A Simple Demo of Repository-level Code Auditing with Cursor

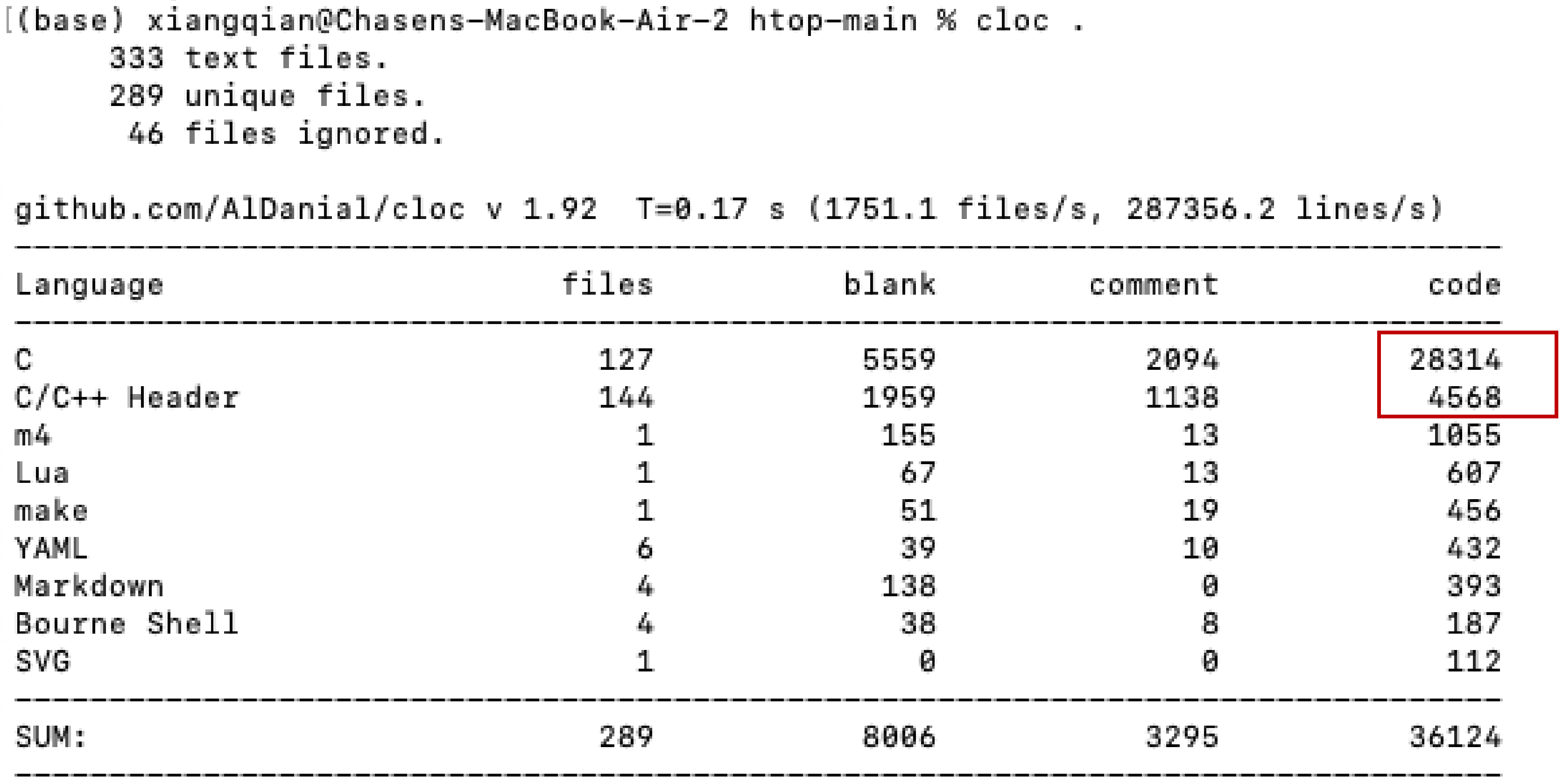
Dr. Chengpeng Wang

PURDUE
UNIVERSITY

Project Info

Project: htop

commit e8236c43

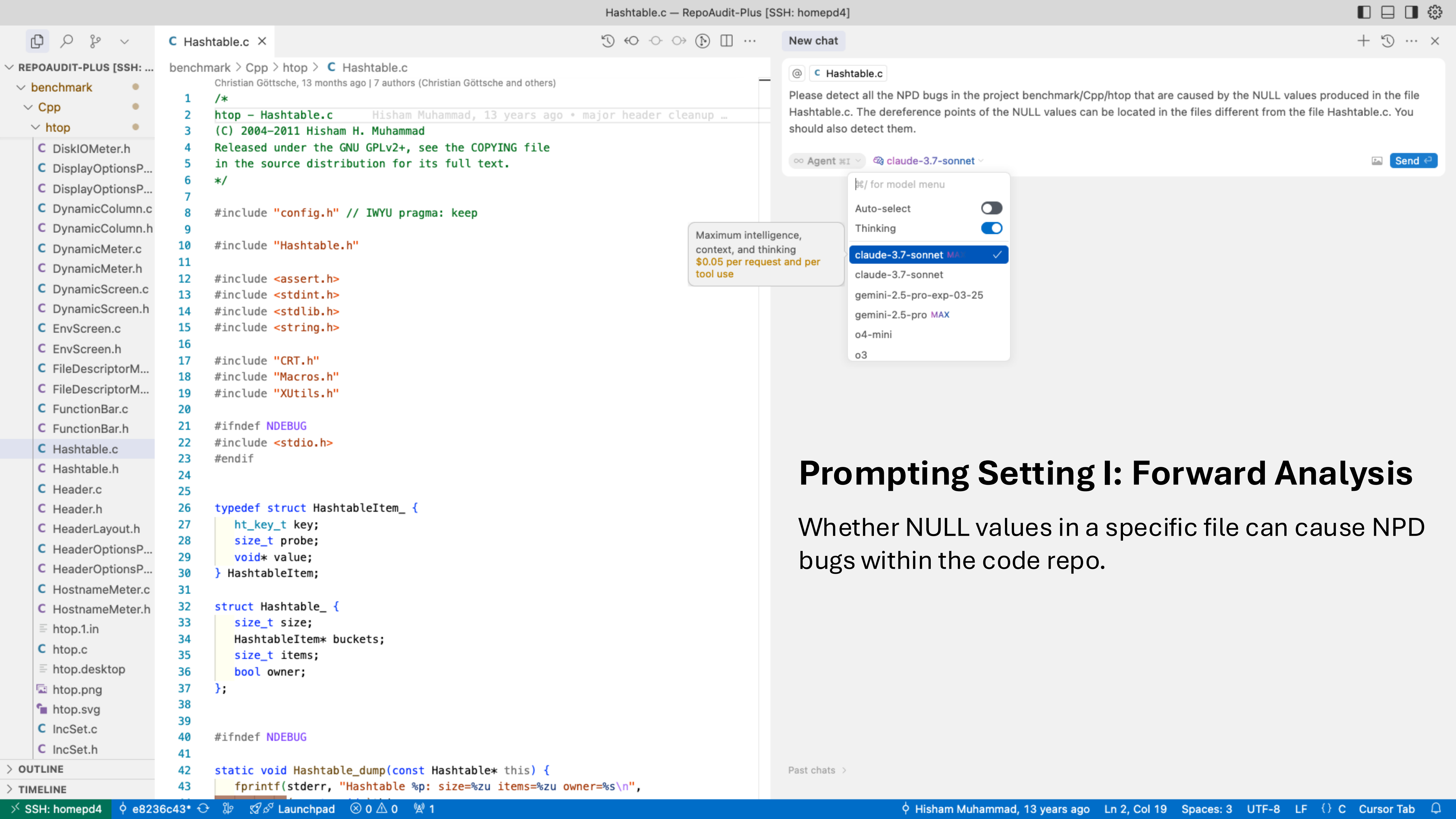


```
[(base) xiangqian@Chasens-MacBook-Air-2 htop-main % cloc .
     333 text files.
     289 unique files.
      46 files ignored.

github.com/AlDanial/cloc v 1.92  T=0.17 s (1751.1 files/s, 287356.2 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                              127           5559           2094          28314
C/C++ Header                   144           1959           1138           4568
m4                               1            155             13           1055
Lua                              1             67             13            607
make                             1             51             19            456
YAML                             6             39             10            432
Markdown                         4            138              0            393
Bourne Shell                     4             38              8            187
SVG                              1              0              0            112
-------------------------------------------------------------------------------
SUM:                           289           8006           3295          36124
-------------------------------------------------------------------------------
```

~32.9 KLoC

Cursor Setting

- Agent: Cursor with Claude-3.7-Sonnet Max (Thinking mode)
- Bug Type: Null Pointer Dereference (NPD)
- Prompting settings
  - Forward Analysis: Target NULL values in a single file
  - Backward Analysis: Target dereferences in a single file
  - Whole Repo Analysis: Target all contexts and dereferences in the repo



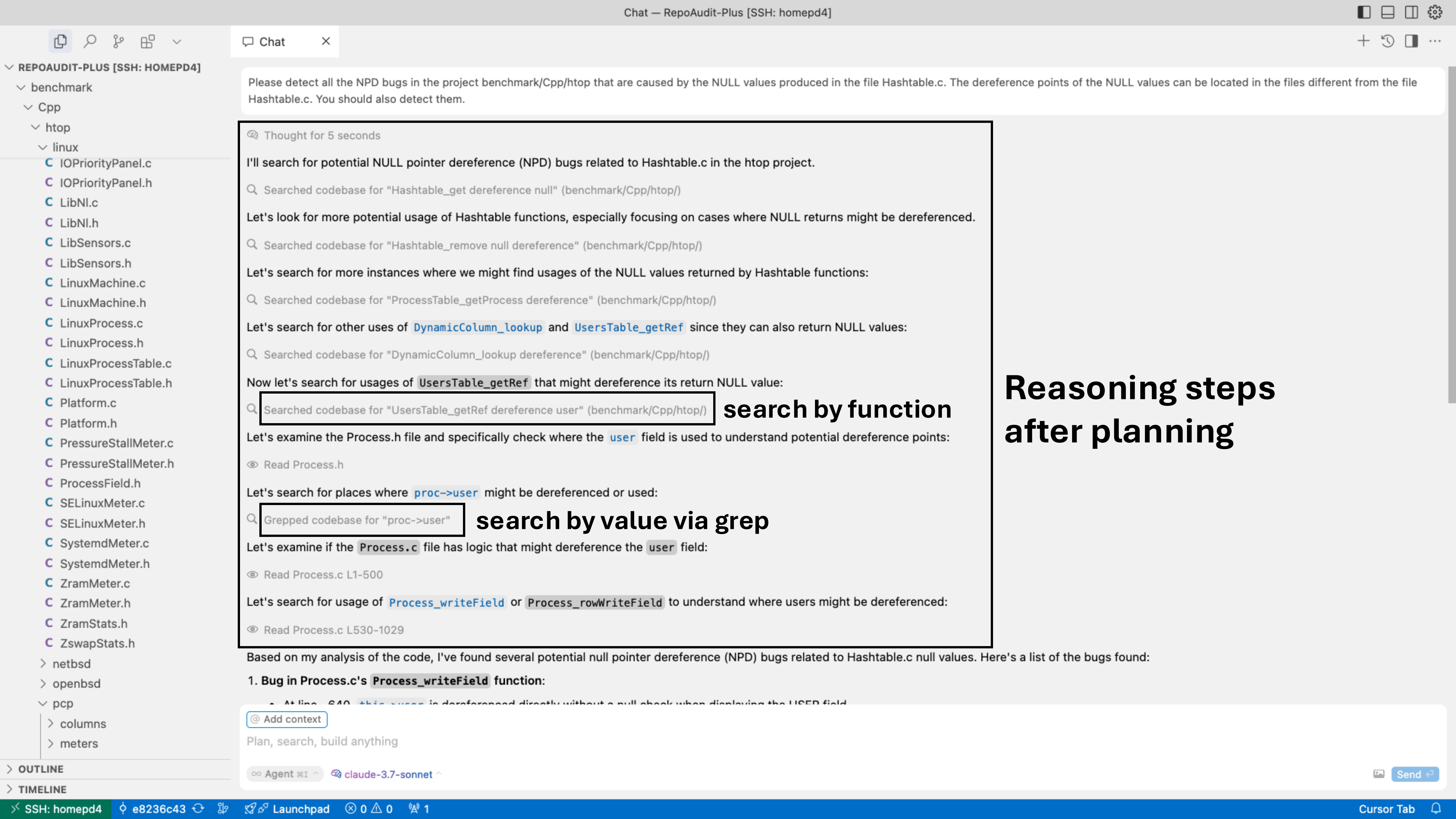
```c
/*
htop - Hashtable.c
(C) 2004-2011 Hisham H. Muhammad
Released under the GNU GPLv2+, see the COPYING file
in the source distribution for its full text.
*/

#include "config.h" // IWYU pragma: keep

#include "Hashtable.h"

#include <assert.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

#include "CRT.h"
#include "Macros.h"
#include "XUtils.h"

#ifndef NDEBUG
#include <stdio.h>
#endif


typedef struct HashtableItem_ {
   ht_key_t key;
   size_t probe;
   void* value;
} HashtableItem;

struct Hashtable_ {
   size_t size;
   HashtableItem* buckets;
   size_t items;
   bool owner;
};


#ifndef NDEBUG

static void Hashtable_dump(const Hashtable* this) {
   fprintf(stderr, "Hashtable %p: size=%zu items=%zu owner=%s\n",
```

Please detect all the NPD bugs in the project benchmark/Cpp/htop that are caused by the NULL values produced in the file Hashtable.c. The dereference points of the NULL values can be located in the files different from the file Hashtable.c. You should also detect them.

Auto-select ○
Thinking ●
claude-3.7-sonnet MAX ✓

Prompting Setting I: Forward Analysis

Whether NULL values in a specific file can cause NPD bugs within the code repo.



Please detect all the NPD bugs in the project benchmark/Cpp/htop that are caused by the NULL values produced in the file Hashtable.c. The dereference points of the NULL values can be located in the files different from the file Hashtable.c. You should also detect them.

Thought for 5 seconds

I'll search for potential NULL pointer dereference (NPD) bugs related to Hashtable.c in the htop project.

Searched codebase for "Hashtable_get dereference null" (benchmark/Cpp/htop/)

Let's look for more potential usage of Hashtable functions, especially focusing on cases where NULL returns might be dereferenced.

Searched codebase for "Hashtable_remove null dereference" (benchmark/Cpp/htop/)

Let's search for more instances where we might find usages of the NULL values returned by Hashtable functions:

Searched codebase for "ProcessTable_getProcess dereference" (benchmark/Cpp/htop/)

Let's search for other uses of `DynamicColumn_lookup` and `UsersTable_getRef` since they can also return NULL values:

Searched codebase for "DynamicColumn_lookup dereference" (benchmark/Cpp/htop/)

Now let's search for usages of `UsersTable_getRef` that might dereference its return NULL value:

Searched codebase for "UsersTable_getRef dereference user" (benchmark/Cpp/htop/)

search by function

Let's examine the Process.h file and specifically check where the `user` field is used to understand potential dereference points:

Read Process.h

Let's search for places where `proc->user` might be dereferenced or used:

Grepped codebase for "proc->user"

search by value via grep

Let's examine if the `Process.c` file has logic that might dereference the `user` field:

Read Process.c L1-500

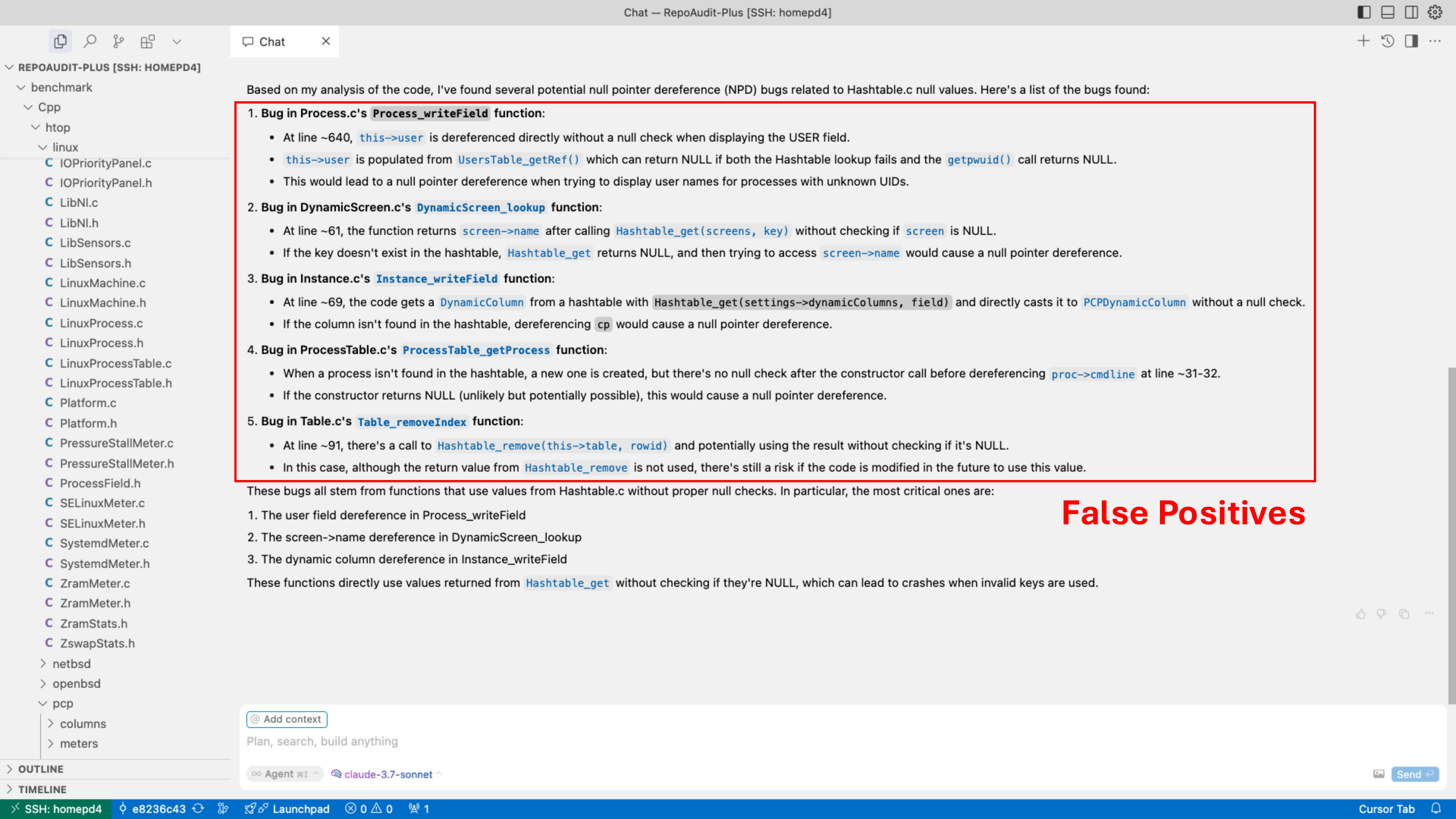
Let's search for usage of `Process_writeField` or `Process_rowWriteField` to understand where users might be dereferenced:

Read Process.c L530-1029

Reasoning steps after planning

Based on my analysis of the code, I've found several potential null pointer dereference (NPD) bugs related to Hashtable.c null values. Here's a list of the bugs found:

1. **Bug in Process.c's `Process_writeField` function:**
    - At line ~640, `this->user` is dereferenced directly without a null check when displaying the USER field.



Based on my analysis of the code, I've found several potential null pointer dereference (NPD) bugs related to Hashtable.c null values. Here's a list of the bugs found:

1. **Bug in Process.c's `Process_writeField` function:**
   - At line ~640, `this->user` is dereferenced directly without a null check when displaying the USER field.
   - `this->user` is populated from `UsersTable_getRef()` which can return NULL if both the Hashtable lookup fails and the `getpwuid()` call returns NULL.
   - This would lead to a null pointer dereference when trying to display user names for processes with unknown UIDs.
2. **Bug in DynamicScreen.c's `DynamicScreen_lookup` function:**
   - At line ~61, the function returns `screen->name` after calling `Hashtable_get(screens, key)` without checking if `screen` is NULL.
   - If the key doesn't exist in the hashtable, `Hashtable_get` returns NULL, and then trying to access `screen->name` would cause a null pointer dereference.
3. **Bug in Instance.c's `Instance_writeField` function:**
   - At line ~69, the code gets a `DynamicColumn` from a hashtable with `Hashtable_get(settings->dynamicColumns, field)` and directly casts it to `PCPDynamicColumn` without a null check.
   - If the column isn't found in the hashtable, dereferencing `cp` would cause a null pointer dereference.
4. **Bug in ProcessTable.c's `ProcessTable_getProcess` function:**
   - When a process isn't found in the hashtable, a new one is created, but there's no null check after the constructor call before dereferencing `proc->cmdline` at line ~31-32.
   - If the constructor returns NULL (unlikely but potentially possible), this would cause a null pointer dereference.
5. **Bug in Table.c's `Table_removeIndex` function:**
   - At line ~91, there's a call to `Hashtable_remove(this->table, rowid)` and potentially using the result without checking if it's NULL.
   - In this case, although the return value from `Hashtable_remove` is not used, there's still a risk if the code is modified in the future to use this value.

These bugs all stem from functions that use values from Hashtable.c without proper null checks. In particular, the most critical ones are:

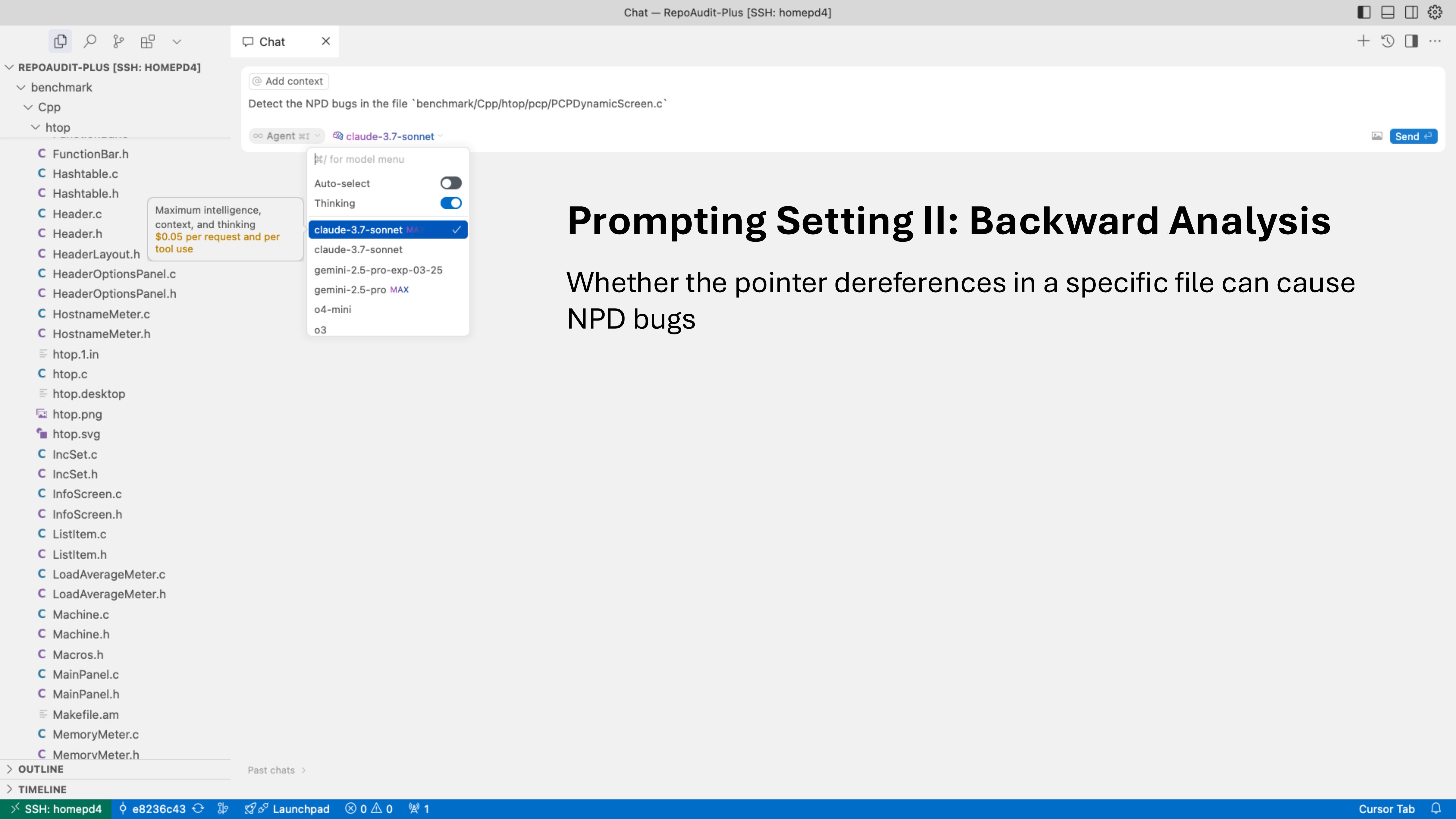
1. The user field dereference in Process_writeField
2. The screen->name dereference in DynamicScreen_lookup
3. The dynamic column dereference in Instance_writeField

These functions directly use values returned from `Hashtable_get` without checking if they're NULL, which can lead to crashes when invalid keys are used.

False Positives

Prompting Setting II: Backward Analysis

Whether the pointer dereferences in a specific file can cause NPD bugs



Chat — RepoAudit-Plus [SSH: homepd4]

Detect the NPD bugs in the file `benchmark/Cpp/htop/pcp/PCPDynamicScreen.c`

Agent ⌘I

claude-3.7-sonnet

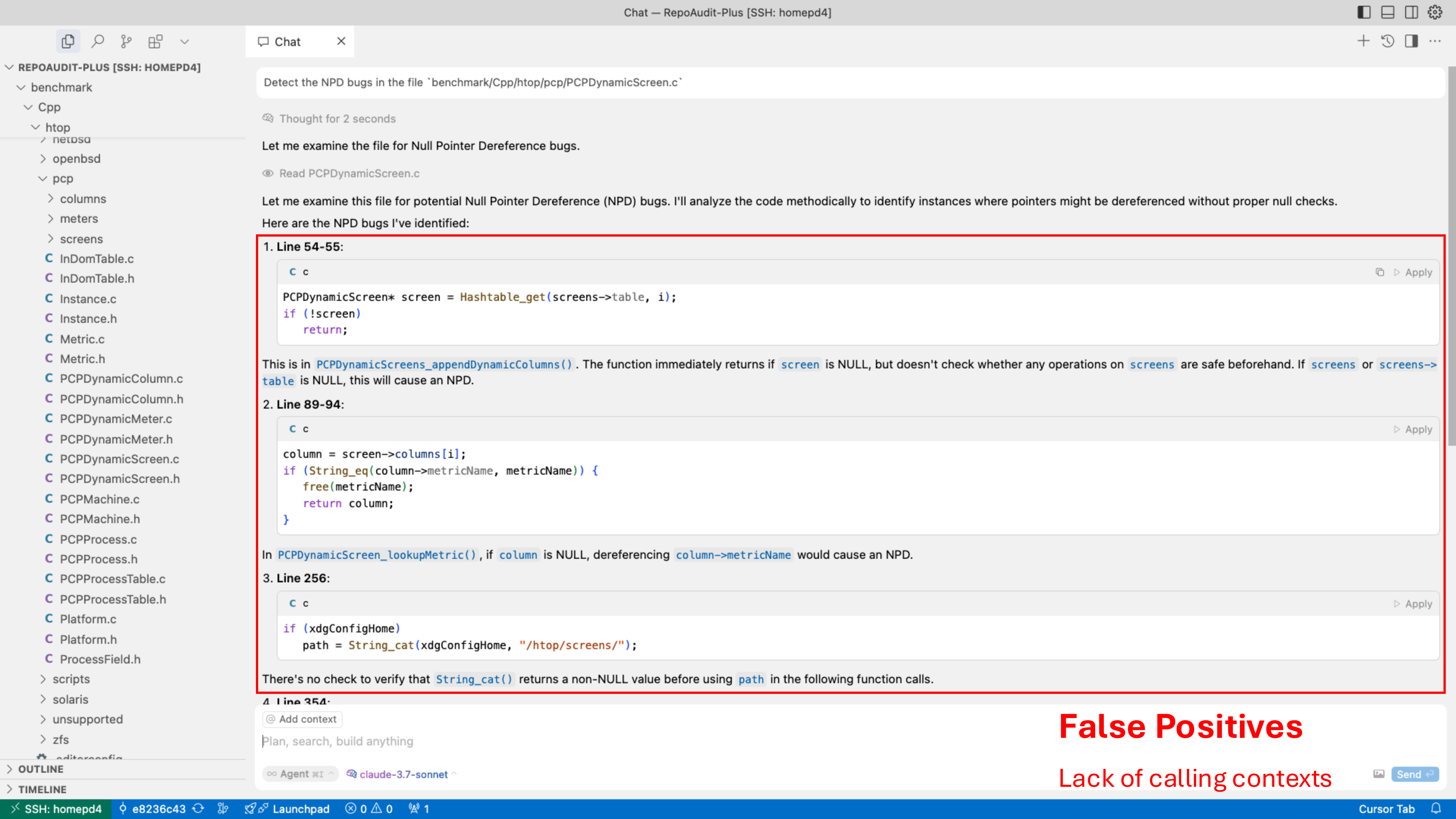
⌘/ for model menu

Auto-select ○

Thinking ●

claude-3.7-sonnet MAX ✓

Maximum intelligence, context, and thinking
$0.05 per request and per tool use



Detect the NPD bugs in the file `benchmark/Cpp/htop/pcp/PCPDynamicScreen.c`

Thought for 2 seconds

Let me examine the file for Null Pointer Dereference bugs.

Read PCPDynamicScreen.c

Let me examine this file for potential Null Pointer Dereference (NPD) bugs. I'll analyze the code methodically to identify instances where pointers might be dereferenced without proper null checks.

Here are the NPD bugs I've identified:

1. **Line 54-55**:

```c
PCPDynamicScreen* screen = Hashtable_get(screens->table, i);
if (!screen)
   return;
```

This is in `PCPDynamicScreens_appendDynamicColumns()`. The function immediately returns if `screen` is NULL, but doesn't check whether any operations on `screens` are safe beforehand. If `screens` or `screens->table` is NULL, this will cause an NPD.

2. **Line 89-94**:

```c
column = screen->columns[i];
if (String_eq(column->metricName, metricName)) {
   free(metricName);
   return column;
}
```

In `PCPDynamicScreen_lookupMetric()`, if `column` is NULL, dereferencing `column->metricName` would cause an NPD.

3. **Line 256**:

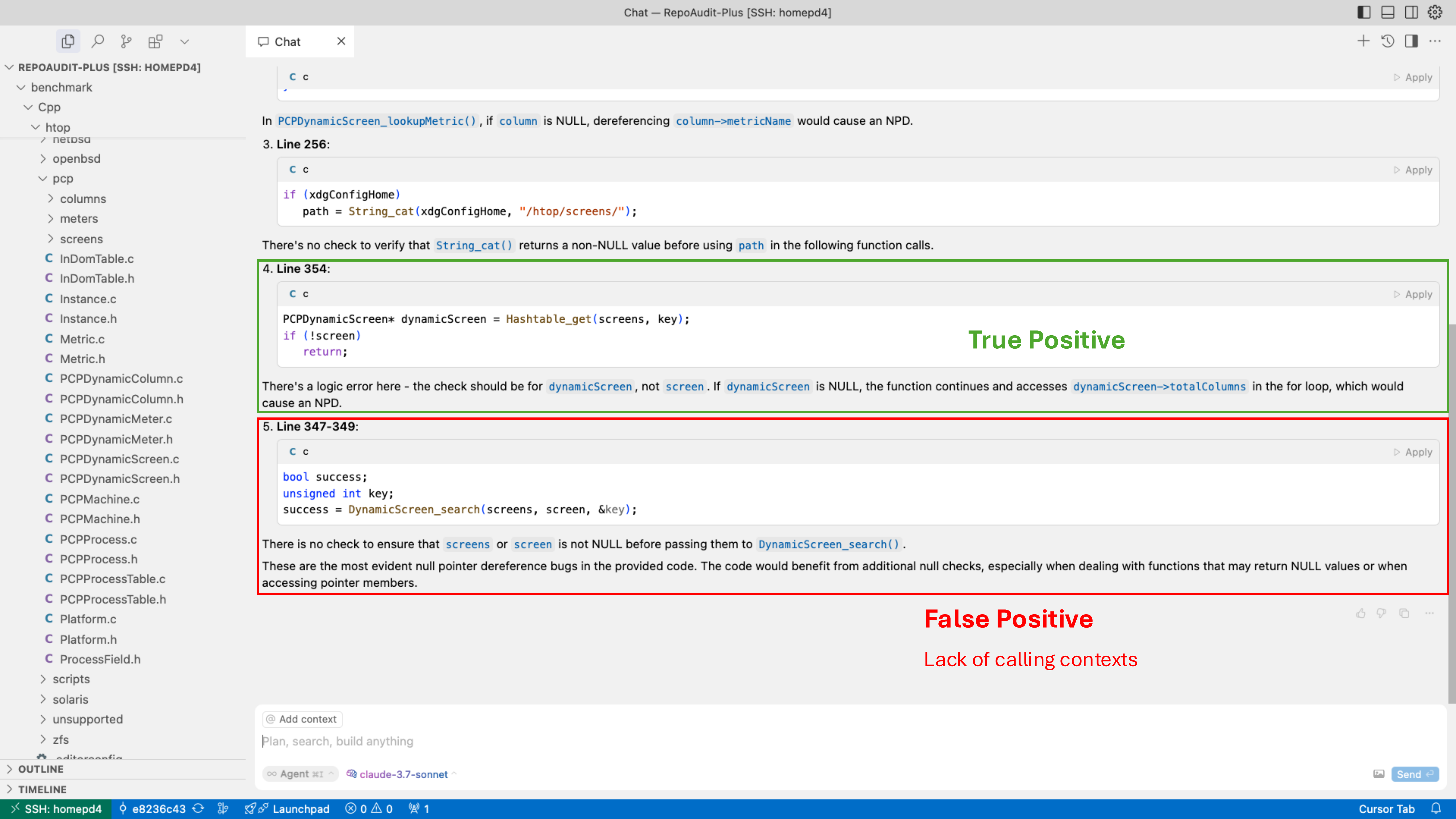
```c
if (xdgConfigHome)
   path = String_cat(xdgConfigHome, "/htop/screens/");
```

There's no check to verify that `String_cat()` returns a non-NULL value before using `path` in the following function calls.

4. Line 354:

False Positives

Lack of calling contexts



In `PCPDynamicScreen_lookupMetric()`, if `column` is NULL, dereferencing `column->metricName` would cause an NPD.

3. **Line 256**:

```c
if (xdgConfigHome)
   path = String_cat(xdgConfigHome, "/htop/screens/");
```

There's no check to verify that `String_cat()` returns a non-NULL value before using `path` in the following function calls.

4. **Line 354**:

```c
PCPDynamicScreen* dynamicScreen = Hashtable_get(screens, key);
if (!screen)
   return;
```

There's a logic error here - the check should be for `dynamicScreen`, not `screen`. If `dynamicScreen` is NULL, the function continues and accesses `dynamicScreen->totalColumns` in the for loop, which would cause an NPD.

True Positive

5. **Line 347-349**:

```c
bool success;
unsigned int key;
success = DynamicScreen_search(screens, screen, &key);
```

There is no check to ensure that `screens` or `screen` is not NULL before passing them to `DynamicScreen_search()`.

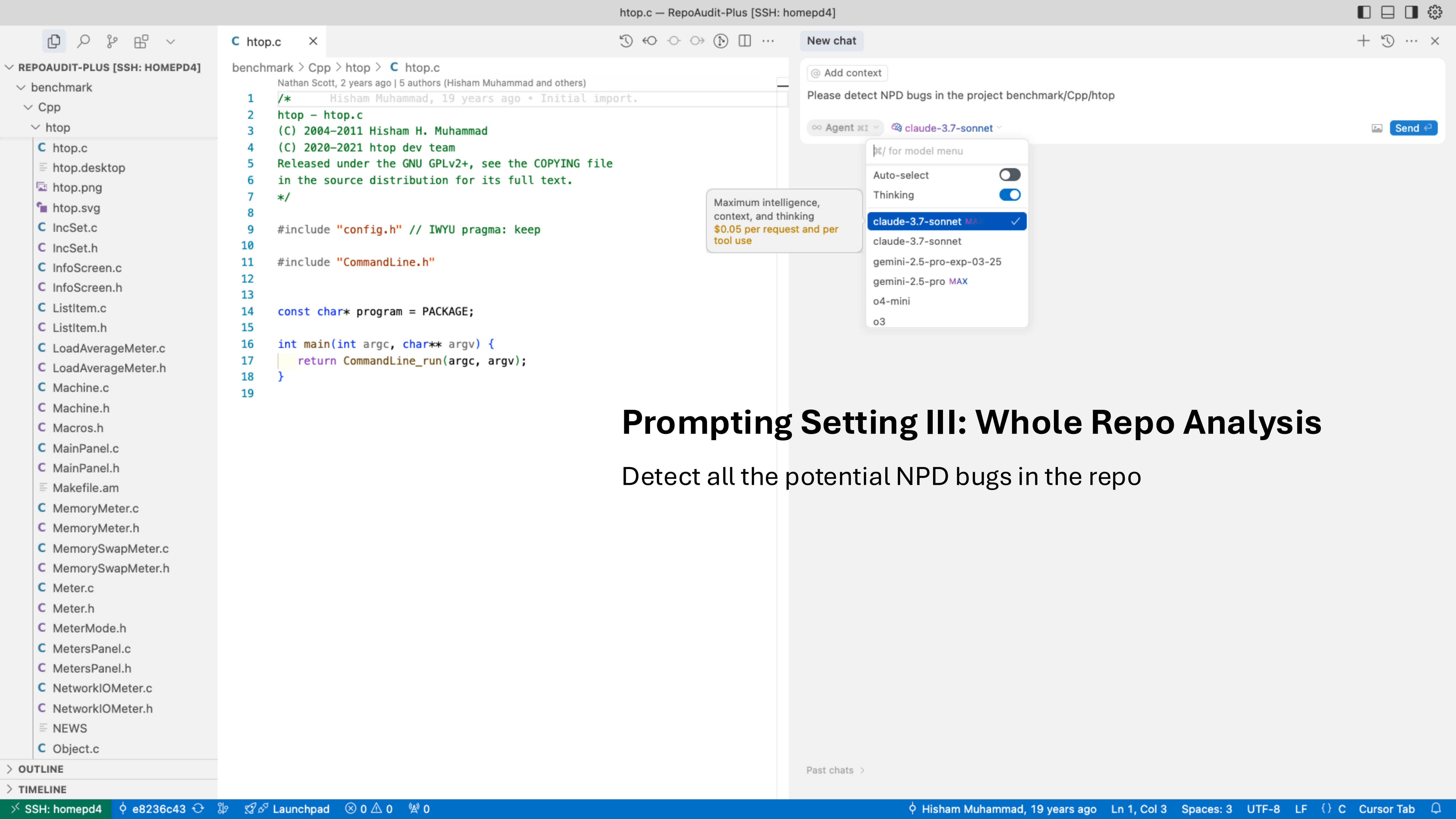
These are the most evident null pointer dereference bugs in the provided code. The code would benefit from additional null checks, especially when dealing with functions that may return NULL values or when accessing pointer members.

False Positive

Lack of calling contexts

Prompting Setting III: Whole Repo Analysis

Detect all the potential NPD bugs in the repo



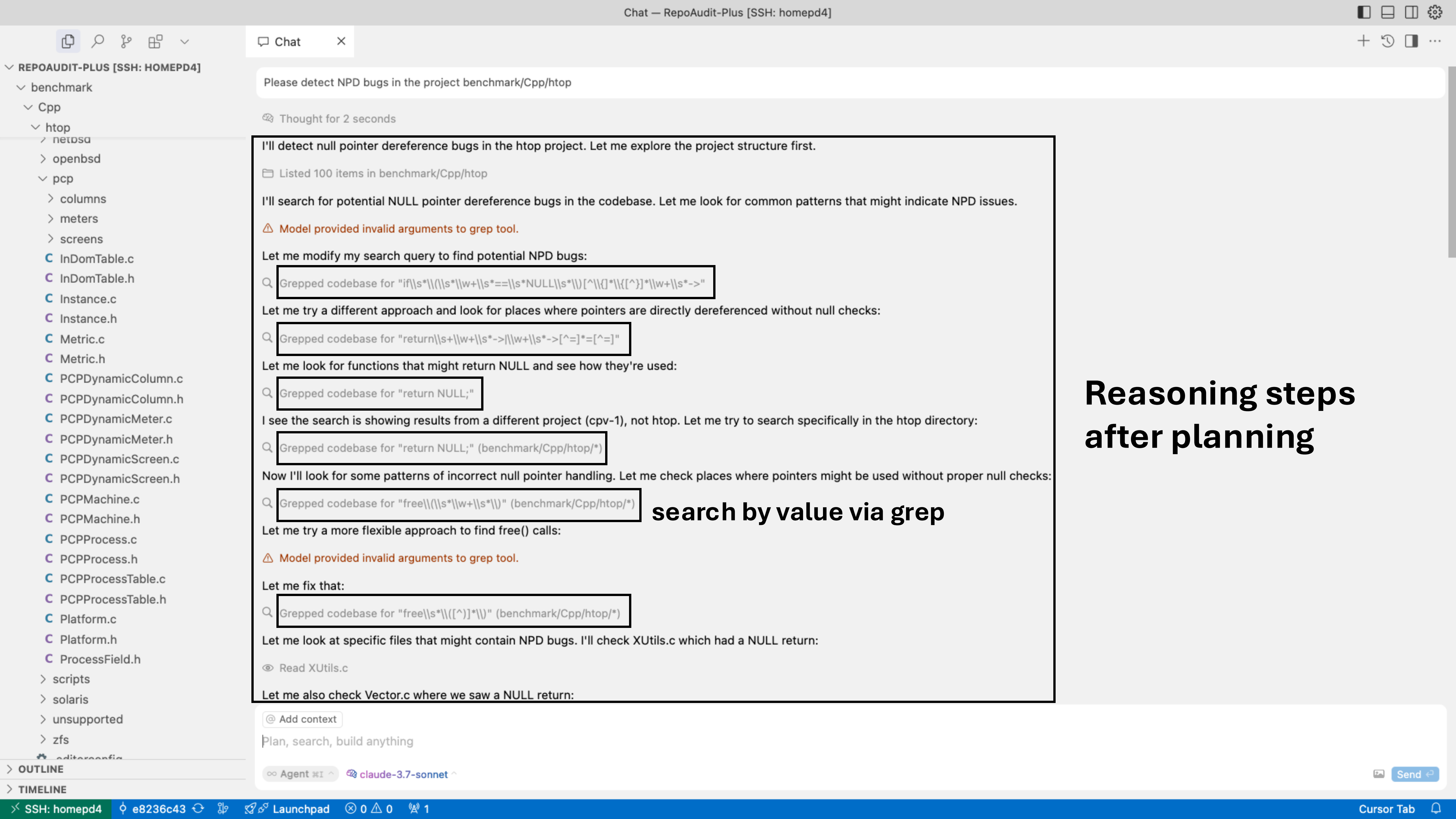
Please detect NPD bugs in the project benchmark/Cpp/htop

Auto-select ○

Thinking ●

Maximum intelligence, context, and thinking
$0.05 per request and per tool use

```
/*
htop - htop.c
(C) 2004-2011 Hisham H. Muhammad
(C) 2020-2021 htop dev team
Released under the GNU GPLv2+, see the COPYING file
in the source distribution for its full text.
*/

#include "config.h" // IWYU pragma: keep

#include "CommandLine.h"


const char* program = PACKAGE;

int main(int argc, char** argv) {
   return CommandLine_run(argc, argv);
}

```



Please detect NPD bugs in the project benchmark/Cpp/htop

Thought for 2 seconds

I'll detect null pointer dereference bugs in the htop project. Let me explore the project structure first.

Listed 100 items in benchmark/Cpp/htop

I'll search for potential NULL pointer dereference bugs in the codebase. Let me look for common patterns that might indicate NPD issues.

⚠ Model provided invalid arguments to grep tool.

Let me modify my search query to find potential NPD bugs:

`Grepped codebase for "if\\s*\\(\\s*\\w+\\s*==\\s*NULL\\s*\\)[^\\{]*\\{[^}]*\\w+\\s*->"`

Let me try a different approach and look for places where pointers are directly dereferenced without null checks:

`Grepped codebase for "return\\s+\\w+\\s*->|\\w+\\s*->[^=]*=[^=]"`

Let me look for functions that might return NULL and see how they're used:

`Grepped codebase for "return NULL;"`

I see the search is showing results from a different project (cpv-1), not htop. Let me try to search specifically in the htop directory:

`Grepped codebase for "return NULL;" (benchmark/Cpp/htop/*)`

Now I'll look for some patterns of incorrect null pointer handling. Let me check places where pointers might be used without proper null checks:

`Grepped codebase for "free\\(\\s*\\w+\\s*\\)" (benchmark/Cpp/htop/*)`

search by value via grep

Let me try a more flexible approach to find free() calls:

⚠ Model provided invalid arguments to grep tool.

Let me fix that:

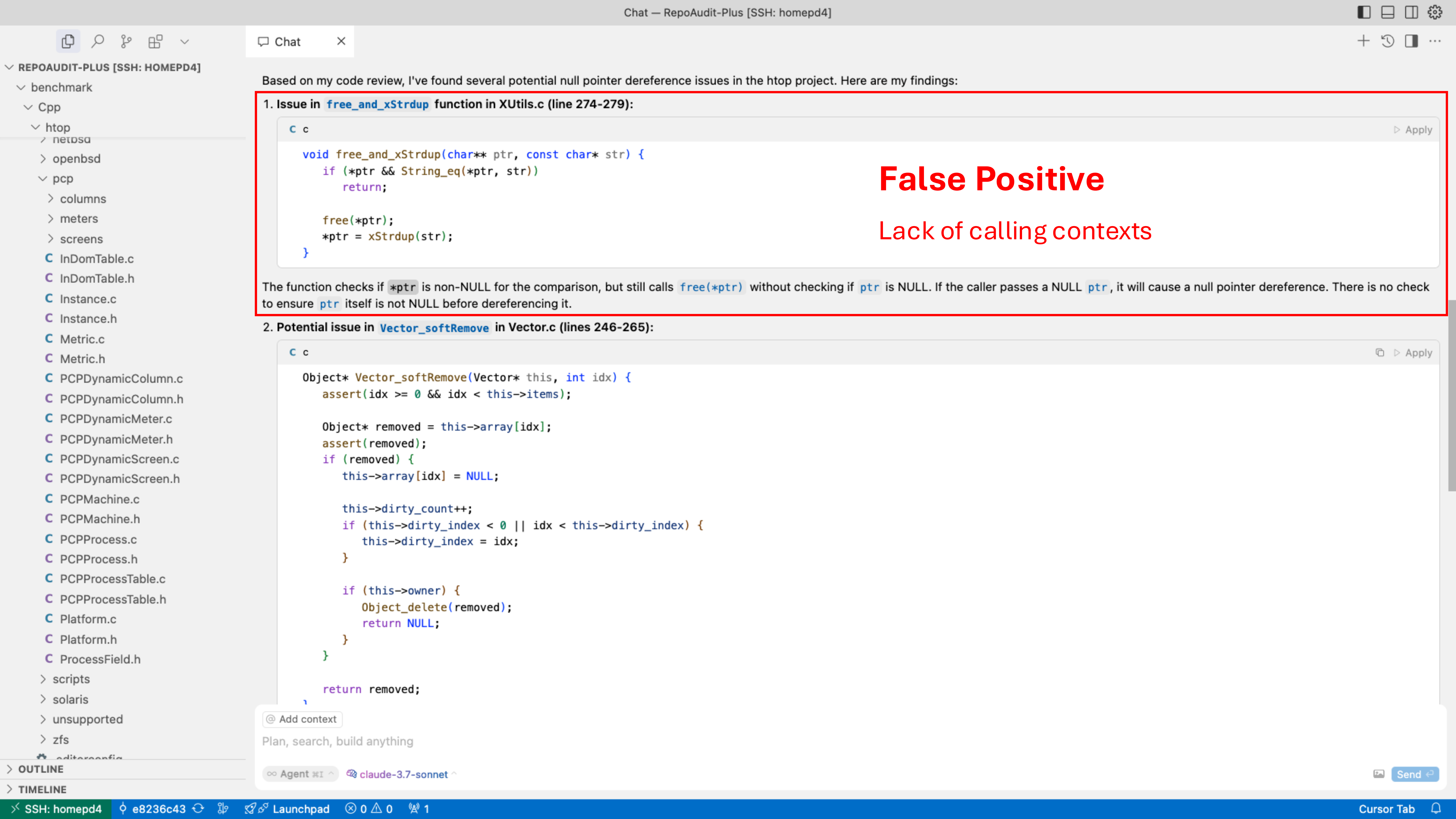
`Grepped codebase for "free\\s*\\([^)]*\\)" (benchmark/Cpp/htop/*)`

Let me look at specific files that might contain NPD bugs. I'll check XUtils.c which had a NULL return:

Read XUtils.c

Let me also check Vector.c where we saw a NULL return:

Reasoning steps after planning



Based on my code review, I've found several potential null pointer dereference issues in the htop project. Here are my findings:

1. **Issue in `free_and_xStrdup` function in XUtils.c (line 274-279):**

```c
void free_and_xStrdup(char** ptr, const char* str) {
   if (*ptr && String_eq(*ptr, str))
      return;

   free(*ptr);
   *ptr = xStrdup(str);
}
```

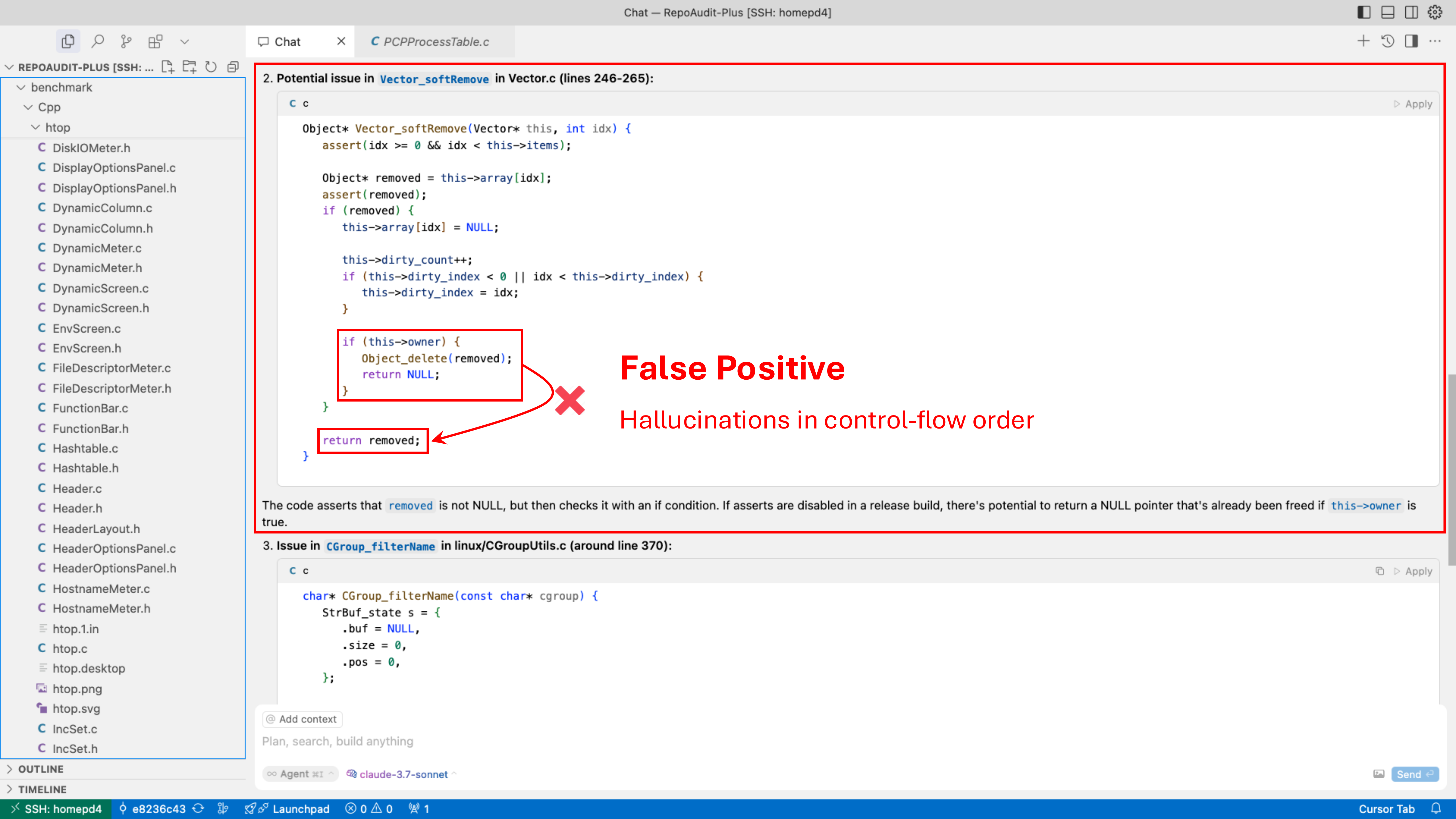
False Positive

Lack of calling contexts

The function checks if `*ptr` is non-NULL for the comparison, but still calls `free(*ptr)` without checking if `ptr` is NULL. If the caller passes a NULL `ptr`, it will cause a null pointer dereference. There is no check to ensure `ptr` itself is not NULL before dereferencing it.

2. **Potential issue in `Vector_softRemove` in Vector.c (lines 246-265):**

```c
Object* Vector_softRemove(Vector* this, int idx) {
   assert(idx >= 0 && idx < this->items);

   Object* removed = this->array[idx];
   assert(removed);
   if (removed) {
      this->array[idx] = NULL;

      this->dirty_count++;
      if (this->dirty_index < 0 || idx < this->dirty_index) {
         this->dirty_index = idx;
      }

      if (this->owner) {
         Object_delete(removed);
         return NULL;
      }
   }

   return removed;
```



2. **Potential issue in `Vector_softRemove` in Vector.c (lines 246-265):**

```c
Object* Vector_softRemove(Vector* this, int idx) {
   assert(idx >= 0 && idx < this->items);

   Object* removed = this->array[idx];
   assert(removed);
   if (removed) {
      this->array[idx] = NULL;

      this->dirty_count++;
      if (this->dirty_index < 0 || idx < this->dirty_index) {
         this->dirty_index = idx;
      }

      if (this->owner) {
         Object_delete(removed);
         return NULL;
      }
   }

   return removed;
}
```

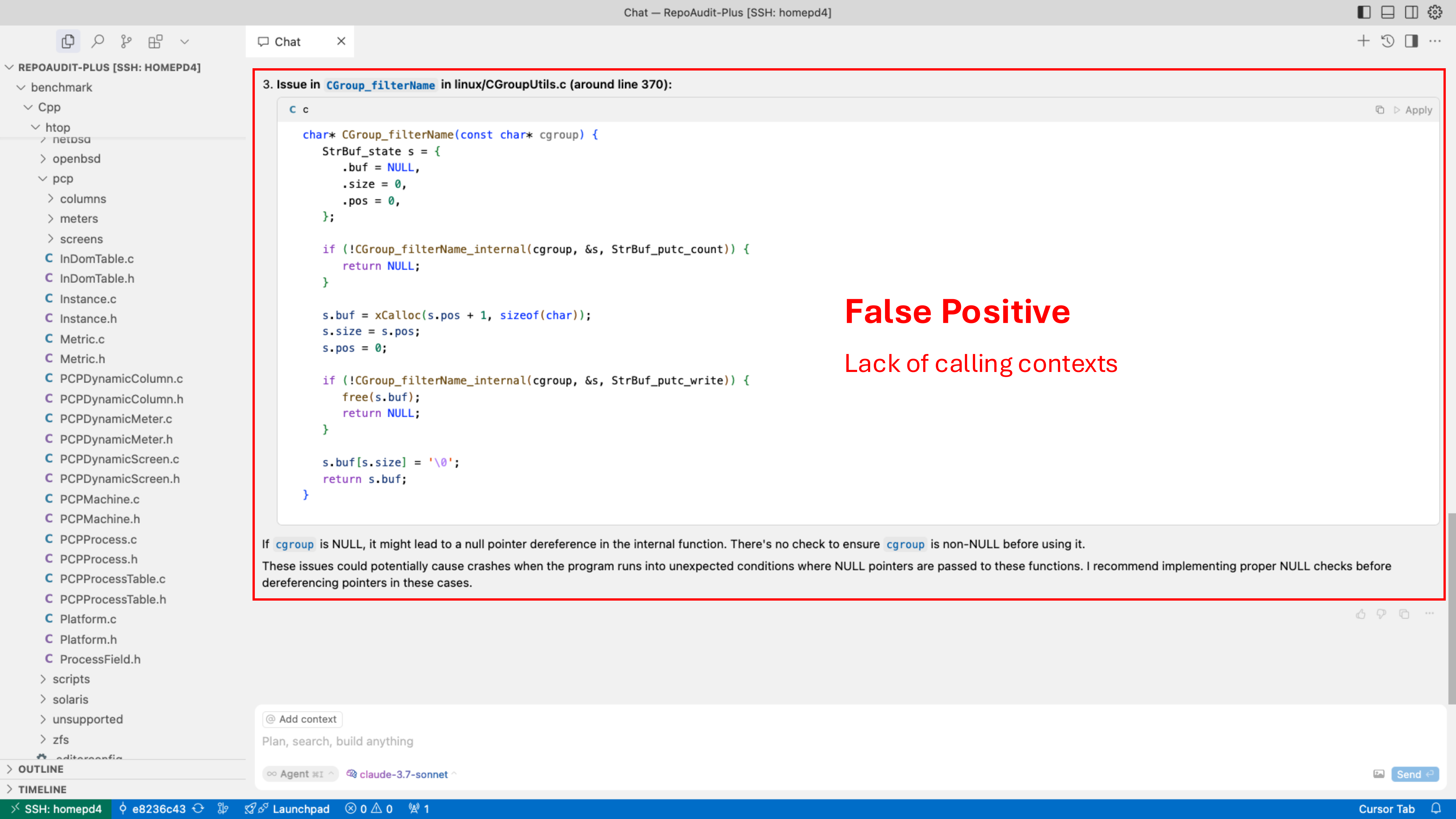
False Positive

Hallucinations in control-flow order

The code asserts that `removed` is not NULL, but then checks it with an if condition. If asserts are disabled in a release build, there's potential to return a NULL pointer that's already been freed if `this->owner` is true.

3. **Issue in `CGroup_filterName` in linux/CGroupUtils.c (around line 370):**

```c
char* CGroup_filterName(const char* cgroup) {
   StrBuf_state s = {
      .buf = NULL,
      .size = 0,
      .pos = 0,
   };
```



3. Issue in `CGroup_filterName` in linux/CGroupUtils.c (around line 370):

```c
char* CGroup_filterName(const char* cgroup) {
   StrBuf_state s = {
      .buf = NULL,
      .size = 0,
      .pos = 0,
   };

   if (!CGroup_filterName_internal(cgroup, &s, StrBuf_putc_count)) {
      return NULL;
   }

   s.buf = xCalloc(s.pos + 1, sizeof(char));
   s.size = s.pos;
   s.pos = 0;

   if (!CGroup_filterName_internal(cgroup, &s, StrBuf_putc_write)) {
      free(s.buf);
      return NULL;
   }

   s.buf[s.size] = '\0';
   return s.buf;
}
```

If `cgroup` is NULL, it might lead to a null pointer dereference in the internal function. There's no check to ensure `cgroup` is non-NULL before using it.

These issues could potentially cause crashes when the program runs into unexpected conditions where NULL pointers are passed to these functions. I recommend implementing proper NULL checks before dereferencing pointers in these cases.

False Positive

Lack of calling contexts

Several Observations

- Cursor with Claude-3.7-Sonnet Max has planning ability
  - Plan different reasoning steps for different auditing requests.
- Tool using in Claude-3.7-Sonnet Max is very simple
  - Mainly grep-like retrieval and indexing-based call graph analysis
- Claude-3.7-Sonnet Max tends to introduce aggressive assumptions for bug detection
  - The return values of many function calls are regarded to be NULL
  - Do not analyze inter-procedural data-flow facts