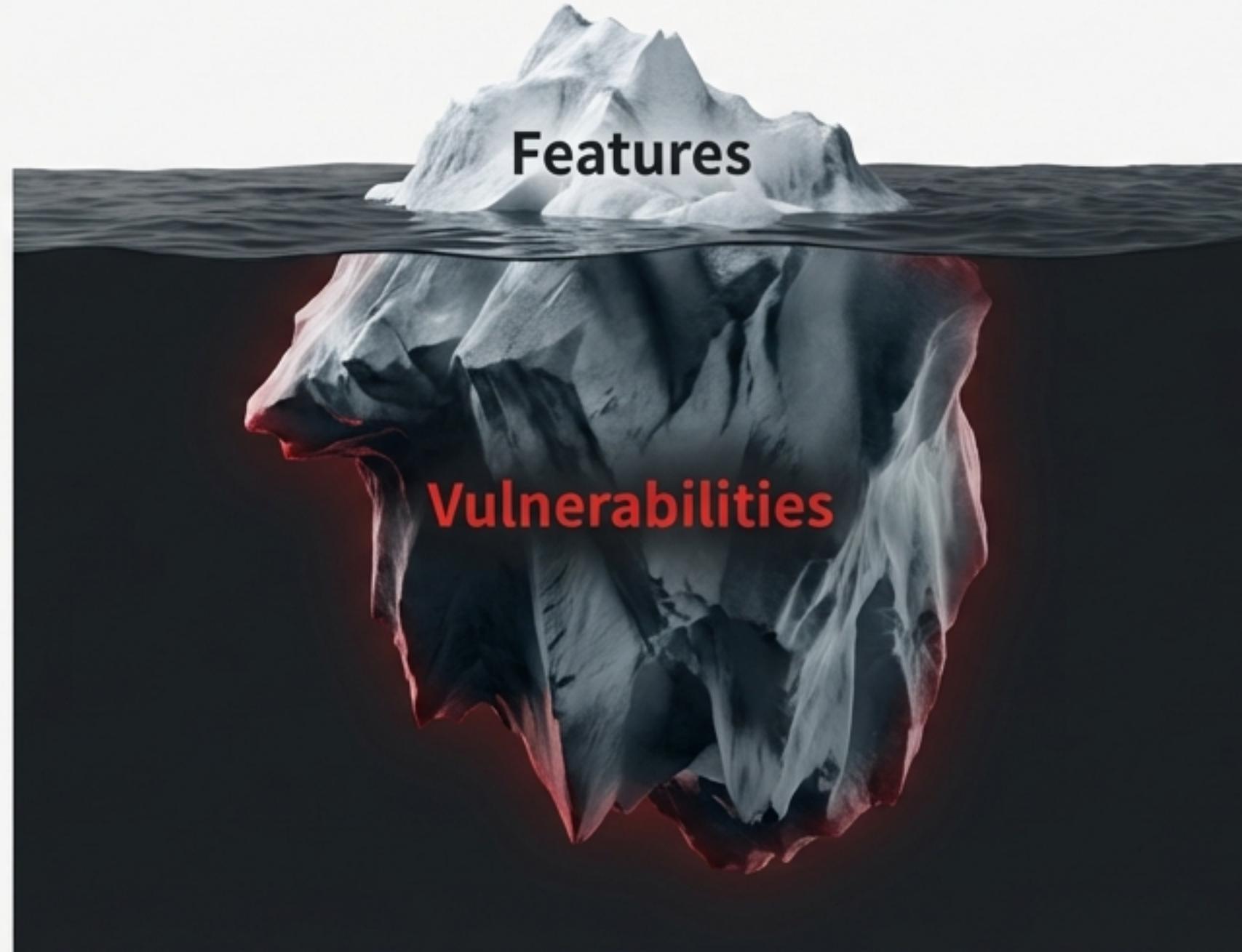


# 從冷知識到漏洞

## 那些你不知道它會出事的寫法

From Trivia to Vulnerabilities: The code you didn't know was a problem.



# 漏洞的三種來源



「我知道有漏洞，  
但上線快來不及了」

技術債  
(Technical Debt)



「好像有碰過  
但我忘了」

記憶失誤  
(Memory Lapse)



「我不知道這樣寫  
有問題」

我們今天的重點  
(Our Focus Today)

# 防禦心法一：不同 Context 的相等，就是不相等

場景：一個忘記密碼的功能。

```
// Node.js - Forgot Password Logic
async function forgotPassword(req, res) {
  const email = req.body.email.toLowerCase();
  // SQL: select * from users where email = ?
  // DB Collation: utf8mb4_unicode_ci
  const user = await safeSQL('select * from users where email = ?', [email]);
  if (user) {
    userService.sendResetLink({
      link: userService.generateLink(user.id),
      to: email,
    }).catch(console.error);
  }
  return res.status(204).end();
}
```



這段程式碼，哪裡有問題？

## 💡 冷知識：MySQL 的 Unicode 相等性

在 `utf8mb4\_unicode\_ci` 定序下，資料庫認為：

'gmail.com' == 'GMAIL.com'  
(不分大小寫)

'gmail.com' == 'gmaîL.com'  
(Unicode 相等)

因為它們的 **Weight String**  相同。

## 💥 攻擊：帳號挾持 (Account Takeover)



# 防禦心法二：你的字串正在被「取代」

場景：一個看似安全的 XSS 過濾器。

```
const tmpl = '<div>
  click <a href="{{value}}>here</a>
</div>';
const value = new URL(location.href).searchParams.get('v');

// 開發者試圖移除特殊字元
const safeValue = value.replace(/[\<\>]/g, ''); ⓘ

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

明明過濾了`<`和`>`，為何還能 XSS？



## 冷知識：`replace()` 的第二個參數不是純文字

當 replacement 是字串時，支援特殊樣式：

- ⚡ \$& 插入匹配到的字串。
- ⚡ \$' 插入匹配字串 後方 的所有內容 ➤。

### 攻擊 Payload : `?v=\$`

```
tmpl.replace('{{value}}', "$")
```

→ `\$` 會將 `{{value}}` 後方  
的字串 (`>click`) 插入。

```
<a href=''  
onclick=alert(1)>click</a>
```

→ `click'>click`

### 修補方案：用函數取代字串

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

Callback 函式的回傳值會被視為純文字，不會解析特殊字元。

# 防禦心法三：你倒是給我看文件啊

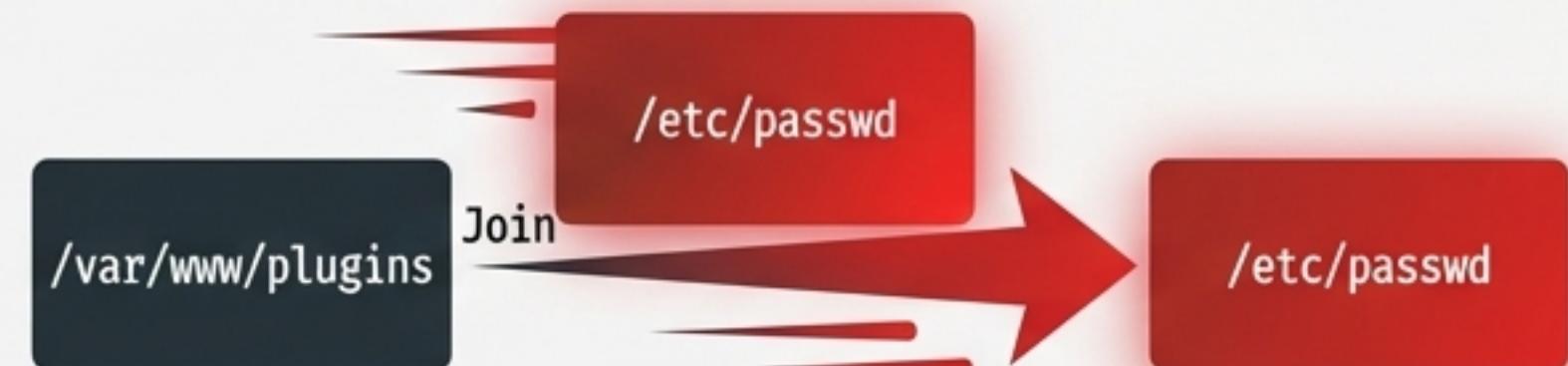
場景：Grafana 的一個路徑穿越漏洞 (CVE-2021-43798)。

```
1 // 開發者認為 Clean() 已經消毒了路徑
2 requestedFile := filepath.Clean(web.Params(c.Req)["*"])
3 // 然後放心地拼接路徑
4 pluginFilePath = filepath.Join(plugin.PluginDir, requestedFile)
5 // nolint:gosec
6 f, err := os.Open(pluginFilePath)
```

## 🧠 💥 The Twist & Weaponization

- 👉 冷知識 1: `filepath.Clean` 只是詞法處理，`Clean("/etc/passwd")` 仍然是 `"/etc/passwd"`。
- 👉 冷知識 2: `filepath.Join` 遇到絕對路徑時，會 **拋棄** 前面的所有路徑。

**ATTACKER'S INPUT: /etc/passwd**



**RESULT: /etc/passwd**

# 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，直接將 URL 參數展開到 React Component。

```
function App() {  
  const params = new URLSearchParams(window.location.search);  
  const obj = Object.fromEntries(params);  
  return (  
    // DANGER: Spreading user-controlled object  
    Hello  
  );  
}
```

## 冷知識 (The Twist)

- React 預設會過濾 `onclick`。
- **例外：**如果 props 包含 `is` 屬性，React 會將其視為 Custom Component，並 **停止過濾** `on` 開頭的屬性。

## 攻擊 (The Weaponization)

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

```
<html onclick="alert(1)>  
  Hello  
</html>
```

# The MSSQL Minefield



Microsoft SQL Server 功能強大、特性豐富。  
但它某些最『貼心』的設計，卻為我們埋下了一片漏洞的雷區。

# 終極權力，終極後果

## Stacked Queries (堆疊查詢)

💡 冷知識: MSSQL 允許用分號 ; 一次執行多條指令。

💥 攻擊: `1; DROP TABLE users; --`



後果: 資料被刪除，甚至透過 `xp_cmdshell` 達成 RCE。

## Trailing Spaces (結尾空白)

💡 冷知識: 'admin' = 'admin ' → TRUE

💥 攻擊: 攻擊者註冊 'admin ' 來繞過應用程式層的 'admin' 黑名單。

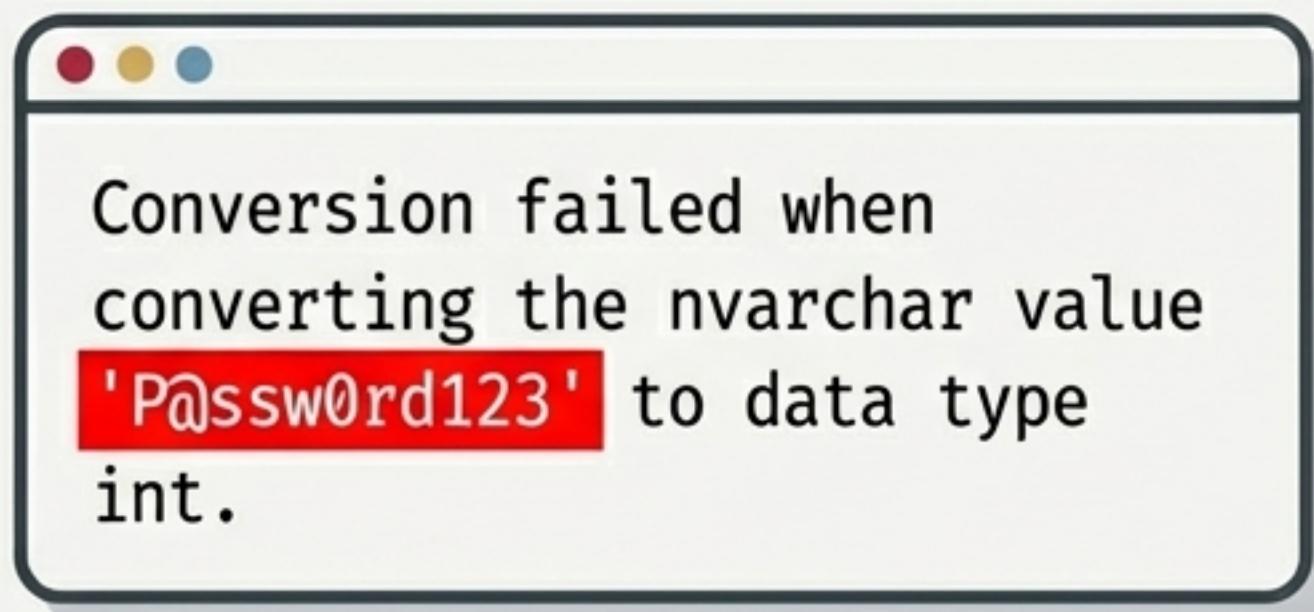


# 當「貼心」變成致命武器

## Error-Based Injection (基於錯誤的注入)

💡 冷知識: 資料轉型失敗時, 錯誤訊息會「貼心地」包含導致錯誤的值。

💥 攻擊: 注入 `CAST(user_password AS INT)`。



## Implicit Conversion DoS (隱式轉型阻斷服務)

💡 冷知識: MSSQL 型別優先級中, `INT > VARCHAR`。

💥 攻擊: 當查詢 `WHERE varchar_column = int_value` 時, MSSQL 會將每一筆 `VARCHAR` 資料轉成 `INT` 再比較。



後果: 索引失效, 全表掃描, CPU 100% 滿載。

# 拆除 MSSQL 雷區 (The MSSQL Shield)



## 針對 Stacked Queries

唯一解：參數化查詢 (Parameterized Queries)。參數會將；視為純文字，而非指令分隔符。



## 針對 Trailing Spaces

在應用程式端，對所有輸入執行 `.trim()`。



## 針對 Error-Based Injection

在正式環境中，關閉詳細資料庫錯誤回報。  
使用 `BEGIN TRY...END CATCH` 捕捉例外。



## 針對 Implicit Conversion DoS

確保應用程式傳入的參數型別 (`sql.VarChar`)  
與資料庫欄位型別完全一致。

# 總結：化冷知識為你的防禦武器

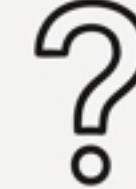
威脅 (Threat)	冷知識 (Trivia)	解法 (Solution)
MySQL Collation	'a' == 'â'	使用 <code>utf8mb4_bin</code>
JS Replace	`\$` 會注入後續字串	使用 Callback <code>() =&gt; val</code>
Go Filepath Join	絕對路徑會覆蓋前方路徑	驗證路徑前綴或用 <code>Base()</code>
React Spread Props	<code>is</code> 屬性會關閉 XSS 保護	使用白名單，勿直接展開
MSSQL Medley	';', `` (space), Errors, <code>INT&gt;VARCHAR</code>	參數化, <code>Trim</code> , <code>Type-Match</code>

# 終極防禦心法：培養好奇心

**最強的防禦不是某個函式或設定，而是一種心態。**



Read the documentation.  
(去讀文件)

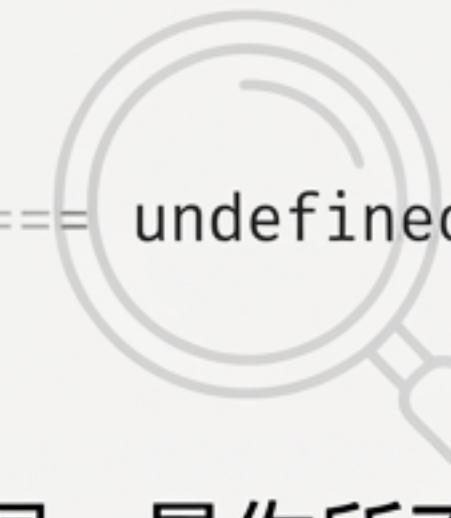


Question the ‘magic’.  
(質疑那些「魔法」)



Write tests for the weird edge cases.  
(為奇怪的邊界條件寫測試)

```
if (userInput === undefined) { // ... }
```



你的最大漏洞，是你所不知道的事。