Uncrackable App level 1 & 2.

6.1 Λύση Uncrackable challenge 1

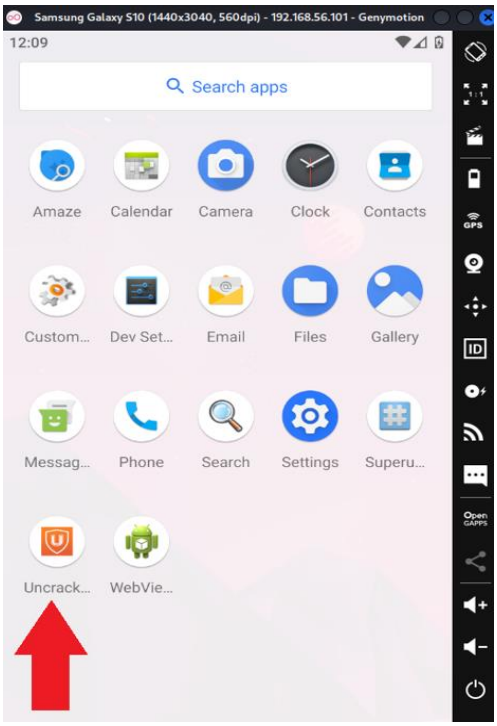
Η 1η δοκιμασία δημιουργήθηκε από τον Bernhard Mueller. Αρχικά, πλοηγηθήκαμε στο επίσημο repository του OWASP MSTG (https://github.com/OWASP/owasp-mstg). Έπειτα, αφού εντοπίσαμε τη θέση που βρισκόταν το APK αρχείο της 1ης δοκιμασίας, το κατεβάσαμε με την εντολή **wget** και το URL του.

Σχήμα 6.1: Λήψη APK αρχείου 1ης εφαρμογής

Το επόμενο βήμα ήταν να εγκαστήσουμε το APK στον emulator που είχαμε δημιουργήσει και εκκινήσει στο Genymotion. Για να είμαστε σίγουροι ότι ο emulator είναι συνδεδεμένος με τον υπολογιστή μας, πατήσαμε την εντολή **adb devices**. Αφού διαπιστώσαμε ότι υπάρχει μια ενεργή συσκευή συνδεδεμένη, εγκαταστήσαμε το APK με την εντολή **adb install UnCrackable-Level1.apk**.

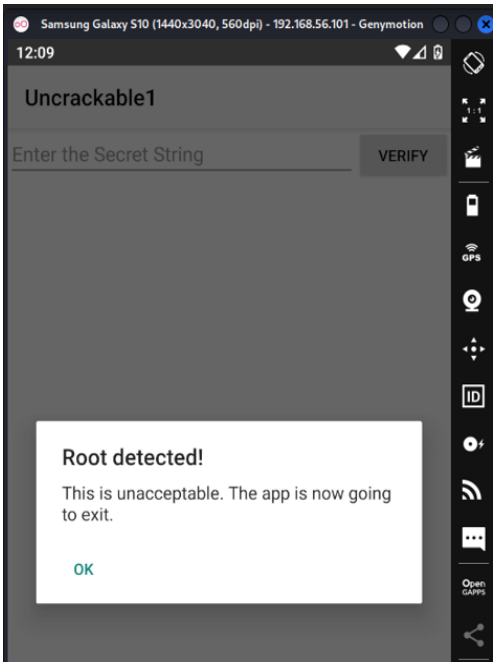


Σχήμα 6.2: Εγκατάσταση APK αρχείου 1ης εφαρμογής (1/2)



Σχήμα 6.3: Εγκατάσταση APK αρχείου 1ης εφαρμογής (2/2)

Κάνοντας «κλικ» πάνω στο εικονίδιό της εκκινήσαμε την εφαρμογή. Αυτόματα εμφανίστηκε ένα Alert Dialog που μας αναφέρει ότι το σύστημα της συσκευής είναι Root και ότι η εφαρμογή πρόκειται να κλείσει.



Σχήμα 6.4: Εμφάνιση μηνύματος Root detected!

Αυτό σημαίνει πως η εφαρμογή έχει προγραμματιστεί με τέτοιο τρόπο ώστε να αποτρέπει την εκτέλεσή της σε root συσκευές και είναι λογικό στην περίπτωσή μας να μην ανοίγει, καθώς τα API's του Genymotion είναι από default root. Στην πραγματικότητα, πολλές εφαρμογές χρησιμοποιούν τη λογική του εντοπισμού root, καθώς είναι ένα μέτρο στην ασφάλεια της εφαρμογής από κάποιον κακόβουλο χρήστη. Ωστόσο, αυτό μπορεί να οδηγήσει σε ευπάθεια, αν χρησιμοποιηθούν κακές πρακτικές στον κώδικα. Συνεπώς, η 1η δοκιμασία που είχαμε να αντιμετωπίσουμε ήταν να παρακάμψουμε τον εντοπισμό του Root.

Όπως αναφέραμε και σε προηγούμενη ενότητα, τα APK αρχεία είναι μια μορφή ZIP που περιέχουν XML αρχεία, dex κώδικα , τα resources αρχεία κ.ά. Επόμενη ενέργεια μας ήταν να κάνουμε unpack το APK αρχείο με τη χρήση του εργαλείου apktool.

Η εντολή που χρησιμοποιήσαμε ήταν η **apktool d –f –r UnCrackable-Level1.apk** με τα εξής options:

d: δηλώνουμε να γίνει decode το αρχείο.

f: διαγράφεται ο τυχόν φάκελος που μπορεί να υπήρχε με το ίδιο όνομα του φακέλου που θα δημιουργηθεί μετά την εκτέλεση της εντολής.

r: αποτρέπεται το decompilation του resources αρχείου και μετατρέπεται το αρχείο classes.dex σε αρχεία «.smali».

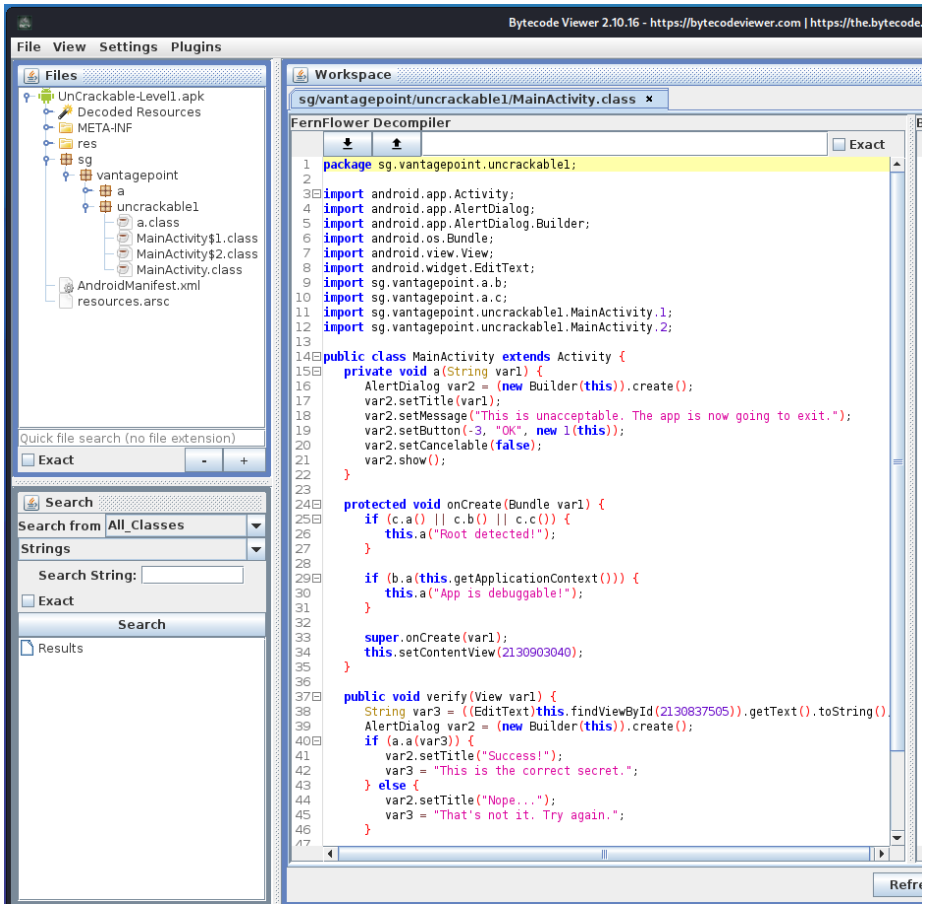
```
┌──(tester㉿kali-pentestlab)-[~/Desktop/Challenge_1]
└─$ apktool d -f -r UnCrackable-Level1.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.5.0-dirty on UnCrackable-Level1.apk
I: Copying raw resources...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...

┌──(tester㉿kali-pentestlab)-[~/Desktop/Challenge_1]
└─$ cd UnCrackable-Level1

┌──(tester㉿kali-pentestlab)-[~/Desktop/Challenge_1/UnCrackable-Level1]
└─$ ls
AndroidManifest.xml  apktool.yml  original  res  resources.arsc  smali
```

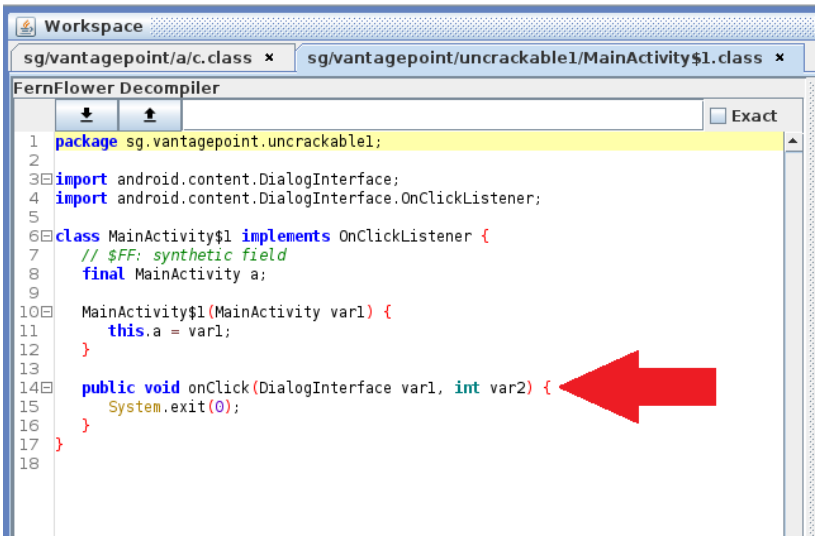
Σχήμα 6.5: Decompilation αρχείου APK

Για να μελετήσουμε τον κώδικα της εφαρμογής, εισάγαμε το APK αρχείο στο πρόγραμμα Bytecode-Viewer.



Σχήμα 6.6: Προβολή decompiled κώδικα (Java Bytecode)

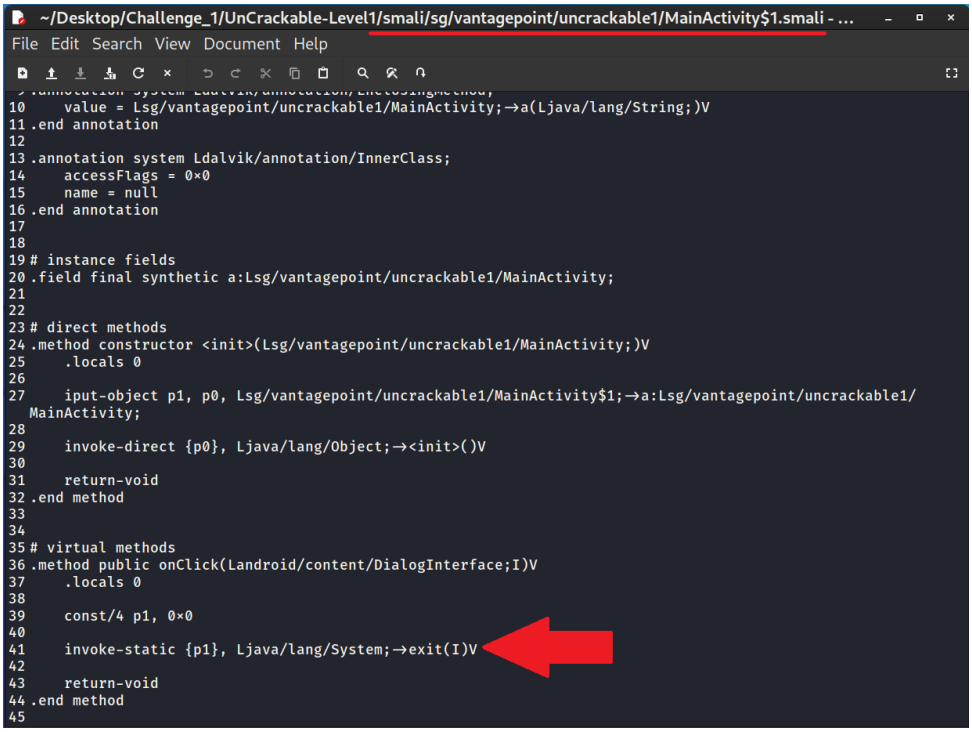
Μελετώντας τις κλάσεις και συγκεκριμένα τη *MainActivity$1.class* στη γραμμή 14 εντοπίσαμε ότι υπάρχει μια μέθοδος *onClick* η οποία εμφανίζει ένα Dialog και μετά κλείνει την εφαρμογή με την εντολή *System.exit(0);*. Αυτό είναι το dialog που είδαμε στο (**Σχήμα 6.4**).



Σχήμα 6.7: Εντοπισμός μεθόδου onClick στην MainActivity$1

Αν επεξεργαστούμε το «.smali» αρχείο της κλάσης *MainActivity$1* στο να μην κλείνει την εφαρμογή, τότε θα έχουμε παρακάμψει το root detection.

Έτσι, ανοίξαμε το αρχείο *MainActivity$1.smali* και στη γραμμή 36 εντοπίσαμε την ίδια μέθοδο που είδαμε και στη *MainActivity$1.class* και πιο συγκεκριμένα επικεντρωθήκαμε στη γραμμή 41 η οποία είναι αυτή που κλείνει την εφαρμογή.



Σχήμα 6.8: Εντοπισμός μεθόδου onClick στο MainActivity$1.smali

Ακολουθεί επεξήγηση της γραμμής 41 :

Invoke-static {p1}: παραπέμπει ότι καλείται μια μέθοδος που παίρνει μια στατική παράμετρο.

Ljava/lang/System: δηλώνεται το path που έχει κάποιο package στο σύστημα. Συγκεκριμένα πρόκειται για την κλάση System που συμπεριλαμβάνει πολλές μεθόδους.

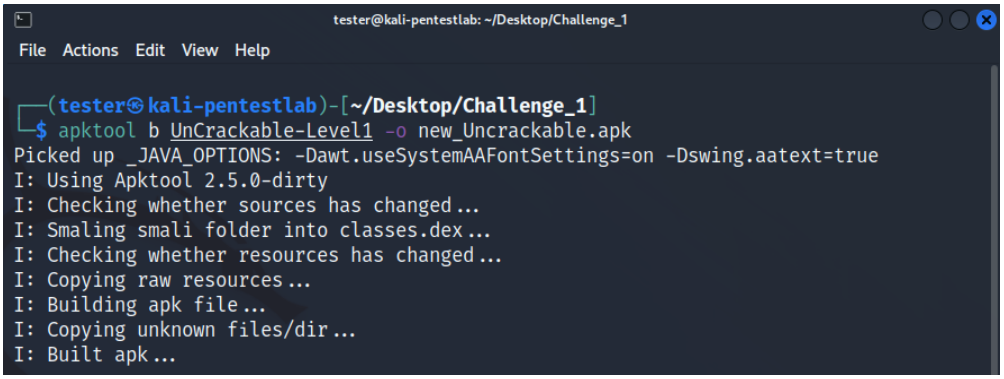
exit(I): αναφέρεται στη μέθοδο exit από την κλάση System και με I συμβολίζεται ότι παίρνει μια Integer παράμετρο

V: δηλώνεται ότι ο τύπος της μεθόδου exit είναι void.

Διαγράφοντας τη γραμμή 41, η void μέθοδος *OnClick* θα γυρνάει την τιμή **null** κάθε φορά που θα εντοπίζει ότι η συσκευή είναι root, με αποτέλεσμα να μην εκτελείται καμιά πρόσθετη ενέργεια.

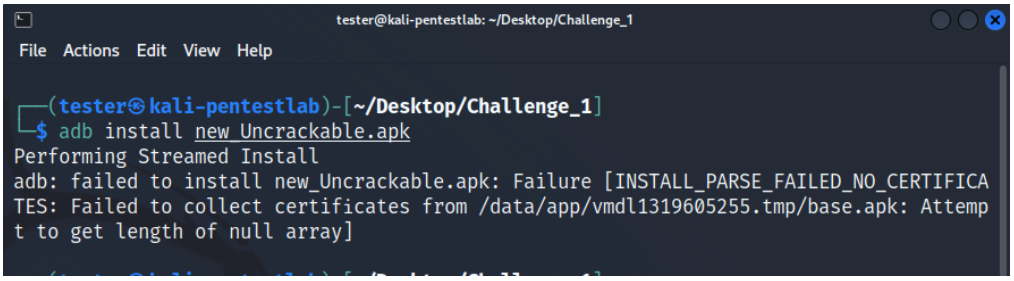
Σχήμα 6.9: Επεξεργασία αρχείου MainActivity$1.smali

Έχοντας επεξεργαστεί το αρχείο *MainActivity$1.smali*, επόμενο βήμα ήταν να δημιουργήσουμε ξανά το APK με τις καινούργιες μετατροπές. Επομένως, πληκτρολογήσαμε την εντολή **apktool b UnCrackable-Level1 –o new_Uncrackable.apk**, με το option **b** να συμβολίζει το build και το **–o** το νέο όνομα που θέλουμε να δώσουμε στο νέο APK.



Σχήμα 6.10: Rebuild αρχείου APK

Αφού δημιουργήσαμε μια νέα εφαρμογή , έπρεπε να την εγκαταστήσουμε στη συσκευή μας. Τρέξαμε ξανά την εντολή **adb install new_Uncrackable.apk** και όπως ήταν λογικό η εφαρμογή δεν εγκαταστάθηκε με επιτυχία καθώς το Android σύστημα επιτρέπει την εγκατάσταση κάποιας εφαρμογής μόνο εάν έχει υπογραφεί από τους προγραμματιστές του.



Επομένως , έπρεπε να δημιουργήσουμε ένα πιστοποιητικό και να υπογράψουμε το καινούργιο APK αρχείο με το keystore που θα είχαμε δημιουργήσει. Το εργαλείο που χρησιμοποιήσαμε για τη δημιουργία του keystore ήταν το Java keytool.

Στο terminal, πληκτρολογήσαμε εντολή **keytool –genkey –v –keystore tester.key.keystore –alias tester_key –keyalg RSA –keysize 2048 –validity 10000**.

Επιγραμματικά τα options:

genkey: δημιουργεί ένα ζευγάρι κλειδιών(public και private key) τα οποία τα τοποθετεί σε ένα X.509 υπογεγραμμένο πιστοποιητικό.

v: το output είναι λεπτομερώς.

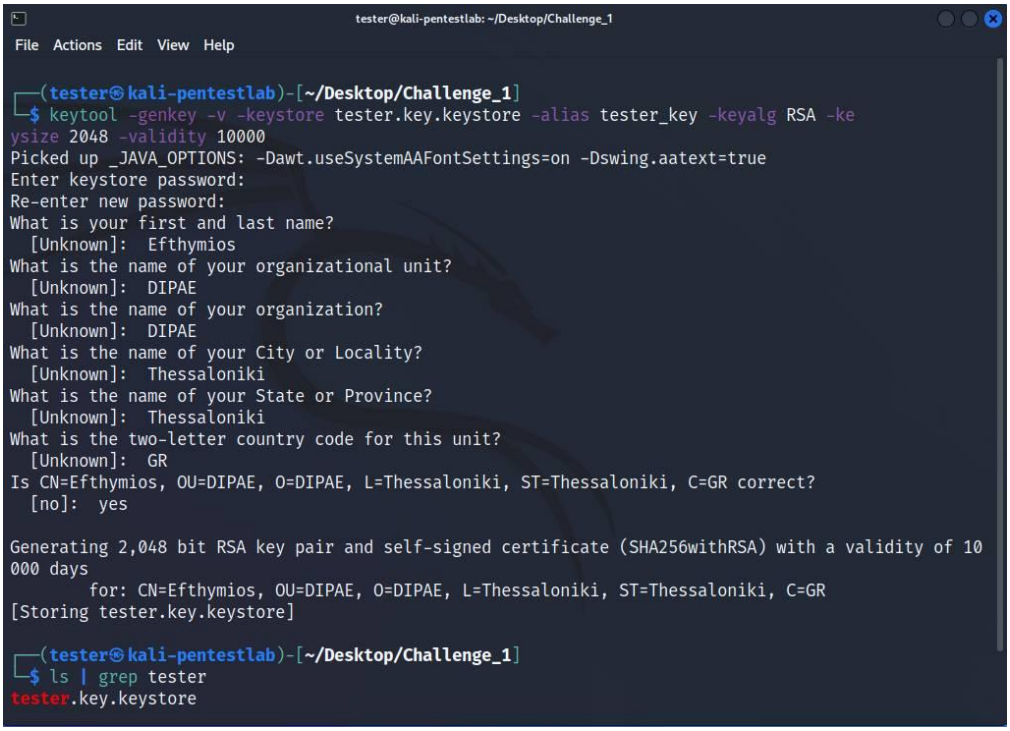
keystore: δηλώνεται το όνομα που θα έχει το keystore.

alias: δηλώνεται το όνομα που θα έχει το alias.

keyalg: δηλώνεται το όνομα του αλγορίθμου που θα έχει το κλειδί

keysize: δηλώνεται το μέγεθος του κλειδιού σε bit που θα έχει

validity: δηλώνεται ο αριθμός των ημερών που θα είναι σε ισχύ το πιστοποιητικό.



Σχήμα 6.11: Δημιουργία keystore

Αφού απαντήσαμε τα ερωτήματα, το keystore δημιουργήθηκε και αποθηκεύτηκε στον φάκελό μας. Ουσιαστικά, δημιουργήσαμε ένα πιστοποιητικό το οποίο είναι ένα RSA κλειδί μήκους 2048 bit και έχει ισχύ για 10.000 μέρες. Επόμενο βήμα , ήταν να επισυνάψουμε το APK μας με το πιστοποιητικό που δημιουργήσουμε.

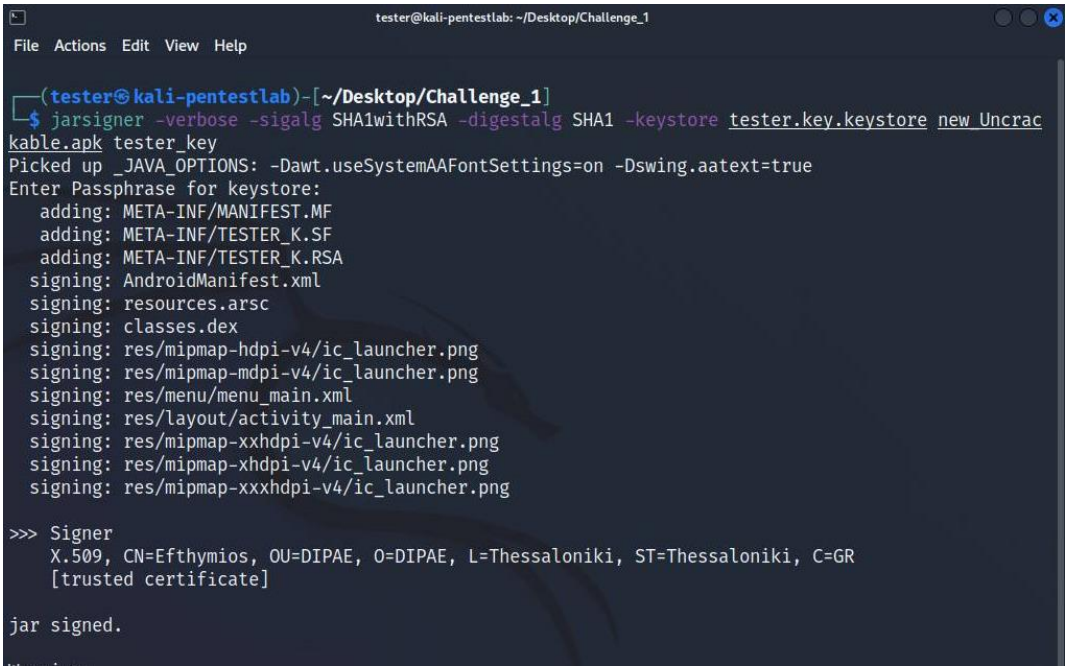
Η εντολή που πληκτρολογήσαμε ήταν η: **jarsigner –verbose –sigalg SHA1withRSA –digestalg SHA1 –keystore tester.key.keystore new_Uncrackable.apk tester_key**

Δηλαδή:

verbose: παρέχονται έξτρα πληροφορίες κατά τη διάρκεια της υπογραφής.

sigalg: διευκρινίζεται το όνομα του αλγόριθμου που θα χρησιμοποιηθεί για να υπογράψουμε το JAR αρχείο. Να σημειωθεί πως ο SHA1 θεωρείται ξεπερασμένος και δε θα πρέπει να χρησιμοποιείται σε κανένα πιστοποιητικό.

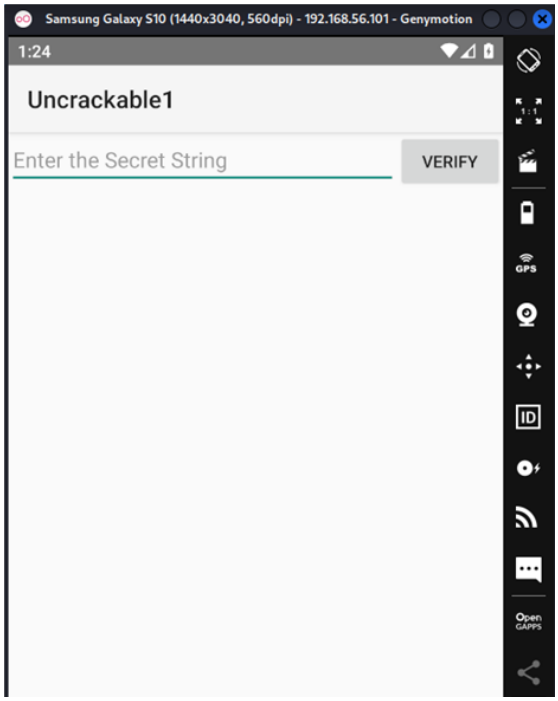
digestalg: διευκρινίζεται το όνομα του αλγορίθμου που θα χρησιμοποιηθεί όταν θα γίνεται digest.



Σχήμα 6.12: Υπογραφή APK

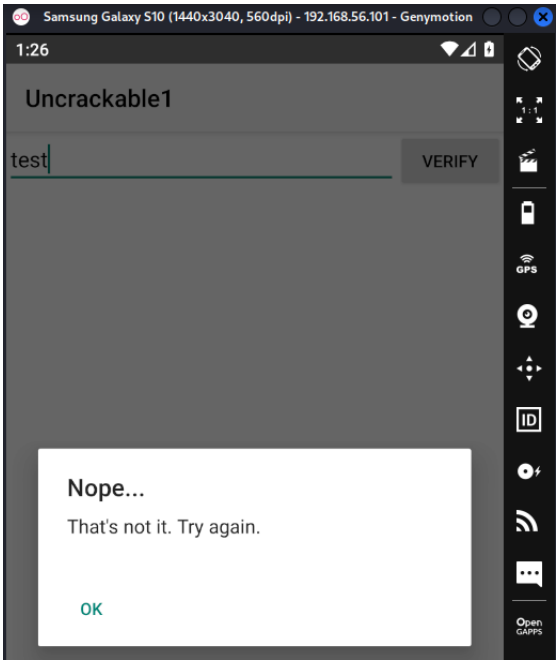
Στη συνέχεια εγκαταστήσαμε την καινούρια εφαρμογή επιτυχώς (**Σχήμα 6.13**) και την ανοίξαμε χωρίς να εμφανιστεί κάποιο μήνυμα.(**Σχήμα 6.14**)

Σχήμα 6.13: Εγκατάσταση καινούργιου APK



Σχήμα 6.14: Άνοιγμα εφαρμογής

Στο πάνω μέρος της οθόνης υπήρχε ένα απλό text field που έγραφε να εισάγουμε το secret string. Πληκτρολογώντας τη λέξη "test" και πατώντας το κουμπί VERIFY, εμφανίστηκε ένα άλλο Alert Dialog που μας ενημέρωσε ότι δεν εισάγαμε το σωστό string. Έτσι, η 2^{η} δοκιμασία που είχαμε να αντιμετωπίσουμε ήταν η εύρεση του σωστού secret string.



Σχήμα 6.15: Δοκιμή εύρεσης secret string

Επιστρέψαμε ξανά στη μελέτη του κώδικα της εφαρμογής και πιο συγκεκριμένα στην κλάση *MainActivity*. Στη γραμμή 37 εντοπίσαμε μια μέθοδο με το όνομα *verify()* η οποία καλεί τη μέθοδο *a()* από την κλάση *sg.vantagepoint.uncrackable1.a* με παράμετρο το string που πληκτρολογήσαμε στο text field. Στην περίπτωση που το string που εισάγαμε ήταν σωστό , η μέθοδος *a()* θα γυρίσει την τιμή **true** και στην οθόνη θα εμφανιστεί το μήνυμα “Success!”.

```
37   public void verify(View var1) {
38      String var3 = ((EditText)this.findViewById(2130837505)).getText().toString();
39      AlertDialog var2 = (new Builder(this)).create();
40      if (a.a(var3)) {
41         var2.setTitle("Success!");
42         var3 = "This is the correct secret.";
43      } else {
44         var2.setTitle("Nope...");
45         var3 = "That's not it. Try again.";
46      }
47
48      var2.setMessage(var3);
49      var2.setButton(-3, "OK", new 2(this));
50      var2.show();
51   }
```

Σχήμα 6.16: Εντοπισμός μεθόδου verify()

Συνεχίζοντας τη μελέτη του κώδικα, μεταφερθήκαμε στην κλάση *a* του πακέτου *sg.vantagepoint.a* όπου υπάρχει μια μέθοδος που συγκρίνει το εισαγόμενο string με το κρυπτογραφημένο με AES και κωδικοποιημένο σε base64 secret string. Σε περίπτωση που είναι ίδιο, επιστρέφει την τιμή **true**.

Workspace

sg/vantagepoint/uncrackable1/a.class ×

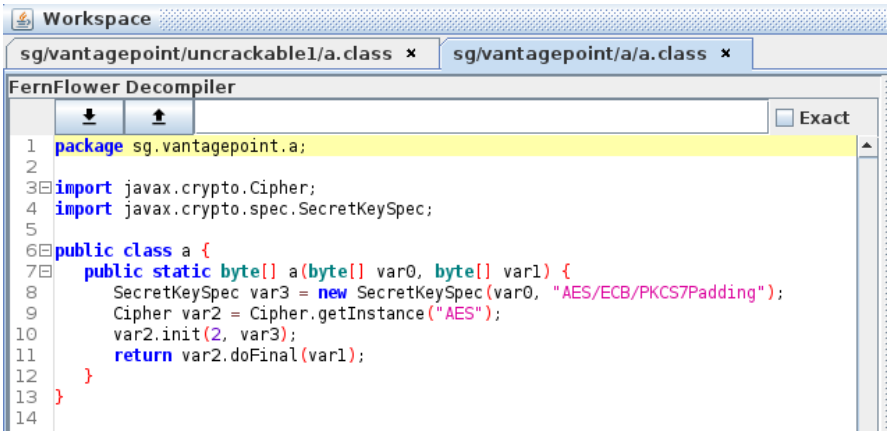
FernFlower Decompiler

Exact

```
1  package sg.vantagepoint.uncrackable1;
2
3  import android.util.Base64;
4  import android.util.Log;
5
6  public class a {
7     public static boolean a(String var0) {
8        byte[] var1 = Base64.decode("5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc=",
9
10       try {
11          var1 = sg.vantagepoint.a.a.a(b("8d127684cbc37c17616d806cf50473cc"), var1)
12       } catch (Exception var3) {
13          StringBuilder var4 = new StringBuilder();
14          var4.append("AES error:");
15          var4.append(var3.getMessage());
16          Log.d("CodeCheck", var4.toString());
17          var1 = new byte[0];
18       }
19
20       return var0.equals(new String(var1));
21    }
22
23    public static byte[] b(String var0) {
24       int var2 = var0.length();
25       byte[] var3 = new byte[var2 / 2];
26
27       for(int var1 = 0; var1 < var2; var1 += 2) {
28          var3[var1 / 2] = (byte)((Character.digit(var0.charAt(var1), 16) << 4) + C
29       }
30
31       return var3;
32    }
33 }
34
```

Σχήμα 6.17: Μελέτη κώδικα κλάσης sg/vantagepoint/uncrackable/a

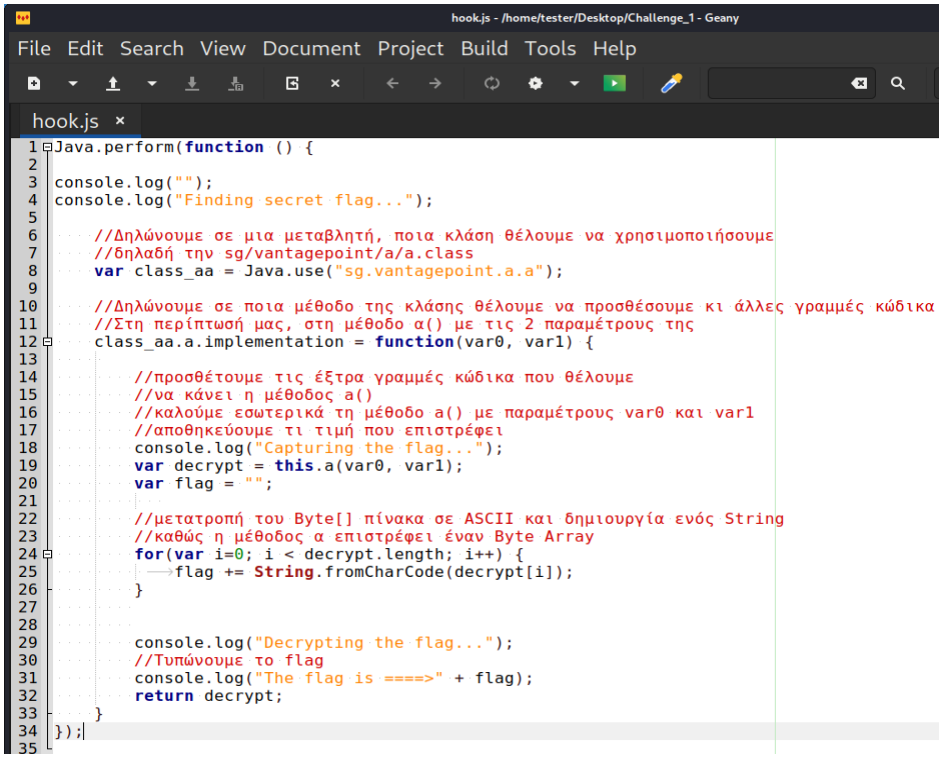
Μετά από ανάλυση του κώδικα, ενδιαφέρον είχε η γραμμή 11 , καθώς από ότι φαίνεται η μέθοδος *sg.vantagepoint.a.a.a()* είναι υπεύθυνη για την αποκρυπτογράφηση του κωδικοποιημένου secret string. Επομένως, αν καταφέρουμε να επεξεργαστούμε τον κώδικα της μεθόδου ώστε να εμφανίζει το secret string πριν συνεχιστεί η διαδικασία της σύγκρισης, τότε θα έχουμε λύσει και τη 2η δοκιμασία.



Σχήμα 6.18: Μέθοδος αποκρυπτογράφησης secret key

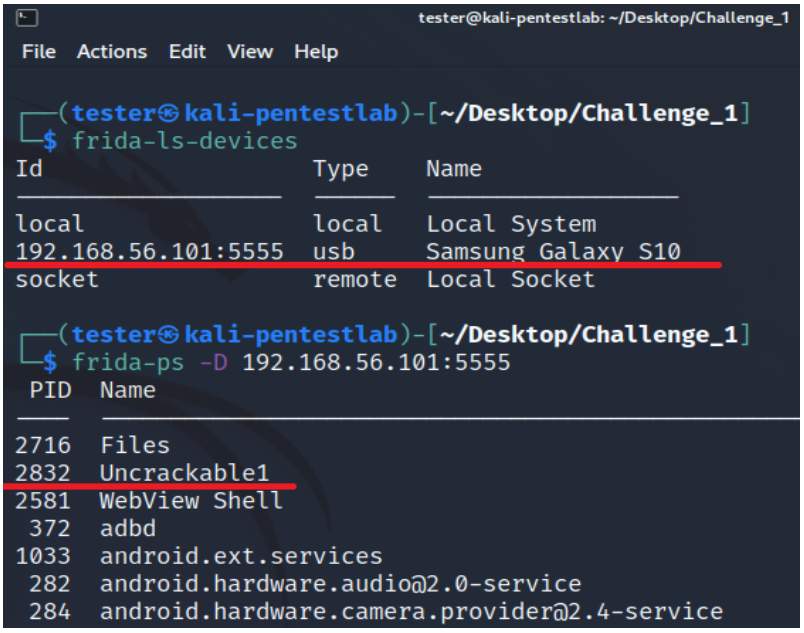
Η παραπάνω διαδικασία υλοποιήθηκε με τη βοήθεια του προγράμματος Frida, καθώς είναι ένα εργαλείο δυναμικής ανάλυσης που επιτρέπει την εισαγωγή και την εκτέλεση JavaScript payloads στον κώδικα της εφαρμογής. Η τεχνική που χρησιμοποιήσαμε ονομάζεται hook. Με αυτήν δηλώνουμε σε ποια κλάση και μέθοδο θέλουμε να έχουμε πρόσβαση και τι παραπάνω εντολές να εκτελούνται.

Με αυτόν τον τρόπο, δημιουργήσαμε ένα JavaScript αρχείο στο οποίο αρχικά δηλώσαμε ότι θέλουμε να εκτελεστεί μια μέθοδος στη Java με την εντολή **Java.perform(function (){})**. Έπειτα, αποθηκεύσαμε σε μια μεταβλητή την κλάση(*sg.vantagepoint.a.a*) στην οποία θέλαμε να έχουμε πρόσβαση με την εντολή **Java.use**. Με την εντολή **a.implementation = function(var0, var1)**, δηλώσαμε ότι θα θέλαμε να χρησιμοποιήσουμε τη μέθοδο a και να προσθέσουμε κι άλλες γραμμές κώδικα. Ακολούθως, καλέσαμε τη μέθοδο α εσωτερικά με την εντολή **this.a(var0, var1)** με τις 2 παραμέτρους της. Αφού μετατρέψαμε το **byte array** που επέστρεψε η μέθοδος α σε ASCII χαρακτήρες με την μέθοδο **fromCharCode**, το αποτέλεσμα το αποθηκεύσαμε σε μορφή string σε μια άλλη μεταβλητή. Τέλος, με την εντολή console.log τυπώνουμε το flag.



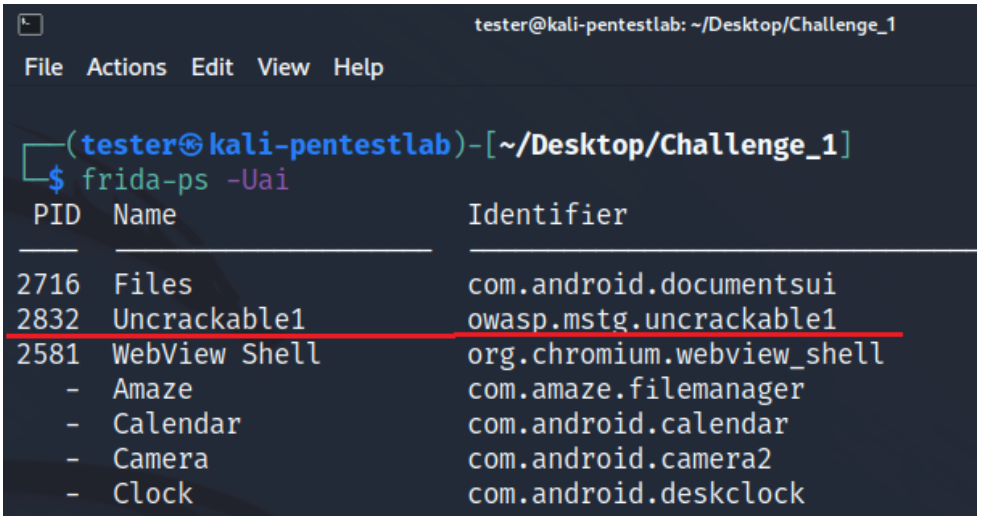
Σχήμα 6.19: Δημιουργία JavaScript payload Frida

Αφού δημιουργήσαμε το script μας, εγκαταστήσαμε τη σωστή έκδοση Frida-server στη συσκευή. Όπως προαναφέραμε το Frida λειτουργεί με τη λογική του Client-Server. Για να βρούμε το αναγνωριστικό της εφαρμογής μας, αρχικά πληκτρολογήσαμε την εντολή **frida-ls-devices** για να βεβαιωθούμε ότι η συσκευή μας ήταν συνδεδεμένη. Στη συνέχεια, πληκτρολογήσαμε την εντολή **frida-ps –D** μαζί με το ID της συσκευής μας για να δούμε τις εγκαταστημένες εφαρμογές της και εντοπίσαμε ότι η εφαρμογή που μας ενδιέφερε είχε το PID 2832.



Σχήμα 6.20: Εντοπισμός PID εφαρμογής στον emulator

Κατόπιν, με την εντολή **frida-ps** και το option **–Uai**, βρήκαμε το αναγνωριστικό της εφαρμογής που επιθυμούσαμε να χρησιμοποιήσουμε την τεχνική hook με το Frida.



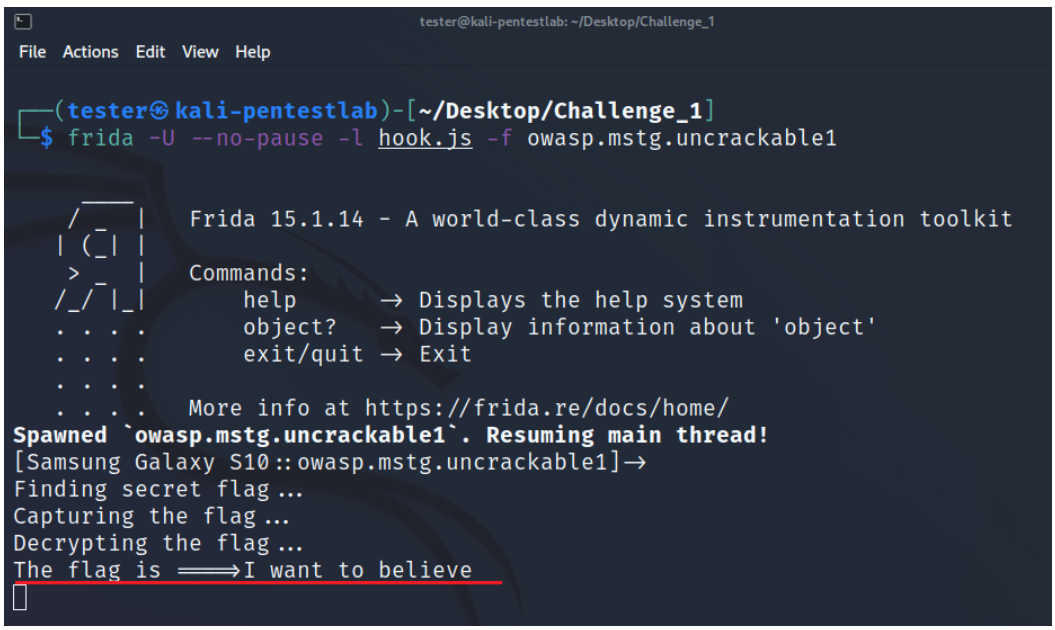
Σχήμα 6.21: Εντοπισμός αναγνωριστικού εφαρμογής

Η τελευταία εντολή που πληκτρολογήσαμε ήταν η **frida –U –no-pause –l hook.js –f owasp.mstg.uncrackable1**, με τα εξής options:

U: εκκινεί το πρόγραμμα frida.

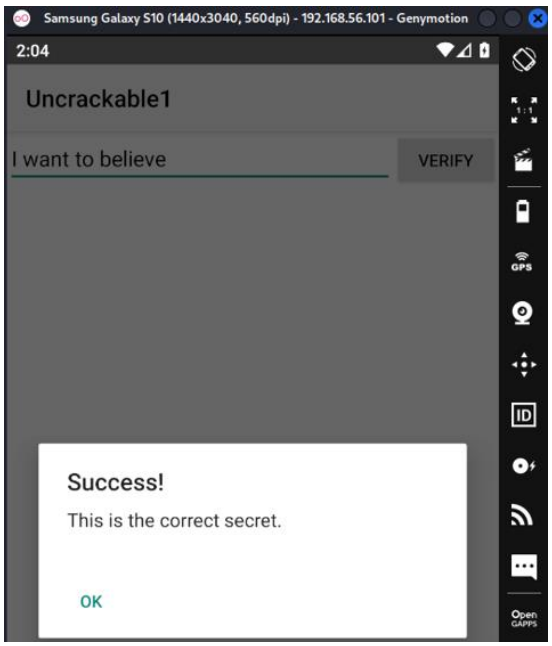
l: δηλώνει το αρχείο JavaScript που θέλουμε να εισάγουμε.

--no-pause –f: εκτελεί αυτόματα την εφαρμογή με τον κώδικα που έχουμε προσθέσει.



Σχήμα 6.22: Εύρεση secret string

Και κάπως έτσι εμφανίστηκε στο terminal το secret string. Αυτήν τη φορά πληκτρολογώντας το στο text field και πατώντας το κουμπί VERIFY, το Alert που εμφανίστηκε έγραφε το μήνυμα “Success!”

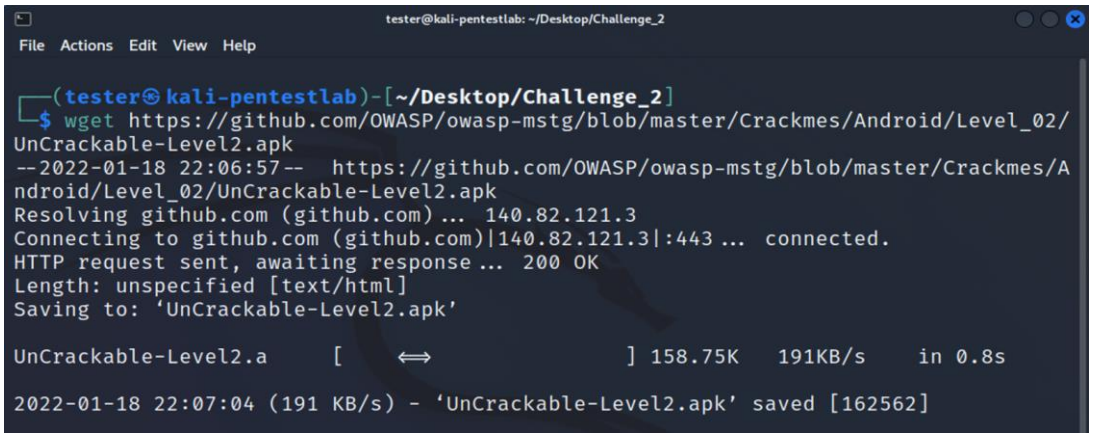


Σχήμα 6.23: Εμφάνιση μηνύματος σωστού secret string

Στο σημείο αυτό ολοκληρώθηκαν οι δοκιμασίες της 1ης εφαρμογής επιτυχώς, καθώς παρακάμψαμε το root detection και βρήκαμε το σωστό secret string.

6.2 Λύση Uncrackable challenge 2

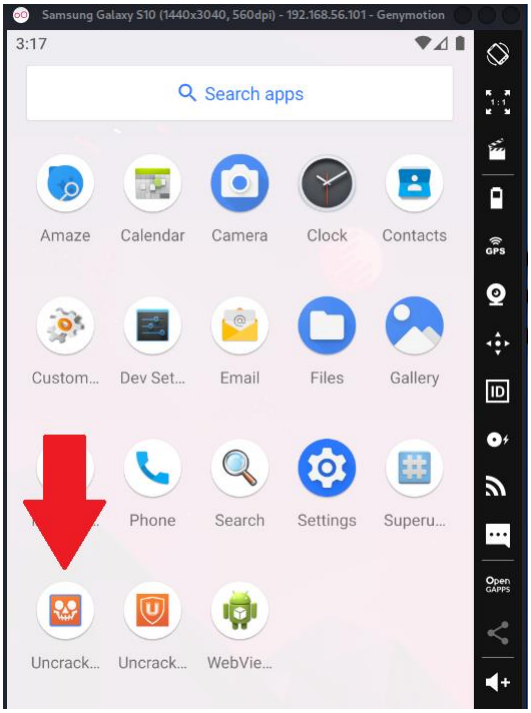
Η 2η δοκιμασία δημιουργήθηκε επίσης από τον Bernhard Mueller και αργότερα συντηρήθηκε από τους τους OWASP MSTG leaders. Όπως και στην 1η εφαρμογή, αρχικά πλοηγηθήκαμε στο επίσημο repository του OWASP MSTG (https://github.com/OWASP/owasp-mstg) και αφού εντοπίσαμε τη θέση που βρισκόταν η 2η εφαρμογή, την κατεβάσαμε με την εντολή **wget** και το URL της.



Σχήμα 6.24: Λήψη APK αρχείου 2ης εφαρμογής

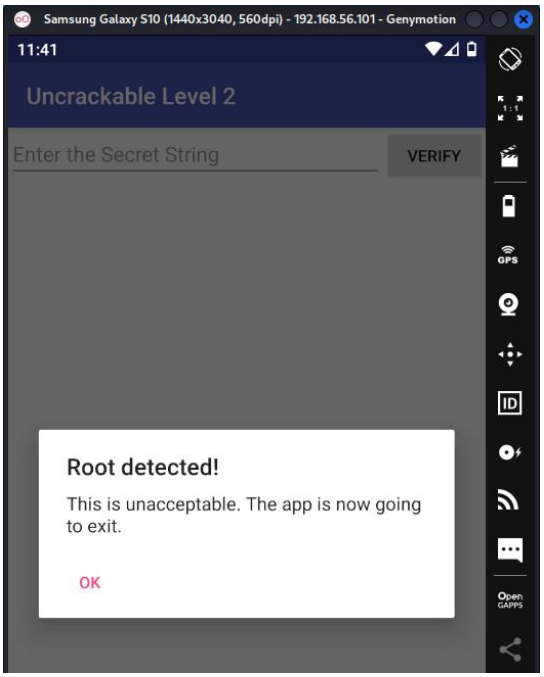
Στη συνέχεια, εγκαταστήσαμε το APK στον emulator που είχαμε εκκινήσει στο Genymotion με την εντολή **adb install UnCrackable-Level2.apk**. Καθώς γνωρίζαμε ότι ο emulator ήταν συνδεδεμένος με τον υπολογιστή μας, δε χρειάστηκε να πληκτρολογήσουμε την εντολή **adb devices**.

Σχήμα 6.25: Εγκατάσταση APK αρχείου 2ης εφαρμογής (1/2)



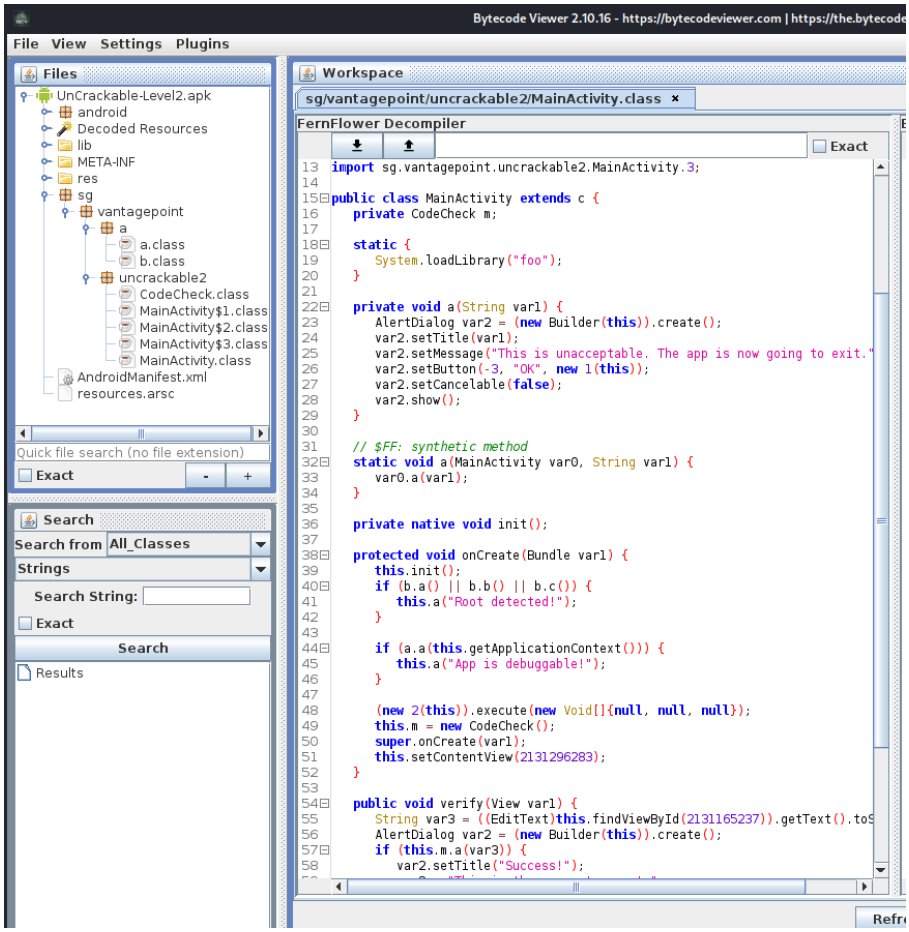
Σχήμα 6.26: Εγκατάσταση APK αρχείου 2ης εφαρμογής (2/2)

Κάνοντας «κλικ» πάνω στο εικονίδιό της εκκινήσαμε την εφαρμογή. Το Alert Dialog που εμφανίστηκε στην οθόνη μας ήταν ολόιδιο με αυτό της 1η εφαρμογής, δηλαδή πως το σύστημά μας ήταν Root και η εφαρμογή θα έκλεινε.



Σχήμα 6.27: Εμφάνιση μηνύματος Root detected!

Για να βρεθεί ο τρόπος παράκαμψης του root detection έπρεπε να αναλυθεί ο κώδικας της εφαρμογής. Έτσι , εισάγαμε το APK αρχείο στο πρόγραμμα Bytecode-Viewer.



Σχήμα 6.28: Προβολή decompiled κώδικα (Java Bytecode)

Μελετώντας το αρχείο *MainActivity.class*, επικεντρωθήκαμε στη γραμμή 39 όπου υπήρχε μια μέθοδος με το όνομα *onCreate*. Μέσα στη μέθοδο, υπήρχε ένα if statement το οποίο έλεγχε την τιμή που επέστρεφαν κάποιες μέθοδοι. Εάν, κάποια από τις 3 μεθόδους *a*, *b* ή *c* της κλάσης *sg/vantagepoint/a/b.class* επέστρεφαν την τιμή true , τότε το σύστημα θα αναγνωριζόταν σαν root και η εφαρμογή θα έκλεινε.

```
37
38  protected void onCreate(Bundle var1) {
39     this.init();
40     if (b.a() || b.b() || b.c()) {
41        this.a("Root detected!");
42     }
43
```

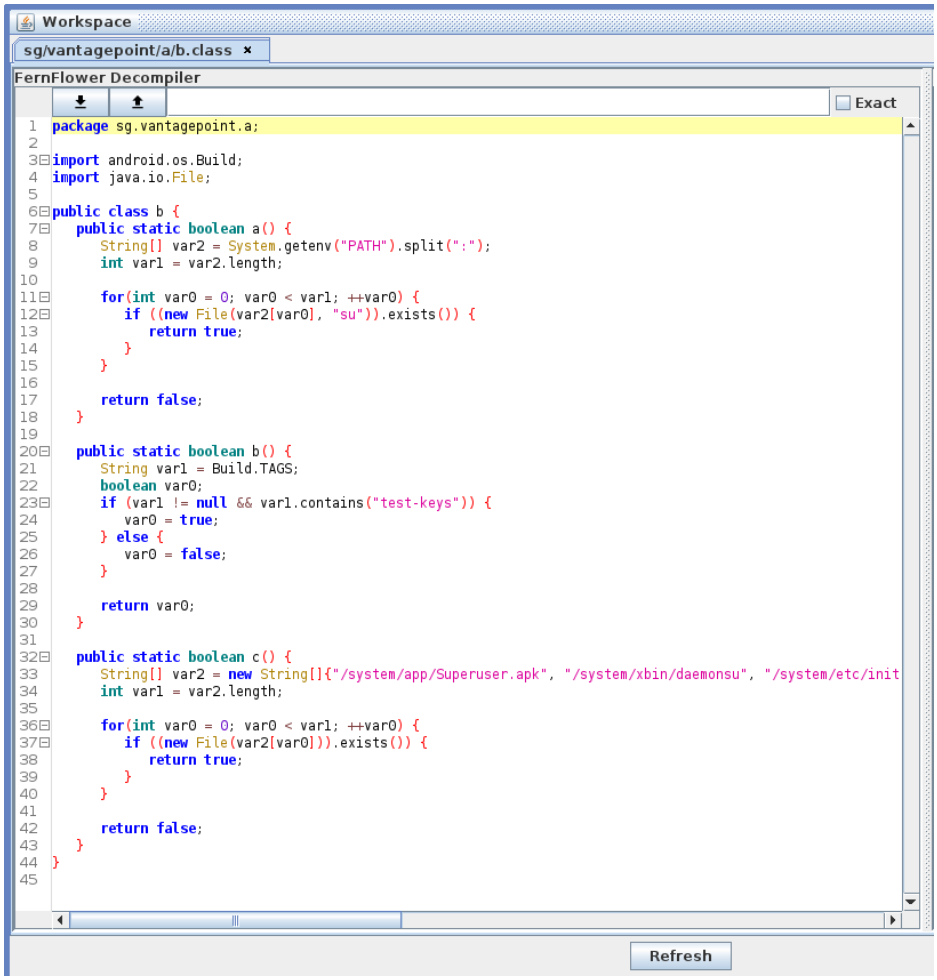
Σχήμα 6.29: Εντοπισμός μεθόδου onCreate στην MainActivity.class

Πηγαίνοντας στην *b.class* για περαιτέρω ανάλυση, κατανοήσαμε πως η κλάση εκτελούσε 3 διαφορετικούς ελέγχους για τον εντοπισμό root στη συσκευή. Πιο αναλυτικά:

Su binary - Η **a** μέθοδος ελέγχει την ύπαρξη του αρχείου SU binary. Εάν υπάρχει, τότε επιστρέφει την τιμή true καθώς η συσκευή είναι root.

Test-keys - Η **b** μέθοδος ελέγχει εάν τα keys που υπογράφτηκαν κατά την έκδοση του kernel πήραν την τιμή release-keys ή την test-keys. Εάν είναι το τελευταίο, τότε σημαίνει πως το kernel υπογράφτηκε με ένα custom key από έναν 3rd party προγραμματιστή. Αυτή η πληροφορία βρίσκεται στο αρχείο /system/build.prop.

/system/ - Η **c** μέθοδος ελέγχει εάν κάποια αρχεία ή binaries είναι προσβάσιμα στο σύστημα. Εάν ναι, τότε η συσκευή είναι root.



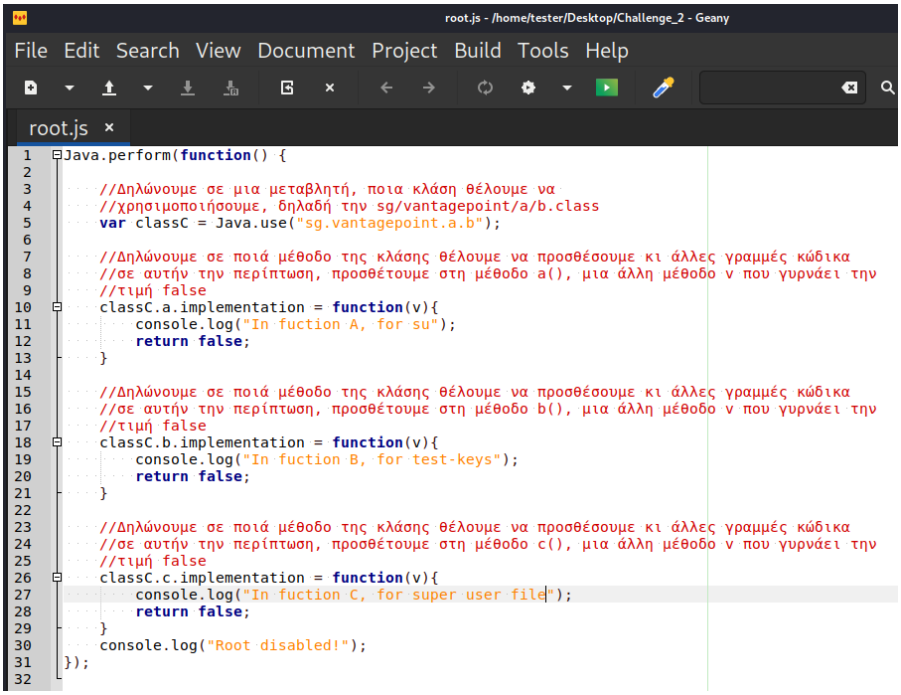

```
1  package sg.vantagepoint.a;
2
3  import android.os.Build;
4  import java.io.File;
5
6  public class b {
7     public static boolean a() {
8        String[] var2 = System.getenv("PATH").split(":");
9        int var1 = var2.length;
10
11       for(int var0 = 0; var0 < var1; ++var0) {
12          if ((new File(var2[var0], "su")).exists()) {
13             return true;
14          }
15       }
16
17       return false;
18    }
19
20    public static boolean b() {
21       String var1 = Build.TAGS;
22       boolean var0;
23       if (var1 != null && var1.contains("test-keys")) {
24          var0 = true;
25       } else {
26          var0 = false;
27       }
28
29       return var0;
30    }
31
32    public static boolean c() {
33       String[] var2 = new String[]{"/system/app/Superuser.apk", "/system/xbin/daemonsu", "/system/etc/init
34       int var1 = var2.length;
35
36       for(int var0 = 0; var0 < var1; ++var0) {
37          if ((new File(var2[var0])).exists()) {
38             return true;
39          }
40       }
41
42       return false;
43    }
44 }
45
```

Σχήμα 6.30: Μελέτη sg/vantagepoint/a/b.class

Επομένως, αν καταφέρουμε να επεξεργαστούμε τον κώδικα της εφαρμογής και συγκεκριμένα τις μεθόδους a,b και c στο να γυρνάνε την τιμή *false* αντί *true*, τότε θα έχουμε παρακάμψει τον εντοπισμό του root.

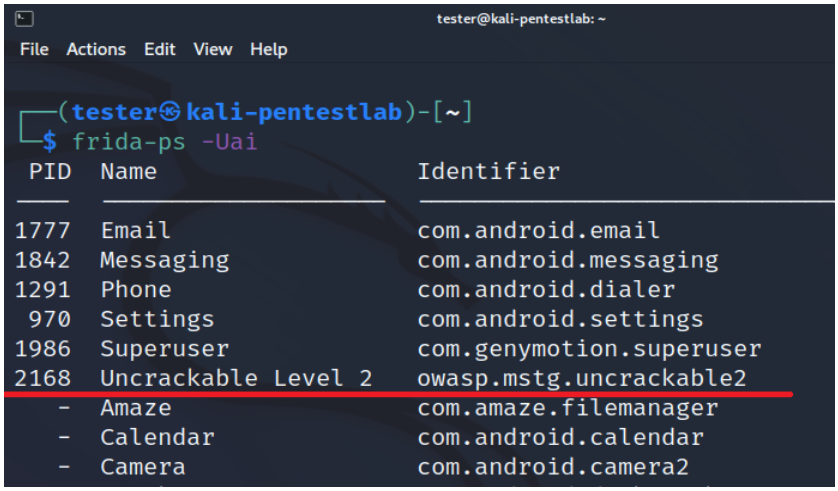
Η παραπάνω διαδικασία υλοποιήθηκε ξανά με το πρόγραμμα Frida και ακολούθησε την ίδια λογική που χρησιμοποιήσαμε στην 1η εφαρμογή.

Δημιουργήσαμε ένα JavaScript αρχείο όπου στην αρχή δηλώσαμε ότι θέλουμε να εκτελεστεί μια μέθοδος στην *Java* με την εντολή **Java.perform(function (){})**. Έπειτα, αποθηκεύσαμε σε μια μεταβλητή την κλάση(*sg.vantagepoint.a.b*) στην οποία θέλαμε να έχουμε πρόσβαση με την εντολή **Java.use**. Στη συνέχεια, προσθέσαμε 3 νέες μεθόδους οι οποίες θα επέστρεφαν την τιμή false, μέσα σε κάθε μία μέθοδο από τις a, b και c με την εντολή **implementation = function(v)**.



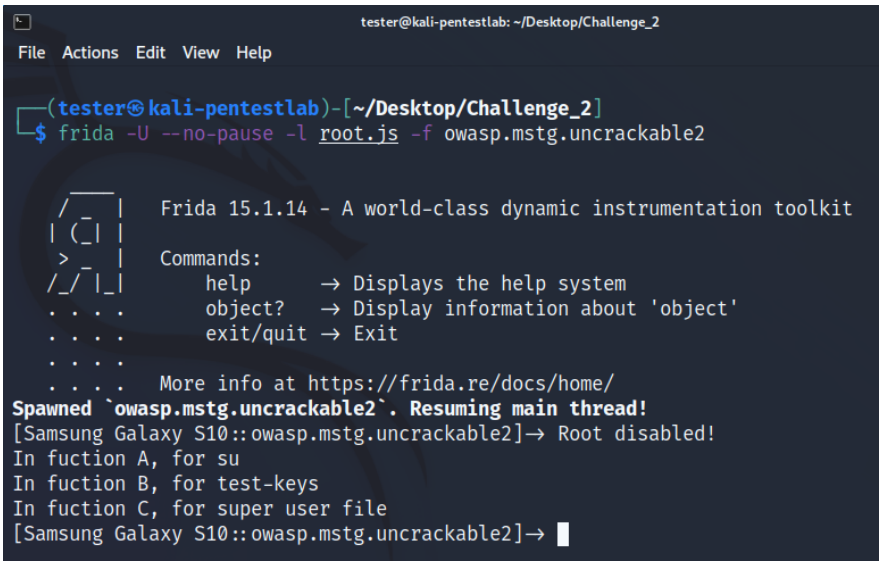
Σχήμα 6.31: Δημιουργία JavaScript payload Frida

Αφού δημιουργήσαμε το script μας και εγκαταστήσαμε τη σωστή έκδοση του Frida-server στον emulator, το επόμενο βήμα ήταν να βρούμε το αναγνωριστικό της 2ης εφαρμογής. Για τον εντοπισμό του, πληκτρολογήσαμε την εντολή **frida-ps –Uai** στο terminal.



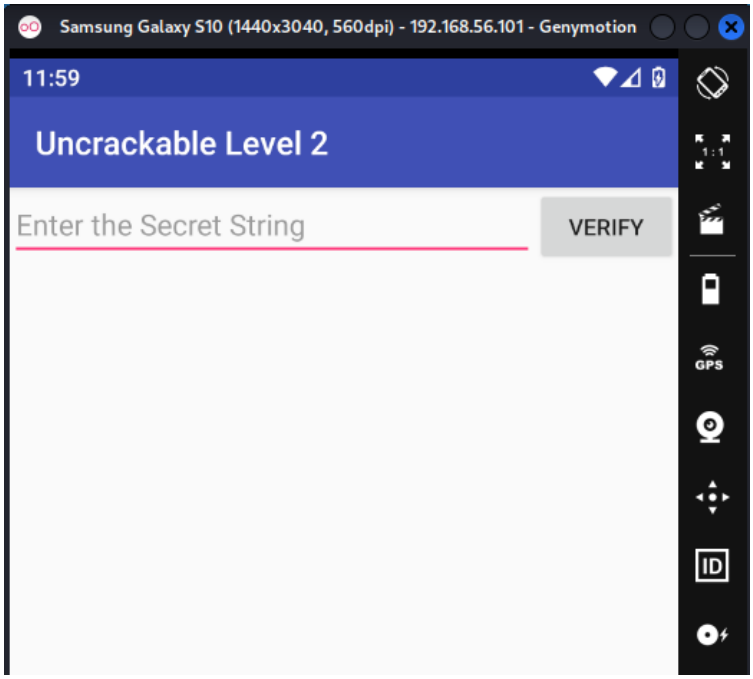
Σχήμα 6.32: Εντοπισμός αναγνωριστικού 2ης εφαρμογής

Για να χρησιμοποιήσουμε το καινούριο JavaScript payload που δημιουργήσαμε με το πρόγραμμα Frida, χρησιμοποιήσαμε την ίδια εντολή με την 1η εφαρμογή με διαφορετικό όνομα αρχείου και αναγνωριστικού.



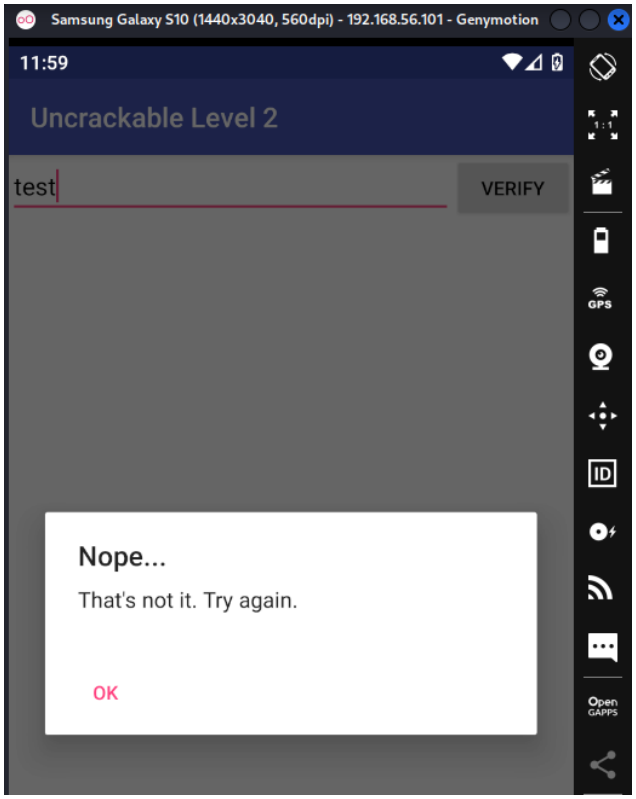
Σχήμα 6.33: Παράκαμψη root detection

Δίνοντας την εντολή αυτή η εφαρμογή εκτελέστηκε αυτόματα και δεν εμφανίστηκε κάποιο *Alert Dialog* για τον εντοπισμό του root.



Σχήμα 6.34: Εκτέλεση 2ης εφαρμογής

Όπως στην πρώτη εφαρμογή έτσι κι σε αυτή, στο πάνω μέρος υπήρχε ένα απλό text field που έγραφε να εισάγουμε το secret string. Πληκτρολογώντας τη λέξη “test” και πατώντας το κουμπί VERIFY, εμφανίστηκε ένα άλλο Alert Dialog που μας ενημέρωσε ότι δεν εισάγαμε το σωστό string. Συνεπώς, η 2η δοκιμασία που είχαμε να αντιμετωπίσουμε ήταν η εύρεση ενός ακόμα secret string.



Σχήμα 6.35: Δοκιμή εύρεσης secret string

Έτσι, επιστρέψαμε ξανά στην ανάλυση του κώδικα και πιο συγκεκριμένα στην κλάση MainActivity. Στη γραμμή 54 εντοπίσαμε μια μέθοδο με το όνομα verify() η οποία καλεί τη μέθοδο *a()* από την κλάση *m* με παράμετρο το string που πληκτρολογήσαμε στο text field. Στην περίπτωση που το string που εισάγαμε ήταν σωστό , η μέθοδος *a()* θα γυρίσει την τιμή **true** και στην οθόνη θα εμφανιστεί το μήνυμα “Success!”.

```
53
54  public void verify(View var1) {
55     String var3 = ((EditText)this.findViewById(2131165237)).getText().toString();
56     AlertDialog var2 = (new Builder(this)).create();
57     if (this.m.a(var3)) {
58        var2.setTitle("Success!");
59        var3 = "This is the correct secret.";
60     } else {
61        var2.setTitle("Nope...");
62        var3 = "That's not it. Try again.";
63     }
```

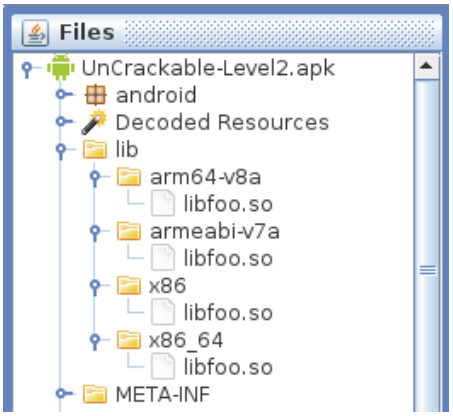
Σχήμα 6.36: Εντοπισμός μεθόδου verify()

Πηγαίνοντας στην αρχή της κλάσης και συγκεκριμένα κάτω από τον ορισμό της MainActivity, παρατηρήσαμε ότι η μεταβλητή *m* ορίζεται σαν ένα Object της κλάσης *Codecheck*. Ακόμα, ακριβώς από κάτω , βρήκαμε πως η εφαρμογή χρησιμοποιεί μια native βιβλιοθήκη με το όνομα foo.

```
14
15 public class MainActivity extends c {
16     private CodeCheck m;
17
18     static {
19         System.loadLibrary("foo");
20     }
```

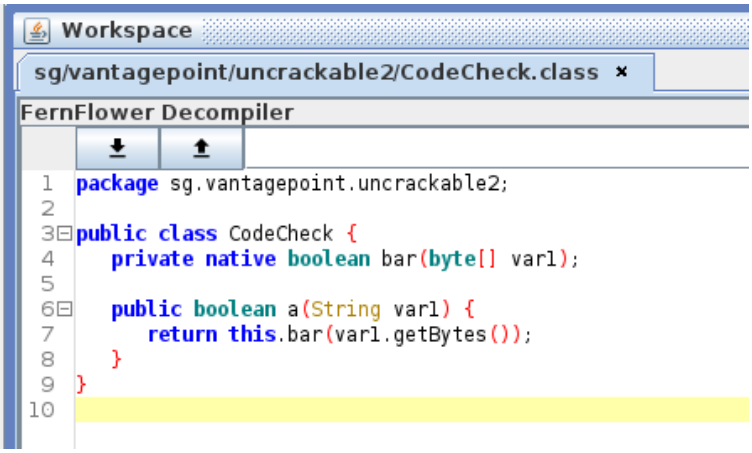
Σχήμα 6.37: Ορισμός Object m και εισαγωγή βιβλιοθήκης foo

Μια τέτοια βιβλιοθήκη μπορεί να βρεθεί στον φάκελο lib/:



Σχήμα 6.38: Προβολή περιεχομένων φακέλου lib

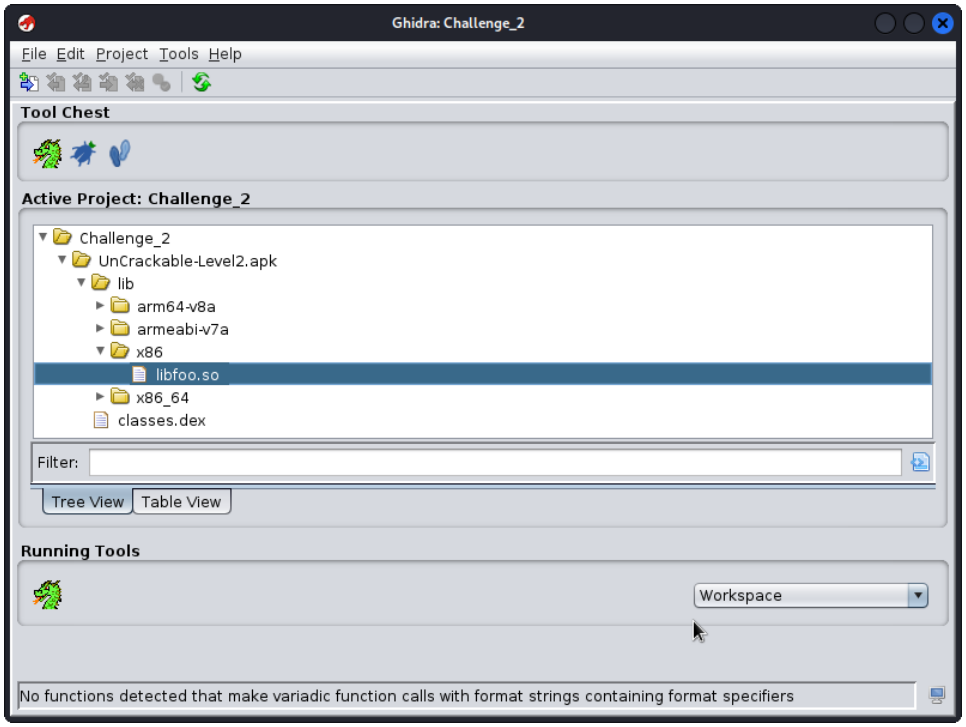
Επόμενο βήμα ήταν να πάμε στην κλάση CodeCheck. Όπως φαίνεται η κλάση CodeCheck χρησιμοποιεί μια native μέθοδο με το όνομα bar, η οποία βρίσκεται στη βιβλιοθήκη foo. Αυτή η μέθοδος συγκρίνει το string που εισάγαμε με το secret και επιστρέφει το αποτέλεσμα. Η μέθοδος a, καλεί τη μέθοδο bar και απλώς γυρνάει την τιμή true ή false σε περίπτωση που το string είναι σωστό ή όχι αντίστοιχα.



Σχήμα 6.39: Μελέτη κώδικα κλάσης CodeCheck

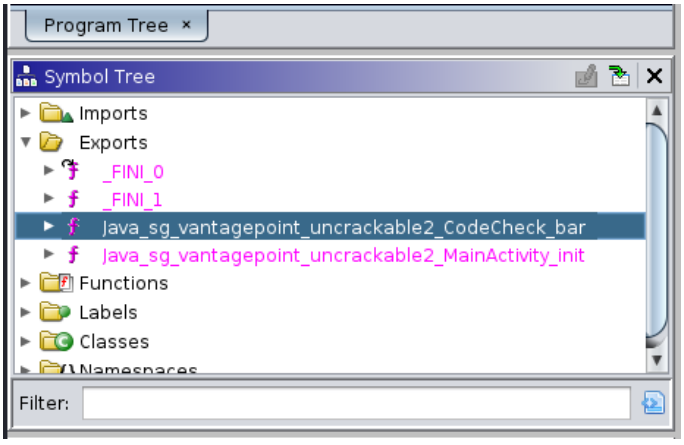
Συνεπώς το επόμενο βήμα που έπρεπε να κάνουμε ήταν να κάνουμε reverse engineering σε κάποιες από τις native βιβλιοθήκες με σκοπό να βρούμε τη μέθοδο σύγκρισης με το secret string. Για την πραγματοποίηση αυτής της διαδικασίας χρησιμοποιήθηκε το πρόγραμμα Ghidra.

Αρχικά δημιουργήσαμε ένα νέο project στο Ghidra με το όνομα Challenge_2. Έπειτα εισάγαμε το APK αρχείο μας και στην ερώτηση για τον τρόπο που θέλουμε να γίνει το import, διαλέξαμε το Batch Import. Στη συνέχεια, πατήσαμε διπλό «κλικ» πάνω στην βιβλιοθήκη libfoo.so της αρχιτεκτονικής x86(τυχαία επιλογή) για να ξεκινήσει το reverse engineering.



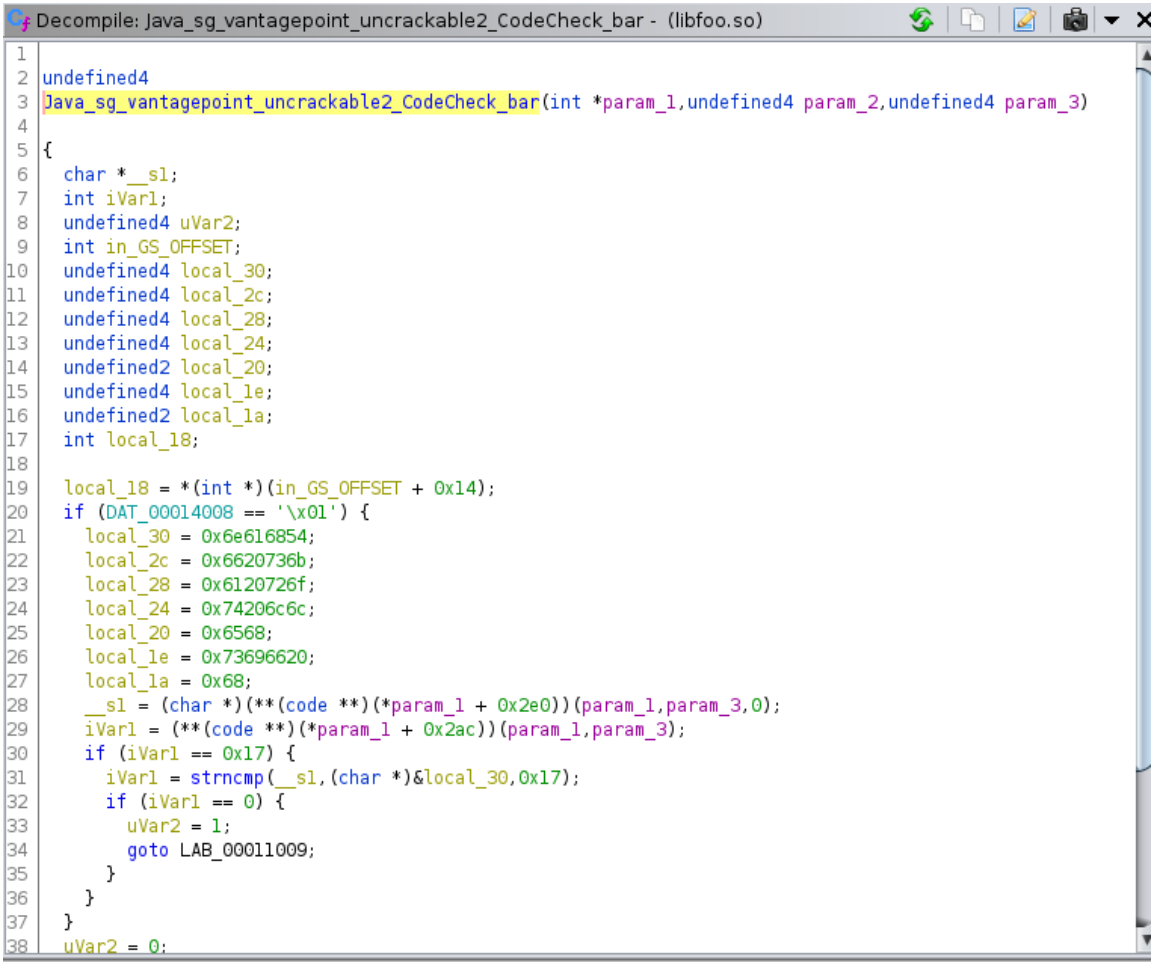
Σχήμα 6.40: Επιλογή βιβλιοθήκης x86/libfoo.so για ανάλυση

Μετά την ανάλυση της βιβλιοθήκης , κατευθυνθήκαμε στο παράθυρο Symbol Tree. Επεκτείνοντας το φάκελο Exports, εντοπίσαμε τη μέθοδο που μας ενδιέφερε δηλαδή την bar , καθώς την καλούσε η κλάση CodeCheck.



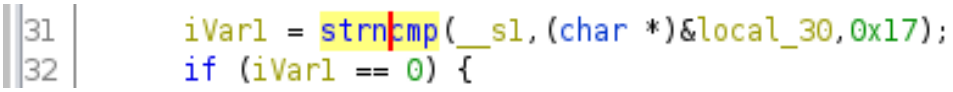
Σχήμα 6.41: Επιλογή exported μεθόδου CodeCheck_bar

Πατώντας «κλικ» πάνω της κατευθυνθήκαμε στον decompile κώδικα της μεθόδου bar.



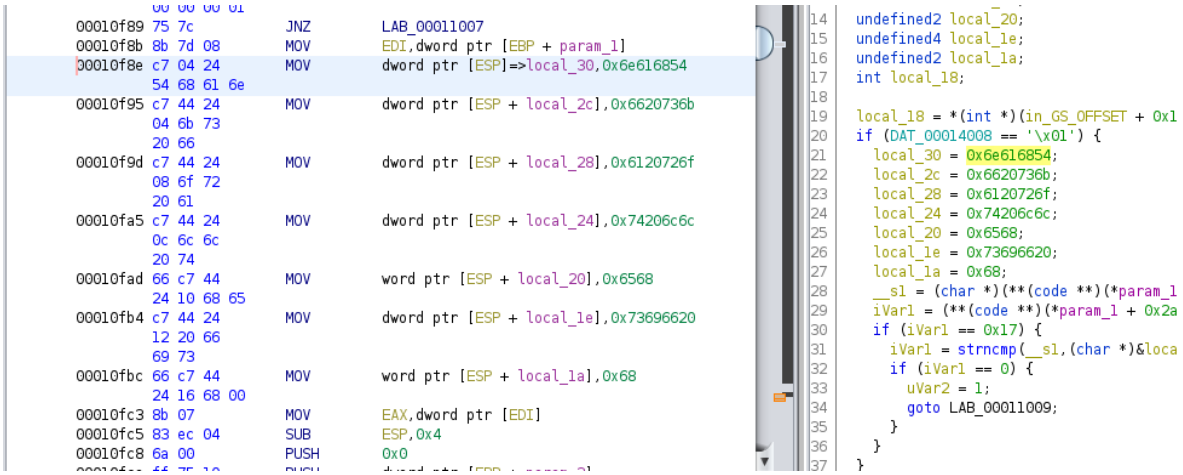
Σχήμα 6.42: Κώδικας μεθόδου bar

Αναλύοντας τον decompiled κώδικα της μεθόδου εντοπίσαμε στη γραμμή 32 μια άλλη μέθοδο, την strncmp(). Αυτή συγκρίνει το string που εισάγαμε με κάτι που αναφέρεται σαν local_30 και ελέγχει αν το string μας έχει μέγεθος όσο με 23 χαρακτήρες(0x17).



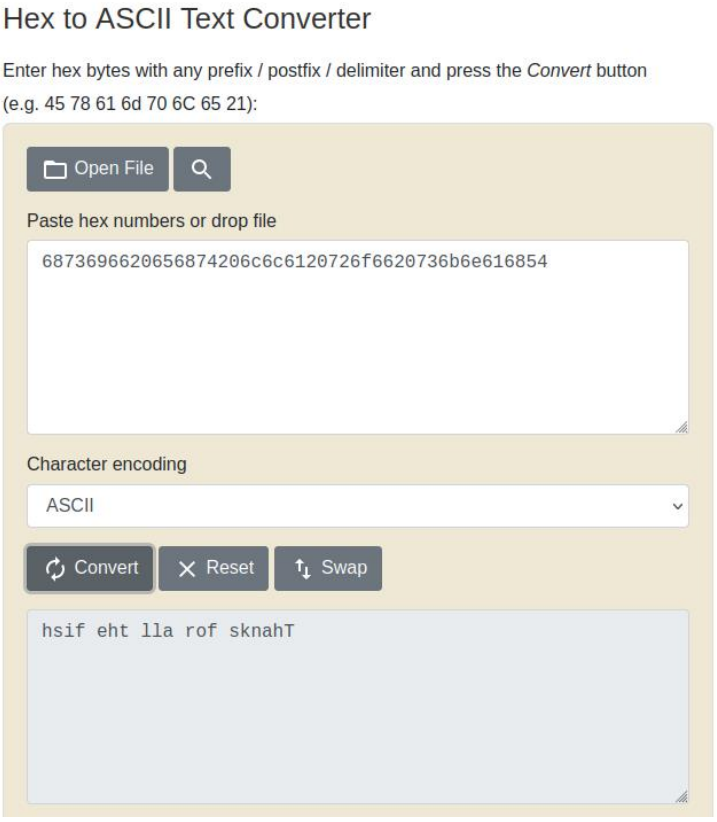
Σχήμα 6.43: Εντοπισμός μεθόδου σύγκρισης strncmp()

Πατώντας πάνω στη μεταβλητή local_30 , μεταφερθήκαμε στο disassembler. Όπως φαίνεται στον κώδικα της assembly, στη θέση local_30 αποθηκεύεται ένα string.



Σχήμα 6.44: Προβολή μεταβλητής local_30 στον disassembler

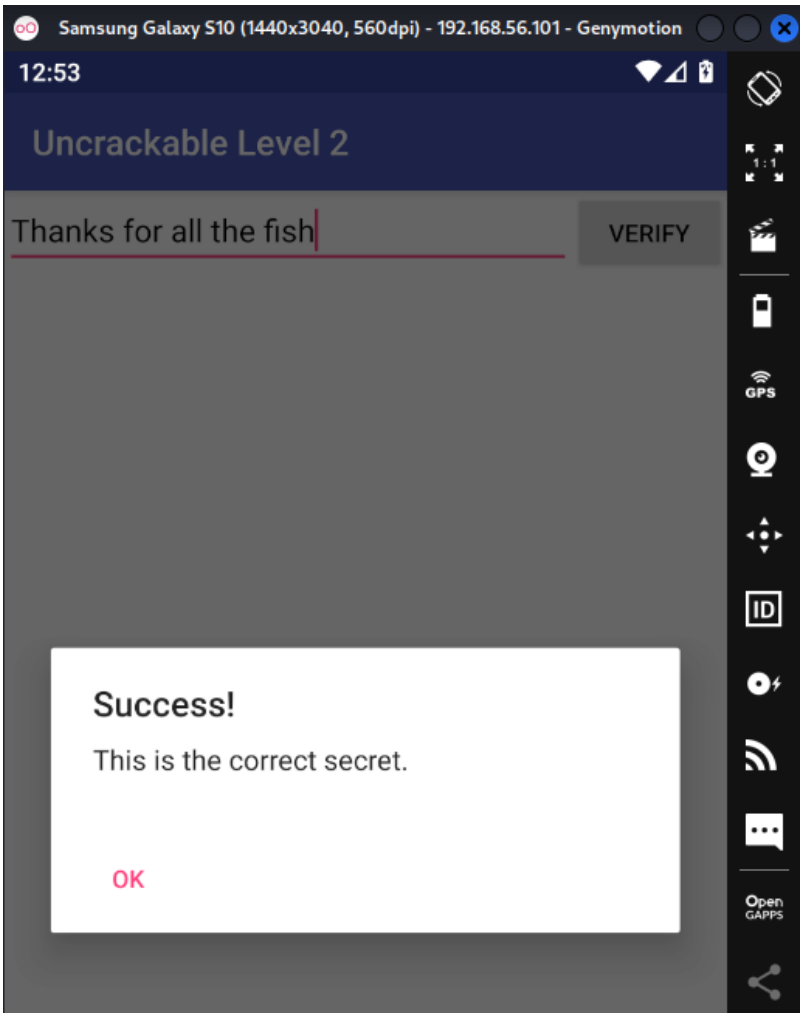
Το string ήταν αποθηκευμένο σε δεκαεξαδική μορφή. Συνεπώς, έπρεπε να το μετατρέψουμε σε κατανοητή μορφή δηλαδή σε ASCII χαρακτήρες. Έτσι, χρησιμοποιήσαμε έναν από το πολλούς online Hex to ASCII converters.



Σχήμα 6.45: Μετατροπή Hex σε ASCII

Εάν αντιστρέψουμε την πρόταση , το string που προκύπτει είναι το: **Thanks for all the fish**.

Αυτήν τη φορά πληκτρολογώντας το στο *text field* και πατώντας το κουμπί *VERIFY*, το *Alert* που εμφανίστηκε έγραφε το μήνυμα “Success!”



Σχήμα 6.46: Εμφάνιση μηνύματος σωστού secret string

Στο σημείο αυτό ολοκληρώθηκαν οι δοκιμασίες της 2ης εφαρμογής επιτυχώς, καθώς παρακάμψαμε το root detection και βρήκαμε το σωστό secret string.

Κεφάλαιο 7ο: Συμπεράσματα και προτάσεις βελτίωσης

7.1 Συμπεράσματα

Στα προηγούμενα κεφάλαια αναλύσαμε λεπτομερώς την αρχιτεκτονική του λογισμικού Android και εξετάσαμε τη λειτουργία των εφαρμογών του. Μολονότι που το Android διαθέτει ορισμένους μηχανισμούς ασφάλειας, όπως η απομόνωση των δεδομένων της εφαρμογής και η εκτέλεση κώδικα από άλλες εφαρμογές, ωστόσο εξακολουθούν να υπάρχουν πολλά ζητήματα ασφάλειας από την μεριά των προγραμματιστών που επηρεάζουν αρνητικά τους χρήστες.

Σκοπός την εργασίας ήταν η ανάδειξη της σημαντικότητας του penetration testing στις εφαρμογές των κινητών συσκευών πριν την επίσημη κυκλοφορία τους στο κοινό. Στις δυο εφαρμογές της εργασίας που επιλέχθηκαν, διαπιστώθηκε η ευπάθεια reverse engineering, που σύμφωνα με τον OWASP ανήκει στη θέση 9 της Top 10 Risk List ευπαθειών για κινητές συσκευές. Αυτό μας επίτρεψε να έχουμε πρόσβαση στον σχεδόν πηγαίο κώδικα της εφαρμογής, να τον μελετήσουμε, να κατανοήσουμε τον τρόπο λειτουργίας του και να τον επεξεργαστούμε.

Δεν υπάρχει κάποιος τρόπος που να αποτρέπει ολοκληρωτικά το reverse engineering στις εφαρμογές, αλλά μπορεί να κάνει τη διαδικασία του δύσκολη. Μερικές από τις τεχνικές που βοηθούν σε αυτό είναι: η χρήση του Proguard με σκοπό το obfuscation του κώδικα, η σωστή ανάπτυξη κώδικα των σημαντικών σημείων της εφαρμογής σε C/C++, κ.ά. Η βέλτιστη όμως λύση για την αποφυγή του reverse engineering είναι η μεταφορά των σημαντικών τμημάτων κώδικα σε έναν απόμακρο server. Για παράδειγμα, η δημιουργία ενός REST API στο server, το οποίο δέχεται τα δεδομένα από μια εφαρμογή, τρέχει τον αλγόριθμο και επιστρέφει τα αποτελέσματα.

Όπως προαναφέρθηκε, παρατηρείται μια ανοδική πορεία στη δημιουργία εφαρμογών του λειτουργικού συστήματος Android. Απόρροια του προηγούμενου είναι η αύξηση επιθέσεων στις εφαρμογές. Αυτό συμβαίνει γιατί σε πολλές περιπτώσεις οι προγραμματιστές δε γνωρίζουν τις βέλτιστες πρακτικές για τη δημιουργία εφαρμογών με υψηλό επίπεδο ασφάλειας και δεν ορίζεται το MAPT ως βασικό στάδιο στη διαδικασία ανάπτυξης μιας εφαρμογής.

Όσο περισσότερες είναι οι εφαρμογές με ευπάθειες, τόσο πιο εύκολα θα προσβάλλεται ο χρήστης και οι βασικοί πυλώνες της ασφάλειας, δηλαδή η εμπιστευτικότητα, η ακεραιότητα και η διαθεσιμότητα. Εν κατακλείδι, συμπεραίνουμε ότι η διαδικασία εύρεσης ευπαθειών μέσω του penetration testing είναι ένα ισχυρό στοιχείο στην προστασία των δεδομένων των χρηστών και των εφαρμογών.

7.2 Μελλοντικές Επεκτάσεις

Κύριος σκοπός των μελλοντικών επεκτάσεων είναι η εκτέλεση των δύο ίδιων πειραμάτων με διαφορετικές τεχνικές, με στόχο την εύρεση όλων των πιθανών ευπαθειών των εφαρμογών. Επιπροσθέτως, η δημιουργία ενός αυτόματου εργαλείου για την εύρεση τρωτών σημείων κατά την εκτέλεση μιας εφαρμογής σε Android συσκευή αποτελεί έναν επιπλέον στόχο για το μέλλον. Τέλος, η παρούσα εργασία μπορεί να επεκταθεί προσθέτοντας τις λύσεις και των άλλων δύο εφαρμογών του OWASP (Uncrackable App Level 3 & 4) παρέχοντας μια ολοκληρωμένη εικόνα όσο αφορά την ασφάλεια των εφαρμογών σε Android και τη μεθοδολογία του Mobile Application Penetration Testing.

Κεφάλαιο 8ο: Βιβλιογραφία

[1] Statista,"Number of smartphone users from 2016 to 2021", *Statista*, August 2021. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] Statcounter, "Mobile Operating System Market Share Worldwide", *Statcounter*, December 2021. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[3] Statista, "Mobile app downloads worldwide from 2018 to 2025, by store", *Statista*, September 2021. [Online]. Available: https://www.statista.com/statistics/1010716/apple-app-store-google-play-app-downloads-forecast/

[4] Financial Times, "WhatsApp voice calls used to inject Israeli spyware on phones", Mehul Srivastava, May 2019. [Online]. Available: https://www.ft.com/content/4da1117e-756c-11e9-be7d-6d846537acab?segmentid=acee4131-99c2-09d3-a635-873e61754ec6

[5] The Guardian, "What is Pegasus spyware and how does it hack phones?", D. Pegg, S. Cutler, July 2021. [Online]. Available: https://www.theguardian.com/news/2021/jul/18/what-is-pegasus-spyware-and-how-does-it-hack-phones

[6] GeeksforGeeks, "Top Programming Languages for Android App Development", *GeeksforGeeks*, November 2021. [Online]. Available: https://www.geeksforgeeks.org/top-programming-languages-for-android-app-development/

[7] Dignited, ""Android 10" (not Android Q) is the official name of the next Android OS", *Dignited*, August 2019. [Online]. Available: https://www.dignited.com/51900/android-10-not-android-q-is-the-official-name-of-the-next-android-os/

[8] Javapoint ,"Android Versions", *Javapoint*. [Online]. Available: https://www.javatpoint.com/android-versions

[9] "Platform Architecture", Developer Android. [Online]. Available: https://developer.android.com/guide/platform

[10] "Virtual Machine in Android: Everything you need to know", T. H. Doan, Medium. [Online]. Available: https://medium.com/android-news/virtual-machine-in-android-everything-you-need-to-know-9ec695f7313b

[11] V. J. M. Perez, *A study of vulnerabilities on Android systems*, Master Thesis, Valencia Polytechnic University, September 2013

[12] L. Xu, *Techniques and Tools for Analyzing and Understanding Android Applications*, PhD thesis, University of California, 2013.

[13] "Android Security Part 1: App Basics", L. Schiefer, hiqes.com, February 2014. [Online]. Available: http://hiqes.com/android-security-part-1/

[14] N. Elenkov, *Android Security Internals*, An In-Depth Guide to Android's Security Architecture, San Francisco, 2015

[15] "Application Signing", Source Android. [Online]. Available: https://source.android.com/security/apksigning/

[16] Imperva, "Penetration testing", *Imperva*. [Online]. Available: https://www.imperva.com/learn/application-security/penetration-testing/

[17] TechTarget, "pen testing (penetration testing)", *TechTarget*, P. Mehta. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/penetration-testing

[18] A. Singhal, T. Winograd, K. Scarfone, *Guide to Secure Web Services*, NIST Special Publication 800-95, Computer Security Division Information Technology Laboratory National Institute of Standards and Technology, Gaithersburg, 128 pages, (August 2007)

[19] Infosec, "Automated Tools vs a Manual Approach", *Infosec Institute*, March 2018. [Online]. Available: https://resources.infosecinstitute.com/topic/automated-tools-vs-a-manual-approach/

[20] "Penetration Testing and the GDPR", Itgovernance. [Online]. Available: https://www.itgovernance.co.uk/green-papers/penetration-testing-and-the-gdpr

[21] T. Wilhelm, *Professional Penetration Testing*. USA, 2010, pp. 13-41

[22] TechBeamers, "Five Types of Penetration Test to Zero in Potential Vulnerabilities", *TechBeamers*, M. Agarwal. [Online]. Available: https://www.techbeamers.com/penetration-test-and-types/

[23] S. Kimmel, J. Selvidge, J. Firch, S. Stankovic, A. Bond, A Beginners Guide To Understanding Penetration Testing, PurpleSec, 2020

[24] Cipher, "The Types of Pentests You Must Know About", *Cipher*. [Online]. Available: https://cipher.com/blog/the-types-of-pentests-you-must-know-about/

[25] Edureka, "What Is Penetration Testing – Methodologies and Tools", *Edureka*, A. Choudary, November 2020. [Online]. Available: https://www.edureka.co/blog/what-is-penetration-testing/

[26] Infosec, "What are black box, grey box, and white box penetration testing?", *Infosec Institute*. [Online]. Available: https://resources.infosecinstitute.com/topic/what-are-black-box-grey-box-and-white-box-penetration-testing/#gref

[27] M. E. Khan, "Different approaches to black box testing technique for finding errors", International Journal of Software Engineering & Applications (IJSEA), Al Musanna College of Technology, Sultanate of Oman, October 2011

[28] M.E. Khan, F. Khan, "A comparative Study of White Box, Black Box and Grey Box Testing Techniques", International Journal of Advanced Computer Science and Applications(IJACSA),Vol. 3, No.6, 2012

[29] Infosec, "How are penetration teams structured?", Infosec Institute. [Online]. Available: https://resources.infosecinstitute.com/topic/how-are-penetration-teams-structured/

[30] V. K. Velu, *Mobile Application Penetration Testing*, Birmingham, UK, 2016

[31] Wired, "The Under Armour Hack Was Even Worse Than It Had To Be", Wired, March 2018. [Online]. Available: https://www.wired.com/story/under-armour-myfitnesspal-hack-password-hashing/

[32] Infosec, "Introduction to the mobile application penetration testing methodology", *Infosec Institute*, January, 2019. [Online]. Available: https://resources.infosecinstitute.com/topic/introduction-mobile-application-penetration-testing-methodology/

[33] ThreatIntelligence, "A Guide to Mobile Application Penetration Testing", *ThreatIntelligence*, July 2021. [Online]. Available: https://www.threatintelligence.com/blog/mobile-application-penetration-testing

[34] AndreaFortuna, "Reverse engineering and penetration testing on Android apps", *AndreaFortuna*, July 2019. [Online]. Available: https://www.andreafortuna.org/2019/07/18/reverse-engineering-and-penetration-testing-on-android-apps-my-own-list-of-tools/

[35] E. Shlomo, "Mobile Penetration Testing Kits and Tools", Eshlomo. [Online]. Available: https://www.eshlomo.us/mobile-penetration-testing-kit/

[36] OWASP, [Online]. Available: https://owasp.org/

[37] OWASP, "OWASP mobile security", *OWASP*, [Online]. Available: https://owasp.org/www-project-mobile-security/

[38] Sectigostore, "OWASP Mobile Top 10 Vulnerabilities & Mitigation Strategies", *Sectigostore*, December 2020. [Online]. Available: https://sectigostore.com/blog/owasp-mobile-top-10/

[39] Cyphere, "OWASP Mobile Top 10 Security Vulnerabilities and Attack Prevention", *The Cyphere*. [Online]. Available: https://thecyphere.com/blog/owasp-mobile-top-10/

[40] E. Barker, *Guideline for Using Cryptographic Standards in the Federal Government: Cryptographic Mechanisms*, NIST Special Publication 800-175B, Computer Security Division Information Technology Laboratory National Institute of Standards and Technology, Gaithersburg, 77 pages, March 2016

[41] OWASP, "Buffer Overflow Attack", OWASP, [Online]. Available: https://owasp.org/www-community/attacks/Buffer_overflow_attack

[42] Kaspersky, "Zeus Virus", Kaspersky, [Online]. Available: https://usa.kaspersky.com/resource-center/threats/zeus-virus

[43] P. Beaten, "How to root Android phones and tablets (and unroot them)", *Digitaltrends*, [Online]. Available: https://www.digitaltrends.com/mobile/how-to-root-android/