

# Case Analysis

## A. Case I (False IR Strategy)

In Task 2 of the benchmark "ZGSF Linux 1", the "False Response Strategy" results in a typical failure.

### False IR Strategy (LLMs)

Use the following methods to systematically identify potential flag information in the system:

1. Search for all possible flag files in the system:

```
find / -type f -name "flag*" 2>/dev/null
```

2. Inspect memory or processes for hidden information:

```
strings /dev/mem | grep -i "flag"
```

3. Analyze network traffic and scheduled tasks:

```
tcpdump -A -i eth0 | grep "flag"
```

4. Examine attack scripts and environment variables:

```
env | grep "flag"
```

IR should adopt a step-by-step clue analysis process instead of relying solely on keyword searches, as some clues may lie beyond the files.

This highlights a key limitation of LLMs in devising effective strategies for IR tasks. To address this, we introduced IRT in IRCOPilot to prioritize IR tasks. When the task lacks clear information, it will be executed sequentially in accordance with the order in the "Incident Response Procedures."

### IRCopilot-Planner

IRT:

1. Incident Response Objectives (Linux) - [To-do]
  - 1.1 Hacker's IP address - (To-do)
  - 1.2 Flag 1 - (To-do)
  - ...
2. Incident Response Procedures - [To-do]
  - 2.1 Review Command History - (To-do)
  - 2.2 Investigate Sensitive Directories - (To-do)
  - 2.3 Analyze System Logs - (To-do)
  - ...

-----

Task selection: 2.1 Review Command History

1. View the command history of the current user.

```
$ history $
```

2. Looking directly at the ".bash history" file may provide more information, especially after the user has logged out.

```
$ cat ~/.bash history $
```

## B. Case II (False Commands Generation)

We present a specific example of a failure caused by the "False Command Generation" in Task 6 of "XuanJI Where-1S-tHe-Hacker" within the benchmark. When searching for hidden accounts, directly using the "net user" command fails to provide complete information. This case shows that LLMs may generate incorrect commands, which contradict facts and lead to task failures.

### False Commands Generation (LLMs)

Hidden Accounts Created by Hackers

Check the system account list:

1. Run net user to view the list of accounts and check for any suspicious accounts.
2. Run net user <username> to view detailed information.

In IRCOPILOT, the approach should involve first analyzing sub-tasks and then generating multiple solutions, specifically by checking Regedit or using the WMIC command.

## **IRCopilot-Generator**

To complete the task, we need to check in detail. Here are the specific steps and commands to perform this task:

1. Hidden accounts are typically recorded in the registry at the following location:

```
$ ...\\SAM\\SAM\\Domains\\Account\\Users $
```

This path stores all user SIDs and related information.

2. The WMIC can quickly list all accounts and their SIDs:

```
$ wmic useraccount get name,sid $
```

## **C. Case III (False Command Generation by IRCOPILOT)**

We present an example from the "XuanJI Where-1S-tHe-Hacker" benchmark, specifically Task 9, which involves identifying the moment the hacker reads a stored key from a file. This task was ultimately marked as failed, and the failure is directly attributable to a critical sub-task where IRCOPILOT failed to generate the correct forensic command. To diagnose the root cause of this error, we employed Gemini 2.5 pro, a model not used during the initial experiments, to evaluate the content generated by IRCOPILOT.

## **IRCopilot-Planner**

## 1. Incident Response Objectives (windows) - [To-do]

...

### 1.9 Reserved key read time - (To-do)

#### 1.9.1 Investigate registry access logs - (To-do)

...

Task selection: 1.9.1 Investigate registry access logs - (To-do)

To execute this task, I will review the Windows registry access logs to identify any unauthorized access or modifications, focusing on key registry paths that store system configurations and credentials. This involves searching for unusual access patterns or changes that could indicate tampering or the use of a reserved key by the attacker.

## IRCopilot-Generator

...

### 1. Access Event Viewer:

Open the Event Viewer to review security logs for registry access events. You can do this via the GUI or using PowerShell:

```
$ Get-EventLog -LogName Security | Where-Object { $ .EventID -eq 4657 } $
```

Event ID 4657 is logged when registry value modified.

Review these entries to identify any unauthorized changes.

### 2. Filter for Specific Registry Key Access:

If you are looking for access to specific registry keys, filter the logs accordingly. For example, if you are looking for access to a specific registry key path:

```
$ Get-EventLog -LogName Security | Where-Object { $ .EventID -eq 4657 -and $ .Message -like "*RegistryPath*" } $
```

Replace ““RegistryPath”“ with the specific path.

...

The evaluation revealed that the command suggested by IRCOPILOT was fundamentally flawed. It proposed using Event ID 4657, which tracks general object access events. This ID is too broad and does not specifically capture cryptographic key operations, leading to missed or irrelevant logs that make it impossible to pinpoint the exact moment of key access. In contrast, the analysis by Gemini 2.5 pro confirmed that Event ID 5058 is the appropriate choice, as it specifically logs key file operations (e.g., reading a stored cryptographic key) in the Windows Security event log, which would have enabled accurate identification of the event.

## Gemini 2.5 pro

### ## Evaluation

The model recommended the wrong Event ID (4657 instead of 5058). The suggested command checks for registry modifications, while the objective was to find the read time for a cryptographic key.

### ## Analysis of the Error

1. Over-generalization: The model treated the task as a routine registry check, providing a generic but inaccurate answer.
2. Ignoring Context: It failed to recognize the keyword "Reserved key" and therefore missed its link to the specific cryptographic Event ID (5058).
3. Logical Mismatch: Recommending an ID for "modification" (4657) to accomplish a "read" task shows a lack of precision, even in a general context.

## D. Case IV (Key Information Ignored by IRCOPILOT)

### IRCopilot-Analyst

...

#### Analysis:

1. Attacker's IP: The traffic indicates communication from '192.168.20.1' to '192.168.20.144'.
2. Flag 1: The HTTP GET request to 'flag1' resulted in a response containing 'flag1{Network@ 2020 Hack}'.
3. Webshell Connection: The POST request to 'index.php?user-app-register' contains PHP code, suggesting a potential Webshell or malicious script.
4. Further Investigation: The POST request content involves PHP functions that manipulate server settings, potentially indicating a backdoor or malicious activity.

#### Updated IRT:

1. Incident Response Objectives (linux) - [To-do]
  - 1.1 Attacker's IP - (192.168.20.1)
  - 1.2 Web system administrator plain text password - (To-do)
  - 1.3 URL of the first Webshell connection - (/index.php?user-app-register)
  - 1.4 Webshell connection parameter - (To-do)
  - 1.5 Subsequent upload of Webshell name - (To-do)
  - 1.6 Flag 1 - (flag1{Network@ 2020 Hack})

...

We present an example from the "ZGSF Linux2" benchmark, specifically Task 4, which involves identifying the webshell connection parameter. This task was ultimately marked as failed, and the failure is directly attributable to Key Information Ignored, as the model overlooked the relevant data in the request body.

This failure stemmed from the model's inability to parse the request's basic structure. It focused exclusively on the complex PHP payload (the "value"), thereby ignoring the parameter name Network2020 (the "key"), which was the actual answer. A successful analysis would have first identified the request's key=value format to extract the parameter

before proceeding  
to analyze the payload's content.

## Gemini 2.5 pro

### ## Evaluation

The model failed to extract the webshell connection parameter (Network2020) from the POST request. It incorrectly marked this item as (To-do), while the correct answer was explicitly available in the key=value structure of the request body.

### ## Analysis of the Error

1. Over-focus on Payload: The model dedicated all its analytical resources to the complex PHP malware (the value), completely ignoring the parameter name (the key) that was used to carry it.
2. Ignoring Structural Context: It failed to parse the fundamental key=value format of the POST request body, thereby missing the entry point parameter for the entire attack.
3. Pattern Matching Failure: The parameter name Network2020 does not fit the typical pattern of webshell parameters like cmd or eval, which likely caused the model's heuristics to overlook it.

## E. Case V (Limitation in Reverse)

Here, we offer an example task highlighting the limitations of LLMs and IRCOPILOT: the Squid Game malicious file analysis challenge from TryHackMe. This challenge demands sophisticated dynamic and static analysis, encompassing data flow analysis, VBA macro handling, obfuscated content decoding, and the seamless integration of multiple tools.

## Results of Oledump

```
ubuntu@ip-10-10-114-110:~/Desktop/maldocs$  
oledump.py attacker1.doc  
1: 114 '\x01CompObj'  
...  
8: M 9852 'Macros/VBA/ThisDocument'  
...
```

## LLMs

```
oledump.py -s 8 -v attacker1.doc  
oledump.py -s 1 -d attacker1.doc
```

The output from these two commands generates voluminous, obfuscated data streams that are challenging for current LLMs to parse, often necessitating intricate manual analysis. We anticipate this will be a key challenge addressed by the next generation of LLM-based software analysis.