



**UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

KLEBER REUDO FILGUEIRAS MAIA SOBRINHO

**DETECTANDO FRAQUEZAS USANDO MODELOS DE LINGUAGEM DE
GRANDE PORTE: UMA AVALIAÇÃO COMPARATIVA**

CAMPINA GRANDE - PB

2024

KLEBER REUDO FILGUEIRAS MAIA SOBRINHO

**DETECTANDO FRAQUEZAS USANDO MODELOS DE
LINGUAGEM DE GRANDE PORTE: UMA AVALIAÇÃO
COMPARATIVA**

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador : Rohit Gheyi

CAMPINA GRANDE - PB

2024

KLEBER REUDO FILGUEIRAS MAIA SOBRINHO

**DETECTANDO FRAQUEZAS USANDO MODELOS DE
LINGUAGEM DE GRANDE PORTE: UMA AVALIAÇÃO
COMPARATIVA**

**Trabalho de Conclusão Curso apresentado
ao Curso Bacharelado em Ciência da
Computação do Centro de Engenharia
Elétrica e Informática da Universidade
Federal de Campina Grande, como requisito
parcial para obtenção do título de Bacharel
em Ciência da Computação.**

BANCA EXAMINADORA:

Rohit Gheyi

Orientador – UASC/CEEI/UFCG

Melina Mongiovi Cunha Lima Sabino

Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 15 de Maio de 2024.

CAMPINA GRANDE - PB

RESUMO

A era digital exige software seguro. Fraquezas no código-fonte podem ter consequências graves, desde falhas de software até ataques cibernéticos. Modelos de Linguagem de Grande Porte (LLMs) como ChatGPT 3.5, Gemini 1.0, Claude 3 Sonnet e Mistral Large surgem como ferramentas promissoras para auxiliar na detecção de fraquezas em código-fonte. Este projeto avalia o desempenho de LLMs na detecção de 56 exemplos de fraquezas em código-fonte. Os resultados demonstram que as LLMs podem ser ferramentas importantes para programadores. Na nossa avaliação, as LLMs detectaram 75% das fraquezas de diferentes tipos como: SQL Injection, Cross-site Scripting, Out-of-bounds Write e Null Pointer Dereference. O Claude 3 Sonnet foi a LLM com o melhor resultado.

DETECTING WEAKNESSES USING LARGE LANGUAGE MODELS: A COMPARATIVE EVALUATION

ABSTRACT

The digital era demands secure software. Weaknesses in the source code can have serious consequences, ranging from software failures to cyber attacks. Large Language Models (LLMs) like ChatGPT 3.5, Gemini 1.0, Claude 3 Sonnet, and Mistral Large emerge as promising tools to assist in detecting weaknesses in source code. This project evaluates the performance of LLMs in detecting 56 examples of source code weaknesses. The results demonstrate that LLMs can be important tools for programmers. In our evaluation, the LLMs detected 75% of the weaknesses of different types, such as SQL Injection, Cross-site Scripting, Out-of-bounds Write, and Null Pointer Dereference. Claude 3 Sonnet was the LLM with the best result.

Detectando Fraquezas usando Modelos de Linguagem de Grande Porte: Uma Avaliação Comparativa

Kleber Sobrinho

Universidade Federal de Campina Grande
kleber.sobrinho@ccc.ufcg.edu.br

Rohit Gheyi

Universidade Federal de Campina Grande
rohit@dsc.ufcg.edu.br

RESUMO

A era digital exige software seguro. Fraquezas no código-fonte podem ter consequências graves, desde falhas de software até ataques cibernéticos. Modelos de Linguagem de Grande Porte (LLMs) como ChatGPT 3.5, Gemini 1.0, Claude 3 Sonnet e Mistral Large surgem como ferramentas promissoras para auxiliar na detecção de fraquezas em código-fonte. Este projeto avalia o desempenho de LLMs na detecção de 56 exemplos de fraquezas em código-fonte. Os resultados demonstram que as LLMs podem ser ferramentas importantes para programadores. Na nossa avaliação, as LLMs detectaram 75% das fraquezas de diferentes tipos como: SQL Injection, Cross-site Scripting, Out-of-bounds Write e Null Pointer Dereference. O Claude 3 Sonnet foi a LLM com o melhor resultado.

Keywords

Modelos de Linguagem, ChatGPT 3.5, Gemini 1.0, Claude 3 Sonnet, Mistral Large, Fraquezas, Código-fonte.

1. INTRODUÇÃO

No cenário empresarial contemporâneo, a segurança do software transcende a mera necessidade, assumindo o papel de um imperativo estratégico para a sobrevivência e o sucesso das organizações. Em um mundo cada vez mais digitalizado e interconectado, sistemas e aplicativos permeiam todos os aspectos das operações, tornando-se alvos frequentes de ataques cibernéticos e fraudes.

Defeitos no código-fonte, mesmo os mais simples, podem ter consequências devastadoras. Falhas de segurança podem levar a perdas financeiras significativas, danos irreparáveis à reputação da organização, interrupções na produtividade, exposição de dados confidenciais e, em casos extremos, comprometer a própria viabilidade do negócio.

A confiabilidade de sistemas e aplicativos está intrinsecamente ligada à capacidade de identificar e remediar prontamente as vulnerabilidades presentes no código-fonte. A tarefa de garantir a segurança do software se torna cada vez mais complexa e desafiadora, exigindo soluções inovadoras e ferramentas avançadas que auxiliem os programadores na árdua missão de detectar e corrigir falhas. Com o surgimento de Modelos de Linguagem de Grande Porte (LLMs), surge uma nova fronteira para enfrentar esses desafios.

Nesse contexto, este artigo investiga até que ponto quatro LLMs são capazes de detectar fraquezas em códigos. LLMs são sistemas de inteligência artificial capazes de processar e analisar grandes volumes de dados, aprendendo padrões e relações complexas que

podem ser aplicados à identificação de vulnerabilidades em código.

A avaliação utilizou as LLMs, ChatGPT 3.5, Gemini 1.0, Claude 3 Sonnet e Mistral Large, na detecção de falhas de segurança em código e apresentou resultados positivos e promissores. Este artigo avalia 56 exemplos de fraquezas em C, C++, Java, Python, dentre outras. Na nossa avaliação, as LLMs detectaram 75% das fraquezas de diferentes com destaque para: *SQL Injection*, *OS Command Injection* e *Use of Hard-coded Credentials*. O Claude 3 Sonnet foi a LLM com o melhor resultado.

Esse artigo está organizado da seguinte forma: a Seção 2 apresenta o background, a Seção 3 descreve a avaliação enquanto que a Seção 4 apresenta os resultados. Já a Seção 5 discute os resultados, enquanto que as seções seguintes apresentam os trabalhos relacionados e as conclusões.

2. FUNDAMENTAÇÃO TEÓRICA

Nessa seção apresentamos uma visão geral sobre fraquezas e LLM.

2.1 Fraquezas

O estudo se baseia na *Common Weakness Enumeration (CWE)*, que possui uma lista abrangente de fraquezas de software identificadas pela comunidade. Essa lista abrange todas as etapas da segurança de software, desde a identificação até a prevenção de possíveis pontos fracos. No contexto deste trabalho, uma "fraqueza" se refere a uma condição ou característica do código que pode ser explorada por atacantes para comprometer a segurança do sistema ou causar falhas.

O foco da pesquisa está na avaliação do desempenho das LLMs na detecção das 15 principais fraquezas da CWE, presentes em listagens publicadas entre 2019 e outubro de 2023. Estas fraquezas, apesar de sua visibilidade, são consideradas as mais desafiadoras. Para tal, foram utilizados todos os exemplos disponíveis no site da CWE [2], abrangendo diversas linguagens de programação como C, C++, Java, Python e Go, a fim de contemplar a variedade de cenários em que as vulnerabilidades podem ocorrer. Por exemplo, a *CWE-190¹ Integer Overflow or Wraparound* ocorre quando o produto realiza um cálculo que pode gerar um estouro de inteiro ou envolvimento circular, quando a lógica assume que o valor resultante será sempre maior que o valor original. Isso pode introduzir outras vulnerabilidades quando o cálculo é usado para gestão de recursos ou controle de execução.

¹ <https://cwe.mitre.org/data/definitions/190.html>

2.2 LLMs

Modelos de Linguagem de Grande Porte (LLMs) são sistemas de inteligência artificial treinados em grandes quantidades de dados textuais. Essa enorme quantidade de dados permite que as LLMs aprendam a linguagem natural de forma complexa e sofisticada, possibilitando a realização de diversas tarefas como responder perguntas, resumir documentos, traduzir idiomas e completar frases [5].

As LLMs representam um avanço significativo na área de inteligência artificial, com potencial para revolucionar a forma como interagimos com a tecnologia. As suas capacidades abrem um leque de possibilidades para diversas áreas. São geralmente compostos por redes neurais artificiais complexas, que são treinadas em grandes conjuntos de dados. O processo de treinamento envolve a análise de milhões de palavras e frases, permitindo que a rede neural aprenda a reconhecer padrões e regras da linguagem natural.

3. METODOLOGIA

Nas próximas seções apresentaremos a metodologia usada para a realização do estudo.

3.1 Questões de Pesquisa

Q_M. O objetivo deste estudo é realizar uma avaliação da capacidade de detecção de fraquezas pelas LLMs, com o propósito de identificar suas implicações no cotidiano dos programadores, com respeito à corretude das respostas geradas, no contexto de identificação de fraquezas em código fonte. Para atingir esse objetivo, exploraremos as seguintes Questões de Pesquisa (QP):

QP₁. Até que ponto o ChatGPT 3.5 consegue detectar fraquezas em código-fonte?

Example 3

This code takes an IP address from the user and verifies that it is well formed. It then looks up the hostname and copies it into a buffer.

```
Example Language: C (bad code)
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);

    /*routine that ensures user_supplied_addr is in the right format for conversion */

    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```

This function allocates a buffer of 64 bytes to store the hostname. However, there is no guarantee that the hostname will not be larger than 64 bytes. If an attacker specifies an address which resolves to a very large hostname, then the function may overwrite sensitive data or even relinquish control flow to the attacker.

Note that this example also contains an unchecked return value ([CWE-252](#)) that can lead to a NULL pointer dereference ([CWE-476](#)).

Figura 1. Exemplo de fraqueza selecionado do CWE [4].

Para conduzir esta análise, procedemos da seguinte forma: inicialmente, solicitamos as LLMs que identificassem possíveis fraquezas e apresentassem soluções para os trechos de código a

QP₂. Até que ponto o Gemini 1.0 consegue detectar fraquezas em código-fonte?

QP₃. Até que ponto o Claude 3 Sonnet consegue detectar fraquezas em código-fonte?

QP₄. Até que ponto o Mistral Large consegue detectar fraquezas em código-fonte?

Para cada uma das perguntas, contaremos a quantidade de fraquezas identificadas de acordo com as informações disponíveis no site da CWE.

3.2 Planejamento

Para avaliar a habilidade das LLMs na detecção de fraquezas, selecionamos um conjunto de 15 tipos de fraquezas diretamente da *Common Weakness Enumeration* [3], com um total de 56 exemplos relacionados a essas fraquezas em C, C++, Java, Python, dentre outras linguagens. Os códigos analisados possuem entre 2 a 44 linhas de código. É relevante destacar que essas fraquezas são consistentes em todas as edições publicadas pelo CWE no período entre 2019 e 2023, indicando que representam os desafios mais presentes no contexto atual, apesar de sua notoriedade e visibilidade.

Em seguida, procedemos submetendo todos os exemplos relacionados a essas fraquezas para uma avaliação minuciosa. Um exemplo representativo de fraqueza é ilustrado na Figura 1, que inclui um código identificativo e uma descrição correspondente.

serem submetidos. Nós avaliamos 4 LLMs: ChatGPT 3.5, Gemini 1.0, Claude 3 Sonnet, e Mistral Large.

Posteriormente, copiamos integralmente o código relacionado à fraqueza diretamente do site da CWE e o inserimos no prompt em inglês, conforme ilustrado na Figura 2, em cada LLM para obter uma resposta. Em seguida, analisamos cada resposta das LLMs comparando-a com a descrição da fraqueza em consideração no site da CWE. As consultas foram realizadas entre os meses de Janeiro e Abril de 2024.

```
prompt = f"""
Identify whether or not there is any weakness in the code snippet I will send below,
tell me the name of the weakness and explain it.

Furthermore, if so, please provide a solution to the weakness(es)

Code: ```{code}```
"""
```

Figura 2. Prompt utilizado em cada LLM para a identificação das fraquezas.

4. RESULTADOS

As LLMs apresentaram resultados promissores na detecção de falhas de segurança em código-fonte (ver Tabela 1). O modelo Claude 3 Sonnet se destacou, identificando corretamente as fraquezas em 45 casos, cerca de 80% dos submetidos, seguido pelo Mistral com 44 detecções. O ChatGPT 3.5 obteve um bom desempenho, com 41 acertos, enquanto o Gemini apresentou a menor taxa de sucesso com 39 casos identificados, cerca de 70% dos submetidos. Considerando as 4 LLMs em conjunto, são detectadas 55/56 fraquezas.

LLMs	Fraquezas
ChatGPT 3.5	41/56
Gemini 1.0	39/56
Claude 3 Sonnet	45/56
Mistral Large	44/56

Tabela 1. Quantidade de fraquezas identificadas por LLM.

A Tabela 2 ilustra o desempenho detalhado das LLMs na detecção de diferentes tipos de fraquezas. Observamos que as LLMs apresentaram resultados semelhantes para a maioria dos tipos de CWEs, com algumas exceções notáveis, como o desempenho do Claude 3 Sonnet na detecção de *CWE-190 Integer Overflow or Wraparound* que identificou corretamente os 100% (4) casos submetidos, enquanto as demais LLMs identificaram no máximo 1 caso.

Tipo de Fraqueza	Quant. Fraquezas	ChatGPT 3.5	Gemini 1.0	Claude 3 Sonnet	Mistral Large
CWE-787 Out-of-bounds Write	7	4	3	2	5
CWE-476 NULL Pointer Dereference	4	4	2	2	3
CWE-79 Cross-site Scripting	5	4	5	5	5
CWE-89 SQL Injection	4	4	4	4	4
CWE-78 OS Command Injection	6	6	6	6	6
CWE-416 Use After Free	2	2	1	2	2
CWE-20 Improper Input Validation	5	4	2	2	4
CWE-22 Path Traversal	6	3	5	6	3
CWE-352 Cross-Site Request Forgery (CSRF)	1	1	0	1	0
CWE-287 Improper Authentication	1	1	1	1	1
CWE-502 Deserialization of Untrusted Data	2	1	2	2	2
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer	4	3	3	3	3
CWE-798 Use of Hard-coded Credentials	4	4	4	4	4
CWE-125 Out-of-bounds Read	1	0	1	1	1
CWE-190 Integer Overflow or Wraparound	4	0	0	4	1

Tabela 2. Quantidade de fraquezas identificadas para cada LLM por tipo de fraqueza.

5. DISCUSSÃO

Nessa seção discutiremos os resultados obtidos.

5.1 Detecção das Fraquezas

Em alguns tipos de fraquezas foram identificadas pelas LLMs avaliadas, como, *CWE-79 Cross-site Scripting*, que apresentou alta taxa de detecção por todas as LLMs, com o Gemini 1.0, Mistral Large e Claude 3 Sonnet apresentando uma detecção perfeita (5/5 casos), e o ChatGPT 3.5 (4/5 casos). Já todos os exemplos das *CWE-89 SQL Injection* e *CWE-798 Use of Hard-coded Credentials* foram detectados (4/4 casos) por todas as LLMs. O mesmo ocorreu para a *CWE-78 OS Command Injection*, onde os 6 casos submetidos foram identificados por todas as LLMs.

Entretanto, algumas fraquezas foram de difícil detecção como, *CWE-190 Integer Overflow or Wraparound*, com destaque para o

bom desempenho do Claude 3 Sonnet (4/4 casos) e o desempenho inferior do Gemini 1.0 e ChatGPT 3.5 (0/4 casos), com o Mistral Large obtendo também um desempenho limitado (1/4 casos). Já a *CWE-787 Out-of-bounds Write* apresentou desempenho variável entre as LLMs, com destaque para o ChatGPT 3.5 (4/7 casos) e Mistral Large (5/7 casos), enquanto o Claude 3 Sonnet (2/7 casos) e Gemini (3/7 casos) apresentaram resultados inferiores. Todas as 15 fraquezas não detectadas pelo ChatGPT 3.5 foram executadas usando o mesmo prompt na sua versão mais nova (Chat GPT 4) e foram detectadas.

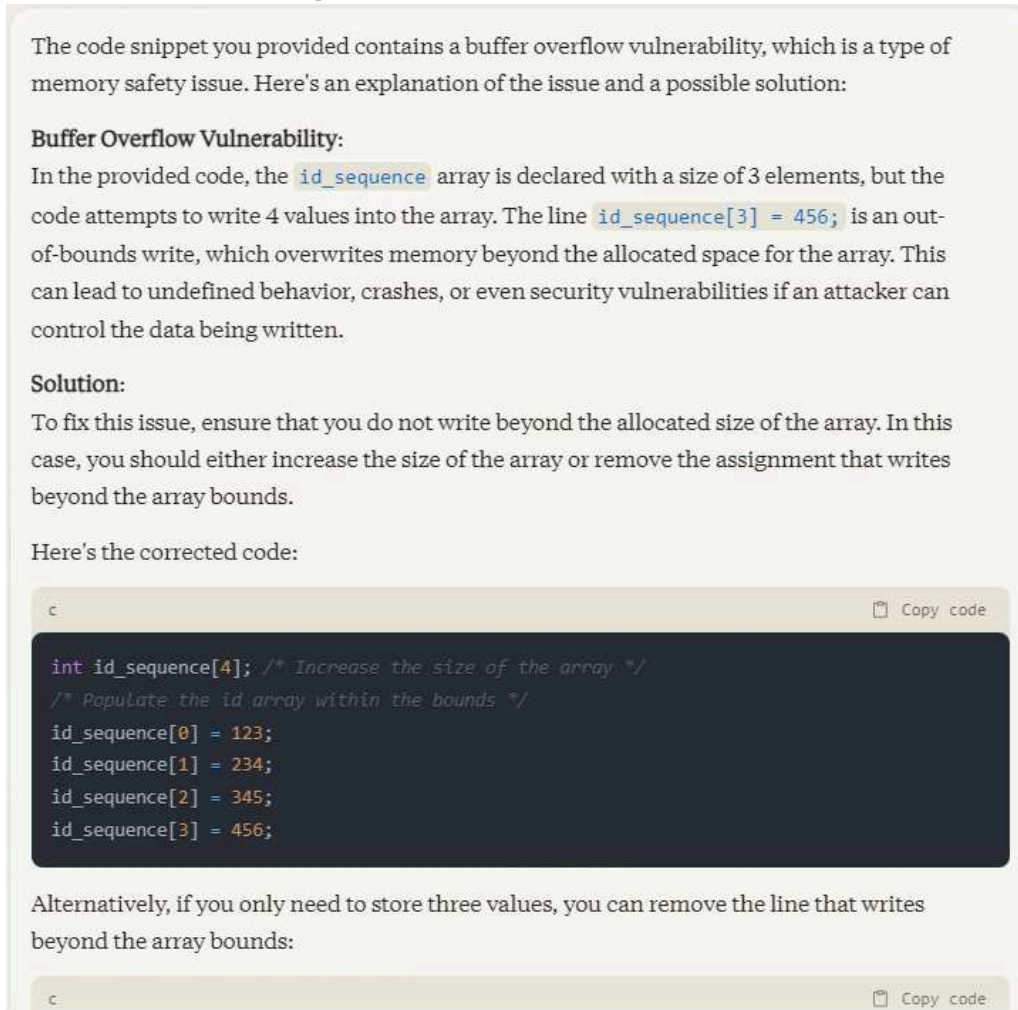
5.2 Explicação

A qualidade das explicações fornecidas pelas LLMs é um aspecto crucial a ser considerado na avaliação de seu potencial para a detecção de fraquezas. Nesse contexto, o Claude 3 Sonnet se destaca por fornecer as explicações mais claras e bem estruturadas entre as LLMs analisadas. A Figura 3 ilustra a detecção da LLM

para o exemplo da Figura 1, correspondente à classe de fraquezas *CWE-787: Out-of-bounds Write*.

Observe como o Claude 3 Sonnet não apenas identifica a fraqueza com precisão, mas também fornece uma explicação detalhada e informativa, que inclui: uma descrição da fraqueza e como ela

ocorre, a linha de código específica onde a fraqueza está presente, impacto da fraqueza, além de orientações sobre como corrigir a fraqueza de forma eficaz.



The code snippet you provided contains a buffer overflow vulnerability, which is a type of memory safety issue. Here's an explanation of the issue and a possible solution:

Buffer Overflow Vulnerability:

In the provided code, the `id_sequence` array is declared with a size of 3 elements, but the code attempts to write 4 values into the array. The line `id_sequence[3] = 456;` is an out-of-bounds write, which overwrites memory beyond the allocated space for the array. This can lead to undefined behavior, crashes, or even security vulnerabilities if an attacker can control the data being written.

Solution:

To fix this issue, ensure that you do not write beyond the allocated size of the array. In this case, you should either increase the size of the array or remove the assignment that writes beyond the array bounds.

Here's the corrected code:

```
c
int id_sequence[4]; /* Increase the size of the array */
/* Populate the id array within the bounds */
id_sequence[0] = 123;
id_sequence[1] = 234;
id_sequence[2] = 345;
id_sequence[3] = 456;
```

Alternatively, if you only need to store three values, you can remove the line that writes beyond the array bounds:

```
c
```

Figura 3. Exemplo da resposta do Claude 3 Sonnet para a detecção da *CWE-787 Out-of-bounds Write*.

Todas as LLMs analisadas apresentaram divergências na terminologia utilizada para nomear as fraquezas em relação ao padrão oficial do CWE. O Claude 3 Sonnet, por exemplo, apresentou essa divergência em 13 dos 45 casos em que identificou corretamente a fraqueza. Já o Mistral Large apresentou divergência em 14 dos 44 casos, o ChatGPT 3.5 em 19 dos 41 casos e o Gemini 1.0 em 15 dos 39 casos. Apesar da divergência na terminologia, as LLMs geralmente descrevem bem a fraqueza em si, mesmo que a categorização no tipo de CWE errado. Por exemplo, as LLMs identificaram uma *CWE Cross-site Scripting* mas a categorizaram como uma *SQL Injection*. Em outro caso, as LLMs apresentam uma nomenclatura mais genérica como a identificação de *Buffer Overflow* tanto para *Out-of-bounds write* quanto para *Out-of-bounds read*. Em alguns poucos casos, as LLMs, com exceção do Claude 3 Sonnet, detectaram a fraqueza e descreveram de forma genérica mas não especificaram um CWE para ela.

5.3 Alucinação

No contexto da geração das LLMs, "alucinações" referem-se à produção de informações falsas, enganosas ou inconsistentes por um modelo de linguagem [6]. Podem ocorrer por diversos fatores, como limitações nos dados de treinamento ou complexidade da linguagem. As alucinações podem ser completamente inventadas ou conter elementos distorcidos da realidade.

Para um caso de fraqueza do tipo *CWE-190: Integer Overflow or Wraparound*, ilustrado na Figura 4, o ChatGPT 3.5 apresenta uma alucinação, que pode ser verificada na Figura 5. O modelo apresenta linhas de código existentes como se fossem novas, ignorando o contexto original.

Example 3

Integer overflows can be complicated and difficult to detect. The following example is an attempt to show how an integer overflow may lead to undefined looping behavior:

```
Example Language: C (bad code)  
short int bytesRec = 0;  
char buf[SOMEBIGNUM];  
  
while(bytesRec < MAXGET) {  
    bytesRec += getFromInput(buf+bytesRec);  
}
```

In the above case, it is entirely possible that bytesRec may overflow, continuously creating a lower number than MAXGET and also overwriting the first MAXGET-1 bytes of buf.

Figura 4. Exemplo de fraqueza do tipo CWE-190: Integer Overflow or Wraparound.

ChatGPT

The provided code appears to be part of a loop that reads data from some input source into a buffer (`buf`). Here are a few considerations:

1. Array Indexing:

Ensure that the array indices do not go out of bounds. The loop should not read more data into `buf` than its size (`SOMEBIGNUM`).

```
c Copy code  
  
while (bytesRec < MAXGET) {  
    bytesRec += getFromInput(buf + bytesRec);  
}
```

2. Initialization:

Make sure that `bytesRec` is properly initialized before entering the loop.

```
c Copy code  
  
short int bytesRec = 0;
```

3. Variable Naming:

Consider using more descriptive variable names. For example, `MAXGET` could be replaced with a more meaningful name based on the context.

4. Input Function (`getFromInput`):

Ensure that the `getFromInput` function returns the correct value and handles errors properly. It should return the number of bytes read or an indication of failure.

```
c Copy code  
  
size_t getFromInput(char *buffer);
```

Also, consider checking the return value to detect errors or the end of input.

```
c Copy code  
  
size_t bytesRead = getFromInput(buf + bytesRec);
```

Figura 5. Alucinação na detecção do ChatGPT 3.5 na detecção da fraqueza do tipo CWE-190.

Conforme apresentado na Tabela 3, o Mistral Large teve o menor número de alucinações (2), enquanto o Gemini 1.0 apresentou o maior (6). O ChatGPT 3.5 e o Claude 3 Sonnet registraram 4 e 5 alucinações, respectivamente.

LLMs	Número de Alucinações
ChatGPT 3.5	4
Gemini 1.0	6
Claude 3 Sonnet	5
Mistral Large	2

Tabela 3. Número de alucinações por LLM.

Embora o estudo tenha apresentado um número relativamente baixo de alucinações pelas LLMs, é crucial considerar os impactos negativos como, redução da confiança dos desenvolvedores nos modelos de linguagem como ferramentas de detecção de fraquezas, convencer os desenvolvedores a corrigir fraquezas inexistentes ou até mesmo ignorar fraquezas reais, comprometendo a segurança do sistema.

5.4 Número de Tentativas

É importante mencionar que as LLMs, devido à sua natureza probabilística, geram respostas variadas para a mesma pergunta, mesmo com o mesmo *prompt* [7]. Essa variabilidade, muitas vezes vista como um desafio, pode ser explorada como uma oportunidade através de tentativas múltiplas. Com o objetivo de identificar fraquezas que podem não ser identificadas na primeira submissão para a LLM, nós submetemos cada exemplo contendo uma fraqueza até três vezes, utilizando um chat exclusivo para cada rodada e o mesmo prompt.

A Tabela 3 apresenta os resultados do estudo. É possível observar uma melhora significativa no Gemini 1.0 (49/56) ao considerar as três rodadas. O modelo ascendeu da última para a segunda posição, empatando com o Mistral Large em quantidade de fraquezas identificadas. Tanto o Gemini 1.0 quanto o Mistral Large se aproximaram do Claude 3 Sonnet, que novamente apresentou o melhor resultado (50/56). O ChatGPT 3.5 obteve o pior desempenho (45/56).

LLMs	Fraquezas identifica das (1ª rodada)	Fraquezas identifica das (2ª rodada)	Fraquezas identificad as (3ª rodada)	Total após 3 tentativas
ChatGPT 3.5	41/56	3/15	1/12	45/56
Gemini 1.0	39/56	8/17	2/9	49/56
Claude 3 Sonnet	45/46	3/11	2/8	50/56
Mistral	44/56	5/12	0/7	49/56

Large				
-------	--	--	--	--

Tabela 4. Quantidade de fraquezas identificadas por rodada para cada LLM.

5.5 Linguagens

A Tabela 5 apresenta a distribuição das fraquezas por linguagem detectadas pelas LLMs. Nota-se que a maioria dos códigos submetidos são em C ou Java, e as LLMs apresentam desempenhos similares na detecção de fraquezas em todas as linguagens, exceto para o ChatGPT 3.5 na linguagem Python, que não identificou nenhuma das 2 fraquezas submetidas, enquanto que as outras LLMs detectam todas (2/2).

Linguagem	LLMs			
	ChatGPT 3.5	Gemini 1.0	Claude 3 Sonnet	Mistral Large
Python	0/2	2/2	2/2	2/2
C	13/22	11/15	15/22	16/22
Java	13/15	11/15	12/15	11/15
Go	1/1	0/1	0/1	0/1
PHP	6/7	6/7	7/7	6/7
JSP	2/2	2/2	2/2	2/2
C#	1/1	1/1	1/1	1/1
SQL	1/1	1/1	1/1	1/1
Perl	4/5	5/5	5/5	5/5

Tabela 5. Quantidade de fraquezas identificadas por linguagem de programação para cada LLM.

5.6 Resolução de Fraquezas

Para todos os casos em que as LLMs detectaram corretamente a fraqueza, elas também propuseram uma solução em potencial, ou seja, soluções que envolviam alterações nas linhas onde a fraqueza estava presente. Devido à natureza fragmentada dos códigos submetidos (geralmente partes de um todo), testar completamente as resoluções propostas pelas LLMs para as fraquezas se mostrou desafiador. Essa atividade será realizada como trabalho futuro.

5.7 Ameaças à Validade

O estudo pressupõe que as LLMs não possuem conhecimentos prévios sobre a CWE e as fraquezas de software catalogadas na plataforma. Isso significa que as LLMs basearam suas detecções e soluções apenas nos dados de treinamento fornecidos, sem qualquer conhecimento prévio ou especializado sobre as vulnerabilidades em questão [19]. Apenas um número limitado de exemplos para cada tipo de fraqueza foi avaliado. É possível que outras fraquezas, não avaliadas no estudo, apresentem diferentes níveis de dificuldade de detecção pelas LLMs. Além disso, o

estudo se concentrou em um conjunto específico de CWEs e poucos exemplos para cada linguagem de programação.

5.8 Respostas às Questões de Pesquisa

A seguir respondemos às questões de pesquisa.

QP₁. Até que ponto o ChatGPT 3.5 consegue detectar fraquezas em código-fonte?

O ChatGPT 3.5 apresentou um bom desempenho na detecção de fraquezas, identificando 41 das 56 fraquezas submetidas. Considerando as 3 rodadas a LLM alcançou a marca de 45 fraquezas corretamente identificadas, tendo o pior desempenho entre as 4 LLMs avaliadas neste trabalho.

QP₂. Até que ponto o Gemini 1.0 consegue detectar fraquezas em código-fonte?

O Gemini 1.0 apresentou um desempenho satisfatório na detecção de fraquezas, identificando 39 das 56 fraquezas submetidas. Considerando as 3 rodadas a LLM alcançou a marca de 49 fraquezas corretamente identificadas.

QP₃. Até que ponto o Claude 3 Sonnet consegue detectar fraquezas em código-fonte?

O Claude 3 Sonnet se destacou entre as LLMs analisadas, obtendo o melhor índice de acerto na detecção de fraquezas, identificando 45 das 56 fraquezas submetidas. Considerando as 3 rodadas a LLM alcançou a marca de 50 fraquezas corretamente identificadas, obtendo o melhor desempenho entre os 4 Modelos de Linguagem considerados neste trabalho.

QP₄. Até que ponto o Mistral Large consegue detectar fraquezas em código-fonte?

O Mistral Large demonstrou capacidade de detecção de fraquezas, identificando 44 das 56 fraquezas submetidas na primeira tentativa. Considerando as 3 rodadas, a LLM alcançou a marca de 49 fraquezas corretamente identificadas.

6. TRABALHOS RELACIONADOS

Souza e Gheyi [9] investigaram a capacidade do ChatGPT 3.5, um chatbot de modelo de linguagem de larga escala, para resolver problemas de programação. Do total de 100 problemas enviados, o modelo de linguagem foi capaz de responder corretamente a 71 problemas em 3 tentativas, com 50 deles da plataforma LeetCode e 21 da plataforma BeeCrowd.

Medeiros et al. [10] propõe uma técnica de caixa branca baseada em amostragem para detectar fraquezas relacionadas à configuração em sistemas configuráveis. A técnica foi avaliada em 24 sistemas populares e altamente configuráveis, identificando 57 fraquezas. Muniz et al. [8] fez um estudo qualitativo de fraquezas em sistemas configuráveis contendo diretivas de pré-processamento com `#ifdefs`, e identificou através de um questionário que os desenvolvedores se preocupam com fraquezas mas não usam as ferramentas adequadas. No nosso trabalho, nós investigamos até que ponto LLMs podem ser úteis para detectar fraquezas em códigos.

Pearce et al. [15] investigaram a capacidade de LLMs, como o Codex da OpenAI e o Jurassic J-1 da AI21, para realizar reparos

de vulnerabilidades "zero-shot" em código (ou seja, sem a necessidade de exemplos prévios). O estudo identifica que as LLMs podem reparar coletivamente 100% das vulnerabilidades geradas sinteticamente e criadas manualmente. No nosso trabalho, focamos na detecção das fraquezas e utilizamos outras LLMs, como o Claude 3 Sonnet e o Mistral Large.

Islam et al. [16] propõem um sistema multifuncional de análise de vulnerabilidade de código (SecRepair) alimentado por um grande modelo de linguagem (CodeGen2) para identificar vulnerabilidades, gerar código corrigido e explicar os problemas em comentários bem elaborados, conseguindo identificar vulnerabilidades *zero-day* e *N-day* em 6 sistemas operacionais IoT (*Internet of Things*) de código aberto no GitHub. No nosso trabalho, utilizamos o site da CWE como fonte para selecionar os códigos para avaliar até que ponto 4 LLMs conseguem detectá-las.

Chan et al. [17] apresentam um sistema que utiliza aprendizado profundo em um grande conjunto de dados para identificar padrões de vulnerabilidade em trechos de código incompletos, ou com sintaxe incorreta. O sistema se baseia em LLMs pré-treinados providas pela OpenAI: `code-davinci-002` e `text-davinci-003` (*zero-shot*, *few-shot* e *fine-tuning*), para detectar cerca de 250 tipos de vulnerabilidades durante a escrita do código. A avaliação em um benchmark de cenários de código de alto risco mostra uma redução de até 90% na taxa de vulnerabilidade. No nosso trabalho, não focamos na identificação de fraquezas em tempo de escrita de código, mas sim na detecção em exemplos de códigos já finalizados usando LLMs.

Tang et al. [18] propõem a CSGVD (*Combining Sequence and Graph embedding for Vulnerability Detection*), que é uma nova abordagem híbrida que combina incorporação de sequência e incorporação de gráficos para capturar informações semânticas e estruturais do código. Os resultados experimentais mostram que o CSGVD obtém 64,46% de precisão no conjunto de dados de benchmark do mundo real do CodeXGLUE para detecção de vulnerabilidades. Em contraste, nosso trabalho utiliza exemplos do CWE para analisar e detectar vulnerabilidades usando 4 tipos de LLMs.

7. CONCLUSÃO

Nesse artigo, avaliamos o potencial das grandes modelos de linguagem na detecção de fraquezas em código-fonte, com foco em sua eficácia, qualidade das explicações, ocorrência de alucinações, performance e capacidade de resolução de problemas. As LLMs demonstraram ser ferramentas promissoras para auxiliar na detecção de fraquezas em código-fonte, com destaque para o Claude 3 Sonnet, que apresentou a maior taxa de acerto geral (80%) na primeira tentativa, e se destacou na detecção de diversos tipos de fraquezas. Todos os dados deste trabalho estão disponíveis online.²

Implicações. A pesquisa demonstra a viabilidade promissora das LLMs na identificação de fraquezas em software. Essa descoberta abre um leque de oportunidades para aprimorar a segurança e confiabilidade do software, especialmente para programadores. LLMs podem auxiliar na identificação de fraquezas em

²

<https://docs.google.com/spreadsheets/d/1LqTJm16LCSR3iyXj1bvvvuFaykGkVGVfXA7bOZx8S4/edit?usp=sharing>

código-fonte, porém é importante reconhecer que os LLMs apresentam limitações. Sua precisão pode variar em diferentes contextos e tipos de fraquezas, indicando que devem ser utilizados como ferramentas auxiliares, não como substitutos para revisões manuais e testes rigorosos de software.

Trabalhos Futuros. Como trabalhos futuros, iremos aprofundar a análise por fraqueza, considerando um número maior de exemplos para cada tipo de CWE. Isso permitirá uma avaliação mais robusta do desempenho das LLMs na detecção de diferentes vulnerabilidades. Além disso, pretendemos ampliar o escopo da pesquisa para incluir um conjunto mais abrangente de fraquezas, considerando diferentes categorias da CWE e indo além das selecionadas neste estudo. Iremos também avaliar o desempenho das LLMs na detecção de fraquezas em código-fonte real maiores, utilizando projetos de repositórios *Open-Source*. Adicionalmente, investigaremos o desempenho de outros prompts e LLMs, como ChatGPT 4, Llama 3, e Gemini 1.5, que possui um *context window* maior permitindo analisar códigos reais maiores. Por fim, pretendemos detectar fraquezas usando LLMs em outros domínios como sistemas configuráveis com *ifdefs* [11, 12, 13, 14].

REFERÊNCIAS

- [1] Importance of Security in Software Development. Disponível em: <<https://medium.com/brainstation23/importance-of-security-in-software-development-f92669838a90>>. Acessado em: Abril 2024.
- [2] Common Weakness Enumeration. Disponível em: <<https://cwe.mitre.org>>. Acessado em: Abril 2024.
- [3] Stubborn Weaknesses in the CWE Top 25. Disponível em: <https://cwe.mitre.org/top25/archive/2023/2023_stubborn_weaknesses.html>. Acessado em: Abril 2024.
- [4] CWE-787: Out-of-bounds Write. Disponível em: <<https://cwe.mitre.org/data/definitions/787.html>>. Acessado em: Abril 2024.
- [5] O que são grandes modelos de linguagem (LLMs)?. Disponível em: <<https://aws.amazon.com/pt/what-is/large-language-model/>>. Acessado em: Abril 2024.
- [6] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, Shuming Shi: Siren's song in the AI ocean: a survey on hallucination in large language models. arXiv preprint arXiv:2309.01219 (2023).
- [7] Why ChatGPT And Other LLMs Generate Different Answers to Same Questions. Disponível em: <<https://blog.avfie.com/why-chatgpt-and-other-llms-generate-different-answers-to-same-questions>>. Acessado em: Abril 2024.
- [8] Raphael Muniz, Larissa Braz, Rohit Gheyi, Wilkerson de L. Andrade, Balduino Fonseca, Márcio Ribeiro: A Qualitative Analysis of Variability Weaknesses in Configurable Systems with *ifdefs*. VaMoS 2018: 51-58.
- [9] Débora Souza, Rohit Gheyi. Estudo de caso: uso do ChatGPT para resolução de problemas de programação. Concurso de Trabalhos de Iniciação Científica em Engenharia de Software (CTIC-ES), 80-89. 2023.
- [10] Flávio Medeiros, Márcio Ribeiro, Rohit Gheyi, Larissa Braz, Christian Kästner, Sven Apel, Kleber Santos: An Empirical Study on Configuration-Related Code Weaknesses. SBES 2020: 193-202.
- [11] Flávio Medeiros, Iran Rodrigues, Márcio Ribeiro, Leopoldo Teixeira, Rohit Gheyi: An empirical study on configuration-related issues: investigating undeclared and unused identifiers. GPCE 2015: 35-44.
- [12] Romero Malaquias, Márcio Ribeiro, Rodrigo Bonifácio, Eduardo Monteiro, Flávio Medeiros, Alessandro Garcia, Rohit Gheyi: The discipline of preprocessor-based annotations does *ifdef* TAG n't *endif* matter. ICPC 2017: 297-307.
- [13] Larissa Braz, Rohit Gheyi, Melina Mongiovi, Márcio Ribeiro, Flávio Medeiros, Leopoldo Teixeira, Sabrina Souto: A change-aware per-file analysis to compile configurable systems with *ifdefs*. Comput. Lang. Syst. Struct. 54: 427-450 (2018).
- [14] Larissa Braz, Rohit Gheyi, Melina Mongiovi, Márcio Ribeiro, Flávio Medeiros, Leopoldo Teixeira: A change-centric approach to compile configurable systems with *ifdefs*. GPCE 2016: 109-119.
- [15] H., Pearce, B., Tan, B., Ahmad, R., Karri, B., Dolan-Gavitt, (2023, May). Examining zero-shot vulnerability repair with large language models. In 2023 IEEE Symposium on Security and Privacy (SP) (pp. 2339-2356). IEEE.
- [16] N. T., Islam, J., Khoury, A., Seong, G. D. L. T., Parra, E., Bou-Harb, P., Najafirad (2024). LLM-Powered Code Vulnerability Repair with Reinforcement Learning and Semantic Reward. arXiv preprint arXiv:2401.03374.
- [17] Aaron Chan, Anant Kharkar, Roshanak Zilouchian Moghaddam, Yevhen Mohylevskyy, Alec Helyar, Eslam Kamal, Mohamed Elkamhawy, and Neel Sundaresan. 2023. Transformer-based Vulnerability Detection in Code at EditTime: Zero-shot, Few-shot, or Fine-tuning? arXiv preprint arXiv:2306.01754 (2023).
- [18] Wei Tang, Mingwei Tang, Minchao Ban, Ziguang Zhao, and Mingjun Feng. 2023. CSGVD: A deep learning approach combining sequence and graph embedding for source code vulnerability detection. Journal of Systems and Software 199 (2023), 111623.
- [19] J., Sallou, T., Durieux, A., Panichella (2023). Breaking the silence: the threats of using LLMs in software engineering. arXiv preprint arXiv:2312.08055.