



Wrocław
University
of Science
and Technology



HR EXCELLENCE IN RESEARCH

Zastosowanie dużych modeli językowych do wykrywania i naprawiania błędów bezpieczeństwa i podatności w kodzie aplikacji webowych

Application of Large Language Models to Detection and Correction Security Breaches and Threats in Web Application Source Codes

Patryk Fidler

259468@student.pwr.edu.pl

Promotor: dr inż. Maciej Piasecki

Jednostka prowadząca: Katedra Sztucznej Inteligencji, Wydział Informatyki i Telekomunikacji
Department of Artificial Intelligence, Faculty of Computer Science and Telecommunications

November 21, 2023



- 1 Cel pracy
- 2 Zakres pracy
- 3 Wyniki realizacji pracy
- 4 Podsumowanie
- 5 Literatura



Cel pracy

- ▶ Opracowanie praktycznych rozwiązań do statycznej analizy kodu dla aplikacji webowych.
- ▶ Badanie skuteczności dużych modeli językowych w wykrywaniu podatności i luk bezpieczeństwa.



Zakres pracy

1. Analiza istniejącej literatury i badań, w szczególności artykułu "Can OpenAI Codex and Other Large Language Models Help Us Fix Security Bugs?".
2. Projektowanie i implementacja narzędzia do statycznej analizy kodu opartego na modelach OpenAI.
3. Przygotowanie zbioru danych z kodem zawierającym potencjalne podatności.
4. Testowanie i porównanie skuteczności z innymi rozwiązaniami, np. oferowanymi przez firmę Snyk.
5. Analiza wyników i formułowanie wniosków.



Część I

Wyniki realizacji pracy

Wyniki realizacji pracy

Założenia, dane wejściowe, kryteria oceny

- ▶ Założenie: Wykorzystanie modeli GPT-3.5, GPT-4, Llama2-code do analizy bezpieczeństwa kodu.
- ▶ Dane wejściowe: Kody źródłowe aplikacji z potencjalnymi błędami bezpieczeństwa.
- ▶ Kryteria oceny: Skuteczność wykrywania podatności takich jak:
Authentication Bypass, Code Injection, Format String Attacks, Integer Overflow, NoSQL Injection, Path Traversal, Prototype Pollution, Resource Injection, SQL Injection, Unsafe Deserialization, XSS Buffer Overflow, Command Injection, Denial Of Service, IDOR LDAP Injection, Open Redirect, PHP Object Injection, Sensitive Data Exposure, SSRFUse After Free, XXE Code Execution, Connection String Injection, File Inclusion, Insecure File Uploads, Log Forging Out of Bounds, PostMessage Security, ReDoS Server Side Template Injection, Symlink Attack, XPATH Injection, Zip Traversal oraz inne.



Wyniki realizacji pracy

Metody i rozwiązania

- ▶ Efektem końcowym jest autonomiczny system umożliwiający statyczną analizę kodu źródłowego oraz wynik badawczy składający się z macierzy błędów, porównania efektywności poszczególnych modeli oraz wykresy skuteczności w różnych technologiach programistycznych.

Metody:

- ▶ Algorytmy fragmentowania
- ▶ In-context learning (uczenie się w kontekście)
- ▶ Retrieval Augmented Generation - RAG (napisany własny kod, w tej chwili dostępne jest za pomocą OpenAI Assistant API)
- ▶ Analiza porównawcza

Środki:

- ▶ Modele językowe GPT-3.5, GPT-4
- ▶ Ollama - środowisko kontenerowe dla modeli językowych, łatwe w użyciu otwartoźródłowe modele językowe
- ▶ OpenAI Assistant API
- ▶ Zbiór danych z kodem zawierającym podatności
- ▶ statyczne testy podatności jak CodeQL
- ▶ Istniejące rozwiązania komercyjne - Snyk
- ▶ Python 3.12
- ▶ Komputer osobisty

Wyniki realizacji pracy

Schematy i wyniki

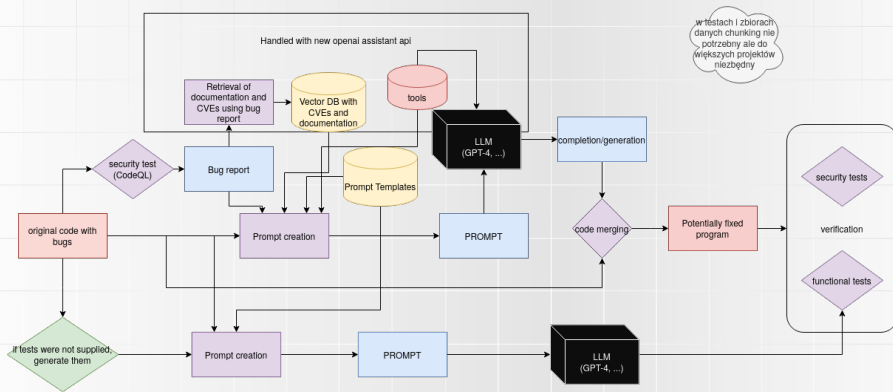


Figure: Schemat blokowy działania programu GPTester



Wyniki realizacji pracy

Oprogramowanie

```
[paris@uwuntu:~/projekty/INZYNIERKA/gptester]
[23:30:03] > ./main.py -h
usage: main.py [-h] [-v] [-m MODEL] [-o OUTPUT] [-t TESTS] [-c] [--command COMMAND]
               [--language LANGUAGE]
               directory

GPTESTER | Static Code Analysis Agent

positional arguments:
  directory              Path to the directory to scan

options:
  -h, --help            show this help message and exit
  -v, --verbose          Print out all the outputs and steps taken
  -m MODEL, --model MODEL
                        Choose the LLM model for code analysis, default: "gpt-4-1106-preview"
  -o OUTPUT, --output OUTPUT
                        Output the results to a file, default:
                        "raports/{name_of_parent_folder}_{timestamp}_raport.md"
  -t TESTS, --tests TESTS
                        Provide a path to functional tests to run on the project
  -c, --codeql           Use codeql to enhance the scan, REQUIRED to install CodeQL-CLI
                        console tool
  --command COMMAND     Provide a build command to run the project for codeql, if no cmake or
                        other file present in the project root directory, default: "make"
  --language LANGUAGE   Provide a programming language of the project for codeql, default:
                        "cpp"
[paris@uwuntu:~/projekty/INZYNIERKA/gptester]
```

Figure: Użycie programu GPTester

repozytorium projektu

<https://github.com/7oziko/gptester>



Wyniki realizacji pracy

Oprogramowanie

```
[paris@uwuntu:~/projekty/INZYNIERKA/gptester]
[23:30:20] > ./main.py Vulnerable-Code-Snippets/Buffer_Overflow/ --verbose --model gpt-4-1106-preview
2023-11-20 23:42:43: Welcome to gptester: the Static Code Analysis Agent
2023-11-20 23:42:43: I will now begin scanning: Vulnerable-Code-Snippets/Buffer_Overflow/, name: Buffer_Overflow
2023-11-20 23:42:43: Beginning scan...
2023-11-20 23:42:43: Found 7 files to scan
2023-11-20 23:42:43: File: example2.c,
...
int _tmain(int argc, _TCHAR* argv[])
{
    char name[64];
    printf("Enter your name: ");
    scanf("%s", name);
    Sanitize(name);
    printf("Welcome, %s!", name);
    return 0;
}
...
2023-11-20 23:42:43: File: gets.c,
#include <stdio.h>
int main () {
    char username[8];
    int allow = 0;
    printf external link("Enter your username, please: ");
    gets(username); // user inputs "malicious"
    if (grantAccess(username)) {
        allow = 1;
    }
    if (allow != 0) { // has been overwritten by the overflow of the username.
        privilegedAction();
    }
    return 0;
}
...
2023-11-20 23:42:43: File: example1.c,
int _tmain(int argc, _TCHAR* argv[])
{
    char name[64];
    printf("Enter your name: ");
    scanf("%s", name);
    Sanitize(name);
    printf("Welcome, %s!", name);
    return 0;
}
...
2023-11-20 23:42:43: File: bofi.c,
...
#include <stdio.h>
#include <string.h>

#define S 100
#define N 1000
```



Wyniki realizacji pracy

Oprogramowanie

```
char str1[256],str2[256];
strcpy(str1,str2);
...
2023-11-20 23:45:11: Tokens inside the directory: 885
2023-11-20 23:45:11: Beginning code analysis...
2023-11-20 23:45:11: Using model: gpt-4-1106-preview
2023-11-20 23:46:38: Writing file Vulnerable-Code-Snippets/Buffer_Overflow/fixed_2023-11-20 23:46:38/example1.c...
2023-11-20 23:46:38: content:
int _tmain(int argc, char* argv[])
{
    char name[64];
    printf("Enter your name: ");
    scanf("%63s", name);
    Sanitize(name);
    printf("Welcome, %s!", name);
    return 0;
}
...
2023-11-20 23:46:38: File 'Vulnerable-Code-Snippets/Buffer_Overflow/fixed_2023-11-20 23:46:38/example1.c' written successfully.
2023-11-20 23:46:38: Writing file Vulnerable-Code-Snippets/Buffer_Overflow/fixed_2023-11-20 23:46:38/example2.c...
2023-11-20 23:46:38: content:
int _tmain(int argc, char* argv[])
{
    char name[64];
    printf("Enter your name: ");
    scanf("%63s", name);
    Sanitize(name);
    printf("Welcome, %s!", name);
    return 0;
}
...
2023-11-20 23:46:38: File 'Vulnerable-Code-Snippets/Buffer_Overflow/fixed_2023-11-20 23:46:38/example2.c' written successfully.
2023-11-20 23:46:38: Writing file Vulnerable-Code-Snippets/Buffer_Overflow/fixed_2023-11-20 23:46:38/gets.c...
2023-11-20 23:46:38: content:
#include <stdio.h>
int main () {
    char username[8];
    int allow = 0;
    printf("Enter your username, please: ");
    fgets(username, sizeof(username), stdin); // replaced gets
    if (grantAccess(username)) {
        allow = 1;
    }
    if (allow != 0) {
        privilegedAction();
    }
    return 0;
}
...
2023-11-20 23:46:38: File 'Vulnerable-Code-Snippets/Buffer_Overflow/fixed_2023-11-20 23:46:38/gets.c' written successfully.
2023-11-20 23:46:38: Writing file Vulnerable-Code-Snippets/Buffer_Overflow/fixed_2023-11-20 23:46:38/bof.c...
```

Figure: Użycie programu GPTester

Figure: Użycie programu GPTester



Wyniki realizacji pracy

Opogramowanie - Podsumowanie napraw

2023-11-20 23:46:56: All the files with identified vulnerabilities have been fixed and written to the "fixed" folder successfully. Here is a summary of the changes made:

- 'example1.c' and 'example2.c': Usage of '%s' in 'scanf' replaced with '%63s' to prevent buffer overflow.
- 'gets.c': Replaced 'gets' with 'fgets' to prevent buffer overflow.
- 'bof1.c': Replaced 'sprintf' with 'snprintf' and corrected the buffer length check in 'strncat' to prevent buffer overflow.
- 'sprintf.c': Increased buffer size by 1 to account for the null terminator and replaced 'sprintf' with 'snprintf' to prevent buffer overflow.
- 'strcpy.c': Replaced 'strcpy' with 'strncpy' and ensured null termination to prevent buffer overflow.
- 'netkit-telnet 0.17.c': Indicated that the code is incomplete and has syntax errors; thus, a complete function context is required for an accurate fix. The current changes are placeholders and do not form a functional fix.

The 'netkit-telnet 0.17.c' file needs a complete review of the function 'env_fix_display' for an accurate fix, as the code snippet has syntax errors and seems to be cut off.



Wyniki realizacji pracy

Oprogramowanie - Szczegółowa analiza i łatki

2023-11-20 23:46:56: I will analyze the given source files for potential vulnerabilities and security bugs. After identifying issues, I will provide the git format patch and write the fixed version of the codes.

Let's begin with 'example2.c' and 'example1.c'. Both files contain similar code with a possible vulnerability:

...

Finally, 'strcpy.c':

...

Let's create patches for these vulnerabilities.

Patch for 'example2.c' and 'example1.c' (same fix for both):

```
-int _tmain(int argc, _TCHAR* argv[])
+int _tmain(int argc, char* argv[])
{
    char name[64];
    printf("Enter your name: ");
-    scanf("%s", name);
+    scanf("%63s", name); // Limit input to 63 characters + null terminator
    Sanitize(name); // Assumed to be a safe sanitization function
    printf("Welcome, %s!", name);
    return 0;
}
```

Patch for 'gets.c':

```
printf("Enter your username, please: ");
```



Wyniki realizacji pracy

Analiza i wnioski

- ▶ Model GPT-4 poprawnie radzi sobie ze znajdowaniem podatności w kodzie.
- ▶ GPT-4 w najnowszej wersji bardzo dobrze potrafi używać dostarczonych mu narzędzi i funkcji.
- ▶ Dalsze testy są potrzebne, aby określić skuteczność modelu GPT-4 oraz innych modeli w wykrywaniu podatności w kodzie.



Część II

Podsumowanie

Podsumowanie

Wnioski oraz postęp w osiągnięciu celu

- ▶ Sukces implementacji praktycznego rozwiązania statycznej analizy kodu.
- ▶ Potwierdzono bardzo dobre zdolności modeli do używania kontekstu, narzędzi i wiedzy z uczenia podstawowego w wykrywaniu podatności.
- ▶ Napisano własny kod do Generowania rozszerzonego pozyskiwaniem danych - Retrieval Augmented Generation.
- ▶ Przystosowanie programu dla dużych projektów.
- ▶ Badanie skuteczności dużych modeli językowych w wykrywaniu podatności i luk bezpieczeństwa.
- ▶ Analiza porównawcza skuteczności modeli językowych w wykrywaniu podatności w różnych stosach technologicznych.
- ▶ Opracowanie wyników badań, stworzenie przejrzystych wykresów oraz wniosków.

(Została osiągnięta większość założonych celów.)

- ▶ “Can OpenAI Codex and Other Large Language Models Help Us Fix Security Bugs?” - Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, Brendan Dolan-Gavitt. Wydawnictwo: Cornell University
- ▶ “Examining Zero-Shot Vulnerability Repair with Large Language Models” - Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, Brendan Dolan-Gavitt. Wydawnictwo: 2023 IEEE Symposium on Security and Privacy
- ▶ “Assisting Static Analysis with Large Language Models: A ChatGPT Experiment” - Haonan Li, Yu Hao, Yizhuo Zhai, Zhiyun Qian. https://www.cs.ucr.edu/~zhiyunq/pub/fseivr23_static_chatgpt.pdf
- ▶ “Detecting Code Vulnerabilities with Large Language Models (LLMs)” - Ravi Lingarkar. <https://www.linkedin.com/pulse/detecting-code-vulnerabilities-large-language-models-llms-lingarkar>
- ▶ https://github.com/chris-koch-penn/gpt3_security_vulnerability_scanner



Dziękuję za uwagę.