中国科学院软件研究所
Institute of Software Chinese Academy of Sciences

蚂蚁技术研究院
Ant Research

中国科学院大学
University of Chinese Academy of Sciences

PROMEFUZZ: A Knowledge-Driven Approach to Fuzzing Harness Generation with Large Language Models

ACM CCS 2025

Yuwei Liu*, Junquan Deng*, Xiangkun Jia, Yanhao Wang, Minghua Wang, Lin Huang, Tao Wei, Purui Su

PROMEFUZZ

Motivation

Library fuzzing requires programs known as **fuzzing harnesses** or **fuzz drivers** to serve as the entry point for the fuzzer.

```c
#include "../cJSON.h"
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
    CJSON *json;
    size_t offset = 4;
    int require_termination = data[1] == '1' ? 1 : 0;
    int buffered            = data[3] == '1' ? 1 : 0;

    json = CJSON_ParsewithOpts((const char *)data + offset, NULL, require_termination);
    if (json == NULL) return 0;
    if (buffered) printed_json = CJSON_PrintBuffered(json, 1, formatted);
    CJSON_Delete(json);
    return 0;
}
```

A Fuzzing Harness from the cJSON library

PROMEFUZZ

Motivation

The generated fuzzing harnesses are expected to meet:

① **Syntactically correct**: Correct initialization and function invocations to form a compilable program.

② **Semantically correct**: Valid call sequences, appropriate flag settings, and other implicit API constraints to ensure meaningful execution.

③ **High coverage**: Effectively exploring the library’s state space.

PROMEFUZZ

Motivation

Challenges: ① **Syntactic** ② **Semantic** ③ **Coverage**

Existing approaches:

- Use real-world consumer code ➔ **satisfy** ① + ②, **but limited on** ③
  - Slice API callers: *FUDGE@FSE'19* | *AFGen@S&P'24*
  - Merge API sequences: *FuzzGen@Sec'20* | *UTopia@S&P'23*

- Mutate API sequences ➔ **improve** ③, **but often break** ②
  - *APICraft@Sec'21* | *GraphFuzz@ICSE'22* | *Hopper@CCS'23* | *Rubick@Sec'23*

PROMEFUZZ

Motivation

Challenges: ① **Syntactic** ② **Semantic** ③ **Coverage**

Existing approaches:

- Leverage the power of LLMs ➔ **potential to achieve** ① + ② + ③ **?**
  - *PromptFuzz@CCS'24* | *CKGFuzzer@ICSE'25* | *OSS-Fuzz-Gen@Google*

PROMEFUZZ

Motivation

```c
// Signature:
pcap_t *pcap_create(const char *device, char *errbuf);
int pcap_activate(pcap_t *p);
int pcap_fileno(pcap_t *p);
int pcap_set_protocol_linux(pcap_t *p, int protocol);
int bpf_validate(const struct bpf_insn * f, int len);
int pcap_minor_version(pcap_t *p);

int LLVMFuzzerTestOneInput(const uint8_t* data, size_t size) {
    if(size<0) return 0;
    // Step 0: Initialize resources
    FILE *input_file_ptr = fopen("input_file", "wb");
    if (input_file_ptr == NULL) {return 0;}
    fwrite(data, sizeof(uint8_t), size, input_file_ptr);
    fclose(input_file_ptr);
    const char* input_file = "input_file";
    // Step 1: pcap_fileno
    pcap_t* pcap = pcap_create(input_file, NULL);
    pcap_activate(pcap);
    int pcap_fd = pcap_fileno(pcap);
    // Step 2: pcap_set_protocol_linux
    pcap_set_protocol_linux(pcap, pcap_fd);
    // Step 3: bpf_validate
    struct bpf_insn bpf_program[1];
    int bpf_len = sizeof(bpf_program) / sizeof(bpf_program[0]);
    int bpf_valid = bpf_validate(bpf_program, bpf_len);
    // Step 4: pcap_minor_version
    int minor_version = pcap_minor_version(pcap);
    // Step 5: Release resources
    pcap_close(pcap);
    return 0;
}
```

A Harness for `libpcap` Generated by PromptFuzz

NAME

pcap_create - create a live capture handle

SYNOPSIS

```
#include <pcap/pcap.h>

char errbuf[PCAP_ERRBUF_SIZE];

pcap_t *pcap_create(const char *source, char *errbuf);
```

DESCRIPTION

pcap_create() is used to create a packet capture handle to capture packets on a device (typically a network interface, see **pcap_findalldevs**(3PCAP) for a more detailed explanation). *source* is a string that specifies the capture device to open, in this function `NULL` means the same as the string "any". *errbuf* is a buffer large enough to hold at least `PCAP_ERRBUF_SIZE` chars.

The returned handle must be activated with **pcap_activate**(3PCAP) before packets can be captured with it; options for the capture, such as promiscuous mode, can be set on the handle before activating it.

The Documentation of `pcap_create`

infrastation on Nov 28, 2023 — Member

In that and similar man pages the clause "*errbuf* is assumed to be able to hold at least `PCAP_ERRBUF_SIZE` chars." implies the pointer must not be NULL.

On a related note, if you manage to find a valid vulnerability, please do as the issue template says. The public bug tracker is not for vulnerability reports.

infrastation closed this as not planned on Nov 28, 2023

Maintainer's Response

PROMEFUZZ

Motivation

This "confirmed bug" turns out to be a false positive — reveals LLM-based methods are still unable to achieve ① + ② + ③. **Why?**

- **Lack domain-specific knowledge** in harness generation.
- Require human assistance to **triage and validate crashes**.

★ Intuition: consider **how a human expert would gather knowledge and reason** when writing fuzzing harnesses.

PROMEFUZZ

Motivation

★ Intuition: consider **how a human expert would gather knowledge and reason** when writing fuzzing harnesses.

We propose PROMEFUZZ:

- Human reads code to find API signatures & data types
  ➥ PROMEFUZZ parses AST to extract code metadata (① **Syntactic**)

- Human consults documentations for usage examples and constraints
  ➥ PROMEFUZZ uses RAG to retrieve and summarize relevant documentation (② **Semantic**)

PROMEFUZZ

Motivation

★ Intuition: consider **how a human expert would gather knowledge and reason** when writing fuzzing harnesses.

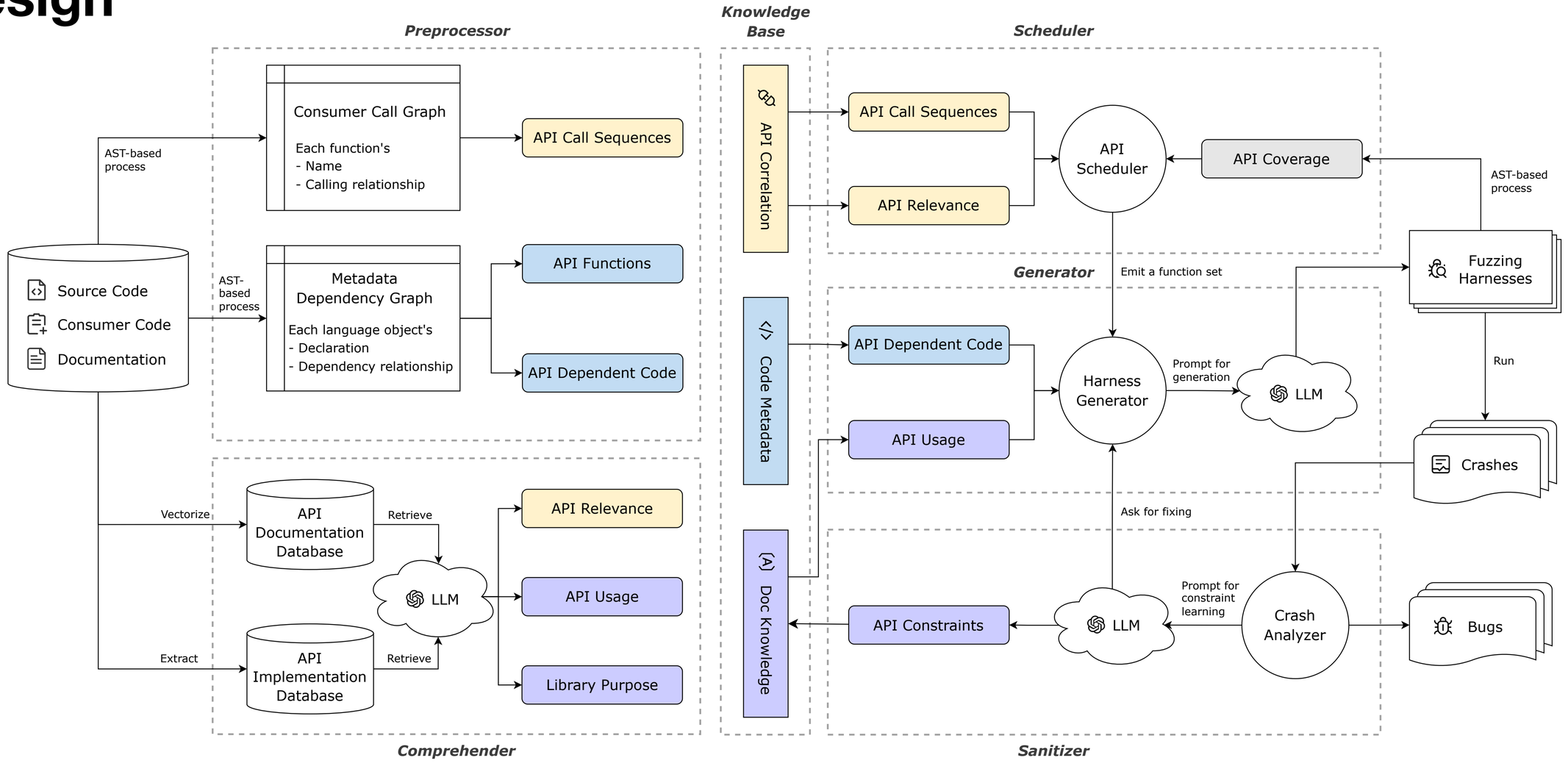
We propose PROMEFUZZ:

- Human adheres to usage conventions (e.g., init → use → cleanup)
  ➥ PROMEFUZZ calculates API correlations to schedule meaningful API sequences (① + ② + ③ **Coverage**)

- Human debugs and learns from errors
  ➥ PROMEFUZZ analyzes crashes to learn and avoid API misuses (① + ②)

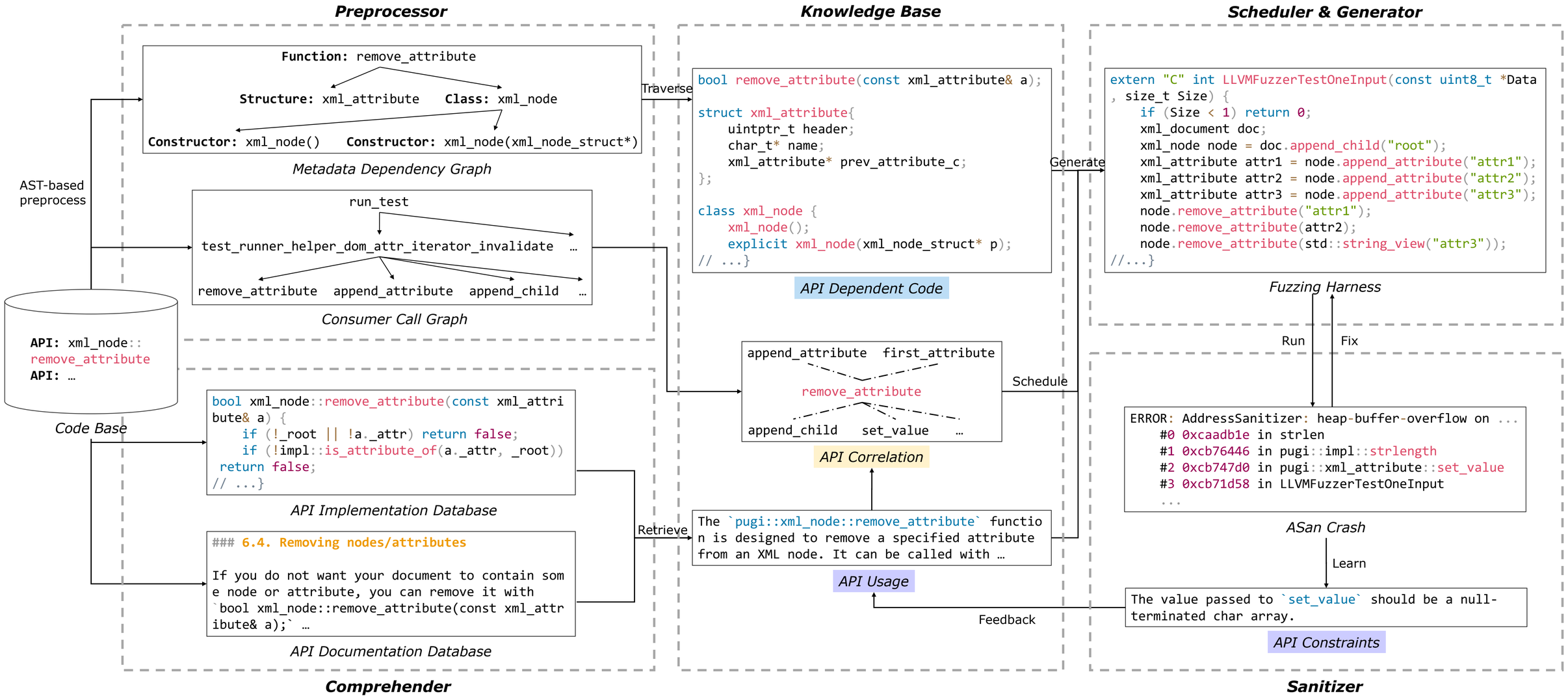
PROMEFUZZ

Design



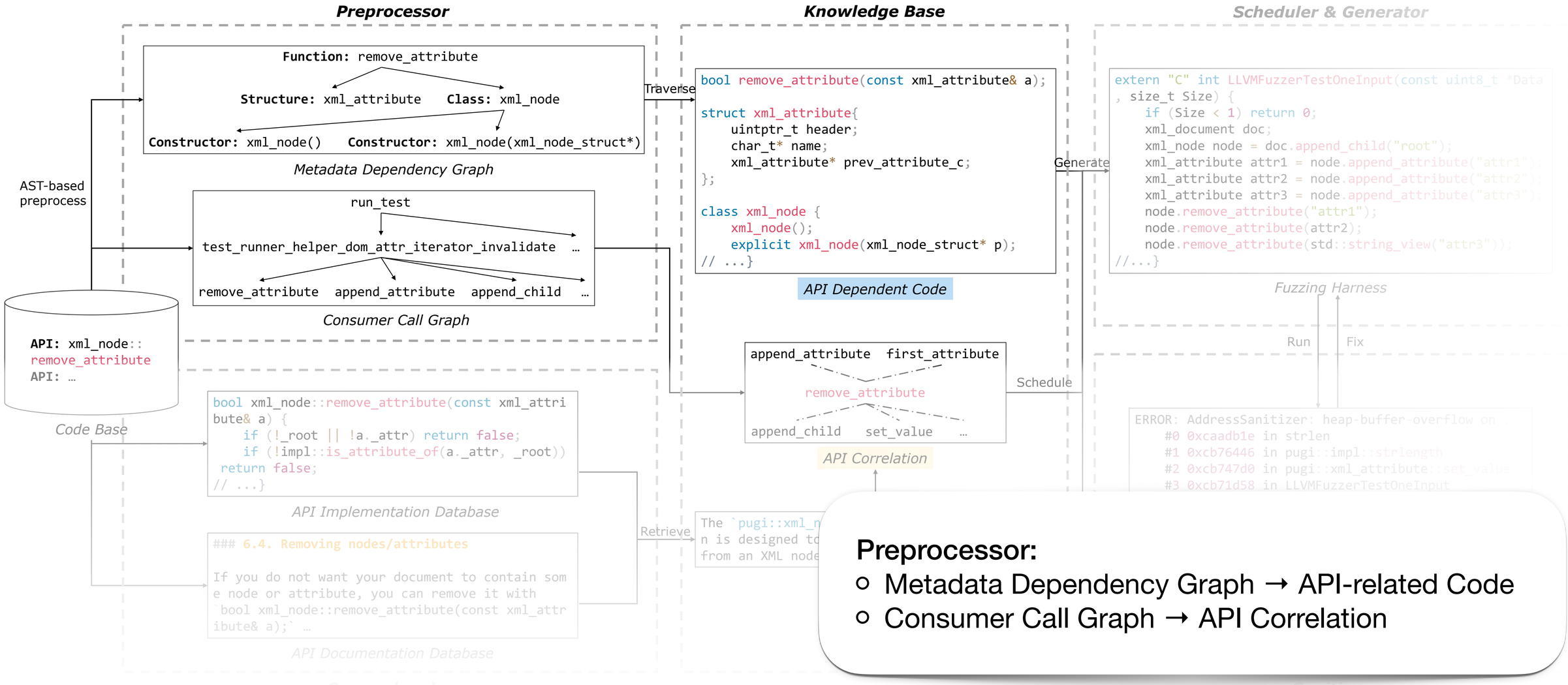
PROMEFUZZ

Design



PROMEFUZZ

Design



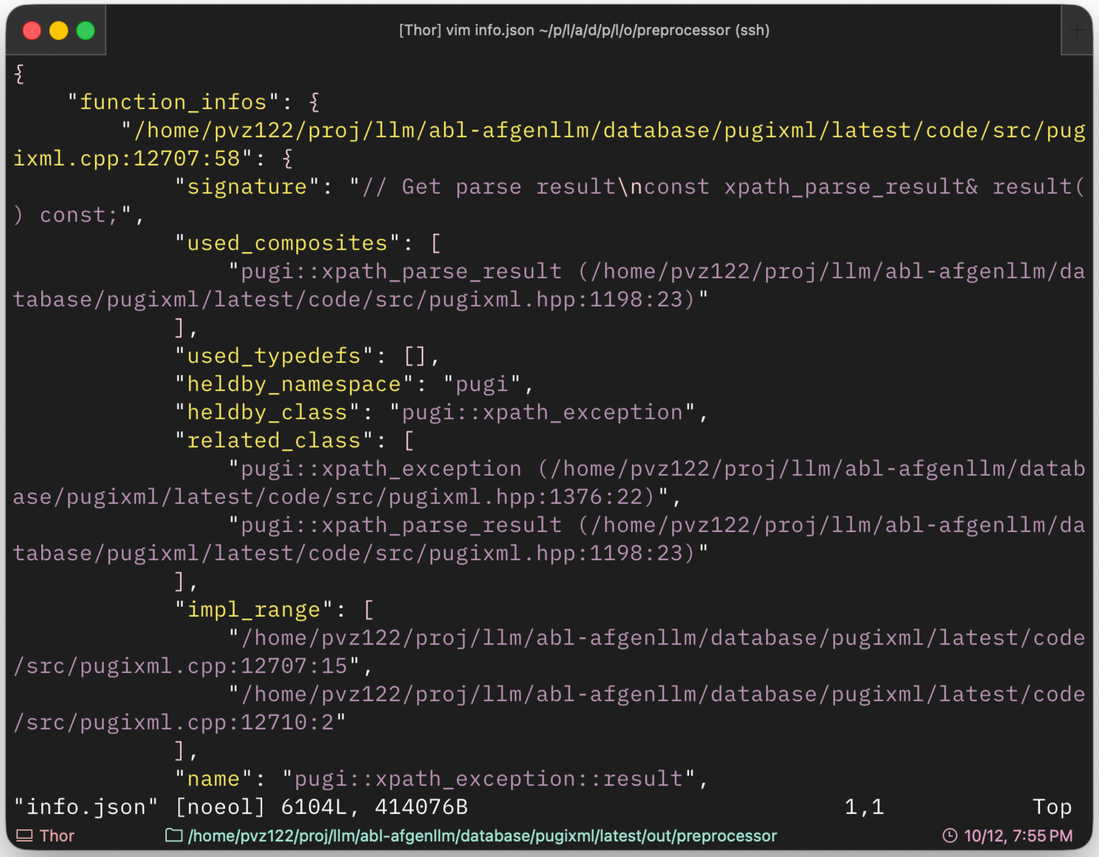
Preprocessor:

- Metadata Dependency Graph → API-related Code
- Consumer Call Graph → API Correlation

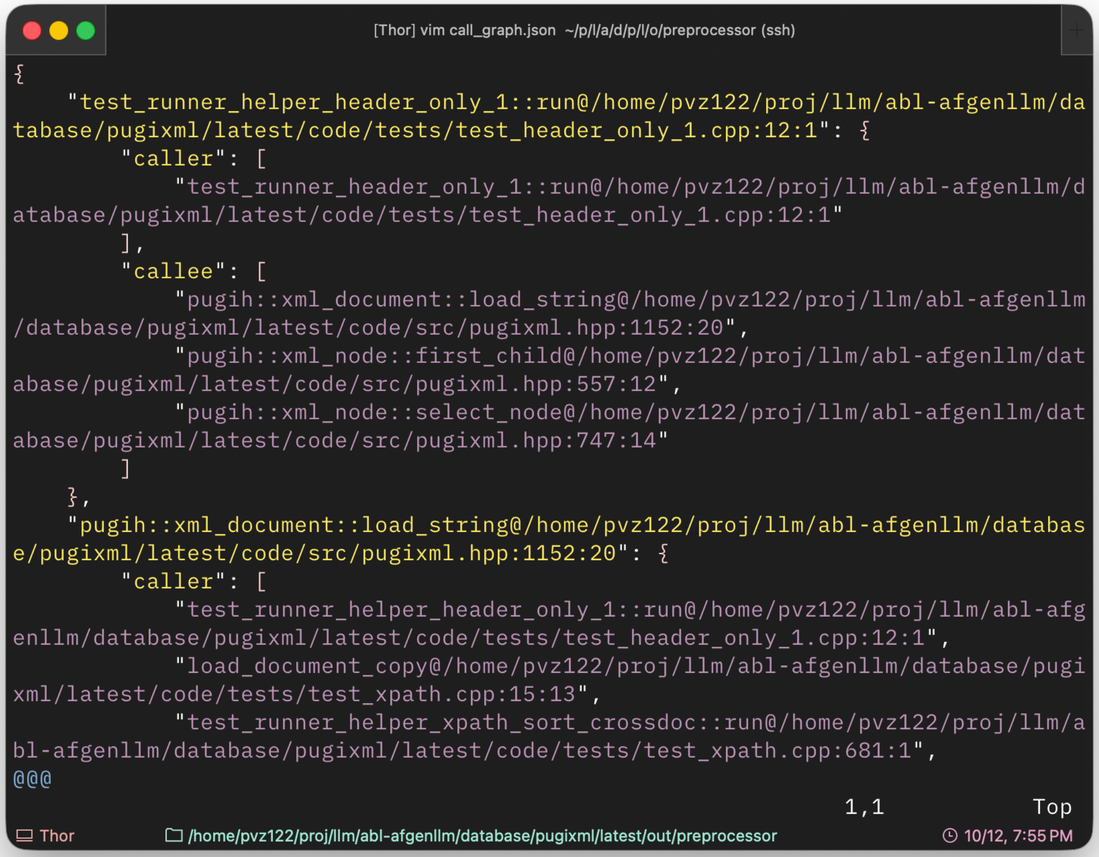
PROMEFUZZ

Design

Metadata Dependency Graph

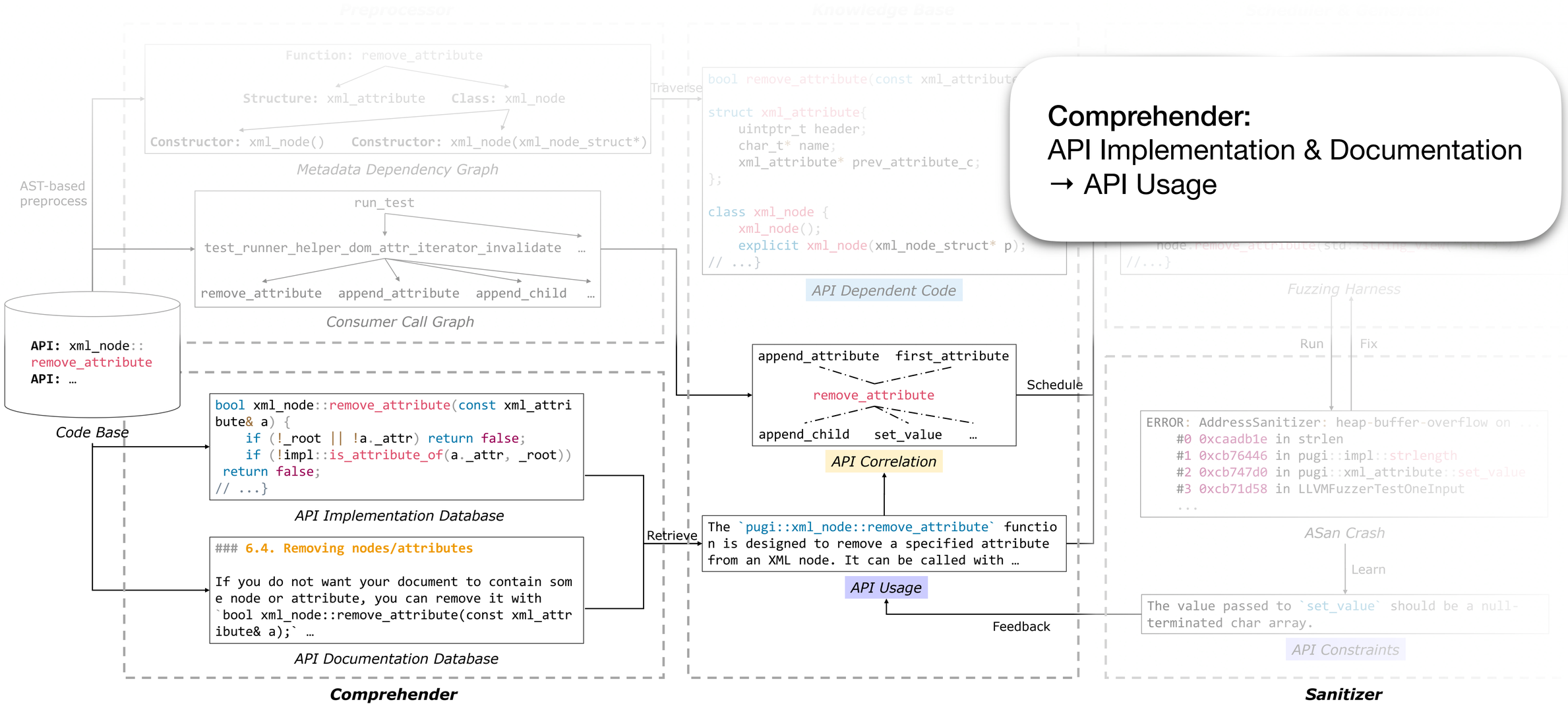


Consumer Call Graph



PROMEFUZZ

Design



Comprehender:
API Implementation & Documentation
→ API Usage

PROMEFUZZ
Design

Using Comprehender, PromeFuzz can extract the parameter constraints of `pcap_create` successfully.

TASK

You are required to identify a particular API function's purpose and usage according to the source code and documentation. Consider:
- What is the API function's design aim?
- How should the function be used?
- How does it behave in various scenarios?

Don't add uncertain information. Be concise (under 300 characters).

CONTEXT

```
# Information about `pcap_create`
## Function Implementation
```c
pcap_t *pcap_create(const char *device, char *errbuf) {
...
## Documentation Excerpts
1. Excerpt from mandoc.txt:
```
NAME
      pcap_create - create a live capture handle
DESCRIPTION
      pcap_create() is used to create a packet capture handle to look at packets on the network. source is a string that specifies the network device to open; on all supported Linux systems, as well as on recent versions of macOS and Solaris, a source argument of "any" or NULL can be used to capture packets from all network interfaces. The latter should not be confused with all available capture devices as seen by pcap_findalldevs(3PCAP), which may also include D-Bus, USB etc.
errbuf is a buffer large enough to hold at least PCAP_ERRBUF_SIZE
...
```

RESPONSE

The `pcap_create` function is designed to create a live capture handle for network packet capture. It initializes a `pcap_t` structure but does not activate the capture; activation is done via `pcap_activate`. The `source` parameter specifies the network device (e.g., "any" or NULL for all interfaces), and `errbuf` stores error messages, which holds at least PCAP_ERRBUF_SIZE characters.

PROMEFUZZ

Design

API Correlation Measurement

```
xml_attribute append_attribute(const char_t* name);

bool remove_attribute(const xml_attribute& a);
```

a) Type Relevance. The return type of `append_attribute` matches the parameter type of `remove_attribute`, enabling a data flow between them.

```
class PUGIXML_CLASS xml_node {
    xml_attribute append_attribute(const char_t* name);
    bool remove_attribute(const xml_attribute& a);
    // ...
}
```

b) Class Scope Relevance. `append_attribute` and `remove_attribute` are both methods of the class `xml_node`, allowing them to operate on the same XML node instance.

```
test_runner_helper_dom_attr_iterator_invalidate
```

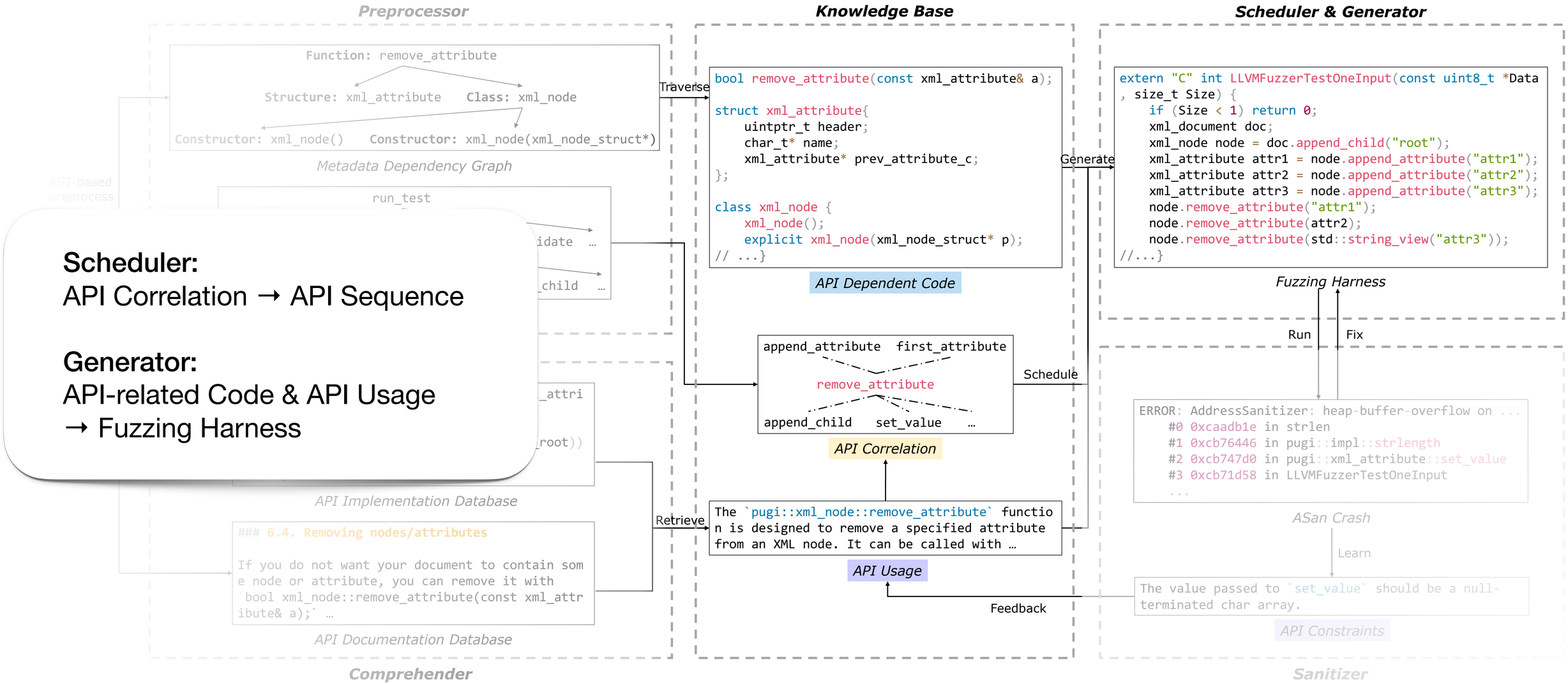
c) Calling Scope Relevance. `append_attribute` and `remove_attribute` are invoked by the same caller, resulting in a distance of 2 in the consumer call graph.

```
*API Function Purpose*
- `append_attribute`: Add a new attribute to an XML node.
- `remove_attribute`: Remove an attribute from an XML node.
```

d) Semantic Relevance. The purposes of `append_attribute` and `remove_attribute` are logically connected, leading the LLM to consider them correlated.

PROMEFUZZ

Design



Scheduler:
API Correlation → API Sequence

Generator:
API-related Code & API Usage
→ Fuzzing Harness

PROMEFUZZ

Design

Prompt Example for Generator

TASK

Write a C++ fuzzing harness program to fuzz the library pugixml using the information provided below. Reply with only the content of the C++ source file, without any additional explanation. Follow:

1. Prepare the environment necessary before invoking the target function. Identify the required data types and classes, initialize them with appropriate values.
2. Conduct any required cleanup afterward. Pay attention to the return value of the function and handle exceptions properly.
3. Wrap all logic within the `extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size)` function, which acts as the entry point for fuzzing.

...

CONTEXT

Target API Functions

You should invoke the functions in the following order: `pugi::xml_node::child` -> `pugi::xml_node::attribute` -> ...

Below is the detailed information:

1. `pugi::xpath_query::evaluate_string`

The `pugi::xpath_query::evaluate_string` function evaluates an XPath query as a string in a given XML context, performing type conversion if needed. It returns the result as a string, handling memory allocation errors with exceptions unless `PUGIXML_NO_EXCEPTIONS` is defined.

This function is a member of class pugi::xpath_query. Its signature is:

```cpp
string_t evaluate_string(const xpath_node& n) const;
size_t evaluate_string(char_t* buffer, size_t capacity, const xpath_node& n) const;
```

You should invoke all 2 overloads above for this function.

...

Related Data Types

```cpp
namespace pugi {
struct xml_node_struct {
```

...

Related Constructor Methods

```cpp
namespace pugi {
class pugi::xml_node {
 explicit xml_node(xml_node_struct* p);
```

...

Required Header Files

You need to include `pugixml.hpp` to invoke functions above.

RESPONSE

```cpp
#include "pugixml.hpp"
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    if (Size < 1) return 0;

    pugi::xml_document doc;
    pugi::xml_parse_result result = doc.load_buffer(Data, Size);
    if (!result) return 0;

    pugi::xml_node root = doc.root();
    pugi::xml_node child1 = root.child("child1");
    pugi::xml_attribute attr = child1.attribute("attr");
    pugi::xml_node child2 = child1.child("child2");

    pugi::xpath_query query("//node");
    pugi::xpath_node xnode(child2);

    bool eval_bool = query.evaluate_boolean(xnode);
    double eval_num = query.evaluate_number(xnode);

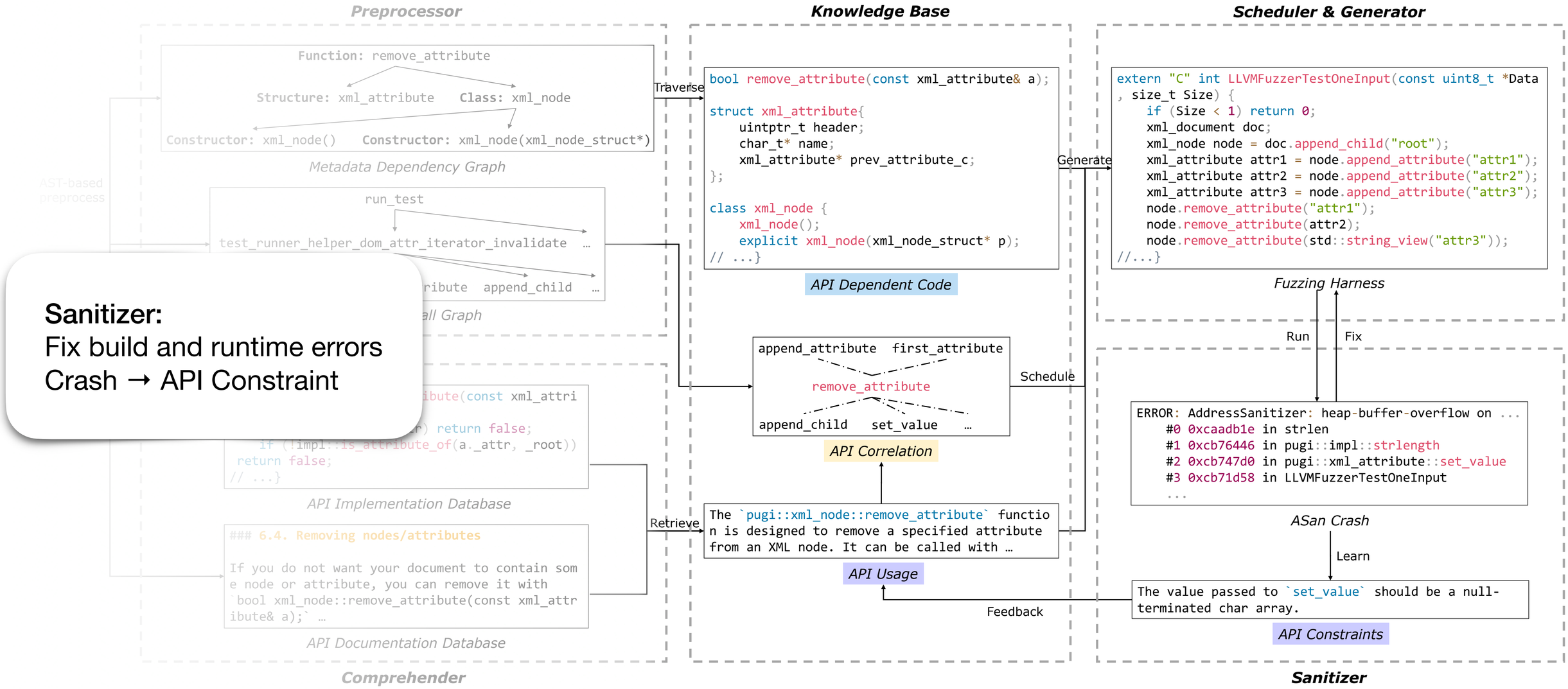
    pugi::string_t eval_str1 = query.evaluate_string(xnode);
    char buffer[1024];
    size_t eval_str2 = query.evaluate_string(buffer, sizeof(buffer), xnode);

    pugi::xpath_node_set eval_node_set = query.evaluate_node_set(xnode);
    pugi::xpath_node eval_node = query.evaluate_node(xnode);

    return 0;
}
```

PROMEFUZZ

Design



Sanitizer:
Fix build and runtime errors
Crash → API Constraint

PROMEFUZZ
Design

Prompt Example for Crash Analyzer

TASK

As an expert developer familiar with the `libtiff` library, you encounter a crash when running fuzz test for it. Your task is to investigate the crash by answering the following questions:
- What kind of crash is it?
- Which part of the library is causing the crash?
- What is the root cause of the crash?
- Is the crash due to a bug in the library or misusing?

Below are the fuzzing harness that triggers the crash and its crash report, please analyze them and offer a concise explanation of it.

CONTEXT

```
# Fuzzing harness
```c
int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
...
    TIFFGetField(tif, TIFFTAG_IMAGEWIDTH, &width);
    TIFFGetField(tif, TIFFTAG_IMAGELENGTH, &height);
    uint32_t *raster = (uint32_t *)malloc(width * height * sizeof(uint32
_t));
    TIFFReadRGBATileExt(tif, col, row, raster, stop_flag);
... }
```

Crash report
Crash reason
AddressSanitizer: SEGV on unknown address 0x7fae47ef6070
Backtrace
The signal is caused by a WRITE memory access.
#0 TIFFReadRGBATileExt
libtiff/tif_getimage.c:3395:9
        memmove(raster + (size_t)(tile_ysize - i_row - 1) * tile_xsize,
...   ^
Related source code
```c
// Part of the `TIFFReadRGBATileExt` function:
    for (i_row = 0; i_row < read_ysize; i_row++) {
        memmove(raster + (size_t)(tile_ysize - i_row - 1) * tile_xsize,
...
## Related data definitions
...
```

RESPONSE

The crash is a segmentation fault (SEGV) caused by a WRITE memory access violation in `TIFFReadRGBATileExt` due to an out-of-bounds `memmove` operation on the `raster` buffer.
Root Cause
The fuzzing harness allocates `raster` based on image dimensions (`TIFFTAG_IMAGEWIDTH`/`TIFFTAG_IMAGELENGTH`), but `TIFFReadRGBATileExt` uses **tile dimensions** for memory offsets. If tile dimensions exceed, the undersized `raster` buffer overflows.
Misuse vs. Library Bug
This is an **API misuse**.The harness incorrectly assumes image dimensions suffice for tile operations, violating libtiff's requirement to size buffers for tile dimensions.Libtiff's implementation is correct but crashes due to the undersized buffer provided by the harness.
```

PROMEFUZZ

Evaluation

PROMEFUZZ achieves:

- **1.50×, 3.88×, 1.91x and 1.40×** higher branch coverage than 3 LLM-based baselines (PromptFuzz, CKGFuzzer and OSS-Fuzz-Gen) and manually crafted harnesses (OSS-Fuzz).
- More unique crashes with **89.7% precision** and uncovers **25 previously unknown vulnerabilities** (21 confirmed by the developer and 3 assigned with CVE IDs).

PromeFuzz

Evaluation

Tested Library					PromeFuzz's Cost			Coverage (#Branch / #API)				
Name	**Version**	**LoC**	**#API**	**#Branch**	**Time**	**Expense**	**Token**	**PromeFuzz**	**PromptFuzz**	**CKGFuzzer**	**OSS-Fuzz-Gen**	**OSS-Fuzz**
c-ares	7978cf7	44K	136	8,237	1h14m	$0.78	3.8M	**6,106** / 113	5,711 / 59	5,677 / 91	3,612 / 28	3,733 / 19
curl	4c50998	187K	96	29,942	31m	$0.28	1.4M	**7,347** / 95	5,006 / 87	6,143 / 91	759 / 27	950 / 7
lcms	04ace9c	46K	358	9,040	52m	$1.59	11.3M	**4,560** / 358	3,672 / 231	3,767 / 227	374 / 32	716 / 31
libjpeg-turbo	36ac5b8	64K	72	10,636	33m	$0.18	1.0M	4,274 / 71	3,520 / 78	N/A	957 / 14	**4,480** / 26
sqlite3	7903db4	180K	270	50,903	1h13m	$2.44	17.1M	**28,225** / 246	26,263 / 160	N/A	17,987 / 17	25,568 / 15
libaom	99fcd81	530K	47	61,360	34m	$0.19	0.9M	**13,697** / 47	13,596 / 45	N/A	11,365 / 20	11,290 / 11
libpcap	2559282	42K	99	5,944	47m	$0.20	1.2M	**3,825** / 89	2,752 / 73	3,358 / 71	502 / 21	3,051 / 10
libvpx	9514ab6	370K	37	32,357	21m	$0.12	0.8M	**5,898** / 37	3,594 / 30	3,578 / 32	3,206 / 14	2,755 / 7
zlib	5a82f71	30K	89	2,720	2h06m	$0.17	0.9M	**2,430** / 89	2,099 / 85	2,343 / 66	1,865 / 34	1,789 / 29
re2	c84a140	28K	79	8,520	31m	$0.13	0.8M	**6,236** / 78	5,222 / 30	N/A	5,408 / 17	6,134 / 36
libmagic	dadc01f	18K	18	6,124	3m	$0.04	0.2M	**3,868** / 18	120 / 9	N/A	2,689 / 8	3,132 / 8
libpng	738f5e7	58K	256	6,999	51m	$1.10	7.6M	**3,538** / 251	3,296 / 228	N/A	881 / 16	2,081 / 27
ngiflib	db19270	1K	7	357	5m	$0.06	0.2M	**351** / 7	283 / 7	323 / 6	40 / 5	N/A
ffjpeg	caade60	2K	40	585	7m	$0.12	0.4M	**571** / 40	375 / 22	N/A	305 / 33	N/A
liblouis	3d95765	38K	30	6,247	10m	$0.07	0.3M	**4,704** / 30	2,220 / 17	N/A	1,723 / 22	3,797 / 5
libtiff	fcd4c86	92K	198	13,528	28m	$0.49	3.5M	**8,096** / 198	7,428 / 136	7,109 / 148	4,983 / 68	5,666 / 13
cjson	12c4bf1	17K	78	924	7m	$0.11	0.6M	**899** / 78	863 / 75	706 / 43	779 / 57	502 / 6
pugixml	06318b0	24K	221	5,957	1h24m	$0.32	2.0M	**3,957** / 221	N/A	N/A	369 / 2	3,385 / 7
tinygltf	a5e653e	193K	65	11,594	8m	$0.12	0.5M	**5,739** / 65	N/A	N/A	1,541 / 2	2,885 / 1
exiv2	04e1ea3	79K	880	44,320	5h46m	$7.17	43.4M	**13,580** / 816	N/A	N/A	8,144 / 7	9,857 / 17
loguru	4adaa18	3K	84	883	10m	$0.15	0.5M	**621** / 66	N/A	190 / 33	233 / 10	N/A
rapidcsv	083851d	6K	49	326	8m	$0.07	0.3M	**205** / 49	N/A	N/A	10 / 8	N/A
SUM			3,209	317,503	18h5m	$15.89	98.4M	**128,727** / 3,062	86,020 / 1,372	33,194 / 808	67,489 / 444	91,771 / 275

Overall Results of PromeFuzz-generated Fuzzing Harnesses

Library	ID	Vulnerability Type	Status	P	C	OG	OF
ngiflib	Issue 34	NULL Pointer Dereference	Fixed	✗	✓	✗	-
	Issue 36	NULL Pointer Dereference	Fixed	✗	✓	✗	-
ffjpeg	Issue 55	Use Uninitialized Variable	Reported	✗	-	✗	-
	Issue 57	Buffer Underflow	Reported	✗	-	✗	-
	Issue 58	Stack Buffer Overflow	Reported	✗	-	✗	-
	Issue 60	Stack Buffer Overflow	Reported	✗	-	✗	-
liblouis	Issue 1708	Invalid Free	Fixed	✗	-	✓	✗
	Issue 1709	Invalid Free	Fixed	✗	-	✗	✗
	Issue 1711	Memory Leak	Confirmed	✗	-	✗	✗
	Issue 1718_1	Memory Leak	Fixed	✗	-	✗	✗
	Issue 1718_2	Memory Leak	Fixed	✗	-	✗	✗
	Issue 1719	Heap Buffer Overflow	Fixed	✗	-	✗	✓
	Issue 1720	Heap Buffer Overflow	Fixed	✗	-	✗	✓
	Issue 1721	Heap Buffer Overflow	Fixed	✗	-	✗	✓
	Security j9qg	Infinite Loop	Confirmed	✗	-	✗	✗
	Security 3wwm	Heap Buffer Overflow	Confirmed	✗	-	✗	✓
	Security w63h	Heap Buffer Overflow	Confirmed	✗	-	✗	✗
libtiff	Issue 669	Reachable Assertion	Fixed	✗	✗	✗	✗
	CVE-2025-45701	Memory Leak	Fixed	✗	✗	✗	✗
exiv2	CVE-2025-26623	Use After Free	Fixed	-	-	✗	✗
libmagic	CVE-2025-55450	Integer Overflow	Fixed	✗	✗	✗	✗
	Bug tracker 640	Integer Overflow	Fixed	✗	✗	✗	✗
rapidcsv	Issue 186	Out-of-bounds Write	Fixed	-	-	✗	-
	Issue 187	Out-of-bounds Write	Fixed	-	-	✗	-
	Issue 188	Out-of-bounds Read	Fixed	-	-	✗	-

P: PromptFuzz **C**: CKGFuzzer **OG**: OSS-Fuzz-Gen **OF**: OSS-Fuzz

Vulnerabilities Found by PromeFuzz

Thanks