Standard guida del settore di terzi sulla "codifica sicura"

Edizione finale 1.0

Data: 19/10/2016,

BT Security

Indice

1. Introduzione

1.1. Background

Il presente documento definisce una guida generale sulle buone prassi per lo sviluppo di software da parte di terzi che forniscono applicazioni o sistemi a BT.

Le vulnerabilità a livello delle applicazioni rappresentano un modo estremamente efficace per accedere senza autorizzazione a dati o sistemi. Pertanto è sempre più importante che la modalità di sviluppo di un software sia incentrata su valutazioni sulla sicurezza.

1.2. Obiettivi

Il presente documento è volto a fornire linee guida su come sviluppare i software in un modo tale che il codice risultante sia affidabile e sicuro. Un progetto di sviluppo di software presenta molti aspetti; è un campo costantemente in evoluzione quindi, per quanto possibile, è stato delineato un insieme di principi fondamentali per le differenti fasi di progettazione. Il presente documento è diviso in tre sezioni principali:

Progettazione del sistema

Questa sezione fornisce linee guida per la fase iniziale della progettazione di un software e definisce diversi elementi da prendere in considerazione prima della scrittura del codice. Seguendo questi passi, durante la vita del progetto sarà più facile fornire un codice sicuro.

Fornitura del sistema

Questa sezione si occupa dello sviluppo del codice stesso e affronta alcuni principi generali (ad esempio la gestione di un codice sicuro) e come evitare alcuni dei problemi/errori più comuni.

Assicurazione del sistema:

Questa sezione si occupa dell'implementazione e della gestione del codice prodotto durante la sua vita. Questa parte potrebbe non essere sempre applicabile a seconda del contratto specifico ma è necessario accertarsi che la propria applicazione o il proprio sistema sia in grado di funzionare nel rispetto di queste linee guida.

Inoltre, abbiamo incluso molteplici appendici che definiscono le specifiche di BT per il controllo degli accessi e la crittografia. È importante assicurarsi che la propria applicazione sia in grado di conformarsi a tali controlli.

1.3. Materiale aggiuntivo e suggerimenti

Vedi link nel documento.

2. Progettazione del sistema

Rif.	Controllo	Motivo
1	I dati dovrebbero essere classificati secondo lo Standard di classificazione dei dati in linea con quanto stabilito all'interno della vostra organizzazione. Sono incluse informazioni sullo sviluppo e progettazione (ad es. progetti, edizioni e requisiti) e dati sull'applicazione. La manutenzione dei dati deve avvenire secondo la Politica di conservazione delle informazioni.	Dopo aver individuato il tipo di dati che dovrà contenere il vostro sistema, è possibile implementare le misure protettive adeguate. È importante proteggere le informazioni sullo sviluppo e la progettazione (in particolari i dati sensibili) poiché accessi non autorizzati potrebbero permettere a qualcuno di prendere di mira l'applicazione.
2	Tutti i requisiti di legge e regolamentari devono essere identificati.	Durante la scrittura di applicazioni che gestiscono alcuni tipi di dati (ad es. informazioni su carte di credito) vi sono specifici requisiti di legge e regolamentari che determineranno la gestione di tali dati.
3	Durante il processo di progettazione è necessario valutare se vi sono requisiti di sicurezza per ciascuna persona che lavora al progetto (ad es. solo il Regno Unito oppure se è richiesto un nulla osta di sicurezza).	I progetti che contengono informazioni sensibili o governative possono richiedere la gestione da parte di personale specifico.
4	Utilizzare tecniche di analisi del rischio per identificare e quantificare rischi di sicurezza dettagliati. Documentare i rischi e le azioni di mitigazione proposte.	Ogni applicazione avrà un insieme differente di rischi associati (ad es. in quale ambiente opera, quale livello di dati gestisce). È importante assicurarsi di aver preso ciò in considerazione e aver determinato quali misure di protezione adottare. L'analisi del rischio può dipendere dall'ambiente e

		a tal fine abbiamo a disposizione parecchi strumenti. OWASP ha prodotto una guida per identificare e valutare i rischi, che include un'utile sezione su come classificarli. In alternativa, Microsoft ha sviluppato uno strumento che aiuta a identificare le minacce alla vostra applicazione, e un altro strumento che serve a individuare la vostra superficie di attacco prima e dopo l'implementazione di nuove applicazioni Se avete domande o avete bisogno di consigli sulla metodologia, contattate il proprietario dello standard. Nota: La tecnica da usare potrebbe essere specifica per il progetto, ad esempio IS1 per un lavoro accreditato da HMG.
5	Definire e documentare un calendario di revisione dei rischi.	Servirà a garantire che i rischi siano aggiornati in base a eventuali modifiche nell'applicazione o nel suo ambiente. Si possono aggiungere nuovi rischi e intraprendere azioni di mitigazione.
6	Documentare l'architettura di sistema per far fronte a ciascun rischio identificato sopra Fornire linee guida in materia di sicurezza per implementazione/progettazioni di basso livello librerie affidabili cifre accettabili algoritmi hash accettabili lunghezze minime delle chiavi procedure di gestione delle chiavi crittografia/separazione dell'archiviazione accesso/trasmissione di credenziali	Servirà a garantire di applicare in modo coerente le azioni di mitigazione adeguate ai rischi che sono stati identificati. Servirà anche a progettare le prove di accettazione.

	protocolli inaccettabili.	
7	Definire e documentare un ciclo di vita di sviluppo.	Avere un ciclo di vita di sviluppo significa che c'è un approccio ripetibile coerente per passare da requisiti/problemi a una risoluzione nella produzione. Senza questo, correggere i problemi di sicurezza può essere difficile e dispendioso in termini di tempo. Microsoft ha un modello che può essere utilizzato come punto di partenza.
8	Documentare e implementare le seguenti procedure di progettazione: Controllo degli accessi (chi può accedere a quali parti dell'applicazione e come tale accesso viene richiesto, gestito e revocato): se non viene effettuato, potrebbero verificarsi accessi indesiderati (ad es. utenti in grado di eseguire compiti amministrativi o limitare l'accesso dopo lo spostamento dei lavori) Sincronizzazione dell'orologio: se gli orologi non sono precisi può essere molto difficile usare i file di log per identificare i problemi, specialmente quando è necessario correlarsi con attività su altri sistemi. Trasferimento di dati: i dati in transito devono essere protetti in modo adeguato a seconda della loro classificazione. Come usare gli strumenti e le tecniche per attaccare il sistema: vi consentirà di anticipare i tentativi e implementare azioni di mitigazione sin dall'inizio.	La sicurezza dell'applicazione dipende fortemente dalla sua gestione durante la sua vita. Definendo e implementando i processi suggeriti è possibile garantire che il lavoro svolto nella fase di progettazione e implementazione non sia annullato durante il funzionamento dell'applicazione. Si tratta in gran parte di seguire "buon senso" in modo naturale. Tuttavia, è importante specificare i processi in modo esplicito per assicurarsi che vengano seguiti durante tutta la vita dell'applicazione (ad esempio se cambia il team di supporto).

	Disaster recovery/fallback: è necessario definirli e testarli in modo chiaro affinché il sistema possa reagire a eventi inaspettati. Documentare e implementare le seguenti procedure di sviluppo: Includere strumenti di prova in integrazione continua: un ambiente di build automatizzato consente di testare automaticamente i problemi più comuni di ciascuna build prima del rilascio dell'applicazione. Utilizzare strumenti di analisi di sicurezza statici e dinamici come parte del processo di sviluppo: vi aiuterà a trovare vulnerabilità sin dalle prime fasi del processo di sviluppo. Tagging dei problemi di sicurezza nel vostro repository dei sorgenti e issue tracker: taggando i bug di sicurezza è possibile valutare l'efficacia delle azioni di mitigazione/correzione da valutare. Possono anche confluire nell'analisi del rischio del progetto e i problemi comuni possono essere condivisi con altri sviluppatori per migliorare la consapevolezza generale. Utilizzare processi di peer review: la peer review del codice può identificare problemi che non possono essere rilevati automaticamente e prevenire vulnerabilità causate da errori di codifica, ad esempio errori logici. Implementazione da un ambiente di	

	build pulito: utilizzare un ambiente di build affidabile e pulito riduce il rischio che l'output possa essere inquinato da malware, librerie non desiderate/obsolete o componenti che potrebbero causare guasti o compromettere la sicurezza. Condivisione delle best practice e dell'esperienza: può aiutare la comunità ad evitare problemi comuni che possono provocare vulnerabilità. Documentare e implementare le seguenti procedure di gestione durante la vita: Applicazione puntuale di patch di sicurezza incluse quelli per componenti di terzi: serve a correggere problemi di sicurezza che potrebbero compromettere la vostra applicazione e i suoi dati. Manutenzione di hardware & software: è importante pianificare la manutenzione di hardware e software supportati in modo da correggere i problemi e ricevere aggiornamenti di sicurezza. Avvio/chiusura applicazione: la chiusura incompleta dell'applicazione può causare comportamenti indesiderati oppure la mancata pulizia degli artefatti. Gestione del lavoro e dell'applicazione: definire il modo in cui l'applicazione gestisce i lavori, specialmente in caso di carichi insolitamente pesanti. Monitoraggio e allarmi: definire e implementare il modo in cui vengono	

	monitorati e gestiti i log e gli altri output. Archiviazione/pulizia dei dati: deve essere possibile gestire i dati in modo da poterli rimuovere o archiviare quando non sono più necessari. Questo potrebbe essere un requisito delle leggi in materia di protezione dei dati. Migrazione: consente la continuazione del servizio dopo un upgrade, ad esempio per correggere problemi di sicurezza. Controllo delle modifiche: è importante che le modifiche all'applicazione siano controllate in modo che sia possibile capire se sono legittime e, se si verificano dei problemi, quali possono essere state le cause.	
9	L'architettura di sicurezza deve essere revisionata e aggiornata ad ogni ciclo o iterazione. Modifiche pianificate all'applicazione devono essere revisionate per assicurarsi che non compromettano i requisiti di sicurezza.	Serve a garantire che sia aggiornata con le modifiche nei requisiti e nell'implementazione del progetto.
10	Utilizzare kit di strumenti di sicurezza standard del settore che vengono sottoposti a manutenzione attiva piuttosto che progettarne di propri.	I kit di strumenti comunemente usati sono testati con cura, è meno probabile che contengano grossi errori e verranno sottoposti a patch in caso di problemi. Risparmierete anche del tempo! Ad esempio: OpenSSL, OpenSSH, Solaris KCF, Windows Cryptography API, Active Directory, OpenLDAP.
11	Utilizzare solo componenti e servizi richiesti per fornire la funzione dei prodotti.	Minimizzando le funzioni non necessarie, si riduce l'esposizione della vostra applicazione e si può ridurre l'impatto di qualsiasi bug.

	Non esporre interfacce diverse da quelle richieste.	
12	Utilizzare solo librerie o componenti di terzi che sono supportati e sottoposti a manutenzione in modo attivo: per software supportati commercialmente, ciò significa che dovete mantenere un accordo di supporto con il vendor per la durata di vita dell'applicazione.	Significa che in caso di problemi di sicurezza, potrete aggiornare i componenti che la vostra applicazione sta usando. Per i software open-source, a volte può essere difficile da valutare poiché la manutenzione si basa sull'impegno della comunità. In caso di dubbio, utilizzare solo i componenti dei quali siete in grado di gestire la manutenzione.
13	La funzione di sicurezza deve essere definita in un insieme minimo di posizioni chiaramente definite e documentate nel codice. Documentare la posizione di una funzione di sicurezza	È più facile valutare l'impatto delle modifiche e correggere i punti deboli di sicurezza. È più facile capire e testare la funzione di sicurezza se si trova in un numero limitato di posizioni piuttosto che in molteplici posizioni all'interno di un codebase di grandi dimensioni.
14	Assicurarsi che siano utilizzate e sottoposte a manutenzione le attuali versioni testate dei componenti.	Versioni precedenti dei software possono spesso contenere bug o punti deboli di sicurezza che possono causare errori o accessi non autorizzati all'applicazione.
15	L'accesso al sistema deve seguire i controlli nella seguente sezione Controllo e gestione degli accessi.	In questo modo solo le persone autorizzate hanno accesso ai dati dell'applicazione.
16	L'implementazione di default del sistema deve prevedere privilegi minimi.	Riduce il rischio che un'implementazione di default si traduca in un accesso con privilegio elevato non necessario. Se è necessario, si deve esplicitamente fare in modo che non si venga dimenticato durante l'implementazione.
17	Il software deve funzionare solo con i privilegi necessarie per svolgere i suoi compiti dichiarati. Se necessario, ottenere un privilegio elevato secondo modalità conosciute come documentato	In caso di funzionamento con un privilegio minimo, l'impatto di bug o punti deboli di sicurezza può essere ridotto poiché il loro exploit avrà solo un accesso minimo.

	nell'architettura di sicurezza e successivamente cederlo il prima possibile.	
18	I compiti che richiedono privilegi elevati persistenti devono essere separati dal software principale dell'applicazione e soggetti al massimo controllo durante la peer review. Prestare particolare attenzione ai compiti che chiamano/richiamano software non attendibili.	In questo modo tutti i bug di sicurezza o i punti deboli nell'applicazione principale non provocheranno accessi con privilegio elevato. Riducendo la quantità di codice che viene eseguito con privilegi elevati, si riducono i rischi di bug o punti deboli di sicurezza con un conseguente accesso con privilegio elevato.
19	Creare un piano per testare la vostra applicazione, che includa delle prove per testare tutti i controlli di sicurezza o le azioni di mitigazione che avete implementato.	
20	Definire i confini di sicurezza che consentono di controllare: i diritti di accesso ai dati l'autenticità e l'integrità dei messaggi la convalida di input e output	Garantisce che i controlli siano applicati nel posto giusto per proteggere i dati nell'applicazione. La mancata implementazione dei controlli nel posto giusto può determinare un accesso non autorizzato o la modifica dei dati.
21	I sistemi devono garantire che l'integrità dei dati sia preservata.	Uno scarso controllo sull'integrità dei dati potrebbe causare delle frodi, la non conformità ai requisiti normativi come Sarbanes-Oxley oppure agli standard di settore come PCI DSS.

3. Fornitura del sistema

Rif.	Politica	Motivo
22	Archiviare il codice sorgente in un ambiente che limita l'accesso in modo che solo chi è autorizzato possa apportare modifiche.	Consentirà di ridurre il rischio di modifiche non autorizzate o dannose che potrebbero compromettere la sicurezza dell'applicazione. Utilizzare un sistema di controllo di revisione, come Git o SVN, con accesso riservato a un gruppo di soggetti

		specifici è il modo più semplice per soddisfare questo controllo.
23	Garantire che tutte le modifiche al codice siano collegate a un soggetto specifico. Ogni modifica deve essere verificabile tramite audit per identificare chi ha fatto cosa, quando e come.	Ciò significa che eventuali modifiche al codice (che introducono ad esempio funzionalità indesiderate) possono essere ricollegate a un soggetto. Utilizzare un sistema di controllo di revisione, come Git o SVN, con accesso riservato a un gruppo di soggetti specifici è il modo più semplice per soddisfare questo controllo.
24	Seguire le linee guida sulla best practice specifiche per il vostro ambiente di sviluppo o tipo di applicazione. Assicurarsi di essere a conoscenza dei problemi comuni della piattaforma o del linguaggio che si sta utilizzando.	Questi sono spesso pubblicati dai vendor dei toolset o da altri gruppi terzi. Capirli e seguirli vi aiuterà a evitare problemi comuni: Ecco un elenco delle guide comunemente utilizzate: Applicazioni web: OWASP top 10 fornisce dettagli sulle vulnerabilità comuni delle web-app. Android Apps: CERT android secure coding standard: Windows / Windows Phone: iOS / Mac OS X: Guida Secure Coding di Apple: Linux: Per ulteriori informazioni o consigli contattare il proprietario dello standard.
25	Utilizzare uno stile di codifica coerente documentato.	Se gli stili di codifica non sono coerenti in tutto il codebase dell'applicazione, può essere difficile capire la logica dell'applicazione e identificare i problemi che possono causare comportamenti indesiderati. La maggior parte dei vendor produce una guida di stile per il loro linguaggio/ambiente; in assenza di altri requisiti si raccomanda di utilizzarla. Alcune guide di stile suggerite:

		Java – Standard di codifica SEI CERT Oracle per Java Linee guida sulla codifica sicura C/C++ - CERT's o linee guida MISRA-C C#: Python – PEP8 php – Guida di stile di CodeIgniter
26	Utilizzare catene di strumenti supportate e affidabili.	Gli strumenti supportati sono quelli che sono prontamente disponibili e sottoposti attivamente a manutenzione (da un singolo vendor, ad es. Microsoft Visual Studio, o da una comunità, ad es. GNU Compilers Collection). Gli strumenti attendibili sono quelli forniti (e firmati) da una parte attendibile (ad es. Microsoft, Ubuntu) o quelli ampiamente disponibili e verificabili tramite hash ben noti (ad es. download di sorgente GCC). Utilizzando strumenti supportati riceverete gli aggiornamenti per correggere i bug e altri problemi di sicurezza. Verificando l'affidabilità degli strumenti che si utilizzano, si garantisce che i vostri toolset non introducano funzionalità indesiderate.
27	Sottoporre a manutenzione le catene di strumenti.	Prima dell'uso assicurarsi che agli strumenti di build vengano applicati tutti gli aggiornamenti di sicurezza. Questo serve a correggere i problemi della catena di strumenti che potrebbero provocare comportamenti indesiderati nei binari di output
28	Utilizzare le caratteristiche di linguaggio e della catena di strumenti che aiutano a identificare o ridurre i problemi semplici.	Abilitando avvisi di runtime o compiler è possibile identificare preventivamente problemi che possono causare bug o punti deboli di sicurezza (ad esempio - fstack-protector nel GCC avvertirà sulle potenziali condizioni di buffer overflow, oppure /GS oppure

		/SafeSEH in VisualC++ renderà difficili da sfruttare gli stack-based buffer overrun)
29	Utilizzare caratteristiche del sistema operativo che riducono i vettori di attacco più comuni.	Questo è specifico del sistema operativo, ma bisognerebbe abilitare le funzioni di sicurezza disponibili per fornire protezione alla vostra applicazione, ad esempio ASLR e DEP. È disponibile il supporto dei sistemi operativi basati su Microsoft. Per Linux, ASLR e DEP di solito sono abilitati di default (per i kernel dopo 2.6.12). SELinux rappresenta un modo di applicare politiche di controllo degli accessi e può essere utilizzato per ridurre l'impatto di un exploit. Redhat ha una buona guida sulle politiche di sviluppo.
30	Sanitizzare tutti gli input secondo l'architettura di sicurezza prima dell'elaborazione. Convertire gli input in una forma canonica prima della sanitizzazione.	Se gli input degli utenti sono utilizzati come base per comandi o operazioni di database allora possono essere utilizzati per provocare operazioni indesiderate ad es. SQL injection. Gli input possono spesso assumere forme insolite, ad esempio oggetti Java serializzati quindi è importante considerare tutti i casi in cui un utente/intermediario potrebbe essere in grado di modificare o iniettare contenuto. Gli input che non sono convertiti possono bypassare la sanitizzazione utilizzando dei set di caratteri non gestiti dalla sanitizzazione.
31	Sanitizzare tutti gli output verso altri sistemi secondo l'architettura di sicurezza.	Laddove l'input dell'utente viene utilizzato come output (sia verso un'altra applicazione o indietro verso l'utente), allora l'input può essere manipolato in modo da causare un comportamento indesiderato in qualsiasi cosa sta ricevendo l'output (ad es. cross-site scripting).
32	Implementare la crittografia secondo i controlli nella sezione seguente crittografia.	La crittografia può essere complessa e difficile da attuare correttamente. Seguendo gli opportuni controlli si può essere sicuri che il livello della crittografia che si sta

		utilizzando sia adeguato alle informazioni che si stanno proteggendo.
33	Implementare meccanismi per evitare operazioni di memorizzazione non sicure. Vedi OWASP	Alcune operazioni di memorizzazione possono far sì che un utente senza privilegio possa manipolare il contenuto della memoria. In alcuni contesti, questo può causare la manipolazione del flusso di programma (ad es. controlli di bypass) o l'esecuzione di un codice arbitrario.
34	Utilizzare strumenti anti-manomissione dove indicato dalla revisione dei rischi.	Consentire la modifica del codice dell'applicazione o del firmware può bypassare le funzioni di sicurezza. Ciò vale per i dispositivi mobili se non si vuole che la vostra applicazione funzioni più su dispositivi jailbroken o rooted. Si consiglia di fare ciò per far sì che per gli utenti sia più difficile eseguire analisi di runtime/debug per invertire la funzionalità dell'applicazione engineer.
35	Utilizzare tecniche anti-reversing dove indicato dalla revisione dei rischi.	Il processo di reverse engineering può rivelare le funzioni di sicurezza. Tuttavia, è necessario ricordare che la maggior parte delle tecniche anti-reversing possono solo aumentare il tempo necessario per il processo di reverse engineering ma non impedirlo del tutto, quindi non dovrebbe essere considerata l'unica forma di difesa della sicurezza.
36	Evitare i file temporanei. Non utilizzare tmpnam() o simili per generare i nomi dei file.	L'utilizzo di file temporanei può far sì che i dati siano ampiamente distribuiti in un disco e possono essere recuperabili in caso di accesso non autorizzato al sistema. Evitare tmpnam() perché tra il momento in cui viene generato il nome e l'apertura del file è possibile che un altro processo abbia creato un file con lo stesso nome utilizzando tmpnam. Ciò potrebbe far sì che l'altro processo influenzi l'integrità dei dati archiviati o leggere dati cui non dovrebbe avere accesso.
37	Non utilizzare password predefinite (hard-coded).	Le password che sono predefinite nell'applicazione di solito sono ben note e condivise tra gli utenti

		consentendo l'accesso agli utenti non autorizzati o rendendo difficile dimostrare chi ha avuto accesso in quanto non sono assegnate a un singolo utente. Inoltre, non è possibile aggiornare e gestire una password hard-coded in conformità con le specifiche di BT per la gestione degli account.
38	Implementare meccanismi per prevenire l'accesso non autorizzato ai dati in memoria.	A seconda della piattaforma e della revisione dei rischi, potrebbe essere necessario prendere provvedimenti per prevenire che qualcuno che ha accesso al computer che esegue l'applicazione possa recuperare dati sensibili dalla memoria. Questo può avvenire: • sovrascrivendo le porzioni di memoria una volta che non sono più necessarie • bloccando le pagine in memoria (pinning) • utilizzando framework progettati per proteggere le informazioni, ad esempio la classe SecureString in .Net
39	Le segnalazioni di errori devono contenere informazioni sufficienti per identificare la causa dei problemi. Dove gli errori vengono mostrati agli utenti sebbene questi non debbano rivelare informazioni sugli elementi interni dell'applicazione. Ad esempio, non utilizzare tipi di eccezione generici per fornire condizioni di errore specifiche.	Segnalazioni di errori incomplete possono vanificare le indagini sui problemi e nascondere attività non autorizzate. Un exploit genera spesso errori nelle fasi iniziali di accesso che, se registrati correttamente, possono essere utili quando si effettuano indagini sul problema. È difficile capire che cosa abbia causato la condizione di errore se viene registrata solo un'eccezione generica. Se la condizione di errore è stata causata da un attacco è più difficile dedurre quali operazioni erano in corso.
40	Tutti i software devono essere testati prima della migrazione nell'ambiente live. Adottare il programma di prove creato	Software non testati potrebbero: Introdurre rischi per la sicurezza inaccettabili Non funzionare in conformità coi requisiti, in particolare i requisiti di sicurezza descritti nella

	come parte della progettazione dell'applicazione.	Registrazione di sicurezza Influenzare negativamente le operazioni in essere.
41	Tutti i controlli di sicurezza devono essere specificamente testati per assicurarsi che non possano essere elusi.	Vulnerabilità non identificate nei controlli di sicurezza potrebbero essere sfruttate nell'ambiente live.
42	I dati delle prove devono essere cancellati dopo un periodo determinato dal proprietario dei dati.	La divulgazione dei dati delle prove potrebbe fornire informazioni sul sistema donatore.

4. Assicurazione del sistema

Rif.	Controllo	Motivo
43	Le vulnerabilità di sicurezza individuate all'interno del codice o di qualsiasi altra componente usata dal codice saranno classificate in base a criteri di valutazione delle vulnerabilità di BT. Le vulnerabilità devono poi essere corrette secondo la tempistica associata a ciascuna categoria. Ciò si può regolare in base ai rischi posti dalla piattaforma in cui è l'applicazione.	Le vulnerabilità non corrette possono essere utilizzate come parte di un attacco all'applicazione o sistema.
44	Tracking e tagging delle modifiche per correggere i problemi di sicurezza in modo esplicito nel controllo delle modifiche.	Tenere traccia delle correzioni in materia di sicurezza vi consentirà di verificare rapidamente la vostra applicazione/ambiente e dimostrare che sono state apportate.

45	Installare versioni dell'applicazione che sono state costruite dalla sorgente usando l'integrazione continua.	L'implementazione da un ambiente di build pulito fa sì che l'applicazione inizi la sua vita operativa partendo da una serie nota di codice e sia possibile risalire al codice sorgente che ha introdotto i problemi.
46	Assicurarsi che si utilizzi l'ultima versione dell'applicazione e delle eventuali componenti di terzi su cui si basa.	Aggiornando i componenti di terzi, si riduce il rischio di un problema di sicurezza in una componente sfruttata.
47	Applicare patch all'ambiente che ospita l'applicazione in conformità con la politica di patching.	È importante applicare le patch di sicurezza regolarmente poiché versioni obsolete dei software spesso diventano bersaglio di un exploit. Un problema di sicurezza nell'ambiente di hosting potrebbe mettere a rischio i dati dell'applicazione.
48	Definire e documentare le seguenti aree di processo: Applicazione di patch di sicurezza Controllo degli accessi Avvio/Arresto Sincronizzazione dell'orologio, Gestione del lavoro e dell'applicazione Trasferimento di dati Monitoraggio e allarmi Gestione dei problemi e dell'escalation Backup e ripristino Archiviazione Migrazione Controllo delle modifiche Disaster recovery/fallback Manutenzione di hardware & software	La mancanza di procedure operative documentate spesso significa che i processi importanti vengono trascurati oppure che il team non sa come eseguirli.

	Log, revisione e azione su casi eccezionali.	

5. Allegati:

5.1.Controllo degli accessi

Rif.	Politica	Motivo
49	Definire i diritti di accesso del sistema per gli utenti e altri sistemi che interagiscono con esso. I diritti di accesso devono essere basati su: Operazioni da effettuare da parte del sistema Interazione tra sistemi Accesso utente alla politica di informazione e dei sistemi Gli utenti devono avere i privilegi minimi per eseguire il loro lavoro.	Una definizione dei ruoli insufficiente potrebbe consentire l'esecuzione di operazioni accidentali o dannose sul sistema.
50	Account condivisi privilegiati* devono essere gestiti come definito.	In mancanza di un controllo formale è difficile rintracciare i responsabili di modifiche che hanno un impatto sulla sicurezza.

51	Progettare processi applicativi che funzionino con i diritti di accesso minimi richiesti per un corretto funzionamento. Operazioni da effettuare da parte del sistema Interazione tra sistemi Non utilizzare account con privilegio per i processi applicativi Documentare tutti gli accessi privilegiati La Privilege escalation deve utilizzare solo le API note La Privilege escalation deve essere solo per il tempo minimo necessario.	I processi applicativi inesecuzione con privilegi eccessivi potrebbero essere sfruttati per accedere ai dati o privilegi di sistema.
52	Le sessioni utente devono essere terminate dopo un massimo di 30 minuti di inattività. In caso di timeout, tutte le informazioni visualizzate sullo schermo devono essere cancellate.	I timeout limitano gli accessi non autorizzati se il sistema resta incustodito.
53	Una sessione utente non deve superare le 12 ore.	Per assicurarsi che gli utenti accedano e si autentichino nuovamente almeno ogni 12 ore.
54	In tutte le porte di gestione, sia locali (terminale console/Craft o server del terminale) sia remote (via EMS), devono essere effettuati controlli degli accessi in base ai privilegi e ai ruoli.	Le interfacce di gestione esposte possono essere sfruttate per accedere senza autorizzazione.

	Applicare una ACL su tutte le interfacce di gestione (IP), limitando la porta e l'indirizzo di origine e il protocollo invocato; in particolare, la possibilità di limitare l'accesso a ICMP, HTTP, SNMP, FTP, TFTP, SSH, Telnet e protocolli di routing avanzati quali OSPF.	
55	Utilizzare solo protocolli di gestione autenticati in modo sicuro ad esempio SNMP v3 (AuthnNoPriv come minimo) SSHv2 - vedere la sezione sulla crittografia HTTPS - vedere la sezione sulla crittografia.	I protocolli di gestione non sicuri possono essere sfruttati per accedere senza autorizzazione.

5.2 Crittografia

Rif.	Politica	Motivo
56	Utilizzare le librerie crittografiche attuali.	Le librerie crittografiche vengono aggiornate periodicamente. Oltre ad aggiornare i pacchetti software secondo le indicazioni del vendor, revisionare e aggiornare periodicamente i pacchetti crittografici.
57	Usare solo suite di cifre standard del settore approvate per la crittografia. Ad es. per TLS SSLv2.	Cifre non approvate possono introdurre vulnerabilità.

	Si noti l'avviso di cui sopra. In caso di dubbio, consultare l'ITSAC.	
58	Per nuove implementazioni, utilizzare l'ultima versione di TLS. Non utilizzare SSL V1, 2 e 3.	Versioni precedenti, fino a TLS1.0 compreso, non sono più considerate sicure.
59	Consentire il Perfect Forward Secrecy.	Gli algoritmi di Perfect Forward Secrecy impediscono che i messaggi catturati vengano decriptati anche se la chiave di autenticazione privata viene compromessa in futuro.
60	Non utilizzare certificati autofirmati.	I certificati autofirmati annullano il vantaggio dell'autenticazione dell'endpoint e diminuiscono in modo significativo la possibilità di rilevare un attacco Man In The Middle.
61	Le password devono essere protette utilizzando una funzione matematica non reversibile unidirezionale (ad es. l'algoritmo di Hashing) con un unico fattore di randomizzazione (sale) come password.	I file di password archiviati possono essere estratti ed è necessario proteggere tutte queste entry per impedire che le password come testo in chiaro vengano recuperate.
62	Le password protette devono essere archiviate lontane dai file di configurazione di sistema ed è necessario implementare un controllo degli accessi in modo che solo gli utenti privilegiati autorizzati possano leggere o copiare il contenuto.	Non deve mai essere possibile recuperare password protette mediante Directory Traversal, SNMPwalk, Configuration Dump o altri meccanismi che possono rendere possibili tentativi di Offline Cracking.

5.2.1 Controlli crittografici web

Rif.	Politica	Motivo
63	I cookie che archiviano o vengono utilizzati per accedere a dati Riservati o di livello superiore devono essere trasportati in modo sicuro.	Vi sono molti metodi per rubare i cookie come l'XSS e lo sniffing.
64	Contenuti protetti e non protetti non deve essere mischiati nella stessa pagina.	I contenuti non sicuri potrebbero potenzialmente rubare informazioni sicure dal contenuto.
65	Utilizzare TLS per tutte le pagine di login e tutte le pagine autenticate. La pagina di login e tutte le pagine successive autenticate devono essere accessibili esclusivamente tramite TLS. La pagina di login e tutte le pagine successive autenticate devono essere accessibili esclusivamente tramite TLS.	Il mancato utilizzo di TLS per la landing page di login consente a un utente malintenzionato di modificare l'azione del modulo di login e pubblicare le credenziali dell'utente in una posizione arbitraria. Il mancato utilizzo di TLS per le pagine autenticate dopo il login consente a un utente malintenzionato di visualizzare l'ID di sessione non criptato e compromettere la sessione autenticata dell'utente. Il mancato utilizzo di TLS può provocare un attacco Man In The Middle.
66	I clienti devono essere istruiti a non memorizzare in cache i dati sensibili.	Il protocollo TLS fornisce riservatezza solo per i dati in transito ma non risolve eventuali potenziali problemi di perdita di dati nel client.
67	Utilizzare gli standard di firma digitale accettati a livello nazionale e internazionale.	L'uso di tecniche di firma digitale non standard può danneggiare la fiducia nella firma; le normative nazionali e internazionali prevedono controlli sul digitale che riassumono i controlli mondiali sull'importazione, sull'esportazione e sulla crittografia nazionale.
68	Non utilizzare certificati Wildcard, in nessun caso.	I certificati Wildcard sono un bersaglio più "attraente"; possono certificare un server indesiderato e rendere vulnerabile un dominio di crittografia.

		Se un server o un sottodominio è compromesso, tutti i sottodomini possono essere compromessi. Se il certificato Wildcard deve essere revocato, servirà un nuovo certificato per tutti i sottodomini.

5.2.2 Gestione generale delle chiavi

Rif.	Politica	Motivo
69	Le chiavi di crittografia per tutte le nuove implementazioni devono soddisfare o superare gli standard minimi.	Chiavi corte o deboli possono essere facilmente compromesse durante il ciclo di vita dei dati.
70	Chiavi simmetriche e asimmetriche devono essere generate utilizzando strumenti e librerie approvati.	Strumenti e librerie non approvati possono potenzialmente generare chiavi deboli.
71	Le password che controllano l'uso di chiavi crittografiche devono fornire una protezione che è equivalente alla chiave stessa.	Le password deboli indeboliscono la chiave crittografica.
72	Un alto dirigente all'interno di BT deve essere responsabile per la sicurezza del materiale sulle chiavi.	Avrà l'anzianità, la capacità e l'affidabilità commisurate alla sicurezza dei dati che le chiavi proteggeranno.
73	L'alto dirigente incaricato delle chiavi deve assicurarsi che le responsabilità nella colonna dettagli siano assegnate ed eseguite da personale adeguatamente qualificato e addestrato:	Responsabilità Politica di gestione delle chiavi Generazione o acquisizione di chiavi

		Progettazione della distribuzione delle chiavi, della durata delle chiavi, della revoca e della struttura di accounting Creazione di procedure di gestione delle chiavi Funzionamento della gestione delle chiavi Protezione delle chiavi private e dei relativi materiali Procedure di emergenza, come la revoca Revisione delle operazioni sulle chiavi Recupero delle chiavi Distruzione delle chiavi

Controllo dei documenti

Standard guida del settore di terzi sulla "codifica sicura"

Autore: BT Security

Edizione 1, pubblicata ad ottobre 2016