

Using ChatGPT to Detect Vulnerabilities in Source Code

OCTOBER 4, 2023

Mark Sherman
Technical Director, Cybersecurity Foundations, CERT
Telephone: +1 412.268.9223
mssherman@cert.org



Copyright 2023 Carnegie Mellon University.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

Carnegie Mellon® and CERT® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM23-1035

Lots of Hype about Using ChatGPT for Coding

Okay, so ChatGPT just debugged my code. For real.

Not only can ChatGPT write code, it can read code. On one hand, that's very helpful. On the other hand, that's truly terrifying.



Written by David Geelitz, Senior Contributing Editor on April 3, 2023

<https://www.zdnet.com/article/okay-so-chatgpt-just-debugged-my-code-for-real/>

How to use ChatGPT to write code

By Allen Helton · March 22, 2023

Greetings, developers! Have you ever had a case of "coder's block" where you need a little help or inspiration to write code? Or need some help understanding code that you inherited from someone else? Well, you're in luck! From generating boilerplate code to debugging or explaining existing code, ChatGPT is a no-brainer way to be a faster, more efficient software engineer.

OpenAI's ChatGPT falls into the category of **generative AI**, which is a type of artificial intelligence that generates new or novel content based on data it's been trained on. That means ChatGPT can not only write essays, jokes and poems for you, it can also be your pair programmer.

So let's dive in and see how you can use it to work smarter, not harder.

<https://www.pluralsight.com/blog/software-development/how-use-chatgpt-programming-coding>

How good is ChatGPT at writing code?

Learn more about ChatGPT and discover how it can be used to generate programming solutions.

Are you looking for a new way to write code? ChatGPT is a revolutionary AI-based system that can help you generate programming solutions quickly and easily. In this article, we'll explain how effective ChatGPT is at writing code and whether or not it really lives up to its promises.

<https://botpress.com/blog/how-good-is-chatgpt-at-writing-code>

AI

ChatGPT Changed How I Write Software

By Allen Helton · 31 May 2023

AI is buzzing right now. All my social media feeds are about some new thing you can do with ChatGPT plugins or showing off a new photo created by generative AI. To be honest, it's pretty cool to see and I'm not mad about it.

<https://www.readyssetcloud.io/blog/allen.helton/chatgpt-changed-how-i-write-software/>

CERT Secure Coding Standards



Collected wisdom from thousands of contributors on community wiki since Spring 2006

<http://securecoding.cert.org>

- SEI CERT C Coding Standard
- SEI CERT C++ Coding Standard
- CERT Oracle Secure Coding Standard for Java



Secure Coding Training and Professional Certificates

- CERT Secure Coding in C and C++
- CERT Secure Coding in Java

International Standards Participation

- ISO/IEC C Programming Language
- ISO/IEC C++ Programming Language

Rule Organization – Noncompliant Code Example

Noncompliant Code Example

In this noncompliant code example, `memcmp()` is used to compare the contents of two structures, including any padding bytes:

```
#include <string.h>

struct s {
    char c;
    int i;
    char buffer[13];
};

void compare(const struct s *left, const struct s *right) {
    if ((left && right) &&
        (0 == memcmp(left, right, sizeof(struct s)))) {
        /* ... */
    }
}
```

Noncompliant Code
Don't try this at home!

Noncompliant code examples or antipatterns in a pink frame—do not copy and paste into your code. Most examples were found in the wild.

Rule Organization – Compliant Solution

Compliant Solution

In this compliant solution, all of the fields are compared manually to avoid comparing any padding bytes:

```
#include <string.h>

struct s {
    char c;
    int i;
    char buffer[13];
};

void compare(const struct s *left, const struct s *right) {
    if ((left && right) &&
        (left->c == right->c) &&
        (left->i == right->i) &&
        (0 == memcmp(left->buffer, right->buffer, 13))) {
        /* ... */
    }
}
```

Compliant solutions in a blue frame that conform with all rules and can be reused in your code

Compliant Code
Fixes noncompliant code.

Experiment

Ask ChatGPT to identify errors in 238 examples of noncompliant code from CERT C Secure Coding Standard

- ChatGPT 3.5 as of March 27, 2023
- Each trial run as a new conversation
- No trial repeated
- All examples (with solutions) available on the web during ChatGPT's training data capture
- Some examples have comments suggesting error
- ChatGPT's performance evaluated by SME

Finding and Fixing an Error

(FIO47-C:Use valid format strings)

Example of noncompliant code:

```
#include <stdio.h>

void func(void) {
    const char *error_msg = "Resource not available to
user.";

    int error_type = 3;

    /* ... */

    printf("Error (type %s): %d\n", error_type, error_msg);

    /* ... */
}
```

ChatGPT:

The format specifier in the printf statement is incorrect. %s is used for printing strings, but error_type is an integer. To print an integer, you should use %d instead. Similarly, error_msg is a string, so you should use %s to print it.

Deep analysis of behavior – finding problem

(CON35-C. Avoid deadlock by locking in a predefined order)

Noncompliant code:

```
#include <stdio.h>
#include <pthread.h>

typedef struct {
    int balance;
    mutex_t balance_mutex;
} bank_account;

typedef struct {
    bank_account *from;
    bank_account *to;
    int amount;
} transaction;

void create_bank_account(bank_account *ba,
                        int initial_amount) {
    bank_account *ba = (bank_account *)malloc(
        sizeof(bank_account));
    if (!ba) return;
    if (!pthread_mutex_init(&ba->balance_mutex,
                           NULL)) {
        /* handle error */
    }
    ba->balance = initial_amount;
    if (!pthread_mutex_lock(&ba->balance_mutex)) {
        /* handle error */
    }
    *ba = 0;
}

int deposit(void *ptr) {
    transaction *tr = (transaction *)ptr;
    if (!pthread_mutex_lock(&tr->from->balance_mutex)) {
        /* handle error */
    }
    /* Not enough balance to transfer */
    if (tr->from->balance < tr->amount) {
        if (!pthread_mutex_unlock(&tr->from->balance_mutex)) {
            /* handle error */
        }
        return -1; /* Deposit error */
    }
    if (!pthread_mutex_lock(&tr->to->balance_mutex)) {
        /* handle error */
    }
    tr->from->balance -= tr->amount;
    tr->to->balance += tr->amount;
}
```

```
if (!pthread_mutex_lock(&tr->from->balance_mutex)) {
    /* handle error */
}

if (!pthread_mutex_lock(&tr->to->balance_mutex)) {
    /* handle error */
}

free(ptr);
return 0;
}

int main(void) {
    pthread_t thr1, thr2;
    transaction *tr1;
    transaction *tr2;
    bank_account *ba1;
    bank_account *ba2;

    create_bank_account(&ba1, 1000);
    create_bank_account(&ba2, 1000);

    tr1 = (transaction *)malloc(sizeof(transaction));
    if (!tr1) return -1;
    tr2 = (transaction *)malloc(sizeof(transaction));
    if (!tr2) return -1;
    tr1->from = &ba1;
    tr1->to = &ba2;
    tr1->amount = 100;

    tr2->from = &ba2;
    tr2->to = &ba1;
    tr2->amount = 100;

    /* Before the deposits */
    if (!pthread_create(&thr1, NULL, deposit, (void *)tr1)) {
        /* handle error */
    }
    if (!pthread_create(&thr2, NULL, deposit, (void *)tr2)) {
        /* handle error */
    }
    return 0;
}
```

ChatGPT:

The code seems to have a potential race condition. Since the two transactions are executed concurrently, it is possible that both threads try to acquire locks on the same two bank accounts in a different order, causing a deadlock.

To avoid this issue, the program should ensure that the locks are acquired in a fixed order. For example, all threads could be required to first acquire a lock on the bank account with the lower address and then on the bank account with the higher address.

Using knowledge beyond knowledge of programming

(ARR38-C Guarantee that Library functions do not form invalid pointers)

ChatGPT:

The code appears to be a portion of the OpenSSL library's `dtls1_process_heartbeat` function that handles the Heartbeat Extension. The Heartbeat Extension has a vulnerability known as the "Heartbleed Bug."

There is a potential buffer over-read issue in the `memcpy` function, which could lead to an information leak.

Not identifying behavior as error - examples

DCL38-C Use the correct syntax when declaring a flexible array member

ChatGPT: There is nothing technically wrong with this code

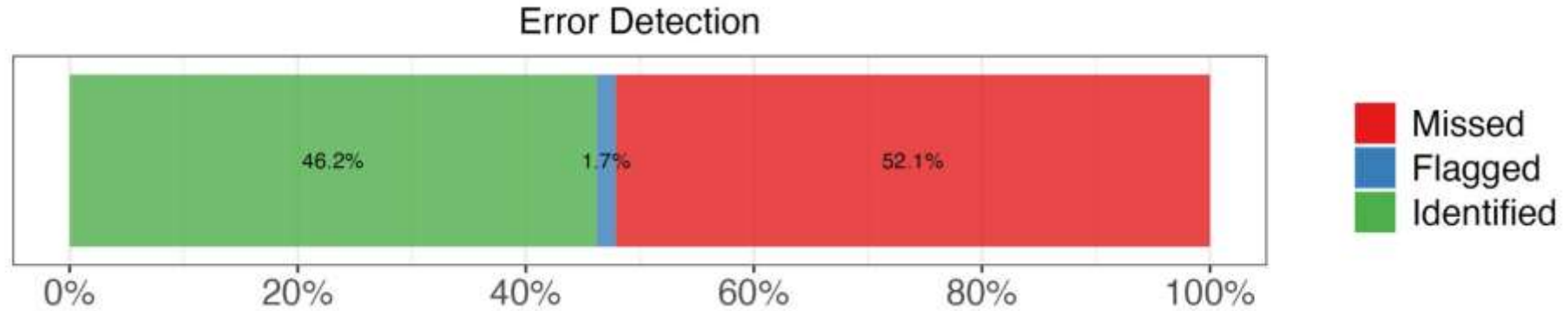
DCL39-C Avoid Information leakage when passing a structure across a trust boundary

ChatGPT: The code appears to be intended to copy a struct test to user space, but there is nothing obviously wrong with it.

EXP33-C Do not read uninitialized memory

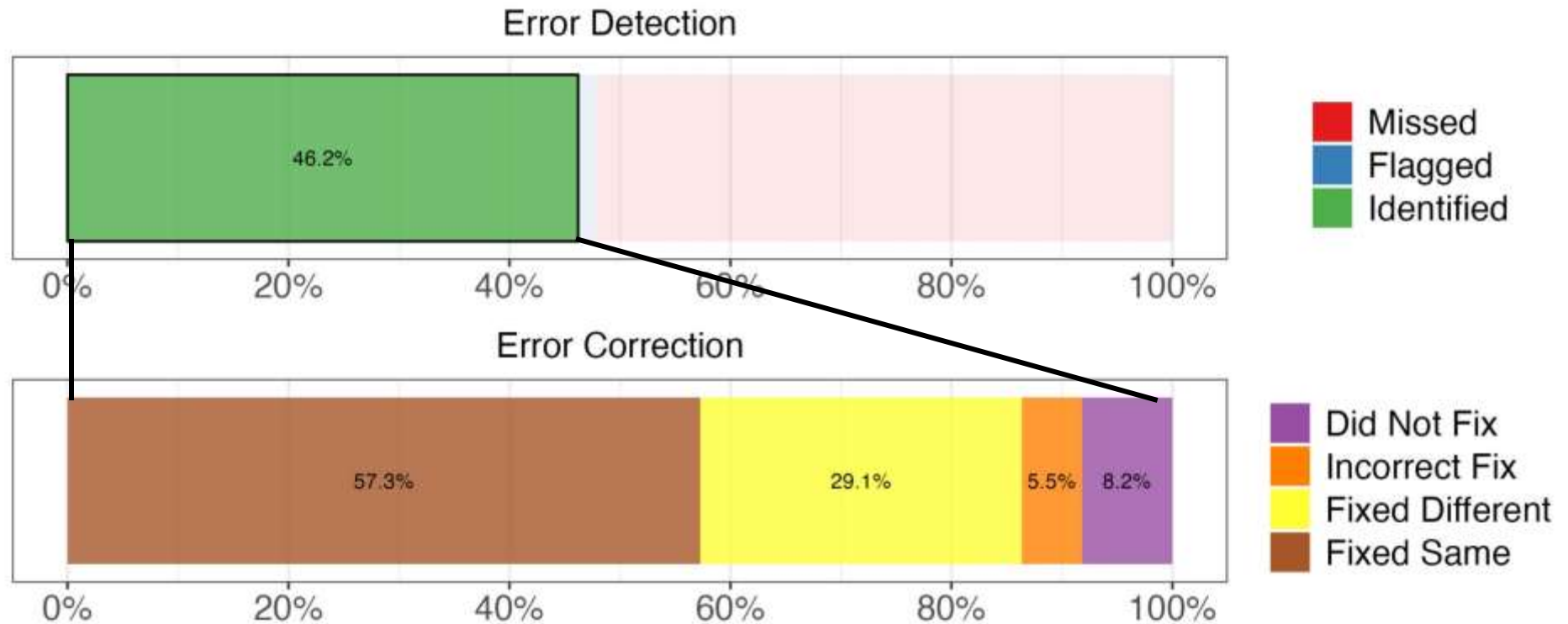
ChatGPT: The code appears to be incomplete as it doesn't have any obvious errors.

Overall Results – Finding Errors



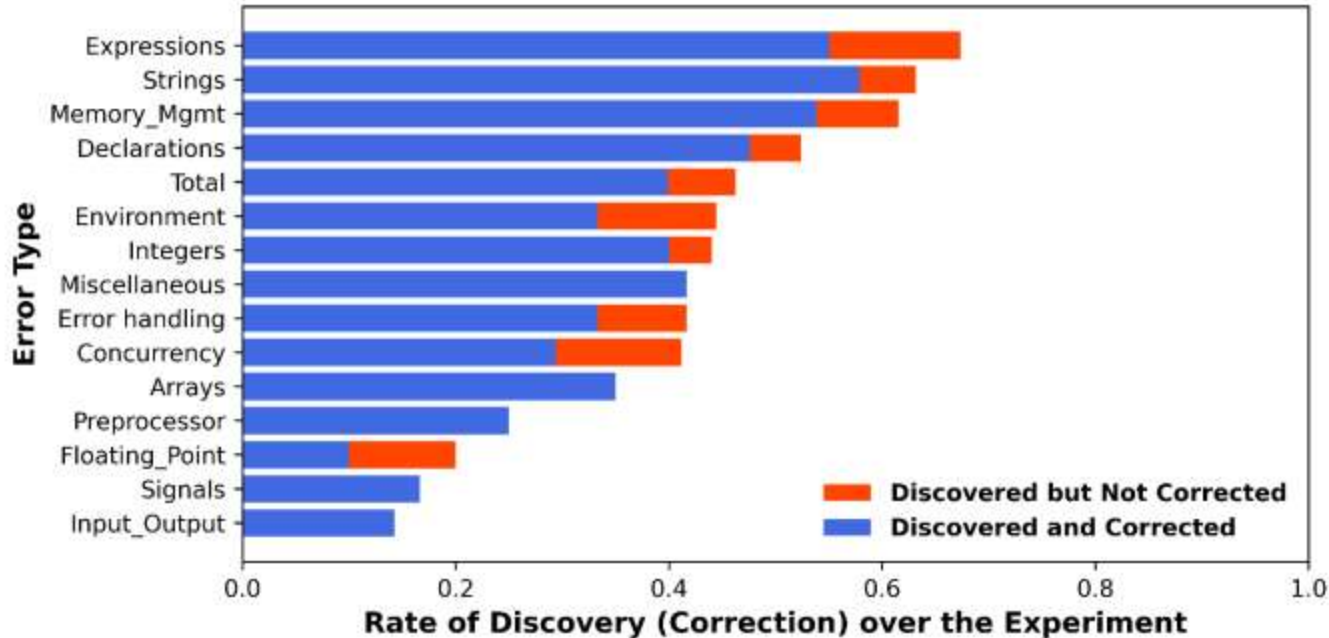
Note: Underlying data in backup

Overall Results – Fixing Errors



Note: Underlying data in backup

Overall Results – By Feature Tested



Seems particularly good at:

- Fixing Integers
- Finding/Fixing Expressions
- Finding/Fixing Memory Management
- Finding/Fixing Strings

Seems particularly challenged by:

- Finding Floating point
- Finding Input/Output
- Finding Signals

It's not that ChatGPT did not find all the errors



"He's not all that smart. I beat him most of the time."

CartoonStock.com

It's that it found as many as it did.

Summary

- Experiment illustrates promise but also limitations
- Like many applications of LLMs, knowledgeable users must review output
- Unfortunately, programmers are not very good at reading and evaluating code
- ChatGPT certainly should not be used to replace programmers, and any other application should be monitored with great care

Acknowledgements



Robert Schiela
Secure Coding Technical Manager

Telephone: +1 412.268.3965
rschiela@cert.org



David Svoboda
Software Security Engineer

Telephone: +1 412.268.3965
Svoboda@cert.org



Matthew Walsh
Senior Data Scientist

Telephone: +1 412.268.9121
mwalsh@cert.org



Clarence Worrell
Senior Data Scientist

Telephone: +1 412.268.9059
cworrell@cert.org



Jamie Glenn
Operations Coordinator

Telephone: +1 412.268.3346
jglenn@cert.org

I want to thank the following members of the Cybersecurity Foundations group at CERT for their expert assistance with evaluating and preparing these materials:

Jamie Glenn	Experiment management
Robert Schiela	C SME
David Svoboda	C SME (Representative, ISO/IEC JTC1/SC22/WG14 (C Programming Language) and lead of Undefined Behaviors Study Group (UBSG))
Matt Walsh	Data analysis and visualization
Clarence Worrell	Data analysis and visualization