Politechnika Śląska

Sprawozdanie z modułu nr 3

WEBAPPSEC 2023/2024

Bezpieczeństwo webaplikacji

Kierunek: Informatyka

Członkowie zespołu:
Grzegorz Koperwas

Gliwice, 2023/2024

Spis treści

1. Wprowadzenie

1.1. Cel projektu

Celem projektu jest zabezpieczenie prostej aplikacji wykonanej w środowisku `FastAPI`, wykorzystującej model `LLava` [2] poprzez narzędzie `llama.cpp` [1].

Narzędzie oryginalnie powstało podczas hackathonu *mhack* w Wrocławiu w tym roku.

1.1.1. Zespół projektowy

- Grzegorz Koperwas :: Wszystko :: Wszystko

2. Założenia projektowe

W aplikacji będziemy zajmować się następującymi problemami, związanymi z `API4:2023 Unrestricted Resource Consumption`:

- Podczas przetwarzania więcej niż dwóch zapytań aplikacja się zacina.
- Aplikacja może zjeść całą pamięć RAM

Nie będziemy się zajmować następującymi podatnościami w aplikacji:

- Długość promptu - Model `LLava` nie przejmuje się promptem zbytnio, aplikacja była używana tylko jako wewnętrzne API ze stałym promptem.
- Rozmiar załączników - Framework `FastAPI` nie ma dobrego mechanizmu na ich ograniczenie, więc tutaj najlepszym sposobem byłoby robienie tego za pomocą serwera *proxy*.

  W przypadku tej aplikacji, była ona dostępna przez proxy Cloudflare z limitem rozmiaru zapytania ustawionym na 100mb.

2.1. Opis aplikacji

Aplikacja to prosty serwer `HTTP`, wykonany w technologii FastAPI w języku programowania `python`. Aplikacja przetwarza zapytania w sposób asynchroniczny, za pomocą wbudowanego w interpreter modułu `asyncio`.

Interesująca nas część programu została przedstawiona na załączniku 1.

3. Realizacja projektu

W celu zabezpieczenia naszego programu musimy:

- Ograniczyć ile procesów `llava` może zostać naraz wywołanych.
- Ograniczyć maksymalną ilość czasu, jaki może być przeznaczony na taki proces.

W tym celu skorzystamy z mechanizmów udostępnionych przez wbudowany w interpreter moduł `asyncio`. Wykorzystamy strukturę `asyncio.Lock` oraz `asyncio.timeout`.

W asynchronicznym programie może dojść do sytuacji, gdzie dwa zapytania są przetwarzane w tym samym czasie. Musimy jednak ograniczyć ile niepowiązanych ze sobą zapytań, może naraz wywoływać model. Najprostsze rozwiązanie to wprowadzić semafor dla fragmentu kodu wywołującego nasz model.

Co więcej, do zapytania dodamy logikę, która po przekroczeniu pewnego czasu podczas wykonywania modelu zabije wszystkie procesy związane z modelem.

Poprawiony kod znajduje się na załączniku 2.

4. Podsumowanie i wnioski

- *Podsumowanie* - Ograniczenie maksymalnej ilości instancji modelu gwarantuje nam pewien maksymalny poziom zużycia pamięci
- *Wnioski* - Podczas używania rozwiązań opartych o LLM musimy zdawać sobie sprawę, że czas oczekiwania na odpowiedź od takiego modelu może być długi

5. Spis literatury

[1] Georgi Gerganov. *llama.cpp*. 2023. URL: `https://github.com/ggerganov/llama.cpp` (term. wiz. 30. 11. 2023).

[2] Haotian Liu i in. *Improved Baselines with Visual Instruction Tuning*. 2023.

```python
@app.post("/image/do_stuff")
async def do_stuff(file: UploadFile, prompt: str):
    command = f"convert /dev/stdin  -scale 1024x1024^ bmp:- | llava -s 2137 [...]"
    proc = await create_subprocess_shell(
        command,
        stdout=asyncio.subprocess.PIPE,
        stdin=asyncio.subprocess.PIPE,
    )
    stdout, _ = await proc.communicate(await file.read())

    if proc.returncode:
        return PlainTextResponse(
            content=f"Process died with exit code {proc.returncode}",
            status_code=500,
        )

    _, not_garbage = stdout.decode().split("prompt:", maxsplit=1)
    not_garbage = not_garbage.split("\n", maxsplit=1)[1]

    not_garbage, _ = not_garbage.split("main:", maxsplit=1)

    return Response(answer=not_garbage.strip())
```

Załącznik 1: Fragment kodu z podatnością.

```python
llava_lock = asyncio.Lock()


@app.post("/image/do_stuff")
async def do_stuff(file: UploadFile, prompt: str):
    command = f"convert /dev/stdin  -scale 1024x1024^ bmp:- | llava -s 2137 [...]"

    async with llava_lock:
        proc = await create_subprocess_shell(
            command,
            stdout=asyncio.subprocess.PIPE,
            stdin=asyncio.subprocess.PIPE,
        )
        try:
            async with asyncio.timeout(20):
                stdout, _ = await proc.communicate(await file.read())
        except TimeoutError:
            terminate_process_with_children(proc.pid)
            return PlainTextResponse(
                content=f"Timeout",
                status_code=500,
            )

    if proc.returncode:
        return PlainTextResponse(
            content=f"Process died with exit code {proc.returncode}",
            status_code=500,
        )

    _, not_garbage = stdout.decode().split("prompt:", maxsplit=1)
    not_garbage = not_garbage.split("\n", maxsplit=1)[1]

    not_garbage, _ = not_garbage.split("main:", maxsplit=1)

    return Response(answer=not_garbage.strip())
```

Załącznik 2: Poprawiony program.