BAB I
PENDAHULUAN

1.1 LATAR BELAKANG

Deteksi intrusi adalah proses pemantauan kejadian yang terjadi pada sistem jaringan komputer dan menganalisanya tanda-tanda dari insiden yang dimungkinkan terjadi. Deteksi intrusi terutama difokuskan pada identifikasi insiden yang mungkin terjadi, mencatat insiden yang terjadi, dan melaporkannya kepada administrator sistem.(NIST, 2007)

Denial Of Service (DoS) telah menjadi ancaman serius terhadap keamanan jaringan akhir-akhir ini. Serangan jenis ini telah terbukti mampu membuat banyak *server down* karena harus melayani banyaknya *request* yang masuk. Situs besar seperti *Wordpress* dan beberapa situs besar lainnya sudah menjadi korban dari serangan *DoS*. Serangan *DoS* sendiri terdiri dari berbagai macam teknik seperti *UDP Flood* dimana teknik ini membuat paket *ICMP* atau *UDP* dalam jumlah yang sangat besar sehingga menghabiskan *bandwith* dari jaringan korban. Teknik lainnya yang sangat berbahaya adalah *syn flood*, dimana penyerang membanjiri korban dengan *request connection* sehingga *server* tidak mampu menangani *request* yang sangat besar. Dalam banyak kasus penyerang mampu membuat *server* dalam kondisi *Denial of Service* dengan teknik ini (*syn flood*). Sekitar 90% Serangan *DoS* disebabkan oleh serangan *syn flood*.(Moore;Volker, 2001).

Paket yang digunakan oleh *syn flood* tidak berbeda dengan paket *TCP SYN* normal kecuali pada alamat pengirim yang palsu, hal ini menyebabkan sulitnya membedakan trafik *syn flood* dengan paket *TCP SYN* normal pada *server* target. Serangan *syn flood* muncul ketika klien tidak merespon *SYN/ACK* dari *server* sehingga layanan berhenti sampai waktunya habis. Pada tipe serangan ini klien tidak pernah merespon karena alamat asalnya dipalsukan. Sasaran penyerang adalah mengirim paket *SYN* kepada *server* dengan lebih cepat daripada waktu yang diperlukan *server* untuk berhenti menunggu respon *ACK* dari klien sehingga menyebabkan *server* begitu sibuk untuk menunggu klien yang tidak menjawab *SYN/ACK server*.

Beberapa metode dikembangkan untuk melakukan pendeteksian pada serangan *syn flood.* Yuichi Ohsita dan Shingo ATA melakukan pendeteksian serangan *syn flood* dengan menganalisa paket *TCP SYN* secara statistik dimana hasil dari penelitian menghasilkan kesimpulan

bahwa algoritma tersebut bisa melakukan pendeteksian dengan 100 *sample SYN rates*. Algoritma lain adalah *EM* (*Expectation Maximization*) yang digunakan untuk mendeteksi *syn flood* oleh Animesh Patcha dan Jung-Min Park dengan tingkat akurasi pendeteksian mendekati 80 persen.

Sebelum diimplementasikan ke dalam jaringan *online* atau *real-time*, implementasi algoritma dilakukan secara *offline.* Ketika penelitian yang dilakukan sesuai dengan hasil yang diharapkan maka selanjutnya dipikirkan mekanisme untuk membawa algoritma tersebut untuk menjadi sebuah produk yang bisa digunakan secara *real-time*. Penelitian secara *offline* banyak dilakukan karena selain untuk menguji algoritma yang digunakan adalah untuk menghindari faktor-faktor lain diluar algoritma tersebut yang mempengaruhi hasil penelitian seperti kemampuan kartu jaringan, perangkat koneksi, kemampuan *hardware*, dan faktor-faktor lainnya sehingga diharapkan hasil pengujian algoritma merupakan hasil murni tanpa ada pengaruh faktor lain yang terjadi pada sistem *real-time.*Salah satu penelitian yang dilakukan secara *offline* adalah *Detecting Novel Network Intrusions Using Bayes Estimators* (Barbara,Wu,Jajodia, 2000). Penelitian ini menggunakan data koneksi jaringan komputer dari *Darpa Intrusion Detection Evaluation* (*DARPA*, *1998*) yang terdiri dari dua tipe data, yaitu data latih dan data uji. Data latih terdiri dari serangan jaringan komputer dalam tujuh minggu yang terdapat pada data jaringan komputer normal. Penelitian ini mengklasifikasikan serangan *DoS* yang memanfaatkan protokol *UDP* seperti *smurf*, *teardrop*, dan *pod.* Total ada 15 serangan yang dideteksi pada penelitian ini dengan akurasi pendeteksian pada sebagian besar serangan adalah 100% namun pada beberapa serangan yang lain akurasi pendeteksian sangat buruk yaitu dibawah 50%.

Algoritma *naïve bayes* secara masal dikembangkan oleh banyak peneliti untuk melakukan pendeteksian *spam* pada *e-mail*. Secara garis besar algoritma ini melakukan penghitungan terhadap *e-mail* berdasarkan beberapa atribut tertentu untuk menentukan *e-mail* yang masuk apakah *spam* atau bukan. Algoritma n*aïve bayes* sendiri adalah sebuah algoritma pengklasifikasi secara probabilitas berdasarkan pada teorema *Bayes* dengan tidak adanya keterkaitan antar atribut. *Naïve bayes* banyak digunakan karena tingkat akurasi dalam mendeteksi *spam* cukup bagus, seperti penelitian yang dilakukan oleh Andy Baskara dengan judul *Klasifikasi Spam Email Menggunakan Metode Pendekatan*

Naive Bayes. Pada penelitian tersebut rata-rata *spam precission* yang dihasilkan adalah 88,88%.

Berdasarkan pemaparan diatas serta akurasi dari algoritma *naïve bayes* yang cukup baik dan dirasa mampu untuk melakukan pendeteksian serangan *syn flood* maka disusunlah skripsi dengan judul ***Pendeteksian Serangan Syn Flood Pada Jaringan Komputer Lokal Menggunakan Algoritma Naïve Bayes.***

1.2 RUMUSAN MASALAH

Dari latar belakang yang telah disebutkan diatas, maka ditarik rumusan masalah sebagai berikut :

- Bagaimana mengaplikasikan algoritma *naïve bayes* untuk melakukan pendeteksian serangan *syn flood* pada jaringan komputer lokal.
- Bagaimana menganalisa keakuratan perangkat lunak dalam mendeteksi serangan pada jaringan komputer dan faktor apakah yang mempengaruhi akurasi algoritma *naïve bayes* dalam melakukan pendeteksian serangan jaringan komputer lokal.

1.3 TUJUAN

Tujuan dari penelitian ini adalah membuat perangkat lunak dengan menggunakan algoritma *naïve bayes* untuk melakukan pendeteksian serangan *syn flood* pada jaringan komputer lokal.

1.4 BATASAN MASALAH

Batasan Masalah dari penelitian ini adalah

- Menggunakan Protokol *HTTP* port 80 pada sisi *server*.
- *Flooding* menggunakan koneksi *TCP* di port 80. .
- Pendeteksian dilakukan secara *offline*
- Banyaknya koneksi untuk *pooling* diabaikan.

1.5 MANFAAT

Manfaat yang dapat diambil dari penelitian ini adalah mendapatkan *Tool* yang bisa melakukan pendeteksian *Denial of Service syn flood* dengan Algoritma n*aïve bayes* untuk mempermudah pekerjaan *Network Administrator* dalam menghadapi serangan *DoS*.

1.6 METODE PENYELESAIAN MASALAH

Metode penyelesaian masalah yang dilakukan pada penelitian ini, yaitu :

- Studi Literatur

Membaca dan mempelajari beberapa literatur (jurnal, buku, dan artikel dari website) mengenai Jaringan Komputer, (*Transmission Control Protocol*) *TCP, Denial Of Service*, *syn flood*. Dan Algoritma *Naïve Bayes*.

- Perancangan dan implementasi perangkat lunak dan sistem

Merancang dan membangun sebuah model jaringan, Melakukan simulasi *Denial of Attack*, dan Perangkat lunak *Berbasis Python* yang berfungsi untuk melakukan pendeteksian *SynFlood*.

- Uji Coba dan analisis hasil implementasi

Menganalisis hasil implementasi, yaitu hasil dari *capture* yang dilakukan perangkat lunak dianalisis apakah tingkat akurasinya sesuai dengan yang diharapkan.

BAB II
TINJAUAN PUSTAKA

2.1 *TRANSMISSION CONTROL PROTOCOL*

Transmission Control Protocol (*TCP*) adalah sebuah protokol komunikasi *reliable end to end* dan berorientasi koneksi yang didesain untuk protokol layer memberikan dukungan untuk aplikasi multi-jaringan. Pada prinsipnya *TCP* mampu beroperasi pada sistem jaringan yang luas dengan menggunakan *packet-switched network. TCP* memiliki karakteristik sebagai berikut

- Berorientasi sambungan (*connection-oriented*) : Sebelum data dapat ditransmisikan antara dua host, dua proses yang berjalan pada lapisan aplikasi harus melakukan negosiasi untuk membuat sesi koneksi terlebih dahulu. Koneksi *TCP* ditutup dengan menggunakan proses terminasi koneksi *TCP* (*TCP connection termination*).
- Dapat diandalkan (reliable): Data yang dikirimkan ke sebuah koneksi *TCP* akan diurutkan dengan sebuah nomor urut paket dan akan mengharapkan paket *positive acknowledgment* dari penerima. Jika tidak ada paket *acknowledgment* dari penerima, maka segmen *TCP* (protocol data unit dalam protokol *TCP*) akan ditransmisikan ulang. Pada pihak penerima, segmen-segmen duplikat akan diabaikan dan segmen-segmen yang datang tidak sesuai dengan urutannya akan diletakkan di belakang untuk mengurutkan segmen-segmen *TCP*. Untuk menjamin integritas setiap segmen *TCP*, *TCP* mengimplementasikan penghitungan *TCP Checksum*.
- *Byte stream*: *TCP* melihat data yang dikirimkan dan diterima melalui dua jalur masuk dan jalur keluar *TCP* sebagai sebuah byte stream yang berdekatan (kontigu). Nomor urut *TCP* dan nomor *acknowlegment* dalam setiap header *TCP* didefinisikan juga dalam bentuk *byte*, meski demikian, *TCP* tidak mengetahui batasan pesan-pesan di dalam byte stream *TCP* tersebut. Untuk melakukannya, hal ini diserahkan kepada protokol lapisan aplikasi dalam *DARPA Reference Model*, yang harus menerjemahkan *byte stream TCP* ke dalam "bahasa" yang dipahami.

- Melakukan segmentasi terhadap data yang datang dari lapisan aplikasi dalam *DARPA Reference Model*.
- Mengirimkan paket secara "*one-to-one*": Hal ini karena memang *TCP* harus membuat sebuah sirkuit logis antara dua buah protokol lapisan aplikasi agar saling dapat berkomunikasi. *TCP* tidak menyediakan layanan pengiriman data secara *one-to-many.*

TCP memiliki segmen yang dikirimkan sebagai datagram-datagram IP, datagram merupakan satuan protokol unit data pada lapisan *internetwork*. Sebuah segmen *TCP* terdiri atas sebuah *header* dan segmen data (*payload*), yang dienkapsulasi dengan menggunakan *header* IP dari protokol IP. Sebuah segmen dapat berukuran hingga 65495 dimana ukuran *header* IP terkecil (20 byte) + ukuran *header TCP* terkecil (20 byte).

Di dalam *header* IP dari sebuah segmen *TCP*, *field Source IP Address* diatur menjadi alamat *unicast* dari sebuah antarmuka host yang mengirimkan segmen *TCP* yang bersangkutan. Sementara itu, *field Destination IP Address* juga akan diatur menjadi alamat *unicast* dari sebuah antarmuka host tertentu yang dituju. Hal ini dikarenakan, protokol *TCP* hanya mendukung transmisi *one-to-one*. Ukuran dari *header TCP* adalah bervariasi, yang terdiri atas beberapa *field* yang. Ukuran *TCP header* paling kecil (ketika tidak ada tambahan opsi *TCP*) adalah 20 byte. *TCP* juga memiliki *flag* (tanda) khusus yang mengindikasikan segmen yang bersangkutan, nama *flag* dijelaskan di tabel 2-1

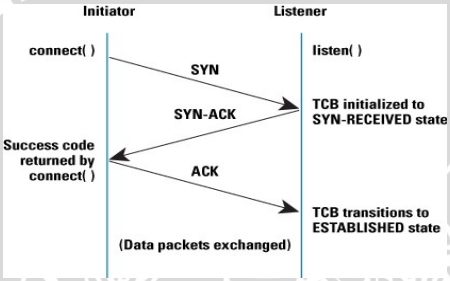
Tabel 2-1 *Flag TCP*

NAMA FLAG	KETERANGAN
URG	Mengindikasikan bahwa beberapa bagian dari segmen *TCP* mengandung data yang sangat penting, dan Field Urgent Pointer dalam header *TCP* harus digunakan untuk menentukan lokasi di mana data penting tersebut berada dalam segmen.
ACK	Mengindikasikan field Acknowledgment mengandung oktet selanjutnya yang diharapkan dalam koneksi. Flag ini selalu diset, kecuali pada segmen pertama pada pembuatan sesi koneksi *TCP*.
PSH	Mengindikasikan bahwa isi dari *TCP* Receive buffer harus diserahkan kepada protokol lapisan aplikasi. Data dalam receive buffer harus berisi

	sebuah blok data yang berurutan (kontinyu), dilihat dari ujung paling kiri dari buffe. dengan kata lain, sebuah segmen yang memiliki flag PSH diset ke nilai 1, tidak bolah ada satu byte pun data yang hilang dari aliran byte segmen tersebut; data tidak dapat diberikan kepada protokol lapisan aplikasi hingga segmen yang hilang tersebut datang. Normalnya, *TCP* Receive buffer akan dikosongkan (dengan kata lain, isi dari buffer akan diteruskan kepada protokol lapisan aplikasi) ketika buffer tersebut berisi data yang kontiyu atau ketika dalam "proses perawatan". Flag PSH ini dapat mengubah hal seperti itu, dan membuat akan *TCP* segera mengosongkan *TCP* Receive buffer.
RST	Mengindikasikan bahwa koneksi yang dibuat akan digagalkan. Untuk sebuah koneksi *TCP* yang sedang berjalan (aktif), sebuah segmen dengan flag RST diset ke nilai 1 akan dikirimkan sebagai respons terhadap sebuah segmen *TCP* yang diterima yang ternyata segmen tersebut bukan yang diminta, sehingga koneksi pun menjadi gagal. Pengiriman segmen dengan flag RST diset ke nilai 1 untuk sebuah koneksi aktif akan menutup koneksi secara paksa, sehingga data yang disimpan dalam buffer akan dibuang (dihilangkan). Untuk sebuah koneksi *TCP* yang sedang dibuat, segmen dengan flag RST aktif akan dikirimkan sebagai respons terhadap request pembuatan koneksi untuk mencegah percobaan pembuatan koneksi.
SYN	Mengindikasikan bahwa segmen *TCP* yang bersangkutan mengandung Initial Sequence Number (ISN). Selama proses pembuatan sesi koneksi *TCP*, *TCP* akan mengirimkan sebuah segmen dengan flag SYN diset ke nilai 1. Setiap host *TCP* lainnya akan memberikan jawaban (acknowledgment) dari segmen dengan flag SYN tersebut dengan menganggap bahwa segmen tersebut merupakan sekumpulan byte dari data. Field Acknowledgment Number dari sebuah segmen SYN diatur ke nilai ISN + 1.
FIN	Menandakan bahwa pengirim segmen *TCP* telah selesai dalam mengirimkan data dalam sebuah koneksi *TCP*. Ketika sebuah koneksi *TCP* akhirnya dihentikan (akibat sudah tidak ada data yang dikirimkan lagi), setiap host *TCP* akan mengirimkan sebuah segmen *TCP* dengan flag FIN diset ke nilai 1. Sebuah host *TCP* tidak akan mengirimkan segmen dengan flag FIN hingga semua data yang dikirimkannya telah diterima dengan baik (menerima paket acknowledgment) oleh penerima. Setiap host akan menganggap sebuah segmen *TCP* dengan flag FIN sebagai sekumpulan byte dari data. Ketika dua host *TCP* telah mengirimkan segmen *TCP* dengan flag FIN dan menerima acknowledgment dari segmen tersebut, maka koneksi *TCP* pun akan dihentikan.

Sumber : RFC 793, September 1981

Dalam *TCP*, klien berkomunikasi dengan *remote node* menggunakan mekanisme *3-way handsake* seperti ditunjukkan pada gambar 2.1. Pada gambar tersebut digambarkan sebuah klien pertama kali mengiring *SYN Request* kepada *server* untuk membuat sebuah koneksi, selanjutnya *server* akan mengirim sebuah pake *SYN/ACK* kepada klien untuk memberikan sinyal bahwa menerima paket *SYN* dari klien, ketika klien menerima paket *SYN/ACK* selanjutnya klien akan mengirim paket *ACK* kepada *server* untuk memberikan sinyal bahwa klien sudah menerima paket *SYN/ACK* dari *server*. Setelah ketiga proses tadi dilalui maka transfer data akan dimulai.



Gambar 2.1 *3-way Handshake TCP*

2.2 *DENIAL OF SERVICE*

Denial of Service juga dikenal sebagai *Distributed Denial of Service (DDoS)*, *Packet Storming*, dan *TribalFlooding*. Metode ini digunakan untuk membuat jaringan *overload* dengan membuat begitu banyak permintaan dimana trafik reguler diperlambat atau kadang-kadang diinterupsi. Sebuah serangan *DoS* reguler tidak sampai memasuki sebuah target karena sasaran hanyalah bertujuan untuk meng-overload target (*router* atau *webserver*) dengan begitu banyak trafik palsu yang tidak dapat diatasi.

Target yang tidak mampu mengatasi serangan trafik palsu akan tidak dapat melayani pengguna legal yang melakukan koneksi sehingga menyebabkan keadaan "*denied service*". Pada *Ddos* trafik palsu dihasilkan oleh banyak komputer yang telah diinfeksi sebelumnya oleh sebuah *daemon* (program *background*) dan komputer-komputer ini selanjutnya dikenal dengan *zombie*. Serangan *DoS* mudah diimplementasikan dan dapat menyebabkan kerusakan yang cukup berarti, dengan cara mengacaukan *server* atau operasi jaringan dan

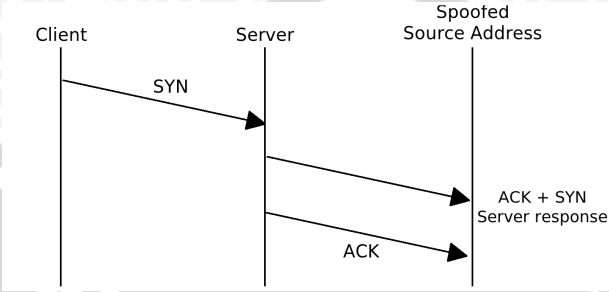
memutuskannya dari koneksi internet. Beberapa serangan *DoS* yang cukup terkenal adalah *syn flood* dan *Smurf Attack*. *Syn flood* memanfaatkan karakteristik koneksi *TCP* dan membentuk *half-open TCP* fiktif untuk melemahkan sumber sistem yang diserang. *Smurf Attack* adalah bentuk *DoS* yang mengeksploitasi pengunaan *Internet Control Message Protocol* (*ICMP*) dan jaringan alamat *broadcast*. Tujuan *smurf* adalah untuk mematikan *host* atau jaringan yang diserang dengan menghabiskan semua *resource*. (Thomas, 2004)

2.3 *SYN FLOOD*

Serangan *syn flood* memanfaatkan mekanisme komunikasi *connection-oriented TCP*. Pada Mekanisme *3-Way Handshake*, kondisi ketika *server* menunggu paket *ACK* dari klien disebut *half-open*. *Server* yang sedang dalam kondisi *half-open* melakukan persiapan untuk komunikasi dengan klien, seperti contoh mengalokasikan b*uffer*. Ketika *server* dalam kondisi *half-open* maka *server* menggunakan sebagian *resource* untuk klien. Jumlah koneksi yang bisa ditangani dalam keadaan *half-open* dikontrol oleh *backlog queue*. Paket *SYN* yang melebihi jumlah yang bisa ditangani oleh *backlog queue* akan dibuang, selanjutnya *server* akan mengirimkan paket *RST* untuk memberitahukan kepada klien bahwa paket *SYN* yang dikirim telah dibuang.

Serangan *Syn Flood* terjadi saat sebuah jaringan dipenuhi paket-paket *SYN* yang menginisiasi koneksi-koneksi yang tidak lengkap dan jaringan tidak dapat memproses koneksi yang sah dengan demikian menyebabkan suhu CPU yang tinggi, masalah pada sistem memori, dan *NIC Usage*. (Thomas, 2004). Gambar 2.2 menunjukkan pandangan singkat dari sebuah serangan *SYN Flood*. Penyerang akan mengirmkan paket *SYN* yang telah dipalsukan, dengan kata lain alamat pengirim bukan dari klien sesungguhnya atau alamat pengirim palsu dimasukkan pada *address field*. *server* menerima paket *SYN* dan mengirimkan *SYN/ACK* pada alamat palsu, jika terdapat *host* yang ternyata memiliki alamat palsu yang dikirim penyerang maka host tersebut akan mengirimkan paket *RST* untuk *SYN/ACK* yang dikirim *server* karena node tersebut tidak mengirimkan paket *SYN* sebelumnya sedangkan jika tidak ada host yang memiliki alamat palsu tersbebut maka paket *SYN/ACK* akan dibuang oleh jaringan komputer dan *server* akan menunggu *ACK* dari alamat palsu secara sia-sia. Ketika b*acklog queue* diisi dengan paket *SYN* dari alamat palsu maka *server* tidak bisa

menerima paket *SYN* dari klien lain yang berusaha melakukan koneksi dengan *server*.



Gambar 2.2 *Syn flood*

Paket yang digunakan oleh *SYN Flood* tidak berbeda dengan paket *TCP SYN* normal kecuali pada alamat pengirim yang palsu, hal ini menyebabkan sulitnya membedakan trafik *SYN Flood* dengan paket *TCP SYN* normal pada *server* target. (Yuichi; Shingo, 2002)

2.4 DETEKSI INTRUSI

Deteksi Intrusi memiliki implementasi *TCP/IP* khusus yang memungkinkan dirinya mengumpulkan paket dan selanjutnya memasang kembali paket-paket tersebut untuk dianalisa. Tidak cukup hanya melakukan s*niffing*, *IDS* harus melakukan analisa dan penyelidikan dengan beberapa cara yang akan disebutkan dibawah ini.

Analisis Protokol

Serangan menggunakan metode mengubah informasi protokol dasar telah banyak digunakan. Misal *ping of death* dapat dilancarkan karena mengubah ukuran paket, dan hal tersebut akan dideteksi oleh verifikasi protokol. *IDS* memiliki sistem verifikasi yang akan mengidentifikasi paket-paket invalid.

Deteksi Anomali

Deteksi dilakukan dengan melakukan monitoring trafik yang menyimpang dibandingkan dengan aktifitas normal. Aktifitas "normal" disini tentunya berbeda pada tiap jaringan komputer. Pendeteksian serangan *syn flood* menggunakan *naïve bayes* menggunakan pendeteksian model ini karena tahapan pembelajaran *classifier* menentukan tingkatan aktivitas dasar koneksi jaringan komputer.

Classifier selanjutnya menentukan koneksi mana yang merupakan sebuah koneksi anomali atau bukan dan menentukan apakah koneksi tersebut diklasifikasikan menjadi serangan atau tidak.

Mencocokkan Pola *Signature*

Mencocokkan pola s*ignature* adalah metode yang paling umum digunakan untuk mendeteksi serangan. Metode ini mencocokkan *signature* yang ada di database *IDS* dengan pola serangan yang muncul di jaringan komputer.

Analisis Log

IDS memiliki kapabilitas untuk menerima pesan *log* dari berbagai peranti dan memeriksa *log* ini untuk kepentingan keamanan peranti yang terkait. (Thomas, 2005)

2.5 *DATA MINING BASED IDS*

Intrusion Detection System (*IDS*) melakukan pengamatan perangkat jaringan komputer dan mencari kegiatan yang ganjil dan mencurigakan pada pola aktifitas di jaringan yang dipantau. Sebuah *IDS* yang komperhensif membutuhkan campur tangan manusia dan waktu yang lama untuk melakukan pengembangan. *Data Mining Based IDS* memerlukan *expert knowledge* lebih sedikit untuk menghasilkan performa yang bagus. Sistem ini juga mampu untuk mengenali serangan yang baru dan serangan yang belum diketahui (Campos, Milenova, 2001).

IDS bisa dibedakan menjadi dua strategi yaitu *Misuse Detection* dan *Anomaly Detection*. *Misuse Detection* mencoba untuk mencocokkan aktivitas yang sudah diobservasi dengan pola serangan yang sudah dikenali. *Anomaly Detection* berusaha untuk mengidentifikasi perilaku yang tidak sesuai dengan perilaku normal. Sesuai dengan definisi *Anomaly Detection* maka pendeteksian serangan *syn flood* menggunakan *naïve bayes* masuk kedalam kategori ini karena strategi pendeteksian ini mengidentifikasi paket koneksi jaringan komputer apakah sesuai dengan paket koneksi jaringan *TCP* normal. Pendekatan ini mampu mendeteksi serangan baru.(Campos, Milenova, 2001).

2.6 *NAÏVE BAYES*

Naïve Bayes termasuk dalam algoritma pembelajaran *Bayes*. Algoritma pembelajaran *bayes* menghitung probabilitas pada eksplisit untuk menggambarkan hipotesa yang dicari. Suatu data pada *naïve*

bayes dipresentasikan konjungsi dari nilai-nilai data atribut dan sebuah fungsi target yang dapat memiliki nilai apapun dari himpunan set domain V (Dumais, dkk. 1998). Sistem dilatih menggunakan data latih lengkap berupa pasangan nilai-nilai atribut dan nilai target kemudian sistem akan diberikan sebuah data baru dalam bentuk (a1,a2,a3,...an) dan sistem diberi tugas untuk menebak nilai fungsi target dari data tersebut.(Mitchell, 1997).

$$V_{nb} = argmax_{vj \in V} P(v_j | a_1, a_2, a_3, ... a_n) \quad \textbf{(2.1)}$$

Teorema *bayes* kemudian digunakan untuk menulis ulang persamaan 2.1 menjadi persamaan 2.2

$$V_{nb} = argmax_{vj \in V} \frac{P(a_1, a_2, a_3, ... a_n | v_j) P(v_j)}{P(a_1, a_2, a_3, ... a_n)} \quad \textbf{(2.2)}$$

Karena $P(a_1, a_2, a_3, ... a_n)$ nilainya konstan untuk semua vj sehingga persamaan 2.2 dapat ditulis menjadi persamaan 2.3

$$V_{nb} = argmax_{vj \in V} P(a_1, a_2, a_3, ... a_n | v_j) P(v_j) \quad \textbf{(2.3)}$$

Tingkat kesulitan menghitung $P(a_1, a_2, a_3, ... a_n | v_j)$ menjadi tinggi karena jumlah parameter $P(a_1, a_2, a_3, ... a_n | v_j)$ bisa jadi akan sangat besar. *Naïve bayes* menyederhanakan hal ini dan bekerja dengan dasar asumsi bahwa atribut-atribut yang digunakan bersifat conditionally independent antara satu dengan lainnya, dengan kata lain dalam setiap kategori, setiap atribut independen satu sama lainnya sehingga dirumuskan pada persamaan 2.4

$$P(a_1, a_2, a_3, ... a_n | v_j) = \prod P(a_i | v_j) \quad \textbf{(2.4)}$$

Substitusi dari persamaan 2.4 dengan persamaan 2.3 menjadi persamaan 2.5

$$V_{nb} = argmax_{vj \in V} P(v_j) \Pi P(a_i | v_j) \quad \textbf{(2.5)}$$

Vnb adalah nilai probabilitas hasil perhitungan *naïve bayes* untuk nilai fungsi target yang bersangkutan. Frekuensi kemunculan *syn flood* menjadi dasar perhitungan nilai dari *P(vj)* dan *P(ai|vj)*. Himpunan set dari nilai-nilai probabilitas ini berkorespondensi dengan hipotesa yang ingin dipelajari. Hipotesa kemudian digunakan untuk mengklasifikasikan data-data baru.

Secara umum dilakukan perkiraan $P(a_i|v_j)$ menggunakan *m-estimate*

$$P(a_i|v_j)=\frac{n_c+m.p}{n+m} \quad \textbf{(2.6)}$$

dimana

a_i = Atribut
v_j = class
n = jumlah dari sampel training untuk $v = v_j$
n_c = jumlah untuk sampel dari $v = v_j$ dan $a = a_i$
p = perkiraan sebelumnya untuk $P(a_i|v_j)$
m = ukuran sampel yang sama

(Meisner, 2003)

2.7 CONTOH APLIKASI *NAÏVE BAYES*

Terdapat data set seperti dibawah ini

Tabel 2-2 *dataset*

No	Warna	Tipe	Pembuat	Dicuri..?
1	Merah	Sport	Domestik	Yes
2	Merah	Sport	Domestik	No
3	Merah	Sport	Domestik	Yes
4	Kuning	Sport	Domestik	No
5	Kuning	Sport	Impor	Yes

6	Kuning	SUV	Impor	No
7	Kuning	SUV	Impor	Yes
8	Kuning	SUV	Domestik	No
9	Merah	SUV	Impor	No
10	Merah	Sport	Impor	Yes

Naïve bayes digunakan untuk mengklasifikasikan sebuah **SUV Merah Domestik**. Langkah pertama dilakukan penghitungan probabilitas

P(Merah|Yes), P(SUV|Yes), P(Domestic|Yes)

P(Merah|No) , P(SUV|No), P(Domestic|No)

Selanjutnya dikalikan dengan P(Yes) dan P(No) masing-masing. Perkiraan terhadap nilai tersebut mengunakan perhitungan dibawah ini.

Yes:	No:
Merah:	Merah:
n = 5	n = 5
n_c = 3	n_c = 2
p = .5	p = .5
m = 3	m = 3
SUV:	SUV:
n = 5	n = 5
n_c = 1	n_c = 3
p = .5	p = .5
m = 3	m = 3
Domestik:	Domestik:
n = 5	n = 5
n_c = 2	n_c = 3
p = .5	p = .5
m = 3	m =3

pada P(Merah|Yes), terdapat lima kasus dimana v_j = Yes, dan pada 3 dari kasus a_i = Merah. Jadi untuk P(Merah|Yes), n = 5 dan n_c = 3. Patut digarisbawahi bahwa semua atribut adalah biner. Diasumsikan bahwa

tidak ada lagi informasi jadi p = 1/(jumlah dari nilai atribut) = 0.5 untuk semua atribut. Nilai m adalah bisa berubah-ubah, (ditetapkan m=3) namun konsisten untuk semua atribut. Sekarang dilalakukan perhitungan dengan nilai $n,n_c,p,$ dan m yang sudah dihitung sebelumnya.

$$P(Merah|Yes) = \frac{3+3*.5}{5+3} = .56 \quad P(Merah|No) = \frac{2+3*.5}{5+3} = .43$$

$$P(SUV|Yes) = \frac{1+3*.5}{5+3} = .31 \qquad P(SUV|No) = \frac{3+3*.5}{5+3} = .46$$

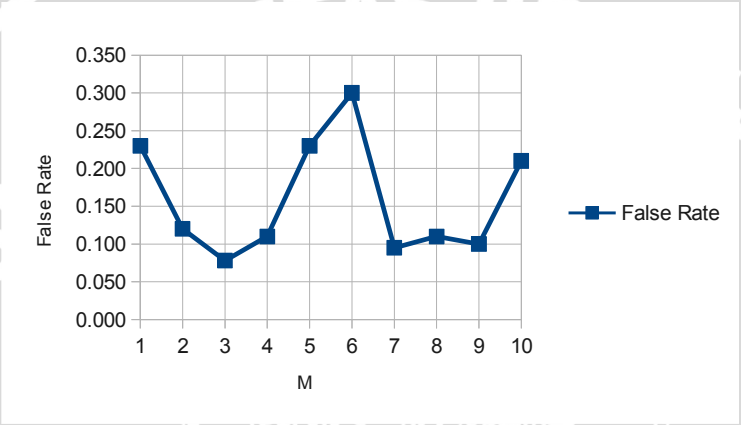
2.8 EVALUASI

Dalam melakukan pengklasifikasian serangan menggunakan naïve bayes, kejadian probabilitas = 0 dimungkinkan terjadi karena itu dibutuhkan persamaan *M-Estimates* untuk meminimalisir hal itu. Permasalahan yang selanjutnya timbul adalah bagaimana menentukan nilai M dalam persamaan *M-Estimates* sehingga didapatkan nilai M bisa menghasilkan tingkat akurasi pendeteksian yang paling tinggi. Nilai M sendiri adalah *arbritary* atau nilainya bebas ditentukan sendiri oleh *user* sehingga dalam percobaan kali ini diujicobakan nilai M mulai dari 1 sampai 10. Akurasi pendeteksian pada tiap nilai M akan ditampilkan dalam tabel 2.3 dan selanjutnya akan digambarkan pada grafik seperti pada gambar 2.3. Nilai *false rate* pada tabel 2.3 didapat dari persamaan 2.8. Dari gambar 2.4 selanjutnya akan ditentukan nilai M berapakah yang paling optimal dalam menghasilkan akurasi pendeteksian yang paling baik.

Tabel 2.3 *M-Estimates*

M	False rate
1	0.230
2	0.120
3	0.078
4	0.110
5	0.230

6	0.300
7	0.095
8	0.110
9	0.1
10	0.21



Gambar 2.3 Grafik *M-Estimates*

Dari gambar 2.3 bisa dilihat bahwa *false rate* terendah berada pada nilai M=3 maka nilai tersebut digunakan untuk melakukan pendeteksian *syn flood* menggunakan algoritma *naïve bayes*.

Receiver Operating Charatecistics (*ROC*) adalah sebuah teknik untuk memvisualisasikan, mengatur, dan memilih *classifier* berdasarkan performanya. *ROC* telah sejak lama digunakan di teori pengklasifikas deteksi sinyal.

Pada pengklasifikasi dan *instance* terdapat empat kemungkinan keluaran. Jika *instance* positif dan pengklasifikasi positif maka dihitung sebagai *true positive*, jika *instance* positif dan pengklasifikasi negatif maka dihitung sebagai *false negative*, jika *instance* negatif dan diklasifikasikan negatif maka dihitung sebagai *true negative*, sedangkan jika *instance* positif dan diklasifikasikan sebagai positif maka dihitung sebagai *false positive.*

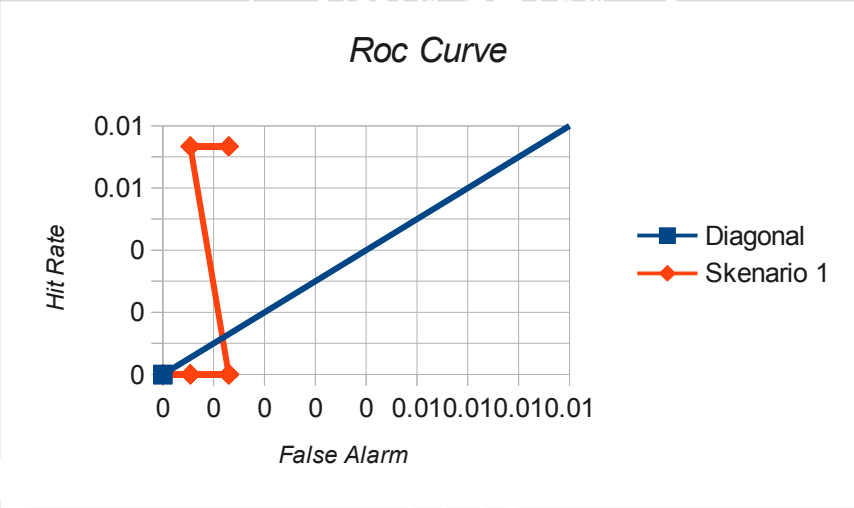
Hit Rate dari sebuah pengklasifikasi dirumuskan pada persamaan 2.7

$$Hit\ Rate \approx \frac{Klasifikasi\ Positif\ \ yang\ benar}{Total\ Positif} \quad \textbf{(2.7)}$$

False Positive rate, juga disebut sebagai *false alarm rate* dirumuskan pada persamaan 2.8

$$False\ Alarm\ rate \approx \frac{Klasifikasi\ Negatif\ \ yang\ salah}{Total\ Negatif} \quad \textbf{(2.8)}$$

Hasil perhitungan *Hit Rate* dan *False Alarm Rate* digambarkan pada Grafik ROC seperti ditunjukan pada gambar 2.4. Grafik *ROC* adalah sebuah grafik dua dimensi dimana *Hit Rate* digambarkan pada sumbu Y dan *False Alarm Rate* digambarkan pada sumbu X. Sebuah grafik *ROC* menggambarkan sebuah *trade-off* dari *True Positive dan False Positives.*



Gambar 2.4 *ROC Curve*

Garis diagonal $y = x$ merepresentasikan strategi dari menebak sebuah *class* secara acak. Sebagai contoh jika sebuah *classifier* secara acak menebak *class positif* pada separuh waktu diharapkan bisa mendapatkan separuh positif dan separuh negatif yang benar.

Tujuan dari *ROC Curve* adalah melihat bagaimana *classifier* memisahkan antara klasifikasi positif dan negatif selainitu juga untuk melihat kecenderungan dari *classifier* dalam melakukan pengklasifikasian. Apakah *classifier* lebih baik berkecenderungan lebih baik dalam mengidentifikasi positif atau negatif. (Fawcett, 2005)

Akurasi dari sebuah pengklasifikasian bisa dilakukan dengan membandingkan hasil klasifikasi dengan kondisi aktual data dan direpresentasikan dalam bentuk prosentase. Akurasi dengan kondisi serangan terdeteksi lebih kecil daripada serangan kondisi aktual dirumuskan pada persamaan 2.9.

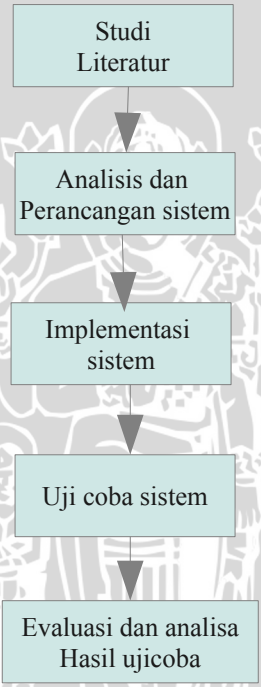
$$Akurasi = \frac{serangan\ terdeteksi}{serangan\ aktual} \times 100 \quad \textbf{(2.9)}$$

Sedangkan akurasi dengan kondisi serangan terdeteksi lebih besar daripada serangan kondisi aktual dirumuskan pada persamaan 2.10

$$Akurasi = \frac{serangan\ aktual - (serangan\ terdeteksi - serangan\ aktual)}{serangan\ aktual} \times 100 \quad \textbf{(2.10)}$$

BAB III
METODOLOGI PENELITIAN

Sistem ini dirancang untuk menghasilkan aplikasi yang mampu melakukan pendeteksian serangan *Syn Flood*. Percobaan ini akan diterapkan pada *Personal Computer* yang difungsikan sebagai *server* dengan Sistem Operasi Linux Fedora dan komputer klien yang berbasis Sistem Operasi Linux Centos. alur dari seluruh proses penelitian ditunjukan oleh gambar 3.1.



Gambar 3-1 Diagram Penelitian

Adapun tahapan pembuatan sistem adalah sebagai berikut :

1. Melakukan studi literatur terhadap *Intrusion Detection System*, sistem jaringan komputer, eksploitasi sistem, dan *Naïve Bayesian Classifier*.

2. Menganalisa dan merancang sistem termasuk pembuatan sistem jaringan dan penerapan algoritma *Naïve Bayes*.
3. Mengimplementasikan Sistem Jaringan Komputer dan perangkat lunak berdasarkan analisa dan perancangan yang dilakukan.
4. Melakukan uji coba terhadap perangkat lunak dan mengumpulkan data hasil pengujian yang dilakukan.
5. Melakukan evaluasi terhadap perangkat lunak dan data hasil uji coba yang diperoleh.

3.1 Analisa Sistem

Pada subbab analisis sistem ini akan dibahas berbagai hasil analisis terhadap sistem dan elemen-elemen yang terkait dan semua hal yang dibutuhkan dalam proses pendeteksian serangan *Syn Flood*.

3.1.1 Deskripsi Umum Sistem

Pendeteksian serangan *Syn Flood* yang akan dibuat merupakan sistem yang membaca sebuah inputan berupa informasi koneksi jaringan komputer dengan menghitung probabilitas apakah koneksi jaringan komputer tersebut termasuk dalam sebagai serangan *Syn Flood*. Metode yang digunakan pada sistem ini adalah pengklasifikasian dengan pendekatan n*aïve bayes*.

Pendeteksian serangan didasarkan pada ekstraksi data *TCPdump* dimana data dipecah dan diambil informasi yang dibutuhkan yaitu informasi *Source Port*, *Destination Port*, flag, *Sequence, Window, dan length*. Dalam pendekatan *naïve bayes* dibutuhkan dua kelompok data yang selanjutnya disebut *dataset*, yaitu *dataset* latih dan data set latih. Pada *dataset* latih terdapat ribuan koneksi jaringan komputer yang sudah teridentifikasi berapa koneksi yang merupakan koneksi legal dan berapa koneksi yang merupakan serangan *syn flood* sedangkan *dataset* uji berisi koneksi jaringan komputer yang belum teridentifikasi ada berapa banyak koneksi legal dan serangan *syn flood*. Secara umum sistem dibagi dalam tiga tahap. Tahap pertama adalah pembuatan *dataset* dengan langkah sebagai berikut.

1. Mempersiapkan dua *host* dimana salah satu diantaranya akan menjadi klien dan lainnya menjadi *server*.
2. Diantara *server* dan klien dipasang *packet sniffer* untuk memantau dan menangkap semua koneksi jaringan komputer yang terjadi diantara klien dan *server*.

3. *klien* melakukan koneksi secara legal dengan menjalankan *script* yang secara berkala melakukan koneksi *TCP* pada protokol *HTTP* di *server* dan melakukan *download file*.
4. Secara simultan klien juga melakukan serangan syn *flood* terhadap *server*.

Tahap kedua adalah proses *preprocessing* dan transformasi *dataset* dari *TCPdump* ke dalam database. Langkah-langkah yang dilakukan adalah.

1. Memisahkan dan membuang informasi yang tidak perlu pada *dataset*.
2. Memasukkan *dataset* yang berbentuk *file* ASCII yang sudah dilakukan *prepocessing* kedalam database.

Tahap ketiga adalah proses pendeteksian serangan *syn flood*. Dalam pendekatan *naïve bayes* dibutuhkan dua *dataset*, yaitu *dataset* latih dan *dataset* uji. *dataset* latih terdiri dari ribuan koneksi jaringan komputer legal dan serangan *syn flood* yang sudah teridentifikasi sedangkan pada *dataset* uji terdapat koneksi jaringan komputer legal dan serangan *syn flood* yang belum teridentifikasi. Proses pendeteksian dilakukan dengan langkah-langkah sebagai berikut.

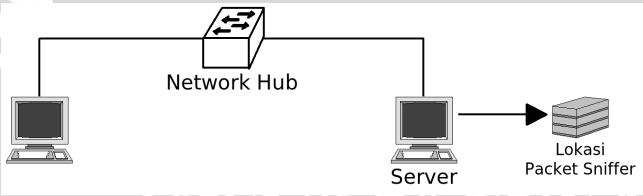
1. *User* menyiapkan data latih.
2. *User* menentukan data uji yang akan dideteksi dengan beberapa parameter yang diperlukan.
3. Untuk setiap koneksi jaringan komputer pada data uji akan dihitung kemungkinan terhadap serangan *syn flood* berdasarkan pada data latih yang sudah disiapkan.
4. Nilai kemungkinan yang lebih tinggi akan menentukan apakah pada tiap koneksi jaringan komputer pada data uji terdapat serangan *syn flood* atau tidak.

3.1.2 Analisis Kebutuhan Perangkat Keras

Penelitian ini mengangkat pembahasan tema tentang keamanan jaringan maka diperlukan perangkat keras yang memadai guna menunjang penelitian baik perangkat lunak dari segi pembuatan *dataset* sampai pendeteksian serangan. Secara *logic* pada sistem jaringan komputer terdapat dua *host* yang berfungsi sebagai klien dan *server* dan satu *packet sniffer* seperti yang ditunjukan pada gambar 3.2

Gambar 3.2 Jaringan komputer secara *logic*

sedangkan secara fisik sistem jaringan komputer membutuhkan peralatan tambahan yaitu kabel jaringan *UTP Cat* 5 dengan konektor RJ-45 dan sebuah *Hub*. *Packet sniffer* sendiri secara fisik berada di *server*. Gambaran dari sistem jaringan komputer secara fisik ditunjukan oleh gambar 3.3



Gambar 3.3 Jaringan komputer secara fisik

3.1.3 Analisis Kebutuhan Perangkat Lunak

Perangkat lunak yang digunakan mencakup kebutuhan dari sistem operasi, *packet sniffer*, *preprocessing tools*, *database*, dan desain dari *detector syn flood*. Dibawah ini adalah daftar perangkat lunak guna menunjang penelitian ini.

1. Alat Bantu Penelitian
   - Tshark (lebih terkenal dengan nama Wireshark) sebagai *packet sniffer*. *Tool* ini dipakai secara luas pada penelitian dan kegiatan praktis sistem jaringan komputer.
   - *Hping* dipilih untuk menjadi tool yang akan melakukan serangan *syn flood* dari klien ke *server*
2. Sistem Pendeteksian
   - Sistem operasi *Linux*
   - *preprocessing* dilakukan dengan menggunakan program yang ditulis menggunakan bahasa *Python.*
   - *Sqlite*, sebuah database sistem yang ringan dipilih karena tabel database yang digunakan sederhana dan kecepatan proses menjadi pertimbangan utama.

- Untuk keperluan pembuatan koneksi jaringan komputer legal dari klien ke *server* dibuat script yang menggunakan *Bash*. *Script* ini secara berkala akan melakukan koneksi *TCP* dan melakukan *downolading file* dari *server*.
- Transformasi *dataset* dari *file ASCII* ke dalam database dibuat dengan bahasa pemrograman *Python*. Proses yang berjalan adalah program akan membaca setiap baris pada *file ASCII* dan memasukkannya ke dalam *database*.
- Pendeteksian *Syn Flood* dibuat dengan menggunakan bahasa *Python* dengan pendekatan *naïve bayes*.

Gambaran umum dari pendeteksian *Syn Flood* ini, user menyediakan data latih dan data uji yang akan digunakan. Dari data latih yang disediakan maka akan disimpulkan ada berapa serangan yang terjadi di data uji.

3.2 Perancangan Sistem

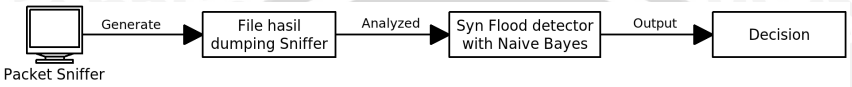
Pada percobaan ini klien akan melakukan koneksi secara langsung kepada *server*. Selain sebagai *legitimate user,* klien juga melakukan serangan *Syn Flood*. *Packet Sniffer* melakukan *capturing* paket pada saat koneksi terjadi antara klien dan *server*. Karena percobaan ini berkaitan dengan *Syn Flood* maka *capturing* dibatasi pada protokol *TCP*. Setelah dilakukan proses *capturing* maka *packet sniffer* membuat *ouput file* dengan format **.pcap* yang selanjutnya dikonversi menjadi *file *.csv* (ditunjukkan oleh gambar 3.4). *file *.csv* yang masih raw ini dilakukan *preprocessing* untuk membuang karakter yang tidak dibutuhkan dalam penelitian sehingga didapatkan data *source, destination, flag, sequence, win, len*.

```
175.45.187.163,175.45.187.162,TCP,53262 > http [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSV=105146121 TSER=0 WS=7
175.45.187.162,175.45.187.163,TCP,"http > 53262 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSV=105123656
175.45.187.163,175.45.187.162,TCP,53262 > http [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSV=105146121 TSER=105123656
175.45.187.163,175.45.187.162,HTTP,GET / HTTP/1.0
175.45.187.162,175.45.187.163,TCP,http > 53262 [ACK] Seq=1 Ack=120 Win=5824 Len=0 TSV=105123656 TSER=105146121
175.45.187.162,175.45.187.163,TCP,[TCP segment of a reassembled PDU]
```

Gambar 3.4 *File* hasil konversi *packet sniffer*

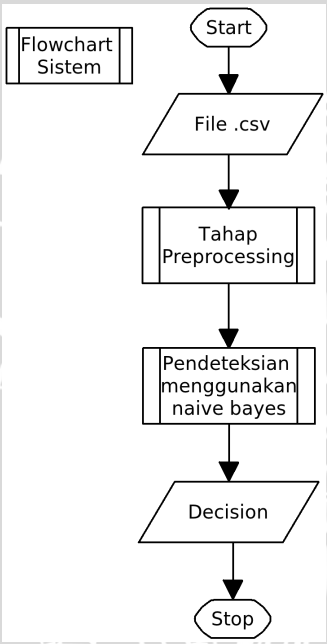
hasil *preprocessing* selanjutnya akan dimasukkan ke dalam *database*. Setelah semua *dataset* latih dan uji dimasukkan ke dalam database maka dilakukan pendeteksian serangan *syn flood* menggunakan *naïve bayes*. Tiap koneksi jaringan komputer dianalisa oleh perangkat lunak pendeteksi serangan *syn flood* dan membuat output apakah koneksi

tersebut merupakan serangan *syn flood* atau bukan. Proses ini digambarkan oleh Gambar 3.5.



Gambar 3.5 Aliran Data Sistem

Proses pada gambar 3.5 diterjemahkan dalam sebuah program pendeteksi serangan syn flood yang ditulis dengan bahasa python. Flowchart dari program tersebut digambarkan pada gambar3.6



Gambar 3.6 Flowchart perangkat lunak pendeteksi serangan *syn flood*

3.2.1 Pembuatan *Dataset*

Pembuatan *dataset* dilakukan dalam jaringan komputer lokal dengan dua *host* dan tidak ada trafik selain yang terjadi antara komputer klien dan *server*. *Packet sniffer* dijalankan dengan melakukan *capturing*

trafik pada protokol *TCP*. Klien menjalankan *bash script* yang melakukan koneksi legal terhadap *server* dan secara simultan melakukan serangan s*yn flood* terhadap *server* dengan menjalankan *tool hping*. Dalam satu satuan waktu *packet sniffer* akan menangkap semua koneksi yang terjadi antara klien dan *server*. Hasil dari *capture* paket dikonversi menjadi format *ASCII* yang selanjutnya akan dilakukan proses *prepocessing*.

Dalam percobaan kali ini *hping* akan melakukan serangan dengan menggunakan *port* acak dengan tipe data *string* sehingga terdiri dari huruf [Aa-Zz]dan angka [0-9]. Untuk keperluan pembuatan *dataset* latih dan *dataset* uji alamat yang di *spoof* akan mengirimkan paket berisi *flag RST (Reset)* terhadap *server* sehingga menjadi indikator bahwa terjadi serangan pada port tersebut. Pada kenyataannya *flag RST* tidak selalu menjadi indikator serangan, *flag RST* muncul ketika sebuah *port* memberikan informasi kepada *server* bahwa *port* bersangkutan akan mengatur ulang *sequence* koneksi menjadi nol (0) kembali karena beberapa hal seperti, *port* klien terlalu lama menunggu respon paket dari *server* sampai terjadi *time-out* (waktu tunggu kadaluarsa), terjadi *packet -loss* (paket tidak sampai kepada tujuan), atau memang karena alamat port tersebut dipalsukan.

Nilai tiap atribut pada *dataset* uji akan berbeda pada *dataset* uji, hal ini dilakukan untuk menguji akurasi dari *naïve bayes* dalam mendeteksi serangan *syn flood.*

3.2.2 Proses *Preprocessing Dataset*

Perangkat lunak yang akan dibuat adalah pendeteksi serangan s*yn flood* berdasarkan hasil *capture packet sniffer*. *File* masukan dari p*acket sniffer* akan dikonversi menjadi format *ASCII* dan selanjutnya dilakukan pendeteksian serangan s*yn flood* oleh sistem.

Proses awal dimulai dengan pembacaan *file* hasil capture dan memasukkan kedalam enam kategori yaitu s*ource port* yang berisi port asal datangnya paket, *destination port* yang berisi port tujuan paket, s*equence* yang berisi informasi rangkaian koneksi, w*in* yang berisi ukuran dari *window* paket, dan *len* yang berisi ukuran segmen paket, kategori terakhir adalah *label* yang menentukan dalam sebuah koneksi apakah terjadi serangan atau tidak.

preprocessing dilakukan sebelum hasil *capture* dimasukkan ke dalam *database*. Tujuan dari *preprocessing* adalah untuk menghasilkan

dataset yang siap digunakan dan meniadakan beberapa informasi yang tidak perlu guna mempercepat waktu pendeteksian dan mengurangi beban kerja *CPU*. Format *Raw ASCII* hasil *capture* ditunjukkan pada gambar 3.7

```
"12","0.003439","10.0.0.162","10.0.0.164","TCP","http >
54592 [ACK] Seq=2503 Ack=121 Win=5824 Len=0 TSV=344272757
TSER=7282597"
"13","0.010938","10.0.0.164","10.0.0.162","TCP","54593 >
http [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1
TSV=7282605 TSER=0 WS=7"
"14","0.010967","10.0.0.162","10.0.0.164","TCP","http >
54593 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460
SACK_PERM=1 TSV=344272765
```

Gambar 3.7 RAW Data

Setelah proses *preprocessing* data dimasukkan kedalam *database*. Tabel *database* ditunjukan pada tabel 3.1

Tabel 3.1 *Field Database*

Field name	Data Type	Keterangan
No	*Autonumber*	Nomer koneksi
Source	*String*	Alamat port asal
Dest	*String*	Alamat *port* tujuan
Sequence	*Integer*	Rangkaian koneksi
Win	*Integer*	Ukuran *window*
Len	*Integer*	Ukuran segmen paket
Label	*Integer*	Label serangan *syn flood*

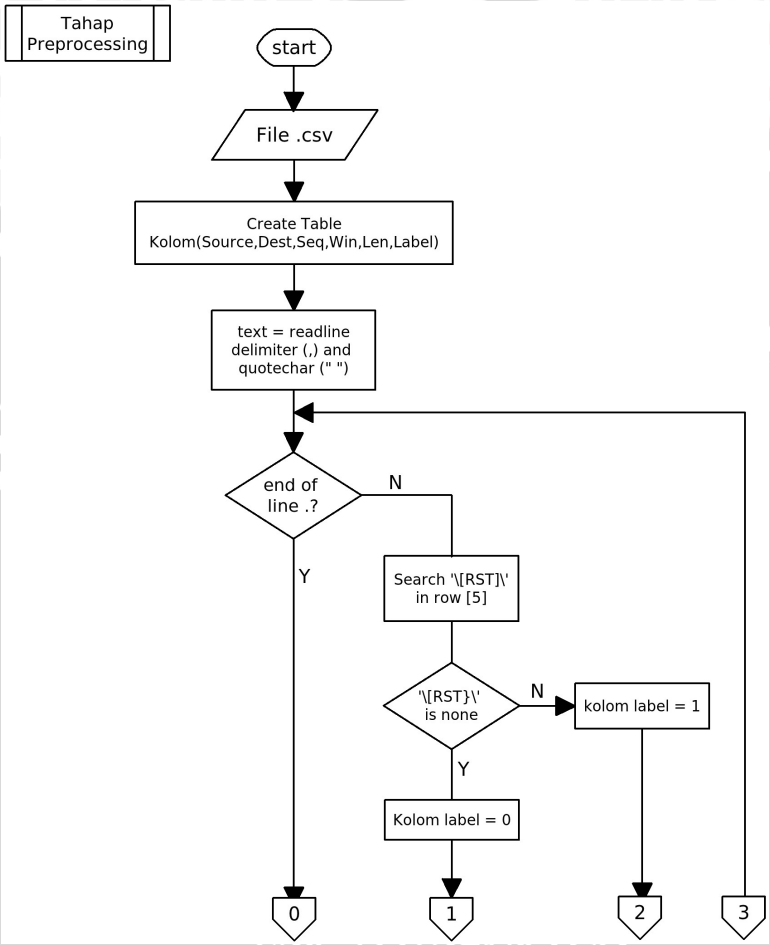
file raw ASCII berisi seluruh koneksi yang terjadi antara klien dan *server*. Satu koneksi direpresentasikan oleh satu baris *string*. Pada tiap baris terdapat informasi yang berkaitan dengan koneksi jaringan komputer serperti Alamat IP tujuan dan asal, *port* tujuan dan asal,

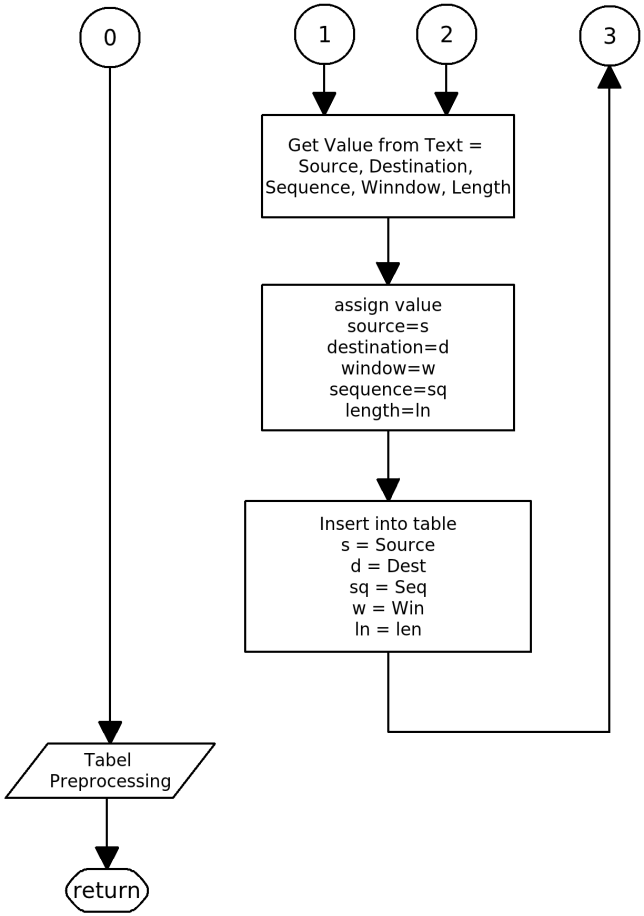
protokol yang digunakan, *Sequence*, *Window Length*, Informasi pada tiap protokol, dan lain sebagainya. Begitu banyak informasi yang terdapat pada tiap baris koneksi namun yang dibutuhkan pada penelitian ini adalah *port* tujuan dan asal, protokol yang digunakan, informasi pada protokol, dan *sequence number*, sehingga informasi yang berasal dari *packet sniffer* harus dipecah sedemikian rupa sehingga hanya informasi yang dibutuhkanlah yang akan dianalisa oleh *naïve bayes*.

Untuk mendapatkan informasi paket yang dibutuhkan maka dilakukan pemisahan *string* pada baris koneksi, Secara teknis hal ini bisa dilakukan dengan menggunakan *Regex*. *Timestamp*, Alamat IP, *string* yang berisi keterangan informasi paket dihilangkan karena tidak digunakan dalam penelitian ini. Khusus untuk informasi *flag RST* dijadikan rujukan dalam menentukan terdapat atau tidaknya serangan pada sebuah koneksi. Adapun tahapan dalam melakukan *prepocessing* dan transformasi ke dalam *database* adalah sebagai berikut.

1. Tabel database yang akan diberi masukkan data disiapkan. Dalam tabel tersebut berisi *source port, destination port, sequence, window, length,* dan *label.*
2. *File .csv* menjadi *inputan* dan selanjutnya dibaca dengan tanda petik *(") (') sebagai* delimiter.
3. Setiap baris pada file .csv dibaca satu persatu.
4. Pada setiap baris koneksi dicek apakah terdapat flag RST. Jika iya kolom label pada tabel database bernilai satu (1) jika tidak maka bernilai nol (0).
5. Nilai dari source, destination, sequence, window, dan length diambil dan dimasukkan pada variabel penampung sementara.
6. Nilai yang sudah berada pada variabel penampung sementara dimasukkan kepada kolom tabel database.
7. Proses tersebut dilakukan terus sampai semua baris koneksi selesai dibaca.

Flowchart proses diatas ditunjukan pada gambar 3.8


Tahap Preprocessing
start
File .csv
Create Table Kolom(Source,Dest,Seq,Win,Len,Label)
text = readline delimiter (,) and quotechar (" ")
end of line .?
N
Y
Search '\[RST]\' in row [5]
'\[RST}\' is none
N
kolom label = 1
Y
Kolom label = 0
0
1
2
3



Gambar 3.8 *preprocessing*

3.2.3 Pendeteksian Serangan

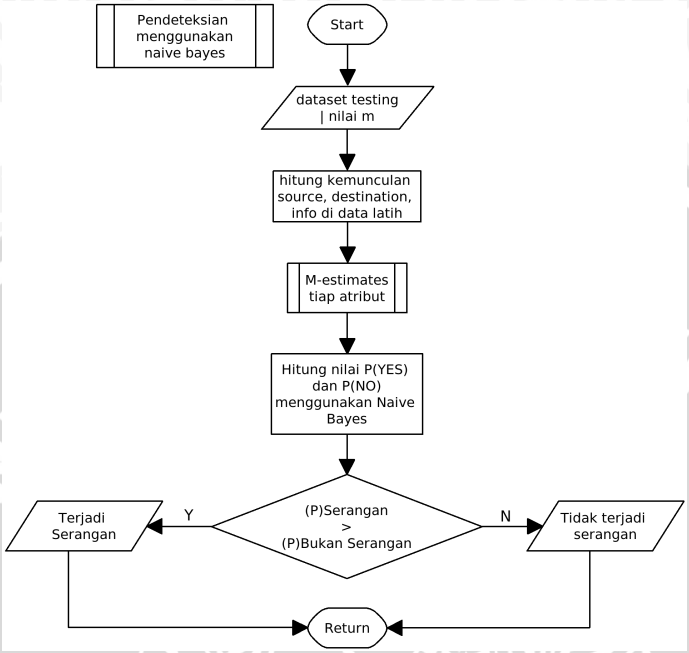
Database yang sudah dibentuk pada Prosedur *preprocessing* selanjutnya dihitung menggunakan algoritma *naïve bayes* sehingga diketahui apakah pada data tersebut terdapat serangan *syn flood* atau tidak. Langkah pertama adalah menghitung banyaknya koneksi yang disimbolkan dengan *n* dan koneksi yang memenuhi syarat label=1 yang disimbolkan dengan n_c untuk setiap tabel pada P=Yes dan P=No pada *dataset* latih.

Tahap selanjutnya adalah menghitung kemungkinan $P(a_i|v_j)$ dengan menggunakan *M-estimaes* untuk masing-masing kategori (gambar 3.10). Hasil penghitungan $P(a_i|v_j)$ menjadi inputan untuk untuk *naïve bayes classifier*. Proses terakhir adalah membandingkan antara P=Yes dan P=No untuk menentukan apakah terdapat *Syn-flood*. Jika P=Yes > P=No maka pada *Dataset* tersebut terdapat serangan *syn flood*, jika sebaliknya maka tidak terjadi serangan *syn flood.*

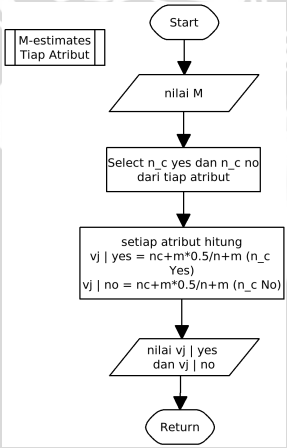
Dataset latih menjadi *trainer* pada penelitian ini sehingga *dataset* latih akan dijadikan acuan oleh mesin apakah pada *dataset* uji terdapat serangan atau tidak. Adapun tahapan dalam melakukan pendeteksian serangan dalam tiap koneksi jaringan komputer adalah sebagai berikut.

1. User menentukan dataset mana yang akan diuji dan memasukkan nilai M pada *m-estimates* yang digunakan.
2. Sistem menghitung kemunculan *source, destination, seq, win*, dan *len* di data latih yang memenuhi syarat label=1 dan syarat label=0.
3. Setiap atribut dihitung dengan menggunakan persamaan 2.6
4. Menghitung nilai P=Yes dan P=No menggunakan persamaan 2.5
5. Jika P=Yes > P=No maka koneksi yang diuji diklasifikasikan sebagai serangan, jika sebaliknya maka diklasifikasikan sebagai bukan serangan

Flowchart proses diatas bisa dilihat pada gambar 3.9



Gambar 3.9 Pendeteksian



Gambar 3.10 *M-Estimates*

3.3 Skenario Pengujian

Pada pengujian sistem pendeteksian *syn flood dataset* dibagi menjadi dua yaitu *dataset* latih dan *dataset* uji. Uji coba dilakukan berdasarkan skenario yang dijabarkan pada subbab 3.2. *dataset* uji terdiri dari 50 koneksi dengan 15 *port* yang berbeda sedangkan *dataset* latih terdiri dari banyak koneksi yang tidak tetap. Pemberian nilai M juga berubah-ubah mulai dari 1 sampai 10. Pengujian menggunakan beberapa skenario sebagai berikut :

1. Skenario pertama
   - *dataset* latih tediri dari 300 koneksi, 93 serangan, dan 207 legal.
   - *dataset* uji terdiri dari 50 koneksi, 10 serangan, dan 40 legal.
   - Nilai M diberikan mulai 1 sampai 10.
   - Terdapat *port* berbeda pada *dataset* uji dibandingkan dengan *dataset* latih.
2. Skenario kedua
   - *dataset* latih tediri dari 300 koneksi, 93 serangan, dan 207 legal.
   - *dataset* uji terdiri dari 50 koneksi, 10 serangan, dan 40 legal.
   - Nilai M diberikan mulai 1 sampai 10.
   - Terdapat port berbeda pada *dataset* uji dibandingkan dengan *dataset* latih.
3. Skenario ketiga
   - *dataset* latih terdiri dari 300 koneksi, 93 serangan, dan 207 legal
   - *dataset* uji terdiri dari 50 koneksi, 16 serangan, dan 34 legal.
   - Nilai M diberikan mulai 1 sampai 10.
   - Terdapat *port* berbeda pada *dataset* uji dibandingkan dengan *dataset* latih.

3.4 Penghitungan Manual

Pada tabel 3.2 terdapat 6 port berbeda yaitu port : 16785, 14786, 14787,18788, 147885, dan http yang tersebar pada *source* dan *destination port*.

Tabel 3.2 Data Latih

No	Source	Destination	Seq	Win	Len	label
1	http	14785	0	5840	0	0
2	14785	http	1	0	0	1
3	14786	http	0	512	0	0
4	http	14786	0	5840	0	0
5	14786	http	1	0	0	1
6	14787	http	0	512	0	0
7	http	14787	0	5840	0	0
8	14787	http	1	0	0	1
9	14788	http	0	512	0	0
10	http	14788	0	5840	0	0
11	147885	http	1	0	0	1

Label : 1 menunjukkan serangan, 0 menunjukkan bukan serangan

Diklasifikasikan instance baru sebagai berikut :

Source = 54594, **Dest** = http, **Seq** = 0, **Win** = 5220, **Len** = 0

Dengan metode *naïve bayes* dideteksi apakah terdapat serangan atau tidak.

P(Source|1), P(Destination-80|1), P(*TCP*|1), P(RST|1)

P(Source|0), P(Destination-80|0), P(*TCP*|0), P(RST|0)

Yes:
Source|54594
$n = 4$
$n_c = 0$
$p = .5$
$m = 3$

Destination|http
$n = 4$

No:
Source|54594
$n = 7$
$n_c = 0$
$p = .5$
$m = 3$

Destination|http
$n = 7$

```
nc = 4
p = .5
m = 3

Seq|0
n = 4
nc = 0
p = .5
m = 3

Win|5220
n = 4
nc = 0
p = .5
m = 3

Len|0
n = 4
nc = 4
p = .5
m = 3
```

```
nc = 3
p = .5
m = 3

Seq|0
n = 7
nc = 7
p = .5
m =3

Win|5220
n = 7
nc = 0
p = .5
m =3

len|0
n = 7
nc = 7
p = .5
m =3
```

Menurut *M-Estimates* dihitung perkiraan ($P\ a_i|v_j$) untuk tiap kemungkinan(*Yes* dan *No*) pada tiap parameter.

$$P(Source|Yes)=\frac{0+3*0.5}{4+3}=0.21 \qquad P(Source|No)=\frac{0+3*0.5}{7+3}=0.15$$

$$P(Dest|Yes)=\frac{4+3*0.5}{4+3}=0.78 \qquad P(Dest|No)=\frac{3+3*0.5}{7+3}=0.3$$

$$P(seq|Yes)=\frac{0+3*0.5}{4+3}=0.21 \qquad P(Seq|No)=\frac{7+3*0.5}{4+3}=0.71$$

$$P(Win|Yes)=\frac{0+3*0.5}{4+3}=0.21 \qquad P(Win|No)=\frac{0+3*0.5}{7+3}=0.15$$

$$P(Len|Yes)=\frac{4+3*0.5}{4+3}=0.5 \qquad P(Len|No)=\frac{7+3*0.5}{7+3}=0.5$$

Kita memiliki P*(Yes)* = 0.5 dan P*(No)* = 0.5 jadi kita bisa memasukkan perhitungan untuk v = Yes

```
P(Yes)*P(Source|Yes)*P(Dest|Yes)*P(Seq|Yes)*P(Win|
Yes)*P(Len|Yes)
= 0.5*0.21*0.78*0.21*0.21*0.5 = 0.001805895
```

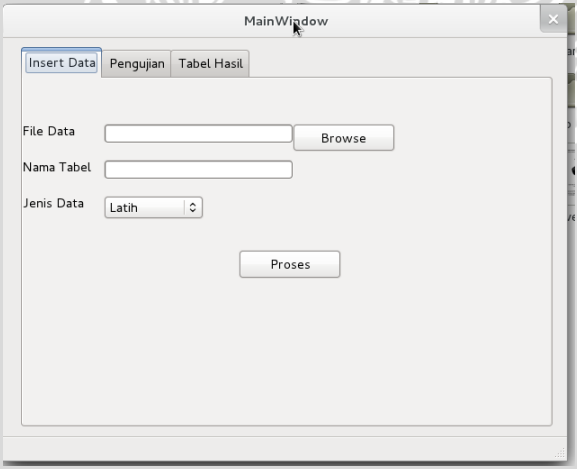
Dan untuk *v = No*

```
P(No)*P(Source|No)*P(Dest|No)*P(Seq|No)*P(Win|
No)*P(Len|No)
= 0.5*0.15*0.3*0.71*0.15*0.5 =  0.001198125
```

Dari Hasil diatas `0.001805895 > 0.001198125` yang artinya *v = Yes* > *v = No* Maka pencarian diklasifikasikan sebagai **Yes** atau dengan kata lain berdasarkan *Data Set* terdapat serangan *Syn Flood*

3.5 Rancangan Antar Muka

User interface berbentuk *GUI* (*Graphical User Interface*) pada sistem operasi *Linux*. *Framework python* dan *Qt* harus sudah terinstall sebelumnya untuk menjalankan program ini. Gambar 3.11 adalah *screenshoot* dari *User Interface*.

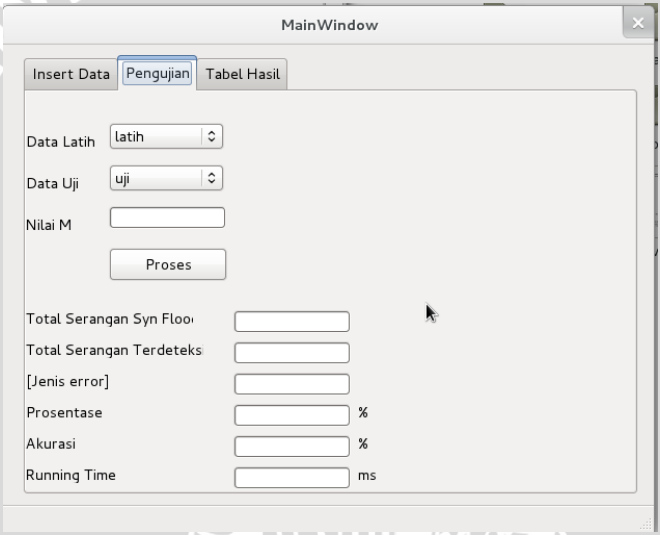


Gambar 3.11 Antar muka utama program

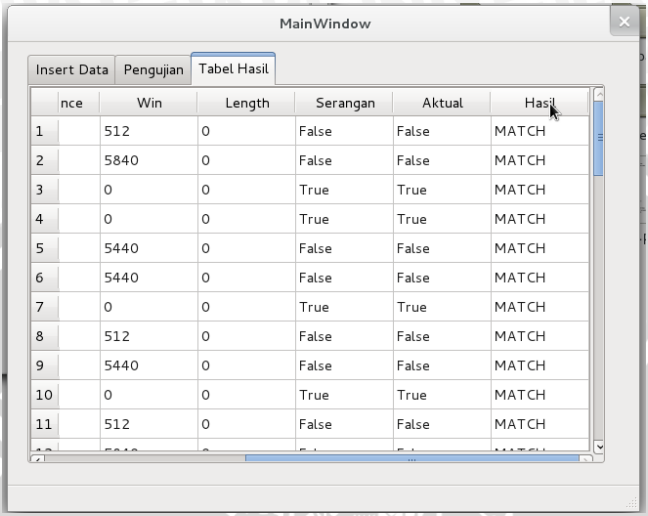
Pada main program terdapat tiga tab yaitu *Insert Data*, Pengujian, dan Tabel hasil. Tab *Insert Data* berisi mekanisme untuk memasukkan *dataset* yang sudah dalam format *ASCII*, ketika *file ASCII* sudah dipilih

maka user akan memasukkan nama *dataset* tersebut dan memasukkannya ke dalam *dataset* latih atau *dataset* uji. Setelah diberikan nama dan dimasukkan ke dalam kategori tombol proses melakukan proses *parsing data* dan memasukkannya ke dalam database.

Tab pengujian berisi pemilihan *dataset* latih dan uji yang akan dilakukan penelitian selanjutnya nilai M akan didefinisikan sendiri oleh pengguna dengan nilai *integer*. Tab Pengujian ditunjukkan oleh gambar 3.12. *Output* dari pengujian akan dicetak pada tab tabel hasil yang ditunjukan oleh gambar 3.13



Gambar 3.12 Antar Muka *Tab* Pengujian



Gambar 3.13 Antar Muka *Tab* Tabel Hasil

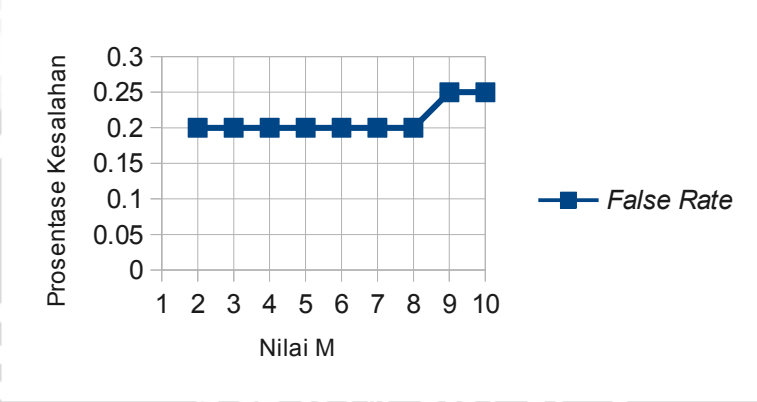
3.6 Pengujian

Pengujian digunakan untuk menganalisa bagaimana tingkat akurasi algoritma *naïve bayes* dalam mendeteksi serangan *syn flood* pada beberapa data set. Pengujian pertama adalah mencari nilai M pada *M-Estimates* yang paling optimal dalam mendapatkan akurasi pendeteksian optimal. Tabel 3.3 menunjukkan pengujian nilai M pada setiap skenario

Tabel 3.3 Tabel Pengujian

Nilai M	***False Detection***
1	0.2
2	0.2
3	0.2
4	0.2
5	0.2
6	0.2
7	0.2

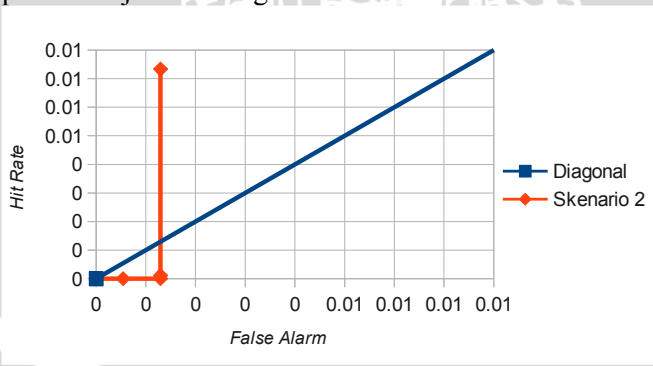
8	0.2
9	0.25
10	0.25



Gambar 3.14 Grafik nilai M dan Prosentase Akurasi

Dari tabel 3.3 selanjutnya dibuatlah grafik nilai M untuk melihat kecenderungan dari akurasi yang dihasilkan seperti ditunjukan pada gambar 3.14.

Setelah didapatkan nilai M yang paling optimal maka dilakukan pengujian menggunakan *ROC Curve* untuk melihat kecenderungan dari *classifier* dalam melakukan pendeteksian serangan. *Hit Rate* dan *False Alarm* dari setiap skenario selanjutnya akan digambarkan dalam sebuah grafik seperti ditunjukan oleh gambar 3.15



Gambar 3.15 *ROC Curve*

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

Implementasi merupakan transformasi representasi rancangan ke sistem dan bahasa pemrograman yang dapat dimengerti oleh komputer. Pada bab ini akan dibahas hal-hal yang berkaitan dengan implementasi sistem pendeteksian *syn flood*.

4.1 Lingkungan Implementasi

Lingkungan implementasi yang akan dipaparkan disini meliputi lingkungan perangkat keras dan lingkungan perangkat lunak.

4.1.1 Lingkungan Perangkat Keras

Perangkat keras yang digunakan dalam pembentukan data set adalah sebagai berikut:

1. Komputer *server*
   - Processor Intel Core Duo
   - 1 GB RAM
   - 160GB Harddisk Drive
2. Komputer klien
   - Processor Intel Core Duo
   - 1 GB RAM
   - 160GB Harddisk Drive
3. Peralatan Jaringan Komputer
   - Kabel UTP Cat 5
   - Konektor RJ-45
   - Switch 16 port

sedangkan perangkat keras yang digunakan untuk pengembangan sistem pendeteksian *syn flood* adalah :

- Intel Core2Duo P7350
- 2 GB RAM
- 320 GB Harddisk Drive

4.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan dalam pembentukan data set adalah sebagai berikut:

1. *server*

- Sistem Operasi Fedora 12
- *Apache Web server* pada port 80
- *Tshark (Wireshark) Packet Sniffer*

2. klien
   - Sistem Operasi Centos RHEL 5
   - *Hping Network Tester*
   - *Bash Script*

sedangkan untuk komputer yang melakukan pendeteksian *syn flood* menggunakan perangkat lunak sebagai berikut:

- Sistem Operasi Fedora 15
- Bahasa pemrograman *Python 2.7*
- Database *Sqlite*

4.2 Implementasi Proses Pembentukan Sistem

Berdasarkan hasil perancangan sistem pada sub bab 3.2 mengenai analisa proses pembentuk sistem dalam sistem pendeteksian serangan *syn flood*, maka pada subbab ini akan dijelaskan implementasi perancangan sistem tersebut.

4.2.1 Pembuatan *Dataset*

Langkah awal dari riset ini adalah pembuatan *dataset*. Dalam proses ini dilakukan simulasi serangan *syn flood* pada jaringan komputer lokal. Komputer klien berfungsi sebagai *host* yang melakukan koneksi *TCP* secara legal terhadap *server* sekaligus sebagai penyerang dengan menggunakan teknik *syn flood*. Koneksi legal dan serangan yang dilakukan menggunakan protokol *HTTP* dengan port 80 pada sisi *server*. Koneksi legal dari klien dibangkitkan dengan menggunakan *bash script* Kode Program 4.1

	web.sh
1	`#!/bin/bash`
2	
3	`counter=0`
4	`limit=10000`
5	`while [ "$counter" -lt "$limit" ]`
6	`do`
7	`wget --output-document=`*`file`*`_donlot http://10.0.0.5`

```
8   sleep 0.0015  #wait in several second
9   counter=`expr $counter + 1`
10  done
11  exit 0
```

Kode Program 4.1 *Legal Connection*

Script diatas melakukan koneksi *TCP* dengan protokol *HTTP* kepada *server* dan melakukan download sebuah yang bernama *file_donlot* dari komputer *server*. Serangan *syn flood* dilakukan dengan menggunakan *hping2* dimana *tool* ini memang banyak digunakan untuk melakukan *testing* penetrasi jaringan komputer. Serangan *syn flood* dijalankan dengan menggunakan kode program 4.2.

```
    hping2
1   hping -i u1 -S -p 80 10.0.0.5
```

Kode Program 4.2 Serangan *Syn Flood*

Koneksi yang terjadi di-*capture* menggunakan *tool Wireshark*, karena proses *capture* dijalankan pada *server* yang hanya terdapat *Command Line Interface* maka *capturing* dilakukan dengan menggunakan *Tshark* dimana *Tshark* ini adalah mode terminal dari *Wireshark*. Perintah untuk melakukan *capture* paket bisa dilihat pada kode program 4.3

```
    Tshark capture command
1   tshark -f "TCP port 80" -i eth0 -w dataset_TCP_1
```

Kode Program 4.3 *Tshark Capture Packet*

Penjelasan dari kode program 4.3 adalah sebagai berikut:

1. tshark adalah nama tool yang dijalankan
2. `-f "TCP port 80"` adalah opsi untuk melakukan *capture packet* protokol *TCP* di port 80.
3. `-i eth0` adalah opsi yang menunjukkan pada *interface* mana paket data yang di *capture*. Pada kasus ini di *eth0*.
4. `-w dataset_TCP_1` adalah opsi untuk menyimpan data hasil *capture*

file hasil *capture* tersebut masih berupa format *binary* dari *wireshark* dan tidak bisa dibaca tanpa program *wireshark* sehingga dilakukan konversi ke *file ASCII* oleh *wireshark* sehingga menjadi *human readable.* Sampel dari *file ASCII* hasil konversi dari *wireshark* bisa dilihat pada tabel 4.1

	Sampel *dataset* raw
	No.","Time","Source","Destination","Protocol","Info" "1","0.000000","175.45.187.164","175.45.187.162","*TCP*","54592 > http [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSV=7282594 TSER=0 WS=7" "2","0.000067","175.45.187.162","175.45.187.164","*TCP*","http > 54592 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSV=344272754 TSER=7282594 WS=6" "3","0.000154","175.45.187.164","175.45.187.162","*TCP*","54592 .l '> http [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSV=7282594 TSER=344272754" "4","0.000286","175.45.187.164","175.45.187.162","HTTP","GET / HTTP/1.0 " "5","0.000320","175.45.187.162","175.45.187.164","*TCP*","http > 54592 [ACK] Seq=1 Ack=120 Win=5824 Len=0 TSV=344272754 TSER=7282594" "6","0.002084","175.45.187.162","175.45.187.164","*TCP*","[*TCP* segment of a reassembled PDU]" "7","0.002104","175.45.187.162","175.45.187.164","HTTP","HTTP /1.1 200 OK (text/html)" "8","0.002191","175.45.187.162","175.45.187.164","*TCP*","http > 54592 [FIN, ACK] Seq=2502 Ack=120 Win=5824 Len=0 TSV=344272756 TSER=7282594" "9","0.002407","175.45.187.164","175.45.187.162","*TCP*","54592 > http [ACK] Seq=120 Ack=1449 Win=8832 Len=0 TSV=7282596 TSER=344272756" "10","0.002497","175.45.187.164","175.45.187.162","*TCP*","5459 2 > http [ACK] Seq=120 Ack=2502 Win=11648 Len=0 TSV=7282596 TSER=344272756" "11","0.003419","175.45.187.164","175.45.187.162","*TCP*","5459 2 > http [FIN, ACK] Seq=120 Ack=2503 Win=11648 Len=0 TSV=7282597 TSER=344272756" "12","0.003439","175.45.187.162","175.45.187.164","*TCP*","http > 54592 [ACK] Seq=2503 Ack=121 Win=5824 Len=0 TSV=344272757 TSER=7282597" "13","0.010938","175.45.187.164","175.45.187.162","*TCP*","5459 3 > http [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSV=728

Gambar 4.1 Sampel *dataset* Raw *file* Hasil Konversi *Wireshark*

4.2.2 Proses *Preprocessing* dan Transformasi ke Database

Pada sampel *dataset* raw terdapat informasi nomer koneksi, waktu konesi, alamat IP asal, alamat IP tujuan, Protokol yang digunakan, alamat *port* asal, alamat *port* tujuan, *flag*, *window size*, *checksum*, dan beberapa informasi tambahan. Program yang ditulis dengan bahasa *python* memanfaatkan *regex* untuk mensortir informasi yang diperlukan yang meliputi *source port*, *destination port*, *sequence*, *window*, *length*, dan *flag*. Setelah informasi yang dibutuhkan tersortir maka dilakukan proses transformasi data kedalam *database*. Proses *preprocessing dataset* raw dan transformasi kedalam *database* ditunjukkan pada kode program 4.4

DBHelper.py

```
import csv
import re
import sqlite3


def insertData(dataPath,tableName):
    conn = sqlite3.connect('dataset.db')

    #check if table exists
          isTableExists=conn.execute('select  count(*)  from
sqlite_master where name=?',(tableName,)).fetchone()

    if isTableExists==0:
        return 0
    else:
        curs = conn.cursor()
           curs.execute('CREATE TABLE %s (no INTEGER PRIMARY
KEY, source TEXT, dest TEXT, seq INTEGER, win INTEGER, len
INTEGER, label INTEGER)' % tableName)
        conn.commit()
        text=csv.reader(open(dataPath,'rb'),delimiter=',',quo
techar='"')
        for row in text:
            textData=row[5]
            if re.search('\[RST\]',textData) is None:
                label=0
            else :
                label=1
            textData=re.sub('Ack=[0-9] ','',textData)
            matchedRegex=re.match('.+Len=[0-9]+',textData)
            textData=matchedRegex.group()
            textData=re.sub('\[.+\] ','',textData)
            textData=re.sub(' >','',textData)
```

```
            textData=re.sub('[A-Za-z]+=','',textData)
            textData=re.sub(' ',';',textData)
            data = textData.split(';')
            data.append(label)
                        curs.execute('INSERT INTO %s VALUES
(null,?,?,?,?,?,?)'% tableName,(data))
        conn.commit()
        conn.close()
        return 1

def listLatihTable():
    conn = sqlite3.connect('dataset.db')
     latihTables=conn.execute("select name from sqlite_master
where name like ?",['%Latih_%']).fetchall()
    conn.close()
    return latihTables;

def listUjiTable():
    conn = sqlite3.connect('dataset.db')
      ujiTables=conn.execute("select name from sqlite_master
where name like ?",['%Uji_%']).fetchall()
    conn.close()
    return ujiTables;
```

Kode Program 4.4 *Prepocessing* dan transformasi kedalam d*atabase*

4.2.3 Pendeteksian Naïve Bayes

Proses ini melakukan pendeteksian terhadap *dataset* yang sudah disediakan. Proses ini menggunakan algoritma *naïve bayes* dengan menggunakan *m-estimates* karena kemungkinan dihasilkan *under-estimate* probabilitas bias karena terdapat probabilitas = 0. Keseluruhan proses ini ditunjukkan pada kode program 4.5 dengan bahasa *python*

classifier.py

```
from __future__ import division
import sqlite3
import time
import math

class NaiveBayes:
    def __init__(self):
        self.result=[]
        self.debug=[]
    def ujiData(self,paramLatih,paramUji,paramM):
```

```
        self.tabelLatih=paramLatih
        self.tabelUji=paramUji
        self.m=paramM
        conn = sqlite3.connect('dataset.db')
        curs = conn.cursor()

        a = curs.execute('SELECT COUNT (*) FROM %s' %
self.tabelLatih)
        latih = int(a.fetchone()[0])

        b = curs.execute('SELECT COUNT (*) FROM %s ' %
self.tabelUji)
        uji = int(b.fetchone()[0])

        c = curs.execute('SELECT COUNT (*) FROM %s where
label = "1" ' % self.tabelUji)
        positive = int(c.fetchone()[0])

       d = curs.execute('SELECT COUNT (*) FROM %s where
label = "0" ' % self.tabelUji)
        negative = int(c.fetchone()[0])

       t0= time.clock()
#BayesFunction
        curs.execute('SELECT * FROM "' + str(self.tabelUji) +
'"')
        result = curs.fetchall()
        countP = 0
       countN = 0
        for resultRow in result :
            s = resultRow[1]
            d = resultRow[2]
            sq = resultRow[3]
            w = resultRow[4]
            ln = resultRow[5]
            la = resultRow[6]
            if la==1:
                la=True
            else:
                la=False

            no = curs.execute('select count (*) from %s where
label = "0" ' % self.tabelLatih)
            n = int(no.fetchone()[0])
            yes = curs.execute('SELECT COUNT (*) from %s
where label = "1"' %self.tabelLatih)
            y = int(yes.fetchone()[0])
            tot = n+y


            source1 = curs.execute('SELECT COUNT (*) FROM %s
WHERE source=? and label = "0"' %self.tabelLatih,(s,))
```

```
            c1n = int(source1.fetchone()[0])
            source2 = curs.execute('SELECT COUNT (*) FROM %s
WHERE source=? and label = "1"' %self.tabelLatih,(s,))
            c1y = int(source2.fetchone()[0])
            prob_c1n = (c1n+self.m*0.5)/(n+self.m)
            prob_c1y = (c1y+self.m*0.5)/(y+self.m)


            dest1 = curs.execute('SELECT COUNT (*) FROM %s
WHERE dest=? and label = "0"' %self.tabelLatih,(d,))
            c2n = int(dest1.fetchone()[0])
            dest2 = curs.execute('SELECT COUNT (*) FROM %s
WHERE dest=? and label = "1" ' %self.tabelLatih,(d,))
            c2y = int(dest2.fetchone()[0])
            prob_c2n = (c2n+self.m*0.5)/(n+self.m)
            prob_c2y = (c2y+self.m*0.5)/(y+self.m)


            seq1 = curs.execute('SELECT COUNT (*) FROM %s
WHERE  seq=? AND label = "0"' %self.tabelLatih,(sq,))
            c3n = int(seq1.fetchone()[0])
            seq2 = curs.execute('SELECT COUNT (*) FROM %s
WHERE  seq=? AND label = "1" '%self.tabelLatih,(sq,))
            c3y = int(seq2.fetchone()[0])
            prob_c3n = (c3n+self.m*0.5)/(n+self.m)
            prob_c3y = (c3y+self.m*0.5)/(y+self.m)


            win1 = curs.execute('SELECT COUNT (*) FROM %s
WHERE  win=? AND label = "0" '%self.tabelLatih,(w,))
            c4n = int(seq1.fetchone()[0])
            win2 = curs.execute('SELECT COUNT (*) FROM %s
WHERE  win=? AND label = "1" '%self.tabelLatih,(w,))
            c4y = int(seq2.fetchone()[0])
            prob_c4n = (c4n+self.m*0.5)/(n+self.m)
            prob_c4y = (c4y+self.m*0.5)/(y+self.m)


            len1 = curs.execute('SELECT COUNT (*) FROM %s
WHERE  len=? AND label = "0" '%self.tabelLatih,(ln,))
            c5n = int(seq1.fetchone()[0])
            len2 = curs.execute('SELECT COUNT (*) FROM %s
WHERE  len=? AND label = "1" '%self.tabelLatih,(ln,))
            c5y = int(seq2.fetchone()[0])
            prob_c5n = (c3n+self.m*0.5)/(n+self.m)
            prob_c5y = (c3y+self.m*0.5)/(y+self.m)

            N =
0.5*prob_c1n*prob_c2n*prob_c3n*prob_c4n*prob_c5n
            Y =
0.5*prob_c1y*prob_c2y*prob_c3y*prob_c4y*prob_c5y
            if Y > N:
```

```
                countP = countP + 1
                countN = countN
                isSerangan=True

            else :
                countN = countN + 1
                countP = countP
                isSerangan=False
            resultRow=[s,d,sq,w,ln,isSerangan,la]
            debugRow=[prob_c1y,prob_c1n,prob_c2y,prob_c2n,pro
b_c3y,prob_c3n,prob_c4y,prob_c4n,prob_c5y,prob_c5n,Y,N,isSera
ngan]
            self.result.append(resultRow)
            self.debug.append(debugRow)
            resultRow=[]
            debugRow=[]
        self.runningTime=time.clock() - t0
        self.jumlahSerangan=positive
        self.jumlahSeranganTerdeteksi=countP
        self.TPR=countP/positive

        if countP > positive:
            self.FPR=math.fabs(positive - countP)/negative

       conn.close()
    def getDebugData(self):
        return self.debug;
    def getResult(self):
        return self.result
    def getTotalSerangan(self):
        return self.jumlahSerangan
    def getTotalSeranganTerdeteksi(self):
        return self.jumlahSeranganTerdeteksi
    def getTPR(self):
       return self.TPR
    def getFPR(self):
       return self.FPR
    def getRunningTime(self):
        return self.runningTime
```

Kode Program 4.5 *Syn Flood Classifier*

Dalam sekali *running,* program *classifier* melakukan pendeteksian sebanyak koneksi jaringan komputer yang terdapat pada *dataset* uji, jika di dalam *dataset* uji terdapat 50 koneksi maka dalam sekali *running* program tersebut melakukan 50 kali pengecekan dan menghasilkan keluaran berupa banyaknya serangan yang terdeteksi dibandingkan dengan kondisi aktual *dataset* uji.

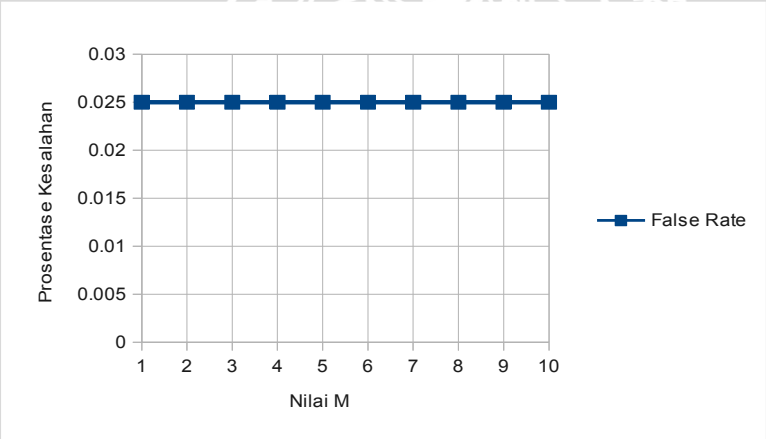
4.3 Implementasi Uji Coba

Pada sub bab ini akan dilakukan pembahasan mengenai pengujian yang telah dilakukan pada sistem dan hasil evaluasi dari klasifikasi hasil sistem.

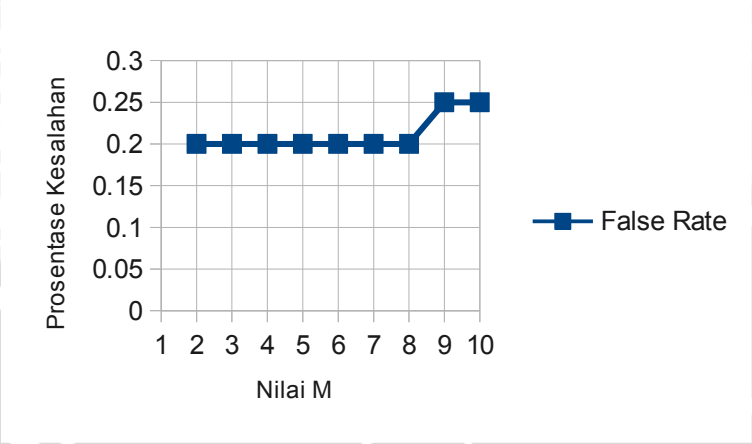
4.3.1 Hasil Evaluasi

Hasil uji coba terhadap sistem dilakukan untuk mengetahui kinerja sistem yang dibangun. Evaluasi tersebut dilakukan dengan membandingkan hasil pendeteksian *syn flood* yang dilakukan oleh sistem dengan klasifikasi *syn flood* sesungguhnya.

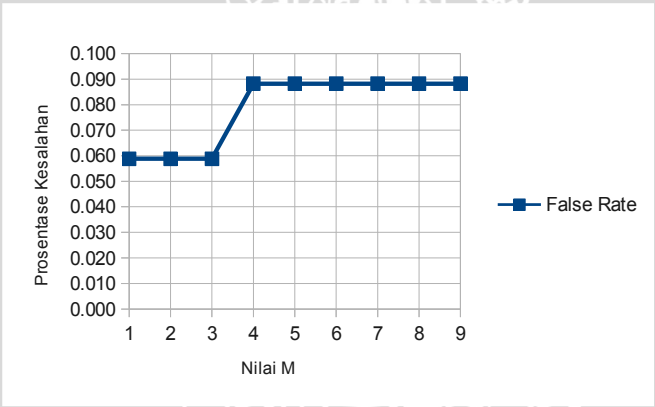
Tahap pertama adalah menentukan nilai M pada *M-Estimates* yang paling optimal sehingga didapatkan akurasi pendeteksian serangan yang maksimal. Pencarian nilai M ini dilakukan dengan cara memasukkan satu persatu nilai pada tiap skenario dan melihat kencederungan akurasi yang dihasilkan. Grafik nilai M pada skenario pertama ditunjukkan pada gambar 4.2, grafik nilai M pada skenario kedua ditunjukkan pada gambar 4.3, dan grafik nilai M pada skenario ketiga ditunjukkan pada gambar 4.4.



Gambar 4.2 nilai M pada Skenario 1



Gambar 4.3 nilai M pada Skenario 2



Gambar 4.4 nilai M pada Skenario 3

Dari grafik diatas bisa diambil kesimpulan bahwa semakin besar nilai M yang diberikan maka tingkat akurasi yang dihasilkan semakin menurun maka percobaan pendeteksian *syn flood* menggunakan *naïve bayes* menggunakan nilai M yang kecil.

Tahap selanjutnya adalah melakukan pengujian pendeteksian syn flood menggunakan *naïve bayes*. Setiap koneksi pada data uji probabilitas *hit rate* dan *false alarm* dimasukkan kedalam sebuah *ROC Curve* untuk mendapatkan gambaran kecenderungan *naïve bayes* dalam

mendeteksi serangan. *ROC Curve* pada skenario pertama ditunjukkan pada gambar 4.5, *ROC Curve* pada skenario kedua ditunjukkan pada gambar 4.6, dan *ROC Curve* pada skenario ketiga ditunjukkan pada gambar 4.7. Secara Keseluruhan *False Alarm* dari ketiga skenario ditujukan oleh tabel 4.1 sedangkan *record* yang bermasalah pada skenario 1 ditunjukan pada tabel 4.2, skenario 2 pada tabel 4.3, dan skenario 3 pada tabel 4.4

Tabel 4.1 *False Alarm Rate*

	Aktual	**Terdeteksi**	***False Alarm***	**Akurasi**
Skenario 1	10	11	0.025	90%
Skenario 2	10	18	0.2	20%
Skenario 3	16	18	0.0588	87.5%

Tabel 4.2 *Record* bermasalah pada skenario 1

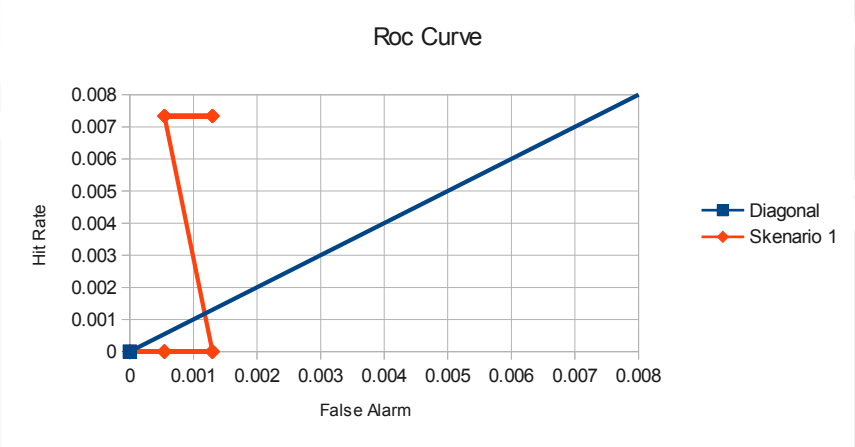
No	***Source***	***Dest***	***Seq***	***Win***	***Len***
1	http	*warmspot*	1	5820	0

Tabel 4.3 *Record* bermasalah pada skenario 2

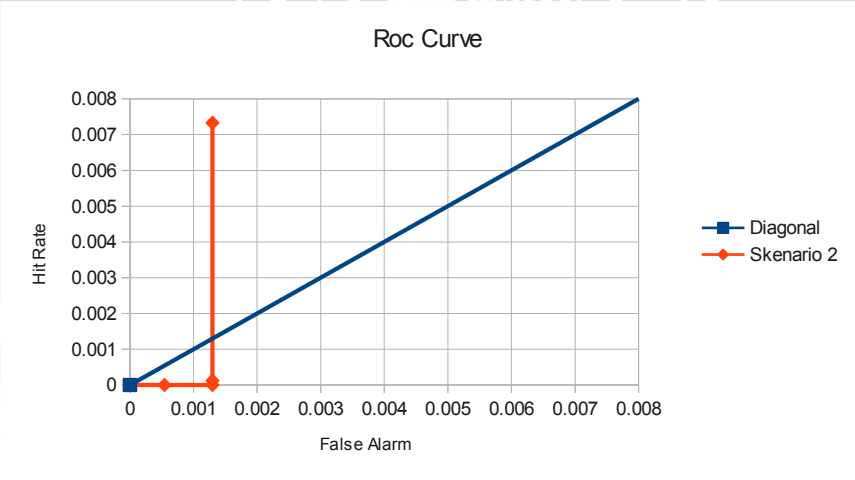
No	***Source***	***Dest***	***Seq***	***Win***	***Len***
1	1491	Http	20	4440	0
2	*Netmap*	Http	3	433	0
3	*Sybase*	Http	23	5880	0
4	*Saiscm*	Http	6	6060	0
5	*Shivadisc*	Http	200	5540	1
6	*Dlswpn*	Http	12	533	0
7	Http	*Remographlm*	1	5444	0
8	Http	*Hydra*	1	5840	0

Tabel 4.4 *Record* bermasalah pada skenario 3

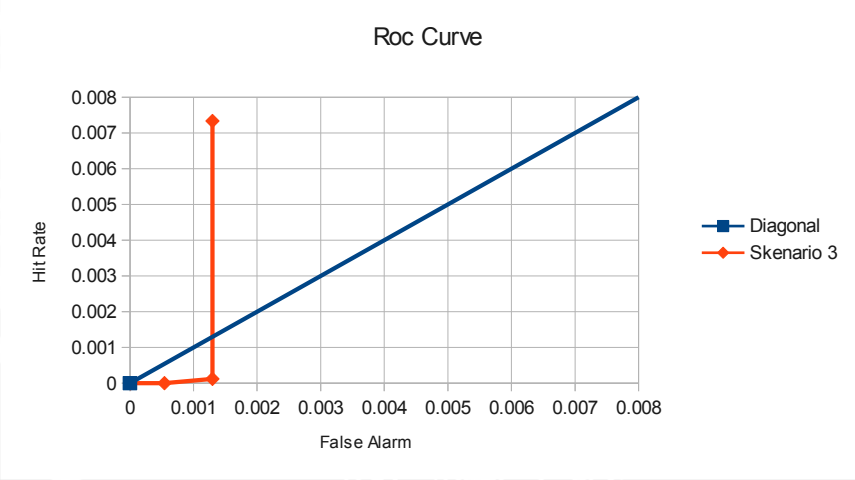
No	*Source*	*Dest*	*Seq*	*Win*	*Len*
1	59497	Http	20	550	0
2	59498	Http	12	550	0



Gambar 4.5 *ROC Curve* pada Skenario 1



Gambar 4.6 *ROC Curve* Skenario 2



Gambar 4.7 *ROC Curve* Skenario 3

4.3.2 Analisa Hasil Percobaan

Hasil uji coba menunjukkan pemilihan nilai M pada *M-Estimates* mempengaruhi tingkat akurasi pendeteksian *naïve bayes*. Seperti terlihat pada gambar 4.2, gambar 4.3, dan gambar 4.4 dimana semakin tinggi nilai M maka tingkat akurasi pendeteksian serangan *syn flood* semakin menurun, hal ini disebabkan karena pemilihan nilai M merepresentasikan keseragaman data yang diuji. Sebagai contoh nilai M = 3 sama artinya tiga *record* pada tiga data uji memiliki kesamaan *pattern*. Jadi semakin tinggi nilai M maka semakin tinggi pula keseragaman *pattern* dari *dataset* sehingga kesalahan pendeteksian semakin besar.

Pada tabel 4.2 diketahui bahwa dari ketiga skenario terjadi *false alarm* dan ketiga skenario tersebut memiliki *false alarm* yang sama yaitu kesalahan dalam mendeteksi koneksi yang bukan serangan dideteksi sebagai serangan. Keadaaan ini bisa dipahami dengan membandingkan hasil *ROC Curve* dari ketiga skenario yang diujikan, garis skenario memiliki area yang lebih besar pada area *Hit Rate* atau *True Positive*. Kondisi ini berarti *classifier*, dalam hal ini *naïve bayes* memiliki kecenderungan lebih baik dalam mengidentifikasi kondisi

positif atau serangan daripada kondisi negatif atau bukan serangan sehingga beberapa kondisi koneksi yang secara aktual bukan serangan diklasifikasikan sebagai serangan oleh *naïve bayes*.

BAB V
PENUTUP

5.1 Kesimpulan

Kesimpulan yang dihasilkan dari skripsi ini adalah.

1. Telah dilakukan implementasi algoritma *naïve bayes* untuk melakukan pendeteksian serangan *syn flood* pada jaringan komputer lokal.
2. *Naïve bayes* sangat dipengaruhi oleh persebaran data dalam melakukan pendeteksian. Kecenderungan kesalahan pendeteksian serangan *syn flood* sangat tergantung dari bagaimana kondisi positif dan negatif dari sebuah *dataset*. Jika sebuah *dataset* memiliki kecenderungan memiliki persebaran positif maka *naïve bayes* memiliki kecenderungan melakukan kesalahan dalam mengidentifikasi data negatif. Selain pola persebaran data pemilihan nilai M ada persamaan M-Estimates berpengaruh terhadap akurasi pendeteksian serangan *syn flood* pada jaringan komputer, semakin tinggi nilai M maka semakin menurun tingkat akurasinya hal ini disebabkan semakin tinggi nya tingkat keseragaman data seiring tingginya nilai M pada *M-Estimates*.

5.2 Saran

Perlu diujicobakan pada jaringan komputer yang lebih luas sehingga bisa diketahui bagaimana algoritma *naïve bayes* melakukan pendeteksian serangan *syn flood* secara *DDoS* (*Distributed Denial Of Service*).

DAFTAR PUSTAKA

Baskara, Andy. Klasifikasi Spam Email Menggunakan Metode Pendekatan Naive Bayes, Malang, 2010

Campos Marcos M., Milenova Boriana L. Creation and deployment of data mining-based intrusion detection systems in Oracle Database l0g: Fourth International Conference on Machine Learning and Applications,Palo Alto, 15-17 December 2005

Daniel Barbara, Ningning Wu, Sushil Jadoja, "Detecting Novel Network Intrusions Using Bayes Estimators", 2002

D. Moore, G. Voelker and S. Savage, "Inferring Internet Denial of Service Activity", Proceedings of USENIX Security Symposium'2001, August 2001.

Dumais, Susan, John Platt, David Heckerman, dan Mehran Sahami. Inductive Learning Algorithms and Representations for Text Categorization. AAAI 98 Workshop on Text Categorization. 1998

Fawcett, Tom.. An Introduction to ROC Analysis, USA, 19 Desember 2005

Meisner, Eric. Naïve Bayes Classifier Example, 2003

Mitchell, T.M. Machine Learning. McGraw-Hill. 1997

Munz Gerhrad. Traffic Anomaly Detection and Cause Identification Using Flow-Level Measurements : Network Architectures and Services NET 2010-06-1 . Series Editor: Georg Carle, Technische Universit¨t M¨nchen, Germany . Technische Universit¨t M¨nchen, Germany. 2009

Ohsita Yuichi, Ata Shingo, and Murata Masayuki. Detecting Distributed Denial-of-Service Attacks by analyzing *TCP* SYN packets statistically: Proceedings of IEEE GLOBECOM 2004, vol. 4, pp. 2043-2049, Dec 2004

RFC 793, Internet Engineering Task Force, 1983

Sungdeok Cha, Junsup Lee, Sangrok Kim, Sanghyun Cho, “ADAM : Web Anomaly Detection Assistant based on Feature Matrix ”. Quality Software, QSIC '09. 9th International Conference. 2009

Thomas Tom. Network Security First-Step,USA: Cisco Press. 2004