

UNIVERSITÀ
DELLA CALABRIA

DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA,
ELETTRONICA E SISTEMISTICA (DIMES)

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di laurea

Requisiti e Verifica della Sicurezza delle Piattaforme Mobili: Lo Standard OWASP MASVS

Relatori:

Prof. Andrea Pugliese
Ing. Michele Ianni

Candidato:

Lorenzo Morelli
Mat. 207038

ANNO ACCADEMICO 2020/2021

*Non smettere MAI di crederci.
per il Lollo del futuro, dal Lollo del passato.*

Indice

Introduzione	ii
1 OWASP e MASVS	1
1.1 Open Web Application Security Project	1
1.1.1 Quali sono i principi dell'OWASP	3
1.1.2 Tecniche OWASP per la qualità del codice	4
1.2 Mobile Application Security Verification Standard	8
1.2.1 Valutazione e certificazione	16
2 Categorie di requisiti	20
2.1 V1: Architecture, Design and Threat Modeling Requirements . .	23
2.2 V2: Data Storage And Privacy Requirements	26
2.3 V3: Cryptography Requirements	33
2.4 V4: Authentication And Session Management Requirements . .	39
2.5 V5: Network Communication Requirements	46
2.6 V6: Platform Interaction Requirements	53
2.7 V7: Code Quality And Build Setting Requirements	59
2.8 V8: Resilience Requirements	65
Conclusioni	78
Bibliografia	80

Introduzione

Il coinvolgimento dei dispositivi mobile all'interno delle vite di tutti è ormai una realtà ben chiara alla massa. Molto meno chiaro è l'impatto e le conseguenze che derivano da questo coinvolgimento. I dispositivi mobile si sono evoluti molto negli ultimi anni, partiti come dispositivi estremamente limitati destinati a persone con specifiche esigenze di business ed arrivati ora ad essere praticamente indispensabili per la vita di tutti i giorni. Sia che l'uso sia finalizzato all'intrattenimento o al business, l'immediatezza e l'utilità di questi dispositivi è agli occhi di tutti, mentre c'è sempre meno consapevolezza di cosa forniscono gli utenti ai provider per poter utilizzare i loro servizi. La sicurezza di tali dispositivi è un punto critico nell'attuale società, ormai gli smartphone sono diventati talmente presenti nelle nostre vite da essere presi costantemente di mira dagli hacker. Questo avviene perché una persona diventa più esposta, e di conseguenza più facilmente ricattabile, se si ha libero accesso al proprio smartphone per dieci minuti piuttosto che libero accesso alla propria residenza per una settimana. I dispositivi mobile, ed in particolare gli smartphone, sono dispositivi estremamente versatili, pieni di funzionalità e con un'enorme disponibilità di software sotto-forma di applicazione. I sistemi operativi mobile adoperano diversi sistemi hardware e/o software per raggiungere determinati gradi di sicurezza mentre spesso le criticità derivano proprio dalle applicazioni. Rendere disponibile sullo store un'applicazione mobile non è un processo complicato e proprio per questo motivo che spesso tante di queste applicazioni sono sviluppate da team o sviluppatori che non hanno basi di sicurezza informatica. Questa inesperienza conduce l'utente finale a ritrovarsi per le mani un software altamente vulnerabile che nel peggiore dei casi gestisce dati sensibili

("particolari" per il nuovo GDPR). L'esistenza di linee guida sullo sviluppo di applicazioni mobile sicure è una necessità sempre più sentita nel panorama europeo in quanto, nell'ultimo periodo, è aumentata la consapevolezza di dover porre l'attenzione su sicurezza e privacy, dovuta soprattutto all'entrata in vigore del nuovo regolamento europeo in materia di trattamento dei dati personali e privacy.

Il MASVS è un documento sviluppato da OWASP e pensato per supportare sviluppatori e tester durante il ciclo di sviluppo di applicazioni mobile. Esso ha lo scopo di definire delle linee guida per lo sviluppo ed il testing di applicazioni mobile sicure definendo un insieme di categorie e requisiti che il software deve possedere per poter essere ritenuto conforme allo standard secondo i tre livelli di sicurezza previsti dallo standard stesso ovvero L1, L2, R. Il MASVS (attualmente alla versione 1.3) è continuamente aggiornato e supportato dalla rete di contributori che contraddistingue tutto il materiale che OWASP mette a disposizione sulle varie piattaforme online liberamente consultabile.

How to Contribute

The MASVS is an open source effort and we welcome contributions and feedback. If you want to contribute additional content, or improve existing content, we suggest that you first contact us on the OWASP MSTG Slack channel:

https://owasp.slack.com/messages/project-mobile_omtg/details/

You can sign up here:

https://owasp.slack.com/join/shared_invite/zt-g398htpy-AZ40HOM1WUOZguJKbblqkw#/

Before you start contributing, please check our [contribution guide](#) which should get you started.

Figura 1: Contributi OWASP

I progetti che OWASP porta avanti per quanto riguarda la sicurezza delle applicazioni mobile sono due ovvero il MASVS ed il MSTG. Il secondo tratta più nello specifico le tecniche che gli sviluppatori devono mettere in atto per poter rispettare i requisiti di alto livello imposti nel MASVS. Analizzando i due documenti ci si imbatte spesso in descrizioni accurate su quali siano le problematiche di sviluppo che causano un determinato rischio di sicurezza ma in pochissimi casi sono presenti dei progetti software (magari open-source) su

cui sia possibile concretamente mettere mano e testare al meglio le cause dei rischi.

Il lavoro di tesi è strutturato in due parti: nella prima vengono trattati il MASVS e la fondazione OWASP, mentre nella seconda vengono esposte categorie e requisiti del documento ed infine vengono discusse le demo associate ai requisiti con i relativi attacchi. L'obiettivo della tesi, infatti, è quello di sviluppare delle demo software (vulnerabili e non) e degli attacchi su tali software. Le prime servono per comprendere, e far comprendere, in maniera approfondita le tecniche proposte ed i requisiti descritti, i secondi sono necessari per evidenziare quali siano le problematiche a cui viene esposto il software quando un determinato requisito non viene rispettato a dovere. lo scopo del lavoro è quello di essere di supporto, a sviluppatori e tester, alla comprensione pratica dei requisiti ed a dare consapevolezza che spesso le tecniche messe in atto per rispettare un requisito, anche se appesantiscono il codice, sono necessarie alla prevenzione di attacchi potenzialmente critici e distruttivi. L'auspicio è quello di proporre questo approccio di sviluppo di test demo, accompagnate da un esempio di attacco, per ogni requisito di ogni categoria perché consentirebbe a tutti di poter testare con mano le app mobile e renderebbe più consapevoli, sia delle tecniche che delle conseguenze, tutti gli sviluppatori che si approcciano al mondo della sicurezza in ambito mobile.

Capitolo 1

OWASP e MASVS

In questo capitolo si andrà ad introdurre il Mobile Application Security Verification Standard (MASVS v1.3, attualmente la revisione più recente) e l'organizzazione che lo ha realizzato ovvero L'Open Web Application Security Project (OWASP). Il MASVS è stato l'elemento cardine del progetto in quanto ha guidato sia la stesura della tesi che la realizzazione delle demo grazie alle varie informazioni rese disponibili dall'OWASP sul proprio repository Github. Il MASVS è il risultato dello sforzo coeso tra la comunità Open-Source e le esigenze del mondo aziendale per stabilire un framework di sicurezza composto da vari requisiti necessari alla progettazione, sviluppo e test di applicazioni mobile (iOS e Android) sicure. Come quasi tutto il materiale fornito da OWASP anche il MASVS è pensato per evolversi e migliorare grazie al contributo della comunità. Prima di introdurre il MASVS è necessario approfondire l'organizzazione che da anni contribuisce al miglioramento della qualità software, in ambito di sicurezza, in tutte le sue possibili fasi e sfaccettature: la Open Web Application Security Project.

1.1 Open Web Application Security Project

Come viene riportato nell'articolo [1], l'Open Web Application Security Project® (OWASP) è una Fondazione no-profit che lavora da anni per migliorare la sicurezza dei software. Attraverso progetti software open-Source

guidati dalla comunità, centinaia di documenti e materiali universalmente distribuiti, decine di migliaia di membri, conferenze educative e formative, la fondazione OWASP è la fonte per sviluppatori e progettisti per rendere “sicuro il web”. Per quasi due decadi le aziende, fondazioni, sviluppatori e volontari hanno supportato la fondazione OWASP ed il suo lavoro.



Figura 1.1: Logo Owasp

Il progetto OWASP è nato nel 2001 con lo scopo di diffondere la cultura della qualità del codice nelle fasi di sviluppo di un software. Negli anni, la comunità OWASP ha prodotto articoli, metodologie, documenti e tool per supportare gli sviluppatori nella realizzazione di software e applicazioni sicure. Tutti possono accedere al materiale messo a disposizione dal sito ufficiale. La Community OWASP ha redatto delle linee guida per lo sviluppo sicuro degli applicativi web, grazie alla collaborazione di moltissimi professionisti del settore. I controlli OWASP sul codice sono annoverati tra le best practice riconosciute a livello internazionale nell’ambito della prevenzione di vulnerabilità di sicurezza e, se applicate correttamente, possono ridurre considerevolmente il rischio di data breach (violazioni dei dati) causati da eventuali incidenti informatici favoriti dalla scarsa qualità del codice di software e applicazioni. L’adozione dei controlli OWASP però, oltre a essere considerata una best practice facoltativa, in alcuni contesti sta progressivamente diventando un requisito obbligatorio. L’entrata in vigore della nuova normativa europea in materia di protezione dei dati personali e di privacy (GDPR) ha accelerato questo processo. L’OWASP pubblica e aggiorna periodicamente una guida ai controlli sul codice dalla metà degli anni 2000. La guida è giunta alla versione 4.2 (pubblicata nel 2020), ma il gruppo di volontari, composto da decine di specialisti di programmazione e cybersecurity, sta lavorando alla versione 5 (è

possibile trovare maggiori informazioni e seguire gli aggiornamenti all’indirizzo su GitHub).

1.1.1 Quali sono i principi dell'OWASP

Entrando nel dettaglio dei principi che propone il framework OWASP, il più banale nonché il più importante è quello per cui la sicurezza va inquadrata come un processo, complesso e costante nel tempo, e non come prodotto. È sicuramente molto comodo sia per una questione di costi che per una semplificazione del processo, pensare che uno scanner di vulnerabilità e un firewall garantiscano una efficace e rapida protezione contro la quasi totalità delle vulnerabilità. Così non è. Infatti, i software utilizzati per i vulnerability assessment sono soltanto il primo gradino di una scala che va percorsa in tutta la sua profondità per garantire livelli di sicurezza adeguati. L’OWASP propone un approccio diverso rispetto al concetto di correzione della singola vulnerabilità tramite patch (senza andare ad indagare approfonditamente la causa di quella vulnerabilità). La sicurezza viene analizzata ed integrata nel ciclo di vita del software (SDLC – Software Development Life Cycle) per evitare il propagarsi ed il ripetersi di vulnerabilità all’interno di un’applicazione. Integrare l’analisi della sicurezza nell’SDLC si collega ad un altro principio dell’OWASP e del buon senso, ovvero che scovare un bug all’inizio del ciclo di vita del software permette una soluzione più rapida e meno costosa lavorando ad un livello di complessità inferiore “test early and test often”. Esistono moltissimi prodotti, commerciali e open source, che possono aiutare lo specialista ad automatizzare moltissime attività e velocizzarle. In questo contesto è importante capire esattamente cosa possono e non possono fare i singoli strumenti scelti per evitare che vengano utilizzati in modo errato o incompleto. È importante, utilizzando diversi software e strumenti, tenere traccia dei risultati e dei test effettuati per sviluppare delle metriche che permettano allo specialista di tracciare la tendenza al miglioramento della sicurezza dell’applicazione analizzata. La metrica deve permettere allo specialista (o al team coinvolto nelle analisi) di verificare che il numero delle vulnerabilità scenda proseguendo nelle analisi. Il progetto

OWASP Metrics fornisce alcune metriche standard per supportare il processo di analisi senza arenarsi nel difficile compito di ricavare delle metriche personalizzate da zero. Se le metriche aiuta a verificare il costante miglioramento della salute del software in tutto il ciclo di sviluppo, la documentazione dei risultati dei test passo per passo, permette di tenere traccia di tutte le verifiche effettuate e di condividere con tutti gli attori coinvolti nello sviluppo lo stato dell'arte delle verifiche di sicurezza.

1.1.2 Tecniche OWASP per la qualità del codice

Di seguito viene riportata una panoramica su quattro tecniche di test che possono essere utilizzate nella costruzione di un processo di verifica della qualità del codice.

Ispezioni e revisioni manuali Le Ispezioni e revisioni manuali sono condotte utilizzando la documentazione a supporto del software o coinvolgendo direttamente gli sviluppatori e gli ideatori della struttura architetturale del software. La tecnica prevede la raccolta di informazioni su processi, servizi e tecnologie utilizzati, ai fini di quantificare la probabilità che emergano problemi di sicurezza ed effettuare i relativi controlli. Tale tecnica è importante per comprendere a fondo se le persone coinvolte nello sviluppo hanno compreso e implementato il processo di sicurezza durante la progettazione e lo sviluppo del software. Tra i vantaggi di questa tecnica si considera il fatto che non richieda un programma a supporto, può essere applicata a varie situazioni ed è particolarmente flessibile. Coinvolge tutto il team e si effettua tendenzialmente già all'inizio del ciclo di vita dello sviluppo del software con tutti i vantaggi che comporta l'approccio del “test early test often”. Sicuramente alcuni dei punti deboli di tale tecnica sono dati dal fatto che l'implementazione richiede molto tempo, non sempre il materiale a supporto (manuali, commenti, schemi) è disponibile e l'inesperienza di chi conduce le verifiche può portare all'inefficacia delle stesse.

Modellazione delle minacce La modellazione si pone come obiettivo quello di aiutare i progettisti a pensare quali possano essere le minacce che le loro applicazioni potrebbero dover affrontare. Può essere vista come una valutazione del rischio per le applicazioni. Il consiglio, anche in questo caso, è di sviluppare un modello di minacce ad inizio SDLC, seguendo preferibilmente l'approccio proposto dallo standard NIST 800-30 (National Institute of Standard and Technology) che prevede:

- la scomposizione dell'applicazione tramite verifica manuale del codice per comprendere il funzionamento del sistema e le sue sotto funzionalità;
- la definizione e classificazione delle risorse suddividendole in beni materiali e immateriali e classificandole per importanza aziendale;
- valutare tutte le potenziali vulnerabilità, tecniche, operazioni e gestioni;
- valutare le potenziali minacce mettendosi dal punto di vista del potenziale attaccante del sistema;
- creare una strategia di mitigazione del rischio, sviluppando controlli per attenuare le minacce ritenute realisticamente probabili dopo gli step di analisi precedenti.

La guida OWASP delinea la metodologia di modellazione delle minacce seguendo i passaggi del NIST e può essere utilizzata come modello di riferimento per testare le applicazioni informatiche.

Revisione del codice sorgente Per revisione del codice si intende il processo di verifica manuale del codice sorgente di un'applicazione alla ricerca di problemi di sicurezza. Moltissime tipologie di vulnerabilità non possono essere riscontrate con gli altri metodi. Tutti gli specialisti in sicurezza informatica sono concordi nell'affermare che non vi è nessuna tecnica migliore del cercare di identificare direttamente nel codice sorgente le vulnerabilità. Tramite l'analisi del codice sorgente, il tester esamina ogni singola riga di codice del

software e classifica con precisione ogni eventuale tipologia di vulnerabilità riscontrata. Esempi di vulnerabilità difficili da riscontrare con le altre tecniche, ma che possono essere individuate con la source code review sono ad esempio i problemi di concorrenza delle risorse, problemi sulla crittografia e problemi di controllo degli accessi; i trojan o gli easter egg, i malware o altre tipologie di codice dannoso. Sicuramente è una tecnica di analisi particolarmente completa, approfondita e veloce, che richiede però specialisti di sicurezza altamente qualificati che sappiano entrare molto nello specifico della singola linea di codice.

Il penetration testing Il penetration testing (o pentesting) di un applicativo è una tipologia di tecnica attiva attraverso la quale il tester tenta di forzare l'accesso allo stesso, simulando un attacco da parte di un hacker (da qui il termine ethical hacker o white hacker) e mettendo il sistema sotto stress. Esistono due principali categorie di pentesting:

- Il whitebox test: Il tester dispone delle credenziali di un normale utente (quindi non credenziali da amministratore) e ha il compito di capire fino a dove un normale utente ha la possibilità di accedere a sezioni dell'applicazione al di fuori dei permessi garantiti alla sua tipologia di utenza o sezioni non ad accesso pubblico;
- Il blackbox test: Il tester non è in possesso di alcuna informazione se non l'url dove è pubblicato l'applicativo. È a carico del tester riuscire ad accedere all'applicazione sfruttando tutte le tecniche possibili per acquisire anche le credenziali e forzare l'accesso (ad esempio con attacchi di tipo brute force). Le credenziali possono essere recuperate direttamente o con tecniche di social engineering, scam o phishing, inducendo in errore l'utente che fornirà direttamente a sua insaputa le credenziali all'attaccante.

Nello svolgimento dei pentest, il tester ha il compito di sfruttare tutte le falle che rendono vulnerabile la sicurezza dell'applicativo. Dal momento che

gli applicativi vengono utilizzati in specifici ecosistemi (es. sistemi operativi) o si appoggiano a funzioni e risorse esterne (es. database o web server), il tester può utilizzare le sue conoscenze per sfruttare eventuali falle dei sistemi e dei servizi che dialogano con esso per effettuare un accesso fraudolento. Un importante responsabilità del pentester è anche quella di rendicontare il tutto mantenendo un documento in cui vengono descritte le tecniche utilizzate e le vulnerabilità sfruttate per tutta la fase di testing. Questo aspetto è molto importante in quanto mette in luce le criticità note del sistema e permette di dare delle valutazioni e delle priorità alla messa in sicurezza del tutto. Questa valutazione è strettamente legata alle competenze del pentester ma il concetto chiave è che nel software possono essere presenti diverse falle di sicurezza (alcune immediatamente sfruttabili ed altre meno) ed è importante riuscire a mettere in sicurezza quelle menzionate dal pentester il più in fretta possibile in quanto si ha la certezza che siano sfruttabili. In ambito aziendale spesso non vengono risolte tutte le vulnerabilità nel minor tempo possibile in quanto sarebbe un approccio utopistico, per questo vengono effettuati dei Vulnerability assessment per avere un quadro generale delle criticità che gravano sul software. Ovviamente non basta avere questi documenti per rendere il software sicuro ma tali report consentono all'azienda di programmare al meglio il futuro (immediato) con lo scopo di risultare conforme alle normative in materia di sicurezza di dati personali e privacy.

Per concludere, tutte le organizzazioni che si occupano di sviluppo software e applicativi non possono più prescindere dal conoscere e padroneggiare le tecniche e i controlli OWASP. In particolar modo, se le soluzioni software progettate da tali società sono poi vendute a grandi player o a pubbliche amministrazioni. In ogni modo, parafrasando Eoin Keary, membro dell'OWASP Global Board, se anche le piccole realtà e gli sviluppatori freelance conoscessero e applicassero le tecniche OWASP, “il mondo sarebbe un luogo dove i software non sicuri sarebbero l'anomalia e non la norma” ed in un mondo del genere, i rischi di violazione dei dati (data breach) sarebbero notevolmente ridotti, con enormi benefici per i diritti e le libertà delle persone.

1.2 Mobile Application Security Verification Standard

Le rivoluzioni tecnologiche possono avvenire rapidamente. Meno di un decennio fa, gli smartphone erano dispositivi limitati, giocattoli costosi per utenti con specifiche esigenze di business. Oggi, gli smartphone sono una parte essenziale delle nostre vite. Siamo arrivati a fare affidamento su di loro per le informazioni, la navigazione e la comunicazione, e sono onnipresenti sia negli affari che nella nostra vita sociale. Ogni nuova tecnologia introduce nuovi rischi per la sicurezza e stare al passo è una delle sfide principali che il mondo della cyber-security deve affrontare. Le difese sono sempre un passo indietro. Per esempio, l'approccio utilizzato da molti è stato quello di applicare i vecchi modi di fare, e di pensare, alle tecnologie odierne. Gli smartphone sono come piccoli computer, e le applicazioni mobile sono proprio come il software classico, quindi sicuramente i requisiti di sicurezza sono simili, purtroppo non è così semplice. I sistemi operativi degli smartphone sono diversi da quelli dei desktop, e le app mobile sono diverse dalle web app. Ad esempio, il metodo classico della scansione dei virus basata sulla firma non ha senso negli ambienti dei moderni sistemi operativi mobile: Non solo è incompatibile con il modello di distribuzione delle app mobile, ma è anche tecnicamente impossibile a causa delle restrizioni del sandboxing. Inoltre, alcune classi di vulnerabilità, come i buffer overflow e gli attacchi XSS, sono meno rilevanti nel contesto delle generiche app mobile rispetto, ad esempio, alle app desktop e alle applicazioni web (con le dovute eccezioni). Nel corso del tempo, questo settore ha ottenuto una maggiore consapevolezza sul panorama delle minacce mobile. A quanto pare, la sicurezza mobile è spesso incentrata sulla questione di protezione dei dati: Le app memorizzano le nostre informazioni personali, immagini, registrazioni, note, dati dell'account, informazioni aziendali, posizione e molto altro. Agiscono come client che ci collegano a servizi che usiamo quotidianamente, e come hub di comunicazione che elaborano ogni singolo messaggio che scambiamo con gli altri. Compromettendo lo smartphone di una persona si ottiene

un accesso senza filtri alla sua vita. Se consideriamo anche che i dispositivi mobile sono più soggetti ad essere persi o rubati ed i malware mobile sono sempre più comuni, la necessità di protezione dei dati diventa ancora più evidente. Uno standard di sicurezza per le app mobile deve quindi concentrarsi su come le app mobile gestiscono, memorizzano e proteggono le informazioni sensibili. Anche se i moderni sistemi operativi mobile come iOS e Android offrono buone API per l'archiviazione e la comunicazione sicura dei dati, queste devono essere implementate e usate correttamente dagli sviluppatori di app per essere efficaci. L'archiviazione dei dati, la comunicazione tra le app, l'uso corretto delle API crittografiche e la comunicazione di rete sicura sono solo alcuni degli aspetti che richiedono un'attenta considerazione. Una questione importante che necessita di un consenso da parte dell'industria è: Quanto è necessario spingersi nella protezione della riservatezza e dell'integrità dei dati. Per esempio, la maggior parte degli specialisti sarebbero d'accordo che un'applicazione mobile dovrebbe verificare il certificato del server in uno scambio TLS. Ma che dire del pinning del certificato SSL? Non farlo comporta una vulnerabilità? Dovrebbe essere un requisito se un'app gestisce dati sensibili, o forse è addirittura controproducente? Abbiamo bisogno di crittografare i dati memorizzati nei database SQLite, anche se il sistema operativo sfrutta il sandboxing delle app? Ciò che è appropriato per un'app potrebbe essere irrealistico per un'altra, ed è proprio da questo bisogno di risposte che nasce il MASVS. Come viene descritto nello standard [4], il MASVS è un tentativo di standardizzare questi requisiti utilizzando livelli di verifica che si adattano a diversi scenari di minaccia. Inoltre, la comparsa di root malware e strumenti di amministrazione remota ha creato la consapevolezza del fatto che i sistemi operativi mobile stessi hanno difetti sfruttabili, quindi le strategie di containerizzazione sono sempre più utilizzate per offrire una protezione aggiuntiva ai dati sensibili e prevenire la manomissione client-side. È qui che le cose si complicano. Questi meccanismi di sicurezza supportate in Hardware e le soluzioni di containerizzazione a livello di sistema operativo, come Android for Work e Samsung Knox, esistono, ma non sono disponibili in modo consisten-

te su diversi dispositivi. In alternativa, è possibile implementare misure di protezione software, ma purtroppo non ci sono standard o processi per verificare questo tipo di protezioni. Di conseguenza, i report dei test di sicurezza delle app mobile sono discordanti. Per esempio, alcuni tester riportano una mancanza di offuscamento o di rilevamento di root in un'app Android come "falla di sicurezza". D'altra parte, misure come la crittografia, il rilevamento di debugger o l'offuscamento del flusso di controllo non sono considerate obbligatorie. Tuttavia, questo modo binario di vedere le cose non ha senso perché la resilienza non può essere trattata con un approccio binario: Dipende dalle particolari minacce da cui ci si vuole difendere. Le protezioni software non sono inutili, ma alla fine possono essere aggirate, quindi non devono mai essere usate in sostituzione ai controlli di sicurezza.



Figura 1.2: MASVS

L'obiettivo generale del MASVS è quello di offrire una linea guida per la sicurezza delle applicazioni mobile (MASVS-L1), permettendo anche l'inclusione di misure di difesa in profondità (MASVS-L2) e protezioni contro le minacce client-side (MASVS-R). Il MASVS ha lo scopo di ottenere quanto segue:

- Fornire requisiti per architetti e sviluppatori di software che cercano di sviluppare applicazioni mobile sicure;
- Offrire uno standard industriale che può essere testato nelle revisioni di sicurezza delle applicazioni mobile;
- Chiarire il ruolo dei meccanismi di protezione del software nella sicurezza mobile e fornire requisiti per verificare la loro efficacia;
- Fornire raccomandazioni specifiche su quale livello di sicurezza sia raccomandato per diversi casi d'uso.

OWASP è consapevole che il 100% del consenso dell'industria è impossibile da ottenere. Tuttavia, il MASVS è utile per fornire una guida in tutte le fasi dello sviluppo e del test delle applicazioni mobile. Il MASVS può essere utilizzato per stabilire un livello ben preciso di sicurezza delle app mobile. I requisiti sono stati sviluppati con i seguenti obiettivi:

- come metrica: Per fornire uno standard di sicurezza rispetto al quale le app mobile esistenti possono essere confrontate e valutate da sviluppatori e proprietari di applicazioni;
- come guida: Per fornire una guida durante tutte le fasi di sviluppo e test delle app mobile;
- durante l'acquisto: Per fornire una linea di base per la verifica della sicurezza delle app mobile.

Modello di sicurezza Il MASVS definisce due livelli di verifica della sicurezza (MASVS-L1 e MASVS-L2), così come una serie di requisiti di resilienza sul reverse engineering (MASVS-R). MASVS-L1 contiene requisiti di sicurezza generici che sono raccomandabili a tutte le app mobile, mentre MASVS-L2 dovrebbe essere applicato alle app che gestiscono dati altamente sensibili. MASVS-R copre ulteriori controlli di protezione che possono essere applicati

se la prevenzione delle minacce lato client risulta essere un obiettivo di progettazione. Soddisfare i requisiti di MASVS-L1 porta ad un'app sicura che segue le best practice di sicurezza e non soffre di vulnerabilità comuni. MASVS-L2 aggiunge ulteriori controlli di difesa (in profondità) come il pinning SSL, ottenendo un'app che è resistente ad attacchi più sofisticati, assumendo che i controlli di sicurezza del sistema operativo mobile siano intatti e che l'utente finale non sia visto come un potenziale avversario. Infine, soddisfare tutti o alcuni sottoinsiemi dei requisiti di protezione del software in MASVS-R aiuta ad impedire specifiche minacce client-side in cui l'utente finale è malintenzionato e/o il sistema operativo mobile è compromesso. Il MASVS raccomanda anche due accorgimenti da prendere in considerazione per comprendere al meglio la divisione dei livelli e le differenze:

- Anche se OWASP raccomanda l'implementazione dei controlli MASVS-L1 in ogni applicazione, l'implementazione di un controllo o meno dovrebbe essere in definitiva una decisione basata sul rischio.
- Si noti che i controlli di protezione del software elencati in MASVS-R e descritti nella OWASP Mobile Security Testing Guide possono essere aggirati e non devono mai essere utilizzati in sostituzione ai controlli di sicurezza. Sono invece destinati ad aggiungere ulteriori controlli di protezione specifici per le minacce alle app che soddisfano anche i requisiti MASVS in MASVS-L1 o MASVS-L2.

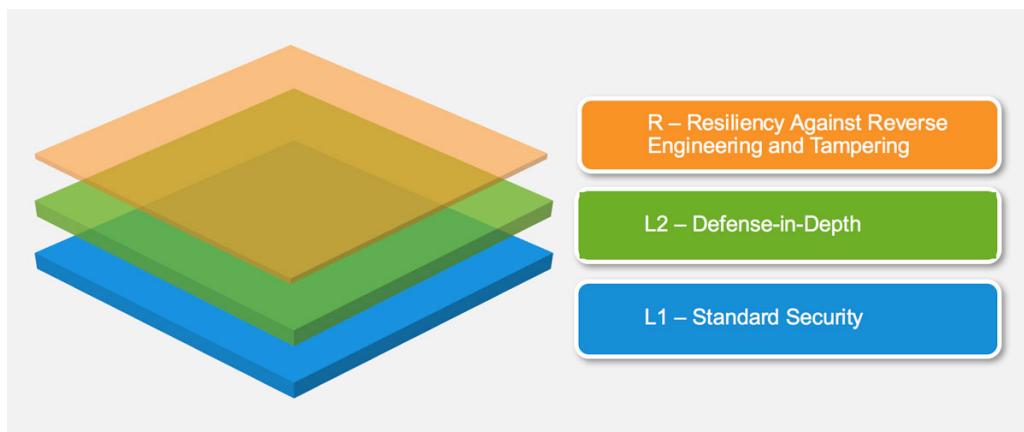


Figura 1.3: MASVS layers

Struttura documento La prima parte del MASVS contiene una descrizione del modello di sicurezza e dei livelli di verifica disponibili, seguita da raccomandazioni su come utilizzare lo standard nella pratica. I requisiti di sicurezza dettagliati, insieme ad una mappatura dei livelli di verifica, sono elencati nella seconda parte. I requisiti sono stati raggruppati in otto categorie (da V1 a V8) in base all'obiettivo tecnico da raggiungere. La seguente nomenclatura è utilizzata in tutto il MASVS e MSTG:

- Categoria di requisiti: MASVS-Vx, ad esempio MASVS-V2: Conservazione dei dati e privacy;
- Requisito: MASVS-Vx.y, ad esempio MASVS-V2.2: "Nessun dato sensibile deve essere scritto nei log dell'applicazione".

MASVS-L1: Sicurezza standard Un'applicazione mobile che ottiene il livello MASVS-L1 aderisce alle best practice di sicurezza delle applicazioni mobile. Soddisfa i requisiti di base in termini di qualità del codice, gestione dei dati sensibili e interazione con l'ambiente mobile. Deve essere fatto un processo di test per verificarne i controlli di sicurezza. Questo livello è appropriato per tutte le applicazioni mobile.

MASVS-L2: Sicurezza in profondità MASVS-L2 introduce controlli di sicurezza avanzati che vanno oltre i requisiti standard. Per soddisfare MASVS-L2, deve esistere un modello di minaccia e la sicurezza deve essere parte integrante dell'architettura e del design dell'applicazione. Sulla base del modello delle minacce, i giusti controlli MASVS-L2 dovrebbero essere stati selezionati e implementati con successo. Questo livello è appropriato per le app che gestiscono dati altamente sensibili, come le app di mobile banking.

MASVS-R: Resilienza al reverse engineering e manomissioni L'app ha una sicurezza aggiornata allo stato dell'arte, ed è anche resiliente a specifici e ben definiti attacchi lato client, come la manomissione, il modding o il reverse engineering per estrarre codice o dati sensibili. Un'app di questo tipo

sfrutta caratteristiche di sicurezza hardware o tecniche di protezione software sufficientemente forti e verificabili. MASVS-R è applicabile alle app che gestiscono dati altamente sensibili e può servire come mezzo per proteggere la proprietà intellettuale o per rendere un'app a prova di manomissione.

Raccomandazioni d'uso Cercando di riassumere i concetti principali espresi precedentemente: le app possono essere verificate rispetto al MASVS L1 o L2 in base alla valutazione preventiva del rischio e al livello generale di sicurezza richiesto. L1 è applicabile a tutte le app mobile, mentre L2 è generalmente consigliato per le app che gestiscono dati e/o funzionalità più sensibili. MASVS-R (o parti di esso) può essere applicato per verificare la resilienza contro minacce specifiche, come il repackaging o l'estrazione di dati sensibili, oltre alla corretta verifica della sicurezza. In sintesi, sono disponibili i seguenti tipi di verifica:

- MASVS-L1
- MASVS-L1+R
- MASVS-L2
- MASVS-L2+R

Le diverse combinazioni riflettono diversi gradi di sicurezza e resilienza. L'obiettivo è di permettere flessibilità: Per esempio, un gioco mobile potrebbe non garantire l'aggiunta di controlli di sicurezza MASVS-L2 come l'autenticazione a 2 fattori per ragioni di usabilità, ma avere una forte esigenza aziendale per la prevenzione delle manomissioni e quindi raggiungere il livello MASVS-L1+R.

Quale livello scegliere L'implementazione dei requisiti di MASVS-L2 aumenta la sicurezza, mentre allo stesso tempo aumenta il costo di sviluppo e potenzialmente peggiora l'esperienza dell'utente finale (il classico trade-off). In generale, L2 dovrebbe essere usato per le app ogni volta che ha senso dal punto di vista del rischio rispetto al costo (cioè, quando la potenziale perdita causata

da una compromissione della riservatezza o dell'integrità risulta essere superiore al costo sostenuto dai controlli di sicurezza aggiuntivi). La valutazione del rischio dovrebbe essere il primo passo prima di applicare il MASVS.

Esempi di applicazione del MASVS in contesti reali

MASVS-L1 Tutte le app mobile. MASVS-L1 elenca le migliori pratiche di sicurezza che possono essere seguite con un impatto ragionevolmente basso sui costi di sviluppo e sull'esperienza dell'utente. Applica i requisiti del MASVS-L1 per tutte le app che non si qualificano per uno dei livelli superiori.

MASVS-L2 Industria sanitaria: App mobile che memorizzano informazioni personali che possono essere usate per il furto d'identità, pagamenti fraudolenti o una varietà di schemi di frode. Per il settore sanitario statunitense, le considerazioni sulla conformità includono la legge sulla portabilità dell'assicurazione sanitaria e la responsabilità (HIPAA), Privacy, sicurezza, regole di notifica delle violazioni e regola sulla sicurezza del paziente.

Industria finanziaria: App che permettono l'accesso a informazioni altamente sensibili come i numeri delle carte di credito, informazioni personali, o permettono all'utente di spostare fondi. Queste app garantiscono controlli di sicurezza aggiuntivi per prevenire le frodi. Le app finanziarie devono garantire la conformità al Payment Card Industry Data Security Standard (PCI DSS), Gramm Leech Bliley Act e Sarbanes-Oxley Act (SOX).

MASVS L1+R App mobile in cui la protezione della proprietà intellettuale (IP) è un obiettivo aziendale. I controlli di resilienza elencati in MASVS-R possono essere utilizzati per aumentare lo sforzo necessario per ottenere il codice sorgente originale e impedire la manomissione/cracking.

Industria videoludica: Giochi con la necessità di prevenire il modding ed il cheating, come i giochi online competitivi. Il cheating è un problema importante nei giochi online, poiché una grande quantità di cheater porta un malcontento generale dei giocatori che può causare il fallimento di un progetto.

MASVS-R fornisce controlli anti-tampering di base per aiutare ad aumentare lo sforzo per la creazione dei cheat.

MASVS L2+R Industria finanziaria: Applicazioni bancarie online che permettono all'utente di spostare fondi, dove tecniche come l'iniezione di codice e la strumentazione su dispositivi compromessi rappresentano un rischio. In questo caso, i controlli di MASVS-R possono essere utilizzati per impedire la manomissione, alzando il livello di difficoltà nello sviluppo di malware.

Tutte le app mobile che, per progettazione, hanno bisogno di memorizzare dati sensibili sul dispositivo mobile, e allo stesso tempo devono supportare una vasta gamma di dispositivi e versioni del sistema operativo. In questo caso, i controlli di resilienza possono essere utilizzati come una misura di difesa in profondità per aumentare lo sforzo per gli attaccanti che mirano a estrarre i dati sensibili. Le app con acquisti in-app dovrebbero idealmente utilizzare controlli lato server e MASVS-L2 per proteggere i contenuti a pagamento. Tuttavia, ci possono essere casi in cui non c'è la possibilità di utilizzare la protezione lato server. In questi casi, i controlli MASVS-R dovrebbero essere applicati in aggiunta al livello L2 per aumentare lo sforzo di reversing e/o manomissione.

1.2.1 Valutazione e certificazione

OWASP, come organizzazione no-profit neutrale, non certifica alcun fornitore, ente di verifica o software. Tutte queste asserzioni di garanzia, marchi di fiducia o certificazioni non sono ufficialmente verificate, registrate o certificate da OWASP, quindi un'organizzazione che si affida a tale visione deve essere cauta sulla fiducia riposta in qualsiasi terza parte o marchio di fiducia che rivendica la certificazione (M)ASVS. Questo non dovrebbe inibire le organizzazioni dall'offrire tali servizi di garanzia, a patto che non rivendichino una certificazione ufficiale OWASP.

Guida per la certificazione delle app mobile Il modo raccomandato per verificare la conformità di un'app mobile con il MASVS è quello di eseguire una

revisione "a libro aperto", il che significa che i tester hanno accesso a risorse chiave come progettisti e sviluppatori dell'app, documentazione del progetto, codice sorgente e accesso autenticato agli endpoint, compreso l'accesso ad almeno un account utente per ogni ruolo. È importante notare che il MASVS copre solo la sicurezza dell'app mobile (lato client) e la comunicazione di rete tra l'app e i suoi endpoint remoti, così come alcuni requisiti di base e generici relativi all'autenticazione dell'utente e alla gestione della sessione. Non contiene requisiti specifici per i servizi remoti (ad esempio i servizi web) associati all'app, salvo un insieme limitato di requisiti generici relativi all'autenticazione e alla gestione della sessione. Tuttavia, MASVS V1 specifica che i servizi remoti devono essere coperti dal modello generale delle minacce ed essere verificati rispetto a standard appropriati, come OWASP ASVS. Un'organizzazione che certifica deve includere in ogni rapporto l'ambito della verifica (in particolare se un componente chiave è fuori dall'ambito), un riassunto dei risultati della verifica, compresi i test passati e quelli falliti, con chiare indicazioni su come risolvere i test falliti. Conservare documenti di lavoro dettagliati, screenshot o filmati, script per sfruttare in modo affidabile e riproducibile un problema, e registrazioni elettroniche dei test, come i log dei proxy di intercettazione e le note associate. Purtroppo non è sufficiente eseguire uno strumento e riportare i fallimenti, questo non fornisce prove sufficienti che i problemi, ad un determinato livello di certificazione, siano stati testati e provati a fondo. In caso di controversia, ci dovrebbero essere sufficienti prove a supporto per dimostrare che ogni requisito verificato è stato effettivamente testato.

Usare la OWASP Mobile Security Testing Guide (MSTG) L'OWASP MSTG [5] è un manuale per testare la sicurezza delle app mobile. Descrive i processi tecnici per verificare i requisiti elencati nel MASVS. L'MSTG include un elenco di casi di test, ognuno dei quali corrisponde a un requisito del MASVS. Mentre i requisiti del MASVS sono di alto livello e generici, l'MSTG fornisce raccomandazioni approfondite e procedure di test in base allo specifico sistema operativo mobile.

Il ruolo degli strumenti di test di sicurezza automatizzati L'uso di scanner del codice sorgente e strumenti di test black-box sono incoraggiati al fine di aumentare l'efficienza quando possibile. Tuttavia, non è possibile completare la verifica MASVS utilizzando solo strumenti automatizzati: Ogni app mobile è diversa, e la comprensione dell'architettura generale, della logica di business e delle insidie tecniche delle specifiche tecnologie e dei framework utilizzati è un requisito obbligatorio per verificare la sicurezza dell'app.

Altri usi Come guida dettagliata all'architettura di sicurezza: Uno degli usi più comuni dello Standard di verifica della sicurezza delle applicazioni mobile è come risorsa per gli architetti della sicurezza. I due principali framework di architettura di sicurezza, SABSA o TOGAF, mancano di una grande quantità di informazioni che sono necessarie per completare le revisioni dell'architettura di sicurezza delle applicazioni mobile. MASVS può essere usato per colmare queste lacune, permettendo agli specialisti della sicurezza di scegliere controlli migliori per i problemi comuni alle applicazioni mobile.

Come sostituzione delle liste di controllo per la codifica sicura off-the-shelf: Molte organizzazioni possono trarre vantaggio dall'adozione del MASVS, scegliendo uno dei due livelli, o biforcando il MASVS e cambiando ciò che è richiesto per il livello di rischio di ogni applicazione in un modo specifico del dominio. Incoraggiamo questo tipo di biforcazione, purché venga mantenuta la tracciabilità, in modo che se un'applicazione ha superato il requisito 4.1, ciò significa la stessa cosa per le copie biforate man mano che lo standard si evolve.

Come base per le metodologie di test di sicurezza: Una buona metodologia di test di sicurezza per app mobile dovrebbe coprire tutti i requisiti elencati nel MASVS. La OWASP Mobile Security Testing Guide (MSTG) descrive casi di test black-box e white-box per ogni requisito di verifica.

Come guida per i test automatizzati di unità e integrazione: Il MASVS è progettato per essere altamente testabile, con la sola eccezione dei requisiti architettonici. I test automatizzati di unità, integrazione e accettazione basati

sui requisiti del MASVS possono essere integrati nel ciclo di vita dello sviluppo continuo. Questo non solo aumenta la consapevolezza della sicurezza da parte degli sviluppatori, ma migliora anche la qualità complessiva delle app risultanti e riduce la quantità di scoperte indesiderate durante i test di sicurezza nella fase di pre-rilascio.

Per la formazione sullo sviluppo sicuro: MASVS può anche essere usato per definire le caratteristiche delle app mobile sicure. Molti corsi di "programmazione sicura" sono semplicemente corsi di hacking etico con una leggera spalmatura di consigli di codifica. Questo non aiuta gli sviluppatori. Invece, i corsi di sviluppo sicuro possono usare il MASVS, con una forte attenzione ai controlli proattivi documentati nel MASVS, piuttosto che, ad esempio, la Top 10 dei problemi di sicurezza del codice.

Capitolo 2

Categorie di requisiti

In un mondo perfetto, la sicurezza dovrebbe essere considerata in tutte le fasi dello sviluppo. In realtà, però, la sicurezza è spesso presa in considerazione solo in una fase avanzata dell’SDLC. Oltre ai controlli tecnici, il MASVS richiede l’esistenza di processi che assicurino che la sicurezza sia stata esplicitamente affrontata durante la pianificazione dell’architettura dell’applicazione mobile, e che i ruoli funzionali e di sicurezza di tutti i componenti siano noti. Poiché la maggior parte delle applicazioni mobile agiscono come client di servizi remoti, bisogna assicurarsi che gli standard di sicurezza appropriati siano applicati anche a questi servizi, testare l’applicazione mobile in modo isolato non è sufficiente. In questo capitolo verranno descritte in maniera approfondita tutte le categorie ed i requisiti definiti dal MASVS e laddove fosse possibile, verranno descritte delle applicazioni “demo” che sono state implementate per due principali motivi:

- verificare e comprendere al meglio un determinato requisito di categoria;
- analizzare la conseguenza che scaturiscono dalla violazione di tale requisito.

Per ogni categoria, quindi, è stato scelto un requisito MSTG-ID che è stato tecnicamente approfondito utilizzando il materiale fornito da OWASP. Il materiale tecnico disponibile specifica la problematica che deriva dalla violazione di un determinato requisito, quali sono gli errori di programmazione/progetta-

zione che danno vita alla problematica e quali test (statici e dinamici) possono essere fatti per verificare la presenza di tale problematica. Il lavoro di tesi aggiunge un ulteriore tassello alla già completa documentazione sviluppata da OWASP ovvero quello di fornire degli ambienti di test in cui si evidenzi la problematica e gli attacchi che possono essere performati su applicazioni che violano un determinato requisito. In effetti nella documentazione messa a disposizione da OWASP non vi è nessun tipo di applicativo consultabile per comprendere al meglio la problematica (salvo rari casi incontrati in cui non è stato possibile visualizzare il codice sorgente dell'app). Si è ritenuto utile sviluppare alcuni esempi che forniscono più chiaramente la problematica descritta e le conseguenze che si possono avere su tale problematica.



Figura 2.1: Android projects

Sono state sviluppate delle demo laddove si è colta la necessità di avere un software pronto e consultabile per comprendere al meglio il requisito selezionato per la determinata categoria. Non tutte le categorie disponibili sono state coperte dalla demo d'esempio in quanto, alcune di esse, sono state ritenute inadatte e/o irrilevanti ai fini della comprensione del contenuto fornito dall'OWASP. Le categorie in questione sono:

- La categoria 1 non è stata presa in considerazione in quanto è l'unica categoria che non comprende requisiti tecnici delle 8 e non necessita di demo implementative per essere compresa al meglio. Al posto della demo è stata trattata la proposta del security.txt evidenziandone gli aspetti più interessanti;

- La categoria 5 è stata accantonata per la necessità di realizzare apparati di rete ben strutturati che ha reso l'idea della demo implementativa meno interessante essendoci già tool molto più avanzati ed immediati per comprendere al meglio la problematica.
- La categoria 8 è stata accantonata perché risulta essere una categoria speciale per cui vale l'approccio “arms race” ovvero vengono implementati dei meccanismi sempre più sofisticati per star dietro agli attacchi che anch'essi sono sempre più sofisticati; questo perché in questa categoria non ci sono delle soluzioni che riescono a risolvere il problema ma vengono analizzate delle tecniche che rendono il lavoro dell'attaccante più complesso ma pur sempre possibile. Proprio per questo approccio, come nella categoria 5, si è ritenuto poco interessante l'idea della demo che, come obiettivo dell'elaborato, dovrebbe essere un input pratico ed immediato messo a disposizione dell'utente e finalizzato alla miglior comprensione del problema.

Per tutte le altre categorie invece si è ritenuto opportuno implementare una demo software per poter toccare con mano l'argomento descritto nella documentazione e aver ben chiare le conseguenze che possono scaturire da piccoli errori, cercando di evidenziare come la messa in sicurezza dell'applicazione, rispetto all'attacco che è possibile performare, risulti essere nettamente l'opzione più vantaggiosa.

2.1 V1: Architecture, Design and Threat Modeling Requirements

Obiettivo del controllo La categoria "V1" elenca i requisiti relativi all'architettura e alla progettazione dell'applicazione. Proprio per questo, essa è l'unica categoria che non possiede casi di test tecnici nella OWASP Mobile Testing Guide. Per coprire argomenti come la modellazione delle minacce, lo SDLC sicuro o la gestione delle chiavi, gli utenti del MASVS dovrebbero consultare i rispettivi progetti OWASP e/o altri standard come quelli collegati qui sotto. Di seguito vengono riportati i requisiti della categoria:

#	MSTG-ID	Descrizione	L1	L2
1.1	MSTG-ARCH-1	Tutti i componenti dell'applicazione sono identificati e noti per essere necessari.	✓	✓
1.2	MSTG-ARCH-2	I controlli di sicurezza non sono mai applicati solo lato client ma anche sui rispettivi endpoint.	✓	✓
1.3	MSTG-ARCH-3	Deve essere ben definita un'architettura di alto livello per l'applicazione mobile e tutti i servizi remoti collegati e deve essere anche gestita la sicurezza in tale architettura.	✓	✓
1.4	MSTG-ARCH-4	I dati considerati sensibili nel contesto dell'applicazione mobile sono chiaramente identificati.	✓	✓
1.5	MSTG-ARCH-5	Tutti i componenti dell'applicazione sono definiti in termini di funzionalità di business e/o funzionalità di sicurezza che forniscono.		✓
1.6	MSTG-ARCH-6	Deve essere prodotto un modello di minaccia per l'app mobile e per i servizi remoti associati, che identifichi potenziali minacce e contromisure		✓

2.1 V1: Architecture, Design and Threat Modeling Requirements 24

#	MSTG-ID	Descrizione	L1	L2
1.7	MSTG-ARCH-7	Tutti i controlli di sicurezza hanno un'implementazione centralizzata.	✓	
1.8	MSTG-ARCH-8	Deve esistere una politica esplicita per come le chiavi crittografiche (se necessarie) vengono gestite, e il ciclo di vita delle chiavi crittografiche è applicato. Idealmente, è possibile seguire uno standard di gestione delle chiavi come NIST SP 800-57.	✓	
1.9	MSTG-ARCH-9	prevedere un meccanismo per imporre gli aggiornamenti dell'applicazione mobile.	✓	
1.10	MSTG-ARCH-10	La sicurezza deve essere affrontata in tutte le parti del ciclo di vita del software.	✓	
1.11	MSTG-ARCH-11	Deve essere prevista una informativa (es: trattamento dei dati personali) chiara e applicata in modo efficace.	✓	
1.12	MSTG-ARCH-12	L'applicazione deve essere conforme alle leggi e ai regolamenti sulla privacy.	✓	✓

Security.txt Quando vengono identificate delle fallo di sicurezza all'interno di servizi web da parte di ricercatori indipendenti spesso tali ricercatori riscontrano problemi nell'andare a comunicare le problematiche a chi di dovere, per inefficienza o addirittura per mancanza di canali di comunicazione ufficiali. Il risultato è che spesso le vulnerabilità non vengono riportate abbassando di fatto la qualità generale della sicurezza di tutto l'ecosistema basato sul web.

2.1 V1: Architecture, Design and Threat Modeling Requirements 25

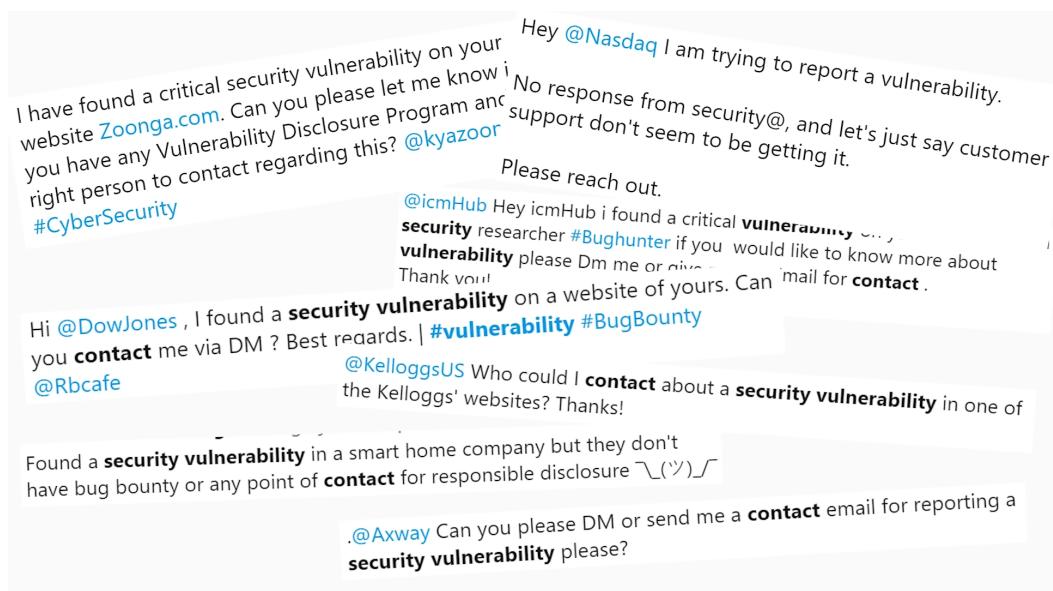


Figura 2.2: Tweet ricercatori

Security.txt [12] è una proposta di standard che ha lo scopo di consentire ai ricercatori di segnalare facilmente le vulnerabilità di sicurezza fornendo un canale ufficiale e sicuro per mettere in comunicazione queste due entità e di conseguenza semplificare il processo di scoperta e correzione delle vulnerabilità. Lo standard prevede semplicemente un file di testo chiamato "security.txt" in una posizione ben nota, simile nella sintassi a robots.txt ma destinato ad essere letto dalle persone che desiderano contattare il proprietario di un sito web per problemi di sicurezza. I file security.txt sono stati adottati già da note compagnie come Google, Dropbox e Facebook.

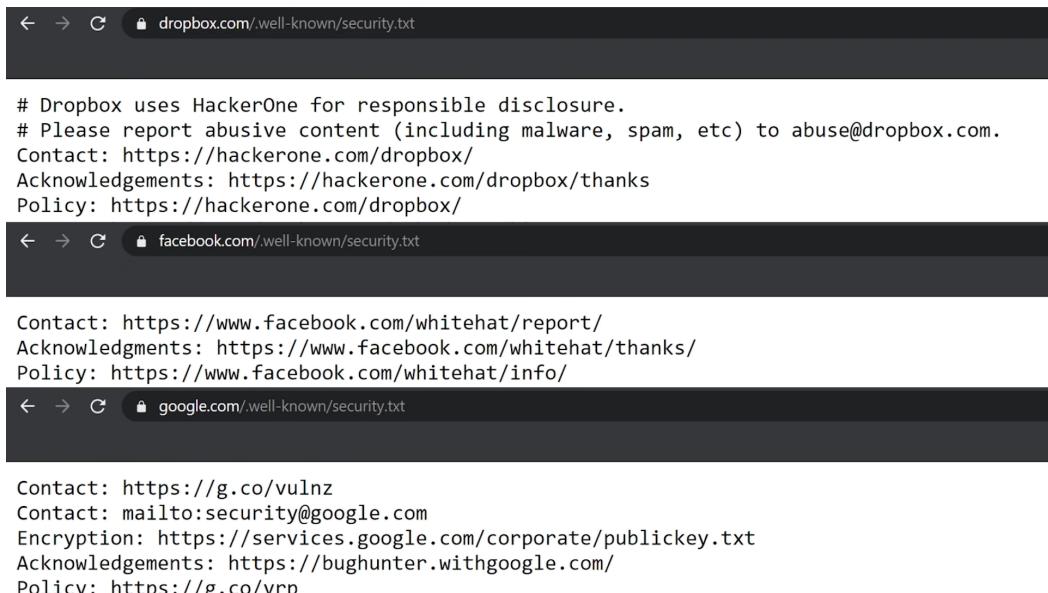


Figura 2.3: Esempi di security.txt

Security.txt è attualmente una bozza di standard che è stata sottoposta a revisione RFC e questo significa che è ancora nelle prime fasi di sviluppo. Per i siti web, il file security.txt dovrebbe essere messo sotto il percorso /.well-known/ (./.well-known/security.txt) [2]. Può anche essere messo nella directory principale (/security.txt) di un sito web, specialmente se la directory /.well-known/ non può essere usata per ragioni tecniche. Il file contiene le informazioni basilari per poter comunicare con i responsabili di sicurezza della piattaforma come: i contatti email o l'uri alla sezione per riportare vulnerabilità, chiavi pubbliche, politiche, metodi di cifratura utilizzati, scadenze delle chiavi ed in generale tutte le informazioni utili per la comunicazione rapida e sicura con chi gestisce la sicurezza della piattaforma.

2.2 V2: Data Storage And Privacy Requirements

ts

Obiettivo del controllo La protezione dei dati sensibili, come le credenziali dell’utente e le informazioni private, sono un punto chiave della sicurezza mobile. In primo luogo, i dati sensibili possono essere involontariamente esposti

ad altre app in esecuzione sullo stesso dispositivo se i meccanismi del sistema operativo come l'IPC (inter-process communication) sono utilizzati in modo improprio. I dati possono anche trapelare involontariamente da cloud storage, dai backup o dalla cache della tastiera. Inoltre, i dispositivi mobile possono essere persi o rubati più facilmente rispetto ad altri tipi di dispositivi, quindi lo scenario in cui un avversario ottiene l'accesso fisico al dispositivo è da non sottovalutare. In questo caso, possono essere implementate protezioni aggiuntive per rendere più difficile il recupero dei dati sensibili. Si noti che, poiché il MASVS è incentrato sulle applicazioni, non copre le politiche a livello di dispositivo come quelle applicate dalle soluzioni MDM (Mobile Device Management). Si incoraggia l'uso di tali politiche in un contesto aziendale per migliorare ulteriormente la sicurezza dei dati.

Definizione di dati sensibili I dati sensibili nel contesto del MASVS riguardano sia le credenziali dell'utente che qualsiasi altro dato considerato sensibile in un particolare contesto, come ad esempio:

- Informazioni di identificazione personale (PII) che possono essere abusate per il furto d'identità: Numeri di previdenza sociale, numeri di carte di credito, numeri di conti bancari, informazioni sanitarie;
- Dati altamente sensibili che porterebbero a danni alla reputazione e/o costi finanziari se compromessi: Informazioni contrattuali, informazioni coperte da accordi di non divulgazione, informazioni gestionali;
- Qualsiasi dato che deve essere protetto per legge o per ragioni di conformità.

#	MSTG-ID	Descrizione	L1	L2
2.1	MSTG-STORAGE-1	Devono essere utilizzati sistemi di “credential storage” per la memorizzazione di dati sensibili, come PII, credenziali utente o chiavi crittografiche.	✓	✓
2.2	MSTG-STORAGE-2	Nessun dato sensibile dovrebbe essere memorizzato al di fuori dello spazio dedicato all'app o del sistema di “credential storage”.	✓	✓
2.3	MSTG-STORAGE-3	Nessun dato sensibile deve essere scritto nei log dell'applicazione.	✓	✓
2.4	MSTG-STORAGE-4	Nessun dato sensibile deve essere condiviso con terze parti a meno che non sia una necessità architetturale.	✓	✓
2.5	MSTG-STORAGE-5	La cache della tastiera deve essere disabilitata sugli input di testo che elaborano dati sensibili.	✓	✓
2.6	MSTG-STORAGE-6	Nessun dato sensibile deve essere esposto tramite meccanismi IPC.	✓	✓
2.7	MSTG-STORAGE-7	Nessun dato sensibile, come password o pin, deve essere esposto attraverso l'interfaccia utente.	✓	✓
2.8	MSTG-STORAGE-8	Nessun dato sensibile deve essere incluso nei backup generati dal sistema operativo.		✓
2.9	MSTG-STORAGE-9	L'app deve rimuovere i dati sensibili dalle componenti (view, input di testo etc.) quando viene spostata in background.		✓
2.10	MSTG-STORAGE-10	L'app non tiene in memoria dati sensibili più a lungo del necessario e la memoria viene cancellata esplicitamente dopo l'uso.		✓

Continua nella prossima pagina

#	MSTG-ID	Descrizione	L1	L2
2.11	MSTG-STORAGE-11	L'app applica una politica (anche minima) di sicurezza per l'accesso al dispositivo, ad esempio richiedendo all'utente di impostare un codice di accesso.	✓	
2.12	MSTG-STORAGE-12	L'app istruisce l'utente sui tipi di informazioni personali trattate, nonché sulle migliori pratiche di sicurezza che l'utente dovrebbe seguire nell'utilizzo dell'app.	✓	
2.13	MSTG-STORAGE-13	Nessun dato sensibile dovrebbe essere memorizzato localmente sul dispositivo mobile. Invece, i dati dovrebbero essere recuperati da un endpoint remoto solo quando necessario e conservati in memoria.	✓	
2.14	MSTG-STORAGE-14	Se è proprio necessario memorizzare localmente dati sensibili, questi dovrebbero essere crittografati utilizzando una chiave derivata dalla memorizzazione hardware che richieda l'autenticazione.	✓	
2.15	MSTG-STORAGE-15	La memoria locale dell'app dovrebbe essere cancellata dopo un numero eccessivo di tentativi falliti di autenticazione.	✓	

Ricerca di informazioni sensibili negli screenshot generati automaticamente (MSTG-STORAGE-9) [3] Uno degli obiettivi principali dei produttori di applicazioni è quello di fornire agli utenti un'esperienza di utilizzo esteticamente piacevole, proprio per questa esigenza gli sviluppatori Android hanno introdotto la funzione di generazione automatica di screenshot da usare quando l'applicazione è in background per poter visualizzare un'an-

teprima nella schermata di gestione delle applicazioni. Questa funzione però può comportare un rischio per la sicurezza. Questa funzionalità può portare alla divulgazione di dati sensibili soprattutto in quelle app che gestiscono dati critici come le applicazioni bancarie e/o sanitarie. Per utilizzare questa funzionalità il sistema operativo effettua, in maniera del tutto automatica, degli screenshot che vengono memorizzati sul dispositivo. La vulnerabilità si presenta quando un'applicazione che sta mostrando dati sensibili viene messa in background e questa azione (malevole o meno) da parte dell'utente crea un file immagine da cui possono essere recuperate informazioni sensibili semplicemente usando un'applicazione malevola (se il dispositivo è rootato) o utilizzando tool appropriati ipotizzando che il dispositivo sia rubato e quindi l'attaccante ne abbia fisicamente l'accesso. Per esempio, catturare uno screenshot di un'applicazione bancaria può rivelare informazioni sul conto dell'utente, il credito, le transazioni e così via.

Analisi statica Lo screenshot dell'attività corrente viene generato quando l'app Android viene messa in background e viene successivamente visualizzato quando l'app torna in primo piano e questa azione può far trapelare informazioni sensibili. Per determinare se l'applicazione può esporre informazioni sensibili attraverso questa vulnerabilità, deve essere controllata l'opzione `FLAG_SECURE`. Se all'interno del codice della specifica attività che gestisce informazioni sensibili è configurato il `FLAG_SECURE` allora l'applicazione non permette il meccanismo degli screenshot. Il settaggio del `FLAG_SECURE` avviene in maniera molto semplice come descritto dal seguente codice java:

```
1 getWindow().setFlags(WindowManager.LayoutParams.FLAG_SECURE ,  
    WindowManager.LayoutParams.FLAG_SECURE);  
2 setContentView(R.layout.activity_main);
```

Se l'opzione invece risulta mancante, l'applicazione è vulnerabile alla cattura dello schermo.

Analisi dinamica testate il requisito in modalità black-box è molto semplice, basta andare in qualsiasi schermata che contiene informazioni sensibili

e cliccare il pulsante home per mandare l'app in background, poi premere il pulsante per il multitasking per vedere l'istantanea. Come mostrato di seguito, se il `FLAG_SECURE` è impostato (immagine a sinistra), l'istantanea sarà vuota; se il flag non è stato impostato (immagine a destra), verranno mostrate le informazioni sull'attività:

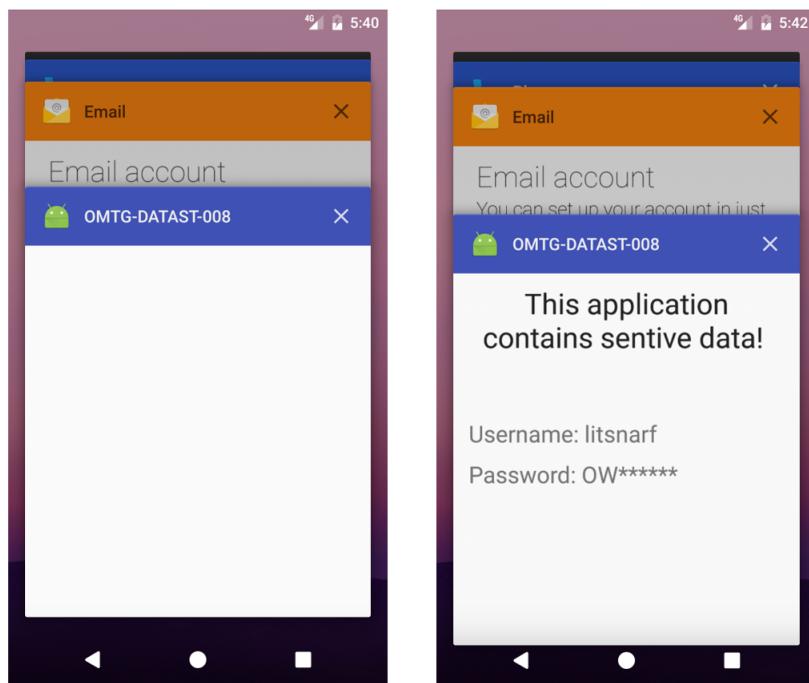


Figura 2.4: Sinistra: Flag abilitato, Destra: Flag disabilitato

Sui dispositivi che supportano la crittografia basata su file (FBE), le istantanee sono memorizzate nella cartella `/data/system_ce/<USER_ID>/<IMAGE_FOLDER_NAME>`. `IMAGE_FOLDER_NAME` dipende dal fornitore, ma i nomi più comuni sono `snapshots` e `recent_images`. Se il dispositivo non supporta FBE, viene usata la cartella `/data/system/<IMAGE_FOLDER_NAME>`. L'accesso a queste cartelle e alle istantanee richiede root.

Applicazione demo Per testare con mano la problematica è stata realizzata un'applicazione Android che permette il login tramite una classica form di autenticazione, l'applicazione è stata creata volontariamente vulnerabile e quindi viola il requisito preso in analisi in questo capitolo.

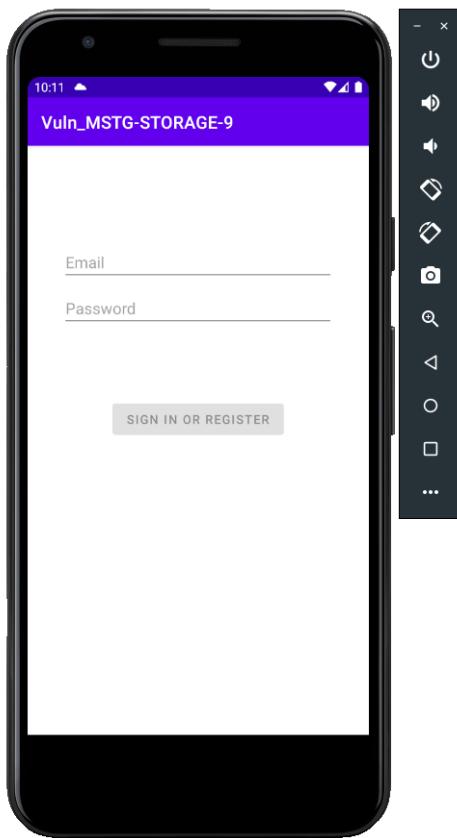


Figura 2.5: Android App vuln-MSTG-STORAGE-9

L’attacco effettuato dimostra come sia possibile ottenere accesso agli screenshot generati dal sistema operativo e raccogliere informazioni sensibili che possono essere sfruttate da un attaccante per poter avere accesso all’account dell’applicazione. In questo contesto l’attacco è stato performato ipotizzando di avere accesso al dispositivo e tramite l’utilizzo del tool `adb` si è riuscito a navigare, ottenendo i permessi di root, all’interno del file system ed a copiare nella cartella `download` gli screenshot dell’applicazione vulnerabile. Tale problematica espone il nome dell’account, la lunghezza della password e addirittura l’ultima lettera della password. Tali informazioni riducono drasticamente lo sforzo che un attaccante deve compiere per ottenere le credenziali d’accesso (per esempio Brute Force) esponendo l’utente a rischi molto gravi. Come detto in precedenza ci sono tantissime informazioni critiche che possono essere violate da questa tipologia di attacco soprattutto quando si parla di contesti bancari e/o sanitari.

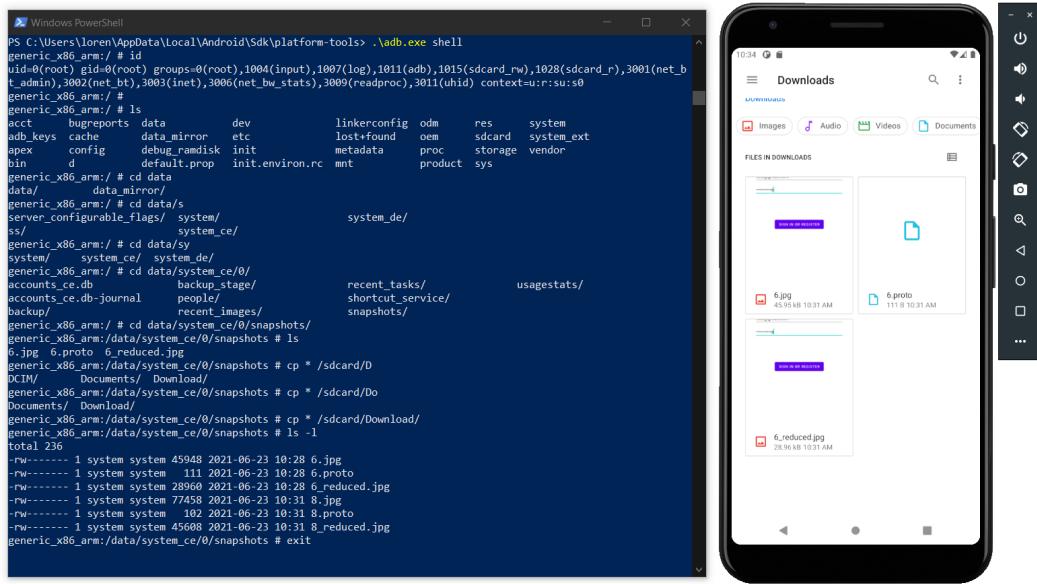


Figura 2.6: Test demo MSTG-STORAGE-9

2.3 V3: Cryptography Requirements

Obiettivo del controllo La crittografia è un ingrediente essenziale quando si tratta di proteggere i dati memorizzati su un dispositivo mobile. È una categoria in cui anche piccoli errori e/o distrazioni possono causare enormi danni, specialmente quando non si seguono le convenzioni standard. Lo scopo dei controlli in questo capitolo è di assicurare che l'applicazione usi la crittografia secondo le migliori pratiche del settore, tra cui:

- Uso di librerie crittografiche collaudate;
- Scelta e configurazione corretta delle primitive crittografiche;
- Scelta di generatori pseudo-casuali adeguati ovunque sia richiesta la casualità.

#	MSTG-ID	Descrizione	L1	L2
3.1	MSTG-CRYPTO-1	L'applicazione non si basa sulla crittografia simmetrica con "Hardcoded key" come unico metodo di crittografia.	✓	✓
3.2	MSTG-CRYPTO-2	L'applicazione utilizza implementazioni comprovate di primitive crittografiche.	✓	✓
3.3	MSTG-CRYPTO-3	L'applicazione usa primitive crittografiche che sono appropriate per il particolare caso d'uso, configurate con parametri che aderiscono alle migliori pratiche del settore.	✓	✓
3.4	MSTG-CRYPTO-4	L'applicazione non utilizza protocolli crittografici o algoritmi che sono ampiamente considerati deprecati per scopi di sicurezza.	✓	✓
3.5	MSTG-CRYPTO-5	L'applicazione non riutilizza la stessa chiave crittografica per scopi multipli.	✓	✓
3.6	MSTG-CRYPTO-6	Tutti i valori casuali sono generati utilizzando un generatore di numeri casuali sufficientemente sicuro.	✓	✓

Testare il generatore pseudo-casuale (MSTG-CRYPTO-6) [9] Questo requisito si concentra sul testing dei valori utilizzati dalle applicazioni che sfruttano generatori pseudo-casuali. Devono essere eseguiti i seguenti controlli:

- Identificare tutte le istanze in cui vengono utilizzati valori casuali e verificare tutte le istanze che tutte le istanze dei generatori siano della classe SecureRandom;
- Verificare se i generatori pseudo-casuali siano considerati crittograficamente sicuri;
- Verificare come vengono utilizzati i generatori psuedo-casuali;

- Verificare la casualità dei valori generati dall'applicazione.

Analisi statica È importante identificare tutte le istanze dei generatori pseudo-casuali e verificare se siano utilizzate implementazioni custom (sconsigliate) o classi note per essere insicure come `java.util.Random`. Questa classe produce una sequenza di numeri identica per uno stesso valore del seed e di conseguenza, ricostruendone lo stato del generatore, è possibile predire tutta la sequenza che dovrebbe essere impredicibile. Il seguente codice mostra l'utilizzo del generatore pseudo-casuale insicuro:

```
1 import java.util.Random;
2 // ...
3 Random number = new Random(123L);
4 // ...
5 for (int i = 0; i < 20; i++) {
6     // Generate another random integer in the range [0, 20]
7     int n = number.nextInt(21);
8     System.out.println(n);
9 }
```

Per ovviare al problema è necessario utilizzare algoritmi testati e considerati attualmente sicuri dagli esperti nel settore, selezionando implementazioni sicure con un la possibilità di gestire seed di lunghezza adeguata. È consigliato utilizzare il costruttore di default per la classe `SecureRandom` e quindi necessario identificare tutte le istanze del generatore dove non è stato utilizzando il costruttore di default e sostituirle. Specificare il valore del seed può ridurre la randomicità della sequenza e quindi preferibile utilizzare il costruttore senza argomenti che usa un seed di sistema per generare numeri (long) random a 128 byte. In generale, se una PRNG (Pseudo-Random number generator) non è considerata crittograficamente sicura (come `java.util.Random`) allora probabilmente è una PRNG statistica e non dovrebbe essere utilizzata in contesti di sicurezza. La conoscenza dello specific PRNG e la possibilità di indovinare il seed rende il generatore prevedibile, utilizzare seed da 128 bit è un buon punto di partenza per produrre numeri abbastanza randomici. Il codice a seguire

mostra come generare numeri “casuali” in maniera sicura:

```
1 import java.security.SecureRandom;
2 import java.security.NoSuchAlgorithmException;
3 // ...
4 public static void main (String args[]) {
5     SecureRandom number = new SecureRandom();
6     // Generate 20 integers 0..20
7     for (int i = 0; i < 20; i++) {
8         System.out.println(number.nextInt(21));
9     }
10 }
```

Come possiamo notare le accortezze da prendere sono davvero basilari, ma la sicurezza che ne deriva da queste scelte è drasticamente maggiore.

Analisi dinamica Una volta che l’attaccante è venuto a conoscenza del tipo di PRNG insicura utilizzata, è abbastanza semplice costruire un test che vada a predire il prossimo numero random che verrà generato dalla sequenza basandosi sui numeri precedentemente generati. Nel caso di generatori insicuri è possibile osservare il pattern di generazione statistico e ricostruire lo stato del generatore. Un ulteriore approccio può essere quello di decompilare l’APK e ispezionare il codice per comprendere al meglio quale PRNG è stata implementata. Si può anche testare la randomicità dei test con tool appositi andando a generare un quantitativo corposo di numeri e verificarne la qualità usando per esempio Burp’s sequencer.

Applicazione demo L’applicazione sviluppata per questo requisito presenta due tab in cui sono stati implementati i due generatori pseudo-casuali visti precedentemente nelle analisi: `java.util.Random` e `java.util.SecureRandom`.

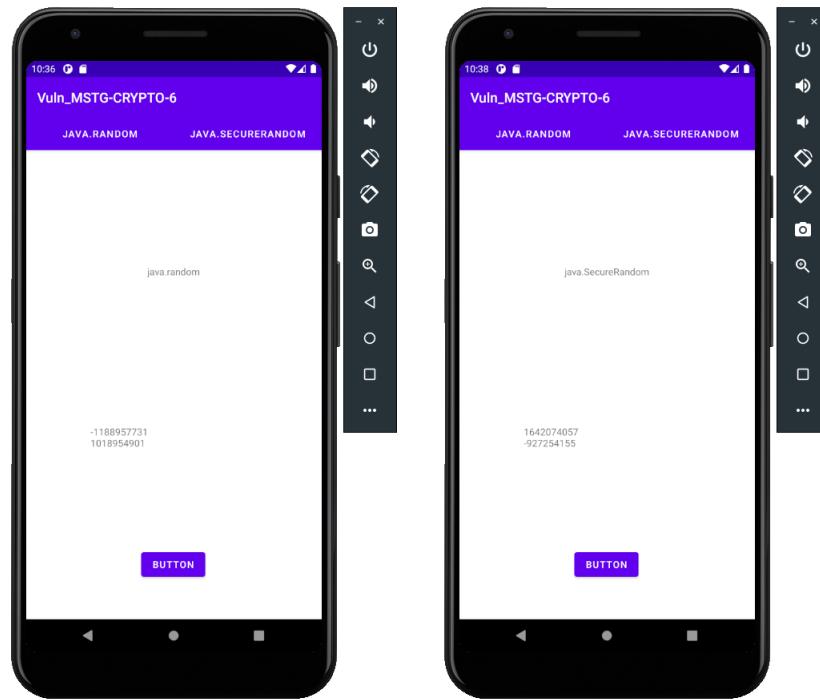


Figura 2.7: Android App MSTG-CRYPTO-6

Per dimostrare la vulnerabilità dovuta violazione di tale requisito è bastato performare un attacco che riuscisse a predire la sequenza del PRNG. Il codice dell'attacco preleva i primi due valori generati dal generatore `java.util.Random` e riesce a ricostruire lo stato ottenendo una copia del generatore e quindi sfruttando tale copia per predire qualsiasi numero random utilizzato all'interno dell'applicazione. Come è possibile vedere in figura 2.8, le sequenze di numeri sul terminale a sinistra prevedono il comportamento del generatore in app andando a predire i valori che il generatore dovrebbe generare in maniera impredicibile.

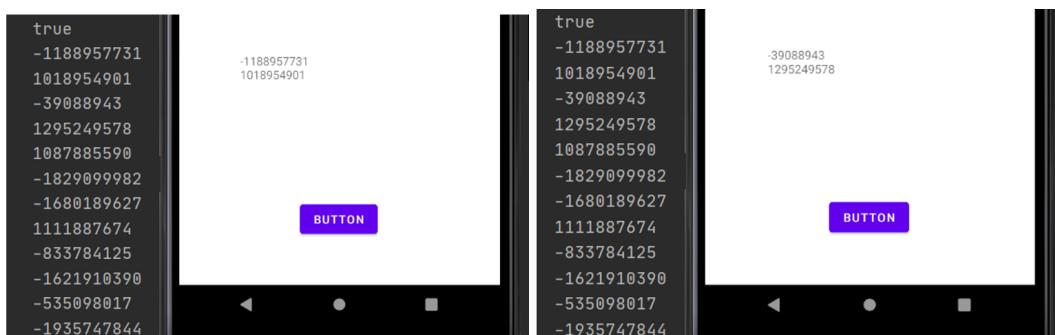


Figura 2.8: Test Demo MSTG-CRYPTO-6

Una volta dimostrata la predicitività della sequenza generata dal PRNG di conseguenza viene dimostrata anche la debolezza. Il codice utilizzato per predire la sequenza è il seguente:

```

1 public boolean replicateState(int nextN, int n, int nextM, int
2                               m) {
3
4     // Constants copied from java.util.Random
5
6     final long multiplier = 0x5DEECE66DL;
7     final long addend = 0xBL;
8     final long mask = (1L << 48) - 1;
9
10    long upperMOf48Mask = ((1L << m) - 1) << (48 - m);
11
12    // next(x) is generated by taking the upper x bits of 48
13    // bits of (oldSeed * multiplier + addend) mod (mask + 1)
14
15    // So now we have the upper n and m bits of two consecutive
16    // calls of next(n) and next(m)
17
18    long oldSeedUpperN = ((long) nextN << (48 - n)) & mask;
19    long newSeedUpperM = ((long) nextM << (48 - m)) & mask;
20
21    // Bruteforce the lower (48 - n) bits of the oldSeed that
22    // was truncated.
23
24    // Calculate the next seed for each guess of oldSeed and
25    // check if it has the same top m bits as our newSeed.
26
27    // If it does then the guess is right and we can add that
28    // to our candidate seeds.
29
30    ArrayList<Long> possibleSeeds = new ArrayList<Long>();
31
32    for (long oldSeed = oldSeedUpperN; oldSeed <= (
33        oldSeedUpperN | ((1L << (48 - n)) - 1)); oldSeed++) {
34
35        long newSeed = (oldSeed * multiplier + addend) & mask;
36
37        if ((newSeed & upperMOf48Mask) == newSeedUpperM) {
38
39            possibleSeeds.add(newSeed);
40
41        }
42
43    }
44
45    if (possibleSeeds.size() == 1) {
46
47        System.out.println("the seed is: " + possibleSeeds.
48                          toString());
49
50        //If there's only one candidate seed, then we found it!
51
52        //System.out.println("found seed: " + possibleSeeds.get
53        //                  (0));
54
55        setSeed(possibleSeeds.get(0) ^ multiplier); // setSeed(
56
57    }
58
59}

```

```

x) sets seed to `(x ^ multiplier) & mask`, so we need
another `^ multiplier` to cancel it out

26     return true;
27 }
28 if (possibleSeeds.size() >= 1) {
29     System.out.println("the seed is: " + possibleSeeds.
30     toString());
31     System.out.println("Didn't find a unique seed. Possible
32 seeds were: " + possibleSeeds);
33     return false;
34 } else {
35     System.out.println("Failed to find seed!");
36 }
37 return false;
38 }
```

2.4 V4: Authentication And Session Management Requirements

Obiettivo del controllo Nella maggior parte dei casi, l’accesso degli utenti ad un servizio remoto è parte integrante dell’architettura complessiva dell’applicazione mobile. Anche se la maggior parte della logica viene mantenuta dall’endpoint, MASVS definisce alcuni requisiti di base su come gli account utente e le sessioni devono essere gestite.

#	MSTG-ID	Descrizione	L1	L2
4.1	MSTG-AUTH-1	Qualsiasi servizio di accesso remoto o form di autenticazione (come l’autenticazione con nome utente/password) fornita dall’app, deve essere eseguita sull’endpoint remoto.	✓	✓

Continua nella prossima pagina

#	MSTG-ID	Descrizione	L1	L2
4.2	MSTG-AUTH-2	Se viene utilizzata una sessione stateful, l'endpoint remoto utilizza identificatori di sessione generati casualmente per autenticare le richieste del client senza mai inviare le credenziali dell'utente.	✓	✓
4.3	MSTG-AUTH-3	Se viene usata l'autenticazione basata su token, il server deve fornire un token generato e firmato usando un algoritmo sicuro.	✓	✓
4.4	MSTG-AUTH-4	L'endpoint remoto termina la sessione esistente quando l'utente si disconnette.	✓	✓
4.5	MSTG-AUTH-5	Devono esistere dei criteri sulla conformità della password che vengono applicati e verificati sull'endpoint remoto.	✓	✓
4.6	MSTG-AUTH-6	L'endpoint remoto implementa un meccanismo di protezione contro l'invio di credenziali errate un eccessivo numero di volte.	✓	✓
4.7	MSTG-AUTH-7	Le sessioni vengono invalidate sull'endpoint remoto dopo un periodo predefinito di inattività, così come i token di accesso devono scadere.	✓	✓
4.8	MSTG-AUTH-8	L'autenticazione biometrica, se presente, non deve essere legata agli eventi (cioè utilizzando un'API che restituisce semplicemente "vero" o "falso") ma deve essere basata sullo sblocco del portachiavi/keystore.		✓
4.9	MSTG-AUTH-9	Esiste un'autenticazione a due fattori applicata all'endpoint remoto ed il requisito 2FA è costantemente richiesto.		✓

Continua nella prossima pagina

#	MSTG-ID	Descrizione	L1	L2
4.10	MSTG-AUTH-10	Le transazioni sensibili richiedono un'autenticazione di livello superiore.	✓	
4.11	MSTG-AUTH-11	L'app registra e informa l'utente di tutte le attività sensibili sul proprio account. Gli utenti sono in grado di visualizzare un elenco di dispositivi autorizzati, visualizzare informazioni contestuali (indirizzo IP, posizione, ecc.) e di bloccare dispositivi specifici.	✓	

Esistenza e implementazioni di una politica per la scelta della password (MSTG-AUTH-5) [7] La robustezza della password è un concetto fondamentale in ambito di sicurezza soprattutto quando il meccanismo della password viene utilizzato per autenticare un utente. Una politica definisce dei requisiti che l'utente finale deve rispettare nella scelta della password. Una politica forte e ben studiata rende gli attacchi di cracking complessi o addirittura impossibili da performare per un attaccante. Questa sezione copre vari tipi di best practice che vengono usate per garantire la robustezza di una password basandosi sul lavoro dell'OWASP: OWASP Authentication Cheat Sheet.

Analisi statica Il primo passo è quello di accertarsi dell'esistenza di una politica e verificare che l'implementazione segua i principi descritti nel OWASP Authentication Cheat Sheet con particolare attenzione sulla lunghezza e sui limiti nella scelta dei caratteri. Identificare tutte le funzioni in cui viene utilizzato il meccanismo della password e verificare che la politica sia correttamente applicata e verificata. Verificare la correttezza della funzione incaricata a mettere in atto la politica controllando che le password che violano le regole imposte siano effettivamente rifiutate.

Zxcvbn Zxcvbn è una libreria che può essere usata per stimare la forza delle password, sviluppata da cracker di password. È disponibile in JavaScript ma anche per molti altri linguaggi di programmazione lato server. Una volta installato, Zxcvbn può essere utilizzato per calcolare la complessità e la quantità di tentativi necessari per ricavare la password. Dopo aver aggiunto la libreria JavaScript Zxcvbn alla pagina HTML, è possibile eseguire il comando Zxcvbn nella console del browser, per ottenere informazioni dettagliate sulla probabilità di ricavare la password, incluso un punteggio che dà un'indicazione immediata sulla complessità della password.

```
> zxcvbn('ThisShouldBeVeryHardToCrack!')
< {password: "ThisShouldBeVeryHardToCrack!", guesses: 9.71881e+21, guesses_log10: 21.98761309187359, sequence: Array
(5), calc_time: 14, ...} ⓘ
  calc_time: 14
  ▶ crack_times_display: {online_throttling_100_per_hour: "centuries", online_no_throttling_10_per_second: "centuri...
  ▶ crack_times_seconds: {online_throttling_100_per_hour: 3.4987716e+23, online_no_throttling_10_per_second: 971881...
  ▶ feedback: {warning: "", suggestions: Array(0)}
    guesses:
    9.71881e+21
    guesses_log10: 21.98761309187359
    password: "ThisShouldBeVeryHardToCrack!"
    score: 4
  ▶ sequence: (5) [{} {}, {}, {}, {}]
  ▶ __proto__: Object
```

Figura 2.9: Zxcvbn output

Il punteggio è definito come segue e può essere usato per fornire una misuristica sulla forza della password, per esempio:

```
1 0 # too guessable: risky password. (guesses < 10^3)
2 1 # very guessable: protection from throttled online attacks.
  (guesses < 10^6)
3 2 # somewhat guessable: protection from unthrottled online
  attacks. (guesses < 10^8)
4 3 # safely unguessable: moderate protection from offline slow-
  hash scenario. (guesses < 10^10)
5 4 # very unguessable: strong protection from offline slow-hash
  scenario. (guesses >= 10^10)
```

Si noti che Zxcvbn può essere utilizzato dagli sviluppatori utilizzando l'implementazione Java (o altre) per guidare l'utente nella creazione di una password forte.

Have I Been Pwned: PwnedPasswords L'approccio precedentemente descritto purtroppo non fornisce una soluzione completa alla scelta di una password sicura, questo perché anche se la password scelta sia effettivamente robusta (per esempio uno punteggio 4 su Zxcvbn) essa potrebbe risultare debole ad attacchi a dizionario. La complessità di una password è strettamente collegata al numero di persone che conoscono il segreto e se una password, per quanto robusta possa essere, è stata rubata, essa potrebbe far parte delle tantissime password registrate all'interno di particolari dizionari e quindi risultarebbe comunque insicura. Al fine di ridurre ulteriormente la probabilità di successo di un attacco a dizionario contro uno schema di autenticazione a fattore singolo (ad esempio solo la password), è possibile verificare se una password è stata compromessa in una violazione dei dati. Questo può essere fatto utilizzando servizi basati su Pwned Passwords API di Troy come "Have I been pwned?". Sulla base dello SHA-1 di una possibile password candidata, l'API restituisce il numero di volte in cui quell'hash della password data è stato trovato nelle varie violazioni raccolte dal servizio.

Ricerca per intervallo Al fine di proteggere il valore della password di origine che viene ricercata, Pwned Passwords implementa anche un modello di k-Anonymity che permette di ricercare una password per hash parziale. Questo permette di passare all'API i primi 5 caratteri dell'hash di una password SHA-1 (non sensibile alle maiuscole) come viene mostrato di seguito:

```
1 GET https://api.pwnedpasswords.com/range/{first 5 hash chars}
```

Quando l'hash di una password con gli stessi primi 5 caratteri viene trovato nel repository di Pwned Passwords, l'API risponderà con un HTTP 200 e includerà il suffisso di ogni hash che inizia con il prefisso specificato, seguito da un conteggio di quante volte la password è apparsa nel set di dati. L'utente dell'API può quindi cercare nei risultati della risposta la presenza del loro hash di origine e se non viene trovato, la password non esiste nell'insieme di dati violati a disposizione del servizio. Un esempio di risposta per il prefisso hash "21BD1" sarebbe il seguente:

```

1 0018A45C4D1DEF81644B54AB7F969B88D65 :1
2 00D4F6E8FA6EECAD2A3AA415EEC418D38EC :2
3 011053FD0102E94D6AE2F8B83D76FAF94F6 :1
4 012A7CA357541F0AC487871FEEC1891C49C :2
5 0136E006E24E7D152139815FB0FC6A50B15 :2
6 ...

```

Una ricerca per intervallo restituisce tipicamente circa 500 suffissi di hash, sebbene questo numero differisca a seconda del prefisso di hash che si sta cercando e aumenti man mano che si aggiungono altre password. Ci sono 1.048.576 diversi prefissi hash tra 00000 e FFFF (16⁵) e ognuno di essi restituirà HTTP 200; non c'è nessuna circostanza in cui l'API dovrebbe restituire HTTP 404. I passi da seguire per poter implementare questo meccanismo sono i seguenti:

- Codificare l'input dell'utente in UTF-8 (ad esempio: la password "test");
- Prendere l'hash SHA-1 del risultato del passo 1 (ad esempio: l'hash di "test" è A94A8FE5C...);
- Copiare i primi 5 caratteri (il prefisso dell'hash) e usarli per una ricerca per intervallo usando la seguente API:

```

1 http GET https://api.pwnedpasswords.com/range/A94A8

```

- Iterare attraverso il risultato e cercare il resto dell'hash (es. FE5CC... fa parte della lista restituita?). Se non fa parte della lista restituita, allora la password per l'hash dato non è stata trovata. Altrimenti, come nel caso di FE5CC..., la chiamata restituirà un contatore che mostrerà quante volte quel hash è stato trovato all'interno del database delle violazioni (ad esempio: FE5CC...:76479).

Questo meccanismo può essere utilizzato in combinazione con la libreria Zxcvbn per aumentare la robustezza della politica implementata.

Applicazione demo La demo realizzata per questo requisito è una applicazione Android che implementa le best practice descritte precedentemente. In questa categoria si è ritenuto più opportuno sviluppare un'applicativo che seguisse la guida fornita da OWASP per la creazione di una politica sulla scelta della password completa e corretta piuttosto che implementare una demo vulnerabile e mostrare un attacco bruteforce su Android. Tale idea è alla base del lavoro di tesi che ha come obiettivo quello di fornire un supporto software, oltre a tutta la documentazione già presente, per comprendere al meglio meglio il requisito specifico e fornire un punto di partenza per rendere il proprio progetto in linea alle specifiche descritte dal MASVS.

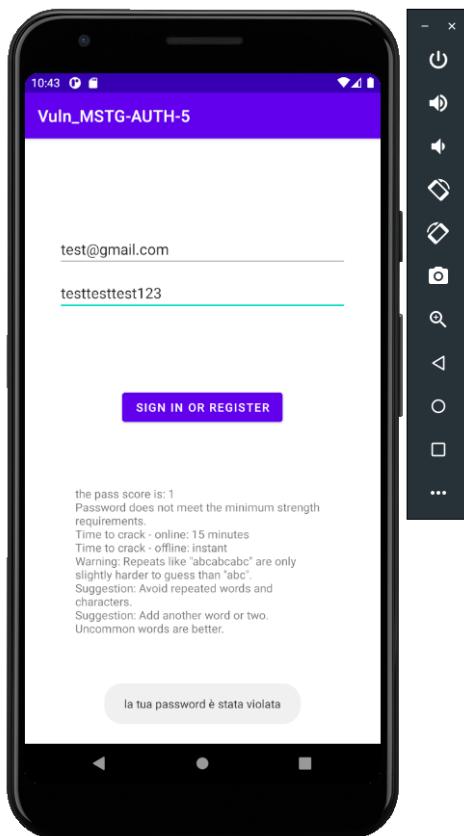


Figura 2.10: Android App vuln-MSTG-AUTH-5

Nell'applicativo sono state implementate le due pratiche descritte per il requisito 5 della categoria sull'autenticazione ed in particolare è stata importata in Android la libreria Zxcvbn per permettere la valutazione della password. La libreria in questione viene utilizzata come monito per l'utente finale cer-

cando di far comprendere la robustezza della password scelta, in questo caso la politica può essere legata al Time to crack che restituisce la libreria o allo score per decidere se accettare o rifiutare una password. In aggiunta è stato implementato il meccanismo di verifica di violazione della password fornito dall'API di "Have I been pwned?". Tale meccanismo è stato implementato seguendo le raccomandazioni fornite dall'OWASP discusse precedentemente, esso permette all'utente di prendere atto che una determinata password, anche se robusta, non dovrebbe essere utilizzata per registrarsi al servizio fornito. Quest'ultimo è anch'esso un meccanismo su cui legare una politica per la scelta della password che, in combinazione con lo score o il Time to crack, risulta essere un'approccio sicuro e abbastanza semplice da implementare per garantire all'utente finale un buon livello di sicurezza in termini di robustezza della password.

2.5 V5: Network Communication Requirements

Obiettivo del controllo Lo scopo dei controlli elencati in questa sezione è di assicurare la riservatezza e l'integrità delle informazioni scambiate tra l'app mobile e gli endpoint. Come minimo, un'app mobile deve utilizzare un canale sicuro e criptato per la comunicazione di rete utilizzando il protocollo TLS con impostazioni appropriate. Il livello 2 aggiunge ulteriori misure di difesa in profondità come il pinning SSL.

#	MSTG-ID	Descrizione	L1	L2
5.1	MSTG-NETWORK-1	Le informazioni trasmette sono cifrate utilizzando TLS. Il canale sicuro è usato consistentemente in tutta l'applicazione	✓	✓
5.2	MSTG-NETWORK-2	Le impostazioni TLS sono in linea con le migliori pratiche attuali, o il più vicino possibile ad esse se il sistema non supporta gli standard raccomandati.	✓	✓

#	MSTG-ID	Descrizione	L1	L2
5.3	MSTG-NETWORK-3	L'applicazione verifica il certificato X.509 dell'endpoint remoto quando viene stabilito il canale sicuro. Solo i certificati firmati da una CA affidabile sono accettati.	✓	✓
5.4	MSTG-NETWORK-4	L'applicazione utilizza il proprio archivio di certificati, oppure fissa il certificato o la chiave pubblica dell'endpoint, e di conseguenza non stabilisce connessioni con endpoint che offrono certificati o chiavi diverse, anche se firmate da una CA affidabile.		✓
5.5	MSTG-NETWORK-5	L'applicazione non deve basarsi su un unico canale di comunicazione insicuro (e-mail o SMS) per le operazioni critiche, come le iscrizioni ed il recupero dell'account.		✓
5.6	MSTG-NETWORK-6	L'applicazione si poggia solo su connessioni e librerie di sicurezza aggiornate.		✓

Verifica della crittografia dei dati sulla rete (MSTG-NETWORK-1 e MSTG-NETWORK-2) [11] Una delle funzioni principali delle app mobile è l'invio/ricezione di dati su reti non affidabili come Internet. Se i dati non sono adeguatamente protetti in transito, un aggressore con accesso a qualsiasi parte dell'infrastruttura di rete (ad esempio, un punto di accesso Wi-Fi) può intercettarli, leggerli o modificarli. Questo è il motivo per cui i protocolli di rete in chiaro sono raramente consigliabili. La stragrande maggioranza delle applicazioni si basa su HTTP per la comunicazione con il backend. HTTPS avvolge HTTP in una connessione cifrata (l'acronimo HTTPS originariamente si riferiva a HTTP su Secure Socket Layer (SSL); SSL è il predecessore depre-

cato di TLS). TLS permette l'autenticazione del servizio di backend e assicura la riservatezza e l'integrità dei dati sulla rete.

Impostazioni TLS raccomandate È importante anche garantire una corretta configurazione TLS lato server. Il protocollo SSL è deprecato e non dovrebbe più essere usato, TLS v1.0 e TLS v1.1 hanno vulnerabilità note e il loro uso è deprecato in tutti i principali browser dal 2020 mentre TLS v1.2 e TLS v1.3 sono considerati la migliore pratica per la trasmissione sicura dei dati. A partire da Android 10 (livello API 29) TLS v1.3 è abilitato di default per una comunicazione più veloce e sicura. Il cambiamento principale apportato al protocollo con la versione 1.3 di TLS è che non è più possibile personalizzare le suite di cifratura e che tutte personalizzazioni sono abilitate quando TLS v1.3 è abilitato, mentre la modalità Zero Round Trip (0-RTT) non è supportata. Quando sia il client che il server sono controllati dalla stessa organizzazione e utilizzati solo per comunicare tra loro, è possibile aumentare a piacimento la sicurezza irrigidendo la configurazione. Se un'applicazione mobile si connette a un server specifico, il suo stack di rete può essere sintonizzato per assicurare il massimo livello di sicurezza possibile per la configurazione del server. La mancanza di supporto nel sistema operativo sottostante può costringere l'applicazione mobile a usare una configurazione più debole e di conseguenza più pericolose.

Terminologia delle suite di cifratura Le suite di cifratura hanno la seguente struttura:

¹ `Protocol_KeyExchangeAlg_WITH_BlockCipher_IntegrityCheckAlg`

Questa struttura può essere descritta come segue:

- Il protocollo utilizzato dal cfrario;
- L'algoritmo di scambio delle chiavi usato dal server e dal client per autenticarsi durante l'handshake TLS;
- Il cfrario a blocchi usato per cifrare il flusso di messaggi;

- Algoritmo di controllo dell'integrità usato per autenticare i messaggi.

Esempio: TLS_RSA_CON_3DES_EDE_CBC_SHA. Nell'esempio qui sopra la suite di cifratura usa:

- TLS come protocollo
- Crittografia asimmetrica RSA per l'autenticazione
- 3DES per la crittografia simmetrica con modalità EDE_CBC
- Algoritmo di hash SHA per l'integrità

Si noti che in TLSv1.3 il KeyExchangeAlgorithm non fa parte della cipher suite, invece è determinato durante l'handshake TLS. Nel seguente elenco sono mostrati i diversi algoritmi disponibili per ogni parte della suite di cifratura.

Protocolli:

- SSLv1
- SSLv2 - RFC 6176
- SSLv3 - RFC 6101
- TLSv1.0 - RFC 2246
- TLSv1.1 - RFC 4346
- TLSv1.2 - RFC 5246
- TLSv1.3 - RFC 8446

Algoritmi di scambio di chiavi:

- DSA - RFC 6979
- ECDSA - RFC 6979
- RSA - RFC 8017
- DHE - RFC 2631 - RFC 7919

- ECDHE - RFC 4492
- PSK - RFC 4279
- DSS - FIPS186-4
- DH_anon - RFC 2631 - RFC 7919
- DHE_RSA - RFC 2631 - RFC 7919
- DHE_DSS - RFC 2631 - RFC 7919
- ECDHE_ECDSA - RFC 8422
- ECDHE_PSK - RFC 8422 - RFC 5489
- ECDHE_RSA - RFC 8422

Cifrari a blocchi:

- DES - RFC 4772
- DES_CBC - RFC 1829
- 3DES - RFC 2420
- 3DES_EDE_CBC - RFC 2420
- AES_128_CBC - RFC 3268
- AES_128_GCM - RFC 5288
- AES_256_CBC - RFC 3268
- AES_256_GCM - RFC 5288
- RC4_40 - RFC 7465
- RC4_128 - RFC 7465
- CHACHA20_POLY1305 - RFC 7905 - RFC 7539

Algoritmi di controllo dell'integrità:

- MD5 - RFC 6151
- SHA - RFC 6234
- SHA256 - RFC 6234
- SHA384 - RFC 6234

Si noti che l'efficienza di una suite di cifratura dipende dall'efficienza dei suoi algoritmi. Per ulteriori informazioni si demanda all'elenco aggiornato delle suite di cifratura raccomandate da usare con TLS. Queste suite di cifratura sono raccomandate sia da IANA nella sua documentazione sui parametri TLS che da OWASP TLS Cipher String Cheat Sheet.

- IANA recommended cipher suites can be found in TLS Cipher Suites.
- OWASP recommended cipher suites can be found in the TLS Cipher String Cheat Sheet.

Si noti che in Android 10 le seguenti suite di cifratura SHA-2 CBC sono state rimosse:

- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384

Alcune versioni di Android e iOS non supportano alcune delle suite di cifratura raccomandate, quindi per scopi di compatibilità si possono controllare le suite di cifratura supportate per le versioni di Android e iOS e scegliere le suite supportate che forniscono il più alto livello di sicurezza.

Analisi statica Identificare tutte le richieste API/servizi web nel codice sorgente e assicurarsi che non vengano usati URL HTTP in chiaro. Assicuratevi che le informazioni sensibili siano inviate su canali sicuri usando HttpsURLConnection o SSLSocket (per la comunicazione a livello di socket usando TLS). Essendo che SSLSocket non verifica l'hostname, usare getDefaultHostnameVerifier per verificare l'hostname. Verificare che il server o il proxy di terminazione su cui termina la connessione HTTPS sia configurato secondo le migliori pratiche. Ulteriori informazioni sono riportate sul foglio di calcolo di OWASP Transport Layer Protection e il Qualys SSL/TLS Deployment Best Practices.

Analisi dinamica Intercettare il traffico di rete in entrata e in uscita dell'app testata e assicurarsi che questo traffico sia cifrato. Si può intercettare il traffico di rete in uno dei seguenti modi:

- Catturare tutto il traffico HTTP(S) e Websocket con un proxy di intercettazione come OWASP ZAP o Burp Suite e assicurarsi che tutte le richieste siano fatte via HTTPS invece che HTTP.
- I proxy di intercettazione come Burp e OWASP ZAP mostreranno solo il traffico HTTP(S). È possibile, tuttavia, utilizzare un plugin Burp come Burp-non-HTTP-Extension o lo strumento mitm-relay per decodificare e visualizzare la comunicazione tramite XMPP e altri protocolli.

Alcune applicazioni potrebbero non funzionare con proxy come Burp e OWASP ZAP a causa del Certificate Pinning. In tale scenario, si prega di abilitare l'opzione "Testare gli archivi di certificati personalizzati e il Certificate Pinning". Per verificare se server supporta le giuste suite di cifratura, ci sono vari strumenti che si possono utilizzare:

- **nscurl** - Si rimanda ad OWASP Test della comunicazione di rete per iOS per maggiori dettagli.
- **testssl.sh** - È uno strumento gratuito a riga di comando che controlla il servizio di un server su qualsiasi porta per il supporto dei cifrari TLS/SSL, dei protocolli e di alcuni difetti crittografici.

Come già discusso nei capitoli precedenti per questo capitolo si è deciso di non implementare un'applicazione demo sia per la necessità di costruire un'infrastruttura di rete correttamente configurata, il che va fuori dagli obiettivi dell'elaborato, sia per la presenza di tool come Burp Suite e OWASP ZAP che semplificano e rendono pratica la comprensione dei requisiti di questo capitolo.

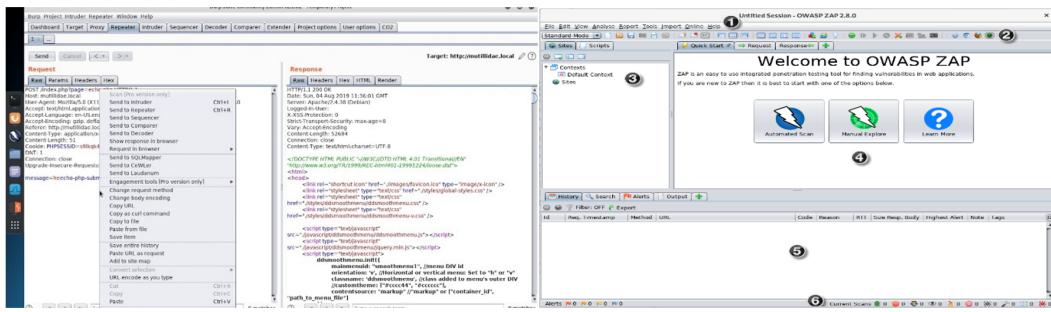


Figura 2.11: Burp e Owasp Zap

2.6 V6: Platform Interaction Requirements

Obiettivo del controllo I controlli in questo gruppo assicurano che l'app utilizzi le API della piattaforma ed i componenti standard in modo sicuro. Inoltre, i controlli coprono anche la comunicazione tra le app (IPC).

#	MSTG-ID	Descrizione	L1	L2
6.1	MSTG-PLATFORM-1	L'applicazione deve richiedere solo l'insieme minimo di permessi necessari al corretto funzionamento.	✓	✓
6.2	MSTG-PLATFORM-2	Tutti gli input da fonti esterne e dall'utente sono validati e, se necessario, sanitizzati. Questo include i dati ricevuti tramite l'UI, meccanismi IPC come gli intent, URL personalizzati e risorse di rete.	✓	✓

Continua nella prossima pagina

#	MSTG-ID	Descrizione	L1	L2
6.3	MSTG-PLATFORM-3	L'applicazione non esporta funzionalità sensibili attraverso schemi URL personalizzati, a meno che questi meccanismi non siano adeguatamente protetti.	✓	✓
6.4	MSTG-PLATFORM-4	L'applicazione non esporta funzionalità sensibili attraverso strutture IPC, a meno che questi meccanismi non siano adeguatamente protetti.	✓	✓
6.5	MSTG-PLATFORM-5	JavaScript è disabilitato nelle WebViews a meno che non sia esplicitamente richiesto.	✓	✓
6.6	MSTG-PLATFORM-6	Le WebViews sono configurate per permettere solo l'insieme minimo di protocol handlers richiesti (idealmente, può bastare solo https). handlers potenzialmente pericolosi, come file, tel e app-id, sono disabilitati.	✓	✓
6.7	MSTG-PLATFORM-7	Se i metodi nativi dell'app sono esposti a una WebView, verificare che la WebView esegua solo il codice JavaScript contenuto nel pacchetto dell'app.	✓	✓
6.8	MSTG-PLATFORM-8	La deserializzazione degli oggetti, se presente, è implementata utilizzando API di serializzazione sicure.	✓	✓
6.9	MSTG-PLATFORM-9	L'app si protegge dagli attacchi di screen overlay. (solo per Android)		✓

Continua nella prossima pagina

#	MSTG-ID	Descrizione	L1	L2
6.10	MSTG-PLATFORM-10	La cache, la memoria e le risorse caricate di una WebView (JavaScript, ecc.) dovrebbero essere cancellate prima che la WebView sia distrutta.	✓	
6.11	MSTG-PLATFORM-11	Verificare che l'applicazione impedisca l'uso di tastiere personalizzate ogni volta che vengono inseriti dati sensibili (solo per iOS).	✓	

Testing delle vulnerabilità di injection (MSTG-PLATFORM-2) [8]

Le applicazioni Android possono esporre funzionalità attraverso schemi di URL personalizzati (che fanno parte degli Intent). Possono esporre funzionalità ad:

- altre applicazioni (tramite meccanismi IPC, come Intents, Binders, Android Shared Memory (ASHMEM), o BroadcastReceivers);
- l'utente (tramite l'interfaccia utente).

Nessuno degli input da queste fonti può essere affidabile quindi tutto deve essere convalidato e/o sanitizzato. La validazione assicura che il comportamento dell'applicazione possa essere solo quello desiderato, qualsiasi input può essere inviato all'app, il che può permettere a un aggressore o a un'app dannosa di sfruttare le funzionalità dell'app. Le seguenti porzioni di codice sorgente dovrebbero essere controllate se qualsiasi funzionalità dell'app è stata esposta:

- Schemi URL personalizzati. Controllate anche il test case "Testing Custom URL Schemes" per ulteriori scenari di test;
- Meccanismi IPC (Intent, Binders, Android Shared Memory, o BroadcastReceivers). Controlla anche il caso di test "Testare se i dati sensibili sono esposti tramite meccanismi IPC" per ulteriori scenari di test;

- Interfaccia utente.

Un esempio di un meccanismo IPC vulnerabile è mostrato qui sotto. Si possono usare i **ContentProvider** per accedere alle informazioni del database, e si possono sondare i servizi per vedere se restituiscono dati. Se i dati non sono validati correttamente, il fornitore di contenuti può essere soggetto a SQL injection mentre altre applicazioni stanno interagendo con esso. Vedere la seguente implementazione vulnerabile di un **ContentProvider**.

```
1 <provider
2     android:name="
3         OMTG_CODING_003_SQL_Injection_Content_Provider_Impl"
4     android:authorities="sg.vp.owasp_mobile.provider.College">
5 </provider>
```

L'AndroidManifest.xml di cui sopra definisce un fornitore di contenuti che è esportato e quindi disponibile per tutte le altre applicazioni. La funzione di query nella classe **OMTG_CODING_003_SQL_Injection_Content_Provider_Impl** dovrebbe essere controllata.

```
1 @Override
2 public Cursor query(Uri uri, String[] projection, String
3 selection, String[] selectionArgs, String sortOrder) {
4     SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
5     qb.setTables(STUDENTS_TABLE_NAME);
6     switch (uriMatcher.match(uri)) {
7         case STUDENTS:
8             qb.setProjectionMap(STUDENTS_PROJECTION_MAP);
9             break;
10        case STUDENT_ID:
11            // SQL Injection when providing an ID
12            qb.appendWhere( _ID + " = " + uri.getPathSegments().get(1));
13            Log.e("appendWhere",uri.getPathSegments().get(1).
14                toString());
15            break;
16        default:
```

```

15         throw new IllegalArgumentException("Unknown URI " +
16             uri);
17     }
18
19     if (sortOrder == null || sortOrder == "") {
20
21         /**
22          * By default sort on student names
23          */
24
25         sortOrder = NAME;
26     }
27
28     Cursor c = qb.query(db, projection, selection,
29     selectionArgs, null, null, sortOrder);
30
31     /**
32      * register to watch a content URI for changes
33      */
34
35     c.setNotificationUri(getContext().getContentResolver(), uri
36 );
37
38     return c;
39 }
```

Mentre l'utente fornisce uno STUDENT_ID a:

```
1 content://sg.vp.owasp\_mobile.provider.College/students
```

l'istruzione della query è soggetta a SQL injection. Ovviamente i prepared statement devono essere usati per evitare l'SQL injection, ma anche la validazione dell'input dovrebbe essere applicata in modo che solo l'input che l'app si aspetta venga processato. Tutte le funzioni dell'app che elaborano i dati che arrivano attraverso l'interfaccia utente dovrebbero implementare la convalida dell'input:

- Per l'input dell'interfaccia utente, può essere utilizzato Android Saripaar v2;
- Per l'input da schemi IPC o URL, dovrebbe essere creata una funzione di validazione.

Applicazione demo Le vulnerabilità relative all'injection (soprattutto SQL Injection) sono probabilmente le più note al di fuori dello specifico ambito di

sicurezza informatica. In questo capitolo non sono state trattate l'analisi statica e dinamica in quanto l'OWASP demanda l'approfondimento sul tema ad un altro elaborato: OWASP Input Validation Cheat Sheet. Più nello specifico questo requisito mette in allerta gli sviluppatori sulle varie interfacce che un attaccante può utilizzare per sottomettere la propria query, e se non vengono messi in sicurezza tutti i meccanismi e le funzionalità che esportano tali servizi allora in quel caso l'attaccante trova la strada spianata per poter sfruttare la vulnerabilità. Come precedentemente descritto un'app può esportare dei servizi che vanno oltre l'interfaccia grafica e le caselle di testo, e tali meccanismi sono un punto di aggancio per gli attaccanti. È estremamente importante ai fini della sicurezza ed integrità di un'applicazione il mettere in sicurezza tutti i meccanismi che accettano input esterni anche se tali input non sono direttamente ricollegabili all'utente (in questo caso sanitizzare l'input e adottare meccanismi sicuri per mitigare le fal当地 di injection come i prepared statement).

Per la demo in questo caso è stato utilizzato un'applicazione vulnerabile che OWASP mette a disposizione per performare attacchi e nello specifico è stato utilizzata l'activity sviluppata volutamente con vulnerabilità di SQL injection. L'aspetto interessante dell'attacco in questo caso non è dovuto all'activity che viene mostrata sotto dove semplicemente tramite una particolare stringa si riesce ad entrare nell'account senza possedere informazioni né sull'username e né sulla password dell'utente; ma l'aspetto interessante è quello sfruttato grazie al meccanismo dei content che esporta l'app stessa, è possibile infatti sottomettere una query da linea di comando che sfrutta il meccanismo di content e interagisce anch'esso con un database diverso da quello sfruttato direttamente dalla GUI dell'applicazione. Anche in quel caso possiamo notare come, costruendo una specifica query malevola, si riescono ad ottenere tutte le informazioni relative al database.

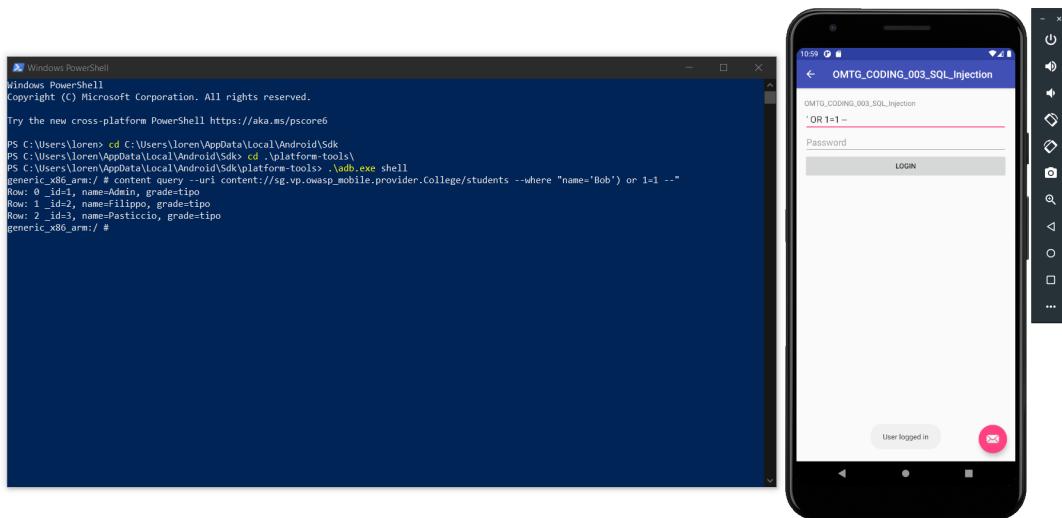


Figura 2.12: Test demo OMTG-CODING-003-SQL

2.7 V7: Code Quality And Build Setting Requirements

Obiettivo del controllo L’obiettivo di questo controllo è quello di garantire che le pratiche basilari di programmazione di codice sicuro siano seguite nello sviluppo dell’app, e che le opzioni di sicurezza offerte dal compilatore siano attivate.

#	MSTG-ID	Descrizione	L1	L2
7.1	MSTG-CODE-1	L’applicazione è firmata e dotata di un certificato valido, di cui la chiave privata è adeguatamente protetta.	✓	✓
7.2	MSTG-CODE-2	L’app è stata costruita in modalità di rilascio, con impostazioni appropriate per una build di rilascio (ad esempio non debuggabile).	✓	✓
7.3	MSTG-CODE-3	I simboli di debug sono stati rimossi dai binari nativi.	✓	✓

Continua nella prossima pagina

#	MSTG-ID	Descrizione	L1	L2
7.4	MSTG-CODE-4	Il codice di debug e il codice di assistenza allo sviluppatore (ad esempio codice di test, backdoor, impostazioni nascoste) sono stati rimossi. L'applicazione non registra errori verbosi o messaggi di debug.	✓	✓
7.5	MSTG-CODE-5	Tutti i componenti di terze parti utilizzati dall'app mobile, come librerie e framework, sono identificati e controllati per vulnerabilità note.	✓	✓
7.6	MSTG-CODE-6	L'app cattura e gestisce eventuali eccezioni.	✓	✓
7.7	MSTG-CODE-7	La logica di gestione degli errori nei controlli di sicurezza nega l'accesso di default.	✓	✓
7.8	MSTG-CODE-8	Nel codice non gestito, la memoria è allocata, liberata e utilizzata in modo sicuro.	✓	✓
7.9	MSTG-CODE-9	Caratteristiche di sicurezza gratuite offerte dalla toolchain, come il byte-code minification, la protezione dello stack, il supporto PIE e conteggio automatico dei riferimenti, sono attivate.	✓	✓

Testing della modalità debug (MSTG-CODE-2) [10] Questo requisito è relativamente semplice da testare ma allo stesso tempo molto pericoloso se viene trascurato. L'attributo `android:debuggable` nell'elemento `Application` che è definito nel manifesto Android determina se l'app può essere sottoposta a debug o meno.

Analisi statica Controllare il file `AndroidManifest.xml` per determinare se l'attributo `android:debuggable` è stato impostato e per trovare il valore del-

l'attributo:

```
1 ...
2 <application android:allowBackup="true" android:debuggable="true"
               android:icon="@drawable/ic_launcher" android:label="@string/app_name"
               android:theme="@style/AppTheme">
3 ...
```

Per una build di rilascio, questo attributo dovrebbe essere sempre impostato a `false` (il valore predefinito).

Analisi dinamica Drozer può essere usato per determinare se un'applicazione è debuggabile o meno. Il modulo Drozer `app.package.attacksurface` mostra anche informazioni sui componenti IPC esportati dall'applicazione.

```
1 dz> run app.package.attacksurface com.mwr.dz
2 Attack Surface:
3   1 activities exported
4   1 broadcast receivers exported
5   0 content providers exported
6   0 services exported
7   is debuggable
```

Se un'applicazione è debuggabile, eseguire i comandi dell'applicazione è banale. Nella shell adb, basta eseguire il comando `run-as` aggiungendo il nome del pacchetto e il comando dell'applicazione al nome binario:

```
1 $ run-as com.vulnerable.app id
2 uid=10084(u0_a84) gid=10084(u0_a84) groups=10083(u0_a83),1004(
      input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r)
      ,3001(net_bt_admin),3002(net_bt),3003/inet),3006(
      net_bw_stats) context=u:r:untrusted_app:s0:c512,c768
```

Android Studio può anche essere usato per eseguire il debug e verificare l'attivazione del debug di un'applicazione. Un altro metodo per determinare se un'applicazione è debuggabile è collegare jdb al processo in esecuzione. Se questo ha successo, il debug sarà attivato. La seguente procedura può essere utilizzata per avviare una sessione di debug con jdb:

1. Usando adb e jdwp, identificate il PID dell'applicazione attiva di cui volete fare il debug:

```
1 $ adb jdwp  
2 2355  
3 16346 <== last launched, corresponds to our application
```

2. Crea un canale di comunicazione utilizzando adb tra il processo dell'applicazione (con il PID) e il computer host utilizzando una specifica porta locale:

```
1 # adb forward tcp:[LOCAL_PORT] jdwp:[APPLICATION_PID]  
2 $ adb forward tcp:55555 jdwp:16346
```

3. Usando jdb, collegare il debugger alla porta del canale di comunicazione locale e avviare una sessione di debug:

```
1 $ jdb -connect com.sun.jdi.SocketAttach:hostname=localhost  
     ,port=55555  
2 Set uncaught java.lang.Throwable  
3 Set deferred uncaught java.lang.Throwable  
4 Initializing jdb ...
```

Alcune note sul debug:

- Lo strumento JADX può essere usato per identificare posizioni interessanti per l'inserimento di breakpoint.
- L'uso dei comandi di base per jdb può essere trovato su Tutorialspoint.
- Se ottenete un errore che dice che "la connessione al debugger è stata chiusa" mentre jdb è legato alla porta del canale di comunicazione locale, uccidete tutte le sessioni adb e iniziate una singola nuova sessione.

Applicazione demo Per questa categoria è stata realizzata una un'applicazione Android volutamente vulnerabile ad attacchi di tipo debug attached. In questa versione sono state volutamente inserite due vulnerabilità che hanno permesso l'attacco grazie all'utilizzo del tool Drozer.

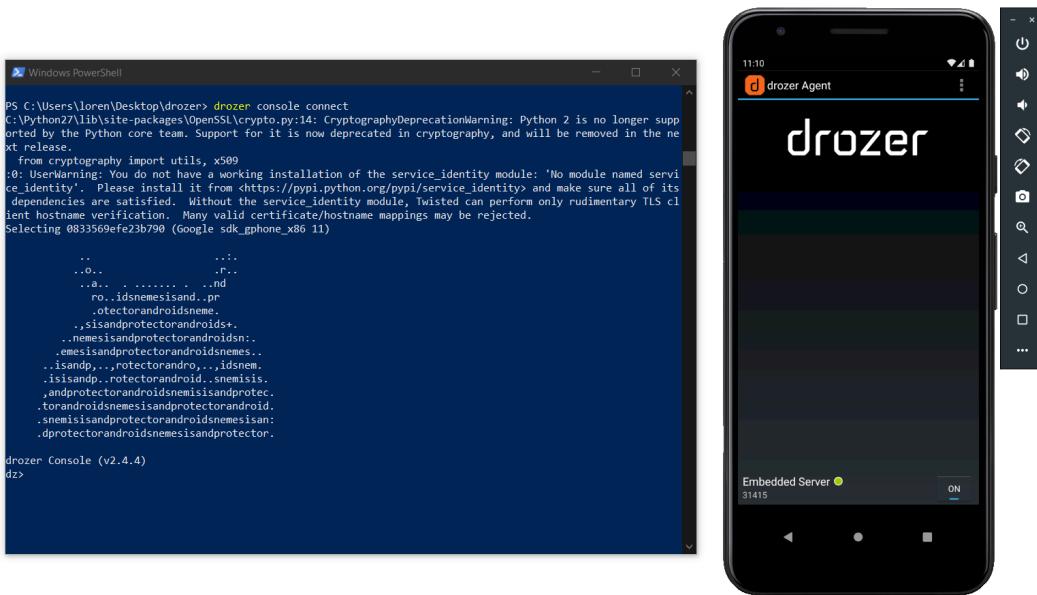


Figura 2.13: Drozer

L'applicazione simula un servizio di autenticazione che permette l'accesso solo agli utenti che possiedono le credenziali corrette. In questo caso viene bypassata l'autenticazione senza l'utilizzo di tecnica di injection come visto nella precedente categoria, ma semplicemente sfruttando la modalità debugging e Drozer. In particolare, sfruttando il tool Drozer si possono visualizzare le informazioni relative alle activity presenti nell'applicazione ed è anche possibile lanciare un'activity a piacimento. In questo caso l'attacco consiste proprio nell'andare a visualizzare quali activity siano presenti all'interno dell'applicazione e tramite il comando apposito è possibile istanziare direttamente l'activity che dovrebbe essere accessibile solo tramite l'autenticazione dell'utente. Grazie a questa tecnica di bypassing, infatti, verrà direttamente istanziata l'activity `DisplaySecret` (simulando un'applicazione che visualizza informazioni sul conto corrente bancario) senza neanche dover trovare il modo di ottenere o attaccare l'account utente. Questa vulnerabilità è ottenuta da due errori molto gravi che il MASVS ha ritenuto opportuno classificare come requisiti ovvero:

- Lasciare attiva la modalità debugging all'interno del manifesto:

```
1 Android:android:debuggable="true"
```

- Lasciare pubblica l'activity segreta accessibile solo tramite l'autenticazione:

```
1 <activity android:name=".DisplaySecret" android:exported="true"/>
```

Come dimostrato dall'attacco performato, queste due piccole sviste di programmazione risultano essere critiche soprattutto quando un'applicazione gestisce dati sensibili.

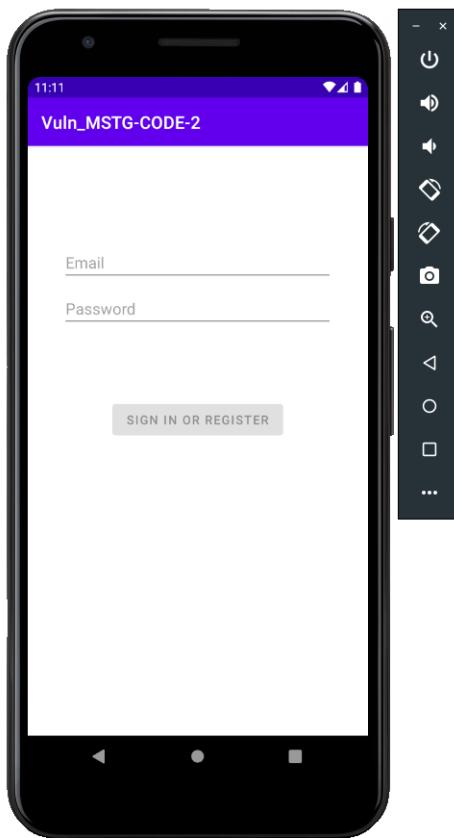


Figura 2.14: Android App vuln-MSTG-CODE-2

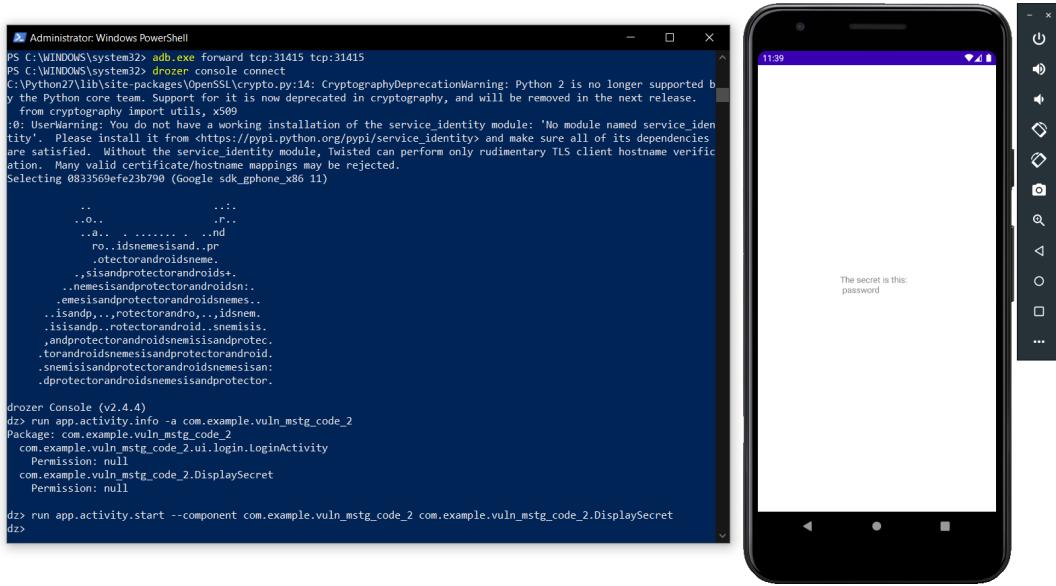


Figura 2.15: Test demo vuln-MSTG-AUTH-5

2.8 V8: Resilience Requirements

Obiettivo del controllo Questa sezione copre le misure di difesa in profondità raccomandate per le app che elaborano o danno accesso a dati o funzionalità sensibili. La mancanza di uno qualsiasi di questi controlli non causa una vulnerabilità ma hanno lo scopo di aumentare la resilienza dell'app contro il reverse engineering e specifici attacchi lato client. I controlli in questa sezione dovrebbero essere applicati secondo le necessità, sulla base di una valutazione dei rischi causati dalla manomissione non autorizzata dell'app e/o dal reverse engineering del codice. Si suggerisce di consultare il documento OWASP "Technical Risks of Reverse Engineering and Unauthorized Code Modification Reverse Engineering and Code Modification Prevention" per un elenco dei rischi aziendali e di minacce tecniche associate. Affinché uno qualsiasi dei controlli nella lista qui sotto sia efficace, l'applicazione deve soddisfare almeno i requisiti del MASVS-L1 (cioè, devono essere verificati i controlli di sicurezza di base), così come tutti i requisiti di numero inferiore in V8. Per esempio, i controlli di offuscamento elencati sotto "impedire la comprensione" devono essere combinati con "impedire l'analisi dinamica e la manomissione" e "bin-

ding dei dispositivi". Si noti che le protezioni del software non devono mai essere usate come sostituzione dei controlli di sicurezza. I controlli elencati nel MASVR-R hanno lo scopo di aggiungere controlli di protezione specifici per le minacce alle applicazioni che soddisfano anche i requisiti di sicurezza MASVS. Si applicano le seguenti considerazioni:

1. Deve essere definito un modello di minaccia che delinei chiaramente le minacce lato client da cui difendersi. Inoltre, deve essere specificato il grado di protezione che lo schema intende fornire. Per esempio, un obiettivo dichiarato potrebbe essere quello di costringere gli autori di malware, che cercano di strumentare l'app, ad investire un significativo sforzo manuale di reverse engineering.
2. Il modello di minaccia deve essere credibile e rilevante. Per esempio, nascondere una chiave crittografica in un'implementazione whitebox potrebbe rivelarsi poco rilevante se un attaccante può semplicemente ottenere il codice della white-box.
3. L'efficacia delle protezioni dovrebbe sempre essere verificata da un esperto umano con esperienza nel testare i particolari tipi di antimanomissione e offuscamento utilizzati (vedere anche i capitoli "reverse engineering" e "valutazione delle protezioni software" nella Mobile Security Testing Guide).

Impedire l'analisi dinamica e la manomissione

#	MSTG-ID	Descrizione	R
8.1	MSTG-RESILIENCE-1	L'app rileva e risponde alla presenza di un dispositivo rooted o jailbroken avvisando l'utente o terminando l'app.	✓
8.2	MSTG-RESILIENCE-2	L'app impedisce il debug e/o rileva, e risponde ad un debugger collegato. Tutti i protocolli di debug disponibili devono essere coperti.	✓

#	MSTG-ID	Descrizione	R
8.3	MSTG-RESILIENCE-3	L'app rileva e risponde alla manomissione di file eseguibili e dei dati critici all'interno della propria sandbox.	✓
8.4	MSTG-RESILIENCE-4	L'applicazione rileva e risponde alla presenza di strumenti e framework di reverse engineering utilizzati sul dispositivo.	✓
8.5	MSTG-RESILIENCE-5	L'app rileva e risponde all'esecuzione in un emulatore.	✓
8.6	MSTG-RESILIENCE-6	L'applicazione rileva e risponde alla manomissione del codice e dei dati nel proprio spazio di memoria.	✓
8.7	MSTG-RESILIENCE-7	L'applicazione implementa più meccanismi in ogni categoria di difesa (da 8.1 a 8.6). Si noti che la resilienza scala con la quantità, la diversità e l'originalità dei meccanismi utilizzati.	✓
8.8	MSTG-RESILIENCE-8	I meccanismi di rilevamento innescano risposte di diversi tipi, tra cui risposte ritardate e silenziose.	✓
8.9	MSTG-RESILIENCE-9	L'offuscamento è applicato alle difese grammatiche, che a loro volta impediscono il de-offuscamento tramite l'analisi dinamica.	✓

Binding del dispositivo

#	MSTG-ID	Descrizione	R
8.10	MSTG-RESILIENCE-10	L'applicazione implementa una funzionalità di "binding del dispositivo" utilizzando un'immagine digitale del dispositivo derivata da più proprietà uniche per il dispositivo.	✓

Impedire la comprensione

#	MSTG-ID	Descrizione	R
8.11	MSTG-RESILIENCE-11	Tutti i file eseguibili e le librerie appartenenti all'app sono crittografate a livello di file e/o di codice ed i dati all'interno degli eseguibili sono crittografati o impacchettati. Una banale analisi statica non rivela codice o dati.	✓
8.12	MSTG-RESILIENCE-12	Se l'obiettivo dell'offuscamento è quello di proteggere i codici sensibili, deve essere utilizzato uno schema di offuscamento che sia appropriato per il particolare task e robusto contro i metodi di de-offuscamento manuali e automatici, considerando le ricerche attualmente pubblicate. L'efficacia dello schema di offuscamento deve essere verificata attraverso test manuali. Si noti che le caratteristiche di isolamento basate sull'hardware sono preferite all'offuscamento, quando possibile.	✓

Impedire le intercettazioni

#	MSTG-ID	Descrizione	R
8.13	MSTG-RESILIENCE-13	Come difesa in profondità, oltre ad avere un canale di comunicazione solido e sicuro, può essere anche applicata la crittografia del payload a livello applicazione per impedire ulteriormente l'intercettazione.	✓

Testare il rilevamento anti-debugging (MSTG-RESILIENCE-2) [6]

Il debugging è un modo molto efficace per analizzare il comportamento delle app in fase di esecuzione. Permette al reverse engineer di passare attraverso il codice, fermare l'esecuzione dell'app in punti arbitrari, ispezionare lo stato delle variabili, leggere e modificare la memoria, e molto altro. Le funzioni di anti-debugging possono essere preventive o reattive. Come il nome implica, l'anti-debugging preventivo impedisce al debugger di attaccarsi in primo luogo; l'anti-debugging reattivo comporta il rilevamento dei debugger e la reazione ad essi in qualche modo (ad esempio, terminando l'app o innescando un comportamento nascosto). Si applica la regola del "più è meglio è": per massimizzare l'efficacia, gli specialisti combinano più metodi di prevenzione e rilevamento che operano su diversi livelli di API e sono ben distribuiti in tutta l'app. Come menzionato nel capitolo "Reverse Engineering and Tampering", si ha a che fare con due protocolli di debug su Android: possiamo eseguire il debug a livello Java con JDWP o sul livello nativo tramite un debugger basato su ptrace. Un buon schema di anti-debugging dovrebbe difendersi da entrambi i tipi di debugging. Nel capitolo "Reverse Engineering e manomissione", si è parlato di JDWP, il protocollo usato per la comunicazione tra il debugger e la Java Virtual Machine. Si è mostrato che è facile abilitare il debug per qualsiasi app modificando il suo file manifest e cambiando la proprietà di sistema ro.debuggable che abilita il debug per tutte le app. Vediamo alcune cose che gli sviluppatori mettono in atto per rilevare e disabilitare i debugger di JDWP.

Controllare il flag debuggable in ApplicationInfo Abbiamo già incontrato l'attributo `android:debuggable`. Questo flag nel Manifest di Android determina se il thread JDWP è avviato per l'applicazione. Il suo valore può essere determinato programmaticamente, attraverso l'oggetto `ApplicationInfo` dell'applicazione. Se il flag è impostato, il manifest è stato manomesso e permette il debugging.

```
1 public static boolean isDebuggable(Context context){  
2     return ((context.getApplicationContext().getApplicationInfo()  
3             .flags & ApplicationInfo.FLAG_DEBUGGABLE) != 0);}
```

isDebuggerConnected Anche se questa tecnica potrebbe essere banale da aggirare per un reverse engineer, è possibile utilizzare la funzione `isDebuggerConnected` dalla classe `android.os.Debug` per determinare se un debugger è collegato.

```
1 public static boolean detectDebugger() {  
2     return Debug.isDebuggerConnected();  
3 }
```

La stessa API può essere chiamata tramite codice nativo accedendo alla struttura globale `DvmGlobals`.

```
1 JNIEXPORT jboolean JNICALL  
    Java_com_test_debugging_DebuggerConnectedJNI(JNIenv * env,  
    jobject obj) {  
2     if (gDvm.debuggerConnected || gDvm.debuggerActive)  
3         return JNI_TRUE;  
4     return JNI_FALSE;  
5 }
```

Controlli del timer `Debug.threadCpuTimeNanos` indica la quantità di tempo che il thread corrente ha impiegato per eseguire il codice. Poiché il debug rallenta l'esecuzione del processo, si può usare la differenza nel tempo di esecuzione per indovinare se un debugger è collegato.

```
1 static boolean detect_threadCpuTimeNanos(){
```

```
2   long start = Debug.threadCpuTimeNanos();
3   for(int i=0; i<1000000; ++i)
4     continue;
5   long stop = Debug.threadCpuTimeNanos();
6   if(stop - start < 10000000) {
7     return false;
8   }
9   else {
10    return true;
11  }
12 }
```

Anti-Debugging tradizionale Su Linux, la system call ptrace è usata per osservare e controllare l'esecuzione di un processo (il tracee) e per esaminare e modificare la memoria e i registri di quel processo. Ptrace è il modo principale per implementare il tracciamento delle chiamate di sistema e il debug del codice nativo. La maggior parte delle tecniche anti-debugging di JDWP (che possono essere sicuri per i controlli basati sul timer) non cattureranno i debugger classici basati su ptrace e quindi, molte tecniche anti-debugging di Android includono ptrace, spesso sfruttando il fatto che solo un debugger alla volta può attaccarsi a un processo.

Controllo di TracerPid Quando si esegue il debug di un'applicazione e si imposta un breakpoint sul codice nativo, Android Studio copierà i file necessari sul dispositivo di destinazione e avvierà il lldb-server che utilizzerà ptrace per collegarsi al processo. Da questo momento in poi, se si ispeziona il file di stato del processo sottoposto a debug (/proc/<pid>/status o /proc/self/status), si vedrà che il campo "TracerPid" ha un valore diverso da 0, che è un segno di debug. Questa accortezza si applica solo al codice nativo, se si sta debuggando un'applicazione Java/Kotlin-only il valore del campo "TracerPid" dovrebbe rimanere 0. Questa tecnica è solitamente applicata all'interno delle librerie native JNI in C, come mostrato nell'implementazione di Google's gperftools (Google Performance Tools) Heap Checker del metodo IsDebugge-

rAttached. Tuttavia, se si preferisce includere questo controllo come parte del codice Java/Kotlin si può far riferimento a questa implementazione Java del metodo hasTracerPid dal progetto Anti-Emulator di Tim Strazzere. Quando si cerca di implementare tale metodo da soli, si può controllare manualmente il valore di TracerPid con adb. Il seguente elenco utilizza l'app di esempio NDK di Google hello-jni (com.example.hellojni) per eseguire il controllo dopo aver collegato il debugger di Android Studio:

```

1 $ adb shell ps -A | grep com.example.hellojni
2 u0_a271 11657 573 4302108 50600 ptrace_stop 0 t com.example.
   hellojni

1 $ adb shell cat /proc/11657/status | grep -e "^TracerPid:" |
2   sed "s/^TracerPid:\t//"
2 TracerPid: 11839

1 $ adb shell ps -A | grep 11839
2 u0_a271 11839 11837 14024 4548 poll_schedule_timeout 0 S lldb-
   server

```

Si può vedere come il file di stato di com.example.hellojni (PID=11657) contiene un TracerPID di 11839, che possiamo identificare come il processo lldb-server.

Usare Fork e ptrace Si può impedire il debug di un processo facendo la fork del processo figlio e attaccandolo al genitore come debugger tramite un codice simile al seguente codice di esempio:

```

1 void fork_and_attach()
2 {
3     int pid = fork();
4     if (pid == 0)
5     {
6         int ppid = getppid();
7         if (ptrace(PTRACE_ATTACH, ppid, NULL, NULL) == 0)
8         {
9             waitpid(ppid, NULL, 0);
10            /* Continue the parent process */

```

```

11         ptrace(PTRACE_CONT, NULL, NULL);
12     }
13 }
14 }
```

Con il processo figlio attaccato, ulteriori tentativi di attaccarsi al genitore falliranno. Possiamo verificarlo compilando il codice in una funzione JNI e impacchettandolo in un'app che eseguiamo sul dispositivo.

```

1 root@android:/ $ ps | grep -i anti
2 u0_a151    18190  201    1535844 54908 ffffffff b6e0f124 S sg.
   vantagepoint.antidebug
3 u0_a151    18224  18190 1495180 35824 c019a3ac b6e0ee5c S sg.
   vantagepoint.antidebug
```

Il tentativo di collegarsi al processo padre con gdbserver fallisce con un errore:

```

1 root@android:/ $ ./gdbserver --attach localhost:12345 18190
2 warning: process 18190 is already traced by process 18224
3 Cannot attach to lwp 18190: Operation not permitted (1)
4 Exiting
```

Si può facilmente aggirare questo fallimento terminando il processo figlio e "liberando" il genitore dall'essere tracciato. Di solito si possono trovare schemi più elaborati, che coinvolgono più processi e thread, nonché una qualche forma di monitoraggio per impedire la manomissione. I metodi comuni includono

- Fork di più processi che si tracciano a vicenda;
- Tenere traccia dei processi in esecuzione per assicurarsi che i processi figli rimangano in esecuzione;
- Monitorare i valori nel filesystem /proc, come TracerPID in /proc/pid/-status.

Vediamo un semplice miglioramento per il metodo visto precedentemente. Dopo il fork iniziale, lanciamo nel genitore un thread extra che controlla continuamente lo stato del figlio. A seconda che l'app sia stata costruita in modalità debug o release (che è indicata dal flag android:debuggable nel manifest), il processo figlio dovrebbe fare una delle seguenti cose:

- In modalità di rilascio: La chiamata a ptrace fallisce e il processo figlio va in crash immediatamente con un errore di segmentazione (codice di uscita 11).
- In modalità debug: La chiamata a ptrace funziona e il processo figlio dovrebbe funzionare normalmente. Di conseguenza, una chiamata a waitpid(child_pid) non dovrebbe mai eseguire la return. Se lo fa, c'è qualcosa di strano e si dovrebbe terminare l'intero gruppo di processi.

Quello che segue è il codice completo per implementare questo miglioramento con una funzione JNI:

```
1 #include <jni.h>
2 #include <unistd.h>
3 #include <sys/ptrace.h>
4 #include <sys/wait.h>
5 #include <pthread.h>
6
7 static int child_pid;
8
9 void *monitor_pid() {
10     int status;
11     waitpid(child_pid, &status, 0);
12     /* Child status should never change. */
13     _exit(0); // Commit seppuku
14
15 }
16
17 void anti_debug() {
18     child_pid = fork();
19     if (child_pid == 0)
20     {
21         int ppid = getppid();
22         int status;
23         if (ptrace(PTRACE_ATTACH, ppid, NULL, NULL) == 0)
24         {
25             waitpid(ppid, &status, 0);
26             ptrace(PTRACE_CONT, ppid, NULL, NULL);
27         }
28     }
29 }
```

```

27         while (waitpid(ppid, &status, 0)) {
28             if (WIFSTOPPED(status)) {
29                 ptrace(PTRACE_CONT, ppid, NULL, NULL);
30             } else {
31                 // Process has exited
32                 _exit(0);
33             }
34         }
35     }
36 } else {
37     pthread_t t;
38     /* Start the monitoring thread */
39     pthread_create(&t, NULL, monitor_pid, (void *)NULL);
40 }
41 }
42
43 JNIEXPORT void JNICALL
44 Java_sg_vantagepoint_antidebug_MainActivity_antidebug(JNIEnv *
45     env, jobject instance) {
46     anti_debug();
47 }
```

Di nuovo, inseriamo questo codice in un'applicazione Android per vedere se funziona. Proprio come prima, due processi appaiono quando eseguiamo la build di debug dell'app.

```

1 root@android:/ $ ps | grep -I anti-debug
2 u0_a152    20267  201    1552508 56796 ffffffff b6e0f124 S sg.
   vantagepoint.anti-debug
3 u0_a152    20301  20267 1495192 33980 c019a3ac b6e0ee5c S sg.
   vantagepoint.anti-debug
```

Tuttavia, se terminiamo il processo figlio a questo punto, anche il genitore termina:

```

1 root@android:/ $ kill -9 20301
2 130|root@hammerhead:/ # cd /data/local/tmp
3 root@android:/ $ ./gdbserver --attach localhost:12345 20267
4 gdbserver: unable to open /proc file '/proc/20267/status'
```

```
5 Cannot attach to lwp 20267: No such file or directory (2)
6 Exiting
```

Per aggirarla, dobbiamo modificare leggermente il comportamento dell'applicazione (i modi più semplici per farlo sono patchare la chiamata a `_exit` con NOPs e agganciare la funzione `_exit` in `libc.so`). A questo punto, siamo entrati nella proverbiale "corsa agli armamenti": L'implementazione di forme più intricate di questa difesa e il suo aggiramento sono sempre possibili.

Bypassare il rilevamento del debugger Non c'è un modo generico per bypassare l'anti-debugging. Il metodo migliore dipende dal particolare meccanismo usato per prevenire o rilevare il debugging e dalle altre difese nello schema generale di protezione. Per esempio, se non ci sono controlli di integrità o sono già stati disattivati, applicare una patch all'app potrebbe essere il metodo più semplice. In altri casi, un framework di aggancio o moduli del kernel potrebbero essere preferibili. I seguenti metodi descrivono diversi approcci per bypassare il rilevamento del debugger:

- Patching della funzionalità anti-debugging: Disabilitare il comportamento indesiderato semplicemente sovrascrivendolo con istruzioni NOP. Si noti che potrebbero essere necessarie patch più complesse se il meccanismo di anti-debugging è ben progettato;
- Usare Frida o Xposed per agganciare API sui livelli Java e nativi: manipolare i valori di ritorno di funzioni come `isDebuggable` e `isDebuggerConnected` per nascondere il debugger;
- Cambiare l'ambiente: Android è un ambiente aperto. Se non funziona nient'altro, si può modificare il sistema operativo per sovvertire i presupposti che gli sviluppatori hanno fatto quando hanno progettato i trucchi anti-debugging.

Valutazione dell'efficacia Controllare i meccanismi di anti-debugging, includendo i seguenti criteri:

- Il collegamento di debugger basati su jdb e ptrace fallisce o causa la terminazione o il malfunzionamento dell'app;
- Più metodi di rilevamento sono sparsi nel codice sorgente dell'app (invece di essere tutti in un singolo metodo o funzione);
- Le difese anti-debugging operano su più livelli API (Java, funzioni di libreria native, assembler/system call);
- I meccanismi sono in qualche modo unici (invece di essere copiati e incollati da StackOverflow o altre fonti).

Lavorare sull'aggiramento delle difese anti-debugging e rispondere alle seguenti domande:

- I meccanismi possono essere aggirati banalmente (per esempio, agganciando una singola funzione API)?
- Quanto è difficile identificare il codice anti-debugging attraverso l'analisi statica e dinamica?
- Avete dovuto scrivere codice personalizzato per disabilitare le difese? Quanto tempo vi è servito?
- Qual è la vostra valutazione soggettiva della difficoltà di bypassare i meccanismi?

Se i meccanismi di anti-debugging mancano o sono troppo facilmente aggirabili, si posso sfruttare i suggerimenti in linea con i criteri di efficacia di cui sopra. Questi suggerimenti possono includere l'aggiunta di più meccanismi di rilevamento e una migliore integrazione dei meccanismi esistenti con altre difese.

Conclusioni

Da sempre OWASP lavora per rendere lo sviluppo di software sicuro un’esperienza accessibile a tutti ed il lavoro di tesi è stato pensato e sviluppato proprio per proporre un ulteriore soluzione al fine di raggiungere questo obiettivo. Il progetto MASVS, come molti progetti OWASP, cerca di semplificare lo sviluppo di applicazioni sicure in un periodo storico in cui le tecnologie diventano sempre più complesse e la necessità di avere un determinato grado di sicurezza diventa fondamentale sia per l’utente finale che per le aziende che sviluppano software.

Il lavoro di tesi è stato realizzato per aggiungere un tassello in più al già ottimo lavoro intrapreso con il MASVS, proponendo delle demo software consultabili e affiancando ad esse degli attacchi specifici che evidenziano i problemi che scaturiscono dalla violazione dei requisiti presi in esame. L’idea di fornire questo approccio di demo e attacco mira ad aumentare la qualità del progetto MASVS ed a favorire l’immediatezza della comprensione dei requisiti rendendo ancora più semplice il lavoro degli sviluppatori. L’auspicio è quello di proporre ed integrare questo approccio in ogni requisito presente nel documento (laddove sia possibile). Il lavoro per sviluppare ciò è notevole ma il mondo open-source spesso ha mostrato la forza dell’approccio legato ai contributi da parte della comunità che permette di rendere tutto molto più realizzabile di quanto non sembri.

Bibliografia

- [1] Daniele Berardo. *Tecniche OWASP per la qualità del codice: cosa sono e chi deve applicarle, alla luce del GDPR.* 2020. URL: <https://cybersecurity360.it>.
- [2] M. Nottingham. *Well-Known Uniform Resource Identifiers (URIs).* 2019. URL: <https://datatracker.ietf.org/doc/html/rfc8615>.
- [3] OWASP. *Finding Sensitive Information in Auto-Generated Screenshots (MSTG-STORAGE-9).* URL: <https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05d-Testing-Data-Storage.md>.
- [4] OWASP. *Mobile Application Security Verification Standard.* Ver. 1.3. 2021. URL: <https://github.com/OWASP/owasp-masvs>.
- [5] OWASP. *Mobile Security Testing Guide.* Ver. 1.1.3. 2019. URL: <https://github.com/OWASP/owasp-mstg>.
- [6] OWASP. *Testing Anti-Debugging Detection (MSTG-RESILIENCE-2).* URL: <https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md>.
- [7] OWASP. *Testing Best Practices for Passwords (MSTG-AUTH-5 and MSTG-AUTH-6).* URL: <https://github.com/OWASP/owasp-mstg/blob/master/Document/0x04e-Testing-Authentication-and-Session-Management.md>.

- [8] OWASP. *Testing for Injection Flaws (MSTG-PLATFORM-2)*. URL: <https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05h-Testing-Platform-Interaction.md>.
- [9] OWASP. *Testing Random Number Generation (MSTG-CRYPTO-6)*. URL: <https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05e-Testing-Cryptography.md>.
- [10] OWASP. *Testing Whether the App is Debuggable (MSTG-CODE-2)*. URL: <https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05i-Testing-Code-Quality-and-Build-Settings.md>.
- [11] OWASP. *Verifying Data Encryption on the Network (MSTG-NETWORK-1 and MSTG-NETWORK-2)*. URL: <https://github.com/OWASP/owasp-mstg/blob/master/Document/0x04f-Testing-Network-Communication.md>.
- [12] EdOverflow & Yakov Shafranovich. *Security.txt. A proposed standard which allows websites to define security policies*. URL: <https://securitytxt.org/>.

Ringraziamenti

Ringrazio i miei relatori, il Prof. Andrea Pugliese e l'Ing. Michele Ianni, per avermi guidato durante tutto il lavoro di tesi.

Ringrazio la mia famiglia che mi ha sempre supportato e spronato a dare il meglio di me anche nelle situazioni più complesse.

Ringrazio i miei due compagni di viaggio Totò e Tony che sono stati semplicemente fantastici. Vi auguro tutto il bene di questo mondo.

Ringrazio Dania che, nonostante tutto, mi ha sopportato e mi è stata sempre vicina nei momenti difficili.

Ringrazio tutti i colleghi e amici più stretti che anche se distanti sono riusciti a darmi una mano durante questo percorso.

Ringrazio infine tutte le persone che mi hanno pensato, che mi vogliono bene e che anche se con poco, mi sono state vicine.