- Trust boundaries. Non utilizzare il modulo OS per la manipolazione di file host ricevuti da una fonte non attendibile o controllata dall'utente.
- Comunicazione protetta. Assicurarsi che venga utilizzata una connessione di rete crittografata.
- Validazione. Il path di un file che si vuole manipolare dev'essere validato in modo corretto: evitare che possa essere inserito da un utente in modo dinamico. Assicurarsi, inoltre, che rispecchi completamente delle regole canoniche.
- Sandbox. Limitare l'accesso al percorso dei file all'interno di una directory specifica.
- White list. Creare una white list di file o directory che possono essere manipolati in modo sicuro e consentire l'accesso solo a questi file o directory.

Esempio:
Forma non corretta: l'applicazione riceve un file path dall'utente e rimuove il file stesso:

```
import os
import sys
[…]
path = sys.stdin.readline()[:-1]
os.remove(path)
```

Forma corretta: l'applicazione restringe l'accesso ad un file ad una specifica directory:

```
import os
import sys
def is_safe_path(basedir, path):
        return os.path.abspath(path).startswith(basedir)
        path = sys.stdin.readline()[:-1]
        if not is_safe_path('/tmp/userfiles', path):
        sys.stdout.write('Not allowed!\n')
        sys.exit()
        os.remove(path)
```

7.5.11 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.
- Eliminare, se possibile, dal codice sorgente le seguenti API vulnerabili:
    - Pickle
      Esempio:
      ```
      import pickle
      data = """ cos.system(S'dir')tR. """
      pickle.loads(data)
      ```
    - PyYAML
      Esempio:
      ```
      import yaml
      document = "!!python/object/apply:os.system ['ipconfig']"
      print(yaml.load(document))
      ```
    - Jsonpickle
    - Metodi encode e store