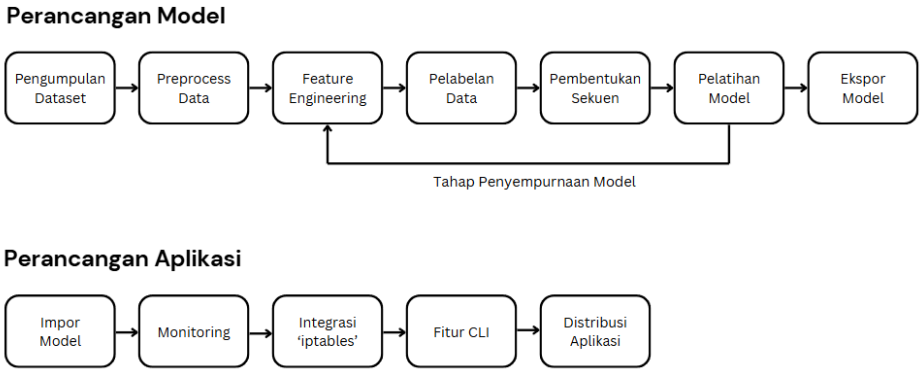
BAB III

METODE PENELITIAN

3.1 Metode Penelitian

Penelitian ini dilakukan melalui beberapa tahapan sistematis yang mengintegrasikan pendekatan eksperimental dan implementasi nyata, sesuai pada gambar 3.1. Metodologi penelitian menggunakan pendekatan metode campuran yang menggabungkan analisis dataset publik dengan implementasi dan validasi pada lingkungan mirip seperti pada PT Maro Anugrah Jaya.



Gambar 3.1 Metode Penelitian

Tahap awal penelitian dimulai dengan pengumpulan dan persiapan data, yang mencakup pengunduhan dataset dari server PT Maro Anugrah Jaya serta dataset LogHub. LogHub merupakan repositori publik yang menyediakan kumpulan dataset log sistem dari berbagai aplikasi dan layanan, yang banyak digunakan sebagai acuan penelitian untuk analisis log, deteksi anomali, dan pemodelan kegagalan sistem [31] [32]. Data mentah yang diperoleh kemudian diproses melalui serangkaian langkah pembersihan dan penyelarasan struktur agar setiap entri log memiliki format yang konsisten. Setelah melalui proses tersebut, dilakukan proses *clustering* yang menghasilkan data yang sudah berlabel. Hal ini dilakukan untuk menentukan kategori atau kelas pada setiap peristiwa yang terekam dalam log autentikasi. Berdasarkan data yang telah terlabel, dilakukan perancangan fitur untuk mengambil fitur-fitur penting dari

data mentah menjadi representasi yang relevan terhadap sebuah peristiwa yang terjadi. Fitur-fitur ini kemudian dibentuk menjadi sekuen temporal, dimana data fitur diorganisir ke dalam urutan waktu agar kompatibel dengan arsitektur *Long Short-Term Memory* (LSTM) yang memerlukan struktur masukan berbentuk rangkaian kejadian. Dengan demikian, seluruh proses memastikan data siap digunakan untuk pelatihan model berbasis LSTM secara optimal.

Pada tahap pengembangan, model dirancang menggunakan arsitektur LSTM dengan parameter yang menyesuaikan kondisi data dan pelatihan. Selama proses pelatihan digunakan strategi pelatihan *train-validation-test split*. Model akan memiliki tahapan validasi data dan juga pengujian skor akhir dari hasil pelatihan, seperti akurasi, presisi, *recall*, *F1-score*, dan *false positive rate*. Penulis melakukan penyempurnaan terhadap model dengan mengubah parameter dalam pelatihan sehingga dapat meningkatkan skor performa model.

Model yang sudah dilatih akan disimpan dalam bentuk *file*. Model ini akan dikemas menjadi sebuah aplikasi berbasis *Command-Line Interface* (CLI). Pengemasan model dilakukan sehingga model bisa dijalankan dalam sebuah sistem operasi untuk dilakukan pengujian terhadap lingkungan produksi.

Pada tahap ini, model sudah terintegrasi dengan aplikasi dan aplikasi akan diterbitkan kedalam repositori GitHub. GitHub adalah platform Git berbasis web untuk pengelolaan versi dan kolaborasi pengembangan perangkat lunak yang memanfaatkan sistem kontrol [33]. Hal ini dilakukan sehingga aplikasi dapat diunduh dan dipasang langsung ke dalam target sistem yang diperlukan.

Pada tahap pengujian aplikasi, dilakukan mekanisme *log replay* dengan memutar ulang data log autentikasi historis yang telah dikumpulkan. Pendekatan ini bertujuan untuk mensimulasikan kondisi operasional sistem secara realistis tanpa bergantung pada serangan langsung di lingkungan produksi. Melalui *log replay*, aplikasi dapat diuji secara konsisten terhadap rangkaian peristiwa autentikasi yang sama, sehingga memungkinkan evaluasi yang adil dan terkontrol. Dalam skenario ini, performa aplikasi berbasis LSTM dibandingkan secara langsung dengan sistem deteksi berbasis aturan Fail2Ban dengan menggunakan log yang identik. Pendekatan ini menilai kemampuan generalisasi model serta mengevaluasi efektivitas aplikasi dalam mendeteksi

serangan SSH *slow-rate* yang tidak teridentifikasi oleh mekanisme deteksi berbasis ambang batas.

3.2 Perancangan Model

3.2.1 Pengumpulan Dataset

Pengumpulan dataset merupakan tahap awal dalam penelitian ini. Berdasarkan riset Marta Catillo, et al. pada penelitian mengenai rendahnya transferabilitas model pembelajaran mesin yang dilatih hanya pada dataset publik [15], penelitian ini tidak hanya mengandalkan satu sumber data. Studi tersebut menunjukkan bahwa performa model yang tinggi pada dataset publik sering kali tidak dapat direplikasi ketika model diterapkan pada lingkungan nyata karena perbedaan karakteristik data.

Oleh karena itu, penelitian ini menggunakan dua sumber dataset utama dengan karakteristik yang berbeda, yaitu dataset log autentikasi dari lingkungan produksi nyata PT Maro Anugrah Jaya dan dataset publik yang diambil dari LogHub.

Pengumpulan dataset dilakukan melalui beberapa pendekatan yang berbeda, seperti berikut:

1. Log Autentikasi Proxmox PT Maro Anugrah Jaya
    a. Antarmuka web Proxmox perusahaan PT Maro Anugrah Jaya diakses.
    b. Pada *shell* server, navigasi ke direktori sistem log dengan menjalankan perintah `cd /var/log` untuk mengakses lokasi penyimpanan berkas log autentikasi sistem.
    c. Ekstraksi data log per-bulan dilakukan menggunakan perintah grep untuk menerapkan filter entri log berdasarkan bulan tertentu. Untuk masing-masing bulan, digunakan perintah `grep "bulan" auth.log > auth_bulan.log`.
    d. Pengunduhan berkas log yang telah di ekstraksi dilakukan menggunakan *Secure Copy Protocol* (SCP) dari terminal lokal dengan perintah `scp user@proxmox_server:/var/log/`

auth.log /local/path. Perintah ini akan dilakukan pada tiap-tiap berkas yang berhasil diekstraksi.

e. Berkas log autentikasi yang telah diunduh disimpan dalam direktori lokal.

2. Log Autentikasi LogHub

a. Repositori diakses menggunakan peramban melalui tautan berikut: https://github.com/logpai/loghub.

b. Berkas log yang sudah disediakan oleh penulis bernama OpenSSH kemudian diunduh.

c. Berkas log autentikasi disimpan dalam penyimpanan lokal untuk tahap selanjutnya.

3.2.2 Prapemrosesan Data

Prapemrosesan data dimulai setelah mengumpulkan berkas masing-masing log autentikasi SSH baik dari PT Maro Anugrah Jaya maupun LogHub. Proses ini mengubah log mentah menjadi format terstruktur yang siap untuk dianalisis dan dilakukan pelatihan model. Pemrosesan ini terdiri dari beberapa tahapan.

1. *Parsing* Log menggunakan *Regular Expression* (Regex)

   Regex adalah pola teks yang digunakan untuk mencari, mencocokkan, atau memanipulasi *string* secara efisien. Regex memungkinkan pendefinisian aturan pencarian yang fleksibel, seperti menemukan karakter, kata, atau pola tertentu dalam teks.

```
header_re = re.compile(
    r'^(?P<mon>[A-Z][a-z]{2})\s+(?P<day>\d{1,2})\s+(?P<time>\d{2}:\d{2}:\d{2})\s+'
    r'(?P<host>\S+)\s+(?P<proc>\w+)\[(?P<pid>\d+)\]:\s+(?P<msg>.*)$'
)
```

   Melalui format regex berikut, berkas log akan selalu mengadung: bulan, tanggal, waktu, nama *host*, alamat IP, nama proses, ID proses, dan pesan dalam suatu peristiwa

2. Klasifikasi Tipe Peristiwa dan Ekstraksi Kolom

Setelah *header* pada log autentikasi berhasil di *parse*, pesan log perlu diklasifikasikan menjadi berbagai tipe peristiwa autentikasi. Proses ini akan dilakukan menggunakan fungsi *classify_message()* yang menggunakan berbagai pola regex yang lain untuk mencocokkan dan mengkstrak informasi dari pesan log.

Pola regex yang digunakan mencakup berbagai tipe peristiwa autentikasi, seperti:

a. *pat_failed_password*: Mengekstrak peristiwa gagal autentikasi kata sandi beserta *username*, *source IP*, dan *port*.
b. *pat_invalid_user*: Mengekstrak peristiwa percobaan masuk dengan *username* yang tidak valid.
c. *pat_pam_auth_failure*: Mengekstrak peristiwa kegagalan autentikasi PAM.
d. *pat_conn_closed*: Mengekstrak peristiwa penutupan koneksi saat login.
e. *pat_recv_disconnect*: Mengekstrak perisitwa pemutusan koneksi yang diterima.

Setiap pola regex akan mencocokkan pesan log dan mengekstrak kolom-kolom seperti *username*, alamat IP asal, *port* asal, dan keterangan lainnya sesuai dengan tipe peristiwa yang terdeteksi. Hasil klasifikasi ini akan menghasilkan kolom baru *event_type* yang mengidentifikasi peristiwa autentikasi.

3. Konversi Tipe Data

Setelah melakukan *parsing*, data mentah akan dikonversi ke tipe data yang sesuai:

a. *Timestamp*: Dikonversi menjadi format ‘datetime64[ns]’ untuk analisa temporal dan suatu waktu peristiwa.
b. *String fields*: Nama *host*, alamat IP, *username*, tipe peristiwa, akan dikonversikan menjadi tipe *string*.
c. *Integer fields*: ID proses, *port* asal akan diubah menjadi tipe ‘Int64’.

d. *Boolean fields*: ‘is_invalid_user’ akan dikonversikan menjadi *boolean*.

4. Integrasi Data dari seluruh Dataset

   Data dari kedua sumber, baik dari PT Maro Anugrah Jaya maupun LogHub akan digabungkan dengan melakukan standarisasi skema terlebih dahulu. Hal ini dilakukan supaya tidak ada kolom lain yang masuk yang dapat menimbulkan nilai yang kosong atau nilai yang bertabrakan. Standarisasi skema meliputi:

   a. Nilai pada ‘timestamp’, ‘host’, ‘process’, ‘pid’, ‘event_type’, ‘session_key’.
   b. Nilai pada ‘username’, ‘is_invalid_user’, ‘source_ip’, ‘source_host’, ‘source_port’.
   c. Nilai pada ‘protocol’, ‘detail’, ‘repeat_count’, ‘raw_message’, ‘payload’.
   d. Terakhir menempatkan kolom baru, ‘dataset_source’, untuk mencantumkan asal data.

   Data kemudian digabungkan menggunakan perintah `pd.concat()` dengan konfigurasi `ignore_index=True` untuk membuat index menjadi berurutan.

5. Pembersihan Data

   Setelah digabungkan, maka data akan divalidasi sekali kali untuk memastikan semua peristiwa autentikasi berhasil diurutkan dengan benar, dan tidak ada kolom peristiwa yang mengandung nilai yang kosong atau IP dengan nilai ‘0.0.0.0’ karena umumnya IP tersebut tidak ada.

6. Penyimpanan Data ke Format *Parquet*.

   Data yang telah diproses disimpan dalam format Parquet menggunakan *library* PyArrow. Format Parquet dipilih karena:

   a. Menghasilkan kompresi yang lebih baik dibandingkan format ‘.csv’ [34].

b. Mempertahankan tipe data yang sudah dikonversi beserta struktur datanya [34].
c. Lebih cepat untuk operasi *filtering* dan agregasi [34].

Data disimpan ke penyimpanan lokal yang akan digunakan untuk tahap pelabelan melalui metode *clustering*.

3.2.3 Perancangan Fitur

Perancangan fitur dilakukan setelah data selesai diproses dan sebelum tahap pelabelan. Tahap ini mengubah data pada level peristiwa menjadi data agregat yang siap untuk dianalisa dan dilakukan pelabelan menggunakan metode *clustering*.

Proses perancangan fitur terdiri dari beberapa tahapan.

1. Agregasi Data per-Jendela Waktu

   Data peristiwa autentikasi yang telah diproses diagregasi berdasarkan sumber alamat IP dan jendela waktu per jam (*hourly window*). Agregasi dilakukan dengan mengelompokkan semua peristiwa autentikasi yang berasal dari alamat IP yang sama dalam rentang waktu satu jam. Jendela waktu yang dibuat menggunakan fungsi dt.floor("1H") yang membulatkan *timestamp* ke bawah ke batas jam terekat.

   Penggunaan jendela waktu dalam agregasi data diperlukan untuk mengurangi dimensionalitas data pada level peristiwa yang memiliki volume besar, sambil tetap mempertahankan informasi temporal yang relevan untuk analisis pola aktivitas. Menurut M. Pourbafrani et al., agregasi per jendela waktu mengubah data pada level peristiwa menjadi deret waktu yang dapat dianalisa menggunakan teknik pembelajaran mesin yang memerlukan input yang terstruktur, seperti *clustering* dan *sequence-based model* (LSTM) [35].

   Pemilihan rentang jendela waktu 1 jam didasarkan pada keseimbangan antara granularitas temporal dan stabilitas statistik.

Jendela yang terlalu pendek dapat menghasilkan fitur yang terlalu bervariasi dan tidak stabil, sementara jendela yang terlalu panjang dapat mengaburkan pola temporal yang penting. Riset yang dilakukan oleh Lee et al., menghasilkan analisa pola serangan SSH *brute force* cenderung terjadi dalam periode yang relatif singkat namun dapat berlangsung selama beberapa jam [36]. Jendela waktu 1 jam cukup untuk menangkap episode serangan sambil mempertahankan kemampuan antara episode lainnya.

2. Ekstraksi Fitur Statistik

   Fitur-fitur yang diekstrak dapat dilihat pada tabel berikut:

Tabel 3.1 Fitur Statistik Dasar

Nama Fitur	**Deskripsi**
n_failed_password	Jumlah percobaan autentikasi yang gagal.
n_distinct_users	Jumlah *username* berbeda yang digunakan.
avg_time_between_attempts	Rata-rata waktu antar percobaan autentikasi.
num_failed_ports	Jumlah sumber *port* yang berbeda untuk percobaan gagal.
success_ratio	Rasio sesi autentikasi berhasil terhadap total percobaan.
login_interval_variance	Varians interval waktu antar percobaan autentikasi.
username_entropy	Entropi distribusi *username* yang digunakan.
time_of_day_avg	Rata-rata waktu dalam hari aktivitas autentikasi.
num_failed_days	Jumlah hari berbeda dengan percobaan autentikasi gagal.

Berikut merupakan penjelasan dari fitur dasar yang diekstraksi:

a. Jumlah Kegagalan Kata Sandi

   Fitur ini mengukur jumlah percobaan yang gagal dalam satu jendela untuk setiap alamat IP. Nilai yang tinggi mengindikasikan aktivitas mencurigakan seperti serangan *brute force*.

$$n_{failed_{password}} = \sum_{i=1}^{N} 1[event_i = \text{failed\_password}]$$

Perhitungan ini dilakukan dengan melakukan *filtering* terhadap peristiwa yang memiliki event_type == "failed_password" dalam setiap jendela waktu, kemudian menghitung jumlahnya menggunakan fungsi len().

b. Jumlah *Username* Berbeda

Fitur ini mengukur jumlah *username* yang berbeda yang digunakan dalam percobaan autentikasi dalam satu jendela waktu. Nilai yang tinggi mengindikasikan serangan yang mencoba berbagai kombinasi *username*, sedangkan nilai yang rendah mengindikasikan serangan yang fokus terhadap *username* tertentu.

$$n_{distinct\_users} = \left|u_j : u_j \in usernames, j = 1, \dots, N\right|$$

Perhitungan ini dilakukan menggunakan fungsi $nunique()$ pada kolom *username* setelah mem-filter peristiwa berdasarkan alamat IP dan jendela waktu.

c. Rata-Rata Waktu Antar Percobaan

Fitur ini mengukur rata-rata waktu (dalam detik) antara dua percobaan autentikasi berturut-turut dalam satu jendela waktu. Nilai yang rendah mengindikasikan serangan cepat sedangkan nilai yang tinggi mengindikasikan serangan lambat atau dalam artian waktunya lebih tersebar/acak.

$$avg\_time\_between\_attemps = \frac{1}{N-1}\sum_{i=1}^{N-1}(t_{i+1} - t_i)$$

Dihitung dengan melakukan konversi *timestamp* ke dalam *integer*, lalu menghitung selisih waktu berturut-turut menggunakan fungsi $np.diff()$, kemudian mengambil nilai rata-rata menggunakan $np.mean()$.

d. Jumlah Sumber *Port* yang Berbeda

Fitur ini mengukur jumlah sumber *port* yang berbeda, yang digunakan dalam perocbaan autentikasi yang gagal. Variasi port

yang tinggi mengindikasikan upaya untuk menghindari deteksi atau penggunaan deteksi *port scanning*.

$$num\_failed\_ports = \left|\{p_j : p_j \in src_{port}, event_j = "failed\_password"\}\right|$$

Perhitungan fitur ini dilakukan pada setiap peristiwa yang memiliki $event\_type == "failed\_password"$, kemudian mehgitung jumlah *port* unik menggunakan $nunique()$ pada kolom 'src_port'.

e. Rasio Kesuksesan

Fitur ini mengukur rasio antara jumlah sesi autentikasi yang berhasil terhadap total percobaan autentikasi (berhasil + gagal). Nilai yang tinggi mengindikasikan aktivitas normal (*benign*), sedangkan nilai yang rendah (mendekati 0) mengindikasikan aktivitas mencurigakan yang hanya terdiri dari percobaan yang gagal.

$$success\_ratio = \frac{n_{accepted}}{n_{failed} + n_{accepted}}$$

Fitur ini dihitung dengan melakukan *filtering* pada setiap peristiwa yang mengandung "*accepted"* untuk mendapatkan *accepted_sessions*, kemudian membagianya dengan total percobaan.

f. Varians Interval Waktu Antar Autentikasi

Fitur ini mengukur varians dari interval waktu antar percobaan autentikasi. Nilai varians yang tinggi mengindikasikan pola waktu yang tidak konsisten, sedangkan varians rendah mengindikasikan pola waktu yang teratur dan konsisten.

$$login\_itvl\_variance = \frac{1}{N-1}\sum_{i=1}^{N-1}((t_{1+1} - t_i) - \bar{\Delta}t)^2$$

Fitur ini dihitung dengan mencari selisih waktu berturut-turut menggunakan $np.diff()$, kemudian menghitung varians menggunakan $np.var()$.

g. Entropi *Username*

Fitur ini mengukur nilai entropi dari distribusi *username* yang digunakan dalam percobaan autentikasi. Nilai entropi yang tinggi mengindikasikan variasi *username* yang besar atau dapat disebut sebagai serangan yang terdistribusi, sedangkan entropi yang rendah mengindikasikan penggunaan *username* yang terbatas atau serangan lebih fokus ke satu *username* saja.

$$H(U) = -\sum_{u \in U} p(u) \log_2 p(u)$$

Fitur ini dihitung menggunakan fungsi $compute\_entropy()$ yang menerapkan rumus Shannon entropy pada daftar *username* setelah menghilangkan nilai *null*.

h. Rata-Rata Waktu dalam Hari Terhadap Aktivitas Autentikasi

Fitur ini digunakan untuk menangkap pola temporal aktivitas autentikasi yang tidak dapat direpresentasikan hanya dalam jumlah peristiwa. Fitur ini dapat memberikan informasi kapan aktivitas tersebut terjadi dalam satu hari.

$$time\_of\_day\_avg = \frac{1}{N} \sum_{i=1}^{N} (t_i \ mod \ 86400)$$

Penghitungan fitur ini dilakukan dengan pertama-tama melakukan konversi *timestamp* ke detik semenjak *epoch*, lalu mengambil modulo sebesar 86400 detik atau 24 jam untuk mendapatkan waktu dalam hari, dan menghitung rata-ratanya menggunakan $np.mean()$.

i. Jumlah Kegagalan dalam Hari

Fitur ini mengukur jumlah hari berbeda dimana alamat IP tersebut melakukan percobaan autentikasi yang gagal. Nilai yang tinggi dari fitur ini mengindikasikan serangan yang

persisten dan berlangsung dalam jangka waktu yang lama, sedangkan nilai yang rendah mengidikasikan serangan jangka pendek. Fitur ini dihitung dengan melakukan filtering peristiwa yang memiliki $event\_type == "failed\_password"$, lalu mengelompokkannya berdasarkan alamat IP, kemudian menghitung jumlah hari unik menggunakan fungsi $nunique()$ pada kolom tanggal.

3. Normalisasi dan Standarisasi Fitur

   Setelah semua fitur diekstrak, dilakukan normalisasi nilai-nilai yang hilang (*missing values*) dengan mengisi nilai 0 untuk fitur-fitur numerik. Selanjutnya fitur-fitur tersebut akan distandarisasi menggunakan StandardScaler dari *library* scikit-learn. Standarisasi dilakukan untuk mengubah setiap fitur sehingga memiliki nilai mean 0 dan nilai standar deviasi 1. Proses standarisasi memastikan semua fitur memiliki skala yang sama sehingga tidak ada fitur yang mendominasi.

3.2.4 Pelabelan Data

Pada penelitian ini, proses pelabelan data tidak dapat dilakukan secara manual karena tidak tersedia label kebenaran (*ground truth*) yang secara eksplisit membedakan antara aktivitas normal dan aktivitas serangan. Oleh karena itu, diperlukan pendekatan otomatis yang mampu menemukan pola perilaku serangan langsung dari data. Pendekatan *clustering* dipilih untuk tujuan ini, karena memungkinkan pengelompokan data berdasarkan kemiripan perilaku tanpa memerlukan label awal.

Pelabelan data dilakukan secara *unsupervised*, menggunakan algoritma *clustering Density-Based Spatial Clustering of Applications with Noise* (DBSCAN). Algoritma ini akan mengelompokkan aktivitas yang sering muncul bersama dalam pola yang mirip, dan mengabaikan aktivitas yang muncul sangat jarang atau menyimpang [37]. Proses ini

menghasilkan label semu (*psuedo-labels*) yang berupa *cluster* perilaku yang menjadi dasar pembentukan kelas serangan.

Pelabelan dilakukan pada data yang telah diagregasi per alamat IP dan per jendela waktu satu jam serta diperkaya dengan fitur-fitur yang sudah dirancang sebelumnya. Tahap awal merupakan pemisahan jendela-jendela yang dianggap melakukan perilaku aktivitas normal (*benign*) berdasarkan keberadaan sesi autentikasi yang berhasil (*accepted_sessions*). Selanjutnya, DBSCAN diterapkan pada subset jendela yang terindikasi serangan (jendela yang berisikan kegagalan autentikasi yang banyak). Pendekatan ini memungkinkan algoritma DBSCAN fokus untuk mempelajari struktur pola sernagan tanpa tercampur oleh volume aktivitas normal.

Berikut merupakan tahapan yang diperoleh untuk melabel data berdasarkan hasil *clustering* dari DBSCAN.

1. Penyusunan Dataset

   Data yang berisikan fitur-fitur yang sudah dirancang disimpan sementara menjadi sebuah DataFrame. Berdasarkan fitur 'accepted_sessions' dan kombinasi jumlah percobaan gagal serta jumlah *username* yang dicoba, dibentuk dua buah label heuristik:

   a. Data dengan $is\_benign = 1$ atau jendela yang memiliki setidaknya satu sesi autentikasi yang berhasil.
   b. Data dengan $is\_attack = 1$ untuk jendela yang tidak memiliki sesi berhasil tetapi menunjukkan aktivitas gagalnya autentikasi atau variasi percobaan *username* yang signifikan.

   Dari pengelompokkan ini, disusun dua kelompok data yang akan dipisahkan. Data dengan jendela aktivitas normal (*benign*) tidak akan melalui proses DBSCAN, sementara data dengan akvitas serangan akan melalui *clustering* DBSCAN. Pemisahan dilakukan agar DBSCAN fokus mempelajari variasi pola serangan tanpa terpengaruh oleh aktivitas normal.

2. Pencarian Parameter DBSCAN

DBSCAN memiliki dua parameter utama, yaitu eps (radius lingkungan) dan min_samples (jumlah minimum titik dalam radius tersebut untuk membentuk suatu *cluster*). Pemilihan nilai kedua parameter ini sangat berpengaruh terhadap hasil *clustering*, karena nilai yang terlalu kecil dapat menyebabkan sebagian besar data dianggap sebagai *noise*, sedangkan nilai yang terlalu besar dapat menggabungkan pola perilaku yang berbeda ke dalam satu *cluster*. Oleh karena itu, dilakukan pencarian parameter menggunakan pendekatan *grid search* sederhana dengan mencoba beberapa kombinasi nilai parameter sebagai berikut:

$$eps \in \{0.2, 0.3, 0.4, 0.5\}$$
$$min\_samples \in \{5, 7, 10, 15\}$$

Untuk setiap kombinasi parameter, algoritma DBSCAN dijalankan pada data yang telah distandarisasi. Hasil *clustering* kemudian dievaluasi berdasarkan beberapa indikator, yaitu jumlah *cluster* yang terbentuk (tidak termasuk *cluster* dengan label −1), jumlah dan proporsi titik yang diklasifikasikan sebagai *noise*, serta nilai *silhouette score* yang dihitung hanya pada titik-titik yang termasuk dalam *cluster* (*non-noise*).

Suatu kombinasi parameter dianggap layak apabila mampu menghasilkan setidaknya tiga *cluster* yang berbeda, sehingga menunjukkan adanya variasi pola perilaku serangan, serta memiliki proporsi *noise* yang tidak terlalu besar agar sebagian besar data tetap dapat dianalisis. Dari kombinasi parameter yang memenuhi kriteria tersebut, dipilih parameter dengan nilai *silhouette score* tertinggi sebagai parameter DBSCAN terbaik. Pemilihan ini didasarkan pada asumsi bahwa nilai *silhouette score* yang lebih tinggi menunjukkan pemisahan *cluster* yang lebih jelas dan konsisten, sebagaimana direkomendasikan dalam evaluasi kualitas *clustering* pada data keamanan oleh Bhuyan et al. [38]

3. Penerapan DBSCAN dan Pemberian Label

Setelah parameter terbaik diperoleh, DBSCAN diterapkan secara penuh pada subset yang memiliki data serangan untuk menghasilkan label *cluster* bagi setiap jendela kandidat serangan. Hasil label DBSCAN kemudian diproyeksikan kembali kedalam DataFrame utama melalui indeks jendela yang bersesuaian. Jendela dengan aktivitas normal akan diberi nilai 0 sebagai representasi kelas *benign*. Lalu jendela dengan kandidat serangan yang termasuk dalam *cluster* DBSCAN diberi label bernilai positif dan digeser dengan *offset* seperti 10, 11, 12, … agar mudah dibedakan dari kelas *benign*. Jendela yang diberi label -1 oleh DBSCAN dipertahankan sebagai *noise* atau pola yang terlalu jarang dan tidak membentuk *cluster* yang padat.

4. Penyimpanan Hasil Pelabelan

   Setelah seluruh jendela memperoleh *cluster*, kolom hasil *cluster* disimpan bersama fitur-fitur agregat yang sudah dibentuk dalam berkas. Berkas ini akan menjadi basis bagi tahapan selanjutnya untuk diberlakukannya karakterisasi semantik setiap *cluster*.

3.2.5 Persiapan Data

Tahapan persiapan data untuk pelatihan bertujuan mengubah hasil pelabelan berbasis *clustering* menjadi bentuk dataset yang sesuai untuk melatih model pembelajaran mendalam berbasis deret waktu. Data yang semula direpresentasikan sebagai jendela waktu independen disusun ulang agar memiliki struktur temporal, label kelas yang konsisten, serta format numerik yang kompatibel dengan model *Long Short-Term Memory* (LSTM). Proses ini menghasilkan data latih dan data uji yang akan digunakan pada tahap pelatihan model.

1. Seleksi dan Pemetaan Label

   Langkah pertama yang dilakukan untuk seleksi data dari hasil *clustering* adalah membuang label *noise* atau cluster dengan label '-1'. Hanya jendela waktu yang tergolong ke dalam *cluster* yang valid yang digunakan untuk tahap pelatihan.

Selanjutnya label *cluster* numerik dipetakan ke dalam label tingkat tinggi yang lebih semantik, hal ini mencakup cluster 0 dipetakan sebagai “BENIGN”, sedangkan *cluster* lainnnya yang merepresentasikan sebagai pola serangan akan dibandingkan berdasarkan karakteristik temporal dari masing-masing *cluster* untuk menjadi “FAST_ATTACK” dan “SLOW_ATTACK”. Label teks “BENIGN”, “FAST_ATTACK”, dan “SLOW_ATTACK”, akan dikodekan menjadi label numerik menggunakan LabelEncoder untuk keperluan input ke model.

2. Pembentukan Urutan Sekuen per-IP

   Karena model LSTM adalah model yang berbasis deret waktu, data jendela tunggal tidak langsung digunakan sebagai input model. Oleh karena itu, jendela waktu disusun terlebih dahulu untuk menjadi urutan sekuen beberapa jendela berturut-turut untuk setiap alamat IP. Untuk setiap alamat IP sumber, seluruh jendela waktu diurutkan secara kronologis.

   Sekuen dibentuk dengan panjang tetap yang ditetapkan sebagai salah satu *hyperparameter* model. Penelitian ini menggunakan panjang sekuen sebesar tiga jendela waktu berturut-turut, sehingga setiap contoh pelatihan merepresentasikan aktivitas satu alamat IP dalam rentang waktu beberapa jam terakhir. Label untuk setiap sekuen ditentukan berdasarkan label jendela waktu terakhir dalam sekuen tersebut.

   Pemilihan panjang sekuen didasarkan pada pertimbangan keseimbangan antara kemampuan model dalam menangkap konteks temporal dan keterbatasan jumlah data yang tersedia. Evaluasi lebih lanjut terhadap pengaruh panjang sekuen terhadap kinerja model dibahas pada bab selanjutnya.

3. Pemisahan Data Latih dan Uji Berbasis IP

Setelah sekuen terbentuk, data dipisahkan menjadi data latih dan data uji. Pemisahan ini dilakukan pada tingkat alamat IP, bukan pada tingkat sekuen, untuk mencegah kebocoran data (*data leakage).* Seluruh alamat IP sumber dibagi ke dalam dua kelompok, yaitu IP untuk data latih dan IP untuk data uji, menggunakan metode pembagian acak dengan proporsi tertentu.

Sekuen dimasukkan ke dalam data latih apabila seluruh jendela waktu dalam sekuen tersebut berasal dari alamat IP yang termasuk dalam kelompok data latih, dan sebaliknya untuk data uji. Melalui pemisahan ini, model diuji menggunakan alamat IP yang tidak pernah muncul pada tahap pelatihan, sehingga evaluasi yang dilakukan pada tahap selanjutnya lebih merefleksikan kemampuan generalisasi model terhadap entitas yang belum pernah dilihat sebelumnya.

4. Penyeimbangan Kelas pada Data Latih

   Apabila distribusi kelas pada data latih menunjukkan ketidakseimbangan, dimana kelas *fast attack* memiliki jumlah sekuen yang jauh lebih besar dibandingkan dengan kelas *benign* maupun *slow attack*, dilakukan penyeimbangan kelas pada data latih.

   Metode penyeimbangan yang dilakukan merupakan metode *undersampling*. Pendekatan ini dipilih karena bersifat sederhana dan tidak menambahkan data sintetis atau duplikasi sekuen yang berpotensi mempengaruhi pola temporal pada data deret waktu.

5. Normalisasi Fitur dan Pembentukan Label *One-Hot*

   Sebelum digunakan sebagai input ke model LSTM, fitur pada data latih dan data uji dinormalisasi untuk memastikan seluruh fitur berada pada skala yang sebanding. Pada tahap perancangan fitur, standarisasi digunakan untuk memastikan skala fitur sebanding. Sementara itu, pada tahap persiapan data untuk pelatihan,

normalisasi dilakukan kembali dengan *scaler* yang dilatih hanya menggunakan data latih, sehingga distribusi fitur sesuai dengan kebutuhan proses optimisasi dan untuk mencegah terjadinya kebocoran informasi dari data uji ke data latih.

Karena *scaler* bekerja pada data dua dimensi, data deret waktu terlebih dahulu diratakan sehingga setiap baris merepresentasikan satu jendela waktu. Setelah proses normalisasi selesai, data dibentuk kembali ke format deret waktu agar dapat digunakan sebagai input model LSTM, tanpa mengubah urutan temporal antar jendela waktu.

Selain normalisasi fitur, label kelas ditransformasikan agar dapat dipahami oleh model klasifikasi multikelas. Label numerik dikonversi ke dalam bentuk *one-hot encoding*, di mana setiap kelas direpresentasikan sebagai vektor biner. Representasi ini memungkinkan model mempelajari probabilitas keanggotaan setiap kelas tanpa mengasumsikan adanya hubungan numerik atau urutan antar kelas.

3.2.6 Pelatihan Model

Pelatihan model difokuskan pada arsitektur LSTM untuk mempelajari pola temporal dari fitur-fitur yang sudah dirancang sebelumnya. Bagian ini mencakup konfigurasi model, strategi optimasi model, serta mekanisme pengendalian *overfitting*.

1. Arsitektur LSTM

   Arsitektur LSTM dirancang untuk menangkap dependensi temporal antar fitur yang telah direpresentasikan sebagai urutan (sekuen). Model dibangun menggunakan struktur sekuen pada Keras, yang memungkinkan penyusunan lapisan secara linear dari input hingga output. Struktur final model adalah sebagai berikut:

$$Input\ (3 \times 10) \rightarrow LSTM(32) \rightarrow Dropout(0.3)$$
$$\rightarrow Dense(16, ReLU) \rightarrow Dropout(0.2)$$
$$\rightarrow Dense(3, Softmax)$$

a. Lapisan LSTM sebanyak 32 unit berfungsi untuk mempelajari pola ketergantungan temporal dan dinamika antar fitur yang diperlakukan sebagai sekuen sepanjang 10 *timestep*.
b. *Dropout* menghilangkan neuron secara acak selama pelatihan untuk mencegah *overfitting* dan meningkatan kemampuan generalisasi model.
c. Lapisan *dense* merupakan lapisan yang terhubung sepenuhnya terhadap seluruh neuron pada lapisan sebelumnya. Lapisan ini melakukan transformasi linear yang diikuti oleh aktivasi non-linear ReLU. Lapisan ini berfungsi untuk mengintegrasikan informasi dari representasi temporal yang dihasilkan oleh model.
d. Lapisan *output* merupakan lapisan yang terhubung sepenuhnya kepada neuron lainnya. Lapisan ini akan memiliki 3 unit, masing-masing mewakili kelas *fast attack*, *slow attack*, dan *benign*. Lapisan ini menghasilkan probabilitas ternormalisasi, sehingga setiap nilai berada pada rentang 0 dan 1 dan totalnya sama dengan 1.

2. Konfigurasi Pelatihan

Selama masa pelatihan, model akan memiliki *hyperparameter* sebagai berikut:

Tabel 3.2 *Hyperparameter* Model

Parameter	**Nilai**
Optimizer	Adam
Learning Rate	0.001
Loss Function	Categorical Crossentropy
Metrics	Accuracy
Batch Size	32
Epoch (Max)	150

Optimasi model deteksi anomali berbasis LSTM dilakukan menggunakan Adam optimizer karena kemampuannya untuk menyesuaikan laju pembelajaran secara adaptif berdasarkan estimasi gradien pertama dan kedua, sehingga memberikan

konvergensi yang stabil selama pelatihan jaringan saraf dalam tugas klasifikasi dan deteksi intrusi. Dalam studi LSTM yang dilakukan oleh Ogunseyi dan Thiyagarajan mengatur konfigurasi pelatihan jaringan menggunakan Adam dengan *learning rate* sebesar 0,001 sebagai nilai *default* untuk menjaga keseimbangan antara kecepatan konvergensi dan kestabilan pembaruan bobot selama latihan model [39]. Lebih lanjut, fungsi kerugian *categorical crossentropy* digunakan sesuai dengan tugas klasifikasi multi-kelas, dimana model LSTM mempelajari distribusi probabilitas antar kelas serangan dan kelas normal dari data jaringan.

Pada tahap evaluasi, model menggunakan akurasi sebagia metrik untuk mengukur tingkat ketepatan prediksi. Selama pelatihan, data diproses dalam kelompok dengan ukuran sebesar 32 (batch_size), ukuran tersebut memberikan keseimbangan antara efisiensi komputasi dan kestabilan gradien. Jumlah iterasi maksimum pelatihan ditetapkan sebanyak 150 *epoch*, memungkinkan model belajar secara bertahap hingga mencapai performa terbaik tanap resiko mengalami *overfitting* berlebihan.

3. *Callback* Pelatihan

   Untuk meningkatkan kemampuan generalisasi model dan mencegah *overfitting* proses pelatihan dilengkapi dengan dua *callback*, yaitu ‘EarlyStopping’ dan ‘ReduceLROnPlateau’.

   a. *Callback* ‘EarlyStopping’

Tabel 3.3 Parameter *Callback* EarlyStopping

Parameter	Nilai
Monitor	val_loss
Patience	20 epoch
Restore Best Weights	True

Fungsi ini digunakan untuk memantau nilai *validation loss* selama pelatihan. Mekanisme ini memiliki *patience* sebesar 20 *epoch*, yang berarti pelatihan akan dihentikan secara otomatis apabila *validation loss* tidak menunjukkan perbaikan dalam 20

iterasi berturut-turut. Selain itu, parameter $restore\_best\_weights = True$ memastikan bahwa bobot model yang digunakan setelah pelatihan adalah bobot terbaik yang dicapai selama proses pelatihan, bukan bobot pada *epoch* terakhir. Pendekatan ini membantu menjaga performa model pada data yang tidak terlihat (*unseen data*).

b. *Callback* 'ReduceLROnPlateau'

Tabel 3.4 Parameter *Callback* ReduceLROnPlateau

Parameter	**Nilai**
Monitor	val_loss
Patience	5 epoch
Faktor Penurunan LR	0.5
min_lr	1×10^{-6}

C*allback* 'ReduceLROnPlateau' berfungsi untuk menyesuaikan nilai laju pembelajaran (*learning rate*) secara dinamis ketika model mulai mengalami stagnansi dalam proses pembelajaran. *Callback* ini memantau *validation loss* dan akan menurunkan nilai *learning rate* sebesar faktor 0.5 apabila tidak terjadi peningkatan performa dalam 5 *epoch*. Nilai *learning rate minimum* (min_lr) juga dijaga agar tidak turun dibawah batas minimum $1x10^{-6}$, sehingga proses pembelajaran tetap berlangsung stabil.

4. Evaluasi Internal

Setelah proses pelatihan selesai, dilakukan tahapan evaluasi internal untuk memastikan model siap dipakai. Tahapan ini mencakup:

a. Seleksi Model Terbaik

Selama masa pelatihan, *callback* 'EarlyStopping' melakukan pemantauan terhadap variabel *validation loss* dan menyimpan bobot terbaik setiap kali *validation loss* membaik. Ketika berhenti, model otomatis mengembalikan bobot terbaik. Hasil pembelajaran adalah bukan model pada *epoch* terakhir, namun model yang memperoleh *validation loss* paling rendah.

b. Ekspor Model

Model terbaik akan disimpan dalam format '.keras'. Format ini menyimpan arsitektur, bobot, dan konfigurasi *optimizer* dalam satu *file*. Penyimpanan ini memastikan model bisa dimuat ulang untuk inferensi atau implementasi tanpa perlu melakukan pelatihan ulang.

c. Penyimpanan Objek Pemrosesan

Selain penyimpanan model, terdapat objek lain seperti *scaler* untuk normalisasi fitur dan *label encoder* untuk pemetaan label kelas. *Scaler* menyimpan parameter statistik seperti mean dan standar deviasi data pelatihan, sehingga data baru dapat dinormalisasi menggunakan parameter yang sama. Konsistensi ini penting untuk menghindari *data shifting* atau *domain shifting*, yaitu kondisi ketika distribusi data input pada saat implementasi model berbeda dari saat pelatihan. *Label encoder* disimpan untuk memastikan pemetaan label *string* ke indeks numerik tetap identik saat model menggunakan data baru.

3.2.7 Evaluasi Model

Evaluasi model pembelajaran mesin pada penelitian ini bertujuan untuk menilai kemampuan intrinsik model LSTM dalam mempelajari dan membedakan pola autentikasi normal dan pola serangan. Evaluasi mengukur sejauh mana model mampu meminimalkan kesalahan deteksi yang berpotensi menyebabkan serangan lolos tanpa teridentifikasi, yang merupakan keterbatasan utama sistem deteksi berbasis aturan yang digunakan oleh perusahaan.

Diberlakukannya pengujian untuk mendapatkan metrik yang sesuai, meliputi:

1. *Learning Curve*

Learning curve digunakan untuk mengevaluasi dinamika proses pelatihan model selama beberapa *epoch* dengan mengamati perubahan nilai *loss* dan akurasi pada data latih dan data validasi.

Kurva ini memberikan gambaran mengenai bagaimana model belajar dari data dan mendeteksi potensi permasalahan seperti *underfitting* atau *overfitting* saat tahap pelatihan.

Pada penelitian ini, *learning curve* diperoleh dengan mencatat nilai *training loss*, *training accuracy*, *validation loss*, dan *validation accuracy* pada setiap *epoch* selama proses pelatihan berlangsung. Dengan membandingkan kurva data latih dan data validasi, dapat diamati apakah peningkatan performa pada data latih diikuti oleh peningkatan performa pada data validasi, yang menandakan proses pembelajaran berjalan secara stabil.

2. Metrik Evaluasi Standar
   a. Akurasi (*Accuracy*)

      Akurasi mengukur proporsi prediksi yang benar dari keseluruhan prediksi yang dibuat oleh model. Akurasi memberikan gambaran umum tentang kemampuan model dalam mengklasifikasikan lalu lintas normal dengan serangan.

$$\text{Accuracy} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}$$

   b. Presisi (*Precision*)

      Mengukur akurasi sistem dalam mengidentifikasi serangan. *Precision* menjadi indikator penting kemampuan model dalam membedakan pola serangan dari aktivitas pengguna asli. *Precision* yang tinggi kritis untuk implementasi produksi guna menghindari peringatan yang tidak relevan yang dapat menurunkan efektivitas respons keamanan.

$$Precision = \frac{TP}{TP + FP}$$

   c. *Recall* (Sensitivitas)

      Mengukur kemampuan sistem untuk mengidentifikasi seluruh serangan yang terjadi dari total serangan yang sebenarnya ada. Recall yang tinggi menunjukkan sistem mampu mendeteksi

serangan meskipun dengan pola yang tidak konvensional. Recall yang rendah mengindikasikan sistem gagal dalam mendeteksi serangan *slow-rate*.

$$Recall = \frac{TP}{TP + FN}$$

d. F1-score

Memberikan keseimbangan antara *precision* dan *recall* melalui mean harmonik kedua metrik tersebut. *F1-score* menjadi indikator utama dalam memberikan evaluasi yang seimbang dibandingkan dengan akurasi tunggal, terutama dalam menghadapi dataset dengan ketidakseimbangan kelas yang signifikan.

$$\text{F1} - \text{Score} = \frac{2 \times (Precision \times Recall)}{Precision + Recall}$$

e. ROC-AUC (*Receiver Operating Characteristics-Area Under Curve*)

Metrik ROC-AUC mengukur kemampuan model dalam membedakan antara lalu lintas normal dan serangan pada berbagai *treshold decision*. Berdasarkan hasil penelitian yang dilakukan oleh Hossain et al., metrik ini menjadi *benchmark* untuk evaluasi performa klasifikasi biner dalam deteksi serangan SSH [12]. Nilai ROC-AUC yang mendekati 1.0 menunjukkan model memiliki kemampuan diskriminasi yang sangat bagus, sementara nilai 0.5 mengindikasikan performa yang tidak lebih baik dari pemilihan secara acak.

$$\text{ROC} - \text{AUC} = \int TPR \, d(FPR)$$

3. Metrik Deteksi Serangan

a. *False Positive Rate* (FPR)

Mengukur tingkat *false alarm* yang dihasilkan sistem, yaitu proporsi aktivitas normal yang salah diidentifikasi sebagai serangan dari total aktivitas normal. FPR yang tinggi dapat

mengindikasikan model belum mampu membedakan pola serangan *slow-rate* dengan variasi normal dari perilaku pengguna.

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

b. *False Negative Rate* (FNR)

Mengukur proporsi serangan yang tidak terdeteksi oleh sistem dari total serangan yang sebenarnya terjadi. FNR yang rendah menunjukkan efektivitas sistem dalam mengatasi keterbatasan sistem deteksi berbasis aturan sedangkan FNR yang tinggi mengindikasikan sistem masih memliki titik buta terhadap varian serangan *slow-rate* tertentu, yang dapat berdampak fatal pada keamanan sistem.

$$\text{FNR} = \frac{\text{FN}}{\text{FN}+\text{TP}} = 1\text{-Recall}$$

c. *True Positive Rate* (TPR)

Mengukur efektivitas deteksi serangan *slow-rate* sebagai proporsi serangan yang berhasil diidentifikasi dari total serangan yang sebenarnya terjadi. Metrik ini menganalisa efektivitas model LSTM dalam mendeteksi serangan SSH *slow-rate*. Hasil tingkat deteksi yang tinggi mengindikasikan model mampu menangkap pola temporal kompleks yang menjadi karakteristik kunci serangan *slow-rate*, sesuai dengan keunggulan LSTM dalam mempertahankan informasi jangka panjang.

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \text{Recall}$$

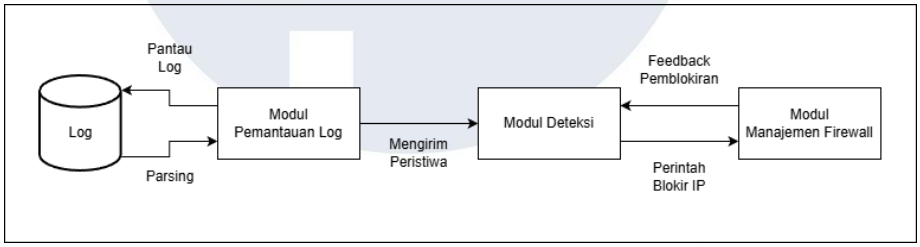
3.3 Perancangan Aplikasi

Penulis memberi nama aplikasi dengan sebutan “SSHGuard”, paket aplikasi bertujuan untuk dapat dipasang pada semua sistem operasi berbasis Linux.

Berikut adalah tahapan perancangan aplikasi:

3.3.1 Arsitektur Aplikasi

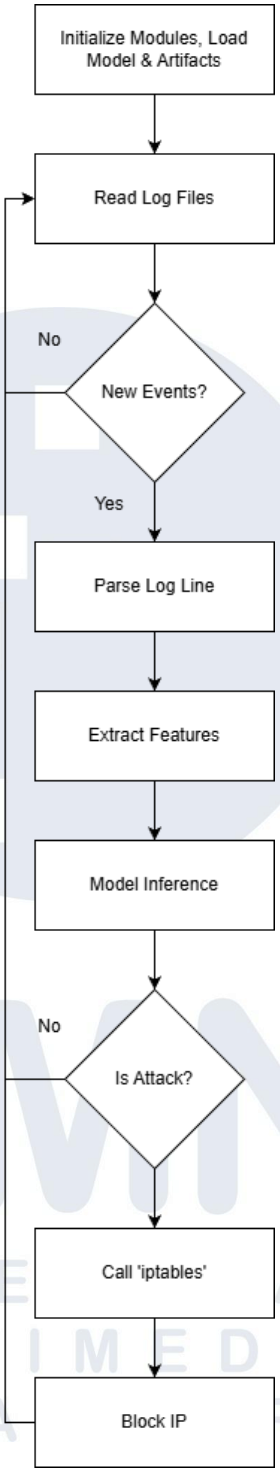
Arsitektur aplikasi SSHGuard terdiri atas tiga komponen utama, yaitu modul pemantauan log, modul deteksi berbasis LSTM, dan modul manajemen *firewall*. Pemisahan komponen ini bertujuan untuk menjaga kejelasan tanggung jawab masing-masing modul serta meningkatkan fleksibilitas dan keterpeliharaan sistem.



Gambar 3.2 Arsitektur Aplikasi

Pada gambar 3.2, arsitektur ini menggambarkan alur pemrosesan data dari sumber log autentikasi hingga tindakan mitigasi keamanan yang diterapkan oleh sistem. Setiap modul bekerja secara terpisah namun saling terhubung, sehingga proses pemantauan, analisis, dan pemblokiran serangan dapat dilakukan secara terstruktur, otomatis, dan berkesinambungan.

1. Alur Kerja Aplikasi



Gambar 3.3 Alur Kerja Aplikasi

Diagram alur pada gambar 3.3 menggambarkan proses kerja utama aplikasi SSHGuard sebagai sistem deteksi dan mitigasi serangan

SSH berbasis pembelajaran mesin. Proses diawali dengan pemuatan model LSTM beserta artefak pendukung seperti *scaler* dan *label encoder*, kemudian dilanjutkan dengan inisialisasi seluruh modul inti aplikasi. Setelah sistem aktif, aplikasi secara kontinu membaca berkas log autentikasi SSH untuk mendeteksi adanya entri baru. Setiap entri log yang ditemukan akan dilakukan *parsing* dan diekstraksi fitur-fiturnya, kemudian dilakukan normalisasi fitur sebelum diproses oleh model LSTM untuk melakukan inferensi. Hasil prediksi dievaluasi untuk menentukan apakah aktivitas tersebut tergolong serangan. Apabila teridentifikasi sebagai serangan, aplikasi akan memanggil mekanisme *firewall* melalui *iptables* untuk melakukan pemblokiran terhadap alamat IP sumber. Seluruh rangkaian proses ini dijalankan secara berulang dalam satu *loop* layanan sehingga memungkinkan pemantauan dan mitigasi serangan SSH secara *realtime* dan berkelanjutan.

2. *Pseudocode*

Pseudocode adalah representasi logika program dalam bentuk langkah-langkah tanpa terikat sintaks bahasa pemrograman tertentu [40]. Berikut merupakan *pseudocode* dari aplikasi sistem deteksi intrusi:

```
BEGIN SSHGUARD_APPLICATION

        LOAD model_from("models/lstm_model.keras")
        LOAD scaler_from("models/scaler.pkl")
        LOAD label_encoder_from("models/label_encoder.pkl")

        INITIALIZE LogMonitor WITH:
          - target_log_file: "/var/log/auth.log"
          - read_mode: "follow"
          - batch_interval_seconds: 5

        INITIALIZE Detector WITH:
          - model
          - scaler
          - label_encoder

        INITIALIZE FirewallManager WITH:
          - backend: "iptables"
          - ban_duration: 600 seconds
          - ban_threshold: 3 detections
```

```
        WHILE application_is_running:

            new_events = LogMonitor.read_new_lines()

            IF new_events IS NOT EMPTY:
                parsed_features = LogMonitor.extract_features(new_events)

                scaled_features = scaler.transform(parsed_features)

                predicted_label = Detector.predict(scaled_features)

                IF predicted_label IN ["FAST_ATTACK",
    "SLOW_ATTACK"]:
                    source_ip = LogMonitor.get_source_ip(new_events)
                    FirewallManager.register_event(source_ip)

                ENDIF
            ENDIF

            FOR each ip IN FirewallManager.offenders:
                IF offender_count(ip) >= ban_threshold AND ip NOT
    already banned:
                    FirewallManager.ban(ip)
                ENDIF
            ENDFOR

            SLEEP(1)
        ENDWHILE

END SSHGUARD_APPLICATION
```

Pseudocode diatas merepresentasikan alur kerja utama dari aplikasi SSHGuard. Aplikasi memadukan tiga komponen inti: pemantauan log, deteksi berbasis model LSTM, dan manajemen *firewall*. Seluruh proses dijalankan secara berkesinambungan dalam satu *loop* layanan.

Tahap pertama dimulai dengan memuat seluruh artefak model yang diperlukan, yakni model LSTM terlatih, objek *scaler* untuk normalisasi fitur, serta *label encoder* untuk melakukan konversi label prediksi dari representasi numerik ke kelas semantik. Ketiga artefak ini bersifat wajib agar proses inferensi dapat menghasilkan prediksi yang konsisten dengan konfigurasi saat pelatihan.

Selanjutnya, komponen pemantauan log diinisialisasi untuk melakukan pembacaan *realtime* terhadap berkas log autentikasi

sistem (pada sistem operasi Linux log autentikasi tersebut disimpan dalam lokasi $/var/log/auth.log$). Komponen ini bertugas untuk menangkap baris log baru, mengekstraksi fitur yang relevan, serta menyusun data sehingga dapat diberikan ke modul deteksi.

Komponen detektor, terdiri atas proses normalisasi menggunakan *scaler* dan inferensi kelas menggunakan model LSTM. Prediksi yang dihasilkan bersifat kategorikal dan merepresentasikan tiga kelas utama: *fast attack*, *slow attack*, dan *benign*.

Jika suatu aktivitas diklasifikasikan sebagai serangan, aplikasi mengekstrak alamat IP sumber dari log autentikasi dan menyerahkannya kepada manajemen *firewall*. Komponen ini bertugas merekam frekuensi kejadian mencurigakan berdasarkan alamat IP serta menerapkan aturan pemblokiran ketika ambang batas tertentu tercapai. Mekanisme ini memastikan bahwa tindakan mitigasi hanya dilakukan terhadap entitas yang menunjukkan pola agresif, sehingga mengurangi kemungkinan pemblokiran keliru.

Seluruh operasi dijalankan dalam *loop* utama aplikasi, yang akan berlangsung selama layanan diaktifkan. Di dalam *loop* ini, aplikasi memproses log baru, melakukan prediksi, mengevaluasi status pelanggar, dan menerapkan pemblokiran apabila diperlukan. Arsitektur ini mendukung pemantauan dan penanggulangan serangan SSH secara otomatis, adaptif, dan berkesinambungan.

3. Modul Pemantauan Log

   Modul ini bertanggung jawab untuk melakukan pembacaan log autentikasi SSH secara *realtime* dari berkas yang ada pada lokasi $/var/log/auth.log$. Berikut merupakan fungsi utama dari modul pemantauan:

   a. *File Tailer*

      Komponen ini memantau pertambahan baris baru pada berkas log autentikasi menggunakan fungsi *file polling*. Komponen ini

berfungsi untuk memastikan sistem merespon secara langsung ketika terjadi peristiwa SSH yang baru tercatat.

b. *Log Parser*

Seperti pada proses perancangan model sebelumnya, setiap baris log diekstraksi untuk memperoleh atribut *timestamp*, IP sumber, dan jenis peristiwa yang terjadi. Modul ini memastikan konsistensi struktur data sama seperti saat pelatihan model.

4. Modul Deteksi

Modul ini menjalankan inferensi dengan model LSTM yang telah dilatih sebelumnya. Komponen utama modul deteksi terdiri dari:

a. *Loader* Artefak Model dan *Preprocessing*

Komponen ini memuat model '.keras', artefak StandardScaler, serta LabelEncoder. StandardScaler dan LabelEncoder.

b. Perancangan Fitur

Menerapkan perancangan fitur pada berkas log yang diteirma oleh modul pemantauan log. Fitur yang digunakan sama seperti pada fitur dalam pelatihan model (lihat tabel 3.2).

c. Model LSTM

Melakukan prediksi terhadap fitur-fitur tersebut menjadi sebuah probabilitas terhadap ketiga kelas yang ada pada model, *fast attack*, *slow attack*, dan *benign*.

d. Pengambil Keputusan

Modul ini mengambil keputusan akhir berdasarkan probabilitas kelas. Komponen ini memastikan hanya prediksi dengan tingkat keyakinan tertentu yang menghasilkan tindakan mitigasi yang akan diambil oleh modul manajemen *firewall*.

5. Modul Manajemen *Firewall*

Modul ini berfungsi untuk mengeksekusi tindakan mtiigasi terhadap IP yang teridentifikasi sebagai ancaman. Komponen pada modul manajemen *firewall* meliputi:

a. Pengendali Kebijakan

Modul pengendali kebijakan bertugas untuk menerjemahkan prediksi model menjadi kebijakan operasional yang bersifat deterministik. Modul ini menginterpretasikan hasil deteksi berdasarkan parameter konfigurasi yang diatur pengguna, seperti:

- enabled: apakah *firewall* diaktifkan
- block_duration: durasi pemblokiran IP dalam detik

Modul ini memastikan setiap keputusan pemblokiran konsisten dengan konfigurasi global.

b. Eksekusi *Firewall*

Komponen ini menambahkan aturan pemblokiran IP ke dalam grup khusus pada 'iptables'. Sebelum menambahkan aturan baru, modul memastikan bahwa aturan untuk IP yang sama belum ada. Hal ini mencegah duplikasi, dan memperkecil kerusakan pada tabel *firewall*.

c. Pelacakan Aturan

Komponen ini berfungsi untuk mencatat status pemblokiran IP dalam struktur data internal aplikasi. Hal ini bertujuan untuk menghindari pemblokiran berulang.

3.3.2 Integrasi Model dan Konfigurasi

Berkas model dan artefak *preprocessing* ditempatkan pada folder $/models$. File tersebut nantinya akan diinstal dalam lokasi $/usr/lib/sshguard/models$ melalui komponen 'setup.py'.

3.3.3 Layanan Sistem dan CLI

Pembuatan aplikasi *command line* dirancang untuk menyediakan antarmuka yang mudah digunakan bagi administrator sistem dalam mengelola sistem deteksi intrusi berbasis LSTM. Aplikasi dikembangkan menggunakan bahasa pemrograman Python dengan

memanfaatkan modul ‘argparse’ untuk menyediakan struktur *command line* yang konsisten dengan standar UNIX/Linux.

Berikut merupakan layanan sistem yang tersedia untuk aplikasi:

1. Struktur *Command Line Interface*

   Aplikasi dirancang dengan hirarki perintah yang sama seperti pola konvensional alat administrasi pada sistem operasi Linux:

   ```
   sshguard [COMMAND]
   ```

   Dimana ‘COMMAND’ merupakan perintah yang dapat dijalankan oleh pengguna untuk melakukan operasi dalam aplikasi.

2. Komponen Utama *Command Line*

   Berikut merupakan perintah dasar yang dapat dijalankan pada aplikasi:

   Tabel 3.5 Perintah Deteksi dalam Aplikasi

Argumen	**Deskripsi**
start	Memulai layanan aplikasi.
stop	Menghentikan proses pemantauan dan layanan aplikasi.
restart	Memulai ulang layanan aplikasi.
status	Menampilkan status layanan aplikasi.
list	Menampilkan daftar alamat IP yang diblokir.
unblock <ip>	Membuka blokir alamat IP.

3. Integrasi dengan Sistem Operasi

   Integrasi aplikasi dengan sistem operasi Linux dilakukan untuk memastikan bahwa modul deteksi dan komponen pendukungnya dapat berjalan secara stabil, aman, serta sesuai dengan standar operasional layanan sistem. Aplikasi menerapkan kode keluar (*exit codes*) yang mengikuti konvensi POSIX, dimana kode 0 digunakan untuk menandai eksekusi yang berhasil, kode 1 untuk menangani kesalahan umum.

Aplikasi menangani sinyal sistem melalui *signal handling* yang mencakup SIGINT (interupsi manual seperti Ctrl+C) dan SIGTERM (terminasi yang dikirim oleh sistem atau *service manager*). Kedua sinyal ditangani dengan mekanisme penghentian bersifat *graceful*, memastikan seluruh proses internal dihentikan secara teratur, *file status* ditutup, *buffer* dibersihkan, dan *rule firewall* dilepaskan.

Integrasi *logging* dilakukan dengan memanfaatkan fasilitas logging bawaan Linux. Aplikasi mendukung pelaporan log ke 'syslog' melalui 'systemd journal,' sekaligus menyediakan *output log* terstruktur ke berkas lokal pada lokasi $/var/log/sshguard.log$.

Pada aspek manajemen proses, aplikasi dirancang untuk dijalankan melalui 'systemd service', yang memberikan mekanisme *auto-restart* apabila proses mengalami kegagalan, serta memastikan bahwa layanan baru berjalan setelah dependensi penting seperti jaringan dan 'syslog' aktif. Integrasi dengan 'systemd' memungkinkan konfigurasi seperti 'RestartPolicy', kontrol sumber daya, isolasi lingkungan eksekusi, dan audit log yang lebih mudah.

3.3.4 Pengemasan Aplikasi

Tahapan ini mencakup proses pembentukan paket aplikasi mulai dari pengemasan Python hingga integrasi ke dalam ekosistem distribusi sistem operasi berbasis Linux atau Debian.

Tahapan ini dirancang agar aplikasi dapat dipasang, diperbarui, dan dijalankan secara konsisten pada berbagai lingkungan produksi:

1. Pengemasan Python

   Pembuatan paket dilakukan sesuai dengan standar distribusi PyPi dan 'setuptools' [41] [42].

   a. Definisi *Metadata* dan Dependensi

      File 'setup.py' digunakan sebagai entri utama untuk mendeklarasikan metadata aplikasi, seperti nama paket, versi, serta dependensi. File 'requirements.txt' digunakan untuk

menjamin kesesuaian antara dependensi pengembangan dengan aplikasi produksi.

b. Distribusi Komponen Eksekusi

Direktori $scripts/sshguard$ disertakan sebagai titik masuk untuk *command-line interface* (CLI), sehingga aplikasi dapat dijalankan secara langsung melalui terminal setelah instalasi.

2. Pengemasan Debian

Tahap ini bertujuan untuk menjadikan aplikasi sebagai paket Debian (.deb) yang dapat dikelola oleh sistem manajemen paket standar seperti 'apt', 'dpkg' dan 'systemd'. Proses pengemasan Debian dilakukan dengan membangun struktur direktori Debian yang berisi berkas *control*, *rules*, *install*, *postinst*, dan *prerm* sebagai fondasi pembentukan paket. File ini mendefinisikan dependensi sistem, prosedur instalasi, serta skrip *hook* yang dijalankan sebelum maupun sesudah instalasi.

3. Integrasi Sistem

Setelah instalasi, skrip 'postinst' secara otomatis menyalin model ke lokasi $usr/lib/sshguard/models$, file konfigurasi ke '/etc/sshguard', dan mendaftarkan layanan ke dalam 'systemd' sehingga aplikasi dapat dijalankan sebagai *daemon*. Manajemen layanan dapat dilakukan melalui perintah standar seperti 'systemctl start', 'stop', dan 'status', serta memanfaatkan fitur *auto-restart*, *dependency ordering*, dan integrasi penuh dengan 'syslog'

3.3.5 Distribusi Aplikasi

Aplikasi akan didistribusikan melalui repositori GitHub pada alamat https://github.com/bintangtimurlangit/sshguard, sehingga pengguna dapat mengakses kode sumber secara terbuka dan membangun aplikasi langsung dari repositori tersebut. Model distribusi ini mengikuti pendekatan *build-from-source*, dimana pengguna diharuskan untuk kloning repositori melalui perintah $git\ clone$ dan kemudian

menjalankan proses pembangunan paket menggunakan *dpkg buildpackage*. Pendekatan ini memberikan fleksibilitas bagi pengguna untuk meninjau kode dan memodifikasi konfigurasi.

3.3.6 Evaluasi Aplikasi

Evaluasi aplikasi dilakukan untuk menilai dampak praktis penerapan sistem deteksi intrusi berbasis LSTM terhadap permasalahan keamanan yang dihadapi perusahaan, yaitu keterbatasan mekanisme berbasis aturan dalam mendeteksi serangan SSH *slow-rate*. Berbeda dengan evaluasi model yang berfokus pada performa klasifikasi, evaluasi aplikasi menitikberatkan pada kemampuan sistem dalam meningkatkan visibilitas serangan, memperluas cakupan deteksi, dan mengurangi titik buta keamanan pada lingkungan operasional.

1. Lingkungan Pengujian

   Pengujian dirancang untuk memastikan bahwa kedua aplikasi, yaitu Fail2Ban dan SSHGuard, dievaluasi dalam kondisi yang setara, terukur, dan dapat direplikasi. Seluruh pengujian dilakukan pada lingkungan terkontrol berbasis Proxmox LXC *container* yang dijalankan di atas platform Proxmox. Dengan pendekatan ini, setiap aplikasi diuji pada *instance* sistem yang terisolasi sehingga terbebas dari interferensi eksternal yang dapat memengaruhi hasil pengujian.

   Lingkungan pengujian aplikasi diatur sebagai berikut:

   Tabel 3.6 Spesifikasi LXC *Container*

Spesifikasi	Nilai
CPU	4 Core
Memori	4096 MB
Sistem Operasi	Ubuntu 24.04
Penyimpanan	20 GB

   Adapun perubahan parameter pada masing-masing *container* sehingga memiliki pengaturan SSH sebagai berikut:

   Tabel 3.7 Parameter SSH pada LXC *Container*

Parameter	Nilai
PermitRootLogin	yes
Password Authentication	yes
Port	22

2. Metode Pengujian

Pengujian dilakukan menggunakan metode *log replay*, yaitu dengan menyuntikkan rangkaian peristiwa autentikasi ke dalam berkas log pada masing-masing *container* Proxmox. Setiap rangkaian log merepresentasikan pola serangan atau aktivitas normal yang telah didefinisikan pada skenario pengujian sebelumnya. Proses injeksi log dilakukan secara berurutan dengan mempertahankan urutan waktu antar entri, sehingga aplikasi pendeteksi serangan memproses peristiwa tersebut seolah-olah berasal dari sistem secara *real-time*.

Pada tahap pengujian ini, digunakan dataset yang berbeda dari dataset pelatihan, yaitu dataset autentikasi yang bersumber dari SecRepo. SecRepo merupakan repositori dataset keamanan siber yang menyediakan berbagai kumpulan data autentikasi dan aktivitas sistem, yang dikurasi dari lingkungan nyata maupun skenario serangan yang telah terdokumentasi [43]. Penggunaan dataset tambahan ini bertujuan untuk mengevaluasi kemampuan sistem dalam mendeteksi serangan pada data yang memiliki karakteristik berbeda dan belum pernah digunakan selama proses pelatihan model.

Selain mendukung generalisasi pengujian, metode *log replay* juga berperan penting dalam meningkatkan keterulangan eksperimen (*reproducibility*). Dengan memanfaatkan rangkaian log autentikasi yang telah direkam, setiap skenario pengujian dapat dijalankan ulang dalam kondisi yang identik tanpa dipengaruhi oleh variasi

lingkungan, waktu eksekusi, atau perilaku sistem yang tidak terkontrol.

3. Metrik Evaluasi

   Evaluasi performa kedua aplikasi dilakukan dengan menggunakna metrik kuantitatif yang digunakan juga dalam evaluasi aplikasi.

   a. *Coverage*

      Coverage merupakan metrik yang digunakan untuk menilai sejauh mana aplikasi mampu melakukan pengamatan dan pengambilan keputusan terhadap aktivitas yang muncul dalam log autentikasi sistem.

      Coverage terhadap IP mengukur persentase jumlah IP unik yang berhasil terobservasi dan diproses oleh aplikasi dibandingkan seluruh IP unik yang terdapat pada benchmark log. Semakin tinggi nilai *coverage* ini, semakin kecil kemungkinan aplikasi melewatkan keberadaan aktivitass baik itu serangan maupun aktivitas *benign*.

      Coverage terhadap deteksi mengukur sejauh mana aplikasi memberikan keputusan deteksi terhadap IP yang diamati, yaitu apakah alamat IP tersebut diklasifikasikan sebagai serangan atau aktivitas *benign*. Nilai *coverage* yang tinggi menunjukkan aplikasi tidak hanya mengamati log tetapi juga konsisten memberikan keputusan atas seluruh data yang masuk.

   b. Metrik Evaluasi Standar

      - Akurasi (*Accuracy*)

        Akurasi mengukur tingkat ketepatan aplikasi dalam mengklasifikasikan seluruh peristiwa yang diuji, baik serangan maupun aktivitas normal. Metrik ini dihitung sebagai perbandingan antara jumlah prediksi yang benar terhadap keseluruhan data pengujian. Nilai akurasi tinggi menunjukkan aplikasi mampu membedakan aktivitas

berbahaya dan aktivitas normal secara efektif, sedangkan nilai akurasi yang rendah mengindikasikan adanya kesalahan klasifikasi, baik berupa serangan yang terlewat maupun aktivitas normal yang salah terdeteksi sebagai serangan.

- *Recall*

  Mengukur kemampuan sistem untuk mengidentifikasi seluruh serangan yang terjadi dari total serangan yang sebenarnya ada. *Recall* yang tinggi menunjukkan sistem mampu mendeteksi serangan meskipun dengan pola yang tidak konvensional. Recall yang rendah mengindikasikan sistem gagal dalam mendeteksi serangan.

- Presisi (*Precision*)

  Presisi mengukur tingkat ketepatan aplikasi dalam mengidentifikasi serangan. Nilai *precision* yang tinggi berarti sebagian besar aktivitas yang dideteksi sebagai serangan memang benar merupakan serangan nyata. Dengan demikian, presisi digunakan untuk mengevaluasi risiko kesalahan pemblokiran terhadap pengguna yang asli (*false positive*).

- *F1-Score*

  F1-score Merupakan gabungan harmonis dari presisi dan recall, khususnya berguna ketika terdapat ketidakseimbangan kelas pada data (misalnya serangan jauh lebih banyak daripada *benign*). F1-score memberikan pandangan yang lebih adil apakah model tidak hanya benar saat mendeteksi serangan, namun juga konsisten tidak melewatkannya.

c. Metrik Evaluasi Deteksi

- *False Negative Rate* (FNR)

  Mengukur proporsi serangan yang tidak terdeteksi oleh sistem dari total serangan yang sebenarnya terjadi. FNR yang rendah menunjukkan efektivitas sistem mendeteksi serangan sedangkan FNR yang tinggi mengindikasikan sistem masih memliki titik buta terhadap sebuah varian serangan.

- *True Positive Rate* (TPR)

  Mengukur kemampuan aplikasi dalam mendeteksi serangan yagn benar-benar terjadi. Nilai TPR yang tinggi menunjukkan bahwa aplikasi dapat mengenali serangan yang datang, sedangkan nilai TPR yang rendah mengartikan bahwa model tidak dapat atau melewati serangan yang terjadi.

- *False Positive Rate* (FPR)

  Nilai FPR menggambarkan aktivitas *benign* yang secara keliru dianggap sebagai serangan. FPR dapat menilai keandalan aplikasi pada kondisi operasional agar tidak menggangu aktivitas pengguna yang valid. Nilai FPR yang tinggi menandakan aplikasi gagal dalam mendeteksi, sehingga menyebabkan pengguna normal ikut terblokir, sedangkan nilai rendah mengindikasikan bahwa aplikasi berfungsi sebagaimana mestinya dalam pemblokiran IP yang dianggap mencurigakan.