NYU | TANDON SCHOOL OF ENGINEERING

LLMs in Security: Demos vs Deployment?

Case studies in vulnerability repair and reverse engineering

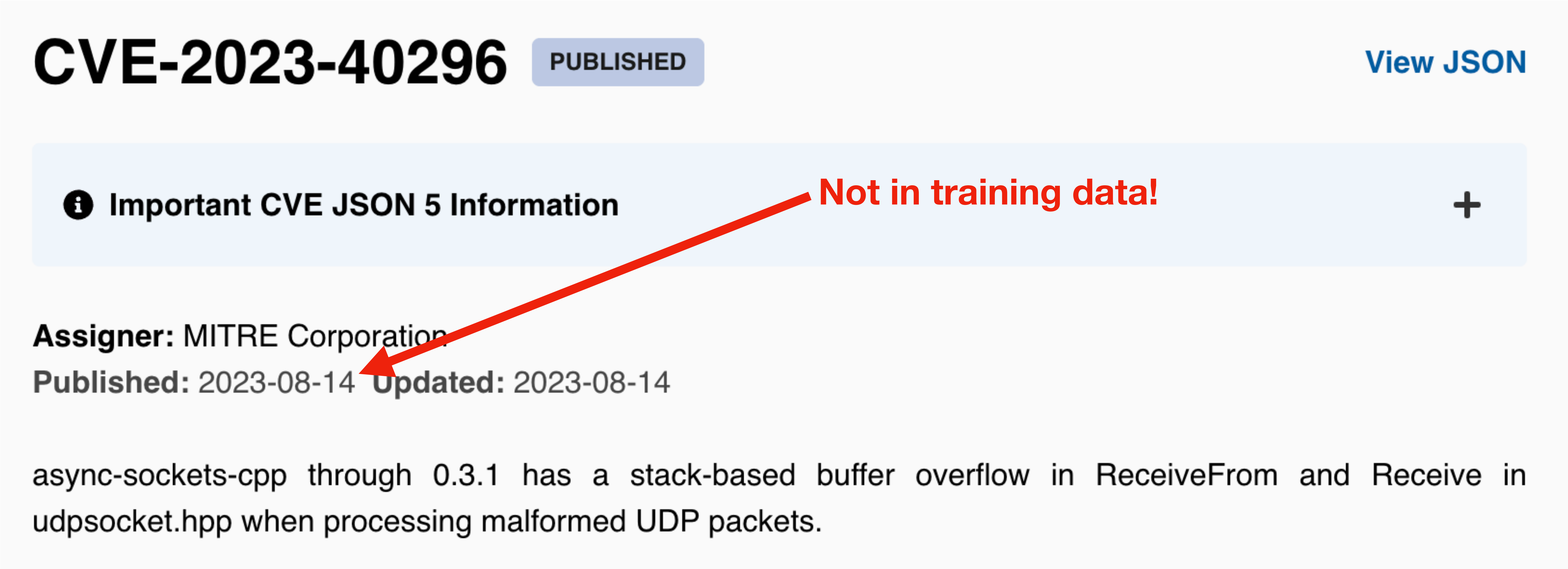
Brendan Dolan-Gavitt
National Academy of Sciences Forum on Cyber Resilience
August 31, 2023

CENTER FOR CYBER SECURITY

How Useful Are LLMs in Security Today?

- LLMs, and particularly GPT4, have produced some *great* security demos
  - Finding bugs, deobfuscation, reverse engineering, fixing vulnerabilities
- Over the past two years, we have done some systematic evaluation of LLMs on security tasks:
  - Vulnerability repair (S&P 2023; https://arxiv.org/abs/2112.02125)
  - Reverse engineering (preprint; https://arxiv.org/abs/2202.01142)
- When evaluated systematically, **LLMs sometimes don't live up to the promise of the initial demos**

Demo: Repairing CVE-2023-40296



CVE-2023-40296

Not in training data!

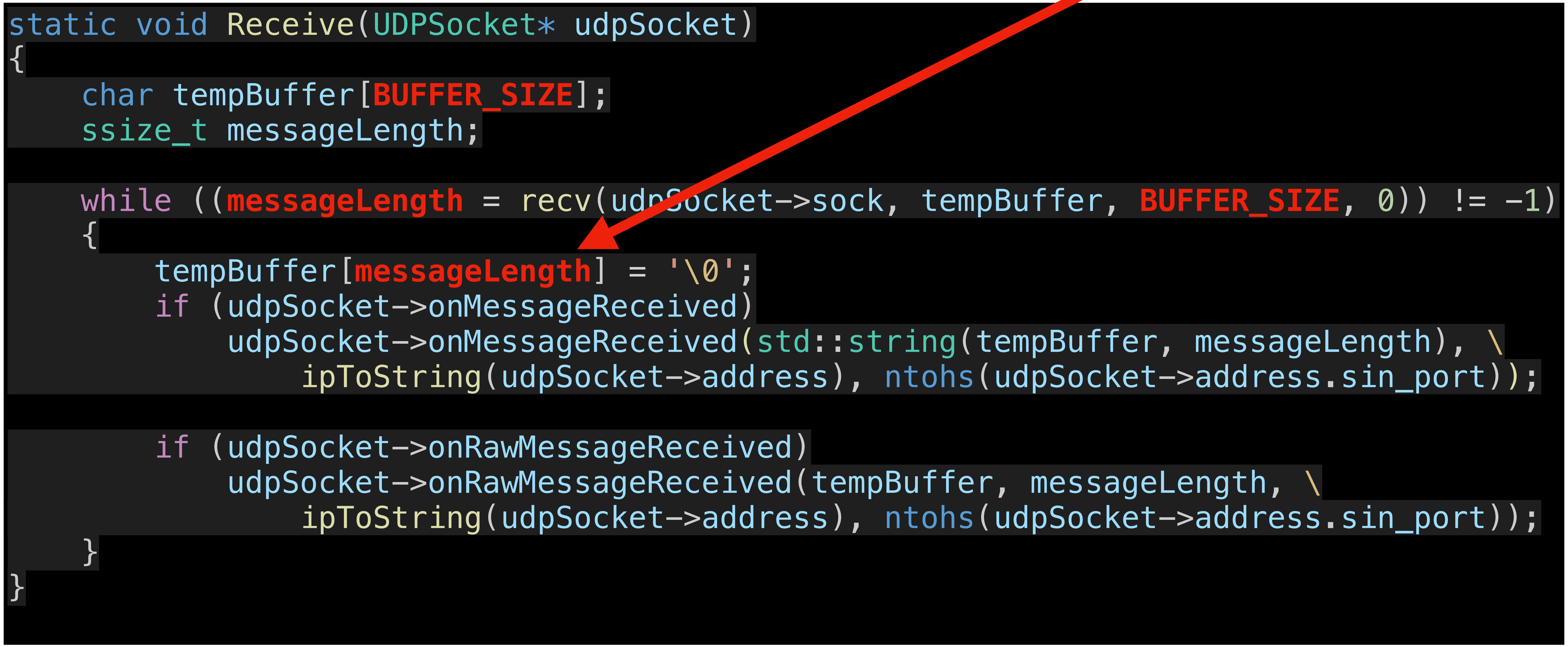
Assigner: MITRE Corporation
Published: 2023-08-14 **Updated:** 2023-08-14

async-sockets-cpp through 0.3.1 has a stack-based buffer overflow in ReceiveFrom and Receive in udpsocket.hpp when processing malformed UDP packets.

The Vulnerability

A classic off-by-one

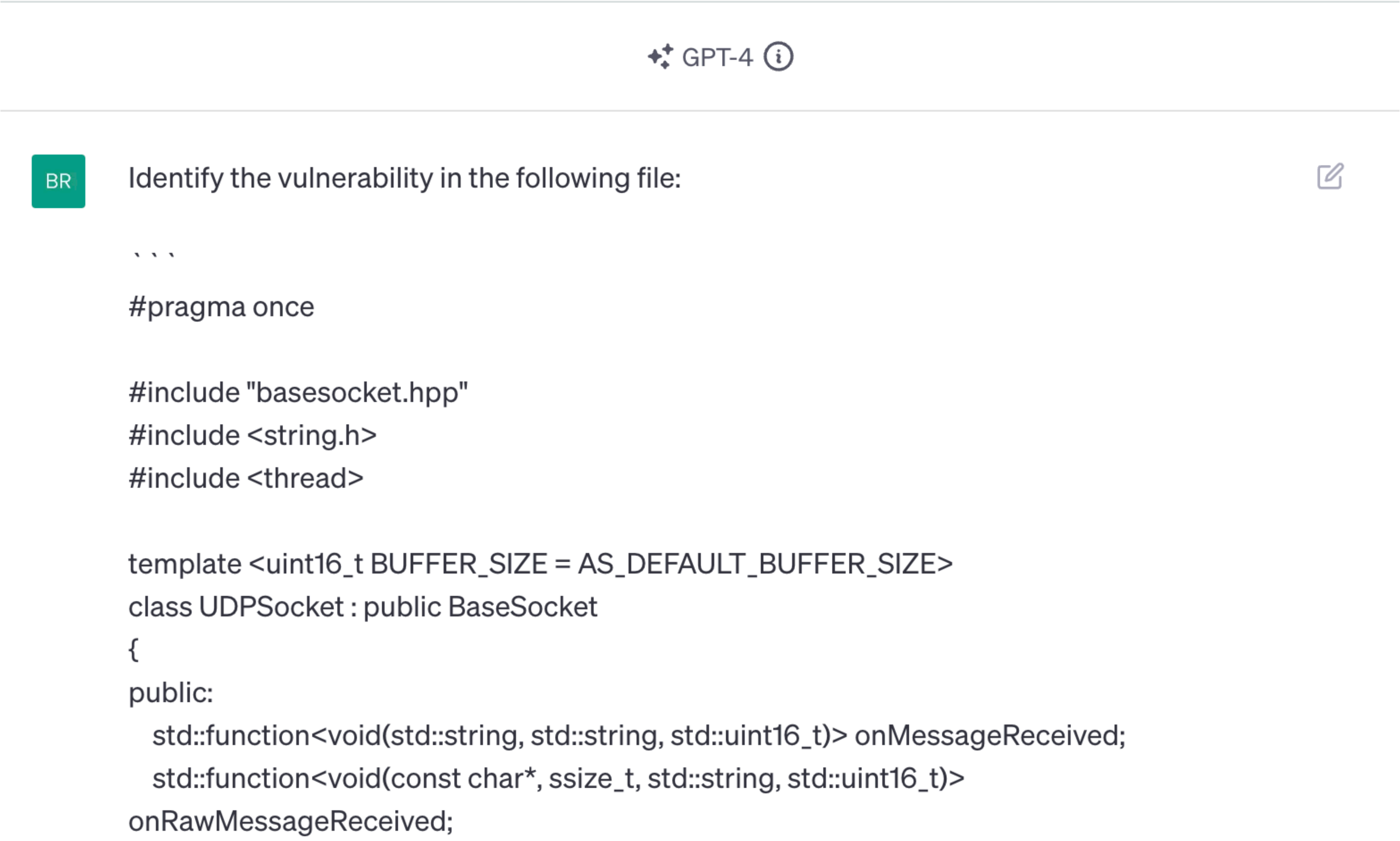
If recv() returns BUFFER_SIZE bytes, a NULL is written one byte past the end of tempBuffer



```
static void Receive(UDPSocket* udpSocket)
{
    char tempBuffer[BUFFER_SIZE];
    ssize_t messageLength;

    while ((messageLength = recv(udpSocket->sock, tempBuffer, BUFFER_SIZE, 0)) != -1)
    {
        tempBuffer[messageLength] = '\0';
        if (udpSocket->onMessageReceived)
            udpSocket->onMessageReceived(std::string(tempBuffer, messageLength), \
                ipToString(udpSocket->address), ntohs(udpSocket->address.sin_port));

        if (udpSocket->onRawMessageReceived)
            udpSocket->onRawMessageReceived(tempBuffer, messageLength, \
                ipToString(udpSocket->address), ntohs(udpSocket->address.sin_port));
    }
}
```

Asking ChatGPT4

Full transcript: https://chat.openai.com/share/c382940f-14e9-4fdd-97df-52ee4429854e

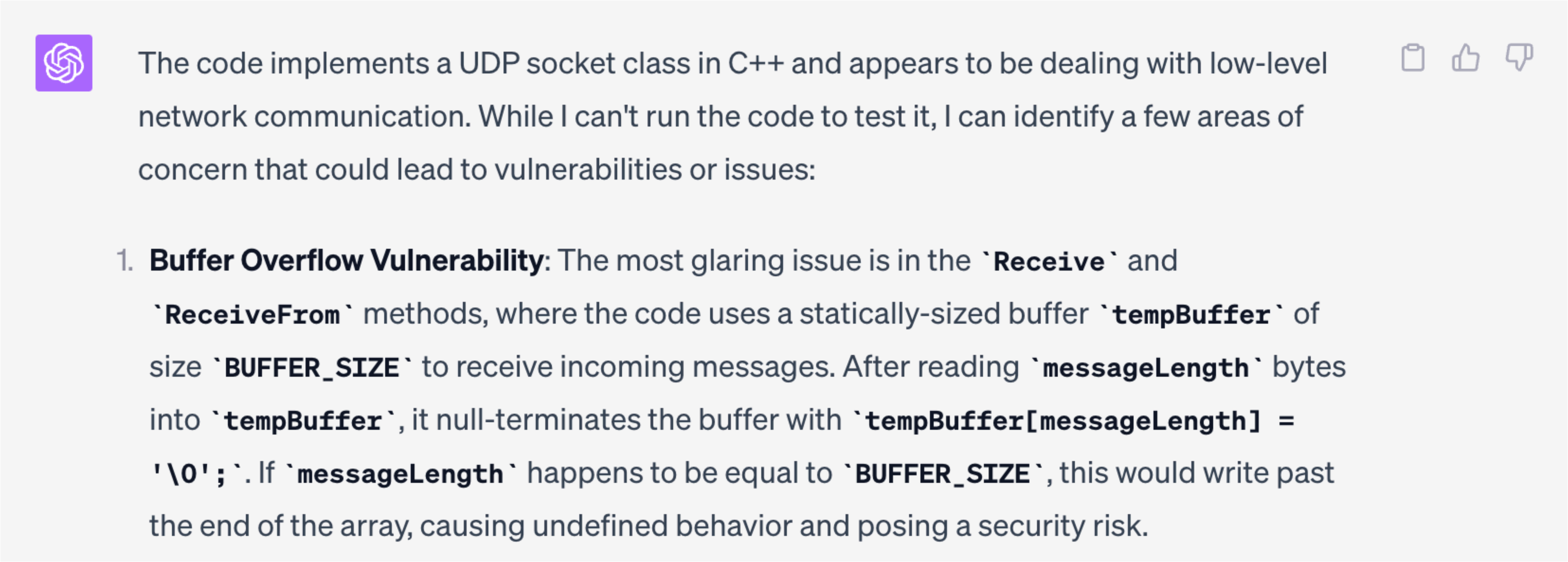


Identify the vulnerability in the following file:

~~~
```
#pragma once

#include "basesocket.hpp"
#include <string.h>
#include <thread>

template <uint16_t BUFFER_SIZE = AS_DEFAULT_BUFFER_SIZE>
class UDPSocket : public BaseSocket
{
public:
  std::function<void(std::string, std::string, std::uint16_t)> onMessageReceived;
  std::function<void(const char*, ssize_t, std::string, std::uint16_t)> onRawMessageReceived;
~~~

[…]

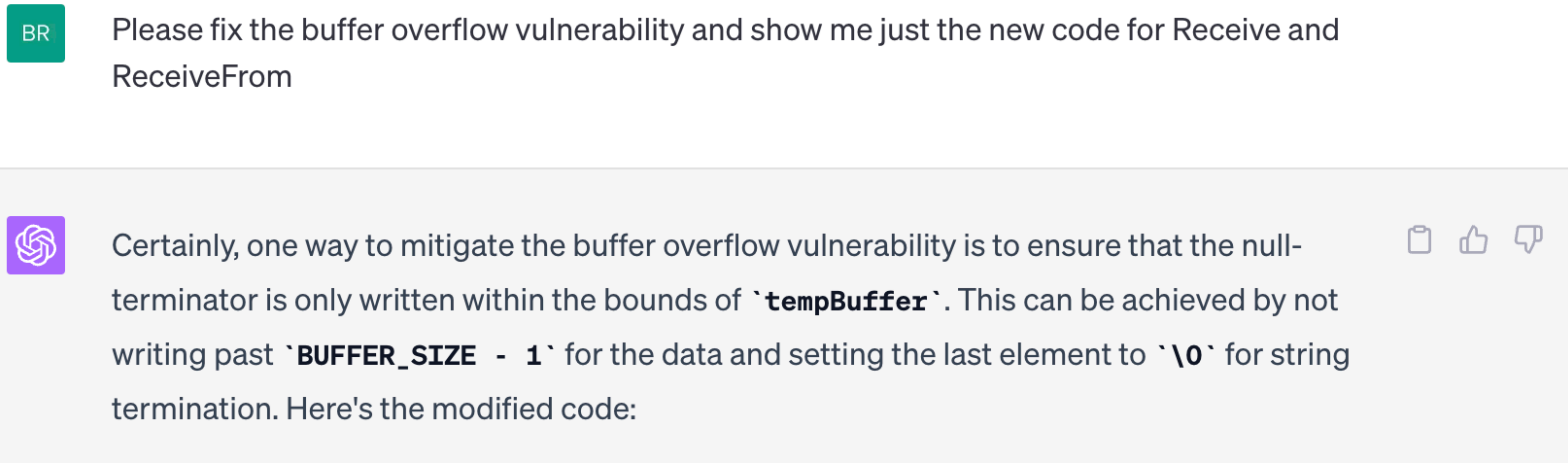
ChatGPT4: Vulnerability Discovery



The code implements a UDP socket class in C++ and appears to be dealing with low-level network communication. While I can't run the code to test it, I can identify a few areas of concern that could lead to vulnerabilities or issues:

1. **Buffer Overflow Vulnerability**: The most glaring issue is in the `Receive` and `ReceiveFrom` methods, where the code uses a statically-sized buffer `tempBuffer` of size `BUFFER_SIZE` to receive incoming messages. After reading `messageLength` bytes into `tempBuffer`, it null-terminates the buffer with `tempBuffer[messageLength] = '\0';`. If `messageLength` happens to be equal to `BUFFER_SIZE`, this would write past the end of the array, causing undefined behavior and posing a security risk.

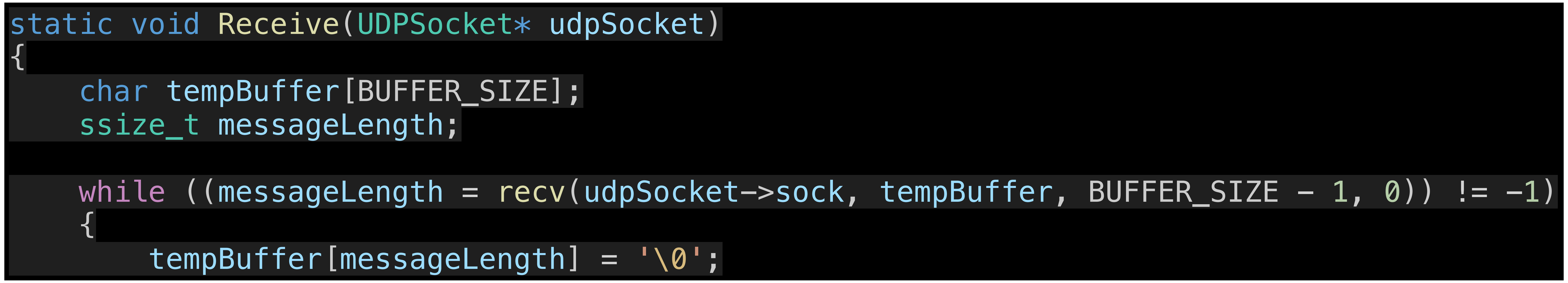
ChatGPT4 Repair



BR

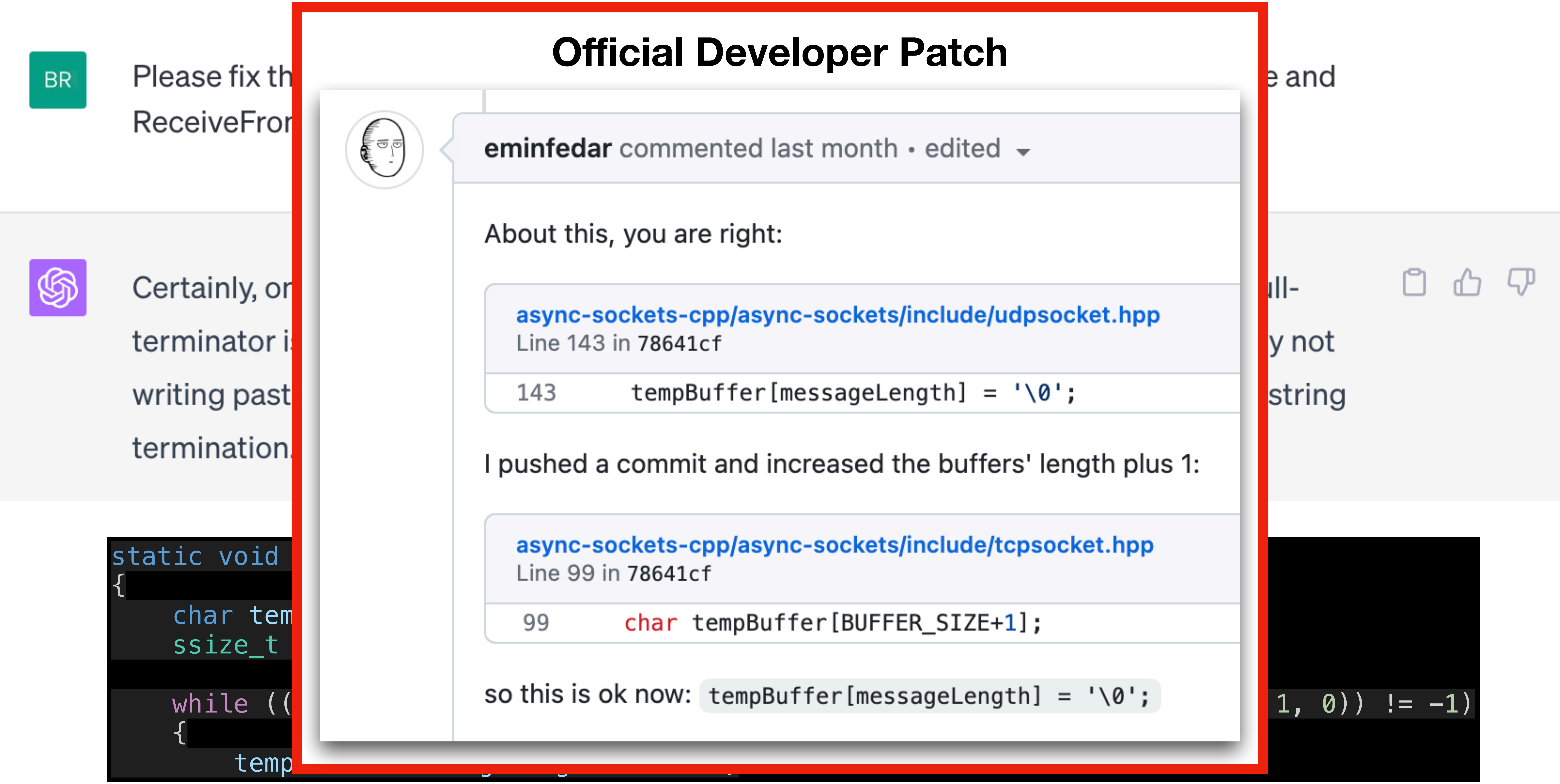
Please fix the buffer overflow vulnerability and show me just the new code for Receive and ReceiveFrom

Certainly, one way to mitigate the buffer overflow vulnerability is to ensure that the null-terminator is only written within the bounds of `tempBuffer`. This can be achieved by not writing past `BUFFER_SIZE - 1` for the data and setting the last element to `\0` for string termination. Here's the modified code:



```
static void Receive(UDPSocket* udpSocket)
{
    char tempBuffer[BUFFER_SIZE];
    ssize_t messageLength;

    while ((messageLength = recv(udpSocket->sock, tempBuffer, BUFFER_SIZE - 1, 0)) != -1)
    {
        tempBuffer[messageLength] = '\0';
```

ChatGPT4 Repair



BR Please fix th … e and
ReceiveFro

Certainly, o … ll-
terminator i … y not
writing past … string
termination.

```
static void
{
    char tem
    ssize_t

    while ((                                      1, 0)) != -1)
    {
        temp
```

Official Developer Patch

eminfedar commented last month • edited

About this, you are right:

async-sockets-cpp/async-sockets/include/udpsocket.hpp
Line 143 in 78641cf

```
143        tempBuffer[messageLength] = '\0';
```

I pushed a commit and increased the buffers' length plus 1:

async-sockets-cpp/async-sockets/include/tcpsocket.hpp
Line 99 in 78641cf

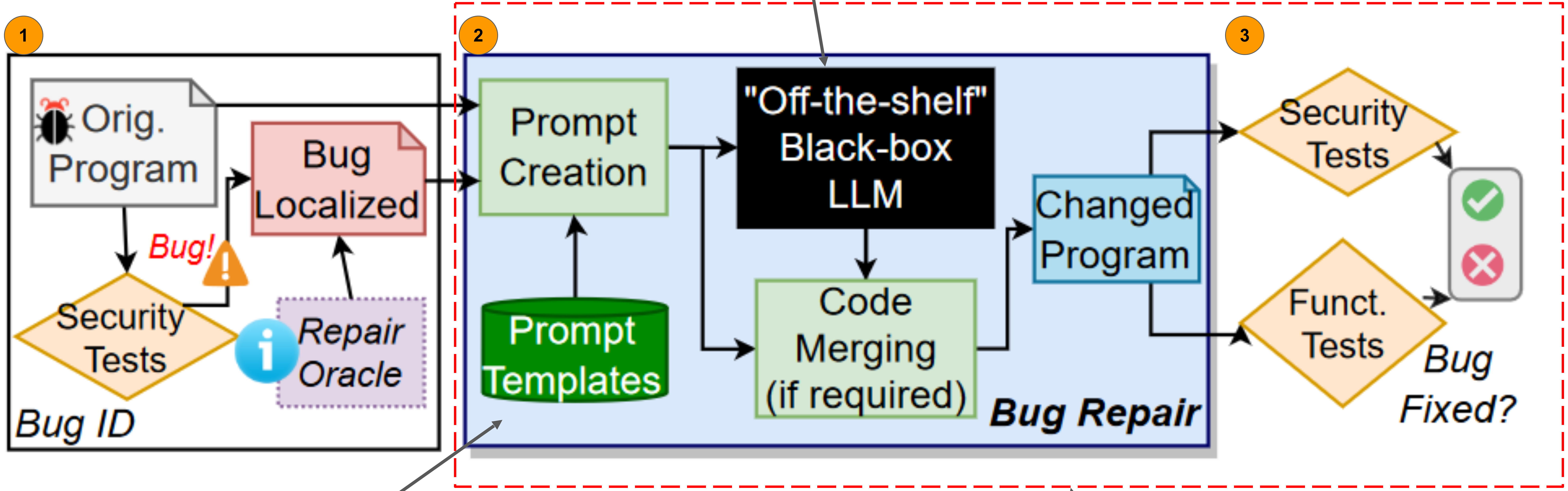
```
99        char tempBuffer[BUFFER_SIZE+1];
```

so this is ok now: `tempBuffer[messageLength] = '\0';`

Systematic Study: A Repair Framework

From "<u>Examining Zero-Shot Vulnerability Repair with Large Language Models</u>", Pearce et al., IEEE S&P 2023

Use an ensemble of LLMs



Use a variety of Prompt Templates

Language Models are non-deterministic:
Repeat until successfully passing both tests?

Promising Results?

Real-world vulns from ExtractFix dataset

	LLM	EF
EF01: LibTIFF Out-of-bounds read	✓	✓
EF02a: LibTIFF Out-of-bounds write	✗	✓
EF02b: LibTIFF Out-of-bounds write	✗	✗
EF07: LibTIFF Off-by-one error	✓	✓
EF08: LibTIFF Shift exp. type error	✓	✓
EF09: LibTIFF DoS by divide by zero	✓	✓
EF10: LibTIFF DoS by divide by zero	✓	✓
EF15: LibXML2 Buffer over-read	✓	✓
EF17: LibXML2 Buffer underflow	✓	✓
EF18: LibXML2 Null pointer deref.	✗	✓
EF20: LibJPEG Buffer over-read	✓	✗
EF22: LibJPEG Buffer overflow	✗	✓

- 12 real-world CVEs:
  - **432** Model/Prompt combinations
  - 19,600 attempted patches,
  - 982 repairs,
  - **8 of 12 scenarios repaired by ensemble.**
    - Cushman-001: 8/12
    - Davinci-001: 7/12
    - Davinci-002: 7/12
    - J1-large: 5/12
    - GPT-CSRC (ours): 4/12
    - Polycoder: 6/12

Reality Check: Do the Patches Work?

(Remember GenProg/RSRepair from program repair?)

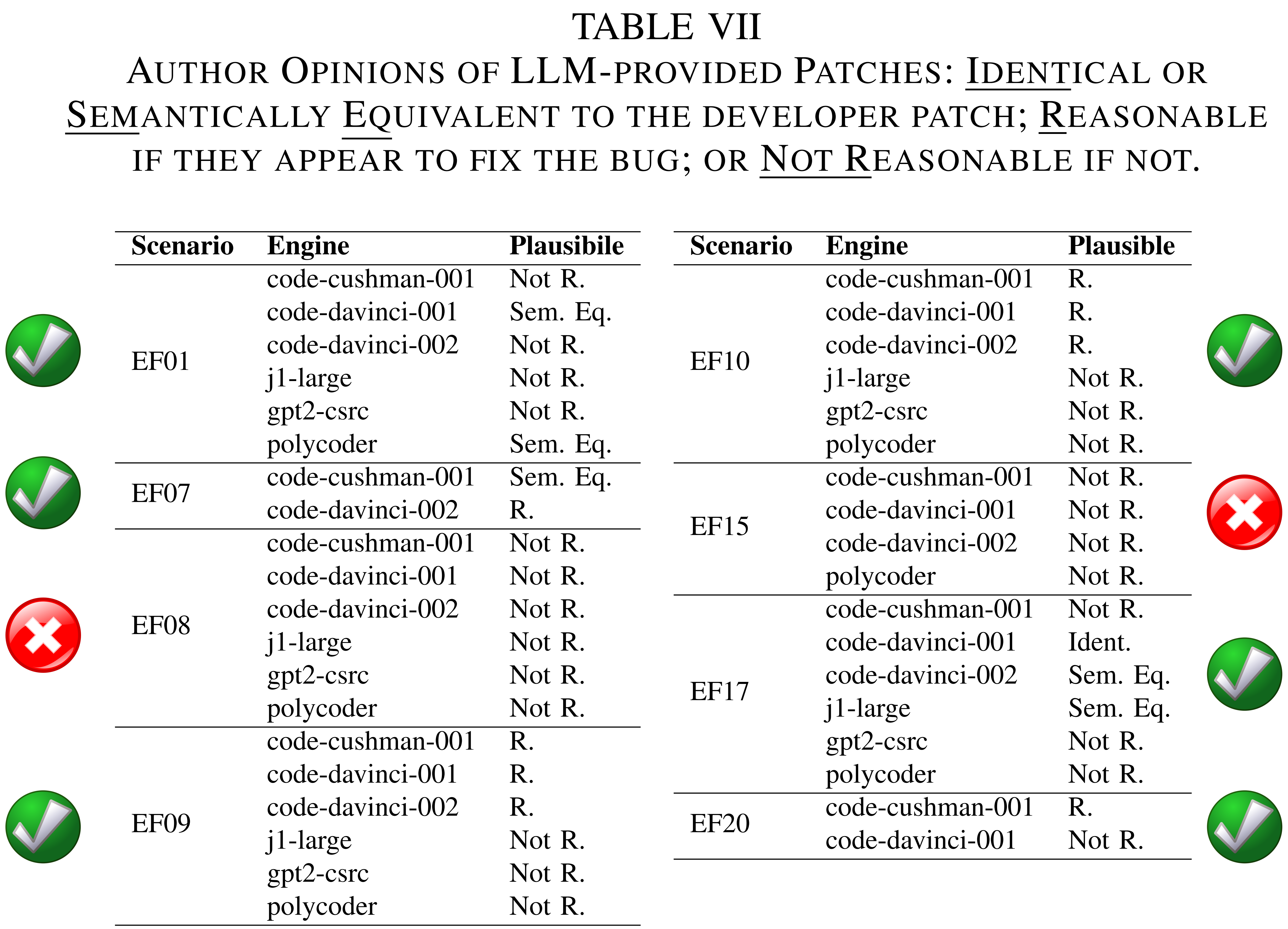


TABLE VII

Author Opinions of LLM-provided Patches: Identical or Semantically Equivalent to the developer patch; Reasonable if they appear to fix the bug; or Not Reasonable if not.

	Scenario	Engine	Plausibile
✓	EF01	code-cushman-001	Not R.
		code-davinci-001	Sem. Eq.
		code-davinci-002	Not R.
		j1-large	Not R.
		gpt2-csrc	Not R.
		polycoder	Sem. Eq.
✓	EF07	code-cushman-001	Sem. Eq.
		code-davinci-002	R.
✗	EF08	code-cushman-001	Not R.
		code-davinci-001	Not R.
		code-davinci-002	Not R.
		j1-large	Not R.
		gpt2-csrc	Not R.
		polycoder	Not R.
✓	EF09	code-cushman-001	R.
		code-davinci-001	R.
		code-davinci-002	R.
		j1-large	Not R.
		gpt2-csrc	Not R.
		polycoder	Not R.

Scenario	Engine	Plausible	
EF10	code-cushman-001	R.	✓
	code-davinci-001	R.	
	code-davinci-002	R.	
	j1-large	Not R.	
	gpt2-csrc	Not R.	
	polycoder	Not R.	
EF15	code-cushman-001	Not R.	✗
	code-davinci-001	Not R.	
	code-davinci-002	Not R.	
	polycoder	Not R.	
EF17	code-cushman-001	Not R.	✓
	code-davinci-001	Ident.	
	code-davinci-002	Sem. Eq.	
	j1-large	Sem. Eq.	
	gpt2-csrc	Not R.	
	polycoder	Not R.	
EF20	code-cushman-001	R.	✓
	code-davinci-001	Not R.	

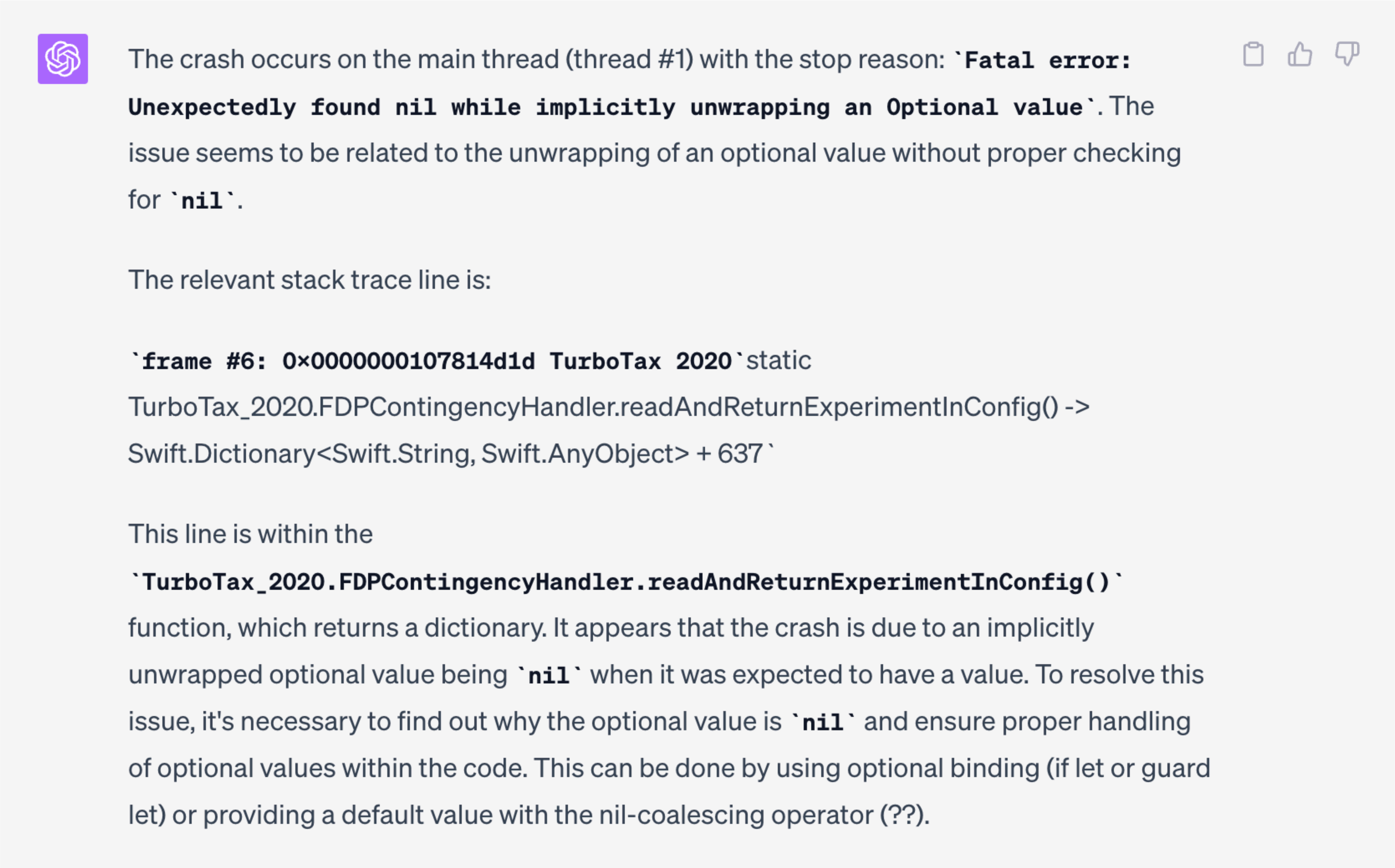
- Testing cannot verify *absence* of bugs
- Manual inspection of top-scoring ‘fixes’ reveals that many fixes ‘**unreasonable**’
- Reduces ‘success’ to 6 of 12 (50%).
- Davinci-001: 4/12
- Davinci-002: 4/12

Demo: Reverse Engineering

Full transcript: https://chat.openai.com/share/b41bcd92-80f8-4aa4-9086-e8aede516ace

- Anecdote: I was doing my taxes with TurboTax (closed source) on OS X, but it kept crashing at a particular point
- Using ChatGPT4, I was able to:
  - Get instructions for attaching the XCode debugger and getting a backtrace
  - Feed the backtrace to GPT4 to identify the problematic code
  - Decompile functions along the way to Objective C / Swift
  - Identify and fix the problem (a missing JSON file in the installation)
- (I don’t have much experience with OS X or Swift/Objective C reverse engineering)

Crash Source Identification



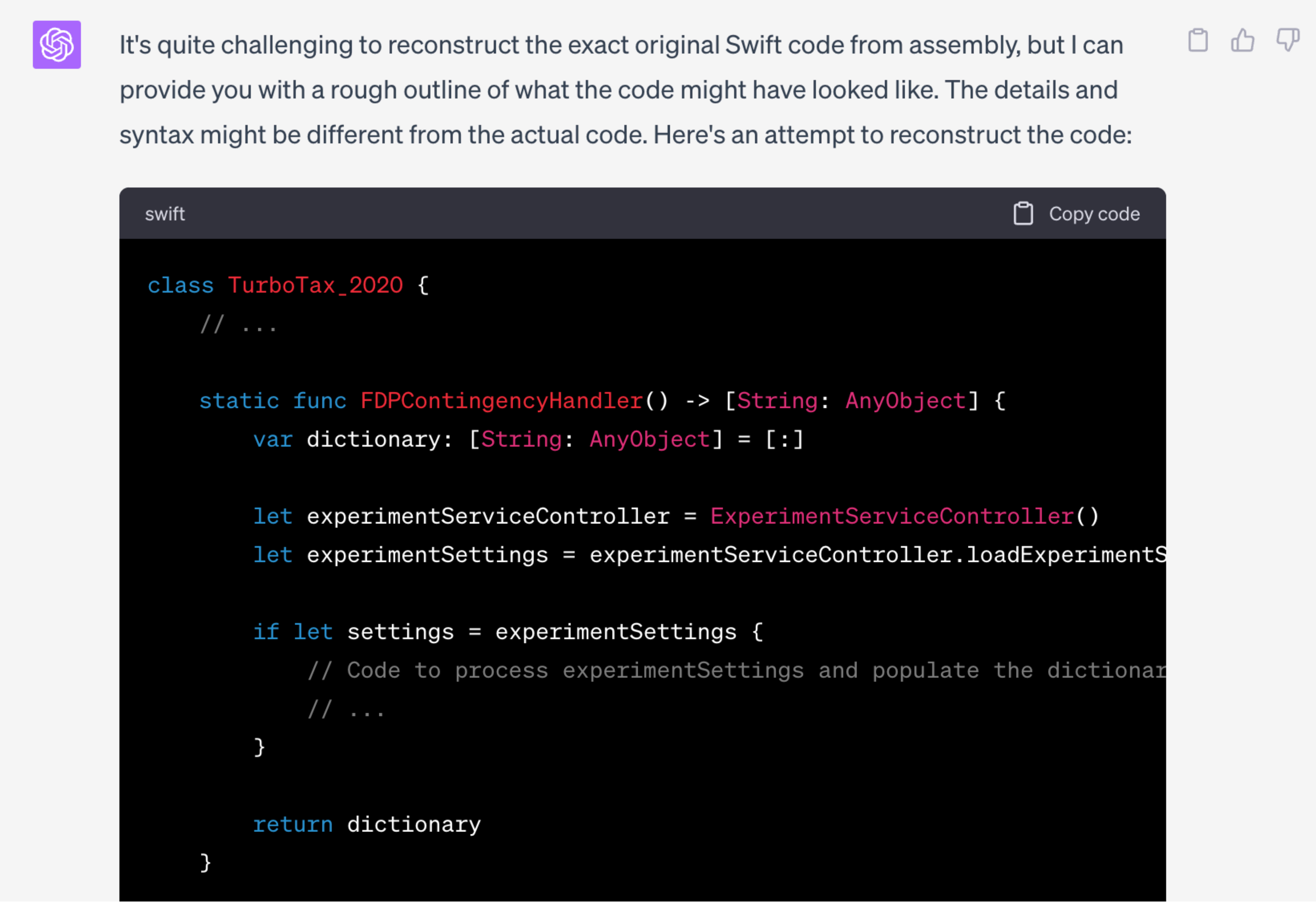
The crash occurs on the main thread (thread #1) with the stop reason: `Fatal error: Unexpectedly found nil while implicitly unwrapping an Optional value`. The issue seems to be related to the unwrapping of an optional value without proper checking for `nil`.

The relevant stack trace line is:

`frame #6: 0x0000000107814d1d TurboTax 2020`static TurboTax_2020.FDPContingencyHandler.readAndReturnExperimentInConfig() -> Swift.Dictionary<Swift.String, Swift.AnyObject> + 637`

This line is within the `TurboTax_2020.FDPContingencyHandler.readAndReturnExperimentInConfig()` function, which returns a dictionary. It appears that the crash is due to an implicitly unwrapped optional value being `nil` when it was expected to have a value. To resolve this issue, it's necessary to find out why the optional value is `nil` and ensure proper handling of optional values within the code. This can be done by using optional binding (if let or guard let) or providing a default value with the nil-coalescing operator (??).

Swift Decompilation

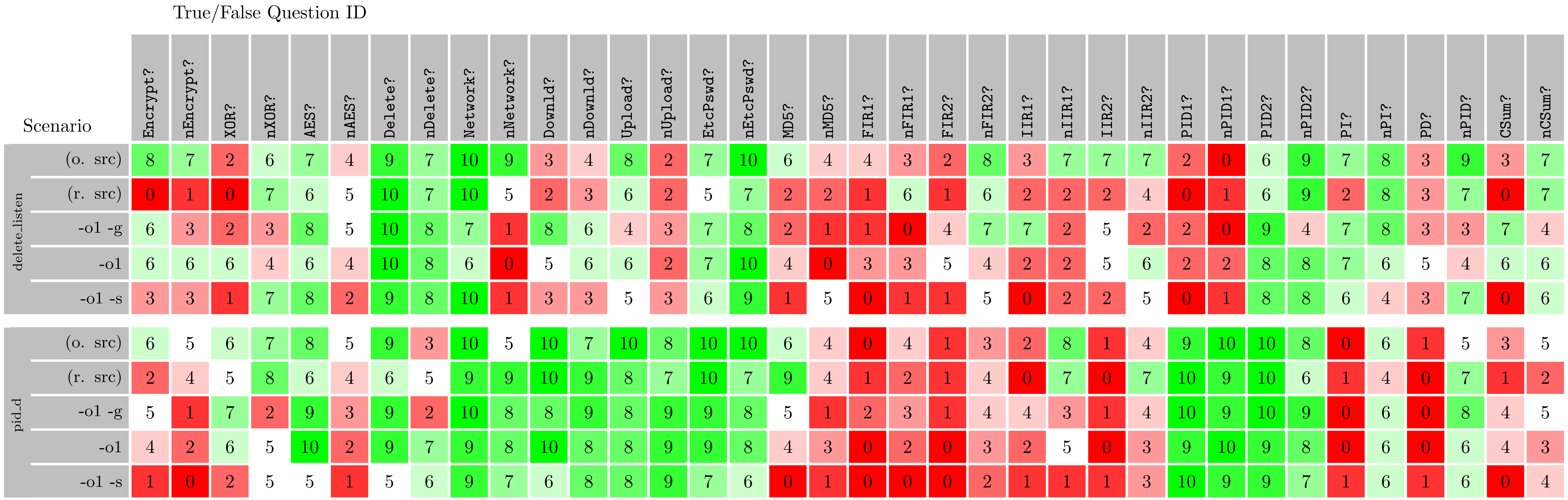


It's quite challenging to reconstruct the exact original Swift code from assembly, but I can provide you with a rough outline of what the code might have looked like. The details and syntax might be different from the actual code. Here's an attempt to reconstruct the code:

```swift
class TurboTax_2020 {
    // ...

    static func FDPContingencyHandler() -> [String: AnyObject] {
        var dictionary: [String: AnyObject] = [:]

        let experimentServiceController = ExperimentServiceController()
        let experimentSettings = experimentServiceController.loadExperimentS

        if let settings = experimentSettings {
            // Code to process experimentSettings and populate the dictionar
            // ...
        }

        return dictionary
    }
```

Systematic Study: Answering True/False Questions

From "Pop Quiz! Can a Large Language Model Help With Reverse Engineering?", Pearce et al. (arXiv preprint)



True/False Question ID

Scenario		Encrypt?	nEncrypt?	XOR?	nXOR?	AES?	nAES?	Delete?	nDelete?	Network?	nNetwork?	Downld?	nDownld?	Upload?	nUpload?	EtcPswd?	nEtcPswd?	MD5?	nMD5?	FIR1?	nFIR1?	FIR2?	nFIR2?	IIR1?	nIIR1?	IIR2?	nIIR2?	PID1?	nPID1?	PID2?	nPID2?	PI?	nPI?	PD?	nPD?	CSum?	nCSum?
delete_listen	(o. src)	8	7	2	6	7	4	9	7	10	9	3	4	8	2	7	10	6	4	4	3	2	8	3	7	7	7	2	0	6	9	7	8	3	9	3	7
	(r. src)	0	1	0	7	6	5	10	7	10	5	2	3	6	2	5	7	2	2	1	6	1	6	2	2	2	4	0	1	6	9	2	8	3	7	0	7
	-o1 -g	6	3	2	3	8	5	10	8	7	1	8	6	4	3	7	8	2	1	1	0	4	7	7	2	5	2	2	0	9	4	7	8	3	3	7	4
	-o1	6	6	6	4	6	4	10	8	6	0	5	6	6	2	7	10	4	0	3	3	5	4	2	2	5	6	2	2	8	8	7	6	5	4	6	6
	-o1 -s	3	3	1	7	8	2	9	8	10	1	3	3	5	3	6	9	1	5	0	1	1	5	0	2	2	5	0	1	8	8	6	4	3	7	0	6
pid_d	(o. src)	6	5	6	7	8	5	9	3	10	5	10	7	10	8	10	10	6	4	0	4	1	3	2	8	1	4	9	10	10	8	0	6	1	5	3	5
	(r. src)	2	4	5	8	6	4	6	5	9	9	10	9	8	7	10	7	9	4	1	2	1	4	0	7	0	7	10	9	10	6	1	4	0	7	1	2
	-o1 -g	5	1	7	2	9	3	9	2	10	8	8	9	8	9	9	8	5	1	2	3	1	4	4	3	1	4	10	9	10	9	0	6	0	8	4	5
	-o1	4	2	6	5	10	2	9	7	9	8	10	8	8	9	9	8	4	3	0	2	0	3	2	5	0	3	9	10	9	8	0	6	0	6	4	3
	-o1 -s	1	0	2	5	5	1	5	6	9	7	6	8	8	9	7	6	0	1	0	0	0	2	1	1	1	3	10	9	9	7	1	6	1	6	0	4

- Wrote small programs and asked Codex (DaVinci-001) **true/false** questions
- Both about source (with and w/o var rename) and Ghidra-decompiled code
- **Result: only ~53% accuracy overall – not much better than chance**

Conclusions

STARCODER

Code Llama

CodeGen2

replit-code
v1.3b

- Why the discrepancy?
  - Demos often focus on easy cases
  - Demos from **GPT4**; systematic experiments from **GPT3** and **Codex**
  - Demos have an expert **human in the loop**
- But: I remain **extremely optimistic** about the promise of LLMs in security applications
  - Current LLMs *generally were not fine-tuned on security tasks!*
  - Naïve extrapolation: from 2019→2023, we went from GPT2 to GPT4
  - Open models + advances in fine tuning (PEFT/LoRA) have made it much **cheaper** and **easier** to experiment with LLMs on domain specific data