Se necessario ricevere XML dall'utente, assicurarsi che il parser XML sia limitato e vincolato. In particolare, disabilitare l'analisi e la risoluzione delle entità DTD, applicare uno schema XML rigoroso sul server e convalidare l'XML in input di conseguenza.
Poiché tutte le funzionalità di analisi XML offerte da PHP si basano sulle librerie libxml, esiste una funzione che impedisce il caricamento di queste entità:

```
<?php libxml_disable_entity_loader(true); ?>
```

La funzione libxml_disable_entity_loader indica al parser di non tentare di interpretare i valori delle entità nell'XML in entrata e di lasciarne intatti i riferimenti. Se si sta usando SimpleXML, questa è davvero l'unica scelta per prevenire un attacco XXE nell'XML in arrivo.
Fortunatamente, gli altri due metodi di analisi XML offrono alcune funzionalità che possono essere utili a proteggere l'applicazione, pur consentendo l'espansione delle entità XML.

```
loadXML($badXml,LIBXML_DTDLOAD|LIBXML_DTDATTR); ?>
```

In entrambi i casi, stiamo aggiungendo alcuni valori costanti predefiniti (per nome) che indicano al parser di non consentire una connessione di rete durante il caricamento (LIBXML_NONET) o di provare ad analizzare l'XML in base al DTD (LIBXML_DTDLOAD|LIBXML_DTDATTR). Entrambi questi metodi contribuiscono alla sicurezza dell'applicazione rispetto ai problemi di XXE.

7.9.12 Unsecure deserialization

Come riconoscerla

La "unsecure deserialization" è una vulnerabilità che si verifica quando un'applicazione utilizza il processo di deserializzazione di dati serializzati non attendibili. Tra la serializzazione da parte del processo originario e la deserializzazione da parte del processo di destinazione, i dati serializzati possono aver subito inserimenti di codice dannoso.
In seguito a deserializzazione di dati inquinati con porzioni di codice malevolo, l'attaccante può infliggere un attacco di denial of service (DoS) o eseguire codice arbitrario.

Come difendersi

Evitare di utilizzare le tecniche di serializzazione/deserializzazione. Se è strettamente necessario utilizzarle, verificare che il dato serializzato non possa essere inquinato e manomesso durante il suo percorso. Ad esempio, garantire la trasmissione attraverso una connessione sicura e criptata.
Controllare l'uso della funzione unserialize() e rivedere come vengono accettati i parametri esterni. Utilizzare un formato di scambio di dati standard sicuro come JSON, tramite json_decode() e json_encode(), se è necessario passare all'utente dati serializzati.

Esempio:
Formato non corretto
Creazione di un utente con una deserializzazioen.

```
//.. JSON Validity Checks ..//
$user_params = json_decode($HTTP_RAW_POST_DATA);
$user = unserialize($user_params);
```

Formato corretto

```
Creazione di un utente senza deserializzazioen
//.. JSON Validity Checks ..//
$user_params = json_decode($HTTP_RAW_POST_DATA);
//.. Parameter Checks ..//
$name = $user_params['Namè];
$email = $user_params['Email'];
$phone = $user_params['Phonè];
$user = new User($name, $email, $phone);
```

Per maggiori informazioni: http://cwe.mitre.org/data/definitions/502.html