SQL Injection (SQL 注入)(上次有講過，這裡可以簡單帶過)

- 解釋：騙資料庫執行它不該執行的指令。
- 偵測點：檢查有沒有 ' or 1=1 -- 這種萬能鑰匙。
- 攻擊成功後永遠成功登入，造成資料外洩

XSS (跨站腳本攻擊)

- 原理：攻擊者把程式碼當成文字輸入，而後端/前端沒有阻擋。瀏覽器看到 <script> 就會直接執行。
- 簡單解釋：在網頁裡埋藏惡意程式碼，去偷別人的瀏覽器資訊。
- 偵測點：檢查有沒有 <script> 、alert() 或javascript:這種腳本標籤。
- 可能造成：
  偷你的 cookie → 偷登入狀態
  換成假的付款頁面
  自動發送動作（like、轉帳、刪除資料）

Path Traversal (路徑遍歷)

- 解釋：嘗試跳出網站的資料夾，去偷看伺服器系統檔案。偷讀檔案、看不該看的東西。
- 偵測點：檢查有沒有 ../、%2e%2e%2f 或 /etc/passwd 這種路徑符號。
- 沒有阻擋就會造成對方讀到系統帳號檔案、密碼、設定檔.....

Suspicious User-Agent (可疑使用者代理)

- 解釋：識別發出請求的是「正常人用的瀏覽器」還是「駭客用的自動化掃描工具」。
- 偵測點：檢查請求標頭（Header）裡有沒有 sqlmap、curl 或 python 等掃描器特徵。
- 正常人用的是：Chrome Safari Edge Firefox
- 駭客常用：curl python-requests sqlmap Nmap BurpSuite，不用瀏覽器，而是用自動化工具掃漏洞。

Brute Force (暴力破解)(這個放最後測，因為攻擊成功會被鎖1分鐘)

- 解釋：瘋狂猜測帳號密碼，直到成功登入。
- 偵測點：使用時間窗口統計法，程式會計算時間，如果同一 IP 在 60 秒內 POST /login 超過 5 次，就判定為攻擊。

SSRF (伺服器端請求偽造)

- 解釋：駭客利用這台伺服器當跳板，去攻擊別人或查看內網。
- 偵測點：檢查網址是不是指向 127.0.0.1 或內網 IP。
- 可能導致：
  內部服務被探測
  Redis, MySQL 無密碼 → 直接被拿下
  內部 metadata 洩漏（雲端常出現）

# 程式碼


DEFAULT_RULES = {
"SQLI_PATTERNS": [
# 基本 or 1=1 類型
" or 1=1",
"' or '1'='1",
"\" or \"1\"=\"1",
" or '1'='1",
" or 1=1 --",
" or 1=1--",
" or 1=1#",
" or 1=1/*",
" union select",
"--",
";--",
"/*",
"*/",
# time-based / function 類型 SQLi
"sleep(",
"benchmark(",
"pg_sleep(",
"waitfor delay",
],
"XSS_PATTERNS": [
"<script",  # <script>...</script>
"onerror=",  # <img src=x onerror=...>
"onload=",  # onload 事件
"javascript:",  # href="javascript:alert(1)"
"alert(",  # alert(1)
# 更多常見事件 handler
"onclick=",
"onmouseover=",

預設攻擊關鍵字規則

```python
def _load_rules_from_file(filename: str = "rules.json") -> None:

    global RULES, MODE, BRUTE_FORCE_WINDOW_SECONDS, BRUTE_FORCE_THRESHOLD
    #我等一下在這個函式裡面要修改外面的變數 RULES、MODE、BRUTE_FORCE_WINDOW_SECONDS、BRUTE_FORCE_THRESHOLD

    if not os.path.exists(filename):
        # 找不到檔案就維持預設規則與模式
        return

    try:
        with open(filename, "r", encoding="utf-8") as f:#打開檔案，讀完自動關掉。
            data = json.load(f)#把 JSON 內容轉成 Python dict，存在 data 裡。

        for key in DEFAULT_RULES.keys():  # 只更新我們認得的 key，避免亂掉
            if key in data and isinstance(data[key], list):
                RULES[key] = data[key]

        # 從 JSON 調整 MODE（允許 LOG_ONLY 或 BLOCK）
        mode = data.get("MODE")
        if isinstance(mode, str) and mode.upper() in ("LOG_ONLY", "BLOCK"):
            MODE = mode.upper()

        #  （選擇性）也可以讓 JSON 調整暴力登入參數
        if isinstance(data.get("BRUTE_FORCE_WINDOW_SECONDS"), int):
            BRUTE_FORCE_WINDOW_SECONDS = int(data["BRUTE_FORCE_WINDOW_SECONDS"])
        if isinstance(data.get("BRUTE_FORCE_THRESHOLD"), int):
            BRUTE_FORCE_THRESHOLD = int(data["BRUTE_FORCE_THRESHOLD"])

    except Exception:
        RULES = DEFAULT_RULES.copy()
        MODE = "LOG_ONLY" # 有問題就直接忽略，維持預設

# 啟動時就先試著載入一次
_load_rules_from_file()
```

嘗試從 `rules.json` 載入規則與模式。
（如果檔案不存在或格式錯誤，就使用 `DEFAULT_RULES` + 預設 `MODE`。可以直接改 `JSON` 檔就能調整 `WAF` 規則，如果 `JSON` 檔格式錯誤或被誤刪，程式會自動退回 DEFAULT_RULES，避免整個偵測壞掉。）

```python
def _collect_fields(input_data: dict) -> Dict[str, str]:

    pieces: Dict[str, str] = {}

    # URL、方法、User-Agent
    raw_url = _to_str(input_data.get("url", ""))
    decoded_url = unquote(raw_url)  # URL 解碼
    pieces["url"] = decoded_url

    pieces["http_method"] = _to_str(input_data.get("http_method", ""))
    pieces["user_agent"] = _to_str(input_data.get("user_agent", ""))

    # params 可能是 GET query string 的參數
    params = input_data.get("params", {}) or {}
    for k, v in params.items():
        raw = _to_str(v)
        decoded = unquote(raw)      # 參數也先解碼一次
        pieces[f"param.{k}"] = decoded

    # body 是 POST/PUT 的內容
    body = input_data.get("body", {}) or {}
    for k, v in body.items():
        raw = _to_str(v)
        decoded = unquote(raw)      # body 內容也先解碼
        pieces[f"body.{k}"] = decoded

    return pieces
```

把所有請求內容『攤平』並『解碼』, (因為駭客常會用編碼來隱藏惡意指令, 所以透過 unquote 函式進行正規化 (Normalization), 確保系統是針對還原後的真實內容進行檢查, 避免被繞過。)

```python
#在所有欄位找關鍵字
def _find_pattern(pieces: Dict[str, str], patterns: List[str]) -> Tuple[bool, str, str]:
    """
    在所有欄位裡面找有沒有出現任一 pattern。
    回傳：
      (是否命中, 該欄位名稱, 該欄位原始內容)
    沒找到則回傳 (False, "", "")。
    """
    for field_name, raw_value in pieces.items():
        value_lower = raw_value.lower()
        for p in patterns:
            if p and p.lower() in value_lower:
                return True, field_name, raw_value
    return False, "", ""
```

在所有欄位裡面找關鍵字。
(在剛剛整理好的 pieces 裡, 一個欄位一個欄位去搜尋。回傳:(是否命中, 該欄位名稱, 該欄位原始內容)沒找到則回傳 (False, "", "")。)

暴力破解偵測。(不同於 SQL Injection 是看關鍵字，暴力破解看的是『頻率』。這裡做了一個滑動視窗 (Sliding Window) 演算法，系統會記憶每個 IP 過去 60 秒內的登入嘗試次數，一旦超過門檻值（Threshold），就會觸發警報。)

```python
def _check_suspicious_ua(input_data: dict) -> Tuple[bool, str]:
    ua = _to_str(input_data.get("user_agent", "")).lower()
    if not ua:
        return False, ""

    for pattern in RULES.get("SUSPICIOUS_UA_PATTERNS", []):
        p = pattern.lower()
        if p and p in ua:
            return True, ua
    return False, ""
```

檢查 User-Agent 是否包含常見掃描器 / 攻擊工具字樣。
(命中時視為 SUSPICIOUS_UA，屬於低～中風險（輕量級告警）。主要是提醒管理員「有工具在掃描」，不代表一定有成功入侵，但值得注意。)

```python
def _check_bruteforce(input_data: dict) -> Tuple[bool, str]:

    ip = _to_str(input_data.get("ip_address", ""))
    url = _to_str(input_data.get("url", "")).lower()
    method = _to_str(input_data.get("http_method", "")).upper()

    # 不是 login 相關的就不算登入嘗試
    if "login" not in url:
        return False, ""

    # 只統計 POST /login (可以視情況調整)
    if method != "POST":
        return False, ""

    now = time.time()
    attempts = _LOGIN_ATTEMPTS.get(ip, [])
    attempts.append(now)

    # 移除超過時間窗的舊紀錄
    cutoff = now - BRUTE_FORCE_WINDOW_SECONDS
    attempts = [t for t in attempts if t >= cutoff]
    _LOGIN_ATTEMPTS[ip] = attempts

    if len(attempts) >= BRUTE_FORCE_THRESHOLD:
        info = f"{ip} tried login {len(attempts)} times in {BRUTE_FORCE_WINDOW_SECONDS} seconds"
        return True, info

    return False, ""
```

```python
def _check_ssrf(input_data: dict) -> Tuple[bool, str]:

    candidates: List[str] = []

    # 先把原始 URL 也算進可能的 payload
    raw_url = unquote(_to_str(input_data.get("url", "")))
    candidates.append(raw_url)

    # params / body 裡的內容
    params = input_data.get("params", {}) or {}
    body = input_data.get("body", {}) or {}

    for d in (params, body):
        for _, v in d.items():
            s = unquote(_to_str(v))
            candidates.append(s)

    for value in candidates:
        v = value.strip()
        if not v:
            continue

        # 找出看起來像 http/https 的 URL
        lower = v.lower()
        url_str = None
        if lower.startswith("http://") or lower.startswith("https://"):
            url_str = v
        else:
            idx = lower.find("http://")
            if idx == -1:
                idx = lower.find("https://")
            if idx != -1:
                url_str = v[idx:]
```

```python
        if not url_str:
            continue

        try:
            parsed = urlparse(url_str)
        except Exception:
            continue

        host = parsed.hostname
        if not host:
            continue

        if _is_private_or_metadata_ip(host):
            # 命中 SSRF 風險
            return True, url_str

    return False, ""
```

兩張都是偵測ssrf攻擊(把可能包含 URL 的地方全部抓出來，對每一個字串，嘗試找出其中的 `http://` 或 `https://` 開頭的網址，呼叫 `_is_private_or_metadata_ip(host)` 判斷是不是，如果是，回傳 (`True`， 這個可疑的 URL)，表示有 SSRF 風險。)

```python
def detect_attack(input_data: dict) -> dict:
    # 預設結果（沒有攻擊）
    result = {
        "is_attack": False,
        "attack_type": "NONE",
        "severity": "LOW",
        "payload": "",
        "should_block": False,   # 先預設 False，最後再由 _apply_block_flag 決定

        "ip_address": _to_str(input_data.get("ip_address", "")),
        "timestamp": _now_tw(),
    }

    # 把所有欄位收集起來（url / params / body / user_agent）
    pieces = _collect_fields(input_data)

    # 檢查 SQL Injection
    hit, field, value = _find_pattern(pieces, RULES["SQLI_PATTERNS"])
    if hit:
        result["is_attack"] = True
        result["attack_type"] = "SQLI"
        result["severity"] = "HIGH"
        result["payload"] = f"{field}: {value}"
        return _apply_block_flag(result)
```

主偵測流程。(這裡採用了 Pipeline（管線）的設計模式。資料進來後，會依序經過 SQL Injection、XSS、路徑遍歷等檢測模組。我們將危害程度最高的 SQLi 放在最前面檢查，一旦命中就立即回傳並阻擋，這樣可以優化系統的效能與反應速度)。