Il Cross Site Scripting può essere reflected o stored. Nel primo caso, uno script inoculato è valido solo all'interno della sessione corrente, ma i suoi effetti possono essere molto dannosi per il sito vittima dell'attacco.
Ancora più grave è il Cross Site Scripting di tipo stored. In questo caso lo script inoculato viene memorizzato come parte integrante della pagina all'interno di un database e rispristinato ogni qual volta la pagina in questione viene caricata. Quest'attacco è stato sfruttato ampiamente nel recente passato, soprattutto laddove veniva consentito agli utenti di inserire recensioni, commenti e altri contributi.
Volendo schematizzare, possiamo categorizzare gli attacchi XSS nelle seguenti tipologie:

- Reflected XSS, in cui la stringa dannosa proviene dalla richiesta dell'utente.
- Stored XSS, in cui la stringa dannosa proviene dal database del sito Web.

Esiste anche un Cross Site Scripting dovuto a una lacuna nella codifica UTF-7, che permettere di mascherare i caratteri "<" e ">", facendoli sfuggire al controllo. Questa minaccia non è più possibile nei moderni browser, ad eccezione di Microsoft Internet Explorer 11.

Come difendersi

- Validare tutti gli input, indipendentemente dalla loro provenienza. Per la validazione, si consiglia l'approccio white list (sono accettati solo i dati che adottano una struttura specificata nella white list, scartando quelli che non la rispettano). Occorre controllare il tipo del dato, la sua dimensione, l'intervallo di validità (range), il formato ed eventuali valori attesi (white list).
- Codificare completamente tutti i dati dinamici prima di incorporarli (encoding). 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.
- Si consiglia di utilizzare la libreria di codifica messa a disposizione da PHP. Fra le varie funzioni sono da ricordare: htmlspecialchars(), htmlentities(), strip_tags() e add_slashes(). Le prime sono utilizzate per l'escaping di codice HTML, l'ultima per bonificare codice Javascript.
- Nell'intestazione di risposta Content-Type HTTP, è necessario definire esplicitamente la codifica dei caratteri (charset) per l'intera pagina.
- Impostare l'attributo HTTPOnly per proteggere il cookie della sessione da indebite letture da parte di script malevoli.

Esempio:
Il dcodice che segue prende in input una variabile utente e la stampa a video:

```
echo $_POST["name"];
```

Se l'utente, invece di inserire il proprio nome, inserisce del codice attivo, per esempio "<script>alert('Attacco XSS!')</script>", crea un caso di attacco XSS.
Per bonificare il codice, la stringa accettata deve essere controllata e depurata dei caratteri dannosi:

```
echo htmlentities($_POST["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').

7.9.2 Code Injection

Come riconoscerla

Un utente malintenzionato potrebbe eseguire codice arbitrario nell'host del server di applicazioni. A seconda delle autorizzazioni dell'applicazione che potrebbero essere carpite, 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

Se possibile, preferite sempre delle white list con valori prefissati. Evitare qualsiasi compilazione dinamica, esecuzione o valutazione del codice. Se è necessario eseguire tutto il codice dinamico in una sandbox isolata, ad esempio AppDomain di .NET o bloccare un thread isolato.
L'applicazione non deve compilare, eseguire o valutare i dati non attendibili, in particolare eventuale input dell'utente. Validare tutti gli input, indipendentemente dalla loro provenienza. La convalida dovrebbe essere basata su una white list: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, e scartati i dati che non rientrano in questa categoria. I parametri devono essere limitati a un set di caratteri consentito e i caratteri riconosciuti come estranei devono essere filtrati e neutralizzati (escaping). Oltre ai caratteri, occorre controllare il tipo del dato, la sua dimensione, l'intervallo di validità (range), il formato ed eventuali valori attesi (white list).
Nel caso fosse assolutamente necessario includere i dati di input in un'esecuzione dinamica, applicare una validazione dell'input molto rigida. Ad esempio, accettare solo interi tra determinati valori.
Configurare l'applicazione da eseguire utilizzando un account utente limitato che non disponga di privilegi non necessari.
L'esecuzione del codice dovrebbe utilizzare un account utente separato e dedicato, fornito dei soli privilegi strettamente necessari, in base al principio denominato "Principle of Least Privilege". Il principio stabilisce che agli utenti venga attribuito il più basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro.

Esempio:
Codice vulnerabile. L'utente può iniettare qualsiasi codice e farlo eseguire:

```
if (isset($_GET['codè'])) {
  $code = $_GET['codè];
  eval($code);
}
```

Codice sicuro. La scelta è possibile all'interno di una white list di funzioni statiche:

```
$method = $_GET[method];
switch ($method) {
 case "methodOne":
  methodOne();
  break;
 case "methodTwo":
  methodTwo();
  break;
 //...
}
```

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