Vulnerabilità		Descrizione	L'applicazione è vulnerabile se	Misure per prevenire la vulnerabilità
RM1	**M1-Improper Platform Usage**	Questa categoria copre l'uso improprio di una funzionalità della piattaforma o il mancato utilizzo dei controlli di sicurezza della piattaforma. Potrebbe includere Android intents, autorizzazioni della piattaforma, uso improprio di TouchID, Keychain o altri controlli di sicurezza che fanno parte del sistema operativo mobile. Esistono diversi modi in cui le app mobili possono sperimentare questo rischio.	La caratteristica dei rischi in questa categoria è che la piattaforma (iOS, Android, Windows Phone, ecc.) fornisce una funzionalità o una capacità documentata e ben compresa., ma l'app non utilizza tale funzionalità o la utilizza in modo errato. Esistono diversi modi in cui le app mobili possono sperimentare questo rischio: • Violazione delle linee guida pubblicate. Tutte le piattaforme hanno linee guida di sviluppo per la sicurezza (c.f., ((Android)), ((iOS)), ((Windows Phone))). Se un'app contraddice le best practices consigliate dal produttore, sarà esposta a questo rischio. • Violazione della convenzione o pratica comune. Non tutte le best practices sono codificate nella guida del produttore. In alcuni casi, ci sono best practices di fatto comuni nelle app mobili. • Uso improprio involontario. Alcune app pensano di fare la cosa giusta, ma in realtà sbagliano parte dell'implementazione. Potrebbe trattarsi di un semplice bug, come impostare un flag errato su una chiamata API o potrebbe essere un misunderstanding sul funzionamento delle protezioni. Gli errori nei modelli di autorizzazione della piattaforma rientrano in questa categoria.	Il secure coding e le pratiche di configurazione devono essere utilizzate sul lato server dell'applicazione mobile.
RM2	**M2-Insecure Data Storage**	Questa categoria riguarda la memorizzazione non sicura dei dati e la perdita involontaria di dati	I dati archiviati in modo non sicuro includono, ma non sono limitati a, i seguenti: • Database SQL; • Log files; • Archivi di dati XML o file manifest; • Archivi di dati binari; •Cookie stores; • Scheda SD; • Cloud sincronizzato. La perdita involontaria di dati include, a titolo esemplificativo, le vulnerabilità di: • Sistema operativo; • Frameworks; • Ambiente di compilazione; •Nuovo hardware; • Dispositivi con root o jailbreak. Nello sviluppo mobile in particolare, questo è più evidente nei processi interni non documentati o poco documentati come: • Il modo in cui il sistema operativo memorizza nella cache dati, immagini, pressioni di tasti, registrazione e buffer; • Il modo in cui il framework di sviluppo memorizza nella cache dati, immagini, pressioni di tasti, registrazione e buffer; • Il modo o la quantità in cui i framework analitici, social o di abilitazione memorizzano nella cache dati, immagini, pressioni di tasti, registrazione e buffer.	È importante comprendere quali tipi di funzionalità vengono usate dall'app e il modo in cui le API gestiscono tali risorse. È fondamentale conoscere come vengono gestiti i seguenti tipi di funzionalità: • URL caching (both request and response); • Keyboard press caching; • Copy/Paste buffer caching; • Application backgrounding; • Intermediate data • Logging; • HTML5 data storage; • Browser cookie objects; • Analytics data sent to 3rd parties.
RM3	**M3-Insecure Communication**	Questa categoria comprende handshaking scadente, versioni SSL non corrette, negoziazione debole, comunicazione in chiaro di risorse sensibili, ecc	Questo rischio copre tutti gli aspetti in cui il trasferimento dei dati avviene in modo non sicuro. Sono comprese le comunicazioni da mobile a mobile, da app a server e tutti gli altri tipi di comunicazione da mobile. Questo rischio include tutte le tecnologie di comunicazione che un dispositivo mobile potrebbe utilizzare: TCP / IP, WiFi, Bluetooth / Bluetooth-LE, NFC, audio, infrarossi, GSM, 3G, SMS, ecc. Rientrano in questa categoria anche tutti i problemi di comunicazione TLS e tutti i problemi NFC, Bluetooth e WiFi. Le caratteristiche principali di questo rischio includono l'impacchettamento di alcuni tipi di dati sensibili e la loro trasmissione all'interno o all'esterno del dispositivo. Alcuni esempi di dati sensibili includono chiavi di crittografia, password, informazioni private dell'utente , dettagli dell'account, token di sessione, documenti, metadati e file binari. I rischi di comunicazioni non sicure riguardano l'integrità dei dati, la riservatezza dei dati e l'integrità dell'origine della comunicazione. Ad esempio: • se i dati possono essere modificati durante il transito, senza che la modifica sia rilevabile (ad es. tramite un attacco man-in-the-middle); • se i dati riservati possono essere esposti, appresi o derivati osservando le comunicazioni in tempo reale (ovvero, intercettando) o registrando la conversazione in tempo reale e attaccandola in seguito (attacco offline); • la mancata corretta impostazione e convalida di una connessione TLS (ad es. controllo del certificato, cifre deboli, altri problemi di configurazione TLS).	•Qualora il livello di rete non e' sicuro ed è suscettibile di intercettazione é opportuno applicare SSL / TLS ai canali di trasporto che l'app mobile utilizzerà per trasmettere informazioni riservate, token di sessione o altri dati sensibili a un'API di back-end o ad un servizio Web. • E' buona prassi che le entità esterne come società terze utilizzino le loro versioni SSL quando un'applicazione esegue una routine tramite il browser / webkit. •Evitare sessioni SSL miste in quanto potrebbero esporre l'ID di sessione dell'utente. •Utilizzare suite robuste di cifratura standard di settore. con lunghezze delle chiave appropriate. •Utilizzare i certificati firmati da un provider di CA attendibile. • Non consentire mai certificati autofirmati ed implementare il certificate pinning per applicazioni sensibili alla sicurezza. •Richiedere sempre la verifica della SSL chain. •Stabilire una connessione sicura solo dopo aver verificato l'identità del server endpoint utilizzando certificati attendibili nella key chain. •Avvisare gli utenti tramite l'interfaccia utente se l'app mobile rileva un certificato non valido. Non inviare dati sensibili su canali alternativi (ad es. SMS, MMS o notifiche). • Se possibile, applicare un livello separato di crittografia a tutti i dati sensibili prima che vengano trasmessi al canale SSL. **Best practice specifiche per iOS** Le classi predefinite nell'ultima versione di iOS gestiscono molto bene la negoziazione del livello di cifratura SSL. Il problema si presenta quando gli sviluppatori aggiungono temporaneamente il codice per bypassare queste impostazioni predefinite per superare gli ostacoli di sviluppo. Oltre alle pratiche generali di cui sopra: •Assicurarsi che i certificati siano validi e non vengano chiusi. •Quando si utilizza CFNetwork, considerare l'utilizzo di Secure Transport API per designare certificati client attendibili. In quasi tutte le situazioni, NSStreamSocketSecurityLevelTLSv1 dovrebbe essere utilizzato per un livello di crittografia standard più elevato. •Dopo lo sviluppo, assicurarsi che tutte le chiamate NSURL (o wrapper di NSURL) non consentano certificati autofirmati o non validi come NSURL class method setAllowsAnyHTTPSCertificate. •Prendere in considerazione l'utilizzo del certificate pinning procedendo come segue: esportare il certificato, includerlo nell'app bundle e ancorarlo all' oggetto attendibile. **Best practice specifiche per Android** •Rimuovere tutto il codice dopo il ciclo di sviluppo che può consentire all'applicazione di accettare tutti i certificati come org.apache.http.conn.ssl.AllowAllHostnameVerifier o SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER. Questi consentono di fidarsi di tutti i certificati. •Se si utilizza una classe che estende SSLSocketFactory, assicurarsi che il metodo checkServerTrusted sia implementato adeguatamente in modo che il certificato del server sia verificato correttamente.

RM4	**M4-Insecure Authentication**	Questa categoria riguarda le informazioni di autenticazione dell'utente finale o di cattiva gestione della sessione. Sono inclusi i seguenti casi: • Non riuscire a identificare l'utente quando ciò dovrebbe essere richiesto • Incapacità di mantenere l'identità dell'utente quando richiesto • Debolezze nella gestione della sessione	Esistono diversi modi in cui un'app mobile può presentare autenticazione non sicura: •Se l'app per dispositivi mobili è in grado di eseguire in modo anonimo una richiesta del servizio API di back-end senza fornire un token di accesso. •Se l'app per dispositivi mobili memorizza password o shared secret localmente sul dispositivo. •Se l'app mobile utilizza una politica di password debole per semplificare l'immissione di una password. •Se l'app per dispositivi mobili utilizza una funzionalità come TouchID.	**Evitare autenticazioni deboli** Evitare i seguenti modelli di progettazione di autenticazione delle applicazioni mobili non sicuri: • Se si esegue il porting di un'applicazione Web sul suo equivalente mobile, l'autenticazione deve prevedere gli stessi fattori e rispettare gli stessi requisiti della versione web. •L'autenticazione locale di un utente può portare vulnerabilità di bypass lato client. Se l'applicazione memorizza i dati localmente, la routine di autenticazione può essere bypassata su dispositivi jailbroken tramite manipolazione del runtime o modifica del file binario. •Laddove possibile, assicurarsi che tutte le richieste di autenticazione vengano eseguite sul lato server. In caso di autenticazione riuscita, i dati dell'applicazione verranno caricati sul dispositivo mobile. Ciò garantirà che i dati dell'applicazione saranno disponibili solo dopo l'autenticazione corretta; •Se è richiesta la memorizzazione dei dati sul lato client, i dati dovranno essere crittografati utilizzando una chiave di crittografia derivata in modo sicuro dalle credenziali di accesso dell'utente. Ciò garantirà che i dati dell'applicazione memorizzati saranno accessibili solo dopo aver inserito con successo le credenziali corrette. •Le funzionalità di autenticazione persistente (Remember Me) implementate all'interno delle applicazioni mobili non devono mai archiviare la password di un utente sul dispositivo; •Idealmente, le applicazioni mobili dovrebbero utilizzare un token di autenticazione specifico per dispositivo che può essere revocato all'interno dell'applicazione mobile dall'utente. Ciò garantirà che l'app possa limitare l'accesso non autorizzato da un dispositivo rubato o smarrito; •Non utilizzare spoofable value per l'autenticazione di un utente. Ciò include identificatori di dispositivi o geolocalizzazione; L'autenticazione persistente all'interno delle applicazioni mobili dovrebbe essere implementata come opt-in e non abilitata per impostazione predefinita; •Se possibile, non consentire agli utenti di fornire PIN a 4 cifre per le password di autenticazione. **Rafforzare l'autenticazione.** •Gli sviluppatori dovrebbero presumere che tutti i controlli di autorizzazione e autenticazione sul lato client possano essere Bypassati da utenti malintenzionati. I controlli di autorizzazione e autenticazione devono essere rafforzati sul lato server ogni volta che è possibile. •A causa dei requisiti di utilizzo offline, le app mobili potrebbero richiedere di eseguire autenticazione o autorizzazione locali nel codice dell'app mobile. In tal caso, gli sviluppatori dovrebbero prevedere i controlli locali di integrità all'interno del proprio codice per rilevare eventuali modifiche non autorizzate al codice stesso.
RM5	**M5-Insufficient Cryptography**	Questa categoria riguarda i casi in cui il codice applica la crittografia alle informazioni sensibili, ma la crittografia è in qualche modo insufficiente o non è eseguita correttamente. Si noti che qualsiasi cosa relativa a TLS o SSL va in M3. Inoltre, se l'app non utilizza affatto la crittografia quando dovrebbe, probabilmente appartiene a M2.	L'uso non sicuro della crittografia è comune nella maggior parte delle app mobili che sfruttano la crittografia. Esistono due modi fondamentali con cui una debole crittografia si manifesta all'interno delle app mobili: • In primo luogo, l'app mobile può utilizzare un processo che contiene crittografia / decrittografia fondamentalmente imperfetto che può essere sfruttato dall'avversario per decrittografare i dati sensibili. • In secondo luogo, l'app per dispositivi mobili può implementare o sfruttare un algoritmo di crittografia / decrittografia di natura debole che può essere decodificato direttamente dall'avversario.	È consigliabile effettuare le seguenti operazioni quando si gestiscono dati sensibili: • Evitare, ove possibile, la memorizzazione di dati sensibili su un dispositivo mobile. • Applicare standard crittografici che resistano alla prova del tempo per almeno 10 anni nel futuro. • Seguire le linee guida del NIST sugli algoritmi raccomandati .
RM6	**M6-Insecure Authorization**	Questa categoria riguarda eventuali errori nell'autorizzazione (ad es. Decisioni di autorizzazione sul lato client, navigazione forzata, ecc.). ed è distinta dai problemi di autenticazione (ad es. Registrazione del dispositivo, identificazione dell'utente, ecc.). Se l'app non autentica affatto gli utenti in una situazione in cui dovrebbe (ad esempio, concedere l'accesso anonimo ad alcune risorse o servizi quando è richiesto l'accesso autenticato e autorizzato), si tratta di un errore di autenticazione e non di un errore di autorizzazione.	È importante riconoscere la differenza tra autenticazione e autorizzazione. L'autenticazione è il processo di identificazione di un individuo. L'autorizzazione è il processo che verifica che l'individuo identificato disponga delle autorizzazioni necessarie per eseguire una determinata attività. È essenzialmente impossibile che si verifichino controlli di autorizzazione su una richiesta in arrivo quando non viene stabilita l'identità del chiamante. Esistono alcune semplici regole da seguire quando si cerca di determinare se un endpoint mobile soffre di autorizzazioni non sicure: • **Presenza di vulnerabilità di Tipo Insecure Direct Object Reference (IDOR)**: se si riscontra una vulnerabilità tipo IDOR, è molto probabile che il codice non esegua un controllo di autorizzazione valido; • **Endpoint nascosti**: in genere, gli sviluppatori non eseguono controlli di autorizzazione sulla funzionalità nascosta di back-end in quanto presumono che la funzionalità nascosta venga vista solo da qualcuno conruolo appropriato; • **Trasmissioni di ruoli o autorizzazioni utente** : se l'app mobile sta trasmettendo i ruoli o le autorizzazioni dell'utente a un sistema back-end come parte di una richiesta, soffre di un'autorizzazione non sicura;	Per evitare controlli di autorizzazione non sicuri, procedere come segue: • Verificare i ruoli e le autorizzazioni dell'utente autenticato utilizzando solo le informazioni contenute nei sistemi di back-end. Evitare di fare affidamento su ruoli o informazioni di autorizzazione che provengono dal dispositivo mobile stesso; • Il codice di back-end deve verificare in modo indipendente che tutti gli identificatori in arrivo associati a una richiesta, forniti con l'identificativo, corrispondano e appartengano all'identità in arrivo;
RM7	**M7 - Client Code Quality**	Questa categoria riguarda i problemi di implementazione a livello di codice nel client mobile ed è distinta dagli errori di codifica lato server. In questa categoria rientrano buffer overflows, vulnerabilità delle stringhe di formato e vari altri errori a livello di codice in cui la soluzione è quella di riscrivere del codice in esecuzione sul dispositivo mobile.	Questa vulnerabilità deriva dall'utilizzo di un' API errata, dall'utilizzo di un'API in modo non sicuro, dall'utilizzo di costrutti del linguaggio non sicuri o da altri problemi a livello di codice. Cosa importante: questo non è un codice in esecuzione sul server. Questo è un rischio che cattura il codice errato che viene eseguito sul dispositivo mobile stesso.	In generale, è possibile evitare problemi di qualità del codice procedendo come segue: •Mantenere modelli di codifica coerenti su cui tutti nell'organizzazione concordano; •Scrivere un codice facile da leggere e ben documentato; •Quando si utilizzano i buffer, verificare sempre che la lunghezza dei dati del buffer in entrata non superi la lunghezza del buffer di destinazione; •Tramite l'automazione, identificare i buffer overflow e le perdite di memoria attraverso l'uso di strumenti di analisi statica di terze parti; •Dare la priorità alla risoluzione di buffer overflow e perdite di memoria rispetto ad altri problemi di "qualità del codice".
		Questa categoria comprende binary patching, modifica delle risorse locali, method hooking, method swizzling e		L'app mobile deve essere in grado di riconoscere in fase di esecuzione se é stato aggiunto o modificato codice rispetto a quello validato come integro al momento della compilazione. L'app deve essere in grado di reagire in modo appropriato in fase di esecuzione a una violazione dell'integrità del codice. Le strategie di riparazione per questo tipo di rischio sono descritte in modo più dettagliato nell'ambito del Progetto di prevenzione del reverse engeneering e di modifica del codice di OWASP. Android Root Detection In genere, un'app che è stata modificata verrà eseguita in un ambiente con jailbreak o root. Pertanto, è ragionevole provare a rilevare questi tipi di ambienti compromessi in fase di runtime e reagire di conseguenza . Esistono alcuni modi comuni per rilevare un dispositivo Android con root. Check for test-keys:

RM8	**M8 - Code Tampering**	modifica dinamica della memoria. Una volta che l'applicazione viene installata sul dispositivo mobile, il codice e i dati vi risiedono. Un utente malintenzionato può modificare direttamente il codice, modificare il contenuto della memoria in modo dinamico, cambiare o sostituire le API di sistema utilizzate dall'applicazione o modificare i dati e le risorse dell'applicazione. Ciò può fornire all'attaccante un metodo diretto per sovvertire l'uso previsto del software a scopo di lucro personale o monetario.	Tecnicamente, tutto il codice mobile è vulnerabile al code tampering. Il codice mobile viene eseguito in un ambiente che non è sotto il controllo dell'organizzazione produttrice del codice. Allo stesso tempo, esistono modi diversi per modificare l'ambiente in cui viene eseguito quel codice. Queste modifiche consentono a un attaccante di manipolare il codice e modificarlo a piacimento. Sebbene il codice mobile sia intrinsecamente vulnerabile, è importante chiedersi se valga la pena rilevare e provare a prevenire modifiche non autorizzate al codice. Prima di decidere se affrontare o meno questo rischio è fondamentale considerare l'impatto aziendale dell' app .	Check for test-keys: • Verifica se build.prop include la riga ro.build.tags = test-keys che indica una build dello sviluppatore o una ROM non ufficiale Controlla i certificati OTA: • Controlla se esiste il file /etc/security/otacerts.zip Controlla alcuni apk root noti: • com.noshufou.android.su • com.thirdparty.superuser • eu.chainfire.supersu • com.koushikdutta.superuser Controlla gli SU binaries : • / System / bin / su • / System / xbin / su •/ Sbin / su •/ Sistema / su • /system/bin/.ext/.su Tentare direttamente il comando SU: • Tentare di eseguire il comando SU e verificare l'id dell'utente corrente; se restituisce 0, il comando SU ha avuto esito positivo
RM9	**M9 - Reverse Engineering**	Questa categoria include l'analisi del codice binario finale per determinare il codice sorgente, le librerie, gli algoritmi e altre risorse. Software come IDA Pro, Hopper, otool e altri strumenti di ispezione binaria forniscono all'attaccante una visione del funzionamento interno dell'applicazione. Questo può essere usato per sfruttare ulteriori vulnerabilità dell'applicazione, nonché per rivelare informazioni su server di back-end, costanti e cifre crittografiche e proprietà intellettuale.	In genere, la maggior parte delle applicazioni è sensibile al reverse engineering per la natura intrinseca del codice. La maggior parte degli attuali linguaggi di programmazione utilizzati per scrivere app sono ricchi di metadati che aiutano notevolmente un programmatore nel debug dell'app. Ma questa caratteristica aiuta anche un malintenzionato a capire come funziona l'app. Si dice che un'app sia sensibile al reverse engineering se un utente malintenzionato può eseguire una delle seguenti operazioni: • Comprende chiaramente il contenuto di una binary's string table • Esegue con precisione analisi interfunzionali • Riesce in modo ragionevolmente accurato a ricreare il codice sorgente a partire dal binario.	Al fine di prevenire un efficace reverse engineering, è necessario utilizzare un tool di offuscamento. Ci sono molti tool di offuscamento gratuiti e commerciali sul mercato. Di contro, ci sono anche molti deoffuscatori sul mercato. Per misurare l'efficacia di qualsiasi tool di offuscamento scelto, prova a deoffuscare il codice utilizzando strumenti come IDA Pro e Hopper. Un buon offuscatore: • Limita i segmenti dei metodi / codici da offuscare; • Ottimizza il grado di offuscamento per bilanciare l'impatto delle prestazioni; • Resiste al deoffuscamento di tool come IDA Pro e Hopper; • Offusca anche le string tables oltre i metodi.
RM10	**M10-Extraneous Functionality**	Spesso, gli sviluppatori includono funzionalità backdoor nascoste o altri controlli di sicurezza di sviluppo interni che usano in ambiente di test e che non sono destinati a essere rilasciati in un ambiente di produzione. Ad esempio, uno sviluppatore può includere accidentalmente una password come commento in un'app ibrida. Un altro esempio include la disabilitazione dell'autenticazione a 2 fattori durante il test.	Questa vulnerabilità si verifica quando vengono lasciate abilitate nell'app funzionalità usate in ambiente di test che non sono previste in fase di rilascio.	Il modo migliore per prevenire questa vulnerabilità è eseguire una revisione manuale del codice sicuro, ricorrendo ad esperti di sicurezza e a sviluppatori esperti del codice utilizzato. Si dovrebbe: • Esaminare le impostazioni di configurazione dell'app per scoprire eventuali switch nascosti; • Verificare che tutto il codice di test non sia incluso nella build di produzione finale dell'app; • Esaminare tutti gli endpoint API a cui accede l'app mobile per verificare che questi endpoint siano ben documentati e disponibili agli utenti; • Esaminare tutti i log per assicurarsi che non ci sia niente di eccessivamente descrittivo sul back-end.