BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi yang terus meningkat membuat kehidupan semakin mudah untuk mengakses informasi, bahkan saat ini setiap perangkat terkoneksi dalam jaringan komputer seperti internet. Dengan pesatnya perkembangan teknologi internet dan teknologi komunikasi, semakin banyak sistem komputer dan jaringan yang telah diserang penyusup secara jahat berupa serangan siber (Deng, 2020.). Di Indonesia, Badan Siber dan Sandi Negara (BSSN) mencatat bahwa pada tahun 2023 terjadi 279,84 juta serangan siber. Serangan siber sendiri ada banyak macamnya seperti anomali trafik, *data breach*, *web defacement* , serta berbagai bentuk serangan lainnya. Serangan siber yang sering terjadi adalah anomali trafik. Data dari BSSN di bulan Agustus tahun 2023 terdapat 78.464.385 anomali trafik di Indonesia (Badan Siber dan Sandi Negara [BSSN], 2023).

Dengan banyaknya anomali trafik perlu sebuah sistem yang dapat mendeteksi sebuah anomali trafik pada jaringan. *Intrusion Detection System* (IDS) digunakan untuk memeriksa data jaringan dan mengidentifikasi aktivitas aktivitas jaringan yang tidak normal (Logeswari et al., 2023). IDS dapat dibagi menjadi 2 sistem yaitu *Network-based Intrusion Detection System* (NIDS) dan *Host-based Intrusion Detection System* (HIDS). NIDS mengumpulkan perilaku jaringan melalui perangkat jaringan (switch, router, dan tap), dan menganalisis serta mengidentifikasi serangan tersembunyi dalam lalu lintas jaringan. NIDS mengumpulkan dan menganalisis sejumlah data yang besar pada lalu lintas jaringan. Data ini menyembunyikan informasi tentang apakah trafik yang terdeteksi itu *benign* atau *malicious* (Gao et al., 2020).

Dataset yang digunakan untuk deteksi intrusi sangat penting dan dapat dibagi menjadi data berbasis paket dan data berbasis aliran. *Dataset* berbasis paket terutama mengumpulkan beberapa informasi meta seperti protokol dan ukuran dalam paket. Sebaliknya, *dataset* berbasis aliran memperlakukan paket yang berbagi atribut dalam rentang waktu tertentu sebagai aliran. Banyak *dataset* deteksi intrusi modern yang berbasis aliran, seperti CICIDS-2017, CICIDS-2018 (Yin et al., 2022). Pada penelitian ini penulis akan membuat sebuah *dataset* yang berbasis aliran.

Untuk mendeteksi jaringan yang anomali dapat menggunakan sebuah metode *machine learning* seperti *supervised machine learning*, *unsupervised machine learning*, dan *deep learning*. Dengan menggunakan metode *supervised machine learning*, perlu untuk melabel data yang sangat banyak dan akan memakan banyak waktu dalam melabel data apalagi untuk *dataset* jaringan yang berbasis aliran sangat banyak data yang perlu untuk di label. Menurut Cognilytica Pelabelan data dapat menghabiskan hingga 80% waktu persiapan data, dan setidaknya 25% dari keseluruhan proyek *machine learning* dihabiskan untuk pelabelan (Buhl, 2023). Oleh sebab itu penulis menggunakan metode *Unsupervised learning* untuk mendeteksi anomali pada jaringan terenkripsi. Dibandingkan dengan menggunakan metode *supervised learning*, metode *Unsupervised learning* karena trafik terenkripsi memiliki fitur yang mirip dengan trafik

benign dan dengan demikian dapat menghindari sistem pendeteksian, khususnya metode *supervised learning*, tidak dapat mendeteksi trafik berbahaya terenkripsi dengan pola yang tidak diketahui, metode *supervised learning* perlu untuk *train data* yang banyak agar mengetahui pola jenis serangan, perlu melabel data secara manual (Fu et al., 2023).

Pada penelitian yang dilakukan menggunakan metode *unsupervised machine learning*, yaitu metode *clustering*. *Clustering* adalah teknik yang digunakan untuk membandingkan informasi dan mengkategorikan informasi yang serupa ke dalam set. Algoritma *clustering* digunakan tidak hanya untuk menetapkan dan mengklasifikasikan informasi, tetapi juga untuk kompresi data dan pembangunan yang sempurna. *Clustering* dapat diterapkan dalam data yang sangat banyak (Al-Sanjary et al., 2020).

Penelitian terkait tentang deteksi anomali dengan menggunakan *clustering* dilakukan oleh Dingsheng Deng (2020) dengan judul Research On Anomaly Detection Method Based On DBSCAN *Cluster*ing Algorithm penulis melakukan research dengan membandingkan metode *clustering DBSCAN* dan *K-Means clustering* untuk medeteksi data anomali. Pada jurnal tersebut ditemukan sebuah kesimpulan bahwa dengan menggunakan *DBSCAN clustering* memiliki karakteristik khusus yaitu kecepatan yang cepat dan *accuracy* yang tinggi dibandingkan dengan *K-Means Clustering*. Pada penelitian ini tidak dijelaskan mengenai *dataset* yang digunakan dan hanya membandingkan antara *DBSCAN* dan *K-Means* dan tidak dijelaskan mengenai pengelompokkan data hasil dari *DBSCAN* dan *K-Means*.

Oleh karena itu, penelitian ini bertujuan untuk mencari algoritma terbaik untuk mengelompokkan data data jaringan terenkripsi dan membandingkan *accuracy* antara *DBSCAN* dan *K - MEAN*S *clustering*. Dengan adanya penelitian ini, diharapkan dapat membantu untuk menemukan algoritma *clustering* terbaik yang lebih efektif dalam mendeteksi pola *malicious* dan *benign* pada trafik jaringan terenkripsi.

1.2 Rumusan Masalah

Adapun rumusan masalah yang menjadi dasar penelitian, yaitu:

a. Bagaimana cara mendeteksi keanomalian pada data trafik jaringan terenkripsi dengan metode *clustering*?
b. Bagaimana perbedaan hasil evaluasi dalam metode *clustering* untuk mengelompokkan data jaringan terenkripsi?

1.3 Tujuan

Berdasarkan permasalahan yang telah dirumuskan sebelumnya, maka tujuan dari penelitian yang akan dilakukan adalah:

a. Mendeteksi keanomalian pada data trafik jaringan terenkripsi dengan metode *clustering*.
b. Menganalisis perbedaan hasil evaluasi untuk mengelompokkan data jaringan terenkripsi dengan menggunakan metode *clustering*.

1.4 Manfaat

Dengan dilakukannya penelitian ini, diharapkan manfaat yang didapatkan antara lain:

a. Menghasilkan sebuah sistem pengelompokkan data untuk *dataset* jaringan terenkripsi dengan menggunakan metode *clustering*.
b. Mengetahui model *clustering* yang paling efektif dalam mengelompokkan data jaringan terenkripsi.

1.5 Ruang Lingkup

Adapun ruang lingkup dalam penelitian ini adalah sebagai berikut:

a. Data yang diambil adalah data jaringan terenkripsi di Fakultas Teknik Universitas Hasanuddin dalam bentuk *PCAP file*.
b. Data jaringan terenkripsi yang digunakan adalah *flow-based*.
c. Pengambilan data hanya dapat digunakan pada topologi yang telah di tentukan di Jaringan Fakultas Teknik Universitas Hasanuddin
d. Penelitian memaparkan performa metode *clustering* seperti *DBSCAN* dan K- *MEAN*S dalam mengelompokkan data jaringan terenkripsi.

1.6 Teori

1.6.1 Deteksi Anomali Trafik

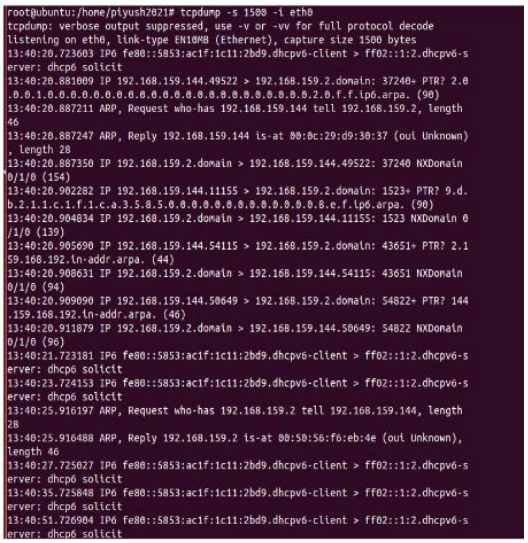
Teknik deteksi anomali trafik adalah proses identifikasi trafik jaringan yang mencurigakan dan berbahaya atau menimbulkan ancaman keamanan. Teknik deteksi anomali dapat mendeteksi serangan siber dengan menganalisis paket jaringan (Vega et al., 2023). Deteksi anomali adalah penemuan pola dalam data atau dalam paket yang tidak mengikuti perilaku (behavior) yang diharapkan. Pola-pola yang tidak mengikuti perilaku dari suatu packet dapat dikatakan sebagai data anomali, tetapi juga memiliki referensi dan bisa dikatakan sebagai outlier pada data. Dalam penelitian yang dilakukan oleh (Sarossy, 2021) dijelaskan bahwa anomali deteksi memiliki cangkupan yang luas seperti:

a. Deteksi instruksi pada keamanan siber digunakan untuk menganalisis pola lalu lintas dalam sistem jaringan, pola lintas yang tidak normal dapat mengidikasikan bahwa komputer telah diretas atau sedang terjadi serangan dengan mengirimkan data sensitif.
b. Deteksi kesalahan digunakan pada sistem yang sangat penting bagi keamanan untuk memastikan sistem tidak mengalami kegagalan, dan mencegah konsekuensu yang tidak dapat diterima.
c. Deteksi penipuan digunakan untuk perawatan kesehatan, asuransi, atau kartu kredit dan digunakan untuk melindungi dari serangan terhadap informasi pribadi dan ekonomi mereka.
d. Pengawasan militer digunakan untuk mendeteksi aktivitas musuh.

1.6.2 Tcpdump

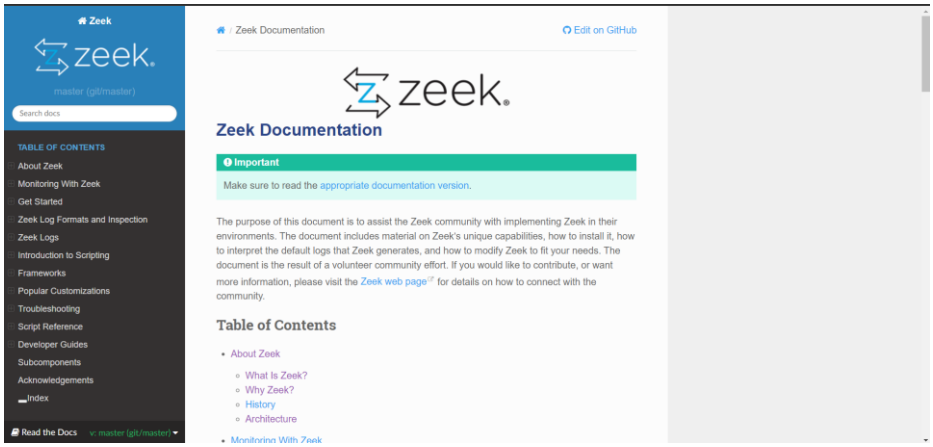
Tcpdump merupakan salah satu *packet sniffer*. *Packet sniffer* merupakan sebuah program komputer yang dapat menangkap lalu lintas yang masuk dan keluar pada jaringan. Tcpdump dapat menyimpan detail paket dan dapat digunakan di kemudian hari. Menurut (Goyal & Goyal, 2017) tcpdump merupakan salah satu alat penangkap lalu lintas jaringan terbaik dan yang paling ringan. Tcpdump sangat mudah untuk dipelajari khususnya yang ingin pemahaman menyeluruh mengenai TCP/IP karena tcpdump membuang paket kedalam format mentah tanpa banyak analisis. Contoh dari tcpdump dapat dilihat pada Gambar 1.

Tcpdump menangkap dan menampilkan header paket dan mencocokkanya dengan serangkaian kriteria. Tcpdump memiliki sebuah *library* seperti libcap. Libcap adalah pustaka C yang digunakan untuk menangkap paket jaringan. Pustaka ini menyediakan antarmuka standar yang kompatibel dengan berbagai sistem operasi umum, termasuk Linux, FreeBSD, dan sistem berbasis Unix lainnya. Dengan menyediakan metode yang seragam, libcap memungkinkan aplikasi untuk menangkap paket data yang melewati jaringan pada berbagai platform tanpa perlu memikirkan perbedaan spesifik antara sistem operasi. Tcpdump menganalisis dan memfilter paket IP dan paket ARP atau protokol apapun pada lapisan yang lebih tinggi dari ethernet. Tcpdump tidak menganalisis paket, tcpdump hanya dapat melaporkan apa yang didapatkan di dalam paket (Jyoti et al., 2023).



Gambar 1. Contoh gambar *capture* dengan *Tcpdump*
Source: Goyal & Goyal, 2017

1.6.3 Zeek Flow Meter



Gambar 2. Halaman beranda Zeek Documentation
Source: docs.zeek.org

Zeek dikembangkan pertama kali oleh Vern Paxson pada tahun 1990an. Pada awal dikembangkan Zeek lebih dikenal dengan nama ”Bro”. Bro digunakan sebagai sarana untuk memahami apa yang terjadi pada jaringan universitas dan jaringan laboratorium nasionalnya. Pada tahun 2018 Bro mengganti nama menjadi Zeek pada akhir tahun 2018 untuk merayakan ekspansi dan pengembangannya yang berkelanjutan.

Zeek adalah sebuah alat analisis lalu lintas jaringan. Zeek mendukung berbagai macam tugas analisis trafik di luar domain keamanan, termasuk pengukuran kinerja dan pemecahan masalah. Zeek secara diam - diam mengamati lalu lintas jaringan. Zeek menginterpretasikan apa yang dilihat dan menciptakan log transaksi, konten *file* dan keluaran yang sepenuhnya disesuaikan, dan cocok untuk tinjauan manual pada disk atau alat yang lebih ramah seperti SIEM (*Security and Information Event Management*). Zeek dapat digunakan untuk mendeteksi berbagai jenis lalu lintas jaringan berbahaya (Gustavsson, 2019).

Zeek dioptimalkan untuk menafsirkan lalu lintas jaringan dan menghasilkan sebuah log berdasarkan lalu lintas tersebut. Zeek bukan sebuah penganalisis protokol seperti Wireshark yang menganalisis setiap detail kecil dari lalu lintas jaringan atau menyimpan data lengkap dalam format *file packet capture* (*PCAP*), Zeek memilih pendekatan yang lebih seimbang. Zeek mencatat log jaringan secara ringkas namun tetap sangat akurat, sehingga memudahkan pemahaman tentang pola dan penggunaan jaringan tanpa menghasilkan data yang berlebihan.

1.6.4 *Machine Learning*

Machine learning atau Pembelajaran Mesin adalah sebuah cabang dari *Artificial Intelligence*, yang bertujuan agar komputer dapat belajar tanpa di programkan (Bi et al., 2019). *Machine learning* memungkinkan untuk komputer agar mempelajari data yang telah dimasukkan. Algoritma *machine learning* dapat memprediksi hasil dari sekumpulan

data yang telah dimasukkan, tetapi *machine learning* memerlukan data yang telah dilatih agar mesin dapat mempelajari data tersebut terlebih dahulu.

Ada berbagai jenis dari *machine learning* seperti *supervised learning*, *unsupervised learning*, dan *semisupervised learning*. *Supervised learning* adalah pembelajaran mesin yang memerlukan label, label adalah nilai dari hasil dalam *machine learning*. *Unsupervised learning* adalah pembelajaran mesin yang tidak memerlukan suatu label, tetapi algoritmanya mengidentifikasi hubungan atau relasi antara setiap data. *Semisupervised learning* adalah gabungan dari *supervised learning dan unsupervised learning*. *Semisupervised learning* melengkapi data yang berlabel dengan data yang tidak berlabel dengan tujuan meningkatkan kinerja model (penelitian menunjukkan bahwa data tak berlabel dapat membantu membangun pengklasifikasi yang lebih baik, tetapi pemilihan model yang tepat sangat penting) (Bi et al., 2019).

1.6.5 *Unsupervised Learning*

Unsupervised learning merupakan salah satu metode yang ada pada *machine learning*. *Unsupervised learning* adalah *machine learning* yang tidak memerlukan suatu label, berbeda dengan *supervised learning* yang membutuhkan suatu label (Bi et al., 2019). Dalam *Unsupervised learning* model akan belajar dari data yang dimasukkan kemudian mesin akan mengelompokkan data tersebut kedalam kelas-kelas yang serupa (Bhatia, 2019).

Unsupervised learning memiliki kesamaan tujuan dan strutkur dengan pendekatan statistik yang mencoba mengidentifikasi subkelompok berdasarkan kemiripannya (Bi et al., 2019). *Unsupervised learning* umumnya memiliki 2 jenis metode, seperti *clustering*, dan *dimensionality reduction* (Sarossy, 2021) *Clustering* adalah mengelompokkan data berdasarkan karakteristik data yang serupa (Bi et al., 2019). *Dimensionality reduction* adalah untuk menyederhanakan data tanpa kehilangan terlalu banyak informasi. Salah satu cara untuk melakukannya adalah dengan menggabungkan beberapa fitur yang berkorelasi menjadi satu .

1.6.6 *Information gain*

Dalam mendeteksi data anomali memiliki banyak tantangan seperti data yang memiliki dimensi yang tinggi yang berdampak ke kompleksitas komputasi dan waktu komputasi. Salah satu cara untuk menangani tantangan tersebut dengan menggunakan teknik *feature selection. Feature selection* membantu dalam menghilangkan fitur yang tidak diperlukan, membantu memahami data, mengurangi waktu komputasi, dan meningkatkan kinerja mesin. Salah satu metode dari *feature selection* adalah *Information gain* (Kurniabudi et al., 2020).

Information gain digunakan untuk mengurangi dimensi data dengan memilih fitur yang paling relevan berdasarkan perhitungan bobot fitur. Fitur yang tidak relevan akan dieliminasi agar meningkatkan performa dari sistem deteksi. *Information gain* menggunakan entropi yang mengukur ketidakaturan sistem. Entropi yang tinggi dianggap tidak dapat diprediksi karena lebih tidak teratur, entropi yang rendah dianggap sebagai sistem yang dapat diprediksi dan lebih teratur (Yin et al., 2022).

Rumus dari entropi didefinisikan (Yin et al., 2022):

$$H(Y) = -\sum_{i=1}^{n} p(yi) log 2p(yi) \quad (1)$$

Disini n adalah jumlah dari kelas dalam *dataset* Y, dan $p(yi)$ adalah probabilitas memilih elemen yi dari setiap kelas pada *dataset* Y.

Rumus entropi kondisional didefinisikan:

$$H(Y \mid X) = -\sum_{i=1}^{m} p(xi) H(Y \mid X = xi) \quad (2)$$

m adalah jumlah kelas dalam *dataset* X, dan $p(xi)$ adalah probabilitas memilih elemen xi dari *dataset* X. Entropi ini membantu dalam memahami berapa banyak ketidak pastian yang tersisa mengenai *dataset* Y setelah mengetahui nilai X.

Rumus dari *Information gain*:

$$IG(Y, X) = H(Y) - H(Y \mid X) \quad (3)$$

Information gain adalah selisih antara entropi awal $H(Y)$ dan entropi kondisional $H(Y \mid X)$. Nilai ini menunjukkan besar informasi dari fitur yang didapatkan mengenai *dataset* Y dan *dataset* X.

1.6.7 *StandardScaler*

StandardScaler merupakan salah satu metode *preprocessing*. *StandardScaler* adalah metode *preprocessing* yang digunakan untuk menstandarkan fitur dengan menghilangkan nilai rata-rata dan mengatur varians menjadi satuan yang sama. Proses ini diterapkan pada setiap fitur dalam sampel. Tujuannya adalah untuk mencegah adanya perbedaan skala yang terlalu besar antar fitur, yang dapat menyebabkan ketidakseimbangan saat proses pelatihan model (Prasetyo et al., 2022). *StandardScaler* menstandarkan fitur dengan mengurangi *mean* dan penskalaan ke varian unit. Varian unit berarti membagi semua nilai dengan deviasi standar yang dapat dapat dilihat pada persamaan (4).

Rumus *StandardScaler* X adalah nilai asli data μ adalah rata-rata nilai sampel dan σ adalah standar deviasi. (Rachmawati et al., 2024):

$$Xstandard = \frac{X - \mu}{\sigma} \quad (4)$$

1.6.8 *PCA (Principal Component Analysis)*

PCA (Principal Component Analysis) merupakan salah satu algoritma reduksi dimensi (Bhatia, 2019). Algoritma reduksi dimensi paling populer dan sering digunakan adalah *PCA* (Geron, 2019). *PCA* adalah sebuah metode atau algoritma dari reduksi dimensi, yang membantu dalam mengurangi dimensi dari *dataset* yang berdimensi tinggi ke dalam subset linier dengan dimensi yang lebih rendah. *PCA* membuat sebuah variabel yang tidak memiliki korelasi yang akan memaksimalkan nilai varians dan meminimalkan kehilangan informasi dari fitur pada *dataset*. (Sarossy, 2021)

Berikut adalah proses dari penggunaan *PCA* (Sarossy, 2021):

a. Menghitung nilai *Covariance matrix* dari *dataset* yang digunakan, matriks ini menentukan dua variabel saling berkorelasi satu sama lain
b. Meghitung nilai *Eigenvectors* dan *Eigenvalues. Eigenvectors* adalah komponent dari utama *Covariance matrix* dan *Eigenvalues* menunjukkan jumlah informasi dalam komponen utama.
c. Memproyeksikan *dataset* ke ruang baru dari komponen utama yang telah dibentuk.

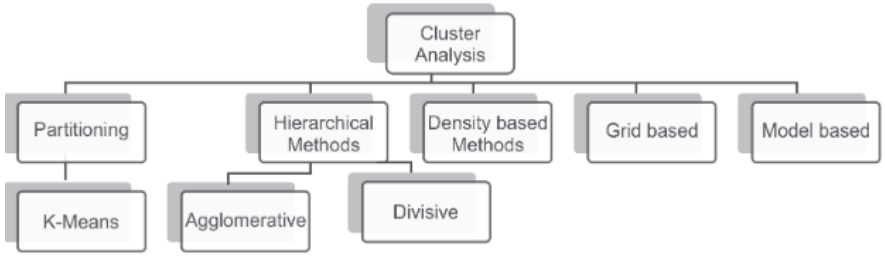
1.6.9 *Clustering*

Clustering merupakan salah satu metode *Unsupervised learning* (Sarossy, 2021). *Clustering* mengelompokkan data berdasarkan kesamaan atau kedekatan antara data tersebut. Data yang diproses dengan metode *clustering* akan mengelompokkan data berdasarkan dari atribut dan fitur yang ada pada data (Bhatia, 2019). *Clustering* berbeda dengan metode klasifikasi, metode *clustering* tidak memerlukan sebuah label untuk dilatih (Bi et al., 2019).

Menurut (Geron A., 2019) ketika melihat sekumpulan tumbuhan yang tidak mirip identik, tetapi tumbuhan itu memiliki beberapa persamaan sehingga dapat mengelompokkan dalam genus atau species yang sama, perlu seorang botanis untuk menentukan species dari bunga tersebut, tetapi tidak perlu seorang botanis untuk mengidentifikasi grup yang memiliki kemiripan, hal tersebut yang dikatakan sebagai *clustering*. Dari buku (Bhatia, 2019) tertulis kemiripan dalam *cluster* adalah seberapa dekat objek-objek tersebut berdasarkan fungsi jarak. Fungsi jarak adalah sebuah metode untuk mengukur kedekatan dua elemen. Elemen yang dimaksud berupa matriks, vektor atau objek lainnya.

Pengaplikasian *clustering* juga cukup banyak seperti, segmentasi kustomer, anomali deteksi, rekomendasi produk dan masih banyak lagi (Geron A., 2019). Menurut (Bhatia, 2019) metode *clustering* dapat dikategorikan menjadi 5 kategori seperti pada Gambar 3:

a. *Partitioning method* : Membangun partisi acak dan secara berulang memperbaiki dengan beberapa kriteria.
b. *Hierarchical method* : Membuat dekomposisi hierachical dari data ataupun objek dari beberapa kriteria
c. *Density-based method :* Metode yang didasarkan dari konektivitas kepadatan fungsi
d. *Grid based method* : Metode yang didasarkan pada struktur perincian beberapa tingkat
e. *Model based method* : Model yang dipertimbangkan untuk setiap *cluster* dan idenya adalah untuk mengidentifikasi yang paling cocok dari model tersebut.



Gambar 3. Kategori *clustering*

Sumber: Bhatia, 2019

1.6.10 *DBSCAN (Density-Based Spatial Clustering of Applications with Noise)*

DBSCAN adalah salah satu metode *clustering* yang berbasis *density based method*. Algoritma *DBSCAN* memilih wilayah berdasarkan kepadatan sebagai titik tengah dari *cluster* (Deng, 2020). Algoritma *DBSCAN* sangat cocok untuk kumpulan data yang besar dengan noise karena dapat mengidentifikasi *cluster* berdasarkan bentuk dan ukuran (Sarossy, 2021). *DBSCAN* adalah algoritma *clustering* yang sangat efektif, ketika berurusan dengan *intrusion detection noise*, karena dapat secara dinamis menggunakan struktur pohon untuk menyimpan informasi terkompresi, sehingga mengurangi pemakaian waktu dan penyimpanan (Deng, 2020). *DBSCAN* sangat bergantung terhadap 2 parameter dalam menentukan *cluster* yaitu ε (epsilon) dan *Min points* (MinPts). Epsilon adalah jarak maksimum antara dua titik agar dianggap bertetangga. MinPts adalah jumlah minimum titik yang harus ada dalam radius ε agar suatu area dianggap cukup padat untuk membentuk *cluster*. (Indini et al., 2022)

Berikut langkah-langkah umum dilakukan oleh algoritma *DBSCAN* (Indini et al., 2022):

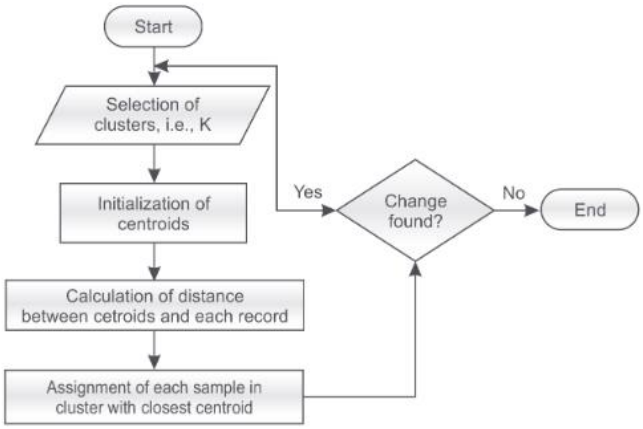
1. Tentukan nilai dari epsilon dan MinPts
2. Tentukan nilai p atau titik awal secara *random* atau acak.
3. Menghitung nilai epsilon atau hitung jarak masing-masing titik yang memiliki kepadatan terhadap titik p dengan rumus Euclidean Distance berikut

   $$D_e = \sqrt{(xi - si)^2 + (yi - ti)^2}$$

   Keterangan
   De : Euclidean Distance
   i : Banyak data
   (x,y) : Titik data
   (s,t) : Titik pusat
4. Sebuah *cluster* terbentuk jika titik sudah mencukupi epsilon lebih dari minimum poin, maka titik tersebut sebagai titik pusat
5. Ulangi tahap 3 dan 4 sampai semua titik dilakukan perhitungan. Lanjutkan ke titik lainnya ketika tidak terdapat titik yang memiliki kepadatan terhadap p atau titik awal.

1.6.11*K-Means*

K–Means merupakan salah satu metode *Cluster* Partitioning. Dalam metode partisi mengelompokkan objek berdasarkan atribut kedalam beberapa partisi (Bhatia, 2019). Dalam algoritma *K-Means*, n objek dikelompokkan kedalam suatu *cluster* atau partisi berdasarkan atribut. Algoritma *K-Means* mengelompokkan objek-objek tersebut kedalam 'k' jumlah *cluster* berdasrkan atribut atau fitur (Bhatia, 2019). 'k' centeroid di inisialisasi secara acak, kemudian dikelompokkan berdasarkan Euclidean Distance antara data dan centroid. Flowchart dari Algoritma *Clustering K-Means* dapat dilihat pada Gambar 4.



Gambar 4. *Flowchart* algoritma *clustering K-Means*
Source: Bhatia, 2019

Dari (Bhatia, 2019) dari Gambar 4 terdapat 5 langkah dari *K-Means clustering* algoritma:

a. Menentukan *k* centeroid atau *cluster* , setelah itu k sampel pelatihan pertama dari n sampel data diambil sebagai *cluster* elemen tunggal. Untuk *cluster* berikutnya berdasarkan centroid terdekat dan centeroid dari *cluster* yang diperoleh kemudian dihitung ulang.
b. Pembuatan *distance matrix* dari centeroid untuk setiap *cluster*. Pada langkah ini jarak matriks dihitung dari setiap sampel data ke setiap *cluster*.
c. Menetapkan setiap data kedalam *cluster* dengan centroid terdekat. Data dikelompokkan berdasarkan jaraknya centroid dari masing-masing *cluster*. Jika data tidak berada dalam *cluster*, maka *cluster* dialihkan ke centroid terdekat, akan berakhir ketika tidak ada lagi perpindahan data ke *cluster* lain.
d. Memperbarui centroid baru untuk setiap *cluster*. Pada langkah ini centroid diperbarui. Lokasi setiap centroid dari *cluster* telah mendapatkan atau kehilangan sampel dengan menghitung rata-rata dari setiap atribut dari semua sampel yang termasuk dalam masing-masing *cluster*.
e. Langkah terakhir mengulangi langkah ke 2 sampai tidak ada lagi perubahan lebih lanjut dan ulangi proses pembaruan setiap lokasi centroid.

1.6.12 *Accuracy, Precision, Recall,* dan *F1 - Score*

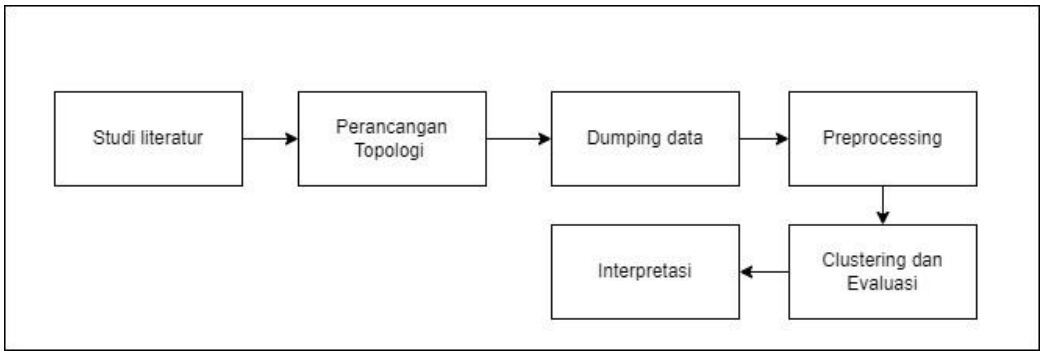
Accuracy, Precision, Recall dan *F1- Score* merupakan salah satu evaluasi kinerja pada model yang digunakan. Berikut adalah penjelasan dari masing – masing matriks:

1. *Accuracy* : mengacu pada proporsi dari prediksi label yang benar sesuai dengan label asli yang diukur dalam bentuk persentase. rumus untuk *accuracy*
   $$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (6)$$
2. *Precision* : mengacu pada evaluasi proporsi nilai dari banyak prediksi positif yang benar dari dari keseluruhan prediksi yang dianggap positif oleh model. Matriks ini cocok ketika false positive tinggi. Rumus dari *precision* :
   $$Precision = \frac{TP}{TP+FP} \quad (7)$$
3. *Recall* : mengacu pada proporsi dari positif aktual yang diidentifikasi secara akurat oleh model. Metrik ini berguna ketika *false negative* (kesalahan dalam tidak mendeteksi kasus positif) dianggap penting. Rumus dari *Recall* :
   $$Recall = \frac{TP}{TP+FN} \quad (8)$$
4. *F1 – Score* : memberikan ukuran yang seimbang antara *precision* dan *recall* untuk menyeimbangkan kedua metrik tersebut. *F1-Score* sangat berguna ketika terdapat ketidakseimbangan antara *precision* dan *recall*, dan tidak ada prioritas yang jelas di antara keduanya. Rumus dari *F1 – Score* :
   $$F1 - score = 2 \times \frac{presisi \times recall}{presisi + recall} \quad (9)$$

BAB II
METODE PENELITIAN

2.1 Tahapan Penelitian

Penelitian "DETEKSI *MALICIOUS* TRAFIK PADA JARINGAN TERENKRIPSI MENGGUNAKAN UNSUPERVISED *MACHINE LEARNING*" terdiri dari beberapa tahapan yang dapat dilihat pada Gambar 5.



Gambar 5. Alur penelitian

Penjelasan tahapan penelitian secara garis besar yaitu sebagai berikut :

a. Studi Literatur

Studi literatur merupakan tahap pertama dalam penelitian ini. Penulis mencari dan mengkaji literatur penelitian, seperti buku, jurnal, e-book, dan penelitian – penelitian sebelumnya yang terkait dengan deteksi anomali, *clustering* dan jaringan terenkripsi yang akan dilakukan sebagai referensi penelitian yang akan dikerjakan.

b. Topologi Jaringan

Pada tahap ini akan menggunakan sebuah topologi jaringan untuk mengcapture trafik jaringan terenkripsi kemudian menyimpannya dalam bentuk *file .PCAP*. Pada tahap ini akan ada 2 jaringan, yaitu jaringan lokal di "Lab Ubiquitos" dan jaringan luar "Fakultas Teknik Universitas Hasanuddin".

c. *Dumping* Data

Pada tahap ini, data yang telah di tangkap dari proses capturing menggunakan topologi jaringan kemudian kita menggunakan Zeek Flow Meter untuk fitur ekstraksi. Contoh dari fitur yang ada seperti *Flow, Subflow, Bulk Transmission*, dan *Definition of Active* dan *Idle Time*. kemudian mengubah *file* packets ke flowmeter karena pada penelitian ini akan menggunakan data *flow-based*. Setelah data telah di transformasi, kemudian mengubah jenis *file .PCAP* menajdi *file* .csv agar *dataset* bisa diproses ke dalam *machine learning*.

d. *Preprocessing*

Pada tahap *preprocessing* data .*PCAP* yang telah diubah menjadi .CSV kemudia di proses dengan metode *preprocessing* untuk menghasilkan data yang siap untuk diimplementasikan ke metode *clustering*. Tahapan *preprocessing* seperti *cleaning, feature selection, label encoding* dan *standardization.*

e. Sistem *Clustering* dan Evaluasi

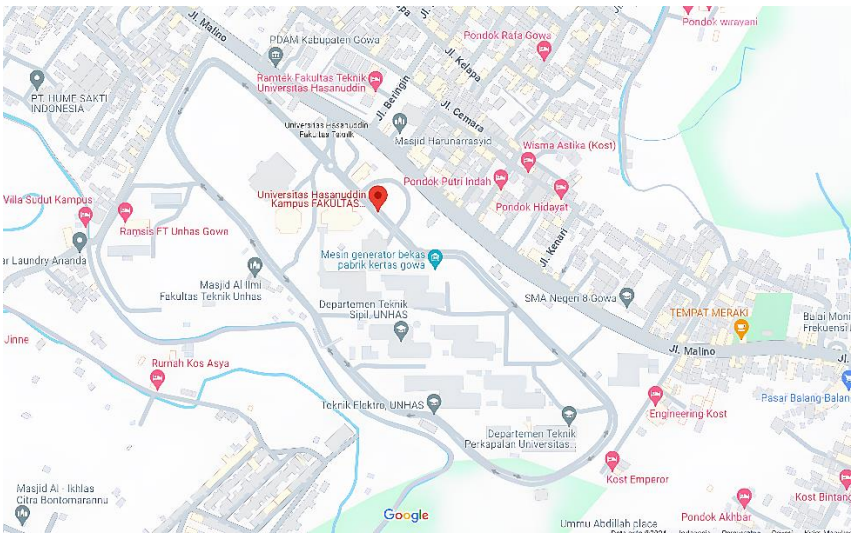
Pada tahap ini setelah melakukan *preprocessing* selanjutnya melakukan proses pembuatan model *clustering* menggunakan algoritma *DBSCAN* dan K- *MEANS clustering* untuk mencari *clustering* yang paling efektif dan cepat dalam memperoses data. Setelah output hasil *clustering* kemudian mengelompokkan masing-masing *cluster* apakah dia termasuk *malicious* atau *benign*. Kemudian melakukan evaluasi performa untuk melihat performa *clustering* yang digunakan.

f. Interpretasi

Pada tahap ini setelah mendapatkan hasil dari *clustering*. Kemudian menganalisis hasil *clustering* untuk melihat pola data apakah di termasuk *benign* ataupun *malicious.* Menentukan *clustering* yang paling efektif dan cepat dalam mengelompokkan data jaringan terenkripsi di fakultas teknik Universitas Hasanuddin seperti menentukan jenis anomali yang terjadi pada data tersebut.

2.2 Waktu dan Lokasi Penelitian

Penelitian dilakukan mulai sejak disetujuinya proposal penelitian pada Februari 2024 sampai November 2024. Penulis melakukan penelitian ini dilakukan di Laboratorium Ubiquitos Computing, Departemen Teknik Informatika, Fakultas Teknik, Universitas Hasanuddin seperti ditunjukkan pada Gambar 6.



Gambar 6. Lokasi penelitian

2.3 Instrumen Penelitian

Instrumen penelitian yang digunakan pada penelitian ini sebagai berikut:

a. *Software*
   1. *Sistem operasi Windows 11 x64*
   2. *Linux*
   3. *Visual Studio Code*
   4. Zeek Flow Meter
   5. *Python*
   6. *Wireshark*
   7. *Tcpdump*
   8. *Microsoft*

b. *Hardware*
   1. *Lenovo Ideapad Gaming 3*
      Laptop yang digunakan memiliki spesifikasi sebagai berikut:
      Sistem Operasi : Windows 11 Home 64-bit
      Layar : 15.6 inci FHD IPS 250nits Anti-glare, 45% NTSC, 165Hz
      Processor : AMD Ryzen 5 5600H with Radeon 3.30 GHz
      RAM : 16 GB
      Storage : 512 SSD
   2. Mini-PC *Victim*
      Sistem Operasi : Linux / Ubuntu LTS 22.0
      GPU : Intel HD
      Processor : Intel Core i5 Gen 8
      RAM : 8 GB
      SSD : 256 GB
   3. Laptom *Attacker*
      Sistem Operasi : Linux / Ubuntu LTS 22.0
      GPU : Nvidia Geforce CUDA M GT 720 2gb
      Processor : Intel Core i3 Gen 3
      RAM : Corsair 16 gb
      SSD : ADATA 1 TB
   4. Komputer Sniffer
      Sistem Operasi : Linux / Ubuntu LTS 22.0
      GPU : Intel HD
      Processor : Intel Core i5 Gen 8
      RAM : 16 GB
      SSD : ADATA 1 TB
      External LAN Card dengan 6 interfaces
   5. Router
   6. Switch

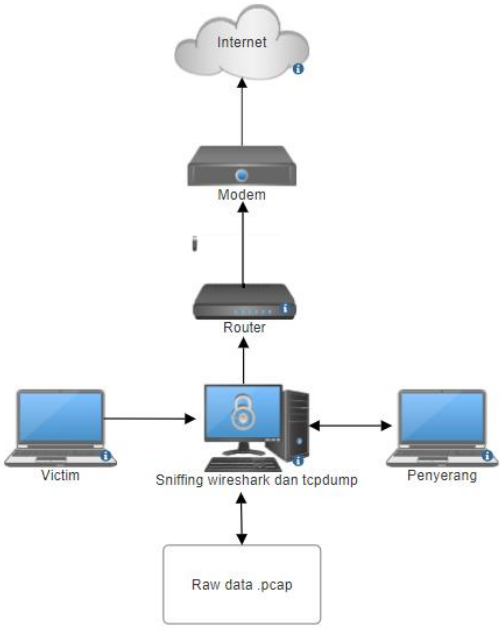
2.4 Sumber Data

Dalam penelitian ini data yang diperoleh oleh penulis merupakan data primer dan data sekunder, sebagai berikut:

a. Data Primer

Pengambilan data dilakukan secara langsung di Lab Ubiquitos selama 1 bulan 5 hari kerja. Data yang diperoleh yaitu adalah data dengan *file .PCAP* kemudian dikonversi menjadi data berbentuk .CSV. Adapun alur pengambilan data yang dilakukan terdapat dua tahap yaitu perancangan topologi jaringan dan *dumping* data.

1. Topologi jaringan

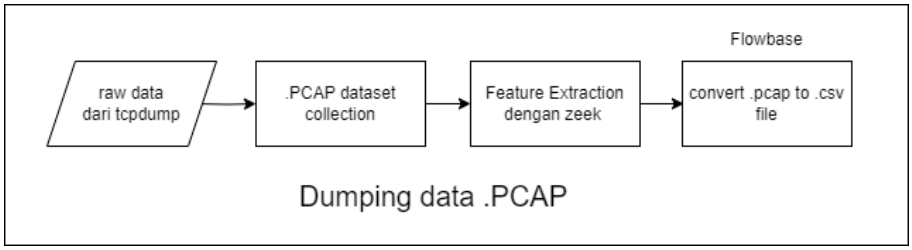


Gambar 7. Rancangan Topologi

Gambar 7 adalah proses untuk menangkap data pada lalu lintas jaringan yang terhubung dengan jaringan yang ada pada jaringan lokal di “Lab Ubiquitos” dan jaringan luar “Fakultas Teknik Universitas Hasanuddin”. Pada penelitian yang dilakukan melakukan sniffing dengan menggunakan Wireshark dan *Tcpdump* untuk memantau aktivitas yang dilakukan victim dan penyerang kemudian data jaringan yang ditangkap dari hasil sniffing dalam bentuk *.PCAP* bisa digunakan untuk melihat paket-paket yang ada didalam *.PCAP*. Data ini bisa digunakan untuk mencari pola serangan, mengidentifikasi paket berbahaya.

2. *Dumping* data

Pada tahap ini, data *raw* data dari *Tcpdump* yang telah ditangkap kemudian, diproses lebih lanjut untuk disimpan sebagai *dataset* untuk diproses kedalam *machine learning*. Kemudian *dataset* yang telah dikumpulkan dianalisis menggunakan Zeek. Zeek digunakan untuk melakukan fitur ekstraksi untuk menghasilkan fitur - fitur yang penting dari lalu lintas jaringan seperti *Flow, Subflow, Bulk Transmission, dan Definition of Active and Idle Time*. Setelah fitur ekstraksi menggunakan zeek kemudian *file .PCAP* di konversi kedalam bentuk *file* .csv. Data yang diambil adalah data yang berbasis aliran. Pada Gambar 8 adalah proses *dumping* data.



Gambar 8. *Dumping* data *.PCAP*

3. *Labeling* data untuk *groundtruth*

Pada tahap ini, data yang telah diubah ke format .csv, kemudian memberikan label *benign* ataupun *malicious* terhadap data, selain memberikan label biner. Tujuan dari label ini untuk mengevaluasi hasil dari label yang dihasilkan dari metode *DBSCAN* dan *K–Means* untuk melihat *accuracy* dari label yang dihasilkan oleh *DBSCAN* dan *K–Means*, seperti yang dilakukan oleh (Sarossy, 2021). Adapun beberapa cara untuk melabel data tersebut:

1. Mengecek weird.log

   Pada *file* weird.log mencatat aktivitas jaringan yang dianggap anomali atau yang menyimpang dari pola yang diharapkan termasuk perilaku yang tidak sesuai dengan standar protokol atau norma jaringan. Zeek mencatat semua anomali ini ke dalam weird.log, sehingga *file* ini memberikan informasi penting tentang insiden atau potensi masalah di jaringan. Setiap entri dalam weird.log berfungsi sebagai label yang dapat digunakan sebagai ground-truth dalam analisis atau pelatihan model.

2. IP proxy dari *attacker*

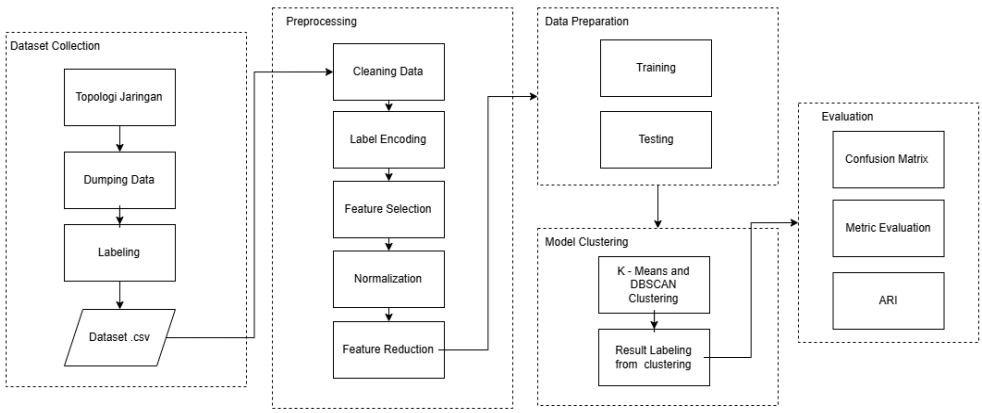
   Pada penelitian ini melihat Ip Proxy yang dipakai oleh *attacker* yang telah ditandai sebelumnya untuk dijadikan sebagai label pada penelitian yang dilakukan.

3. Mengecek secara manual

Pada penelitian ini memerikasa setiap “flow” secara individual. Fitur-fitur yang perlu diperhatikan seperti "flow_duration", "originp", "originh", "responp", "responh", dan "protocol", serta ukuran payload jika diperlukan. Pada fitur “flow_duration” menjadi indikator untuk melihat aktivitas di jaringan. Informasi tentang alamat IP dan port asal ("originp", "originh") serta alamat IP dan port tujuan ("responp", "responh") memberikan konteks penting terkait komunikasi antar node dalam jaringan. Konteks ini diperoleh melalui analisis terhadap port dan layanan yang terlibat dalam "flow" tersebut. Ada kondisi di mana "originp" dan "responp" berasal dari port yang tidak secara langsung terkait. Hal tersebut dapat dilihat dengan referensi website https://speedguide.net/.

2.5 Perancangan Sistem

Pada tahap ini dijelaskan secara rinci perancangan sistem *machine learning*, sesuai dengan alur penelitian yang ada pada Gambar 5. Perancangan dari sistem *machine learning* dapat dilihat pada Gambar 9.



Gambar 9. Alur Rancangan Sistem

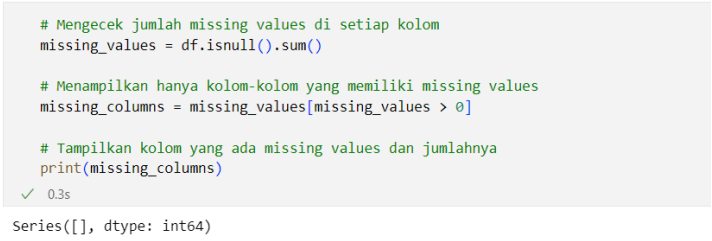
2.5.1 Input *Dataset*

Dataset yang telah dikumpulkan dari proses *dumping*. *Dataset* yang di *dumping* merupakan *dataset* lalu lintas pada jaringan terenkripsi. Data yang diambil adalah data yang berbasis aliran. Aliran ini merupakan kumpulan paket-paket data yang memiliki ciri-ciri khusus dan dianggap sebagai sesi komunikasi antara dua perangkat dalam jaringan. Pada penilitian jumlah data pada *dataset* sebanyak 510177 jumlah data untuk digunakan di model *clustering*. Pada *dataset* jumlah data untuk data yang bernilai *malicious* sebanyak 153725 dan data *benign* sebanyak 356452 yang digunakan sebagai *groundtruth* untuk hasil label *clustering*.

2.5.2 *Preprocessing*

a. *Cleaning* Data

Pada tahap ini mengecek data yang missing values, lalu membersihkan jika terdapat missing values, yang dapat dilakukan dengan menghapus data tersebut jika jumlahnya lebih sedikit atau jika tidak terlalu berpengaruh ke *dataset*. Kemudian menghilangkan duplikat data karena dapat mempengaruhi model *machine learning* yang digunakan jika terdapat nilai yang duplikat, menghilangkan kolom yang tidak di perlukan. Pada *dataset* yang digunakan tidak terdapat data *missing values* dan data duplikat seperti yang terlihat pada Gambar 10 dan Gambar 11.

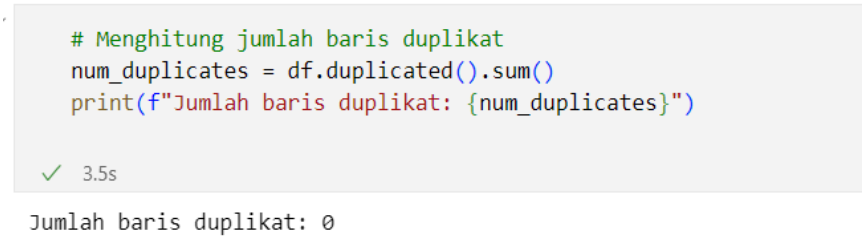


```
# Mengecek jumlah missing values di setiap kolom
missing_values = df.isnull().sum()

# Menampilkan hanya kolom-kolom yang memiliki missing values
missing_columns = missing_values[missing_values > 0]

# Tampilkan kolom yang ada missing values dan jumlahnya
print(missing_columns)
```

✓ 0.3s

```
Series([], dtype: int64)
```

Gambar 10. Kolom yang memiliki missing values



```
# Menghitung jumlah baris duplikat
num_duplicates = df.duplicated().sum()
print(f"Jumlah baris duplikat: {num_duplicates}")
```

✓ 3.5s

```
Jumlah baris duplikat: 0
```

Gambar 11. Kolom yang memiliki data duplikat

Pada Gambar 12 menghapus 3 fitur pada *dataset* karena kolom-kolom tersebut memiliki jumlah nilai unik yang sama dengan jumlah total baris dalam *dataset*. Ini menunjukkan bahwa setiap baris dalam kolom tersebut memiliki nilai yang berbeda satu sama lain (tidak ada duplikasi), sehingga tidak memberikan informasi yang berguna untuk analisis atau model *machine learning*. Fitur seperti ini sering kali tidak relevan untuk tujuan prediksi karena tidak dapat membantu membedakan antara kategori yang berbeda.

```
filtered_columns = df.columns[df.nunique() == 510177]

print(filtered_columns)
```

✓ 1.3s

```
Index(['Unnamed: 0.1', 'Unnamed: 0', 'uid'], dtype='object')
```

Gambar 12. Kolom yang memiliki nilai unik 510177

b. *Label encoding*

Pada tahap ini melakukan *label encoding*. Tujuan dari teknik *label encoding* adalah untuk mengonversi kategori nonnumerik ke dalam angka numerik sehingga model pembelajaran mesin dapat memprosesnya. Setiap nilai kategori yang berbeda diberikan angka unik. *Label encoding* mengganti nilai-nilai kategori dengan angka unik, tapi tetap mempertahankan informasi setiap kategori. Fitur-fitur yang memiliki tipe data nonnumerik pada *dataset* dapat dilihat pada Gambar 13.

```
Index(['originh', 'responh', 'protocol', 'Target', 'Activity'], dtype='object')
```

Gambar 13. Fitur-fitur dengan tipe data nonnumerik

Fitur-fitur nonnumerik kemudian dikonversi menjadi tipe data numerik agar dapat diproses oleh *feature selection* menggunakan *Information gain*. Tipe data setelah diproses dengan teknik *label encoding* dapat dilihat pada Gambar 14.

```
0   time_spent          510177 non-null  float64
1   originh             510177 non-null  int32
2   originp             510177 non-null  int64
3   responh             510177 non-null  int32
4   responp             510177 non-null  int64
5   protocol            510177 non-null  int32
```

Gambar 14. Fitur-fitur setelah *label encoding*

Pada fitur ‘Target’ menggunakan *label encoding* manual dengan mengubah nilai dari Target menjadi numerik dengan menentukan nilai masing-masing dari setiap data. Pada Gambar 15 merupakan proses mengubah tipe data dari fitur ‘Target’.

```
df.replace({'Benign':0, 'Malicious':1}, inplace=True)
```

✓ 0.3s

Gambar 15. Fitur-fitur setelah *label encoding* manual

c. *Feature selection*

Pada tahap ini melakukan pengurangan fitur dalam *dataset*. Tujuan dari tahap ini adalah untuk menyederhanakan *dataset*, tetapi tetap mempertahankan karakteristik pentingnya untuk meningkatkan kinerja dan performa dari *machine learning*. Pada tahap ini menggunakan metode *information gain* untuk menentukan fitur yang paling relevan berdasarkan perhitungan bobot fitur. Fitur yang tidak relevan akan dieliminasi agar meningkatkan performa dari sistem deteksi *machine learning*. Pada penelitian yang dilakukan menggunakan *information gain*, menentukan bobot fitur berdasarkan dari hasil rata-rata bobot pada *information gain*. Adapun langkah-langkah dalam menentukan bobot dari setiap fitur menggunakan *information gain*.

1. Menentukan nilai *entropi* dari target (Y):

Tabel 1. Kode menentukan entropi

```
# Fungsi untuk menghitung entropi
def entropi(y):
    # Menghitung frekuensi dari setiap kelas
    p_values = np.bincount(y) / len(y)
    # Menghitung entropi berdasarkan p(y_i)
    return -np.sum([p * np.log2(p) for p in
p_values if p > 0])
```

Pada Tabel 1 menghitung entorpy pada nilai target Y. Pada langkah pertama adalah menghitung frekuensi dari setiap kelas pada target y. Kemudian menghitung nilai entropi dari setiap kelas pada target y.

2. Menentukan nilai kondisional entropi.

Tabel 2. Kode menentukan kondisional entropi

```
# Fungsi untuk menghitung kondisional entropi
def conditional_entropy(X, y):
    unique_X = np.unique(X)
    cond_entropy = 0.0

    for x_val in unique_X:
# Mendapatkan subset dari y ketika X == x_val
        subset_y = y[X == x_val]
# Menghitung probabilitas p(x_i)
        p_x = len(subset_y) / len(y)
# Entropi dari subset y | X = x_val
        entropi_subset = entropi(subset_y)
# Menambahkan weighted entropi ke conditional entropi
        cond_entropy += p_x * entropi_subset
```

```
    return cond_entropy
```

Pada Tabel 2 merupakan Fungsi *conditional_entropy*(X, y) menghitung kondisional entropi dari variabel target y berdasarkan input X, yang mengukur ketidakpastian pada y setelah mengetahui nilai X. Fungsi ini pertama-tama mengidentifikasi nilai unik dalam X menggunakan np.unique(X) dan menginisialisasi kondisional entropi sebagai 0.0. Untuk setiap nilai unik dalam X, subset dari y yang sesuai diambil, probabilitas kemunculan nilai tersebut (p_x) dihitung, dan entropi subset dihitung menggunakan fungsi *entropi*. Nilai entropi ini kemudian dikalikan dengan probabilitas p_x dan ditambahkan ke kondisional entropi total. Setelah semua nilai unik diproses, fungsi mengembalikan kondisional entropi yang menunjukkan ketidakpastian pada y setelah mempertimbangkan informasi dari X.

3. Menentukan nilai *information gain*

Tabel 3. Kode menentukan nilai *information gain* setiap fitur

```
# Fungsi untuk menghitung information gain
def information_gain(X, y):
    # Entropi   split
    initial_entropy = entropi(y)
    # Entropi setelah mengetahui nilai X
    cond_entropy = conditional_entropy(X, y)
    # information gain adalah selisih antaraentropi
awal dan conditional entropi
 return initial_entropy - cond_entropy
```

Pada Tabel 3 menunjukkan hasil dari perhitungan pada langkah-langkah sebelumnya kemudian dikurangkan untuk memperoleh nilai *information gain* hasil dari fungsi ini menunjukkan seberapa besar informasi yang diperoleh tentang y dari X.

d. *Standardization*

Pada tahap ini mengubah nilai-nilai dalam *dataset* sehingga berada dalam rentang tertentu, biasanya antara 0 dan 1. Proses ini penting dalam banyak algoritma *machine learning* dan analisis data, karena membantu dalam mengurangi bias dan meningkatkan performa model. Pada penilitian ini menggunakan *StandardScaler* sebagai metode dalam *standardization.* Berikut merupakan hasil dari data 10 fitur pertama setelah melakukan *standardization* pada Tabel 4.

Tabel 4. Kode *StandardScaler*

```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(X)
```

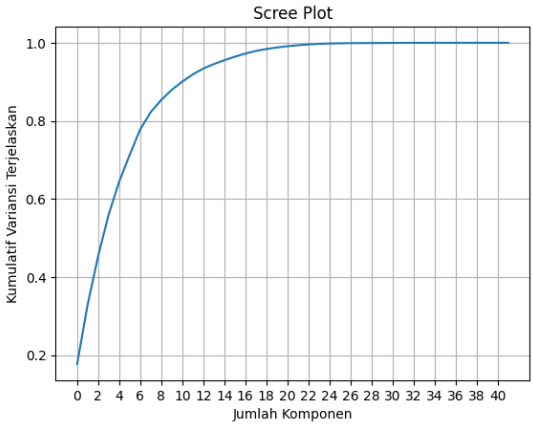
e. *Feature reduction*

Pada tahap ini melakukan pengurangan fitur dalam *dataset*. Tujuan dari tahap ini adalah untuk menyederhanakan *dataset*, tetapi tetap mempertahankan karakteristik pentingnya untuk meningkatkan kinerja dan performa dari *machine learning*. Pada penelitian ini menggunakan *PCA* (Principal Component Analysis). Potongan kode dari *PCA* dapat dilihat pada Tabel 5.

Tabel 5. Kode *PCA*

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Data harus sudah distandarisasi
PCA = PCA().fit(scaled_data)

# Melihat variansi yang dijelaskan oleh setiap komponen
explained_variance = PCA.explained_variance_ratio_

# Plot variansi kumulatif
plt.plot(np.cumsum(explained_variance))
plt.xticks(np.arange(0, 42, 2))
plt.xlabel('Jumlah Komponen')
plt.ylabel('Kumulatif Variansi Terjelaskan')
plt.title('Scree plot')
plt.grid(True)
plt.show()
```

Pada penelitian ini memilih 12 komponen, didasarkan pada analisis visual dari *scree plot*, yang menunjukkan distribusi variansi yang terjelaskan oleh masing-masing komponen. Berdasarkan *scree plot*, variansi kumulatif yang terjelaskan mencapai titik optimal pada komponen ke-12, setelah itu penambahan komponen berikutnya tidak memberikan peningkatan signifikan dalam variansi yang terjelaskan. Oleh karena itu, pada penelitian ini memutuskan untuk menggunakan 12 komponen utama dalam analisis ini, yang dianggap cukup untuk menangkap sebagian besar informasi penting dalam data tanpa menambahkan kompleksitas yang berlebihan. Dapat dilihat pada Gambar 16.



Gambar 16. *Scree plot* dalam menentukan jumlah komponen utama

f. Split *dataset*

Pada tahap persiapan data, dilakukan pembagian *dataset* menjadi dua bagian, yaitu *train data* dan testing data, dengan proporsi 80% untuk *train data* dan 20% untuk testing data. Pada tahap ini menguji model *clustering* dengan data test.

2.5.3 Model *Clustering*

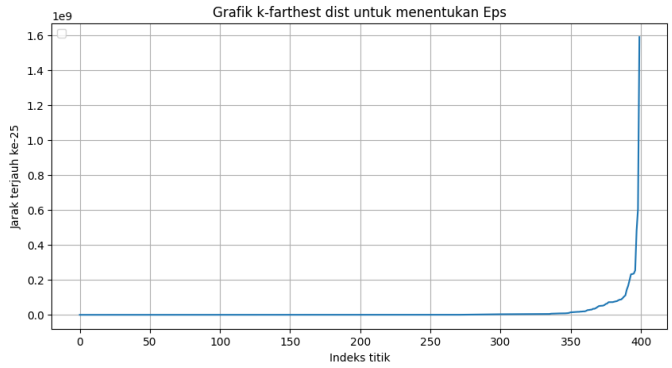
a. *DBSCAN*

Pada tahap ini setelah melakukan *preprocessing* dan data preparation, selanjutnya memasukkan hasil dari data split data yaitu data training dan data testing untuk menguji model yang dibuat. Pada tahap model dengan *DBSCAN* memerlukan dua parameter yang akan digunakan yaitu MinPts dan epsilon. Dalam menentukan parameter *DBSCAN*, untuk parameter MinPts dapat menggunakan rumus:

$$MinPts = 2 * dimensi\ data - 1 \qquad (10)$$

Dimensi data pada rumus diatas diartikan sebagai jumlah fitur dalam *dataset*. Pada penelitian ini jumlah fitur yang digunakan akan diambil dari nilai *PCA* yaitu sebanyak 12 fitur, sehingga pada penelitian ini nilai dari MinPts berjumlah 2 * 12 - 1 = 23.

Pada algoritma *DBSCAN* perlu untuk menentukan nilai dari epsilon untuk menentukan radius lingkungan di sekitar titik data, yang membantu dalam mengidentifikasi titik-titik yang berada di dalam sebuah *cluster*. Pada penelitian ini untuk menentukan nilai epsilon menggunakan *k-distance* plot yang merupaka metode umum dalam *DBSCAN*. Dari hasil *k-distance* plot pada Gambar 17 nilai epsilon berada di disekitar 0.6, karena ada kenaikan tajam padat titik -titik tersebut pada sumbu Y, yang menandai elbow point.



Gambar 17. Grafik *k-farthest distance* untuk menentukan epsilon

Setelah menentukan kedua parameter dalam *DBSCAN*, selanjutnya memasukkan nilai tersebut kedalam model *DBSCAN* yang digunakan. Pada Gambar 18 merupakan algoritma dari *DBSCAN* yang digunakan.

```
DBSCAN(D, eps, MinPts):
    C = 0
    for each unvisited point P in dataset D:
        mark P as visited
        NeighborPts = regionQuery(P, eps)
        if sizeof(NeighborPts) < MinPts:
            mark P as NOISE
        else:
            C = next cluster
            expandCluster(P, NeighborPts, C, eps, MinPts)

expandCluster(P, NeighborPts, C, eps, MinPts):
    add P to cluster C
    for each point P' in NeighborPts:
        if P'.status == 'unvisited':
            mark P' as visited
            NeighborPts' = regionQuery(P', eps)
            if len(NeighborPts') >= MinPts:
                NeighborPts = NeighborPts + NeighborPts'
        if P' not in any cluster:
            add P' to cluster C

regionQuery(P, eps):
    neighbors = []
    for each point Q in dataset:
        if euclideanDistance(P, Q) <= eps:
            neighbors.append(Q)
    return neighbors

euclideanDistance(P, Q):
    return sqrt((P.x - Q.x)^2 + (P.y - Q.y)^2)
```

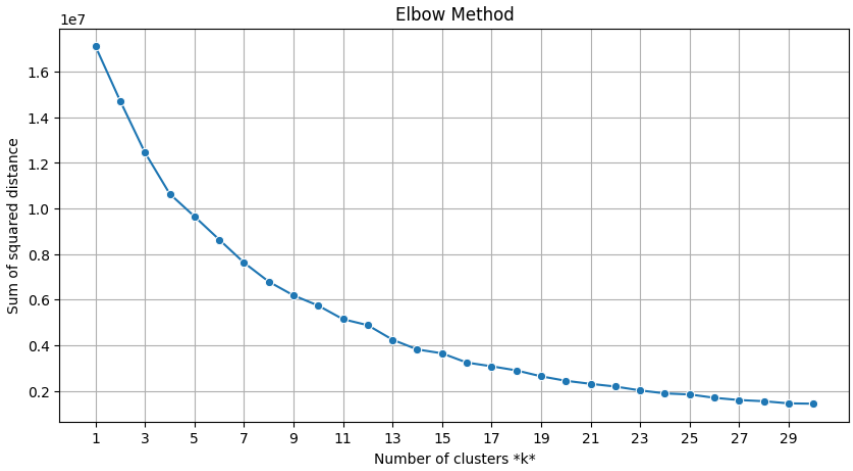
Gambar 18. Algoritma *DBSCAN*

Berdasarkan Gambar 18 penjelasan dari algoritma *DBSCAN* pada penelitian ini sebagai berikut:

a. Masukkan nilai *dataset* yang akan dikelompokkan dan juga nilai MinPts dan epsilon.
b. Setelah memasukkan nilai parameter dari *DBSCAN*. *DBSCAN* dimulai dengan memilih titik data yang belum dikunjungi dan menandainya sebagai telah dikunjungi. Algoritma kemudian menggunakan fungsi regionQuery untuk mencari tetangga dalam radius epsilon. Jika jumlah tetangga kurang dari MinPts, titik tersebut dianggap sebagai noise atau outlier dan tidak dimasukkan ke dalam *cluster*.
c. Jika sebuah titik memiliki tetangga yang cukup sesuai MinPts, titik tersebut menjadi core point dari *cluster* baru. Fungsi expand*Cluster* kemudian memperluas *cluster* dengan menambahkan tetangga dan tetangga dari tetangga hingga semua titik yang memenuhi kepadatan dimasukkan.
d. *DBSCAN* menggunakan jarak Euclidean untuk mengukur kedekatan antara dua titik. Jarak ini menentukan apakah suatu titik berada dalam radius epsilon dari titik lainnya dan dapat dianggap tetangga, membantu dalam proses pengelompokan data.

b. *K-Means*

Pada penelitian ini menggunakan model lain seperti *K-Means* untuk dibandingkan dengan algoritma *DBSCAN* untuk melihat performa antara dua algoritma dan melihat hasil evaluasi. Pada *K – Means* perlu untuk menentukan nilai k (jumlah *cluster*). Pada penelitian ini menggunakan *elbow method* untuk melihat siku pada grafik inertia pada *cluster* yang terbentuk. *Elbow method* yang terbentuk dapat dilihat pada Gambar 19, nilai *elbow method* yang terlihat pada *cluster* 9, dan digunakan seebagai paramater untuk menenetukan jumlah *cluster*.



Gambar 19. *Elbow method*

```
def initialize_centroids(data, k):
    centroids = []
    for i = 1 to k:
        centroids.append(random.choice(data))
    return centroids

def calculate_distance(data_point, centroid):
    sum = 0
    for i = 1 to n:
        sum += (data_point[i] - centroid[i])^2
    return sqrt(sum)

def assign_clusters(data, centroids):
    clusters = {}
    for each data_point in data:
        min_distance = infinity
        cluster = None
        for each centroid in centroids:
            distance = calculate_distance(data_point, centroid)
            if distance < min_distance:
                min_distance = distance
                cluster = centroid
        if cluster not in clusters:
            clusters[cluster] = []
        clusters[cluster].append(data_point)
    return clusters

def update_centroids(clusters):
    new_centroids = []
    for cluster in clusters:
        new_centroid = calculate_mean(clusters[cluster])
        new_centroids.append(new_centroid)
    return new_centroids

def has_converged(old_centroids, new_centroids):
    return old_centroids == new_centroids

def kmeans(data, k):
    centroids = initialize_centroids(data, k)
    while True:
        clusters = assign_clusters(data, centroids)
        new_centroids = update_centroids(clusters)
        if has_converged(centroids, new_centroids):
            break
        centroids = new_centroids
    return centroids, clusters
```

Gambar 20. Algoritma *K - Mean*s

Berdasarkan Gambar 20 penjelasan dari algoritma *K-Means* pada penelitian ini sebagai berikut:

a. Pada langkah pertama, k titik data dipilih secara acak dari *dataset* sebagai *centroid* awal. *Centroid* ini akan digunakan sebagai titik pusat untuk memulai proses *clustering*. Pemilihan secara acak ini bertujuan untuk memberikan titik referensi awal yang akan digunakan untuk mengelompokkan data.
b. Pada setiap iterasi, jarak setiap titik data ke semua *centroid* dihitung. Jarak ini digunakan untuk menentukan titik data mana yang lebih dekat ke *centroid* tertentu. Data kemudian dimasukkan ke dalam *cluster* yang memiliki *centroid* dengan jarak terdekat. Dalam hal ini, jarak diukur menggunakan *Euclidean Distance*.
c. Setelah semua data dikelompokkan ke dalam *cluster* masing-masing, posisi *centroid* diperbarui. *Centroid* baru dihitung sebagai rata-rata dari semua titik data dalam *cluster* tersebut. Dengan menghitung rata-rata ini, posisi *centroid* akan bergerak lebih dekat ke pusat dari titik-titik data di *cluster*, sehingga *cluster* menjadi lebih representatif terhadap distribusi data.

d. Setelah pembaruan *centroid*, algoritma melakukan pengecekan untuk melihat apakah ada perubahan antara *centroid* lama dan *centroid* baru. Jika tidak ada perubahan yang signifikan, artinya posisi *centroid* telah stabil dan algoritma akan berhenti. Tahap ini disebut konvergensi, di mana *cluster* dianggap telah terbentuk secara optimal.
e. Jika *centroid* belum konvergen, algoritma akan mengulangi langkah-langkah dari penetapan *cluster* hingga pembaruan *centroid*. Proses ini terus berulang hingga *centroid* tidak mengalami perubahan yang berarti atau perubahannya sangat kecil, menandakan bahwa proses *clustering* telah selesai.

c. Analisis *Cluster*.

Dalam penelitian ini, menganalisis karakteristik setiap *cluster* untuk menentukan aktivitas yang terjadi di setiap *cluster* dan menentukan apakah dia termasuk label *benign* ataupun *malicious*.

2.5.4 Analisis Hasil Evaluasi

Pada tahap ini dilakukan dengan melihat hasil evaluasi *dataset* dilakukan dengan menggunakan *confusion matrix* sebagai dasar untuk menghitung metrik evaluasi seperti *accuracy*, *precision*, *recall*, dan *F1-Score* pada algoritma *cluster*isasi *DBSCAN* dan *K-Means*. *Confusion matrix* memungkinkan identifikasi jumlah prediksi benar dan salah dalam kategori positif dan negatif, yang kemudian digunakan untuk menghitung *accuracy*, *precision*, *recall*, dan *F1-Score*. Kemudian membandingkan kinerja *DBSCAN* dan *K-Means* dalam mendeteksi aktivitas *malicious* pada data lalu lintas jaringan terenkripsi.