CYCRAFT

真相只有一個！

基於大型語言模型的證據導向漏洞發掘

Dange, oalieno

CRAFTCON TAIWAN

AGENTIC AI
BLUE TEAM X RED TEAM

林殿智（Dange）

資深資安主任研究員

趙偉捷（oalieno）

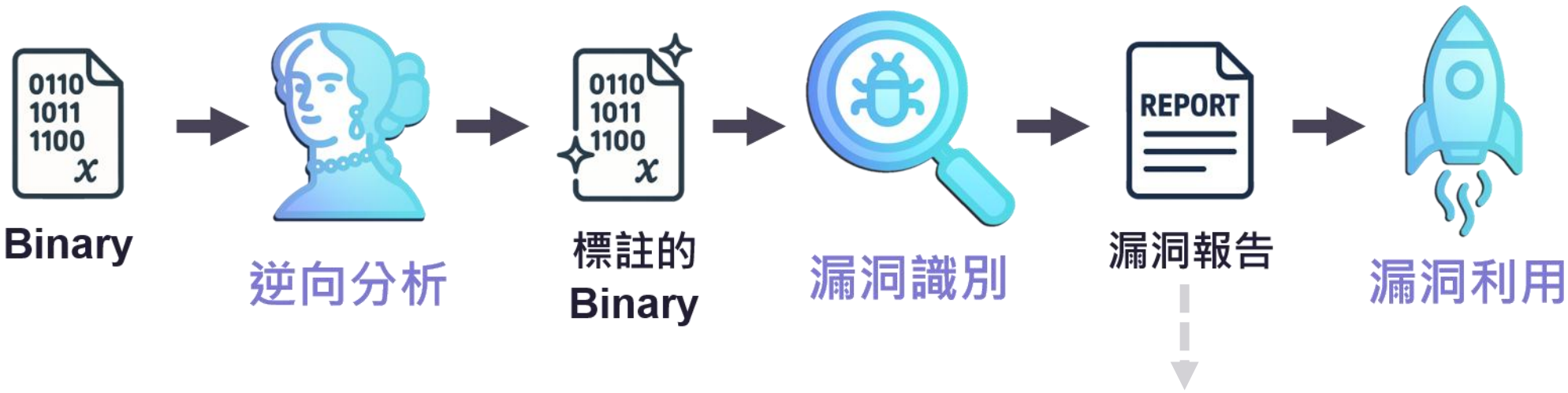
資深資安研究員

漏洞挖掘基本流程


0110
1011
1100
x
Binary
逆向分析
0110
1011
1100
x
標註的
Binary
漏洞識別
REPORT
漏洞報告
漏洞利用


```
__int64 __fastcall sub_1330(__int64 a1, __int64 a2)
{
  if ( !sub_125B(a1, "show_config") )
    return sub_1189(&unk_205D, (int)a2 + 32, ...);
  result = sub_125B(a1, "update_notes");
  if ( !(_DWORD)result )
  {
    sub_12CC(a2 + 32);
    return sub_1189(&unk_2080, "update_notes", ...);
  }
  return result;
}
```


```
__int64 __fastcall handle_command(__int64 a1, __int64 a2)
{
  if ( !my_strcmp(a1, "show_config") )
    return my_printf(&unk_205D, (int)a2 + 32, ...);
  result = my_strcmp(a1, "update_notes");
  if ( !(_DWORD)result )
  {
    my_gets(a2 + 32);
    return my_printf(&unk_2080, "update_notes", ...);
  }
  return result;
}
```




```
__int64 __fastcall handle_command(__int64 a1, __int64 a2)
{
  if ( !my_strcmp(a1, "show_config") )
    return my_printf(&unk_205D, (int)a2 + 32, ...);
  result = my_strcmp(a1, "update_notes");
  if ( !(_DWORD)result )
  {
    my_gets(a2 + 32);
    return my_printf(&unk_2080, "update_notes", ...);
  }
  return result;
}
```

這段程式碼可能存在**嚴重的安全漏洞**，以下是具體分析：

1. `my_gets()` 使用導致的 Buffer Overflow

cpp Copy code

```
my_gets(a2 + 32);
```

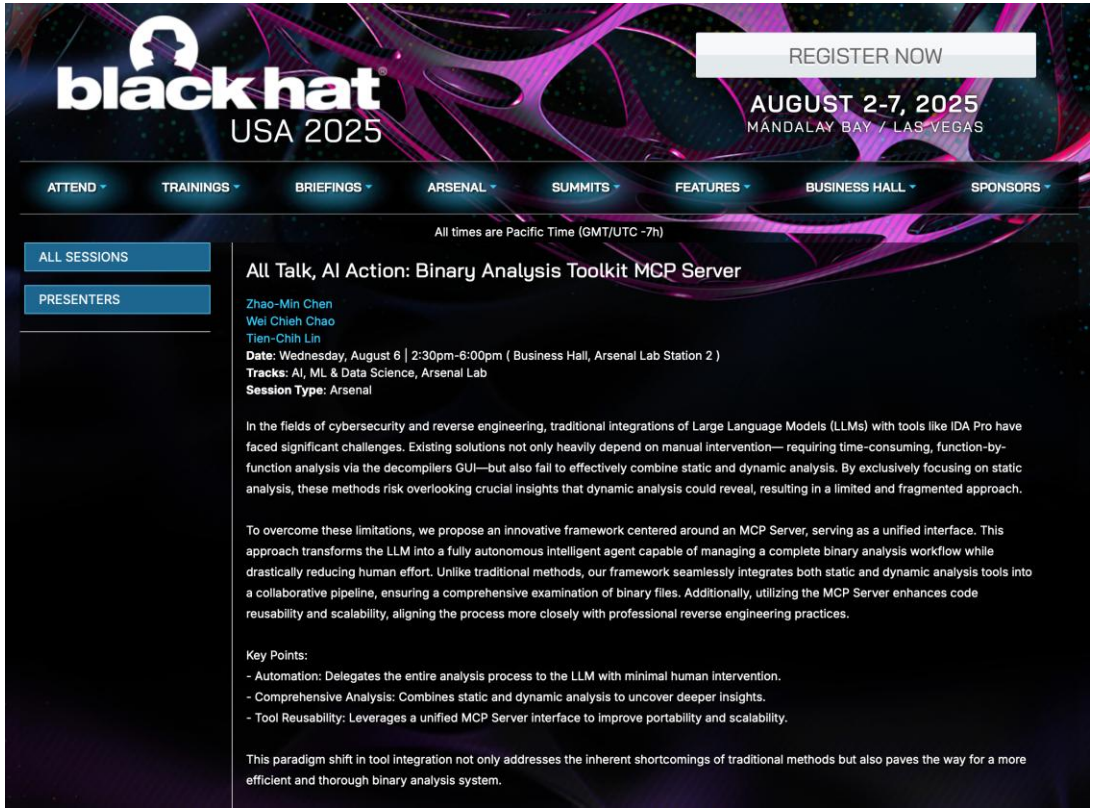
問題：

`my_gets()` 類似於 `gets()`，是**不安全的函式**，因為它不會限制輸入長度。

🎉 成果產出 🎉

Black Hat USA 2025 Briefing !

Black Hat USA 2025 Arsenal !



Reverse Engineering by LLM

使用 LLM 逆向遇到的問題

來看一個例子

```
void sub_11F7(const char *a1) {
  size_t v2;
  char s[56];

  s[0] = 0;
  v2 = 50 - strlen(s);

  printf("Append username '%s'\n", a1);
  printf("Remaining space: %zu bytes.\n", v2);
  sub_11C9(s, a1, v2);
  printf("Logged: %s\n", s);
}
```

```
void log_username(const char *username) {
    char buffer[56];
    size_t remaining_space;

    buffer[0] = '\0';
    remaining_space = 50 - strlen(buffer);

    printf("Append username '%s'\n", username);
    printf("Remaining space: %zu bytes.\n", remaining_space);
    safe_append(buffer, username, remaining_space);
    printf("Logged: %s\n", buffer);
}
```

來看一個例子

```
void sub_11F7(const char *a1) {
  size_t v2;
  char s[56];

  s[0] = 0;
  v2 = 50 - strlen(s);

  printf("Append username '%s'\n", a1);
  printf("Remaining space: %zu bytes.\n", v2);
  sub_11C9(s, a1, v2);
  printf("Logged: %s\n", s);
}
```

```
void log_username(const char *username) {
    char buffer[56];
    size_t remaining_space;

    buffer[0] = '\0';
    remaining_space = 50 - strlen(buffer);

    printf("Append username '%s'\n", username);
    printf("Remaining space: %zu bytes.\n", remaining_space);
    safe_append(buffer, username, remaining_space);
    printf("Logged: %s\n", buffer);
}
```

這個函式真的 safe 嗎?

strcat	無限制 (overflow!)
strncat	限制複製長度

```
void log_username(const char *username) {
    char buffer[56];
    size_t remaining_space;

    buffer[0] = '\0';
    remaining_space = 50 - strlen(buffer);

    printf("Append username '%s'\n", username);
    printf("Remaining space: %zu bytes.\n", remaining_space);
    safe_append(buffer, username, remaining_space);
    printf("Logged: %s\n", buffer);
}
```

```
char * sub_11C9(char *a1, char *a2, size_t a3)
{
  return strcat(a1, a2);
}
```

根本沒檢查長度!

Not Safe!

LLM 產生幻覺

```
void sub_11F7(const char *a1) {
  size_t v2;
  char s[56];

  s[0] = 0;
  v2 = 50 - strlen(s);

  printf("Append username '%s'\n", a1);
  printf("Remaining space: %zu bytes.\n", v2);
  sub_11C9(s, a1, v2);
  printf("Logged: %s\n", s);
}
```

username ok

remaining_space ok

~~safe_append~~

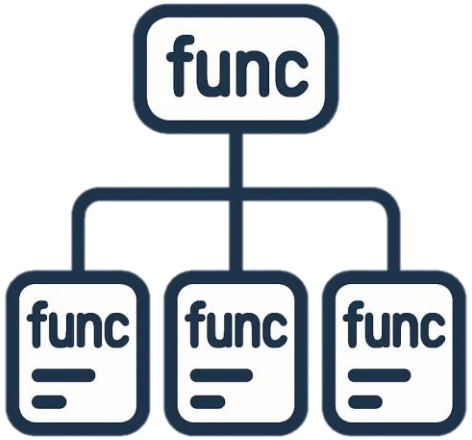
證據不充分

LLM 產生幻覺!

核心問題

- 核心問題其實是 **Context 不足**，導致 LLM 做出錯誤判斷

以函式為單位

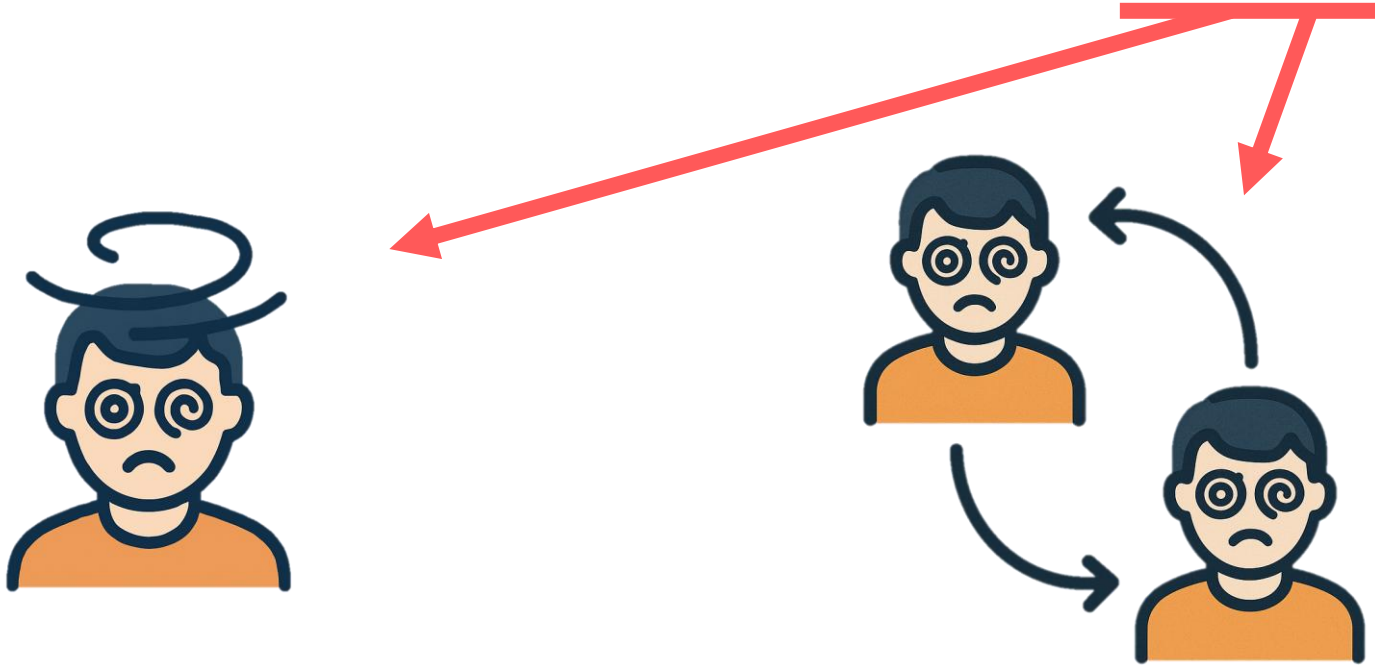


無法一次看整隻程式

沒有老師傅的經驗

核心問題

- 核心問題其實是 Context 不足，導致 LLM 做出**錯誤判斷**



LLM 產生**幻覺**

幻覺會傳遞

我們的方法

LLM 逆向工程

雪拉比

雪拉比可利用其操控時間的能力，將枯萎的植物回復到它們最健康、最有活力的時間點，從而達到「還原」的效果。

Cite: Pokémon

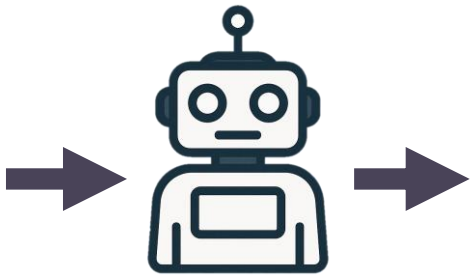
提供更多線索 (更多 Context)

```
void sub_11F7(const char *a1) {
  size_t v2;
  char s[56];

  s[0] = 0;
  v2 = 50 - strlen(s);

  printf("Append username '%s'\n", a1);
  printf("Remaining space: %zu bytes.\n", v2);
  sub_11C9(s, a1, v2); // hint: library func strcat detected
  printf("Logged: %s\n", s);
}
```

不需要完全依賴 LLM
老師傅有一些線索可以提供給 LLM

驗證 LLM 的判斷

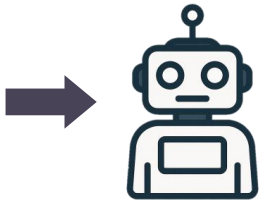
```
void sub_11F7(const char *a1) {
  size_t v2;
  char s[56];

  s[0] = 0;
  v2 = 50 - strlen(s);

  printf("Append username '%s'\n", a1);
  printf("Remaining space: %zu bytes.\n", v2);
  sub_11C9(s, a1, v2); // hint: library func strcat detected
  printf("Logged: %s\n", s);
}
```



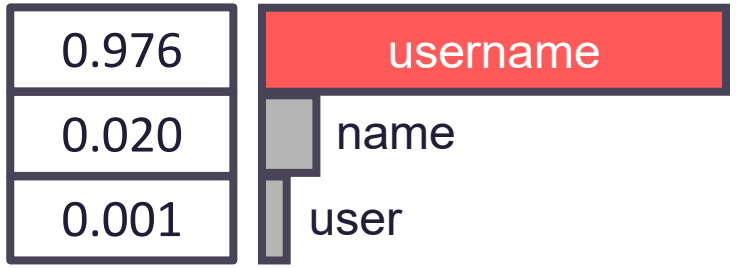
```
{
    "variables": [
        {
          "original_name": "a1",
          "new_name": "username"
        },
        {
          "original_name": "v2",
          "new_name": "remaining_space"
        },
        {
          "original_name": "sub_11C9",
          "new_name": "safe_append"
        },
    ]
}
```

驗證 LLM 的判斷: Token Probability

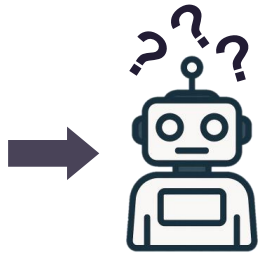
…s":[{"original_name":“a1","new_name":"



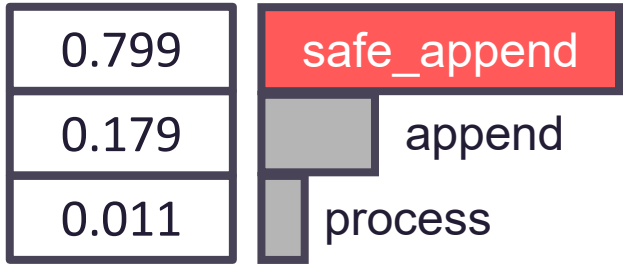
LLM 信心十足



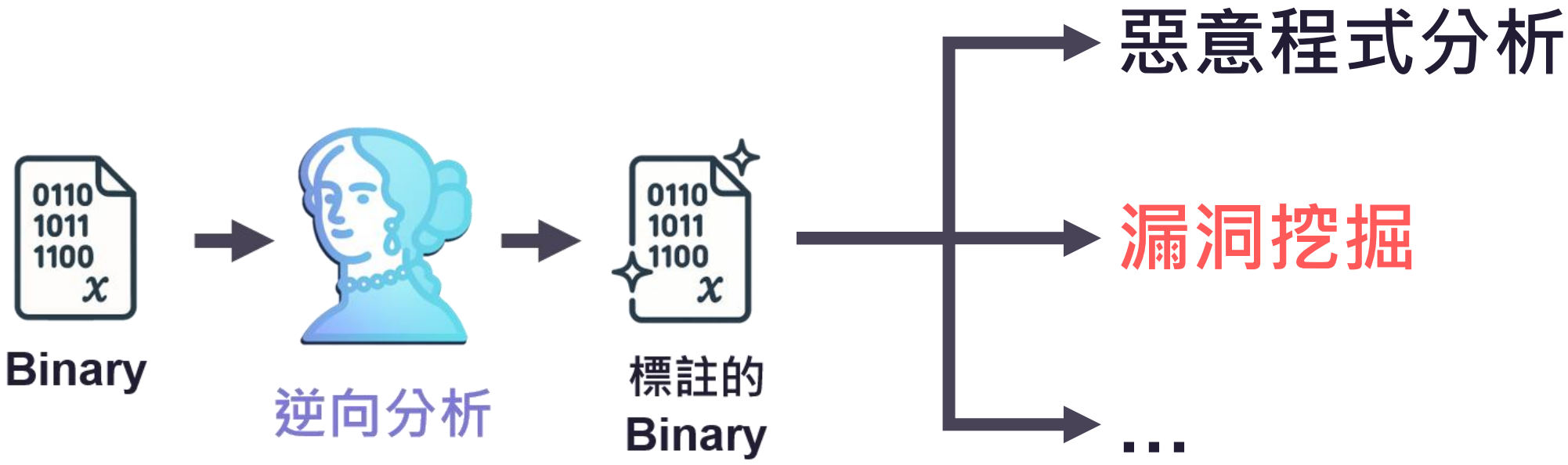
…iginal_name":“sub_11C9","new_name":"



LLM 沒有把握



有不同的應用情境


0110
1011
1100
x
Binary
逆向分析
0110
1011
1100
x
標註的
Binary
惡意程式分析
漏洞挖掘
...

有不同的應用情境


0110
1011
1100
x
Binary
逆向分析
0110
1011
1100
x
標註的
Binary
漏洞識別
REPORT
漏洞報告
漏洞利用

Vulnerability Discovery by LLM

共同研究： Eason

請問這段程式碼中是否存在漏洞？

```
void copy(char *input) {
    char buffer[10];
    strcpy(buffer, input);
}
```

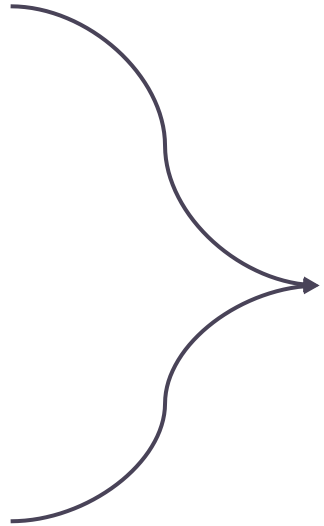
上下文的資訊將會嚴重影響結果

Unsafe

```
void getUserInput() {
    char userInput[100];

    printf("請輸入一個字串: ");
    fgets(userInput, sizeof(userInput), stdin);

    copy(userInput);
}
```

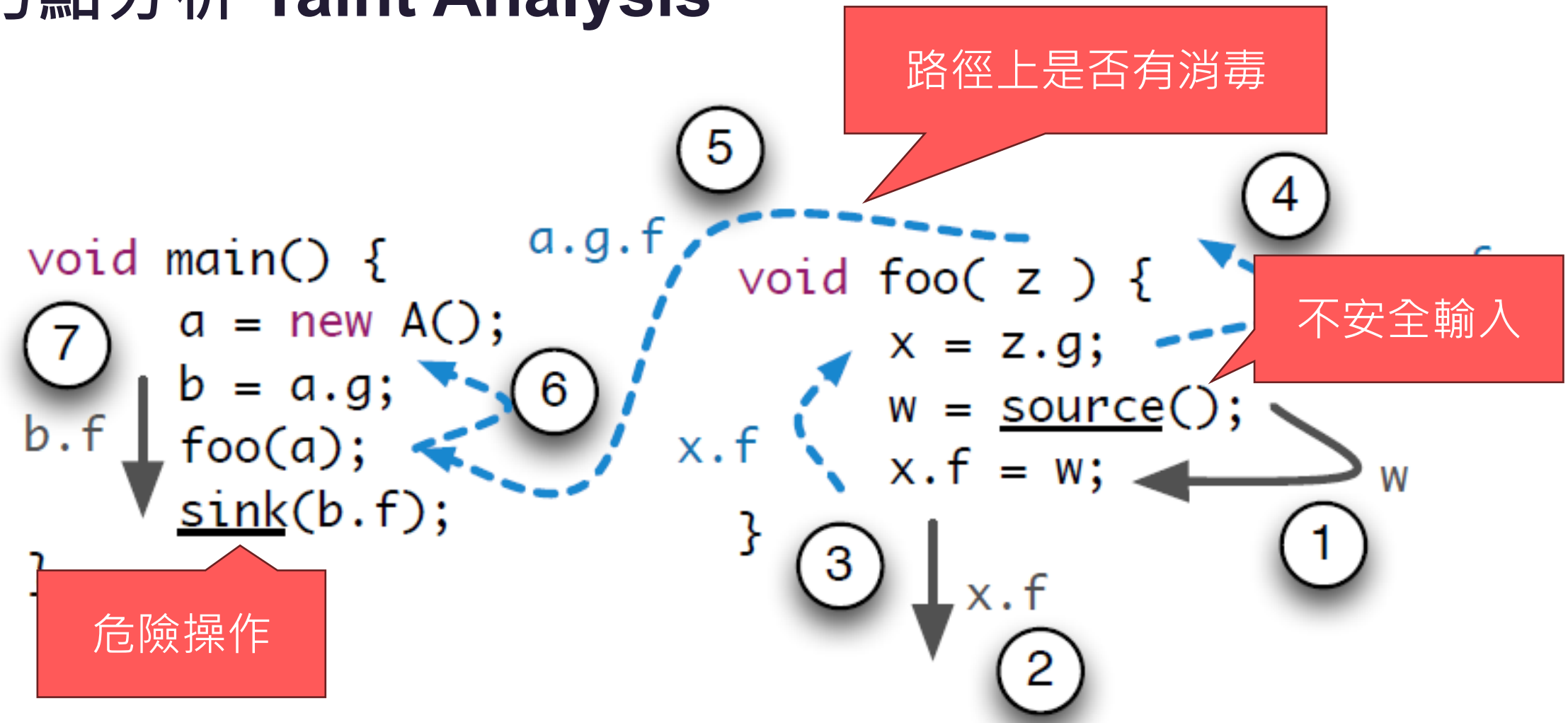
Safe

```
void getUserInput() {
    char userInput[100];

    printf("請輸入一個字串: ");
    fgets(userInput, sizeof(userInput), stdin);

    if (strlen(userInput) < 10) {
        copy(userInput);
    } else {
        printf("錯誤: 輸入的字串太長! 長度必須小於 10 個字元.\n");
    }
}
```



```
void copy(char *input) {
    char buffer[10];
    strcpy(buffer, input);
}
```

污點分析 Taint Analysis



https://blogs.uni-paderborn.de/sse/tools/flowdroid/

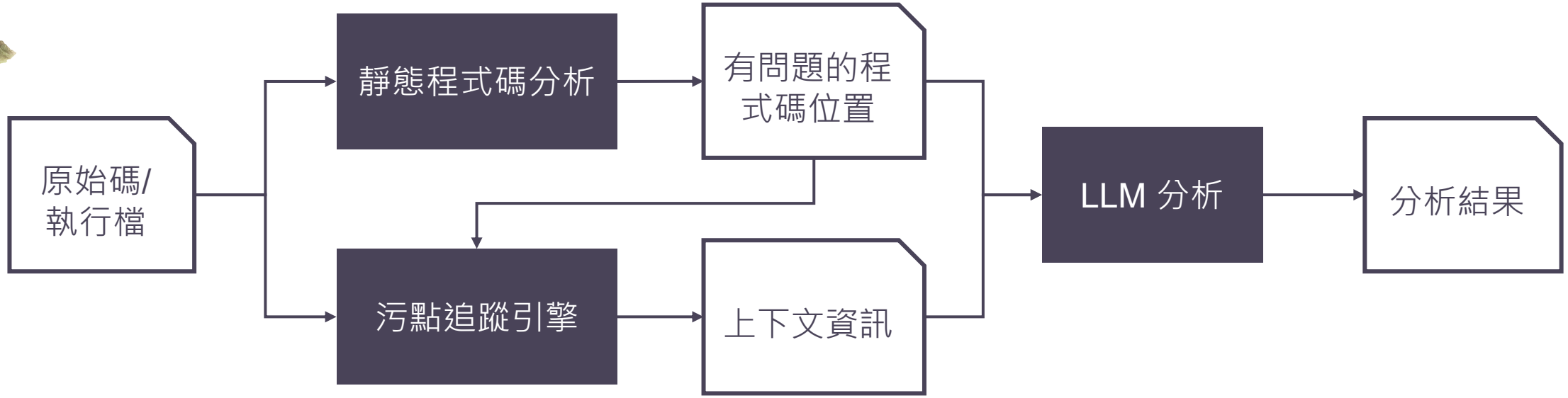
LLM 挖洞系統

地鼠 Diglett

無情地挖洞機器人

Cite: Pokémon

架構



Cite: Pokémon

在真實情境中，分析深度該設多少？

```
void copy(char *input) {
    char buffer[10];
    strcpy(buffer, input);
}
```

```
void d(char *input) {
    copy(input);
}
```

```
void c(char *input) {
    d(input);
}
```

```
void b(char *input) {
    c(input);
}
```

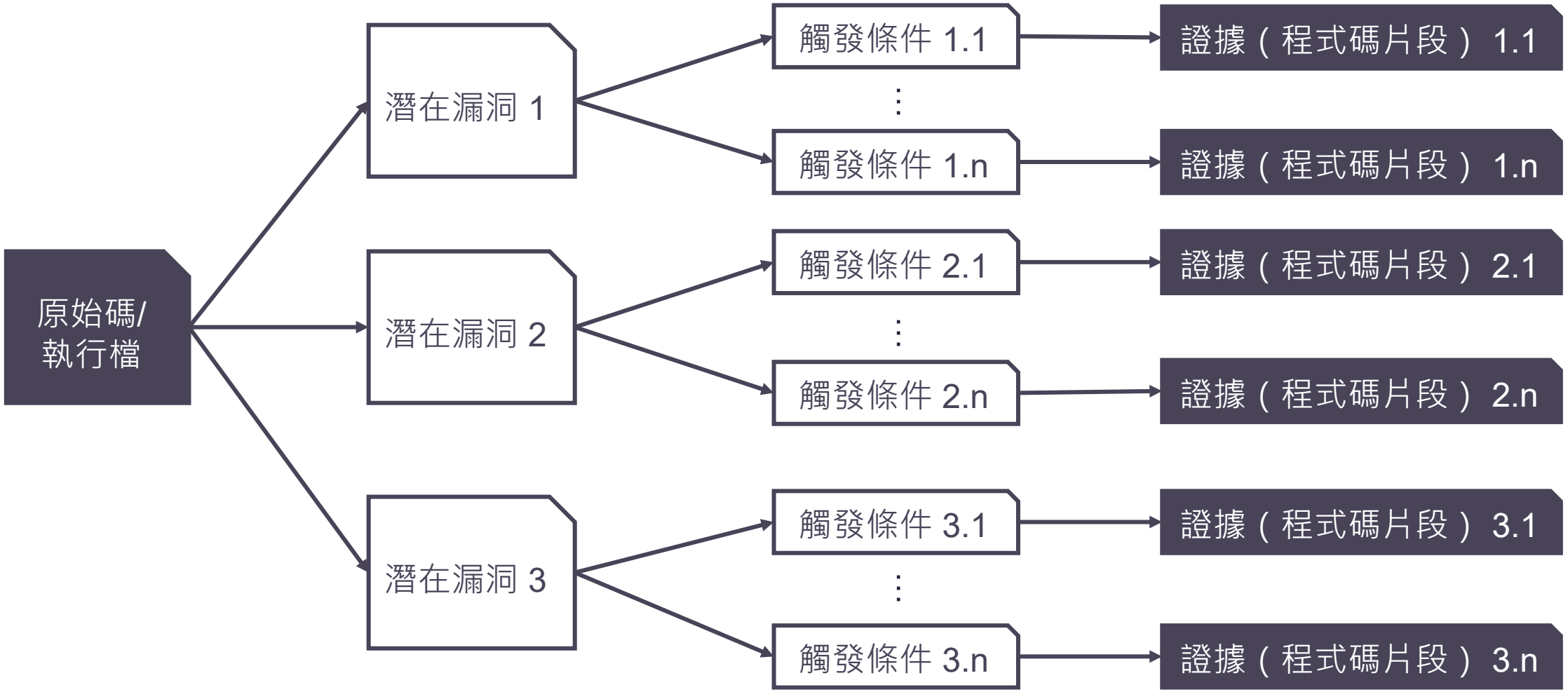
```
void a() {
    char userInput[100];
    fgets(userInput, sizeof(userInput), stdin);
    b(userInput);
}
```

何不讓 AI Agent 自行取證 

小模型直接使用 MCP 找漏洞遇到的挑戰

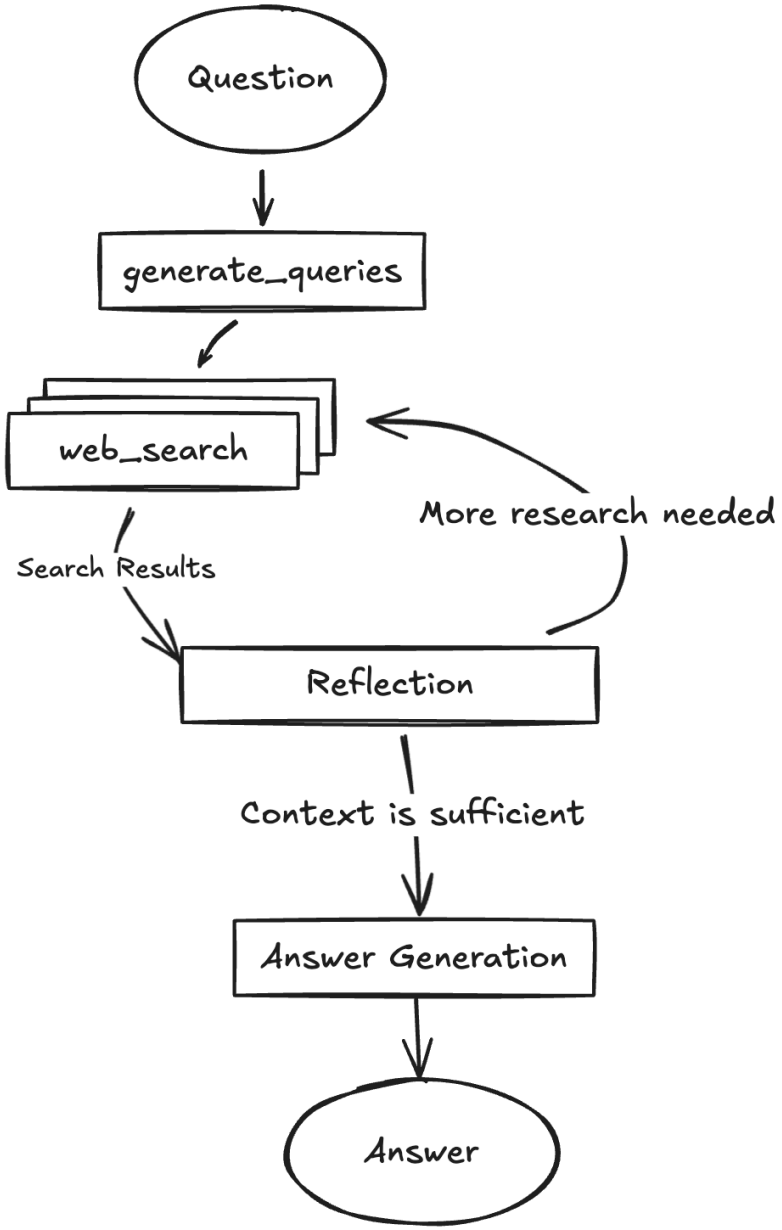
- 缺乏系統：憑感覺找漏洞
- 專注力不夠：Context Window 不夠大時，容易失焦
- 討好人類：不確定時，偏好回答有漏洞

威脅建模：Attack Tree



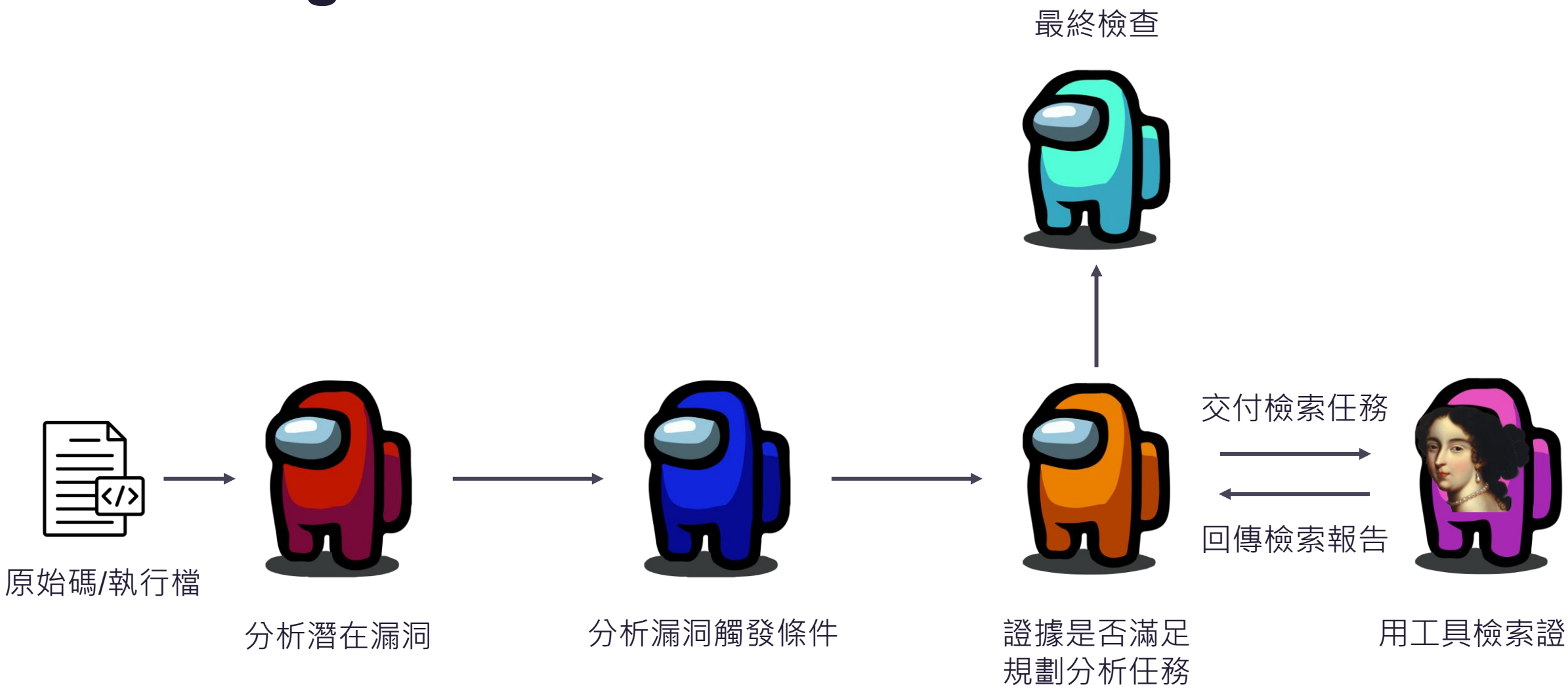
Deep Research

- 依照使用者的請求產生 Query
- 透過 Reflection 反思是否有資訊缺口
  - 有，則繼續新的查詢
  - 無，產生最終答案



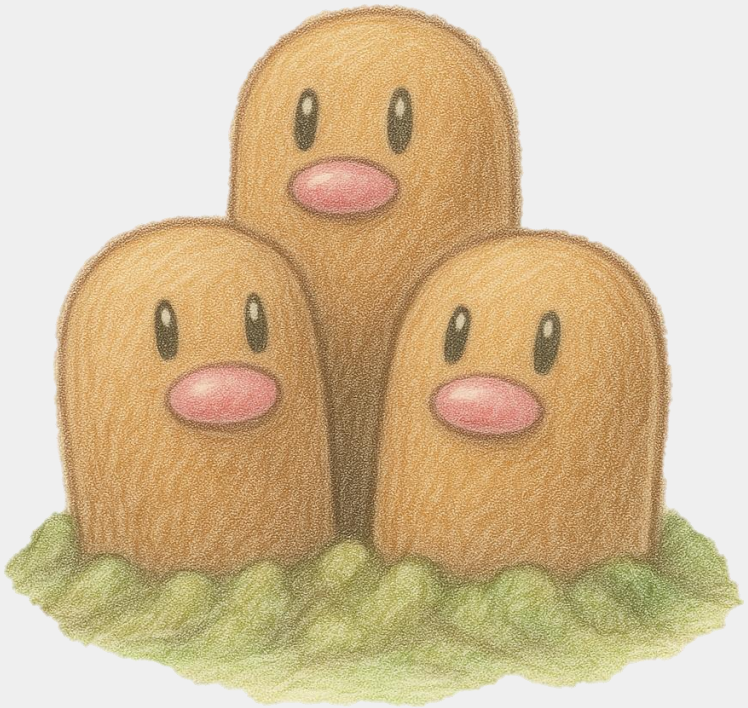
https://github.com/google-gemini/gemini-fullstack-langgraph-quickstart

Multi-Agent Mode



Cite: Among Us

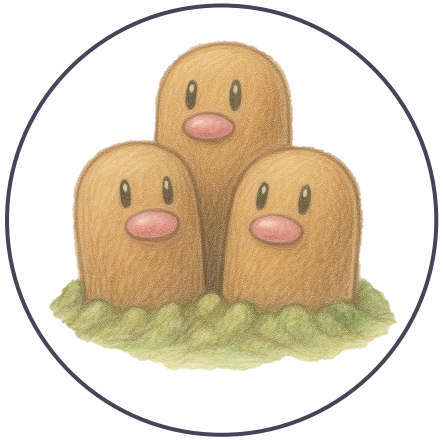
LLM 挖洞系統 2.0



三地鼠

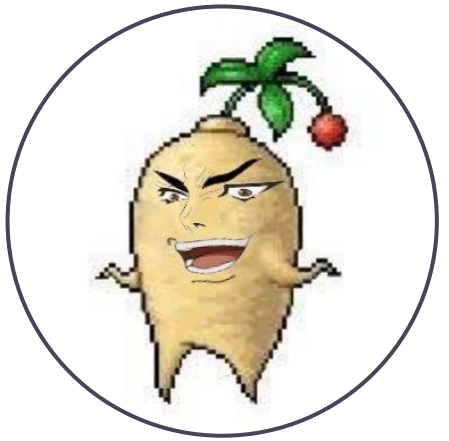
三隻無情地挖洞機器人

Cite: Pokémon



xxx 函數處理 Gzip 標頭時的錯誤處理邏輯，當 inflate 函數返回錯誤並嘗試重新解析輸入流為 Gzip 格式時，它未能充分驗證 FEXTRA 欄位長度和 FNAME/FCOMMENT 欄位的終止符，導致越界讀取漏洞。

函數內根本沒有在解析 Gzip 結構，推論是 LLM 在幻覺了！



確認存在 Gzip 結構解析，且漏洞確認存在。

Cite: Pokémon

```
if ( v10 == 1 && v15 && !v3 && *(_BYTE *)v24 == 31 && *(_BYTE *)(v24 + 1) == 0x8B )
{
    ...
    if ( (v16 & 4) != 0 )
        v21 += *(unsigned __int16 *)(v24 + 10);
    if ( (v16 & 8) != 0 )
    {
        for ( i = v21; *i; ++i )
        ;
        v21 = i + 1;
    }
    if ( (v16 & 0x10) != 0 )
    {
        for ( j = v21; *j; ++j )
        ;
        v21 = j + 1;
    }
    ...
}
```

尋找字串的結束符號 \0，但不檢查長度

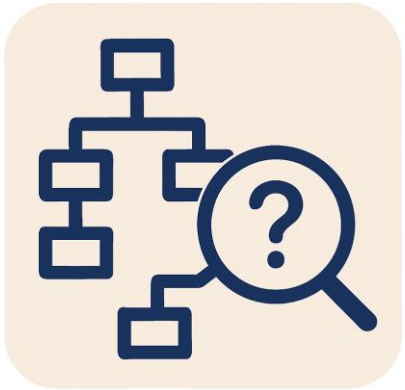
FLG (FLaGs)

bit 0	FTEXT
bit 1	FHCRC
bit 2	FEXTRA
bit 3	FNAME
bit 4	FCOMMENT
bit 5	reserved
bit 6	reserved
bit 7	reserved

https://www.rfc-editor.org/rfc/rfc1952.html

挑戰

初始分析的缺漏

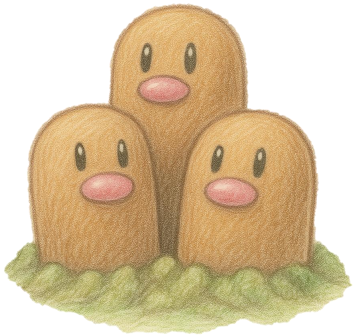


證據檢索策略不穩定

工具選擇障礙

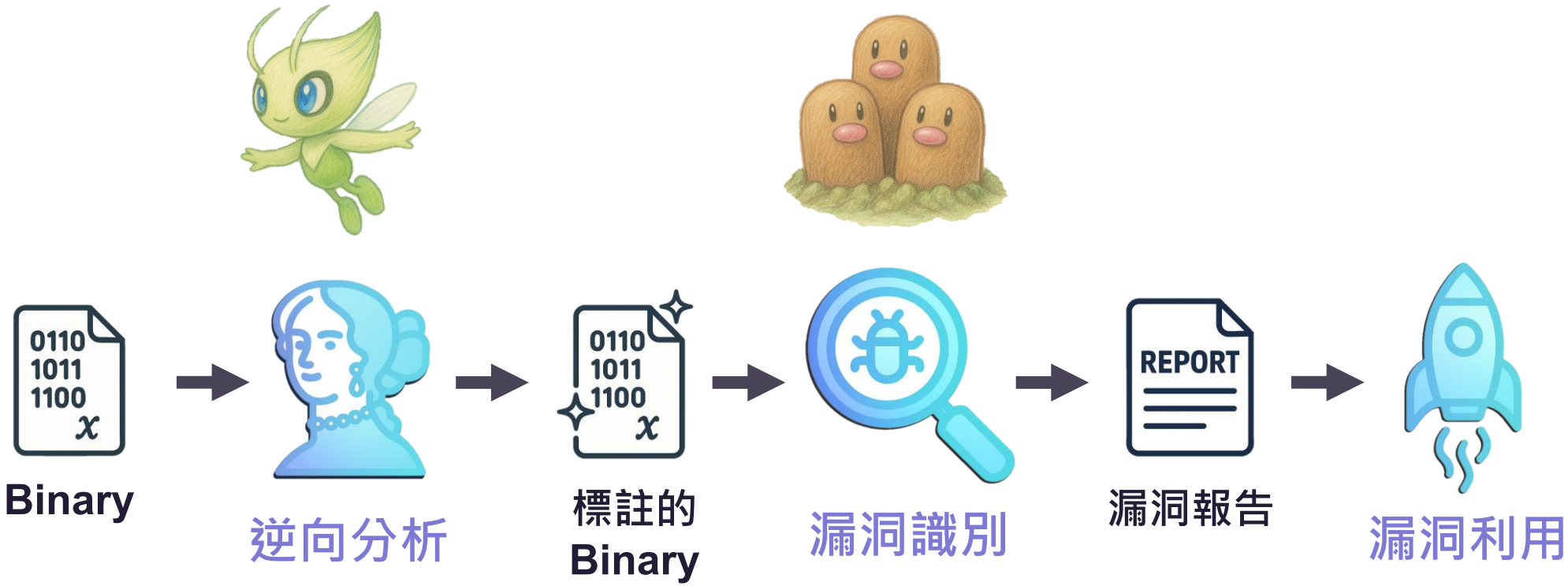
非預期解驗證

比較兩個方法



比較項目	Taint Analysis + LLM	Agent Mode LLM
耗費總 token 數	少	多（反覆搜集線索）
單次 context length	多（一次把所有資訊一並提供）	少（只提供必要資訊）
找線索的彈性	壞（由人工初始決定分析深度）	好
找線索的穩定性	高	低（LLM-driven）
消耗運算資源與時間	少	多

Cite: Pokémon



Cite: Pokémon

Takeaways

- Garbage In, Garbage Out
  - Context 不足會導致 LLM 有幻覺
- 逆向工程策略
  - 老師傅的經驗可以幫助 LLM 克服幻覺
  - 阻止幻覺擴散，需要驗證 LLM 的信心度
- 漏洞分析策略
  - 利用污點分析與 Agent mode 自主蒐證來強化 context 的搜集