ospita l'applicazione. In tal caso l'attaccante potrebbe anche utilizzare il server come piattaforma per ulteriori attacchi contro altri sistemi.
Il pericolo si manifesta quando un malintenzionato riesce ad eseguire codice arbitrario nell'host dell'application server. Si potrebbero avere le seguenti problematiche:

- Possibilità di modificare i permessi all'interno di file o directory nel file system(read / create / modify / delete);
- Modifiche della struttura del sito web;
- Permettere delle connessioni di rete non autorizzate verso il server da parte dell'attaccante;
- Permettere ad utenti malintenzionati la gestione dei servizi con possibili start and stop dei servizi di sistema;
- Acquisizione completa del server da parte dell'attaccante.

**Come difendersi**

Ove possibile, le applicazioni dovrebbero evitare di incorporare dati controllabili dall'utente per acquisire codice che verrà eseguito dinamicamente. In quasi ogni situazione esistono metodi alternativi più sicuri per l'implementazione di funzioni applicative che non siano manipolabili per iniettare codice arbitrario.
Se si ritiene inevitabile integrare i dati forniti dall'utente nel codice eseguito dinamicamente, i dati devono essere validati rigorosamente. Idealmente, dovrebbe essere utilizzata una white list di specifici valori accettati. Altrimenti, dovrebbero essere accettate solo stringhe alfanumeriche brevi. Gli input contenenti altri dati, inclusi eventuali metacaratteri di codice eseguibile, devono essere respinti.
L'uso di exec() ed eval() va evitato per la possibilità di incorrere in una code injection.

Esempio:
Il seguente esempio mostra due funzioni che impostano un nome a paritre da una request. La prima funzione utilizza exec per eseguire la funzione setname. Ciò è pericoloso in quanto un malintenzionato potrebbe approfittarne per eseguire codice arbitrario sul server.
Ad esempio, potrebbe fornire il valore "'+ subprocess.call (' rm -rf ') +'", che distruggerebbe il file system del server.
La seconda funzione chiama direttamente la funzione setname e il parametro fornito dall'utente viene utilizzato come dato. Nessun codice potrebbe qui essere eseguito.

```
def esecuzione_codice_non_sicura(request):
  if request.method == 'POST':
    nome = base64.decodestring(request.POST.get('nomè, ''))
    #NON SICURO - Permette all'utente di eseguire del codice arbitrario.
    exec("setname('%s')" % nome)

def esecuzione_codice_sicura(request):
  if request.method == 'POST':
    nome = base64.decodestring(request.POST.get('nomè, ''))
    #SICURO - Il parametro utente solo un valore che non verrà eseguito.
    setname(nome)
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/94.html,
Improper Control of Generation of Code ('Code Injection') CWE-94

### 7.5.3 Command Injection

**Come riconoscerla**

Si è in presenza di un attacco di command injection, noto anche come OS injection, quando l'input utente non verificato viene utilizzato, in tutto o in parte, come argomento di funzioni che eseguono comandi di shell. Tramite questa vulnerabilità un aggressore potrebbe eseguire comandi di sistema operativo arbitrari sull'host dell'application server. In base alle autorizzazioni dell'applicazione, potrebbe: