Should I Trust ChatGPT to Review My Program?
© 2023 Carnegie Mellon University

Mark Sherman
Technical Director
Carnegie Mellon University/Software Engineering Institute

Disney’s Coronado Springs Resort | Lake Buena Vista, Florida
Should I Trust ChatGPT to Review My Program?

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.


DM23-0888
Should I Trust ChatGPT to Review My Program?

SEPTEMBER 26, 2023

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

© 2023 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.
Agenda

• Introduction
• Background
• Experiment
• Results
• Conclusions
According to Capers Jones’s research, ... “average” code has 6,000 defects per million lines of code. .... [SEI] Research has shown that an estimated 5 percent of these defects can become vulnerabilities.

https://insights.sei.cmu.edu/blog/a-cybersecurity-engineering-strategy-for-devsecops-that-integrates-with-the-software-supply-chain/

That’s 3 vulnerabilities per 10,000 lines of code

Can ChatGPT help?
Most Vulnerabilities Are Caused by Programming Errors

64% of the vulnerabilities in the NIST National Vulnerability Database due to programming errors
• 51% of those were due to classic errors like buffer overflows, cross-site scripting, injection flaws

Top vulnerabilities include
• Integer overflow
• Buffer overflow
• Missing authentication
• Missing or incorrect authorization
• Reliance on untrusted inputs (aka tainted inputs)

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.

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

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

How to use ChatGPT to write code

Greetings, developer! 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

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. It’s honest, it’s pretty cool to see and I’m not mad about it.

https://www.readyselcloud.io/blog/allen.helton/chatgpt-changed-how-i-write-software/
Strong predictions are being made

ChatGPT Writes Code: Will It Replace Software Developers?

Conclusion

ChatGPT is undoubtedly a revolutionary AI tool that will bring huge benefits for software developers. From speedy code generation to simplifying redundant programming tasks, ChatGPT is the solution many in the development community have been looking for.

https://semaphoreci.com/blog/chatgpt-software-developers
• Leading-edge research global university turning disruptive ideas into successes

• 2022-2023 *U.S. News and World Report* rankings:
  - #1 in artificial intelligence, computer engineering, cybersecurity, management information systems, mobile/web applications, programming languages, software engineering, and quantitative analysis
  - #1 in overall computer science

• Creating inspired and inventive solutions through sponsored research, faculty and student engagement, executive education, licensing and tech transfer, start ups, and colocation
CMU Software Engineering Institute (SEI)
Bringing innovation to the U.S. Government

- Federally Funded Research and Development Center (FFRDC) chartered in 1984 and sponsored by the DoD
- Leader in researching complex software engineering, cyber security, and artificial intelligence (AI) engineering solutions
- Critical to the U.S. Government’s ability to acquire, develop, operate, and sustain software systems that are innovative, affordable, trustworthy, and enduring
Deep history of code analysis

**Goal:** Reduce number of code vulnerabilities before code gets to operational environments

- Secure design patterns
- University courses
  - CMU
  - Stevens Institute
  - Purdue
  - University of Florida
  - Santa Clara University
  - St. John Fisher College
- SEI Secure Coding courses
- ISO/IEC TS 17961 C Secure Coding Rules
- ISO/IEC TS 17961 C Secure Coding Rules
- Online course & Professional Certificate
- Instructor-led on-site course
- Analyzer conformance test
- Influence international standards bodies
- Adoption by analyzer tools:
  - LDRA
  - Klocwork
  - Interactive Development Environments
  - Compiler-enforced Buffer Overflow Elimination
  - C and C++ Thread Safety Analysis
  - DidFail Android Flow analysis
- Adoption by software developers and acquirers:
  - Cisco
  - Oracle
  - Various defense contractors
- Adoption by software developers and acquirers:
  - Various defense contractors

Goal: Reduce number of code vulnerabilities before code gets to operational environments
Today's Extension

**Goal:** Reduce number of code vulnerabilities before code gets to operational environments

University courses
- CMU
- Stevens Institute
- Purdue
- University of Florida
- Santa Clara University
- St. John Fisher College

Secure design patterns

ISO/IEC TS 17961 C Secure Coding Rules

Influence international standards bodies

SEI Secure Coding courses

Used as input to MISRA C

Instructor-led on-site course

Analyzer conformance test

Online course & Professional Certificate

Adoption by analyzer tools:
- LDRA
- Klocwork

Adoption by software developers and acquirers:
- Cisco
- Oracle
- Various defense contractors

Adoption by software developers and acquirers:
- Interactive Development Environments
- Compiler-enforced Buffer Overflow Elimination
- C and C++ Thread Safety Analysis
- DidFail Android flow analysis

Applying AI to Code Analysis
- Watson
- SCAIFE
- ChatGPT

SCALe conformance testing

2003
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
Buffer Overflow has Many Causes

Rule Organization – Title & Definition

DCL22-CPP. Functions declared with [[noreturn]] must return void

As described in MSC55-CPP. Do not return from a function declared [[noreturn]], functions declared with the [[noreturn]] attribute must not return on any code path. If a function declared with the [[noreturn]] attribute has a non-void return value, it implies that the function returns a value to the caller even though it would result in undefined behavior. Therefore, functions declared with [[noreturn]] must also be declared as returning void.

Concise but not necessarily precise title

Precise definition of the rule
Rule Organization – Noncompliant Code Example & Compliant Solution

Noncompliant Code Example

In this noncompliant code example, the function declared with \[\text{[noreturn]}\] claims to return an int:

```c
#include <cstdlib>

[noreturn] int f() {
    std::exit(0);
    return 0;
}
```

This example does not violate MSC55-CPP: Do not return from a function declared [noreturn] because std::exit() is declared [noreturn], so the return 0; statement can never be executed.

Compliant Solution

Because the function is declared [noreturn], and no code paths in the function allow for a return in order to comply with MSC55-CPP: Do not return from a function declared [noreturn], the compliant solution declares the function as returning void and elides the explicit return statement:

```c
#include <cstdlib>

[noreturn] void f() {
    std::exit(0);
}
```

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

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

Noncompliant Code Example

Mismatches between arguments and conversion specifications may result in undefined behavior. Compilers may diagnose type mismatches in formatted output function invocations. In this noncompliant code example, the `error_type` argument to `printf()` is incorrectly matched with the `s` specifier rather than with the `d` specifier. Likewise, the `error_msg` argument is incorrectly matched with the `d` specifier instead of the `s` specifier. These usages result in undefined behavior. One possible result of this invocation is that `printf()` will interpret the `error_type` argument as a pointer and try to read a string from the address that `error_type` contains, possibly resulting in an access violation.

```c
#include <stdio.h>

void func(void) {
    const char *error_msg = "Resource not available to user."
    int error_type = 3;
    /* ... */
    printf("Error (type %s): %s\n", error_type, error_msg);
    /* ... */
}
```

Compliant Solution

This compliant solution ensures that the arguments to the `printf()` function match their respective conversion specifications:

```c
#include <stdio.h>

void func(void) {
    const char *error_msg = "Resource not available to user."
    int error_type = 3;
    /* ... */
    printf("Error (type %d): %s\n", error_type, error_msg);
    /* ... */
}
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:

```c
#include <stdio.h>
#include <iostream.h>

int main()
{
    int balance1, balance2;
    int bank_account[2];

    // Code snippet...
}
```

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.
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.
Sometimes misses the obvious
(ENV31-C. Do not rely on an environment pointer following an operation that may invalidate it)

Noncompliant Code:

This noncompliant code example accesses the `envp` pointer after calling `_putenv_s()`:

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, const char *argv[], const char *envp[]) {
    if (_putenv_s("MY_NEW_VAR", "new value") != 0) {
        /* Handle error */
    }
    if (envp != NULL) {
        for (size_t i = 0; envp[i] != NULL; ++i) {
            puts(envp[i]);
        }
    }
    return 0;
}
```

Because `envp` no longer points to the current environment, this program has unanticipated behavior.

ChatGPT:

the code is missing an `#include` directive for the `_putenv_s` function, which is defined in the `stdlib.h` header file. Therefore, you should add `#include <stdlib.h>` at the top of the file.
Finding trivial error and not finding substantiative error
(FIO46-C Do not access a closed file)

Noncompliant Code Example
In this noncompliant code example, the `stdout` stream is used after it is closed:

```c
#include <stdio.h>

int close_stdout(void) {
    if (fclose(stdout) == EOF) {
        return -1;
    }

    printf("stdout successfully closed.\n");
    return 0;
}
```

Substantiative error

ChatGPT:
The code appears to be missing a main function, which is required for all C programs. Without a main function, the code cannot be compiled and executed.
Identifies a non-existent problem and misses real problem

(ENV33-C. Do not call system())

**Noncompliant example:**

In this noncompliant code example, the `system()` function is used to execute `any_cmd` in the host environment.

```c
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

enum { BUFFERSIZE = 512 };

void func(const char *input) {
  char cmdbuf[BUFFERSIZE];
  int len_wanted = snprintf(cmdbuf, BUFFERSIZE, "any_cmd '%s'", input);
  if (len_wanted >= BUFFERSIZE) {
    /* Handle error */
  } else if (len_wanted < 0) {
    /* Handle error */
  } else if (system(cmdbuf) == -1) {
    /* Handle error */
  }
}
```

**ChatPGT:**

It's recommended to use a function that is specifically designed to handle string concatenation and avoid buffer overflows, such as `snprintf()`.  

*Note: `snprintf()` does not overflow*
Identifies a non-existent problem and misses real problem (ERR33-C. Detect and handle standard library errors)

Noncompliant example:

In this noncompliant code example, the function utf8_to_wcs() attempts to convert a sequence of UTF-8 characters to wide characters. It first invokes setlocale() to set the global locale to the implementation-defined en_US.UTF-8 but does not check for failure. The setlocale() function will fail by returning a null pointer, for example, when the locale is not installed. The function may fail for other reasons as well, such as the lack of resources. Depending on the sequence of characters pointed to by utf8, the subsequent call to mbstowcs() may fail or result in the function storing an unexpected sequence of wide characters in the supplied buffer wcs.

```c
#include <locale.h>
#include <stdlib.h>

int utf8_to_wcs(wchar_t *wcs, size_t n, const char *utf8, 
    size_t *size) {
    if (NULL == size) {
        return -1;
    }
    setlocale(LC_CTYPE, "en_US.UTF-8");
    *size = mbstowcs(wcs, utf8, n);
    return 0;
}
```

ChatGPT

The function signature is missing the wchar.h header file, which defines the wchar_t type. This should be added at the top of the file: ✏️ wchar_t does not require wchar.h
Overall Results – Finding Errors

Error Detection

- Missed
- Flagged
- Identified

Note: Underlying data in backup
Overall Results – Fixing Errors

Error Detection

- Missed: 46.2%
- Identified: 53.8%

Error Correction

- Did Not Fix: 57.3%
- Incorrect Fix: 29.1%
- Fixed Different: 5.5%
- Fixed Same: 8.2%

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

Note: Underlying data in backup
Overall Results - Miscellaneous

Fixed correctly but not in response to error 2
Misidentified other problem 17
Identified other problem* 7

Secure Coding course final exam (multiple choice) 19/40 (48%)

*Usually trivial based on code fragment, e.g., missing “main”
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
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
"He’s not all that smart. I beat him most of the time."

CartoonStock.com
### Overall Results – Raw Data

<table>
<thead>
<tr>
<th>Feature</th>
<th>Preprocessor</th>
<th>Declarations</th>
<th>Expressions</th>
<th>Integers</th>
<th>Floating Point</th>
<th>Arrays</th>
<th>Strings</th>
<th>Memory Mgmt</th>
<th>Input Output</th>
<th>Environment</th>
<th>Signals</th>
<th>Error handling</th>
<th>Concurrency</th>
<th>Miscellaneou</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tests</td>
<td>4</td>
<td>21</td>
<td>49</td>
<td>25</td>
<td>10</td>
<td>20</td>
<td>19</td>
<td>13</td>
<td>21</td>
<td>9</td>
<td>6</td>
<td>12</td>
<td>17</td>
<td>12</td>
<td>238</td>
</tr>
<tr>
<td>Found error</td>
<td>1</td>
<td>11</td>
<td>33</td>
<td>11</td>
<td>2</td>
<td>7</td>
<td>12</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>110</td>
</tr>
<tr>
<td>% found</td>
<td>25%</td>
<td>52%</td>
<td>67%</td>
<td>44%</td>
<td>20%</td>
<td>35%</td>
<td>63%</td>
<td>62%</td>
<td>14%</td>
<td>44%</td>
<td>17%</td>
<td>42%</td>
<td>51%</td>
<td>52%</td>
<td>46%</td>
</tr>
<tr>
<td>Fixed error</td>
<td>1</td>
<td>10</td>
<td>27</td>
<td>10</td>
<td>1</td>
<td>7</td>
<td>11</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>95</td>
</tr>
<tr>
<td>% fixed of found</td>
<td>100%</td>
<td>91%</td>
<td>82%</td>
<td>91%</td>
<td>50%</td>
<td>100%</td>
<td>92%</td>
<td>88%</td>
<td>100%</td>
<td>75%</td>
<td>100%</td>
<td>80%</td>
<td>71%</td>
<td>100%</td>
<td>86%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Tests</th>
<th>Found Error</th>
<th>Missed Error</th>
<th>Flagged</th>
</tr>
</thead>
<tbody>
<tr>
<td>238</td>
<td>110</td>
<td>124</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Resolutions</th>
<th>Fixed Same Way</th>
<th>Fixed Different Way</th>
<th>Incorrect Fix</th>
<th>Did Not Fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>63</td>
<td>32</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>


[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.