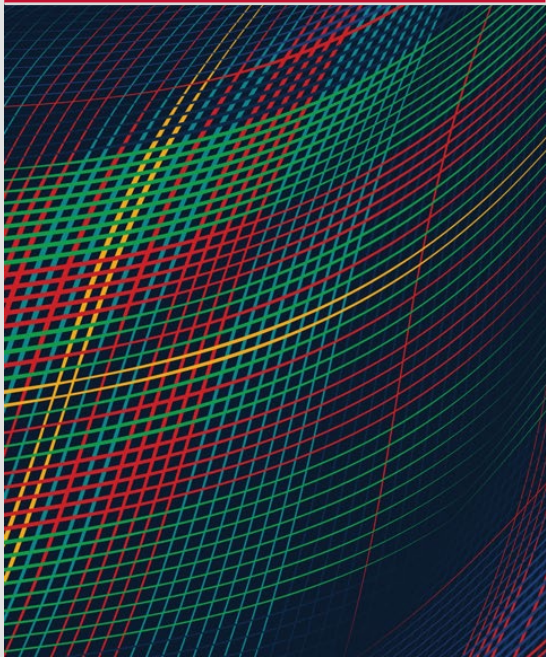
Carnegie Mellon University
Software Engineering Institute



Should I Trust the Next Generation of LLMs to Check My Program?

DAFITC – Department of the Air Force Information Technology and Cyberpower Event

SEPTEMBER 26, 2024

Mark Sherman
Technical Director, Cybersecurity Foundations, CERT
Telephone: +1 412.268.9223
mssherman@sei.cmu.edu

DM24-1002

Agenda

- **Background**
- **Experiment**
- **Results**
- **Summary**

Carnegie Mellon University (CMU)

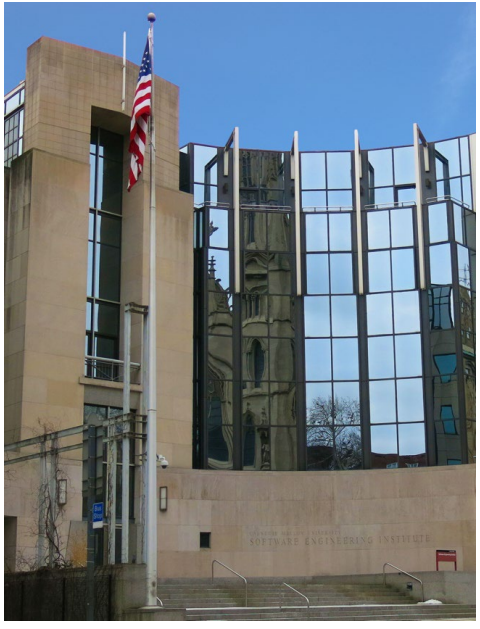
Pioneering discoveries on a global scale

- 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

Carnegie Mellon University Software Engineering Institute



- 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

Lots of Hype about Using ChatGPT for Coding

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

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

How to use ChatGPT to write code

By Amber Israelsen | March 22, 2023

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

How good is ChatGPT at writing code?

"ChatGPT is a revolutionary AI-based system that can help you generate programming solutions quickly and easily"

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

Ai

ChatGPT Changed How I Write Software

By Allen Helton __ 31 May 2023

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

Strong predictions are being made

8 Mar 2023 · Culture

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

Business view of opportunity

> The places that we're seeing tremendous results on [applying Gen AI] are coding. We need 70% less coders from third parties to code as the AI handles most of the coding, the human only needs to look at the final 30% to validate it.
>
> Murray Auchincloss, CEO & Director
> BP Oil
> Earnings Call May 7, 2024

Academics start weighing in

Watch Out, Software Engineers: ChatGPT Is Now Finding, Fixing Bugs in Code

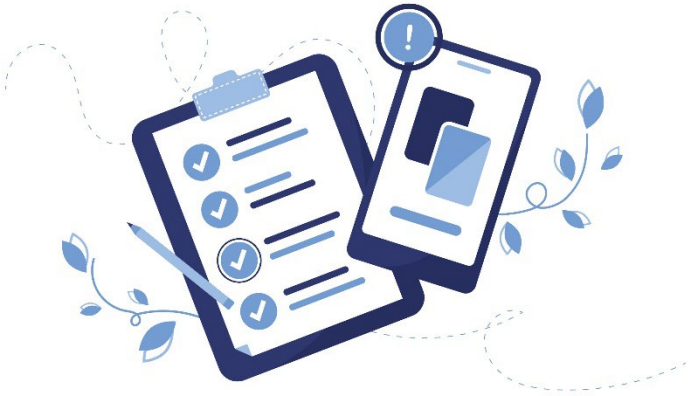
By Emily Dreibelbis
January 27, 2023

Courtesy of computer science researchers from Johannes Gutenberg University and University College London …

Researchers gave 40 pieces of buggy code to four different code-fixing systems … ChatGPT solved 19 problems, Codex solved 21, CoCoNut solved 19, and standard APR [automated program repair] methods figured out seven.

https://www.pcmag.com/news/watch-out-software-engineers-chatgpt-is-now-finding-fixing-bugs-in-code

LLMs still generate a lot of insecure code - Opinion



"The use of AI tools has likely accelerated the pace of software code production and sped up new code deployment. … In reality, AI coding tools continue to consistently generate insecure code."

"AI Code, Security and Trust: Organization Must Change Their Approach," 2023 Snyk AI-Generated Code Security Report, https://go.snyk.io/2023-ai-code-security-report-dwn-typ.html

LLMs still generate a lot of insecure code - Students

"AI code assistants …have been found to produce insecure code in lab environments. … We find that participants who had access to an AI assistant wrote significantly less secure code than those without access to an assistant."

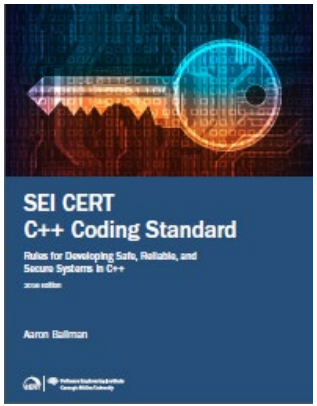
Do Users Write More Insecure Code with AI Assistants
Perrry, Srivastava, Kmar, Boneh, CCS '23, Nov 26-20, 2023,
https://arxiv.org/pdf/2211.03622

LLMs still generate a lot of insecure code – Synthetic Vulnerabilities

"We produce 89 different scenarios for Copilot to complete, producing 1,689 programs. Of these, we found approximately 40% to be vulnerable."

Pearce, Ahmad, Tan, Dolan-Gavin, Larri, Asleep at the Keyboard? Assessing the Security of GitHubb CoPilot's Code Contributions, IEEE Symposium on Security and Privacy, 2022, https://arxiv.org/pdf/2108.09293

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

Example Standard Page (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.

```
#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);
  /* ... */
}
```

Compliant Solution

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

```
#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);

  /* ... */
}
```

Example Standard Page (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.

```
#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);
  /* ... */
}
```

(type %s): %d\n

Compliant Solution

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

```
#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);

  /* ... */
}
```

(type %d): %s\n

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

ChatGPT 3.5 results (last year)

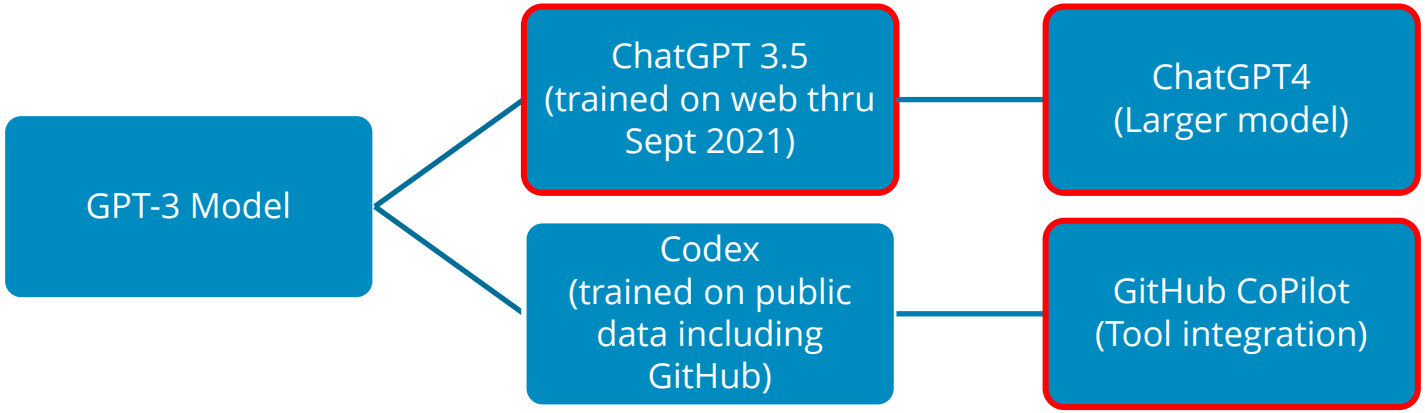
	Found	Missed	Noticed but not a problem
ChatGPT 3.5	110	124	4
	46%	52%	2%

Experiment – How is the state of the practice improving?

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
- ChatGPT 4 and GitHub CoPilot as of February 16, 2024

Systems used in experiments



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.

```
#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);
  /* ... */
}
```

(type %s): %d\n

Compliant Solution

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

```
#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);

  /* ... */
}
```

(type %d): %s\n

Finding and fixing an error

(FIO47-C:Use valid format strings)

Carnegie Mellon University Software Engineering Institute

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);
  /* ... */
}
```

ChatGPT3.5:

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.

Finding and fixing an error – ChatGPT 4.0 & CoPilot

Carnegie Mellon University Software Engineering Institute

ChatGPT 4.0

The code snippet you've shared contains a mistake in the `printf` function call. The format specifiers in the `printf` function are not correctly matched to the types of the variables being printed. Specifically, the format specifier `%s` is used for strings (character arrays), and `%d` is used for integers. However, in your `printf` call, the specifiers are incorrectly matched to the variables `error_type` and `error_msg`.

The correct `printf` statement should be as follows:

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

Here, `%d` correctly corresponds to `error_type` (an integer), and `%s` correctly corresponds to `error_msg` (a string). This change correctly matches the format specifiers to the types of the variables being printed.

CoPilot

The issue with your code is in the printf function call. You are trying to print an integer (error_type) with the %s format specifier, which is for strings, and a string (error_msg) with the %d format specifier, which is for integers.

Here is the corrected code:

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

This will correctly print the error_type as an integer and error_msg as a string.

Deep analysis of behavior – finding problem

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

Carnegie Mellon University Software Engineering Institute

Noncompliant code:

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

typedef struct {
  int balance;
  mtx_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 *nba = (bank_account *)malloc(
    sizeof(bank_account)
  );
  if (nba == NULL) {
    /* Handle error */
  }

  nba->balance = initial_amount;
  if (thrd_success
      != mtx_init(&nba->balance_mutex, mtx_plain)) {
    /* Handle error */
  }

  *ba = nba;
}

int deposit(void *ptr) {
  transaction *args = (transaction *)ptr;

  if (thrd_success != mtx_lock(&args->from->balance_mutex))
    /* Handle error */
  }

  /* Not enough balance to transfer */
  if (args->from->balance < args->amount) {
    if (thrd_success
        != mtx_unlock(&args->from->balance_mutex)) {
      /* Handle error */
    }
    return -1; /* Indicate error */
  }
  if (thrd_success != mtx_lock(&args->to->balance_mutex)) {
    /* Handle error */
  }

  args->from->balance -= args->amount;
  args->to->balance += args->amount;

  if (thrd_success
      != mtx_unlock(&args->from->balance_mutex)) {
    /* Handle error */
  }

  if (thrd_success
      != mtx_unlock(&args->to->balance_mutex)) {
    /* Handle error */
  }

  free(ptr);
  return 0;
}

int main(void) {
  thrd_t thr1, thr2;
  transaction *arg1;
  transaction *arg2;
  bank_account *ba1;
  bank_account *ba2;

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

  arg1 = (transaction *)malloc(sizeof(transaction));
  if (arg1 == NULL) {
    /* Handle error */
  }
  arg2 = (transaction *)malloc(sizeof(transaction));
  if (arg2 == NULL) {
    /* Handle error */
  }
  arg1->from = ba1;
  arg1->to = ba2;
  arg1->amount = 100;

  arg2->from = ba2;
  arg2->to = ba1;
  arg2->amount = 100;

  /* Perform the deposits */
  if (thrd_success
      != thrd_create(&thr1, deposit, (void *)arg1)) {
    /* Handle error */
  }
  if (thrd_success
      != thrd_create(&thr2, deposit, (void *)arg2)) {
    /* Handle error */
  }
  return 0;
}
```

ChatGPT3.5:

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.

But wait – there's more

Trying a second time:

The program you provided seems to be a multi-threaded program that simulates bank account transactions using the C11 thread library (threads.h). While the code appears to be well-structured, there are a few issues and potential improvements:

1. **Error Handling:** There are comments indicating where errors should be handled, but there is no actual error handling code. It's crucial to handle errors properly to avoid crashes or undefined behavior. You should include error handling code in these sections, such as printing an error message and terminating the program gracefully.
2. **Memory Leak:** The program allocates memory for `arg1` and `arg2` using `malloc`, but it doesn't free this memory after the threads have finished executing. You should consider freeing this memory after the threads have completed their transactions to prevent memory leaks.
3. **Thread Joining:** It's a good practice to wait for the threads to finish their execution before exiting the `main` function. You can do this using `thrd_join` to ensure that the main thread doesn't exit before the worker threads complete their transactions.

What happened to the race condition?

But wait – there's still more

Trying a third time:

The code you've provided seems to be an implementation of a simple bank account system using threads and mutexes for synchronization. There are a few potential issues and improvements that could be addressed. ... Here's a specific issue in the code:

Memory Leak: There is a potential memory leak in deposit function where ptr (which points to arg1 or arg2 in main) is freed at the end of deposit. However, the memory allocated for arg1 and arg2 is not freed in main after the threads have finished execution.

What happened to the race condition?

There is even more … trying the “correct” program – December 2023

Carnegie Mellon University Software Engineering Institute

Mutex Deadlock: The program uses mutexes (`mtx_t`) for locking to prevent race conditions during account transactions. However, there is a possibility of a deadlock if two threads try to transfer funds in opposite directions between the same accounts simultaneously. To prevent this, you should ensure that the order in which you lock mutexes is consistent across all threads. One way to achieve this is to assign a unique identifier to each account and lock them in a consistent order based on these identifiers.

The compliant program has the same error as the noncompliant program!

What might be going on?

There is even more … trying the “correct” program a second time – in July 2024

The code you provided appears to implement a simple bank account system with multi-threaded transaction handling.

…

It also correctly orders the locking of mutexes to prevent deadlocks (args->from->id < args->to->id).

Nailed the issue completely.

No error detected.

Deep analysis of behavior – Fixing problem

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

Carnegie Mellon University Software Engineering Institute

ChatGPT4.0

1. ***Deadlock Potential**:** The code has a potential for deadlock because it tries to lock two mutexes (`args->from->balance_mutex` and `args->to->balance_mutex`) without any mechanism to avoid deadlock. If two threads run `deposit` simultaneously, with one transferring from account A to B and the other from B to A, both could end up holding one lock and waiting for the other, causing a deadlock.

```
mtx_t *first_lock, *second_lock;
 if (args->from < args->to) {
   first_lock = &args->from->balance_mutex;
   second_lock = &args->to->balance_mutex;
 } else {
   first_lock = &args->to->balance_mutex;
   second_lock = &args->from->balance_mutex;
 }
```

CoPilot

The main issue with this code is that it can lead to a deadlock situation.

A deadlock can occur when two threads, thr1 and thr2, try to lock the mutexes of ba1 and ba2 at the same time but in the opposite order. If thr1 locks ba1 and thr2 locks ba2, and then thr1 tries to lock ba2 while thr2 tries to lock ba1, both threads will wait for each other indefinitely, causing a deadlock.

To avoid this, you should always lock the mutexes in the same order in all threads. You can do this by sorting the bank accounts by their memory addresses before locking:

```
if (args->from < args->to) {
  if (thrd_success != mtx_lock(&args->from->balance_mutex)) {
    /* Handle error */
  }
  if (thrd_success != mtx_lock(&args->to->balance_mutex)) {
    /* Handle error */
  }
} else {
  if (thrd_success != mtx_lock(&args->to->balance_mutex)) {
    /* Handle error */
  }
  if (thrd_success != mtx_lock(&args->from->balance_mutex)) {
    /* Handle error */
  }
}
```

Determining correction can be difficult – EXP42-C Standard

Noncompliant Example

```
#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)))) {
    /* ... */
  }
}
```

Compliant recommendation

```
#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))) {
    /* ... */
  }
}
```

Determining correction can be difficult – EXP42-C Standard

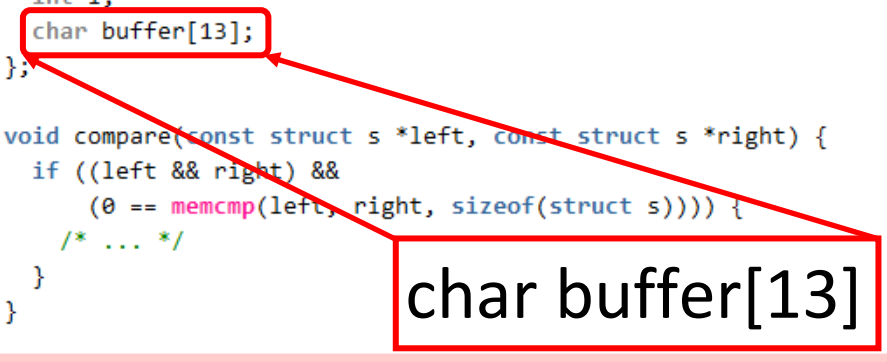
Carnegie Mellon University Software Engineering Institute

Noncompliant Example

```c
#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)))) {
    /* ... */
  }
}
```



Compliant recommendation

```c
#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))) {
    /* ... */
  }
}
```

- Depends on the interpretation of “buffer” as an array of bytes or a string
- CERT Coding Standard assumed array of bytes

Determining correction can be difficult – EXP42-C Standard

Noncompliant Example

```
#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)))) {
    /* ... */
  }
}
```

char buffer[13]

Compliant recommendation

0 == memcmp(left->buffer, right->buffer, 13)

```
  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))) {
    /* ... */
  }
}
```

ChatGPT 3.5's reasoning for EXP42-C

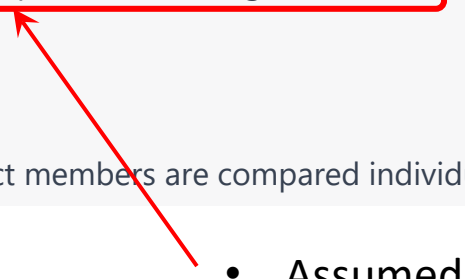
The code appears to be comparing two instances of a struct **s** using **memcmp()**.

However, this may not always produce the expected results, especially if the struct contains padding bytes between its members. The padding bytes may have different values in the two instances of the struct, which would cause **memcmp()** to return a non-zero value even if the actual data members are the same.

To avoid this issue, it is recommended to compare the struct members individually, instead of using **memcmp()**. For example, the code could be modified as follows:

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

Here, the struct members are compared individually using **==** for **char** and **int** members and **strcmp()** for the **buffer** member.



- Assumed "buffer" is a string
- If string, should be using strncmp, not strcmp

ChatGPT 3.5's reasoning for EXP42-C

The code appears to be comparing two instances of a struct **s** using **memcmp()**.

However, this may not always p
members. The padding bytes r
return a non-zero value even if the actual data members are the same.

0 == strcmp(left->buffer, right->buffer)

) to

To avoid this issue, it is recommended to compare the struct members individually, instead of using **memcmp()**. For example, the code could be modified as follows:

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

Here, the struct members are compared individually using **==** for **char** and **int** members and **strcmp()** for the **buffer** member.

- Assumed "buffer" is a string
- If string, should be using strncmp, not strcmp

Adjudication of EXP42-C

Credit given for both identification and fix

Argument for giving ChatGPT credit

- Identified the key issue
- Fix potentially consistent with interpretation of data structure

➢ Recognizing the potential of LLM

Argument for not giving ChatGPT credit

- Should have inferred array was not a string since it has a fixed length
- Should have inferred array was not a string since the identifier is “buffer”
- Picked the wrong string comparison

➢ Cutting and pasting the “fix” would not fix the problem

ChatGPT 4's reasoning for EXP42-C follows implied test assumptions


```
int are_structs_equal(const struct s *left, const struct s *right) {
  if (left == right) {
    // They are the same instance or both are NULL.
    return 1;
  }
  if (!left || !right) {
    // One is NULL and the other is not.
    return 0;
  }
  // Compare each member individually.
  return (left->c == right->c) &&
         (left->i == right->i) &&
         (0 == memcmp(left->buffer, right->buffer, sizeof(left->buffer)));
}
```

"The code snippet you've provided defines a struct `s` and a function `compare` that compares two instances of `s` for equality using `memcmp`. While at first glance, the `compare` function might seem correct, there are a couple of issues related to the use of `memcmp` for comparing structs in C that need to be considered:

1. **Padding bytes**: The C standard allows compilers to insert padding bytes between members of a struct to satisfy alignment requirements. The size of `struct s` includes these padding bytes, but their values are indeterminate. Because `memcmp` compares memory byte by byte, it will also compare these padding bytes. If the padding bytes have different values (which is very possible since they're not explicitly initialized), `memcmp` may report that two structs are different, even if all their actual member variables are identical.

2. **Floating point numbers**: If your struct contained floating point numbers, comparing them with `memcmp` could lead to incorrect results due to the way floating point numbers are represented in memory. This is not directly relevant to your current struct but is something to keep in mind when using `memcmp` for comparison.

3. **Pointers**: If the struct contained pointer members, `memcmp` would only compare the pointer values (addresses) rather than the data they point to. This is not an issue in your current struct but is another consideration when using `memcmp`.

To work around the padding byte issue, you might need to compare each member individually, especially if there's a possibility of padding bytes between members. For the current struct `s`, comparing each member individually would look something like this:"

ChatGPT 4's reasoning for EXP42-C follows implied test assumptions


```
int are_structs_equal(const struct s *left, const struct s *right) {
  if (left == right) {
    // They are the same instance or both are NULL.
    re
  }
  if (!left || !right) {
    // One is NULL and the other is not.
    return 0;
  }
  // Compare each member individually.
  return (left->c == right->c) &&
         (left->i == right->i) &&
         (0 == memcmp(left->buffer, right->buffer, sizeof(left->buffer)));
}
```

0 == memcmp(left->buffer, right->buffer, sizeof(left->buffer))

"The code snippet you've provided defines a struct `s` and a function `compare` that compares two instances of `s` for equality using `memcmp`. While at first glance, the `compare` function might seem correct, there are a couple of issues related to the use of `memcmp` for comparing structs in C that need to be considered:

1. **Padding bytes**: The C standard allows compilers to insert padding bytes

initialized), `memcmp` may report that two structs are different, even if all their actual member variables are identical.

2. **Floating point numbers**: If your struct contained floating point numbers, comparing them with `memcmp` could lead to incorrect results due to the way floating point numbers are represented in memory. This is not directly relevant to your current struct but is something to keep in mind when using `memcmp` for comparison.

3. **Pointers**: If the struct contained pointer members, `memcmp` would only compare the pointer values (addresses) rather than the data they point to. This is not an issue in your current struct but is another consideration when using `memcmp`.

To work around the padding byte issue, you might need to compare each member individually, especially if there's a possibility of padding bytes between members. For the current struct `s`, comparing each member individually would look something like this:"

CoPilot's reasoning for EXP42-C follows implied test assumptions

```
int compare(const struct s *left, const struct s *right) {
  if (left == NULL || right == NULL) {
    return 0; // or handle error as you see fit
  }
  if (left->c != right->c) {
    return 0;
  }
  if (left->i != right->i) {
    return 0;
  }
  if (memcmp(left->buffer, right->buffer, sizeof(left->buffer)) != 0) {
    return 0;
  }
  return 1;
}
```

"The problem with this code is that it uses `memcmp` to compare two structures. This is not safe because the padding bytes (which are not initialized) between the structure members could be different, leading to a false negative result even if the meaningful fields are identical."

CoPilot's reasoning for EXP42-C follows implied test assumptions

```
int compare(const struct s *left, const struct s *right) {
  if (left == NULL || right == NULL) {
    return 0; // or handle error as you see fit
  }
  if (left->c != right->c) {
    retu
  }
  if (left->i != right->i) {
    return 0;
  }
  if (memcmp(left->buffer, right->buffer, sizeof(left->buffer)) != 0) {
    return 0;
  }
  return 1;
}
```

"The problem with this code is that it uses `memcmp` to compare two structures. This is not safe because the padding bytes (which are not members could be different, leading to a false negative result even if the meaningful fields are identical."

memcmp(left->buffer, right->buffer, sizeof(left->buffer)) != 0

Using knowledge beyond knowledge of programming

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

ChatGPT3.5:

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.

Discovering Heartbleed – ChatGPT 4

The code snippet you've provided is a simplified version of the function involved in the OpenSSL Heartbleed bug. The issue with this code lies in the handling of the heartbeat request, specifically in the way it processes the `payload` length without properly validating that the length specified by the request matches the actual amount of data available in the received heartbeat message. This can lead to a buffer over-read.

Here are the main problems with this code:

1. **Lack of Payload Length Validation:** The code does not validate that the `payload` length specified in the heartbeat request is less than or equal to the actual size of the received heartbeat message. This allows an attacker to specify a payload length that is greater than the actual amount of data in the request, causing the server to read beyond the end of the heartbeat message buffer when creating the response. This can result in the server returning memory contents that could contain sensitive information, such as cryptographic keys, passwords, etc.

2. **Buffer Allocation Based on Unverified Input:** The `buffer` is allocated based on the unverified `payload` length received in the heartbeat request (`1 + 2 + payload + padding`). Since the payload length is not checked against the actual size of the data received, this can lead to the aforementioned security vulnerability.

To fix this vulnerability, it's critical to add a check that ensures the `payload` length specified in the heartbeat message does not exceed the size of the message itself. Here's a conceptual fix:

```
if (payload + 1 + 2 > s->s3->rrec.length) {
    /* Payload length is greater than the received data size,
       indicating an attempt to exploit Heartbleed bug. */
    return; // Handle error appropriately
}
```

Includes a fix

This check ensures that the `payload` length, when combined with the sizes for the message type and payload length fields, does not exceed the total length of the received record (`s->s3->rrec.length`). Implementing such a check prevents the buffer over-read and mitigates the Heartbleed vulnerability.

Discovering Heartbleed - CoPilot

The code you've provided is susceptible to the Heartbleed bug, a serious vulnerability in the OpenSSL cryptographic software library. This vulnerability allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet.

The problem lies in the fact that the payload length is trusted and is not checked before the memcpy call. If a malicious client sends a Heartbeat request with a small payload but a large payload length, the server will send back a response that includes data from the server's memory.

Here's the problematic part:

```
n2s(p, payload);
pl = p;

/* ... More code ... */

buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;

/* Enter response type, length, and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
```

Identifies actual code error

The memcpy function copies payload bytes from pl to bp. If payload is larger than the actual data in pl, memcpy will read beyond the allocated memory, causing a buffer over-read.

To fix this, you need to ensure that the payload length does not exceed the actual data length. This can be done by adding a check before the memcpy call.

Not identifying behavior as error – DCL38 - still

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

ChatGPT3.5: There is nothing technically wrong with this code

ChatGPT4: the provided code snippet is an appropriate use of a flexible array member within a struct for dynamic array allocation in C.

CoPilot: The code seems to be correct in terms of syntax and logic.

Not identifying behavior as error – DCL39 - still

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

ChatGPT3.5: The code appears to be intended to copy a struct test to user space, but there is nothing obviously wrong with it.

ChatGPT4: At a glance, the code seems correct in terms of syntax and [the] use …However, there are a few considerations and potential issues … **Potential Padding Issues**:

CoPilot: The code you've provided is a C code snippet. It seems to be mostly correct, but there's one potential issue that could arise due to the structure padding.

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`.

```
#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;
}
```

Could return error making mbstowcs call undefined

ChatGPT3.5

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

Revisiting ERR33 – Better style but no fix

Carnegie Mellon University Software Engineering Institute

ChatGPT4:

Your code snippet for converting a UTF-8 encoded string to a wide character string (`wchar_t`) seems mostly correct, but there are a few considerations and potential issues to be aware of:

(Stylistic suggestions, none fix the problem but all of them could be applied.)

CoPilot:

The function utf8_to_wcs converts a UTF-8 string to a wide character string. However, it doesn't handle the case when mbstowcs returns (size_t)-1, which indicates an error in the conversion process.

(Stylistic use of functions but does not fix the problem.)

Not identifying behavior as error – EXP33 – found

EXP33-C Do not read uninitialized memory

ChatGPT3.5: The code appears to be incomplete as it doesn't have any obvious errors.

ChatGPT4: There are a couple of issues with the provided code … **Uninitialized Variable:**

CoPilot: The issue with this code is that the sign variable in the is_negative function is not initialized.

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()`:

```
#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
}
```

Invalid reuse of envp is problem

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

ChatGPT3.5:

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.

Revisiting ENV-31: sometimes better

ChatGPT4: ****Modification of Environment Variables**:** Modifying environment variables using `_putenv_s` (or any similar function) affects the process's environment. However, the `envp` parameter in `main` reflects the environment variables as they were at the start of your program. Changes made to the environment after the program starts are not reflected in `envp`. This means that the new environment variable set by `_putenv_s` will not appear in the `envp` array. If you want to check the current environment, including changes, you should use `getenv` or similar.

CoPilot: There doesn't seem to be anything wrong with this code

Finding trivial error and not finding substantiative error

(FIO46-C Do not access a closed file)

Carnegie Mellon University Software Engineering Institute

Noncompliant Code Example

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

```
#include <stdio.h>

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

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

Substantiative error

ChatGPT3.5:

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.

Revisiting FIO46 – Found the error

Carnegie Mellon University Software Engineering Institute

ChatGPT4

The issue with the provided code snippet lies in the attempt to use `printf` after `stdout` has been closed using `fclose(stdout)`.

CoPilot

The issue with your code is that you're trying to print to stdout after it has been closed.

Trying a serious security violation

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

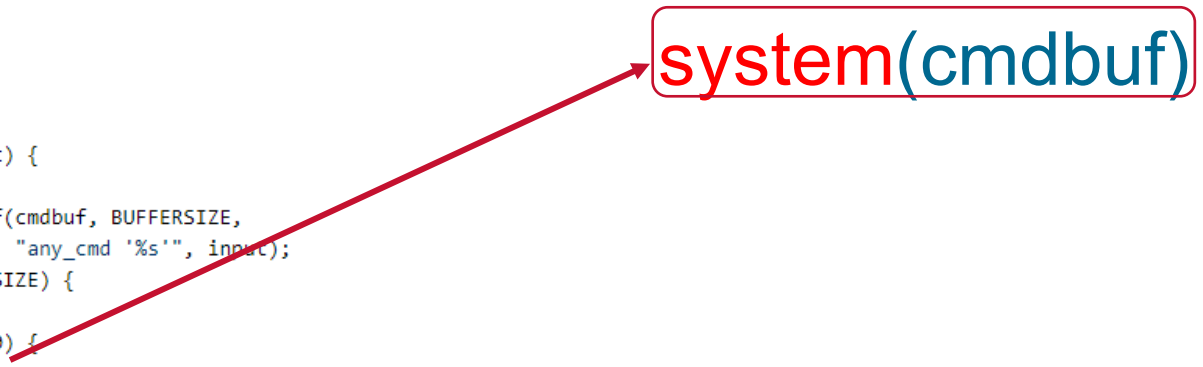
Noncompliant example:

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 */
  }
}
```



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.

```
#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 */
  }
}
```

system(cmdbuf)

Already using snprintf

ChatPGT3.5:

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

Revisiting ENV33 – Better

Carnegie Mellon University Software Engineering Institute

ChatGPT4:

Security Vulnerability (Command Injection): The most critical issue with this code is that it directly incorporates user input (`input`) into a command string that is executed by the system.

Revisiting ENV33 – Better – and surprising guardrails

Carnegie Mellon University Software Engineering Institute

ChatGPT4:

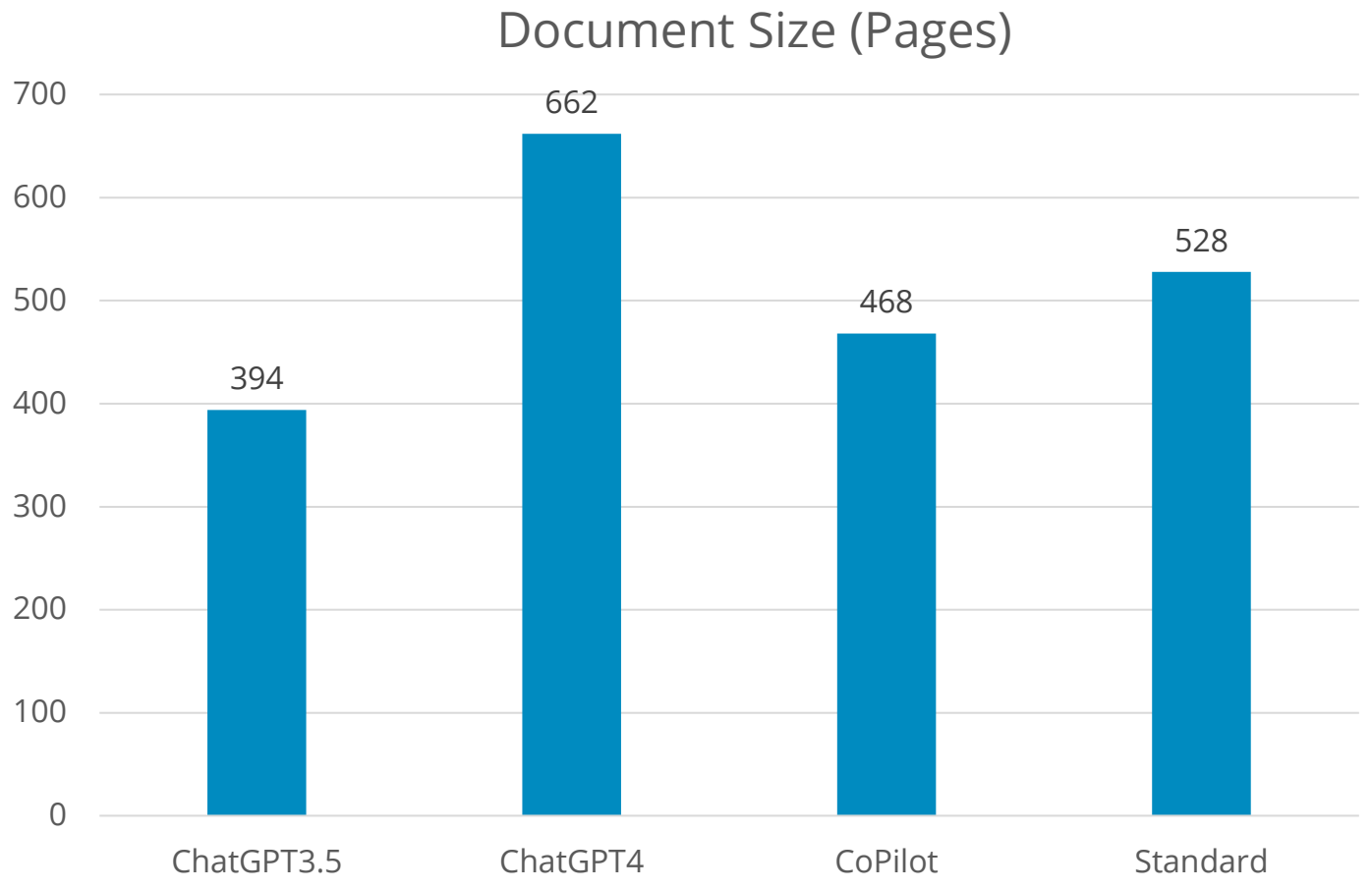
Security Vulnerability (Command Injection): The most critical issue with this code is that it directly incorporates user input (`input`) into a command string that is executed by the system.

CoPilot:

Sorry, the response was filtered by the Responsible AI Service. Please rephrase your prompt and try again. Learn more.

(Tried 3 times.)

Responses more “chatty”



Overall results

	Found**	Missed	% Found*
ChatGPT 3.5	114 (110)	124	48%
ChatGPT 4	187 (175)	51	79%
CoPilot	172 (160)	66	72%

*Combining reports as “noticed but not error” as a “error found”

** Parenthetical values exclude “noticed but not error” reports

*** Not combining “noticed” and “found”

Overall results

	Found**	Missed	% Found*
ChatGPT 3.5	114 (110)	124	48%
ChatGPT 4	187 (175)	51	79%
CoPilot	172 (160)	66	72%

	Both found	Both Missed	Agreed	Improved**	Regressed
ChatGPT 3.5 vs 4	104	41	145	83 (75)	10
ChatGPT 3.5 vs CoPilot	103	55	158	69 (62)	11

*Combining reports as “noticed but not error” as a “error found”

** Parenthetical values exclude “noticed but not error” reports

*** Not combining “noticed” and “found”

Overall results

	Found**	Missed	% Found*
ChatGPT 3.5	114 (110)	124	48%
ChatGPT 4	187 (175)	51	79%
CoPilot	172 (160)	66	72%

	Both found	Both Missed	Agreed	Improved**	Regressed
ChatGPT 3.5 vs 4	104	41	145	83 (75)	10
ChatGPT 3.5 vs CoPilot	103	55	158	69 (62)	11

	ChatGPT 4 and CoPilot Common Changes***	Only ChatGPT 4 Changes***	Only CoPilot Changes***
Improved	54	21	8
Regressed	4	4	8

*Combining reports as “noticed but not error” as a “error found”

** Parenthetical values exclude “noticed but not error” reports

*** Not combining “noticed” and “found”

Workable Model?

The places that we're seeing tremendous results on [applying Gen AI] are coding. We need 70% less coders from third parties to code as the AI handles most of the coding, the human only needs to look at the final 30% to validate it.

Murray Auchincloss, CEO & Director
BP Oil
Earnings Call May 7 2024

One Analyst's Opinion

"If you're not a great coder, you're not going to be able to have the judgement to know if the code that comes out is going to do what you want in the way you want.

Bret Greenstein, Data and AI Leader, PwC,
Can AI coding Tools meet enterprise expectations,
(quoted by Lindsey Wilkinson, Feb 26. 2024,
https://www.ciodive.com/news/generative-ai-coding-tools-reality-versus-hype/708426/

Users defer to perceived authority

> "Participants with access to an AI assistant were also more likely to believe they wrote secure code, suggesting that such tools may lead users to be overconfident about security flaws in their code."

Do Users Write More Insecure Code with AI Assistants

Perrry, Srivastava, Kmar, Boneh, CCS '23, Nov 26-20, 2023, https://arxiv.org/pdf/2211.03622

> "Respondents commonly found security issues with AI suggestions but they also strongly believed that AI suggestions were secure."

"AI Code, Security and Trust: Organization Must Change Their Approach," 2023 Snyk AI-Generated Code Security Report, https://go.snyk.io/2023-ai-code-security-report-dwn-typ.html,

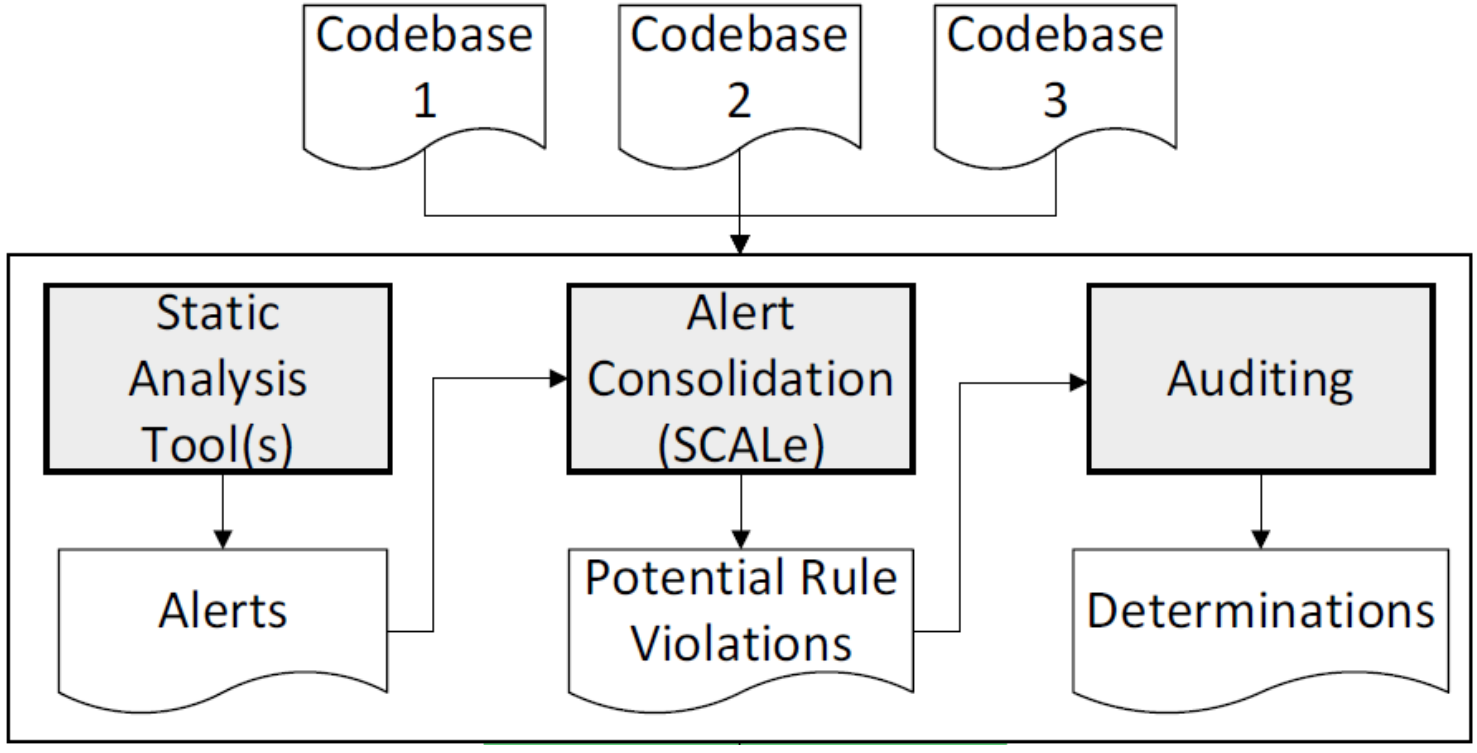
Strategies to effectively use LLMs for code analysis

Multiple evaluators

Generate and test

Prompt Engineering

Multiple evaluators combine results



Flynn, Lori. "Challenges and Progress: Automating Static Analysis Alert Handling with Machine Learning." (2018). https://insights.sei.cmu.edu/documents/4174/2018_017_101_518025.pdf

What Areas Improved?

Common mistakes getting good coverage.

	Total tests	3.5 Found error	4/C Improve-ments over 3.5	4/c + 3.5 Found error	% Improve ment	% Effective ness
PRE	4	1	1	2	100%	50%
DCL	21	11	1	12	9%	57%
EXP	49	33	14	47	42%	96%
INT	25	11	6	17	55%	68%
FLP	10	2	4	6	200%	60%
ARR	20	7	8	15	114%	75%
STR	19	12	4	16	33%	84%
MEM	13	8	4	12	50%	92%
FIO	21	3	6	9	200%	43%
ENV	9	4	1	5	25%	56%
SIG	6	1	0	1	0%	17%
ERR	12	5	4	9	80%	75%
CON	17	7	6	13	86%	76%
MSC	12	5	3	8	60%	67%

Generate and Test – Looking for the correct program

Used HumanEval and HumanEval+ benchmarks

Benchmarks have testing suite to evaluate correct implementation*

LLM	Probability of Generating a Correct Program	
	Generating 1 trial program	Generating 100 trial programs
GPT-4	88.4%	n/a
Phind-CodeLlama	71.3%	96.2%
ChatGPT	73.2%	94.0%
StableLM	2.4%	15.8%

Subset of Table 3, Liu, Zia, Wang, Zhang, Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation, 37th Conference on Neural Information Processing Systems (NeurIPS 2023), https://arxiv.org/pdf/2305.01210

*Authors found 10% of benchmarks were incorrectly programmed

Volume needed to generate correct code

"Existing DL-based [Deep Learning] APR [Automated Program Repair] tools typically have to generate hundreds to thousands of candidate patches and take hours to validate these patches to fix enough bugs. Recent work shows that 93% of developers are only willing to review up to ten patches."

Jiang, Liu, Lutellier, Tan, "Impact of Code Language Models on Automated Program Repair," 2023 IEEE/ACM 45rd International Conference on Software Engineering (ICSE)

Challenges with Generate and Test

- Resources needed to generate programs
- Resources needed to evaluate programs
- Available method to test programs

Apply prompt engineering

Universal law is for lackeys. Context … is for kings.

Captain Lorca
Star Trek: Discovery

Prompting improves results

Watch Out, Software Engineers: ChatGPT Is Now Finding, Fixing Bugs in Code

By Emily Dreibelbis
January 27, 2023

Courtesy of computer science researchers from Johannes Gutenberg University and University College London …

Researchers gave 40 pieces of buggy code to four different code-fixing systems … ChatGPT solved 19 problems, Codex solved 21, CoCoNut solved 19, and standard APR methods figured out seven. …

Providing … hints to ChatGPT, its success rate … increased, fixing 31 out of 40 bugs.

https://www.pcmag.com/news/watch-out-software-engineers-chatgpt-is-now-finding-fixing-bugs-in-code

Giving Guidance Improves Performance

Prompt 0: No guidance
Prompt 1: Generate fix for CWE-xxx
Prompt 2: Include static analyzer alerts
Prompt 3: Add context of error
Prompt 4: Describe repair task

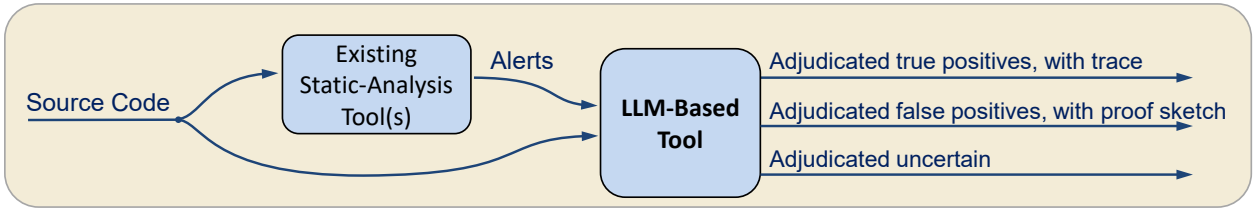
Adapted from Figure 2, Liu, Wang, Zheng, Zhang, "Prompt Fix: Vulnerability Automatic Repair Technology Based on Prompt Engineering," 2024 Workshop on Computing Networking and Communications, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10556123

Prompt Level	% Fixed (low)	% Fixed (high)
0	20	30
1	29	40
2	45	60
3	72	90
4	45	50

Prompt Level	Programs examined	Error Found	% Found
No Prompt	852	646	76%
Extensive Prompt	1015	988	97%

Extracted and adapted from Table 1, Vulnerable C/C++ Programs; Zhang, Liu, Zeng, Yang, K. Li, Y. Li, "Prompt-Enhanced Software Vulnerability Detection Using ChatGPT," 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), https://dl.acm.org/doi/pdf/10.1145/3639478.3643065

Drive LLM to Validate Itself



Use other tools to generate candidate issues

Use prompt engineering to focus on candidate issues

Use LLM to generate patches

Use LLM to generate evidence of patch correctness

Validate patch correctness externally, e.g., SMT solvers (aka theorem prover)

Figure 1. Using LLMs for SA Alert Adjudications. Klieber, Flynn, “Using LLMs for SA alert adjudication and rationales”, CrossTalk, 2024

Summary

- Experiments illustrate 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, technology assistance is necessary and being developed
- Technology is improving, can be applied when managing expectations

Acknowledgements

Robert Schiela
Deputy Technical Director

Telephone: +1 412.268.3965
rschiela@cert.org

David Svoboda
Software Security Engineer

Telephone: +1 412.268.3965
Svoboda@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))

Contact Us

Carnegie Mellon University

Software Engineering Institute

4500 Fifth Avenue

Pittsburgh, PA 15213-2612

412-268-5800

888-201-4479

info@sei.cmu.edu

www.sei.cmu.edu