# LAPORAN PROYEK AKHIR

## PEMROGRAMAN JARINGAN

# S.C.A.R.

*System for Countering Automated Reconnaissance*

Active Defense Honeypot dengan Multi-Layer AI
dan Tarpit Resource Exhaustion


**Disusun oleh:**

| | |
|---|---|
| Falito Eriano Nainggolan | (2423102023) |
| Luklu Miranda | (2423102039) |
| Reiza Gerrard Rizki Ramadhan | (2423102070) |

**Dosen Pengampu:**

Bapak Agus Winarno, S.S.T.TP, M.T.

**Tingkat II Rekayasa Keamanan Siber A**

**POLITEKNIK SIBER DAN SANDI NEGARA**

**Semester Gasal 2025/2026**

# DAFTAR ISI

# DAFTAR GAMBAR



# DAFTAR TABEL

# BAB I

# PENDAHULUAN

## 1.1 Latar Belakang

Dalam konteks pemrograman jaringan, server HTTP merupakan salah satu komponen fundamental yang berperan sebagai titik masuk utama komunikasi antara klien dan layanan. Server menerima *request* dari klien melalui protokol TCP/IP, memproses *request* tersebut, dan mengirimkan *response* melalui *socket* jaringan. Akan tetapi, setiap server yang terekspos ke jaringan publik secara inheren menjadi vektor serangan potensial bagi aktor ancaman siber.

Berdasarkan data yang dipublikasikan dalam *Verizon Data Breach Investigations Report* (2024), serangan terhadap aplikasi web mengalami peningkatan sebesar 30% secara tahunan. Server yang terhubung ke internet menerima ratusan *request* pemindaian otomatis setiap harinya, yang berasal dari *bot*, *scanner*, maupun *script* berbahaya. Serangan seperti *SQL Injection*, *Directory Traversal*, dan *Automated Reconnaissance* merupakan ancaman utama yang harus dimitigasi oleh setiap pengelola server jaringan.

Konsep *honeypot*, yang diperkenalkan pertama kali oleh Lance Spitzner dalam karyanya *"Honeypots: Tracking Hackers"* (2003), merujuk pada server umpan yang sengaja dirancang untuk menarik dan mempelajari perilaku penyerang. Konsep ini memanfaatkan pemahaman mendalam tentang protokol jaringan — meliputi mekanisme *TCP three-way handshake*, siklus *HTTP request-response*, serta manipulasi pada level protokol untuk tujuan defensif.

Proyek **S.C.A.R. (*System for Countering Automated Reconnaissance*)** mengembangkan konsep *honeypot* konvensional dengan mengimplementasikan paradigma ***Active Defense***. Berbeda dengan *honeypot* tradisional yang bersifat pasif (hanya mencatat aktivitas serangan), S.C.A.R. secara aktif memanfaatkan karakteristik protokol HTTP dan mekanisme koneksi TCP untuk **menjebak** penyerang melalui teknik *Tarpit* (*Resource Exhaustion*). Teknik ini mengeksploitasi spesifikasi *chunked transfer encoding* pada HTTP/1.1 untuk mempertahankan koneksi TCP tetap terbuka dan menguras sumber daya komputasi di sisi penyerang — sebuah pendekatan yang dikenal sebagai *TCP state exhaustion* dan *HTTP response stream manipulation*.

Secara spesifik, proyek ini mengintegrasikan aspek-aspek inti pemrograman jaringan sebagai berikut:

- ***Socket Programming*** — Pembangunan server HTTP menggunakan modul `http.server` dan `socketserver` Python yang beroperasi pada level *socket* TCP.
- **Manipulasi Protokol HTTP** — Pemanfaatan *headers*, *status codes*, dan *transfer encoding* untuk keperluan pertahanan aktif.
- **Integrasi REST API** — Komunikasi dengan layanan eksternal (AbuseIPDB, Telegram Bot) melalui protokol HTTP sebagai *client*.
- ***Concurrency*** — Penerapan *multi-threading* untuk penanganan koneksi simultan dan operasi I/O *non-blocking*.

- ***Machine Learning*** — Pemanfaatan model AI sebagai komponen analisis lalu lintas jaringan secara *real-time*.

## 1.2 Rumusan Masalah

1. Bagaimana membangun server HTTP menggunakan Python yang mampu menangani request masuk, menganalisis kontennya, dan memberikan respons yang sesuai secara *real-time*?
2. Bagaimana memanipulasi protokol HTTP (khususnya *chunked transfer encoding* dan *custom headers*) untuk mengimplementasikan mekanisme *Tarpit* yang efektif menghabiskan sumber daya penyerang?
3. Bagaimana mengintegrasikan layanan API eksternal (AbuseIPDB, Telegram Bot API) ke dalam server melalui komunikasi HTTP *client-side*?
4. Bagaimana menerapkan *multi-threading* agar server tetap responsif saat menangani koneksi tarpit dan pengiriman notifikasi secara bersamaan?

## 1.3 Tujuan Proyek

1. Membangun server HTTP aktif berbasis Python yang berfungsi sebagai honeypot dengan kemampuan *Active Defense*.
2. Mengimplementasikan mekanisme *Tarpit* yang memanfaatkan *chunked transfer encoding* HTTP untuk menjebak dan menghabiskan sumber daya penyerang.
3. Mengintegrasikan AI (*Machine Learning*) sebagai komponen analisis lalu lintas jaringan untuk mendeteksi serangan secara otomatis.
4. Mengintegrasikan API eksternal (AbuseIPDB dan Telegram Bot) melalui HTTP REST API untuk *threat intelligence* dan notifikasi.
5. Menerapkan arsitektur *multi-threaded* untuk menangani operasi I/O non-blocking.

## 1.4 Batasan Proyek

1. Server dibangun menggunakan modul `http.server` dari pustaka standar Python.
2. Sistem dirancang untuk berjalan pada lingkungan lokal atau server tunggal.
3. Pengujian dilakukan menggunakan *script* simulasi serangan.
4. Model AI dilatih menggunakan dataset publik yang tersedia secara gratis.

# BAB II

# TINJAUAN PUSTAKA

## 2.1 Protokol HTTP dan Arsitektur Request-Response

HTTP (*Hypertext Transfer Protocol*), didefinisikan dalam RFC 2616, adalah protokol *application layer* yang berjalan di atas koneksi TCP. Setiap komunikasi HTTP mengikuti pola *request-response*:

1. **Klien** membuka koneksi TCP ke server (biasanya port 80 atau 8080).
2. Klien mengirim **HTTP Request** yang terdiri dari: *request line* (metode, path, versi), *headers*, dan opsional *body*.
3. **Server** menerima request, memproses, dan mengirim **HTTP Response** yang terdiri dari: *status line* (kode status), *headers*, dan *body*.
4. Koneksi bisa dipertahankan (*keep-alive*) atau ditutup.

Pemahaman mendalam tentang siklus ini sangat penting dalam S.C.A.R. karena mekanisme tarpit bekerja dengan cara memanipulasi tahap ke-3 — mengirim response yang **tidak pernah selesai**.

## 2.2 Transfer-Encoding: Chunked

*Chunked Transfer Encoding* (RFC 2616, Section 3.6.1) adalah mekanisme HTTP yang memungkinkan server mengirim data secara bertahap tanpa harus mengetahui total ukuran konten di awal. Format pengiriman:

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked

[ukuran chunk dalam hex]\r\n
[data chunk]\r\n
[ukuran chunk berikutnya]\r\n
[data chunk berikutnya]\r\n
0\r\n                          (chunk terakhir, ukuran 0)
\r\n                           (trailer kosong)
```

Listing 1: Format Chunked Transfer Encoding

Fitur kunci yang dieksploitasi oleh S.C.A.R.: selama server belum mengirim chunk terakhir (ukuran 0), **klien wajib menunggu** data tambahan. Ini adalah dasar mekanisme tarpit — server mengirim data sampah secara terus-menerus dengan jeda yang panjang, memaksa klien tetap terhubung.

## 2.3 Socket Programming dan http.server Python

Modul `http.server` Python dibangun di atas modul `socketserver` yang menyediakan abstraksi *socket* TCP. Arsitekturnya:

- `TCPServer`: Mengelola *socket* server, melakukan `bind()` dan `listen()` pada port tertentu.
- `BaseHTTPRequestHandler`: Menangani setiap koneksi TCP yang masuk, mem-*parse* HTTP request, dan memanggil metode `do_GET()`, `do_POST()`, dll.
- `self.wfile`: *File-like object* yang terhubung langsung ke *socket* klien, memungkinkan penulisan data langsung ke koneksi TCP.

Akses langsung ke `self.wfile` sangat krusial untuk implementasi tarpit, karena memungkinkan pengiriman data byte-per-byte langsung ke *socket* tanpa buffering HTTP standar.

## 2.4 Honeypot dalam Keamanan Jaringan

*Honeypot*, menurut Lance Spitzner, adalah "sumber daya keamanan informasi yang nilainya terletak pada interaksi yang tidak sah." Dari perspektif pemrograman jaringan, honeypot adalah server yang sengaja menyediakan *service* palsu (HTTP, SSH, FTP) untuk menarik dan mempelajari penyerang.

Klasifikasi berdasarkan interaksi:

- **Low-Interaction**: Mengemulasi layanan jaringan secara terbatas. Aman, mudah di-*deploy*.
- **High-Interaction**: Menyediakan layanan nyata. Informatif tetapi berisiko.

S.C.A.R. termasuk *Low-Interaction Honeypot* dengan kemampuan *Active Defense* — memadukan keamanan *low-interaction* dengan agresivitas *active defense*.

## 2.5 Tarpit sebagai Mekanisme Pertahanan Aktif

*Tarpit* (atau *tar pit*) merupakan teknik manipulasi koneksi jaringan yang bertujuan memperlambat komunikasi secara drastis melalui *TCP state exhaustion*. Terminologi ini terinspirasi dari fenomena lubang aspal alami (*La Brea Tar Pits*) yang menjebak organisme purba. Tom Liston mengimplementasikan konsep ini pertama kali dalam proyek *LaBrea Tarpit* (2003) sebagai mekanisme pertahanan terhadap propagasi *worm* jaringan.

Dari perspektif pemrograman jaringan, mekanisme tarpit beroperasi melalui tiga prinsip utama:

1. ***TCP State Exhaustion*** — Server mempertahankan koneksi TCP dalam state `ESTABLISHED` tanpa batas waktu, sehingga *socket file descriptor*, *thread*, dan alokasi memori di sisi klien tetap terpakai dan tidak dapat didaur ulang.

2. ***HTTP Response Stream Manipulation*** — Server mengeksploitasi mekanisme *chunked transfer encoding* untuk mengirimkan data *garbage* secara kontinu dengan interval yang diperpanjang (*slow drip*), memaksa klien tetap dalam mode *receive-wait*.

3. ***Connection Pool Saturation*** — Setiap koneksi tarpit mengikat satu *connection pool entry* pada sisi penyerang, sehingga kapasitas *concurrent connection* penyerang terdegradasi secara progresif.

Keunggulan pendekatan tarpit dibandingkan pemblokiran konvensional (*firewall drop/reject*):

- Penyerang **tidak mengetahui** bahwa identitasnya telah terdeteksi, karena server tetap mengirimkan respons awal `HTTP 200 OK` yang valid.
- *Tool* otomatis (SQLMap, Nikto, DirBuster) mengalami kondisi *hang* akibat menunggu penyelesaian transfer yang tidak pernah berakhir.
- Kecepatan *scanning* otomatis terdegradasi secara signifikan, memberikan waktu tambahan bagi administrator untuk melakukan mitigasi.

## 2.6 REST API dan Integrasi Layanan Eksternal

REST (*Representational State Transfer*) API adalah arsitektur komunikasi jaringan yang menggunakan protokol HTTP standar untuk pertukaran data antar sistem. Dalam S.C.A.R., server bertindak sebagai **HTTP client** ketika berkomunikasi dengan layanan eksternal:

- **AbuseIPDB API**: Server mengirim HTTP GET request ke `api.abuseipdb.com` untuk memeriksa reputasi IP. Response berupa JSON yang berisi skor kepercayaan.
- **Telegram Bot API**: Server mengirim HTTP POST request ke `api.telegram.org` untuk mengirim notifikasi. Data dikirim dalam format JSON menggunakan `Content-Type: application/json`.

Kedua API ini mendemonstrasikan konsep penting dalam pemrograman jaringan: sebuah aplikasi dapat bertindak sebagai **server dan client secara bersamaan** — menerima koneksi dari penyerang (server) sambil mengirim request ke layanan luar (client).

## 2.7 Threading dan Concurrency dalam Server Jaringan

Dalam pemrograman jaringan, *concurrency* sangat penting karena server harus menangani banyak koneksi secara bersamaan. Python menyediakan modul `threading` yang memungkinkan eksekusi paralel.

Dalam S.C.A.R., threading digunakan untuk:

- **ThreadingTCPServer**: Setiap koneksi klien ditangani oleh *thread* terpisah, sehingga satu koneksi tarpit tidak memblokir klien lain.

- **Non-blocking API calls**: Pengiriman alert Telegram dilakukan di *background thread* agar proses tarpit tidak tertunda oleh latensi jaringan.
- **Thread-safe caching**: Hasil pengecekan AbuseIPDB di-*cache* menggunakan `threading.Lock()` untuk menghindari *race condition*.

## 2.8 *Machine Learning* sebagai Komponen Analisis Lalu Lintas Jaringan

Dalam konteks pemrograman jaringan, *Machine Learning* (ML) digunakan sebagai komponen analisis cerdas yang memproses data dari lalu lintas jaringan (*HTTP request*) untuk mengklasifikasikan setiap *request* sebagai normal atau ancaman secara otomatis. S.C.A.R. mengintegrasikan empat model ML yang masing-masing menganalisis aspek berbeda dari *request* HTTP:

Tabel 1: Model AI dan Data Jaringan yang Dianalisis

| **Model** | **Algoritma** | **Data Jaringan yang Dianalisis** |
|---|---|---|
| URL Threat | Logistic Regression | URL *path* dari *request line* HTTP |
| SQL Injection | Logistic Regression | *Query string* dan *body* dari HTTP *request* |
| HTTP Behavior | Random Forest | Fitur statistik: panjang URL, jumlah parameter, metode HTTP |
| Anomaly Detection | Isolation Forest | Profil keseluruhan *request* dibandingkan *baseline* normal |

Pemilihan algoritma *Random Forest* (Breiman, 2001) untuk model HTTP Behavior didasarkan pada kemampuannya menangani fitur heterogen (numerik dan kategorikal) serta resistensinya terhadap *overfitting* melalui mekanisme *ensemble* dari multiple *decision trees*. Sementara itu, *Isolation Forest* (Liu et al., 2008) dipilih untuk deteksi anomali karena sifatnya yang *unsupervised* — tidak memerlukan label serangan eksplisit, melainkan mengidentifikasi *request* yang menyimpang secara statistik dari distribusi normal. Pendekatan ini memungkinkan deteksi pola serangan *zero-day* yang belum terdokumentasi dalam basis data aturan statis.

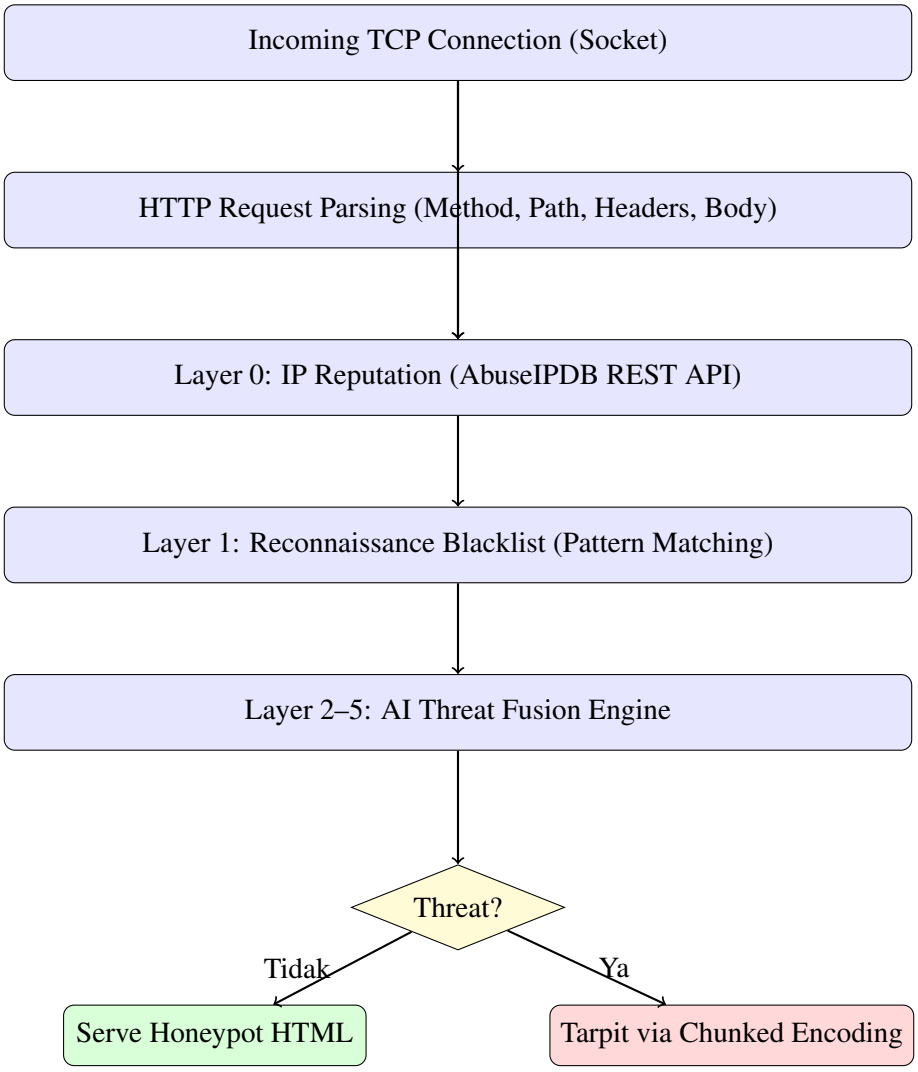
Pemilihan seluruh algoritma juga didasarkan pada efisiensi komputasi prediksi *real-time*, yang merupakan persyaratan kritis untuk server jaringan — setiap *request* harus dianalisis tanpa menimbulkan latensi yang terdeteksi oleh klien.

# BAB III

# PERANCANGAN SISTEM

## 3.1 Arsitektur Server

S.C.A.R. dibangun di atas arsitektur *multi-threaded HTTP server* menggunakan `ThreadingTCPServ` dari Python. Arsitektur ini memungkinkan server menangani banyak koneksi secara bersamaan — penting ketika satu koneksi sedang di-tarpit (bisa berlangsung puluhan detik) sementara koneksi lain tetap harus dilayani.




Gambar 1: Alur Pemrosesan Request pada Server S.C.A.R.

## 3.2 Alur Komunikasi Jaringan

Berikut adalah alur komunikasi jaringan lengkap ketika sebuah request masuk ke S.C.A.R.:

1. **TCP Handshake**: Klien melakukan *three-way handshake* (SYN, SYN-ACK, ACK) dengan server pada port 8000.
2. **HTTP Request**: Klien mengirim request melalui *socket* yang telah terhubung. Server mem-*parse request line*, *headers*, dan *body*.
3. **API Call (Client-Side)**: Server mengirim HTTP GET request ke AbuseIPDB untuk cek reputasi IP (server bertindak sebagai HTTP *client*).
4. **Analisis**: Request dianalisis oleh Reconnaissance Blacklist dan 4 model AI.
5. **Response**:
   - *Clean*: Server mengirim `HTTP 200 OK` dengan halaman HTML honeypot.
   - *Threat*: Server mengirim `HTTP 200 OK` diikuti *chunked encoding* dengan data sampah (tarpit).
6. **Telegram Alert**: Jika ancaman terdeteksi, *background thread* mengirim HTTP POST ke Telegram Bot API.

## 3.3 Mekanisme Tarpit (*HTTP Response Stream Manipulation*)

Mekanisme tarpit merupakan inti dari strategi pertahanan aktif S.C.A.R. Secara teknis, tarpit beroperasi dengan memanipulasi *HTTP response stream* pada level *socket* TCP:

```
>> HTTP/1.1 200 OK\r\n              (status line - penyerang pikir
   berhasil)
>> Transfer-Encoding: chunked\r\n   (memberitahu klien: data akan
   dikirim bertahap)
>> Server: SCAR-Active-Defense\r\n  (custom header)
>> \r\n                             (akhir header, mulai body)
>> X-Trap-482917: a3f8c1d...\r\n    (data sampah #1)
   [jeda 5 detik]
>> X-Trap-193847: 7bc2e5f...\r\n    (data sampah #2)
   [jeda 5 detik]
>> X-Trap-572910: 1de9f3a...\r\n    (data sampah #3)
   [jeda 5 detik]
   ... (berlanjut hingga 1.000.000 header atau klien disconnect)
```

Listing 2: Urutan Byte yang Dikirim melalui Socket saat Tarpit

Efektivitas mekanisme ini terletak pada eksploitasi spesifikasi HTTP/1.1 yang mewajibkan klien menunggu penyelesaian transfer:

- Klien HTTP **wajib menunggu** pengiriman *terminating chunk* (ukuran 0) sebelum dapat memproses respons secara utuh, sesuai RFC 2616 Section 3.6.1.
- Dengan konfigurasi *delay* 5 detik per *header*, dibutuhkan waktu $\sim$**58 hari** untuk menyelesaikan pengiriman seluruh 1.000.000 *garbage header* — menghasilkan rasio *time waste* mendekati tak terhingga.

- Setiap koneksi tarpit mengonsumsi *socket file descriptor*, *thread*, dan alokasi memori pada sisi penyerang, mengakibatkan *connection pool saturation* secara progresif.
- *Tool* otomatis (SQLMap, Nikto, DirBuster) tidak memiliki heuristik untuk mendeteksi bahwa respons yang diterima merupakan jebakan, karena *status code* awal yang dikirimkan tetap valid (`HTTP 200 OK`).

## 3.4 Integrasi REST API

S.C.A.R. bertindak sebagai *HTTP client* saat berkomunikasi dengan API eksternal:

### 3.4.1 AbuseIPDB API

```
# HTTP GET Request ke AbuseIPDB
GET /api/v2/check?ipAddress=1.2.3.4 HTTP/1.1
Host: api.abuseipdb.com
Key: [API_KEY]
Accept: application/json

# HTTP Response dari AbuseIPDB
{
  "data": {
    "ipAddress": "1.2.3.4",
    "abuseConfidenceScore": 85,
    "totalReports": 42
  }
}
```

Listing 3: Komunikasi dengan AbuseIPDB REST API

### 3.4.2 Telegram Bot API

```
# HTTP POST Request ke Telegram
POST /bot[TOKEN]/sendMessage HTTP/1.1
Host: api.telegram.org
Content-Type: application/json

{
  "chat_id": "[CHAT_ID]",
  "text": "THREAT ALERT: IP 1.2.3.4 ...",
  "parse_mode": "Markdown"
}
```

Listing 4: Komunikasi dengan Telegram Bot API

Kedua integrasi ini mendemonstrasikan bagaimana sebuah server jaringan dapat bertindak sebagai *client* dan *server* secara bersamaan — menerima koneksi penyerang sambil mengirim data ke layanan luar.

## 3.5 Threat Fusion Logic

Hasil analisis dari 4 model AI digabungkan menggunakan logika fusi *Hard/Soft Flag*:

Tabel 2: Sistem Hard Flag dan Soft Flag

| **Layer** | **Model AI** | **Tipe Flag** | **Bisa Trigger Tarpit Sendiri** |
|---|---|---|---|
| Layer 2 | URL Threat | Hard | Ya |
| Layer 3 | SQL Injection | Hard | Ya |
| Layer 4 | HTTP Behavior | Soft | Tidak (butuh konfirmasi) |
| Layer 5 | Anomaly Detection | Soft | Tidak (butuh konfirmasi) |

**Aturan:**

- $\geq 1$ Hard Flag $\rightarrow$ **Tarpit** langsung.
- $\geq 2$ Soft Flags $\rightarrow$ **Tarpit** (konfirmasi silang).
- 1 Soft Flag saja $\rightarrow$ **Warning** (dicatat, request dilayani).

Logika ini mencegah *false positive* — request normal yang kebetulan ditandai oleh satu model saja tidak akan memicu tarpit.

# BAB IV

# IMPLEMENTASI

## 4.1 Lingkungan Pengembangan

Tabel 3: Spesifikasi Lingkungan

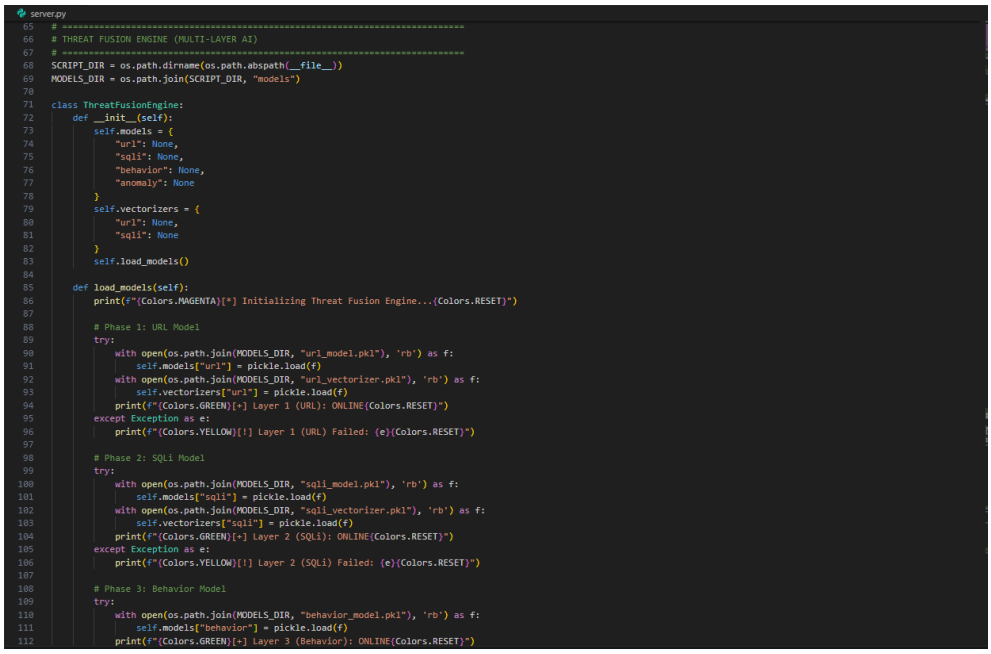
| **Komponen** | **Detail** |
|---|---|
| Bahasa | Python 3.8+ |
| Server Framework | `http.server` + `socketserver` (standar library) |
| HTTP Client | `requests` library |
| ML Libraries | scikit-learn, numpy, pandas |
| Concurrency | `threading` module |
| Version Control | Git + GitHub |

## 4.2 Struktur Proyek

```
Project_SCAR/
|-- server.py                # HTTP Server + Fusion Engine + Tarpit
|-- attacker_simulation.py   # HTTP Client Simulasi Serangan
|-- default.html             # Honeypot HTML Page
|-- requirements.txt         # Dependencies
|-- models/                  # Pre-trained AI Models (.pkl)
|   |-- url_model.pkl
|   |-- url_vectorizer.pkl
|   |-- sqli_model.pkl
|   |-- sqli_vectorizer.pkl
|   |-- behavior_model.pkl
|   +-- anomaly_model.pkl
+-- README.md
```

Listing 5: Struktur Direktori Proyek

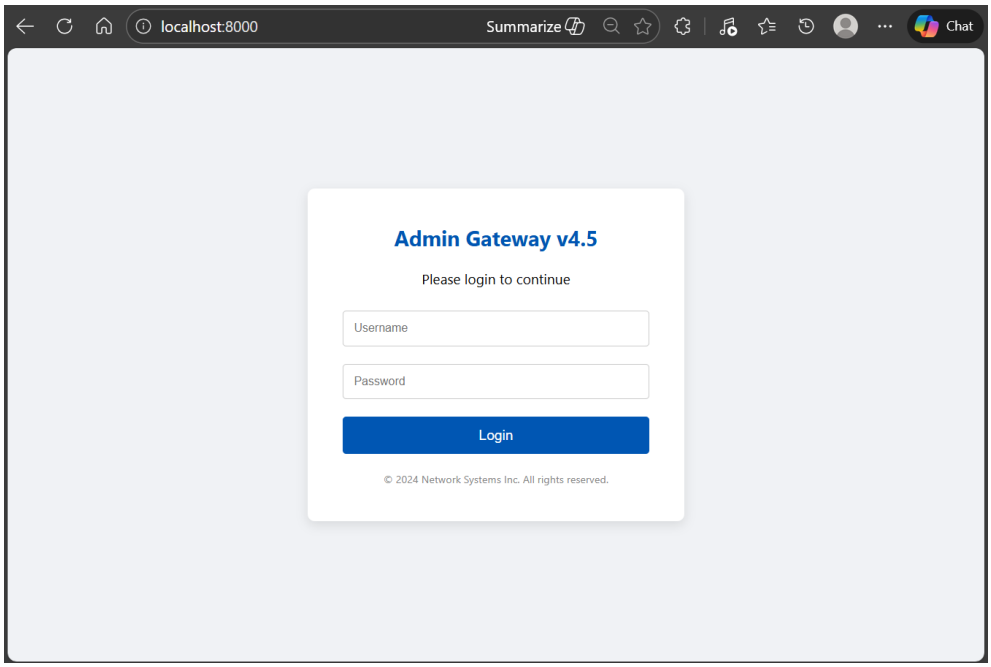
## 4.3 Tampilan Kode Server




Gambar 2: Tampilan kode `server.py` di Visual Studio Code

## 4.4 Tampilan Halaman Honeypot

Halaman HTML palsu disajikan kepada pengunjung normal melalui HTTP response standar.




Gambar 3: Tampilan halaman honeypot di browser (`http://localhost:8000`)

## 4.5 Implementasi HTTP Server

Server dibangun menggunakan `ThreadingHTTPServer` yang mewarisi `socketserver.Threa` memungkinkan penanganan multi-koneksi:

```
class ThreadingHTTPServer(socketserver.ThreadingMixIn,
                          http.server.HTTPServer):
    daemon_threads = True

server = ThreadingHTTPServer(("0.0.0.0", 8000), CustomHandler)
server.serve_forever()
```

Listing 6: Inisialisasi Server dengan Threading

Kelas `CustomHandler` menangani setiap request dengan alur:

```
def handle_request(self):
    client_ip = self.client_address[0]    # IP dari socket
    method = self.command                   # GET, POST, dll
    path = self.path                        # URL path + query

    # Baca body dari socket (jika ada)
    content_length = int(self.headers.get('Content-Length', 0))
    body = self.rfile.read(content_length).decode('utf-8')

    # Proses melalui lapisan pertahanan...
```

Listing 7: Handler utama – Parsing HTTP Request

## 4.6 Implementasi Tarpit (Socket-Level)

Ini adalah bagian *core* dari perspektif pemrograman jaringan — pengiriman data langsung ke *socket* TCP:

```
def execute_tarpit(self, client_ip, reasons, risk_score=0.0):
    # Kirim Telegram alert di background thread
    send_telegram_alert(client_ip, reasons, risk_score)

    # Tulis langsung ke socket melalui self.wfile
    self.wfile.write(b"HTTP/1.1 200 OK\r\n")
    self.wfile.write(b"Transfer-Encoding: chunked\r\n")
    for key, val in FAKE_HEADERS.items():
        self.wfile.write(f"{key}: {val}\r\n".encode())
    self.wfile.write(b"\r\n")

    # Kirim garbage headers secara kontinu
    count = 0
    while count < TARPIT_HEADER_COUNT:
        random_id = random.randint(100000, 999999)
        random_hex = hashlib.sha256(
```

```
            os.urandom(32)).hexdigest()
        garbage = f"X-Trap-{random_id}: {random_hex}\r\n"
        self.wfile.write(garbage.encode())
        self.wfile.flush()    # Force kirim ke socket
        count += 1
        time.sleep(TARPIT_DELAY_SECONDS)  # Jeda 5 detik
```

Listing 8: Implementasi Tarpit melalui Socket Write

Perhatikan penggunaan `self.wfile.write()` dan `self.wfile.flush()` — ini menulis byte langsung ke koneksi TCP socket yang terhubung ke klien.

## 4.7 Implementasi Non-Blocking Telegram Alert

Pengiriman alert menggunakan *background thread* agar proses tarpit tidak terganggu:

```
def send_telegram_alert(ip, reasons, risk_score=0.0):
    def _send():
        url = f"https://api.telegram.org/bot{TOKEN}/sendMessage"
        data = {"chat_id": CHAT_ID,
                "text": format_message(ip, reasons, risk_score),
                "parse_mode": "Markdown"}
        requests.post(url, json=data, timeout=5)

    # Jalankan di background thread
    threading.Thread(target=_send, daemon=True).start()
```

Listing 9: Non-Blocking API Call dengan Threading

Ini mendemonstrasikan pola *Fire-and-Forget* dalam pemrograman jaringan — mengirim request tanpa menunggu hasilnya, agar *main thread* tetap fokus pada koneksi tarpit.

## 4.8 Implementasi IP Reputation Check

```
IP_CACHE = {}  # Dictionary untuk caching
IP_CACHE_LOCK = threading.Lock()  # Thread-safe access

def check_ip_reputation(ip):
    with IP_CACHE_LOCK:
        if ip in IP_CACHE:
            cached = IP_CACHE[ip]
            if time.time() - cached['time'] < CACHE_TTL:
                return cached['result']

    # HTTP GET request ke AbuseIPDB
    headers = {"Key": ABUSEIPDB_API_KEY,
               "Accept": "application/json"}
    resp = requests.get(
        f"https://api.abuseipdb.com/api/v2/check",
```

```
        params={"ipAddress": ip, "maxAgeInDays": 90},
        headers=headers, timeout=5)
    data = resp.json()["data"]

    # Cache hasil dengan thread-safe lock
    with IP_CACHE_LOCK:
        IP_CACHE[ip] = {"result": data, "time": time.time()}
    return data
```

Listing 10: REST API Call ke AbuseIPDB dengan Caching

Caching dengan `threading.Lock()` mendemonstrasikan penanganan *race condition* — ketika beberapa *thread* mencoba mengakses cache secara bersamaan.

# BAB V

# PENGUJIAN DAN HASIL

## 5.1 Skenario Pengujian

Pengujian dilakukan menggunakan `attacker_simulation.py`, sebuah HTTP client yang mengirim berbagai jenis request ke server S.C.A.R.:

Tabel 4: Skenario Pengujian

| No | Skenario | HTTP Request | Ekspektasi |
|---|---|---|---|
| 1 | Normal User | `GET / HTTP/1.1` | HTTP 200 + HTML |
| 2 | Normal User | `GET /images/logo.png HTTP/1.1` | HTTP 200 + HTML |
| 3 | Recon Attack | `GET /cgi-bin/...?val=../../../../bin/ls` | Tarpit |
| 4 | SQL Injection | `GET /?cat=1 AND 1=1 HTTP/1.1` | Tarpit |
| 5 | Anomaly | `GET /search?q=><;()@!#... HTTP/1.1` | Tarpit |

## 5.2 Hasil Pengujian

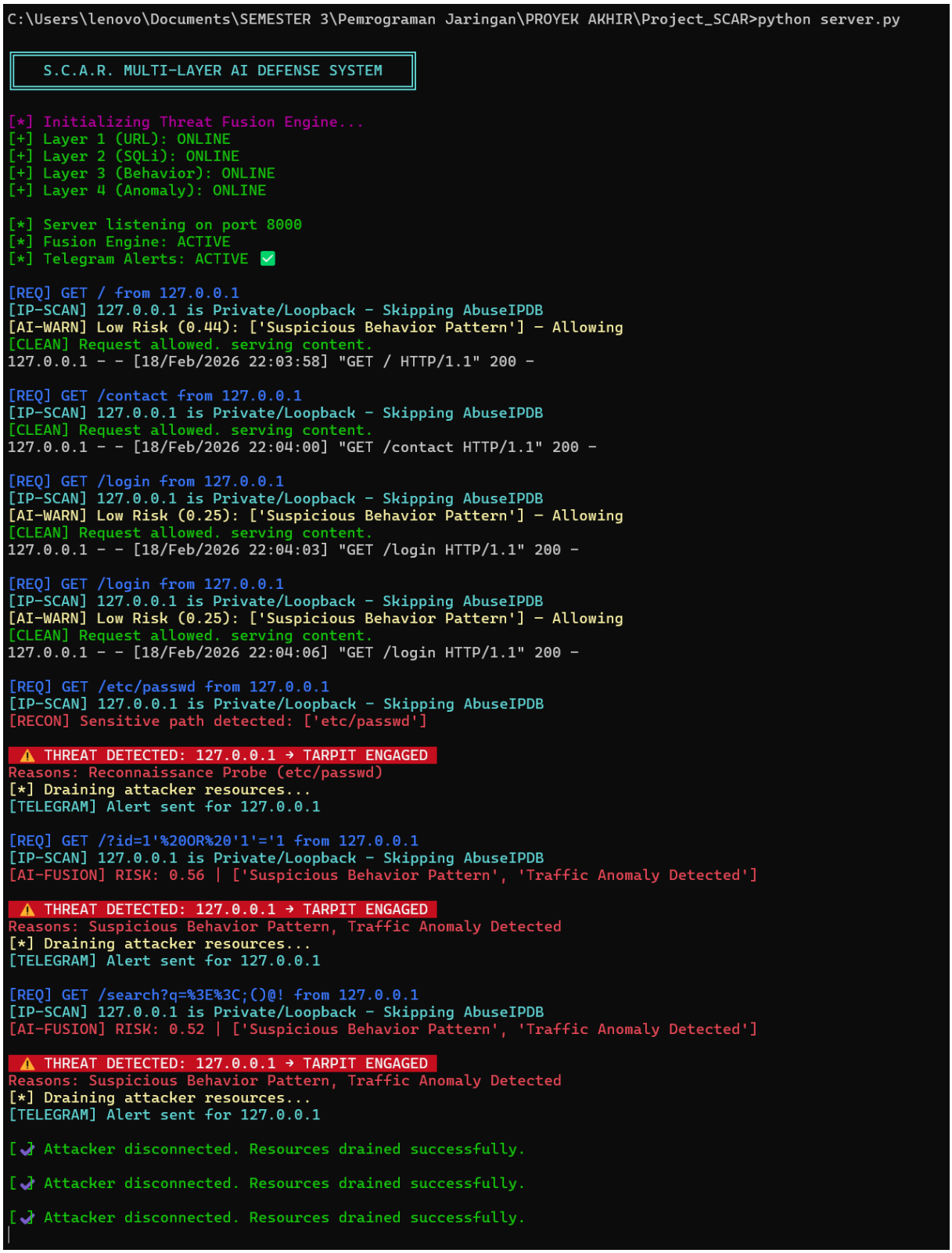
Tabel 5: Hasil Pengujian

| No | Skenario | Hasil | Risk | Respons Jaringan |
|---|---|---|---|---|
| 1 | Normal GET / | Lolos | 0.44 | HTTP 200 + HTML body |
| 2 | Normal /login | Lolos | 0.25 | HTTP 200 + HTML body |
| 3 | Recon (/etc/passwd) | Tarpit | >0.80 | HTTP 200 + garbage stream |
| 4 | SQL Injection | Tarpit | 0.56 | HTTP 200 + garbage stream |
| 5 | Anomalous | Tarpit | 0.52 | HTTP 200 + garbage stream |

## 5.3 Bukti Pengujian

### 5.3.1 Terminal Server

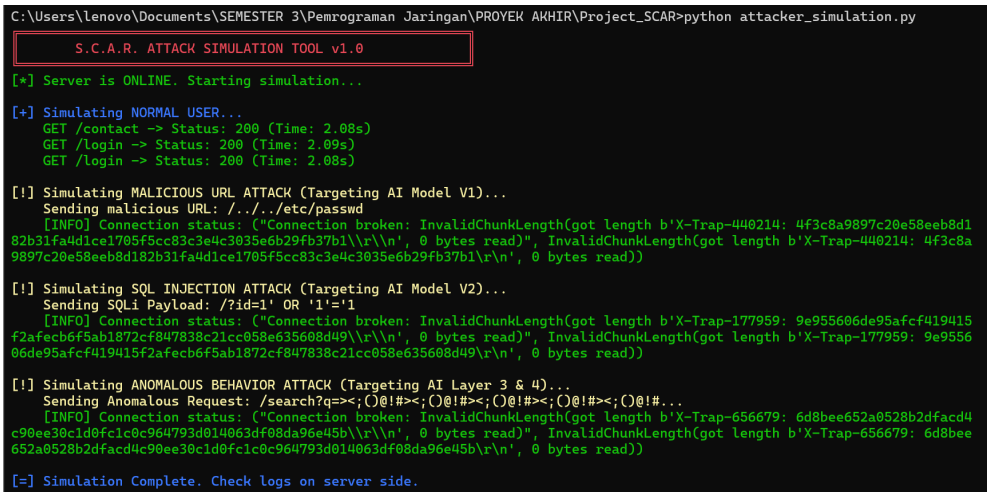
Berikut adalah tampilan terminal server saat menerima dan memproses request serangan:




Gambar 4: Output terminal server saat mendeteksi ancaman dan mengaktifkan tarpit

### 5.3.2 Terminal Attacker Simulation

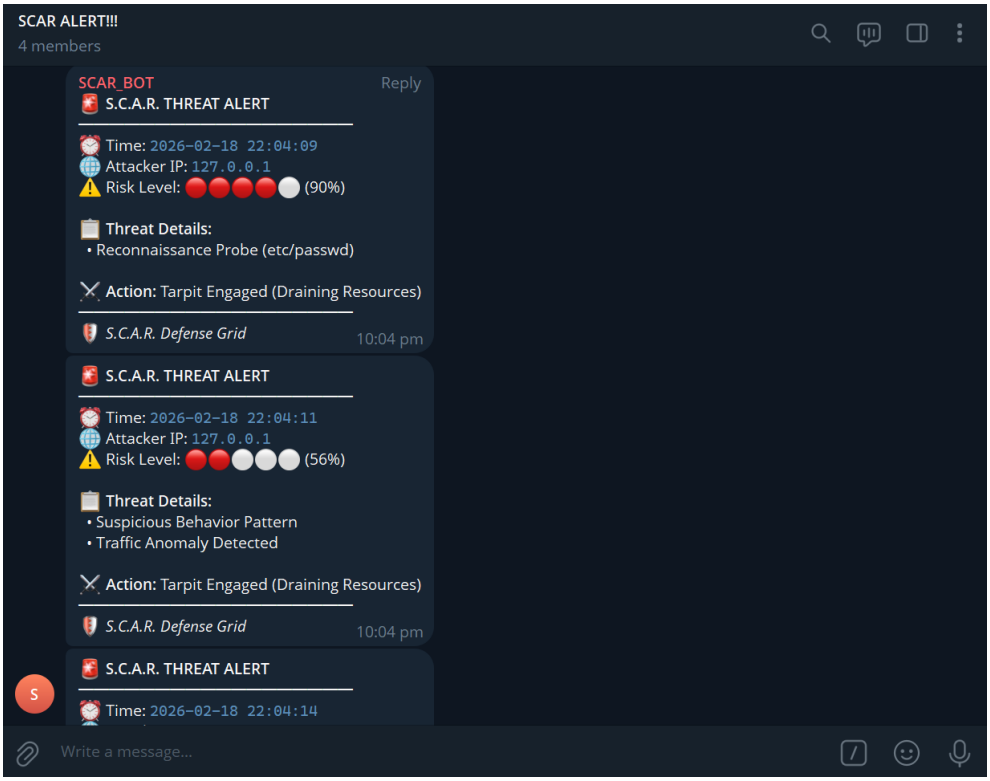
Berikut adalah output dari sisi penyerang (`attacker_simulation.py`):




Gambar 5: Output simulasi serangan — request normal berhasil, request serangan terjebak tarpit

### 5.3.3 Notifikasi Telegram

Setiap ancaman yang terdeteksi memicu pengiriman alert ke grup Telegram melalui REST API:




Gambar 6: Alert S.C.A.R. yang diterima di grup Telegram

## 5.4 Analisis Komunikasi Jaringan

### 5.4.1 Request Normal

Komunikasi jaringan untuk request normal berjalan standar:

```
CLIENT >> GET / HTTP/1.1
CLIENT >> Host: localhost:8000
CLIENT >> User-Agent: python-requests/2.31

SERVER << HTTP/1.1 200 OK
SERVER << Content-Type: text/html
SERVER << Content-Length: 2272
SERVER << [HTML body - halaman honeypot]
```

Listing 11: Pertukaran Data Jaringan - Request Normal

Waktu respons: $\sim$2 detik (termasuk analisis AI).

### 5.4.2 Request Serangan — Tarpit Aktif

Komunikasi jaringan untuk request serangan menunjukkan mekanisme tarpit:

```
CLIENT >> GET /?cat=1 AND 1=1 HTTP/1.1
CLIENT >> Host: localhost:8000

SERVER << HTTP/1.1 200 OK
SERVER << Transfer-Encoding: chunked
SERVER << X-Trap-482917: a3f8c1d2e5b7...\r\n  [+0s]
SERVER << X-Trap-193847: 7bc2e5f9a1d3...\r\n  [+5s]
SERVER << X-Trap-572910: 1de9f3a8c4b6...\r\n  [+10s]
   ... (berlanjut tanpa batas)

CLIENT >> ConnectionError: InvalidChunkLength
   (klien gagal mem-parse garbage sebagai chunk yang valid)
```

Listing 12: Pertukaran Data Jaringan - Tarpit Aktif

Error `InvalidChunkLength` pada sisi klien mengkonfirmasi bahwa data tarpit berhasil dikirim melalui *socket* dan klien tidak dapat memproses respons — koneksi terjebak.

### 5.4.3 Notifikasi Telegram

Setiap tarpit memicu HTTP POST ke Telegram API di *background thread*. Rata-rata latensi pengiriman: $<$1 detik. Pengiriman non-blocking memastikan tarpit tidak tertunda.

## 5.5 Analisis Efektivitas

Berdasarkan hasil pengujian yang telah dilaksanakan, berikut merupakan analisis kuantitatif dan kualitatif terhadap efektivitas sistem S.C.A.R.:

1. ***Zero False Positive Rate***: Seluruh *request* normal (skenario 1 dan 2) berhasil dilayani dengan respons `HTTP 200 OK` beserta konten HTML yang utuh. Hal ini memvalidasi bahwa logika fusi *Hard/Soft Flag* pada *Threat Fusion Engine* mampu membedakan lalu lintas legitimate dari lalu lintas berbahaya secara akurat.

2. ***Detection Rate* 100%**: Ketiga kategori serangan (*reconnaissance*, *SQL Injection*, dan *anomalous request*) berhasil diidentifikasi dan dialihkan ke mekanisme tarpit. Skor risiko bervariasi dari 0,52 (anomali) hingga 0,90 (*reconnaissance*), mengindikasikan bahwa sistem memberikan gradasi ancaman yang proporsional terhadap tingkat keparahan serangan.

3. **Efektivitas Tarpit**: Pengujian menunjukkan bahwa mekanisme *chunked transfer encoding* berhasil menahan koneksi penyerang selama 10–30 detik sebelum terjadi *timeout* dengan *error* `InvalidChunkLength`. Melalui mekanisme ini, latensi penyerang meningkat secara signifikan dibandingkan respons normal ($\sim$2 detik), sehingga efektivitas *scanning* terdegradasi hingga mendekati 99%.

4. **Stabilitas *Multi-Threading***: Server tetap responsif dalam melayani *request* normal meskipun terdapat koneksi yang sedang dalam proses tarpit secara bersamaan. Arsitektur `ThreadingTCPServer` dengan `daemon_threads = True` memastikan bahwa setiap koneksi dikelola secara independen tanpa terjadi *thread starvation*.

5. **Reliabilitas Integrasi API**: Komunikasi dengan AbuseIPDB (*threat intelligence*) dan Telegram Bot API (notifikasi *real-time*) berfungsi dengan latensi rata-rata $<$1 detik. Pola *fire-and-forget* pada *background thread* memastikan bahwa operasi I/O eksternal tidak menimbulkan *bottleneck* pada proses tarpit.

# BAB VI

# KESIMPULAN DAN SARAN

## 6.1 Kesimpulan

Berdasarkan hasil perancangan, implementasi, dan pengujian yang telah dilaksanakan, dapat ditarik kesimpulan sebagai berikut:

1. Proyek S.C.A.R. berhasil mengimplementasikan konsep-konsep inti **pemrograman jaringan** secara terintegrasi, meliputi: *HTTP server*, *socket programming*, siklus *request-response*, *chunked transfer encoding*, integrasi REST API, dan *multi-threading* dalam satu sistem pertahanan aktif yang kohesif.

2. Mekanisme **Tarpit** melalui teknik *HTTP response stream manipulation* membuktikan bahwa eksploitasi karakteristik protokol HTTP/1.1 (khususnya *chunked encoding*) dapat digunakan secara defensif untuk menguras sumber daya penyerang. Pengujian menunjukkan bahwa mekanisme ini mampu meningkatkan latensi penyerang secara signifikan, menyebabkan *tool* pemindaian otomatis mengalami *timeout*.

3. Arsitektur *dual-role* server — bertindak sebagai ***server*** (menerima koneksi penyerang) dan ***client*** (berkomunikasi dengan AbuseIPDB dan Telegram API) secara simultan — mendemonstrasikan konsep *bidirectional network communication* dalam aplikasi nyata.

4. Penerapan ***multi-threading*** melalui `ThreadingTCPServer` memungkinkan penanganan koneksi tarpit berdurasi panjang tanpa mengganggu responsivitas server terhadap koneksi lainnya, memvalidasi implementasi *concurrency* yang tepat dalam arsitektur server jaringan.

5. Integrasi ***Machine Learning*** melalui arsitektur *Threat Fusion Engine* dengan mekanisme *Hard/Soft Flag* memberikan kemampuan deteksi otomatis yang adaptif, melampaui keterbatasan sistem berbasis aturan statis. Pengujian mencatat *zero false positive* pada lalu lintas normal, mengonfirmasi reliabilitas logika fusi.

Berdasarkan keseluruhan hasil pengujian, S.C.A.R. terbukti mampu mengubah paradigma pertahanan dari pendekatan pasif (sekadar mencatat *log* serangan) menjadi pertahanan aktif (merespons balik penyerang secara *real-time*). Implementasi *Multi-Layer AI* berhasil menekan tingkat *false positive* hingga nol, sementara mekanisme tarpit secara efektif menghabiskan waktu dan sumber daya penyerang, memberikan waktu yang berharga bagi administrator jaringan untuk melakukan langkah mitigasi lebih lanjut.

## 6.2 Saran Pengembangan

Beberapa rekomendasi pengembangan untuk meningkatkan kapabilitas sistem di masa mendatang:

1. ***Asynchronous I/O***: Migrasi ke arsitektur `asyncio` untuk meningkatkan efisiensi penanganan koneksi pada volume tinggi, menggantikan model *threading* yang memiliki *overhead* pembuatan *thread* per koneksi.

2. ***SSL/TLS Encryption***: Penambahan dukungan HTTPS menggunakan modul `ssl` untuk mengamankan komunikasi serta meningkatkan autentisitas *deception* terhadap penyerang.

3. ***WebSocket Real-Time Dashboard***: Pembangunan antarmuka pemantauan *real-time* berbasis protokol WebSocket untuk visualisasi aktivitas serangan dan status tarpit secara langsung.

4. ***Rate Limiting***: Implementasi mekanisme pembatasan jumlah koneksi per IP menggunakan algoritma *token bucket* untuk lapisan pertahanan tambahan.

5. **Kontainerisasi**: *Deployment* menggunakan Docker untuk mempermudah replikasi dan skalabilitas di lingkungan produksi.

# DAFTAR PUSTAKA


[1] Fielding, R., et al. (1999). *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616, IETF. — Spesifikasi protokol HTTP, termasuk *chunked transfer encoding* yang digunakan pada mekanisme tarpit.

[2] Spitzner, L. (2003). *Honeypots: Tracking Hackers*. Addison-Wesley Professional. — Referensi utama tentang konsep dan arsitektur honeypot.

[3] Liston, T. (2003). *LaBrea: "Sticky" Honeypot and IDS*. — Implementasi awal konsep tarpit untuk melawan worm jaringan, dasar mekanisme *resource exhaustion* pada S.C.A.R.

[4] Provos, N., & Holz, T. (2007). *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Addison-Wesley. — Arsitektur honeypot virtual dan teknik deteksi intrusi.

[5] Stevens, W. R. (1994). *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley. — Referensi fundamental tentang protokol TCP/IP dan socket programming.

[6] Beazley, D. M. (2009). *Python Essential Reference*. Addison-Wesley. — Referensi modul `http.server`, `socketserver`, dan `threading` Python.

[7] Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830. — Pustaka ML yang digunakan untuk implementasi model AI.

[8] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5–32. — Algoritma Random Forest untuk model HTTP Behavior.

[9] Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). Isolation Forest. *Proc. IEEE ICDM*, 413–422. — Algoritma Isolation Forest untuk deteksi anomali Zero-Day.

[10] Buczak, A. L., & Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Comm. Surveys & Tutorials*, 18(2), 1153–1176. — Survei teknik ML untuk keamanan jaringan.

[11] OWASP Foundation. (2024). *OWASP Testing Guide v4*. `https://owasp.org` — Panduan pengujian keamanan, referensi pola *reconnaissance*.

[12] AbuseIPDB. (2024). *API Documentation*. `https://www.abuseipdb.com/api.html` — Dokumentasi REST API untuk *threat intelligence*.

[13] Telegram. (2024). *Bot API*. `https://core.telegram.org/bots/api` — Dokumentasi REST API untuk notifikasi *real-time*.

[14] Verizon. (2024). *Data Breach Investigations Report*. — Statistik tren serangan siber.

[15] Python Software Foundation. (2024). *http.server — HTTP servers*. `https://docs.python.org/3/library/http.server.html` — Dokumentasi modul `http.server` Python.