BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan seluruh tahapan penelitian yang telah dilaksanakan, mulai dari pemahaman masalah, pengolahan data, pemodelan, hingga evaluasi, dapat ditarik kesimpulan yang menjawab rumusan masalah sebagai berikut:

1. Proses pengubahan log syslog Fortigate dari format .txt menjadi data tabular yang siap dianalisis berhasil dilakukan melalui beberapa tahapan kunci.
    a) Pertama, dikembangkan sebuah fungsi parsing menggunakan Python dengan pustaka *Regular Expression* (RegEx) di dalam models.py. Fungsi ini dirancang khusus untuk menangani format log Fortigate yang memiliki *header* syslog diikuti oleh serangkaian pasangan key=value. Fungsi ini mampu secara efektif mengabaikan *header* untuk proses ekstraksi fitur, sambil tetap menyimpan seluruh baris log mentah (_raw_log_line_) untuk keperluan analisis dan tampilan.
    b) Kedua, data yang telah diparsing kemudian ditransformasi menjadi struktur data tabular menggunakan Pandas DataFrame.
    c) Ketiga, dilakukan seleksi fitur berdasarkan Analisis Data Eksploratif (EDA) untuk memilih atribut yang paling relevan dan informatif, serta membuang fitur dengan varians rendah atau *missing values* yang sangat tinggi.

d) Keempat, dilakukan pra-pemrosesan akhir pada data tabular terpilih, yang meliputi penanganan nilai kosong, *encoding* fitur kategorikal menggunakan LabelEncoder, dan normalisasi seluruh fitur ke dalam rentang [0, 1] menggunakan MinMaxScaler, sehingga menghasilkan *dataset* numerik yang siap digunakan oleh model *machine learning*.

2. Pembangunan dan penerapan model deteksi anomali menggunakan Autoencoder dan One-Class SVM berhasil diimplementasikan dalam sebuah alur kerja yang terstruktur.
    a) Pembangunan dan pelatihan model dilakukan secara *unsupervised* di dalam train_script.ipynb, di mana kedua model dilatih hanya menggunakan data log normal (data/syslog.txt atau data/syslog_normal.txt) yang telah diproses.
    b) Autoencoder dibangun menggunakan *library* TensorFlow dan Keras dengan arsitektur *multi-layer* (misalnya, Input(23) -> Dense(12) -> Dense(8) -> Dense(12) -> Dense(23)). Model ini dilatih untuk meminimalkan *reconstruction error* (MSE) dan dilengkapi dengan mekanisme EarlyStopping untuk mencegah *overfitting*.
    c) One-Class SVM diimplementasikan menggunakan sklearn.svm.OneClassSVM dengan parameter seperti kernel='rbf' dan nu=0.005 untuk mempelajari batas (boundary) yang melingkupi data normal.
    d) Penerapan kedua model ini diwujudkan dalam sebuah aplikasi *dashboard* interaktif yang dibangun menggunakan Streamlit

(pages/1_Dashboard.py). Aplikasi ini memungkinkan pengguna untuk mengunggah file log baru, yang kemudian akan diproses dan dievaluasi oleh kedua model yang telah dilatih untuk mendeteksi anomali secara *real-time*.

3. Performa model dalam mengidentifikasi akses mencurigakan, berdasarkan metrik evaluasi *precision, recall, F1-score*, dan *ROC AUC*, menunjukkan bahwa model Autoencoder secara signifikan lebih unggul daripada One-Class SVM untuk kasus penggunaan dan *dataset* ini.
    a) Pada pengujian skala besar menggunakan *dataset* evaluasi berlabel, Autoencoder berhasil mencapai Recall 1.0000 (mendeteksi semua anomali), Precision 0.7918, dan F1-Score 0.8838. Hasil ini menunjukkan keseimbangan yang sangat baik antara kemampuan mendeteksi ancaman dan meminimalkan alarm palsu, dengan hanya menghasilkan 23.668 *False Positives* dari 221.121 data normal.
    b) One-Class SVM pada pengujian yang sama hanya mencapai Precision 0.6018 dan F1-Score 0.7507, dengan jumlah *False Positives* yang jauh lebih tinggi (59.371).
    c) Kedua model menunjukkan kemampuan pemisahan kelas yang sangat baik dengan skor ROC AUC yang mendekati sempurna (~0.998). Namun, F1-Score dan Precision yang lebih tinggi pada Autoencoder menjadikannya model yang lebih praktis dan dapat diandalkan.

d) Hasil dari 10 skenario pengujian dengan rasio anomali yang bervariasi lebih lanjut mengkonfirmasi keunggulan Autoencoder yang secara konsisten menghasilkan jumlah *false positive* yang sangat rendah.

4. Keunggulan performa Autoencoder dapat diatribusikan pada kemampuannya untuk mempelajari representasi data yang kompleks, yang lebih cocok untuk data log dibandingkan dengan pendekatan berbasis batas dari One-Class SVM.

Temuan ini sejalan dengan penelitian oleh Van der Veen et al. (2022), yang dalam studi perbandingan pada beberapa dataset log, menemukan bahwa Autoencoder secara konsisten menunjukkan performa yang sangat baik dan stabil. Hal ini menunjukkan bahwa kemampuan Autoencoder untuk belajar "struktur" data normal secara mendalam lebih efektif untuk data log yang memiliki banyak variasi.

Lebih lanjut, seperti yang dijelaskan oleh Schlegl et al. (2017), metode yangbelajar untuk memodelkan distribusi data (seperti Autoencoder) lebih unggul daripada metode yang mencoba menemukan batas geometris yang kaku (seperti One-Class SVM). Dengan kata lain, OC-SVM kesulitan mendefinisikan satu "pagar" yang bisa mencakup semua variasi aktivitas normal yang sah pada data log, sehingga menghasilkan banyak false positive. Sebaliknya, Autoencoder lebih fleksibel karena ia belajar "aturan main" dari data normal dan akan menandai anomali setiap

kali ada aktivitas yang melanggar aturan tersebut, di mana pun posisinya dalam ruang fitur.

5.2 Saran

Untuk pengembangan sistem di masa depan, terdapat beberapa area yang dapat dieksplorasi lebih lanjut:

1. **Optimasi *Hyperparameter* dan *Threshold***: Meskipun performa Autoencoder sudah sangat baik, terdapat potensi untuk mengurangi sisa *false positive* (23.668) dengan melakukan *tuning* yang lebih halus pada threshold_percentile atau melakukan *hyperparameter tuning* yang lebih ekstensif pada arsitektur Autoencoder.
2. **Rekayasa Fitur Lanjutan**: Untuk fitur dengan kardinalitas sangat tinggi seperti srcip dan dstip yang pada akhirnya dihilangkan, di masa depan dapat dieksplorasi teknik representasi yang lebih canggih daripada LabelEncoder, seperti *IP embedding* atau konversi menjadi fitur-fitur baru.
3. **Implementasi *Online Learning* atau *Retraining* Berkala**: Sistem saat ini menggunakan model statis. Untuk lingkungan produksi nyata, disarankan untuk mengimplementasikan mekanisme *retraining* model secara berkala (misalnya, mingguan atau bulanan) dengan data normal baru untuk beradaptasi dengan perubahan pola lalu lintas jaringan yang sah.
4. **Mekanisme Umpan Balik (*Feedback Loop*)**: Aplikasi dapat dikembangkan lebih lanjut untuk menyertakan fitur di mana analis dapat memberikan umpan balik pada hasil deteksi (misalnya, menandai sebuah deteksi sebagai "*False Positive*"). Umpan balik ini dapat dikumpulkan dan

digunakan untuk *semi-supervised learning* atau untuk memvalidasi dan menyempurnakan model pada siklus *retraining* berikutnya.