inoculato ed eseguito nel periodo in cui dura la sessione. Gli XSS stored, viceversa, sono script malevoli che sono stati memorizzati su una base dati e vengono pertanto incorporati nella pagina (e quindi eseguiti) ogni volta che qualcuno ne fa richiesta.
Siamo di fronte ad DOM based XSS se i dati malevoli, contenenti tag HTML e script, vengono incorporati direttamente nell'HTML della pagina, in modo che il browser visualizzerà queste informazioni come parte della pagina web eseguendo in maniera silente gli script. Chi visualizza la pagina modificata in modo fraudolento non sarà in grado di riconoscere l'inganno.

Come difendersi

- Per prima cosa è necessatio convalidare tutti gli input, indipendentemente dalla fonte: la convalidazione dovrebbe essere basata su una white list (una lista di valori ammessi), per cui verrebbero accettati solo i dati compresi e rifiutati tutti gli altri.
- Oltre a controllare che i valori siano compresi fra quelli ammessi o che rientrino in un determinato intervallo di validità, occorre verificare che corrispondano alle attese anche il tipo, la dimensione e il formato dei dati in input.
- Un altro accorgimento consiste nell'encoding di tutti i dati dinamici, cioè nella neutralizzazione dei caratteri pericolosi, in modo da rendere inattivi eventuali inserimenti malevoli. La codifica dovrebbe essere sensibile al contesto, in base al tipo di dato che si vuole neutralizzare: se ci si aspetta che possa esserci codice HTML abusivo, occorre codificare gli eventuali tag HTML, se ci si potrebbe trovare di fronte a uno script, allora bisogna codificare gli elementi sintattici di Javascript, ecc.
- È opportuno attivare lo standard Content Security Policy (CSP) in modo che solo le risorse scaricate da fonti attendibili possano essere utilizzate. Impostare l'attributo HTTPOnly a true per impedire il furto dei cookie.
- La maggior parte dei template di Python oggi fanno l'escaping dell'input, anche se questa funzione può essere disattivata. Il modulo flask fa l'escaping dell'HTML.

Esempio:
Codice non corretto:

```
@app.route("/")
def hello():
  name = request.args.get('namè)
  return "Hello %s" % name
```

Codice corretto:

```
from flask import escape
@app.route("/")
def hello():
  name = request.args.get('namè)
  return "Hello %s" % escape(name)
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/79.html,
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').

7.5.2 Code Injection

Come riconoscerla

Le vulnerabilità legate all'iniezione di codice sul lato server sorgono quando un'applicazione incorpora dati controllabili da parte dell'utente in una stringa che viene valutata dinamicamente da un interprete di codice. Se i dati dell'utente non sono rigorosamente convalidati, un malintenzionato può utilizzare l'input per iniettare codice arbitrario che verrà eseguito dal server.
Le vulnerabilità legate all'iniezione di codice sul lato server sono in genere molto gravi e portano a una completa compromissione dei dati, delle funzionalità dell'applicazione e spesso persino del server che