

從冷知識到漏洞

From Trivia to Vulnerabilities

漏洞的三種來源



「我知道，但來不及了」

先建票，之後修



「我忘了」

像寶可夢 PIPLUP 一樣遺忘了細節



「我不知道這樣有問題」

這就是本次演講的重點——**冷知識**

防禦心法一：不同 Context 的相等就是不相等

場景

密碼重設功能，帳號盜用風險

```
const user = await safeSQL(  
  'select * from users where email = ?',  
  [email]  
);  
// ...  
userService.sendResetLink({  
  link: ....,  
  to: email, // 寄信給「使用者輸入」的 email  
});
```

攻擊

Payload: test@gmaîL.com

1. 資料庫回傳 test@gmail.com 的使用者資料。
2. 重設連結寄到攻擊者的 test@gmaîL.com。
3. 帳號被盜用 。

冷知識

在 MySQL utf8mb4_unicode_ci 中...

'gmail.com'  'gmaîL.com'

因為它們的 Weight String 相同

防禦

- 將 Collation 改為 utf8mb4_bin。
- 程式碼端再次驗證。

```
ALTER TABLE users MODIFY COLUMN email  
VARCHAR(255) COLLATE utf8mb4_bin;
```

防禦心法二：小心你的字串被取代

場景

開發者試圖過濾 < > \" 字元來防止 XSS

```
const safeValue = value.replace(/<>"]/g, '');
document.body.innerHTML = tmpl.replace(
  '{{value}}', value
);
```

攻擊

Payload: ?v=\$'

輸入 '\$' 會導致 replace 注入 {{value}} 後面的所有內容，攻擊者可藉此閉合標籤並注入 onclick 事件 。

冷知識

`string.replace(pattern, replacement)` 的秘密

當 replacement 是字串時，`\$` 有特殊意義。

- \${} : 插入匹配到的字串
- \$' : 插入匹配字串 **後方** 的所有內容 

防禦

不要傳入字串，改用 **Callback Function**。

```
document.body.innerHTML = tmpl.replace(
  '{{value}}', () => value
);
```

防禦心法三：你倒是給我看文件啊

Clean is not clean, Join is not join

場景

讀取插件檔案，卻導致任意檔案讀取

```
requestedFile := filepath.Clean(...)  
pluginFilePath = filepath.Join(...)  
f, err := os.Open(pluginFilePath)
```

攻擊

Grafana CVE-2021-43798 

Payload: /etc/passwd

Join("plugins", "/etc/passwd") 直接
變成 /etc/passwd，成功讀取系統檔案。

冷知識

- `filepath.Clean`：只做詞法處理，
`Clean("/etc/passwd")` 仍然是
"/etc/passwd" .
- `filepath.Join`：如果參數是絕對路徑，
會忽略前面的所有路徑 .

防禦

- 只取檔名。
`requestedFile := filepath.Base(...)`
- 檢查最終路徑的前綴。
`if !strings.HasPrefix(finalPath, ...)`

防禦心法四：絕對！不要！把使用者的輸入照單全收

場景

為了方便，將 URL 參數直接展開到 React Component 中

```
const obj = Object.fromEntries(params);  
<Hello {...obj} />
```

攻擊

Payload: ?is=huli&onclick=alert(1)

React 會 render 出 `<div is="huli"\ onclick="alert(1)">Hello</div>`，成功觸發 XSS 💣。

(註：*React 19* 已修正此行為)

冷知識

React 的例外規則

當 Props 中包含 `is` 屬性時，React 會將其視為 **Custom Component** 🚧，並且不會過濾 `on` 開頭的事件屬性！

防禦

拒絕展開 (**Spread**)，使用白名單 (**Allow-list**)。

```
const { className, id } = obj;  
return <Hello className={className} id={id} />;
```

防禦心法五：謹慎使用 '/regex/'

場景

使用 Regex 檢查字串是否包含惡意 HTML 標籤

```
/.*[<].*[>=].*/s
```

攻擊

WAF Bypass

```
if (!is_php($input))
```

攻擊者傳入超長字串，使 preg_match 回傳 false。

!false 變成 true，成功繞過檢查 🚧。

冷知識

ReDoS：不好的 Regex（如 `.*` 重複）遇到特定字串會導致大量回溯 (backtracking)，使程式卡死 🐌。

PHP `preg_match`：當回溯次數超過上限，preg_match 回傳 false (錯誤)，而不是 0 (不匹配) !。

防禦

嚴格檢查 PHP 的回傳值。

```
$result = preg_match(...);  
if ($result === false) {  
    throw new Exception("Regex error");  
}
```

防禦心法六：微軟的「貼心」 有時是多餘的

場景

應用程式層有使用者名稱黑名單，但資料庫是 MSSQL

1. Attacker inputs "admin "
2. Node.js check passes:
`blocklist.includes("admin ") is false`
3. SQL query sent: WHERE username = 'admin '

冷知識

MSSQL 會忽略字尾的空白

'admin' = 'admin ' is TRUE
LEN('admin ') → 5
DATALENGTH('admin ') → 6

攻擊

Logic Bypass / Account DoS

攻擊者用 "admin " 註冊，在應用層是新帳號，但在資料庫層卻匹配到真的 "admin" 帳號，導致權限盜用或邏輯混亂。

防禦

在應用程式端 trim()。

`const username = req.body.username.trim();`

在 SQL 端使用 DATALENGTH 驗證。

`...AND DATALENGTH(username) = DATALENGTH(@u)`

你的新防禦軍火庫



- 🛡 MySQL Unicode: 為敏感欄位使用 `utf8mb4_bin` Collation。
- 🔒 React Props: 絶不展開使用者輸入，使用白名單明確指定屬性。
- ⚙️ JS Replace: 永遠使用 Callback Function () => value' 來替換。
- 👁️ Regex: 在 PHP 中，必須用 `==false` 嚴格檢查錯誤。
- 📁 Go Filepath: 用 `filepath.Base()` 取檔名，或驗證路徑前綴。
- 💽 MSSQL Spaces: 總是 `trim()` 使用者輸入，或用 `DATALENGTH` 驗證。

**“不要只依賴直覺，
寫好測試，確保你的 Output 是對的。”**

(Don't just rely on intuition. Write good tests. Ensure your output is correct.)

