

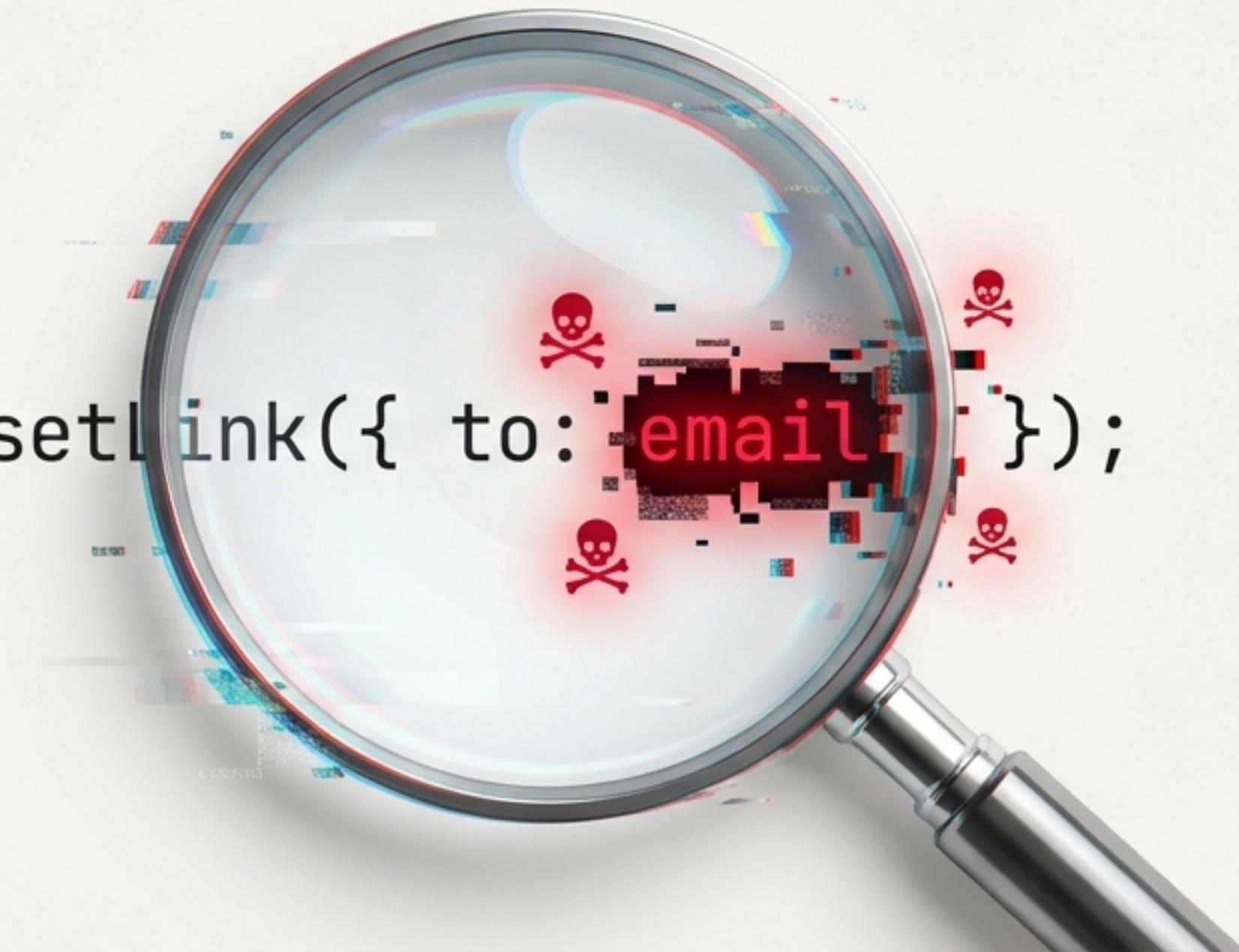
# 從冷知識到漏洞： 你不懂的 Web，駭客懂

WebConf Taiwan 2025 | Huli

# 你的程式碼，真的安全嗎？

最危險的，不是我們知道的 Bug。而是那些藏在最顯眼之處，  
源自於我們自以為理解的功能的漏洞。

```
userService.sendResetLink({ to: email });
```



# 漏洞的三種來源



## 1. 沒時間修

「先開票，之後修。」



## 2. 我忘了

「好像碰過但忘了細節。」



## 3. 我不知道

這是我們今天的主題。

# 防禦心法一：當「相等」不再相等

```
async function forgotPassword(req, res) {  
  const email = req.body.email.toLowerCase();  
  // SQL: select * from users where email = ?  
  const user = await safeSQL('select * from  
    users where email = ?', [email]);  
  if (user) {  
    userService.sendResetLink({  
      link: userService.generateLink(user.id),  
      to: email, // !!! 寄信給「使用者輸入」的 email  
    }).catch(console.error);  
  }  
  return res.status(204).end();  
}
```



# 冷知識：MySQL 的 Unicode 定序陷阱



## 冷知識

在 MySQL `utf8mb4\_unicode\_ci` 定序中，`gmail.com` 和 `gmaÎL.com` 被視為相等，因為它們的 **Weight String** 相同。



## 攻擊手法



攻擊者

提交 `test@gmaÎL.com`



資料庫

將重設連結寄到攻擊者控制的  
`test@gmaÎL.com`。



伺服器

結果：帳號劫持 (Account Takeover)

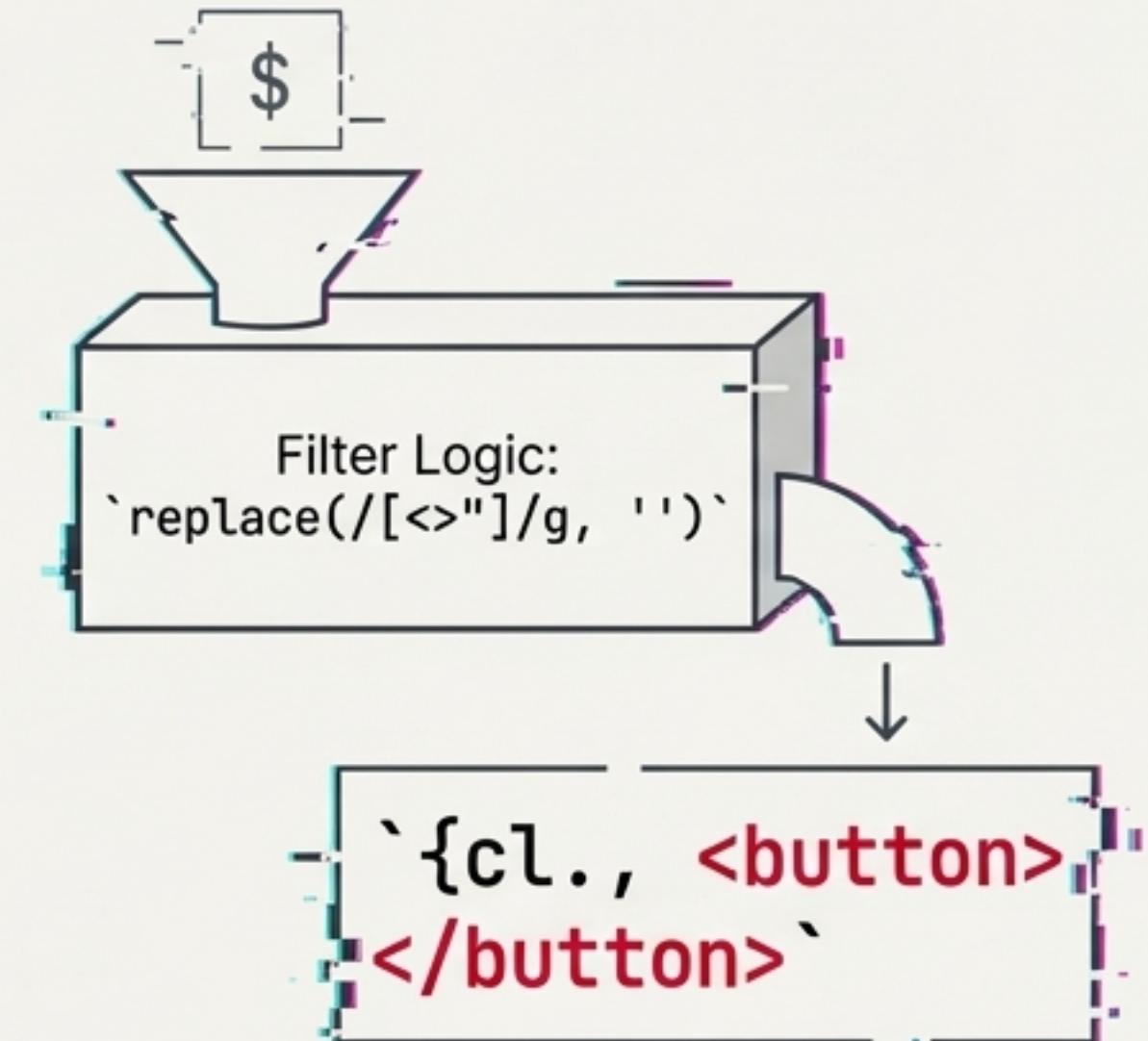
# 防禦心法二：你的字串，正在背叛你

```
const tmpl = '<button>{{value}}</button>';
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, '');

// A subtle red glow emanates from this line
document.body.innerHTML = tmpl.replace('{{value}}', value);
```

Input: `?v=\$`



意外輸出：模版注入了`\$click\$`字串。

# 冷知識：`replace()` 的第二個參數不是字串，是魔法

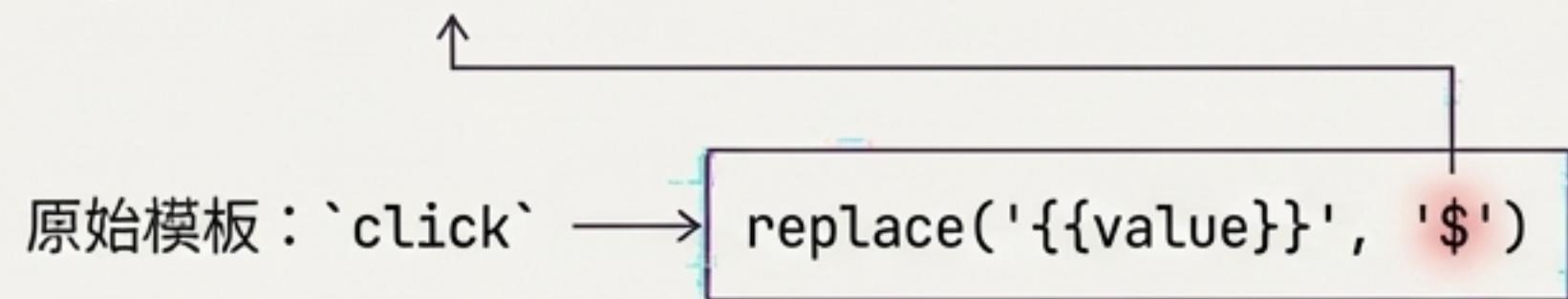
## 冷知識

JavaScript `replace(pattern, replacement)` 的 `replacement` 參數支援特殊模式：

- `\$\$` → 插入 \$
- `\$&` → 插入匹配到的字串
- `'\$` → 插入匹配字串**前方**的所有內容
- `'\$` → 插入匹配字串**後方**的所有內容

## 攻擊手法

Payload : `?v=\$`



原始模板：`click` →

結果：注入的 HTML 變成 `<a href="click"></a>`，  
攻擊者可藉此閉合屬性並注入 `onclick=alert(1)`，  
成功繞過過濾。

# 防禦心法三：「Clean」不代表乾淨，「Join」不保證安全

```
requestedFile := filepath.Clean(web.Params(c.Req)[ "*" ])
pluginFilePath = filepath.Join(plugin.PluginDir, requestedFile)

// nolint:gosec
f, err := os.Open(pluginFilePath)
```

過度的自信



# 冷知識：`Clean` 只整理語法，`Join` 尊重絕對路徑

## 冷知識

``filepath.Clean``: 只做「詞法處理 (Lexical Processing)」。它會將 `a/../../b` 轉為 `b`，但不會移除開頭的 `/`。

``filepath.Join``: 當任何一個參數是絕對路徑時 (如 `/etc/passwd`)，它會忽略之前的所有路徑。

## 攻擊手法

真實案例 : Grafana CVE-2021-43798

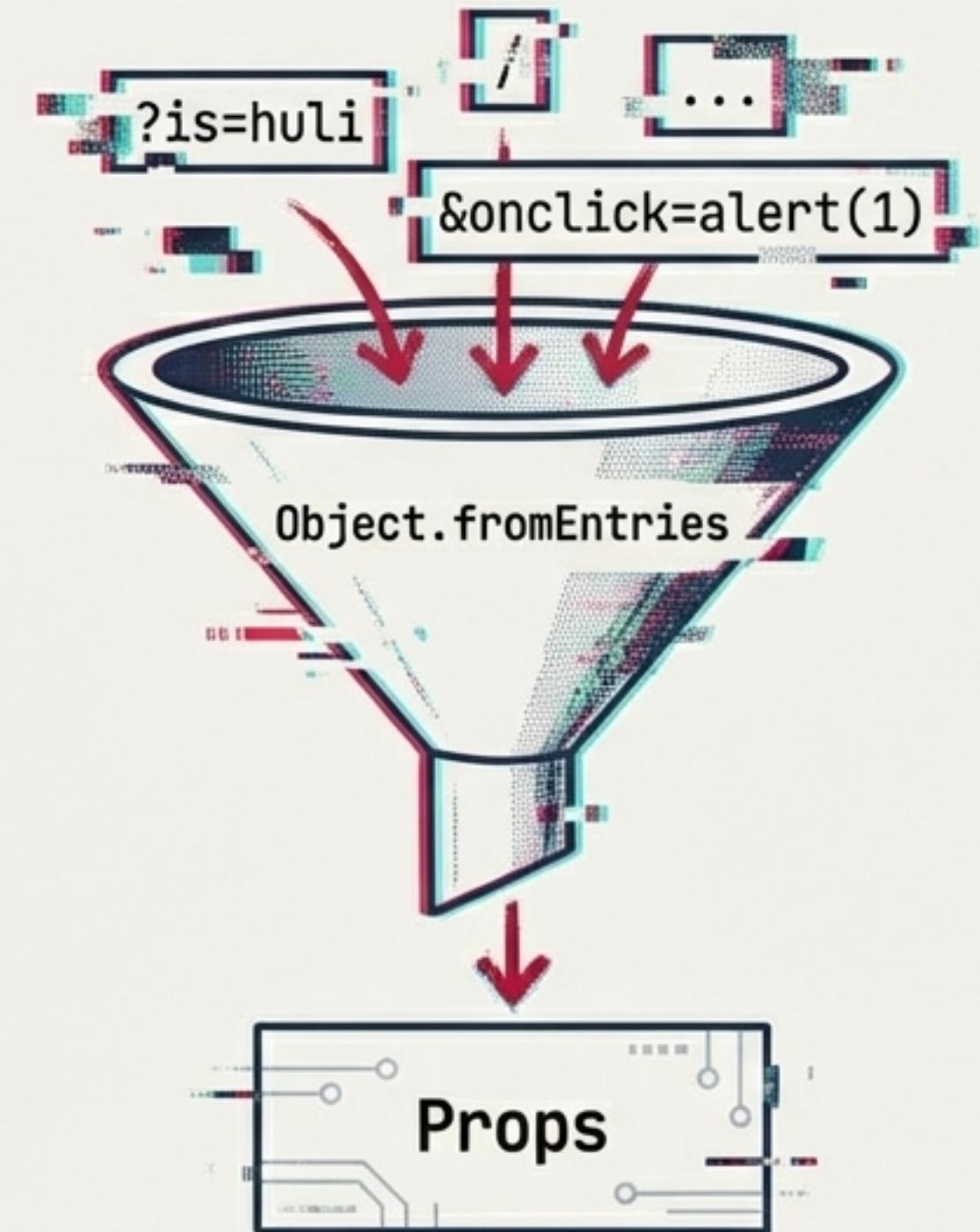
Payload: `../../../../../etc/passwd`

~~Join("plugins/welcome/", ".../.../...passwd")~~ → ../../../../../etc/passwd

被解析為直接存取系統檔案，導致路徑穿越漏洞。

# 防禦心法四：不要相信使用者的「慷慨」

```
function App() {  
  const params = new URLSearchParams(window.location.search);  
  const obj = Object.fromEntries(params);  
  
  return (  
    <h1 {...obj}>Hello</h1>  
  );  
}
```



# 冷知識：一個 `is` 屬性，關閉 React 的安全模式

## 🧠 冷知識

React 通常會過濾 `onclick` 這類事件處理器。

**例外：**如果 Props 包含 `is` 屬性，React 會將其視為 **Custom Component**。在此模式下，React 不會過濾 `on` 開頭的屬性，而是直接傳遞給 DOM。

## 💥 攻擊手法

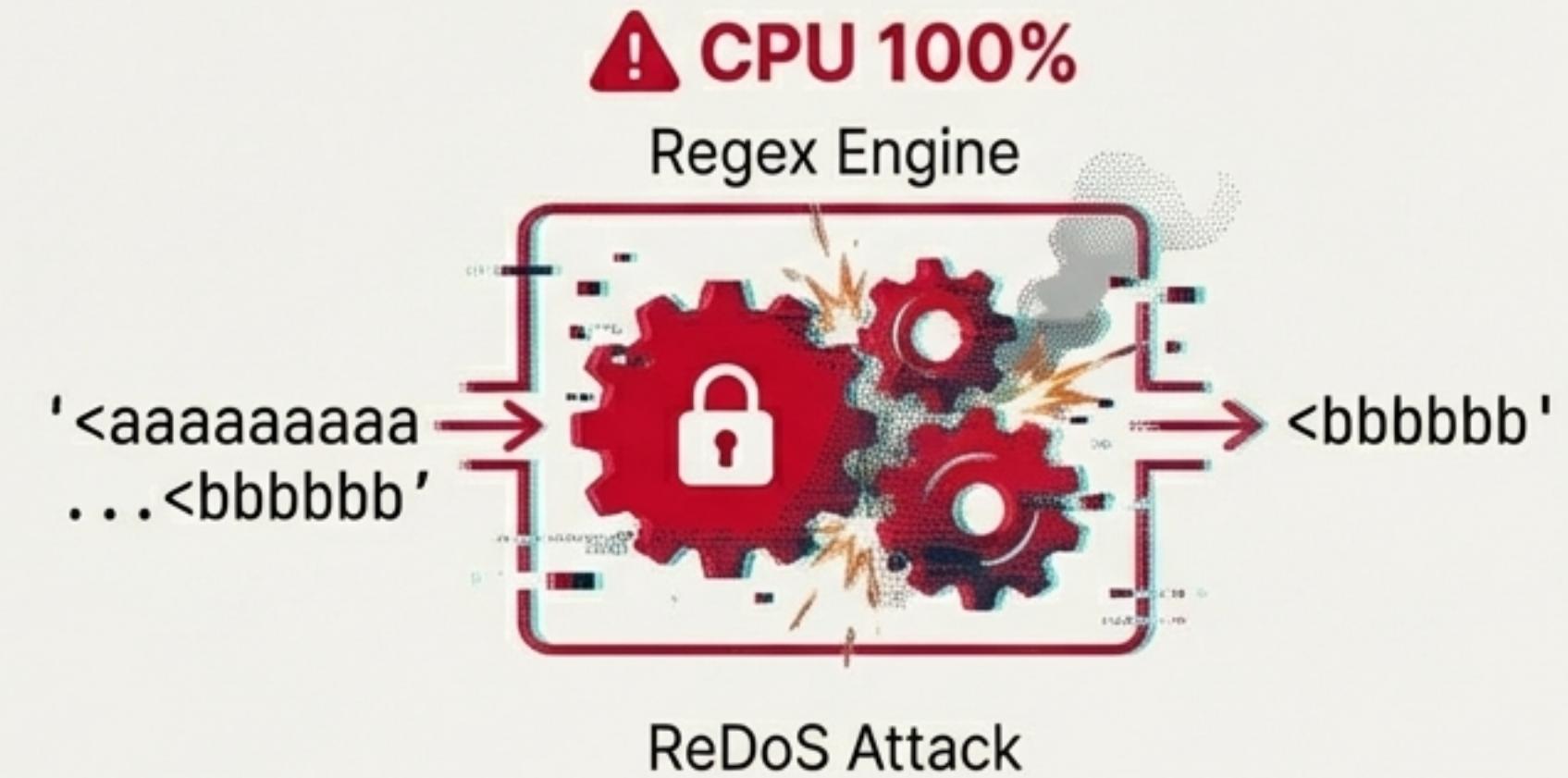
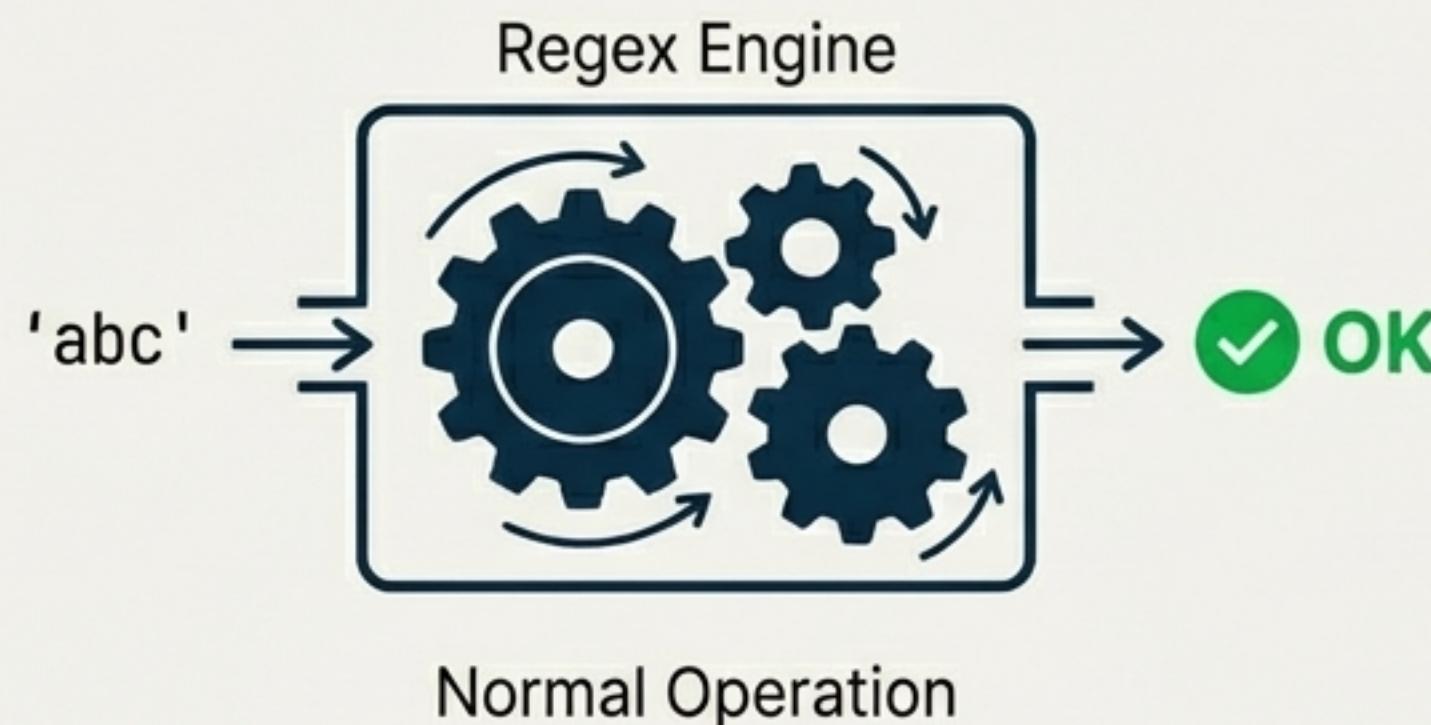
**Payload :** `?is=huli&onclick=alert(1)`

**結果 (HTML) :** <div is="huli" onclick="alert(1)">Hello</div>

\*(註：React 19 已修正此行為)

# 防禦心法五：你以為的滴水不漏，其實是 CPU 黑洞

```
function isUnsafeString(str) {  
    // 檢查是否包含 < ... > 結構  
    const regex = /.*[<].*[>=].*/s;  
    return regex.test(str);  
}
```



# 冷知識：Regex 會崩潰，而 `false` 不是 0



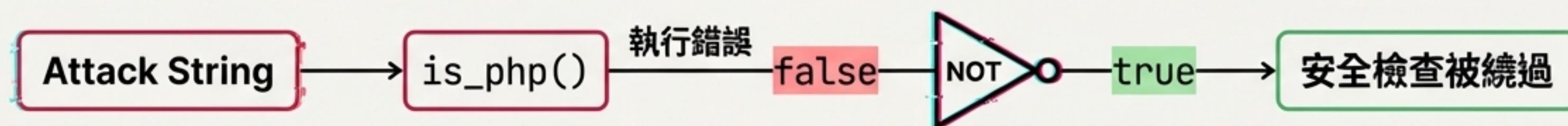
## 冷知識

1. **ReDoS**：寫得不好的 Regex (`.\*` 疊加) 遇到特定字串時會引發大量**回溯 (Backtracking)**，導致 CPU 100% 癱瘓。
2. **PHP `preg\_match`**：當回溯超過上限 (`pcre.backtrack\_limit`) 時，它不回傳 `0` (不匹配)，而是回傳 `false` (**執行錯誤**)。



## 攻擊手法

情境`：`if (!is\_php(\$input)) { // 執行安全操作 }`



條件判斷變成 `!false`，結果為 `true`，成功**繞過**安全檢查。

# 五個必須記住的資安真相



**Unicode ≠ 直覺相等：**`gmaÎL` 在資料庫中可能等於 `gmail`。



**Replace 是個函式：**`\$` 在 `replace` 中不是普通字串。



**路徑不是字串：**`Clean` 不等於 `Sanitize`。



**Props 是 API：**絕不直接展開使用者輸入。



**Regex 會失敗：**`false` 在 PHP 中不等於「沒有匹配」。

**「不要只依賴直覺，寫好測試，  
確保你的 Output 是對的。」**



Huli  
@huli