Generating API Parameter Security Rules with LLM for API Misuse Detection

Jinghua Liu, Yi Yang, Kai Chen, Miaoqian Lin

INSTITUTE OF INFORMATION ENGINEERING, CAS

BACKGROUND

APIs are widely used in software development to enable reuse and speed up development, offering diverse features.

Cryptography
5478 APIs
26.4k stars (Github)

Database
294 APIs
7.2k stars (Github)

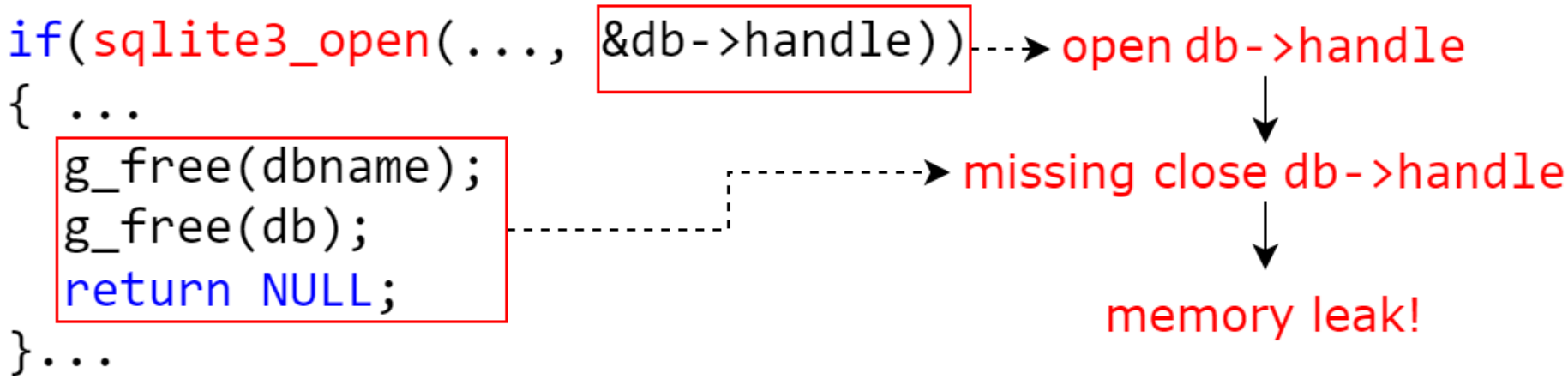
BACKGROUND

Violating API parameter security rules (APSRs) can cause security issues

An APSR of *sqlite3_open*

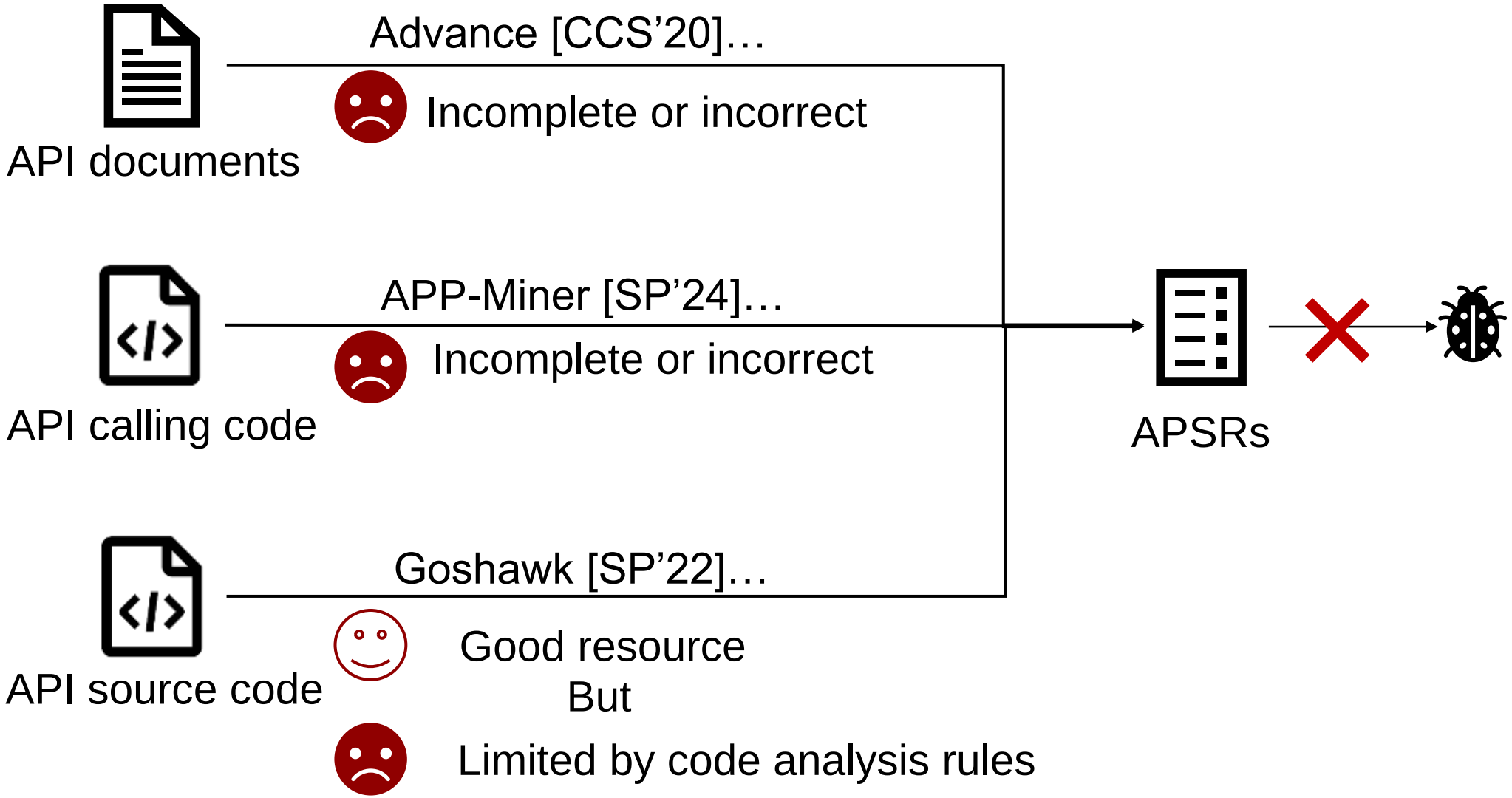
Users should release the second parameter when no longer needed

API misuse

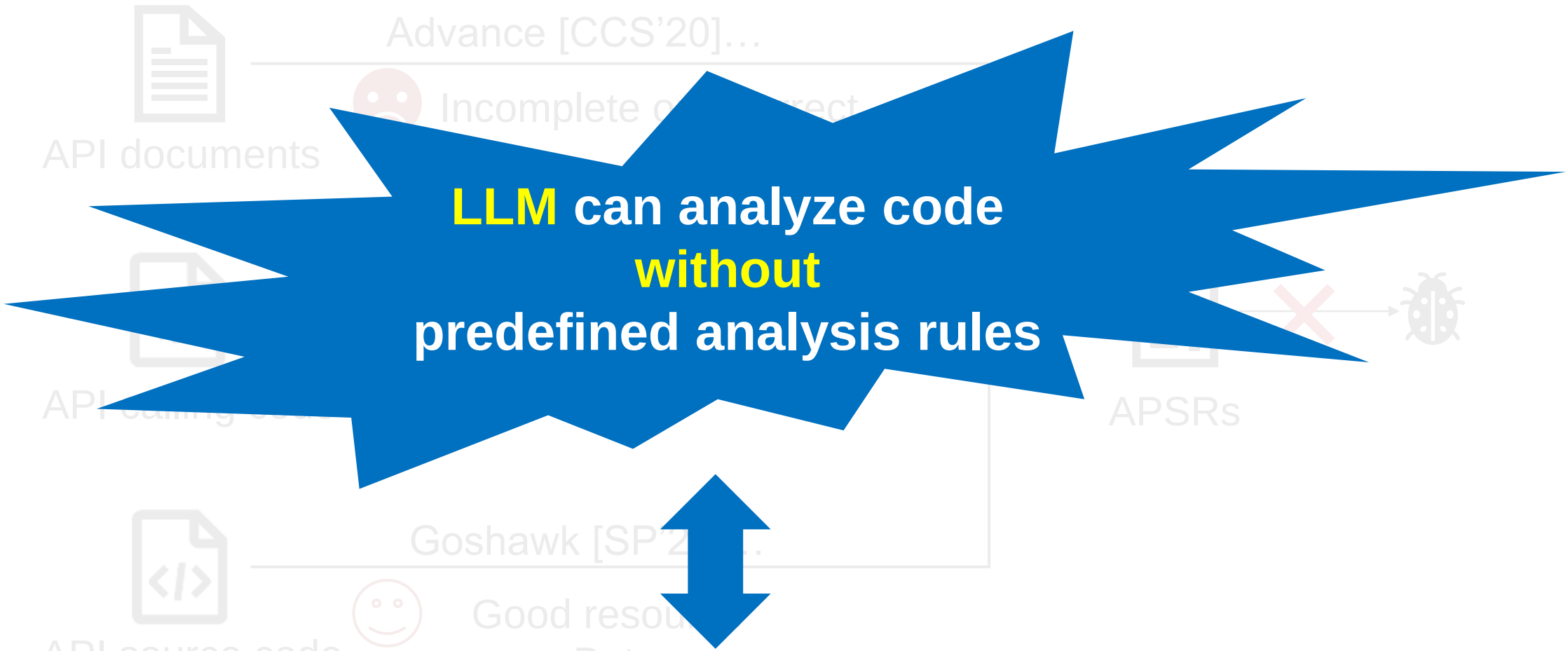


How to detect API misuse caused by incorrect parameter use?

Limitation of Previous Work



Motivation



Limited by code analysis rules

Challenge

- **Challenge-1: Incorrect APSRs**

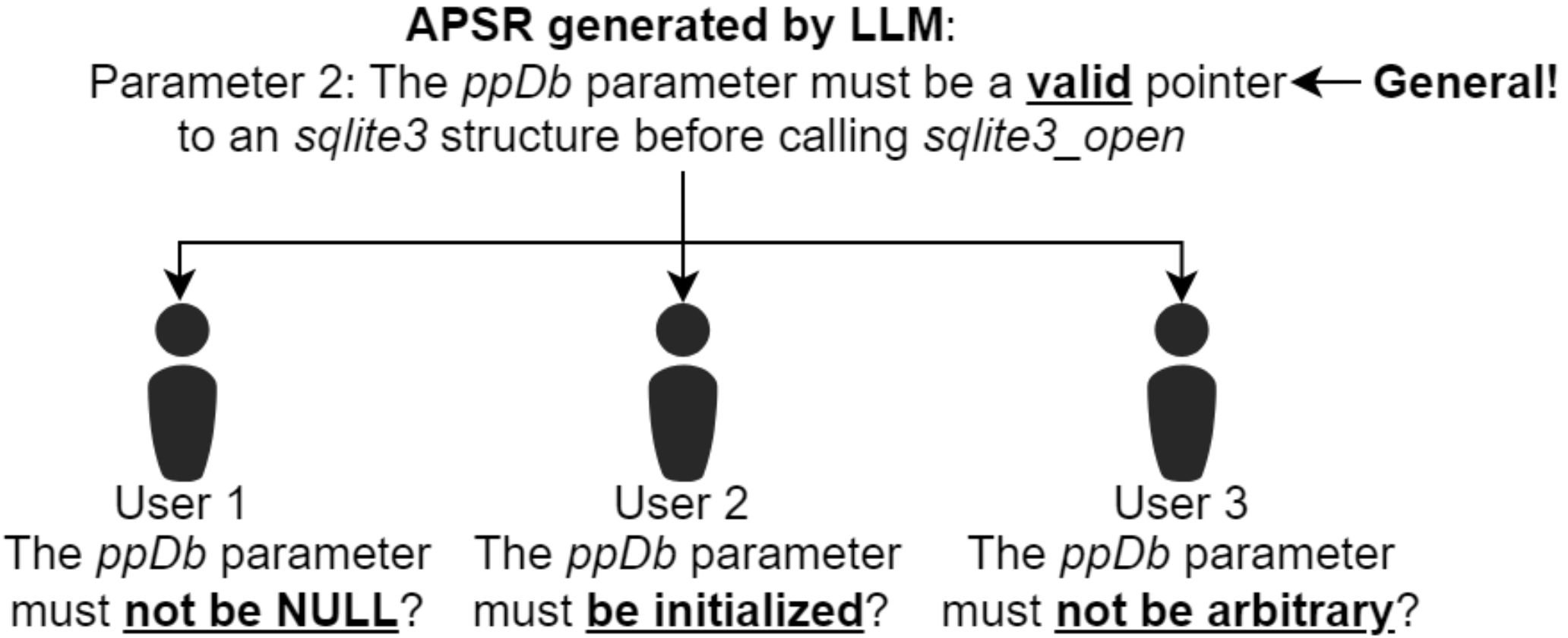
Please generate parameter-related security rules for the **free** API (a C standard library API). Please answer concisely.

The parameter passed to free must be a **non-NULL** pointer

Incorrect API misuse!

Challenge

- **Challenge-2: Overly general APSRs**



Insights

```c
#include <stdlib.h>
int main() {
    int *ptr = NULL;
    free(ptr);
    return 0;
}
```

Execute

Success

```c
#include <stdlib.h>
int main() {
    int *ptr = (int*)malloc(sizeof(int));
    if (ptr != NULL) {
        *ptr = 42;
        free(ptr);
        free(ptr);
    }
    return 0;
}
```

Execute

```
==19232==ERROR: … attempting double-free…
    #0 0x7f698741040f in __interceptor_free
    #1 0x560436144242 in main
```

Violate correct APSRs can lead to runtime errors

Solve Challenge-1

Insights

Correct
API Calling Code

```
sqlite3 *db;

int status = sqlite3_open(filename, &db);
```

API Misuse

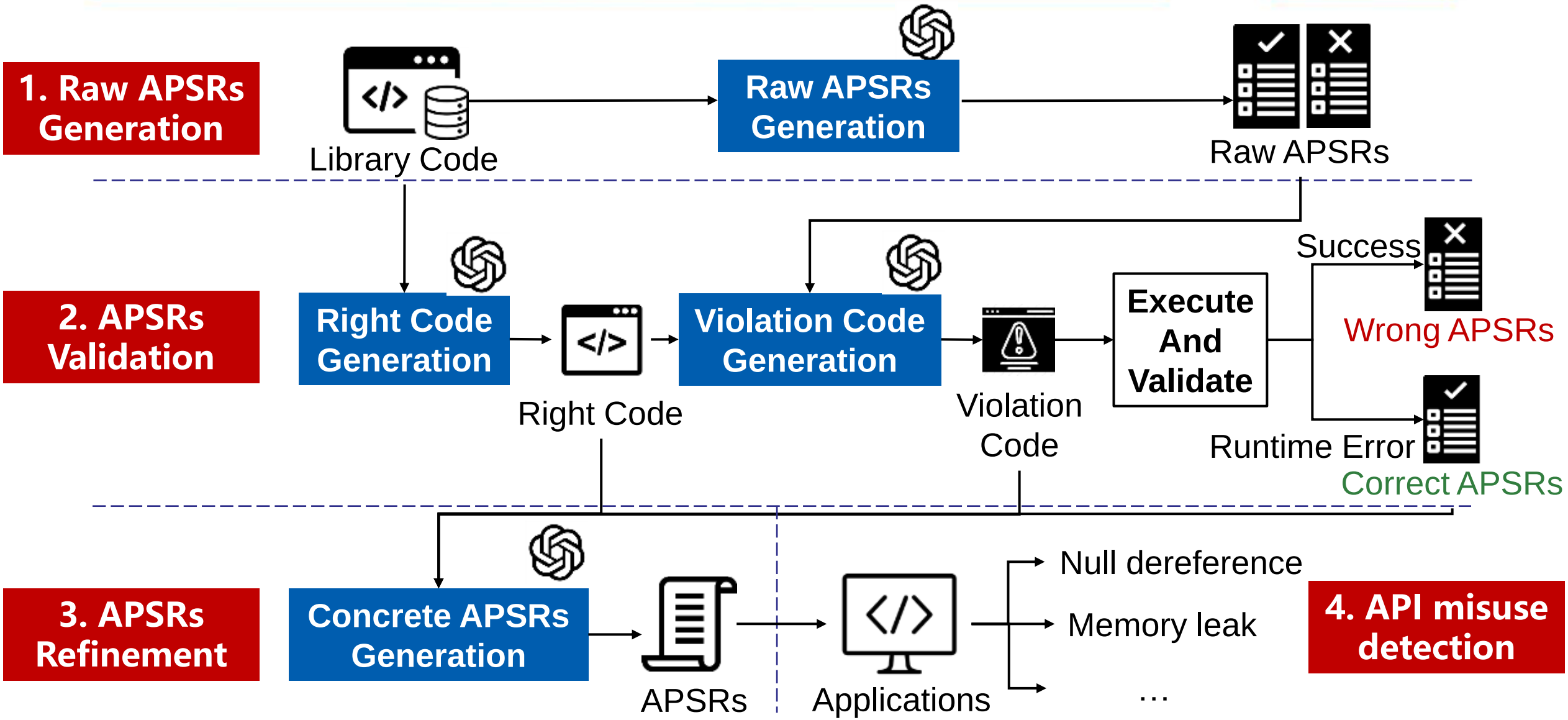
```
sqlite3 *db = NULL;

int status = sqlite3_open(filename, db);
```

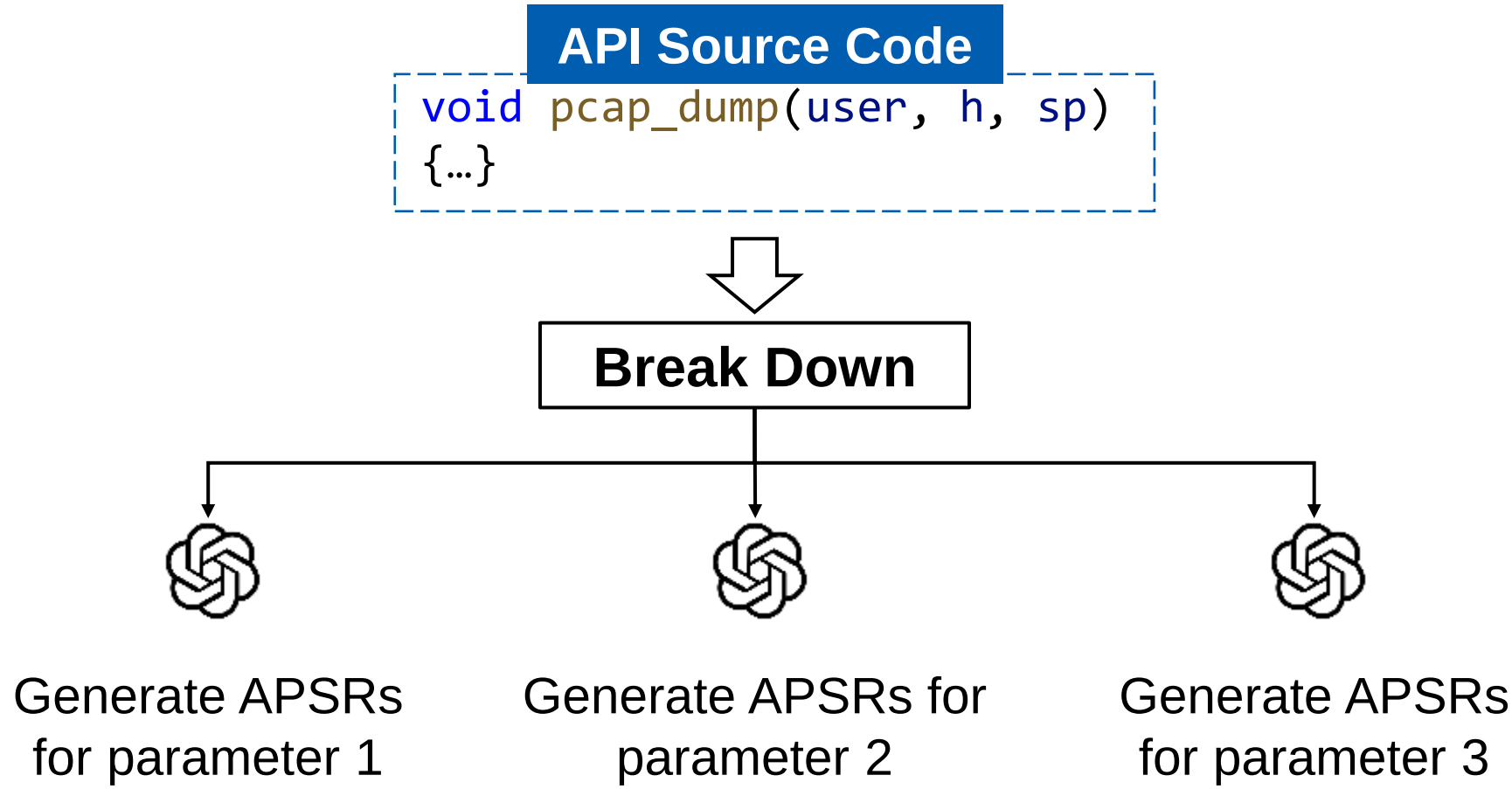
Concrete APSR : parameter 2 must not be NULL

Concrete APSRs should describe violation operations → Solve **Challenge-2**

Overview


1. Raw APSRs Generation
Library Code
Raw APSRs Generation
Raw APSRs
2. APSRs Validation
Right Code Generation
Right Code
Violation Code Generation
Violation Code
Execute And Validate
Success
Wrong APSRs
Runtime Error
Correct APSRs
3. APSRs Refinement
Concrete APSRs Generation
APSRs
Applications
Null dereference
Memory leak
…
4. API misuse detection

1. Raw APSRs Generation



1. Raw APSRs Generation

:

API Source Code

```
void pcap_dump(u_char
*user, const struct
pcap_pkthdr *h, const
u_char *sp)
{
    ......
    fwrite(sp, h->caplen…);
}
```

Instructions

Please analyze the **source code** of the API **<pcap_dump>** and all the information provided before it. Then, respond with the following tasks:

Task 1: Provide a description of the **functionality** of the function.

Task 2: **Locate the code** related to the **Parameter <1 user>.**

Task 3: Identify the **security rules** that should be observed for the **Parameter <1>** to prevent misuse of the API **<pcap_dump>**. For each rule, include **code snippets that demonstrate a violation** of the rule..

:

1. **Parameter 1 must not be NULL**

Raw APSRs

2. APSRs Validation

Execution feedback-checking

Raw APSR of *sqlite3_open*:

Parameter 1: The filename should be validated to ensure it refers to a legitimate, existing database file

Violation code snippet

```
...
const char *filename="non_existing.db";
 ... = sqlite3_open(filename, NULL);
...
```

⇩ Execute

==**Error**: The signal is caused by a READ memory access.
==Hint: address points to the zero page.

⇩

Parameter 1: The filename should be validated to ensure it refers to a legitimate, existing database file

2. APSRs Validation

Execution feedback-checking

Raw APSR of *sqlite3_open*:

Parameter 1: The filename should be validated to ensure it refers to a legitimate, existing database file

Violation code snippet

```
...
const char *filename="non_existing.db";
 ... = sqlite3_open(filename, NULL);
...
```

invalid!

NULL pointer dereference!

Not related to the APSR!

Parameter 1: The filename should be validated to ensure it refers to a legitimate, existing database file

How to determine if the code only violates the target APSRs?

2. APSRs Validation

Right Code Generation

:

API Source Code

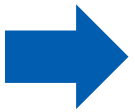
```
void pcap_freealldevs
(pcap_if_t*alldevs){
    ......
}
```

Instructions

Generate a complete code that calls the function in Linux

:

```
...
pcap_findalldevs(&alldevs..);
pcap_freealldevs(alldevs);
...
```



Right Code

2. APSRs Validation

Violation Code Generation and Validation

:

Right Code

```
pcap_findalldevs(&alldevs..);
pcap_freealldevs(alldevs);
```

Raw APSRs

```
Parameter 1:
must not be NULL
```

Instructions

Please modify the **right code** to **violate** this APSR

:

Violation Code

```
pcap_freealldevs(NULL);
```

⇩ Execute

Success

⇩

```
Parameter 1:
must not be NULL
```

Right Code

```
pcap_freealldevs(devs);
pcap_freealldevs(devs);
```

Raw APSRs

```
Parameter 1:
must be valid
```

Instructions

⇩ Execute

==ERROR:… double-free

⇩

```
Parameter 1:
must be valid
```

3. APSRs refinement

Code differential analysis to identify key operations

Right Code

```
....
fn = BIO_get_callback_ex(bio);
```

Violation Code

```
....
bio = NULL;
bio_method = BIO_s_bio();
 if (bio_method == NULL) {
    …
    return 123;
 }
bio_callback = BIO_get_callback_ex(bio);
```

3. APSRs refinement

Code differential analysis to identify key operations

Right Code

```
....
fn = BIO_get_callback_ex(bio);
```

Violation Code

```
....
bio = NULL;
bio_method = BIO_s_bio();
 if (bio_method == NULL) {
    …
    return 123;
 }
bio_callback = BIO_get_callback_ex(bio);
```

must not be NULL?

must not create other object before?

How to identify the key operations in complex modifications?

3. APSRs refinement

Violation Code1

```
....
BIO_free(bio);
bio = NULL;
fn = BIO_get_callback_ex(bio);
```

Violation Code2

```
....
bio = NULL;
bio_method = BIO_s_bio();
 if (bio_method == NULL) {
    …
    return 123;
 }
bio_callback = BIO_get_callback_ex(bio);
```

Execute

Runtime Error Message

======ERROR: AddressSanitizer:…
==The signal is caused by a READ memory access.
==Hint: address points to the zero page.
#0 … in BIO_get_callback_ex (/usr/local/lib64/libcrypto.so.3…)
#1 …in main …/BIO_get_callback_ex.c:50
…

3. APSRs refinement

Cluster the violation codes and identify the common operations

:

Right Code

Violation Code-1

Violation Code-2

Runtime Error Message

Instructions

1. Enumerate the **key differences** between the correct and error API calls.
2. Determine the **potential root cause** of the error based on the REM.
3. For **each identified difference**, evaluate its **probability** of being the primary root cause of the error based on all the information.
4. **Rank** these causes based on their likelihood.
5. For the most probable causes, **generates specific security rules** to **prevent such errors** in the future…

:

```
Parameter must not be NULL
```

Concrete APSRs

4. API misuse Detection

APSRs

Frequently pattern

Must not pass *[VALUE]*

…

Detection template

isNCheck(API)

…

Match

APSR

An APSR of **EVP_DigestInit**:
Parameter 1: The caller **Must NOT pass NULL** as the `ctx` parameter when calling `EVP_ DigestInit`

Detection Code

```
from FunctionCall target
where
 isTargetAPI(target)//instantiating the API name
 and isNCheck(target)// check whether there is a check
select target, target.getLocation().toString()
```

Evaluation Results

- Various types of APSRs on eight popular libraries
  - 8 types of APSRs, achieving a precision of 92.3%.
  - 210 new API misuse on 47 applications
  - 355 new APSRs

GPTAid generates new APSRs and detects new API misuse

Evaluation Results

- Comparison with SOTA

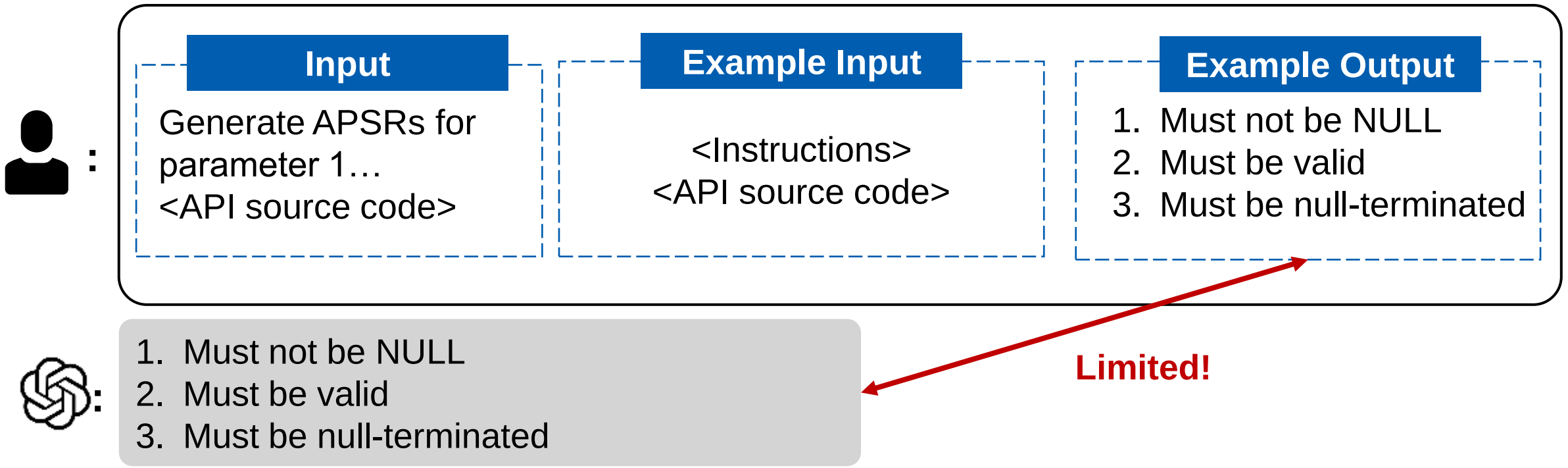
	GPTAid	Advance	IPPO	Goshawk	All
#APSRs	53	7	/	8	58
#Bugs	243	99	0	10	306

GPTAid covers more APSRs and bugs

Lessons for prompt design

☹ Insufficient information cause fabrication

☹ Examples might be harmful



Conclusion

- **A Novel approach** to generate accurate and concrete APSRs using LLM
- Detect **210 new bugs**
- Generate **355 new APSRs**
- Valuable **suggestions**
  - **prompt design**

- https://github.com/icy17/GPTAid/

Thank You

For Your Attention

INSTITUTE OF INFORMATION ENGINEERING, CAS