semplice, o scomponendolo in componenti molto semplici e totalmente separati. Una volta limitate le vie d'accesso, è necessario iniziare a pensare a "Quanti attacchi diversi sono possibili contro ciascuna via d'accesso? Anche in questo caso, è opportuno limitare il tutto rendendo le vie di per sé piuttosto semplici. Come una porta con una chiave unica, invece di una porta che può essere aperta con cinque chiavi diverse. Una volta fatto ciò, è necessario contenere i possibili danni che un attacco potrebbe arrecare se portato con successo. Ritornando all'esempio dell'edificio, dovremmo fare in modo che una singola porta consenta l'accesso ad una sola stanza. Il fatto di aumentare la complessità del codice determina con buona probabilità una crescita del livello di entropia e di conseguenza un proporzionale incremento dell'esposizione del software a possibili vulnerabilità. La complessità rende la sicurezza facile da ignorare e quindi aumenta la probabilità che questa possa fallire. All'atto pratico:

- Riutilizzare quei componenti di cui è nota l'affidabilità.
- Evitare architetture complesse e codifiche che presentano l'uso di doppi negativi, es: un costrutto condizionale del tipo "if(!item.isNotFound())" dovrebbe essere ricondotto nella sua forma semplice "if(item.isFound())".

13) Risolvere nel modo corretto le problematiche di sicurezza.
Una volta che un problema di sicurezza viene identificato, è importante attuare un test e comprendere la causa che origina il problema al livello di massima 'profondità'. Quando si utilizzano schemi predefiniti di progettazione, è probabile che eventuali problematiche di sicurezza presenti su un modulo vengano diffuse in tutte le baseline di codice, per cui è essenziale sviluppare la giusta risoluzione senza introdurre regressioni. Per esempio, un utente ha scoperto di poter vedere le informazioni di un altro utente manipolando il proprio cookie. La correzione sembra essere relativamente semplice, ma poiché il codice di gestione dei cookie è condiviso tra tutte le applicazioni, una modifica ad una sola applicazione può essere estesa a tutte le altre. La correzione, ed il contesto su cui opera (una correzione può avere effetti diversi su implementazioni differenti) deve quindi essere verificata su tutte le applicazioni interessate.

5.2.2 Pratiche di secure design

5.2.2.1 Best practice di secure design per le applicazioni web

Il seguente elenco definisce, a titolo esemplificativo e non esaustivo, un insieme di buone pratiche che, se adottate in fase di design, consentono di risolvere a priori vulnerabilità applicative note considerate maggiormente critiche.

- Gestione degli errori e attività di logging
  - Mostrare all'utente finale messaggi di errore di carattere generico.
    - Descrizione: I messaggi di errore non devono mai rivelare dettagli sullo stato interno dell'applicazione. Ad esempio, percorsi del file system e informazioni sullo stack non devono essere mai esposte all'utente attraverso messaggi di errore. (Vedi CWE 209).
  - Gestire tutte le eccezioni nel codice sorgente.
    - Descrizione: Non consentire mai che si verifichi un'eccezione non gestita attraverso i linguaggi e i framework impiegati nello sviluppo di applicazioni web. La gestione degli errori deve essere implementata in modo tale da comprendere anche gli errori imprevisti restituendo l'output all'utente sempre in modo controllato. (Vedi CWE 391).
  - Nascondere, bloccare o gestire gli errori generati da eventuali framework o software di terze parti in uso.
    - Descrizione: L'uso di un eventuale framework o piattaforma di sviluppo può generare messaggi di errore predefiniti. Questi dovrebbero essere bloccati o