Preliminary Result

1. Vulnerability details

A Double-Free vulnerability exists in the C/C++ function `static void __init st_of_clkgen_mux_setup(..)`. Here is the detail breakdown:

Task	**GroundTruth**
Whether code is vulnerable or not	Yes
What kind of vulnerability it has	Double Free
The explanation of vulnerability	After successfully freeing parents in the normal flow, the program will continue to the error label err, which causes `kfree(parents)` to be called a second time, leading to the double free.
Fix suggestions	To fix this, you need to add a `return` **statement** after the normal flow to prevent the code from falling into the error labels.

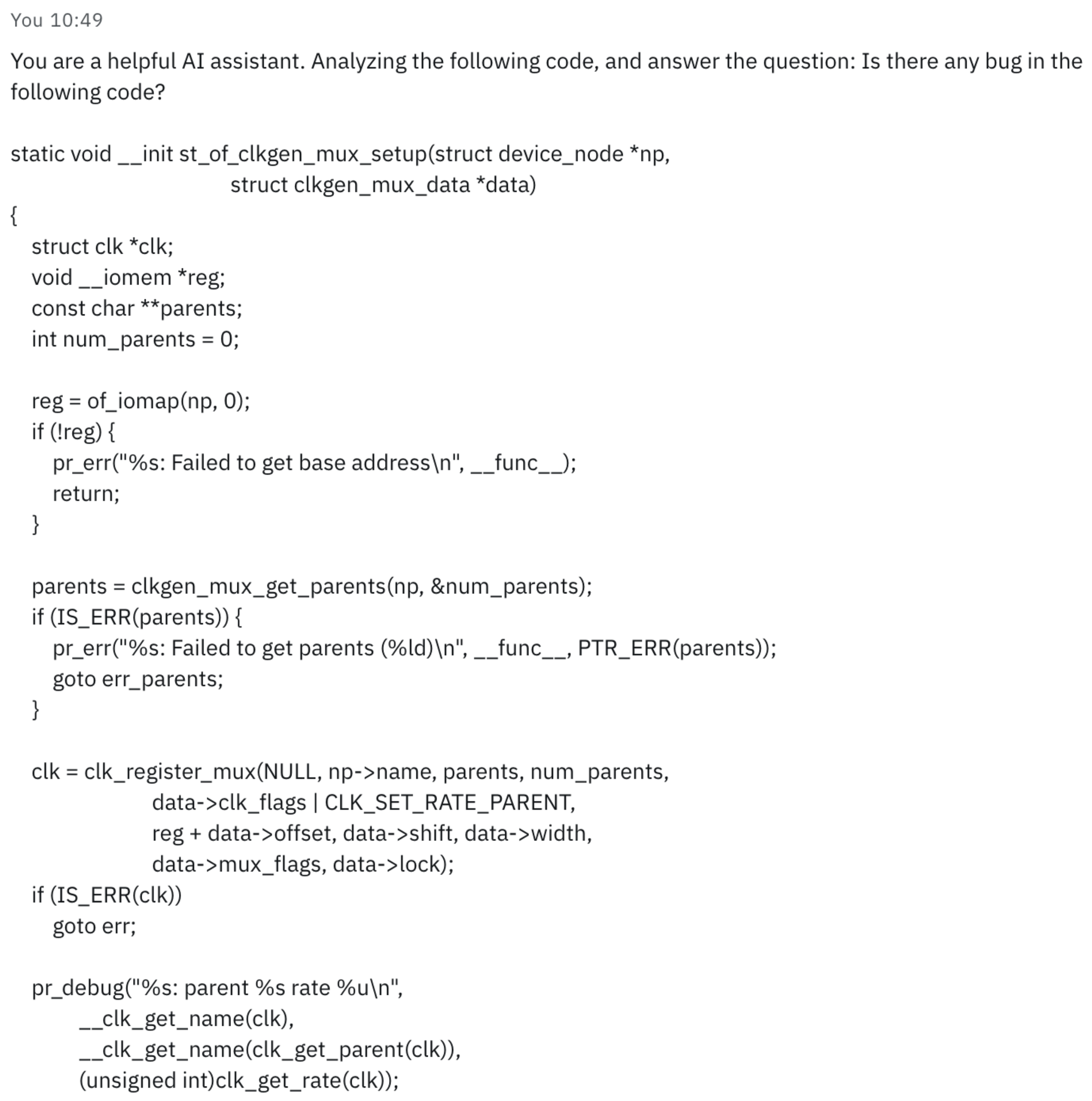
```
static void __init st_of_clkgen_mux_setup(struct device_node *n
        struct clkgen_mux_data *data)
{
    struct clk *clk;
    void __iomem *reg;
    const char **parents;
    int num_parents = 0;

    reg = of_iomap(np, 0);
    if (!reg) {
        pr_err("%s: Failed to get base address\n", __func__);
        return;
    }

    parents = clkgen_mux_get_parents(np, &num_parents);
    if (IS_ERR(parents)) {
        pr_err("%s: Failed to get parents (%ld)\n",
                __func__, PTR_ERR(parents));
        goto err_parents;
    }

    clk = clk_register_mux(NULL, np->name, parents, num_parents,
                data->clk_flags | CLK_SET_RATE_PARENT,
                reg + data->offset,
                data->shift, data->width, data->mux_flags,
                data->lock);
    if (IS_ERR(clk))
        goto err;

    pr_debug("%s: parent %s rate %u\n",
            __clk_get_name(clk),
            __clk_get_name(clk_get_parent(clk)),
            (unsigned int)clk_get_rate(clk));

    kfree(parents);
    of_clk_add_provider(np, of_clk_src_simple_get, clk);

err:
    kfree(parents);
err_parents:
    iounmap(reg);

}
```

2. Experiment details

We tested two popular Large Language Models (LLMs), **llama-3-70b-instruct and ChatGPT**, to investigate their capability in detecting, explaining, and fixing this double-free bug. Neither model could perfectly complete these tasks. The Llama model suffered from hallucination and an inability to correctly analyze complex control flows. ChatGPT performed better but fell short in providing efficient and clear fix suggestions.

2.1 Meta-llama/llama-3-70b-instruct

> `User Input:`



You 10:49

You are a helpful AI assistant. Analyzing the following code, and answer the question: Is there any bug in the following code?

```
static void __init st_of_clkgen_mux_setup(struct device_node *np,
                                          struct clkgen_mux_data *data)
{
    struct clk *clk;
    void __iomem *reg;
    const char **parents;
    int num_parents = 0;

    reg = of_iomap(np, 0);
    if (!reg) {
        pr_err("%s: Failed to get base address\n", __func__);
        return;
    }

    parents = clkgen_mux_get_parents(np, &num_parents);
    if (IS_ERR(parents)) {
        pr_err("%s: Failed to get parents (%ld)\n", __func__, PTR_ERR(parents));
        goto err_parents;
    }

    clk = clk_register_mux(NULL, np->name, parents, num_parents,
                           data->clk_flags | CLK_SET_RATE_PARENT,
                           reg + data->offset, data->shift, data->width,
                           data->mux_flags, data->lock);
    if (IS_ERR(clk))
        goto err;

    pr_debug("%s: parent %s rate %u\n",
             __clk_get_name(clk),
             __clk_get_name(clk_get_parent(clk)),
             (unsigned int)clk_get_rate(clk));
```

> ```
>     kfree(parents);
>     of_clk_add_provider(np, of_clk_src_simple_get, clk);
>
>   err:
>     kfree(parents);
>   err_parents:
>     iounmap(reg);
>   }
> ```

> `The system output:`
>
> AI 10:50
>
> The code provided appears to be a function to set up a clock multiplexer (mux) in a Linux kernel driver. It maps the register address of the mux, retrieves the parent clocks, registers the mux clock, and adds it to the clock provider.
>
> There is one potential issue with this code:
>
> 1. **Resource Leak:** If `clk_register_mux` fails, the function jumps to the `err` label, which frees the `parents` array but not the mapped register address `reg`. This can cause a resource leak. To fix this, the `iounmap(reg)` call should be moved to the `err` label as well.
>
> Here's the corrected code:
>
> ```c
> static void __init st_of_clkgen_mux_setup(struct device_node *np,
>                                           struct clkgen_mux_data *data)
> {
>   struct clk *clk;
>   void __iomem *reg;
>   const char **parents;
>   int num_parents = 0;
>
>   reg = of_iomap
> ```

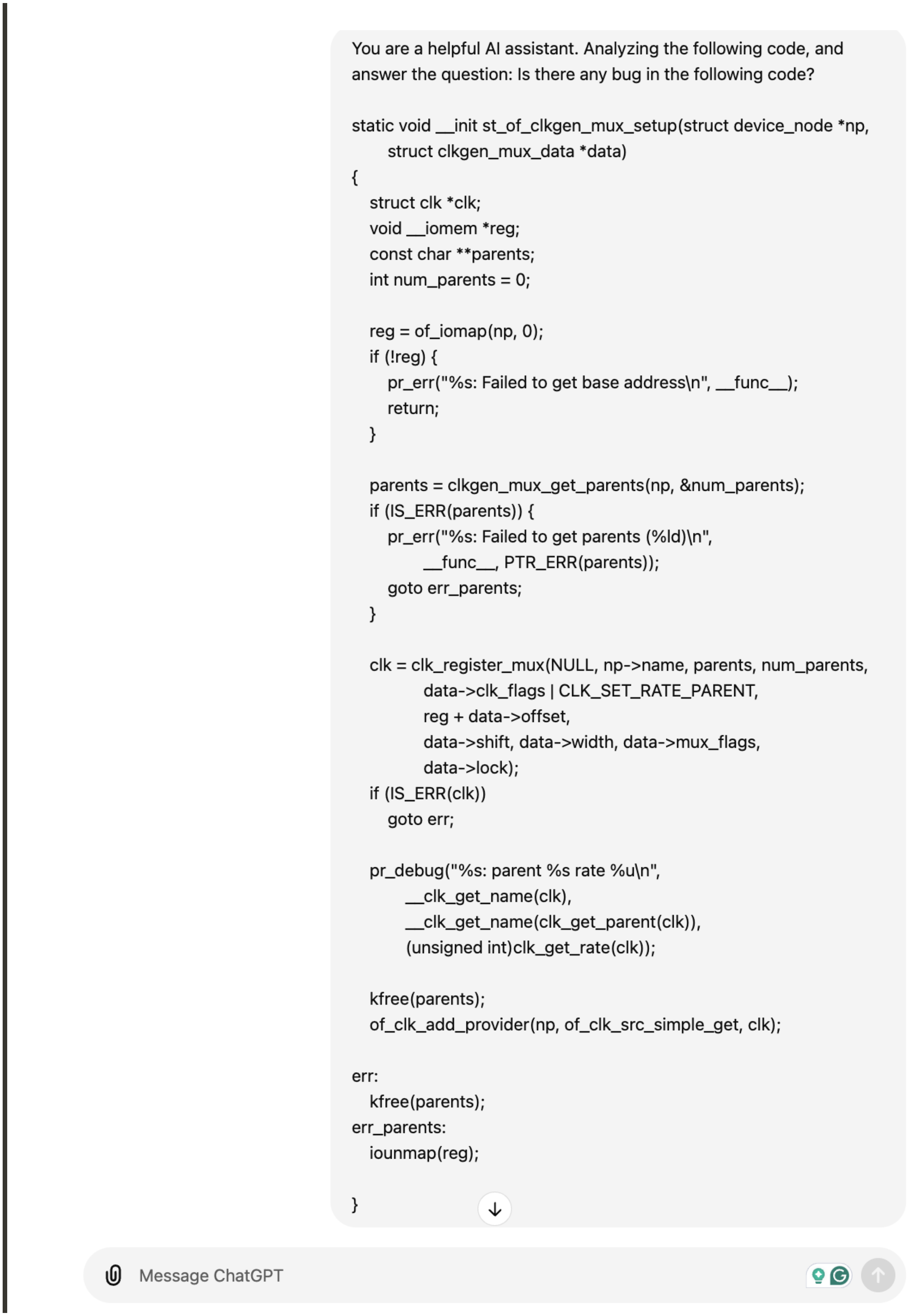
Observation of experiment result

The ability of Meta-llama/llama-3-70b-instruct

Tasks	Model Output	Model Performance	Observation
Whether code is vulnerable or not	The code is vulnerable	Correct	The model identify the code is vulnerable.
What kind of vulnerability it has	Resource Leak	Wrong	However, the model can not identify which lines of the code is vulnerable and what type of bug it has.
The explanation of vulnerability	If `clk_register_mux` fails, the function jumps to the `err` label, which frees the `parent` array but not the mapped register address `reg`.	Wrong	The model suffer from hallucination and failed to correctly analyze the complex control flows involving pointer allocation and deallocation, leading to incorrect reasoning and explanations
Fix suggestions	To fix this, the `iounmap(reg)` call should be moved to the `err` label as well.	Wrong	The fix suggestions are totally wrong.

2.2 OpenAI/ChatGPT

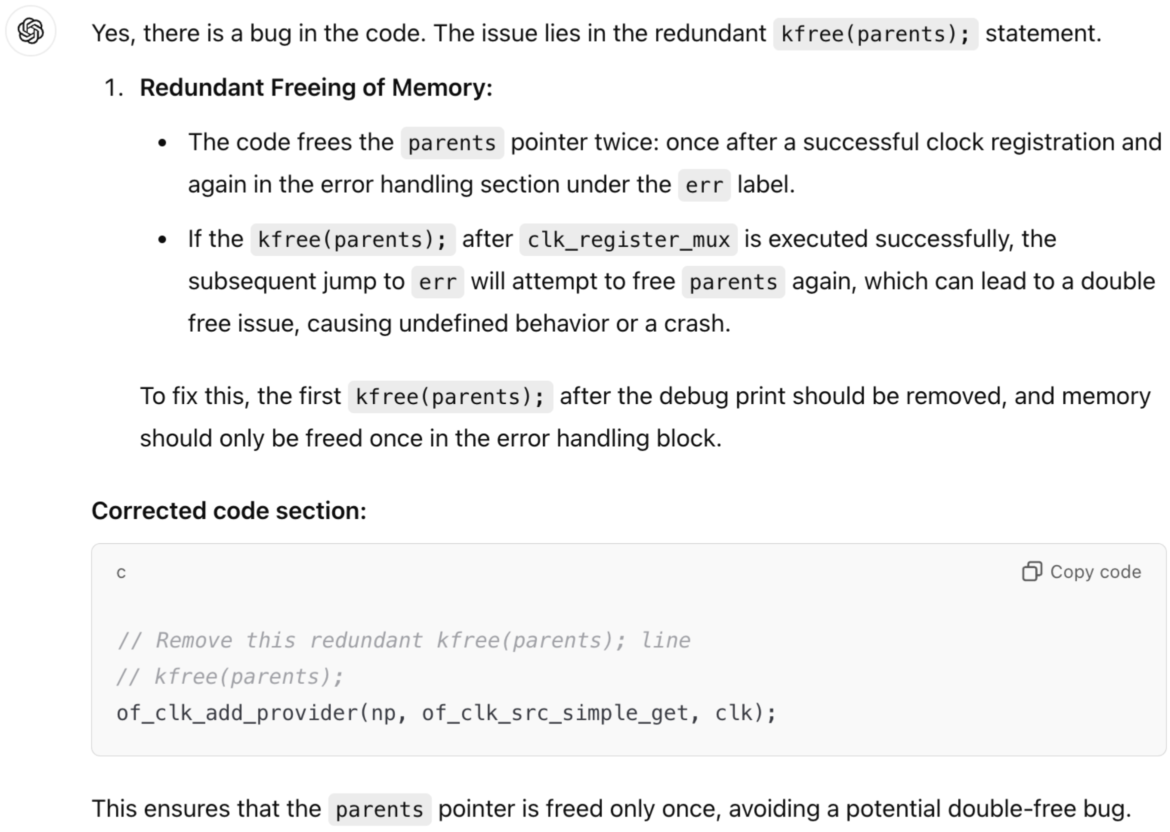
> `User input:`



You are a helpful AI assistant. Analyzing the following code, and answer the question: Is there any bug in the following code?

```
static void __init st_of_clkgen_mux_setup(struct device_node *np,
        struct clkgen_mux_data *data)
{
    struct clk *clk;
    void __iomem *reg;
    const char **parents;
    int num_parents = 0;

    reg = of_iomap(np, 0);
    if (!reg) {
        pr_err("%s: Failed to get base address\n", __func__);
        return;
    }

    parents = clkgen_mux_get_parents(np, &num_parents);
    if (IS_ERR(parents)) {
        pr_err("%s: Failed to get parents (%ld)\n",
                __func__, PTR_ERR(parents));
        goto err_parents;
    }

    clk = clk_register_mux(NULL, np->name, parents, num_parents,
                data->clk_flags | CLK_SET_RATE_PARENT,
                reg + data->offset,
                data->shift, data->width, data->mux_flags,
                data->lock);
    if (IS_ERR(clk))
        goto err;

    pr_debug("%s: parent %s rate %u\n",
            __clk_get_name(clk),
            __clk_get_name(clk_get_parent(clk)),
            (unsigned int)clk_get_rate(clk));

    kfree(parents);
    of_clk_add_provider(np, of_clk_src_simple_get, clk);

err:
    kfree(parents);
err_parents:
    iounmap(reg);

}
```

> `The system output:`



Yes, there is a bug in the code. The issue lies in the redundant `kfree(parents);` statement.

1. **Redundant Freeing of Memory:**
    - The code frees the `parents` pointer twice: once after a successful clock registration and again in the error handling section under the `err` label.
    - If the `kfree(parents);` after `clk_register_mux` is executed successfully, the subsequent jump to `err` will attempt to free `parents` again, which can lead to a double free issue, causing undefined behavior or a crash.

    To fix this, the first `kfree(parents);` after the debug print should be removed, and memory should only be freed once in the error handling block.

Corrected code section:

```c
// Remove this redundant kfree(parents); line
// kfree(parents);
of_clk_add_provider(np, of_clk_src_simple_get, clk);
```

This ensures that the `parents` pointer is freed only once, avoiding a potential double-free bug.

Observation of experiment result

The ability of Meta-llama/llama-3-70b-instruct (1)

Tasks	Model Output	Model Performance	Observation
Whether code is vulnerable or not	The code is vulnerable	Correct	The model identify the code is vulnerable.
What kind of vulnerability it has	Redundant Freeing of Memory	Correct	The model can correctly identify double-free bug in the code

Tasks	Model Output	Model Performance	Observation
The explanation of vulnerability	The code frees the `parents` pointer twice: once after a successful clock registration and again in the error handling section under the `err` label.If the `kfree(parents);` after `clk_register_mux` is executed successfully, the subsequent jump to `err` will attempt to free `parents` again, which can lead to a double free issue, causing undefined behavior or a crash.	Correct	The model can interpret the code execution flow and identify the buggy flow related to double-free vulnerabilities.
Fix suggestions	To fix this, the first `kfree(parents);` after the debug print should be removed, and memory should only be freed once in the error handling block.	Wrong	The model provide an inefficient and unclear fix suggestion. It keeps the function flow going through the error handling section (the `err` label) even when everything works correctly. This adds unnecessary complexity and reduces efficiency because, even in a success case, the function would not need to execute the `err` block, which is typically reserved for error handling.