- Se è assolutamente necessario includere dati esterni nell'esecuzione dinamica, è consentito passare i dati come parametri al codice, ma non eseguire direttamente i dati utente.
- Se è necessario passare dati non attendibili all'esecuzione dinamica, applicare una convalida dei dati molto rigorosa. Come al solito, occorre convalidare tutti gli input, indipendentemente dalla fonte. I parametri devono essere limitati a un set di caratteri consentito e l'input non convalidato deve essere eliminato. Oltre ai caratteri, occorre controllare il tipo di dati, la loro dimensione, l'intervallo di validità, il formato e l'eventuale corrispondenza all'interno dei valori previsti (white list). Sconsigliata invece la black list, ossia un elenco di valori non consentiti: l'elenco sarebbe sempre troppo limitato, rispetto ai casi che potrebbero verificarsi.
- L'account con il quale l'applicazione viene avviata deve avere molte restrizioni e non deve godere di privilegi non necessari.

Esempio:
codice vulnerabile:

```
eval (location.hash);
```

La funzione eval esegue dinamicamente del codice. Va evitata.
Il seguente codice è invece ragionevolmente sicuro, poiché manda in esecuzione una funzione staticamente codificata:

```
window.setTimeout(funzioneCodificata(), 1000);
```

Per maggiori informazioni vedere http://cwe.mitre.org/data/definitions/94.html

7.4.3 Client DOM Stored Code Injection

Come riconoscerla

Un malintenzionato potrebbe causare l'esecuzione di contenuti ingegnerizzati nel browser, riscrivendo le pagine Web e inserendo script dannosi. L'utente legittimo sarebbe quindi indotto a fidarsi di ciò che gli viene proposto. Ciò permetterebbe all'attaccante di rubare la password dell'utente, richiedere informazioni sulla sua carta di credito, fornire informazioni false o eseguire malware. La vittima continuerebbe la sua attività ignara del pericolo, salvo poi accusare i responsabili del sito per i danni subiti.
La pagina web dell'applicazione esegue alcune azioni eseguendo codice sul lato client, concatenando dati di input da una cache sul lato client, come un cookie, la LocalStorage dell'HTML5 o un database locale. Codice dannoso eventualmente presente nei dati potrebbe avviare attività progettate da un attaccante.

Come difendersi
Occorre evitare qualsiasi esecuzione dinamica del codice. Se è proprio necessaria, anziché utilizzare i dati sul lato client, inclusi i dati precedentemente memorizzati nella cache dalla stessa applicazione, utilizzare solo dati attendibili provenienti dal server.

Per maggiori informazioni vedere: http://cwe.mitre.org/data/definitions/94.html

7.4.4 Client DOM Stored XSS

Come riconoscerla

Un malintenzionato può utilizzare l'accesso legittimo all'applicazione per inviare dati ingegnerizzati al database dell'applicazione. Quando un altro utente accede in seguito, le pagine Web potrebbero essere riscritte con i dati salvati e potrebbero essere attivati script dannosi.
L'applicazione crea pagine web che includono dati provenienti dal database, incorporati direttamente nell'HTML della pagina. Il browser, quindi, li visualizza come parte della pagina.
Il problema nasce quando questi dati salvati sono stati immessi da un altro utente. Se i dati includono frammenti HTML o Javascript malevoli, anche questi vengono visualizzati (o eseguiti), sebbene la vittima non si accorga dell'inganno sottostante. La vulnerabilità è perciò il risultato dell'incorporazione di dati

arbitrari provenienti dal database, senza prima codificarli. La codifica trasforma i caratteri malevoli in normale testo, e il browser non può più trattarli come codice valido HTML/Javascript.

Come difendersi

- I parametri devono essere limitati a un set di caratteri consentito e l'input non convalidato deve essere eliminato. Oltre ai caratteri, occorre controllare il tipo di dati, la loro dimensione, l'intervallo di validità, il formato e l'eventuale corrispondenza all'interno dei valori previsti (white list). Sconsigliata invece la black list, ossia una lista di valori non consentiti: l'elenco sarebbe sempre troppo limitato, rispetto ai casi che potrebbero verificarsi.
- La convalida non sostituisce la codifica (encoding), ossia la neutralizzazione di tutti i caratteri potenzialmente eseguibili. Tutti i dati dinamici, indipendentemente dall'origine, devono essere codificati prima di incorporarli nell'output. 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 per Javascript, ecc.
- L'account con il quale l'applicazione viene avviata deve avere molte restrizioni e non deve godere di privilegi non necessari.
- Nell'intestazione della risposta HTTP Content-Type, definire esplicitamente la codifica dei caratteri (set di caratteri) per l'intera pagina.
- Impostare il flag httpOnly sul cookie di sessione, per impedire agli exploit XSS di rubarlo.

Esempio:
La funzione Javascript che segue utilizza dati del database, senza verificarli, per creare dinamicamente uno script:

```
function renderUserProfileTable(res, connection, user_id) {
  connection.query('SELECT id,name,description from user WHERE id= ?',
[user_id],function(err, results) {
          var table = "<table>"
          table += "<table class='profile-html-tablè>"
          table += "<tr><td>" + results[0].name + "</td></tr>"
          table += "<tr><td>" + results[0].description + "</td></tr>"
          table += "</table>"
          res.render("profile", table)
  });
}
```

Qui di seguito la funzione viene bonificata tramite l'encoding dei valori letti dal database, effettuato prima di incorporarli nella pagina web:

```
var htmlencoder = require('htmlencodè);

function renderUserProfileTable(res, connection, user_id) {
connection.query('SELECT id,name,description from user WHERE id= ?',
[user_id],function(err, results) {
          var table = "<table>"
          table += "<table class='profile-html-tablè>"
          table += "<tr><td>" + htmlencoder.htmlEncode(results[0].name) +
"</td></tr>"
          table += "<tr><td>" + htmlencoder.htmlEncode(results[0].description) +
"</td></tr>"
          table += "</table>"
          res.render("profile", table)
  });
}
```

Per maggiori informazioni vedere: http://cwe.mitre.org/data/definitions/79.html