

BAB I

PENDAHULUAN

1.1 Latar Belakang Masalah

Keamanan saat ini telah menjadi hal yang sangat penting, terutama dalam sistem informasi. Banyak hal telah dilakukan untuk mencoba mengamankan suatu sistem informasi, salah satunya ialah dengan menggunakan firewall.

Penggunaan *firewall* masih memiliki kelemahan yang masih dapat ditembus, sehingga dicari suatu cara atau metoda yang mampu mengurangi kelemahan dari suatu sistem informasi. Perluasan dari penggunaan *firewall* ini salah satunya ialah dengan menggunakan metoda *port knocking*.

1.2 Tujuan

Makalah ini ditulis dengan tujuan untuk memahami lebih dalam mengenai metoda *port knocking* dan mengimplementasikannya secara sederhana. Dengan

memahami metoda ini diharapkan mampu memberi wawasan mengenai perbaikan dalam system keamanan.

1.3 Perumusan Masalah

Dalam penyusunan makalah ini, penulis merumuskan dan membatasi masalah menjadi:

- definisi dari metoda *port knocking*
- cara kerja metoda *port knocking*
- implementasi *port knocking* secara sederhana

1.4 Metoda Penelitian

Penelitian yang dilakukan untuk penulisan makalah ini dilakukan melalui tahapan sebagai berikut:

1.4.1 Penentuan Topik

Penentuan topik ini dilakukan untuk mengetahui apa yang akan dibahas dan dijelaskan dalam penulisan makalah ini

1.4.2 Studi Literatur

Studi literatur dilakukan dengan mencari artikel atau bacaan yang relevan dari internet untuk kemudian digunakan menjadi acuan dalam penulisan makalah ini. Daftar literatur yang digunakan dapat dilihat pada daftar pustaka.

1.4.3 Penentuan Rule

Pada bagian ini akan ditentukan rule-rule yang harus diterapkan agar metoda *port knocking* ini dapat berjalan dengan baik. Baik itu pada *firewall* maupun pada algoritma pemrograman *port knocking*.

1.4.4 Implementasi dan Pengujian

Setelah rule ditetapkan maka dibuat program yang sesuai sehingga metoda *port knocking* dapat diterapkan. Setelah itu dilakukan pengujian dengan melakukan ‘ketukan’ yang sesuai dan dilihat hasilnya.

1.5 Sistematika Penulisan

Makalah ini ditulis dalam lima bab, yaitu:

- **BAB I: PENDAHULUAN**

Pada bab ini dibahas mengenai latar belakang masalah, tujuan pembuatan makalah, perumusan masalah, dan metoda penelitian.

- **BAB II: PORT KNOCKING**

Pada bab ini dibahas mengenai definisi dari port knocking, cara kerja dari port knocking, keunggulan dan kelemahan dari metoda port knocking.

- **BAB III: IMPLEMENTASI PORT KNOCKING**

Pada bab ini diperlihatkan implementasi dari metoda port knocking secara sederhana yang dapat digunakan pada *home network* dan pengujian yang dilakukan untuk mengetahui apakah program sudah berjalan sesuai yang diinginkan.

- **BAB IV: KESIMPULAN DAN SARAN**

Bab ini berisi kesimpulan dari percobaan yang telah dilakukan dan saran untuk memperbaiki kekurangan yang masih ada sehingga metoda dan program yang ada dapat lebih dikembangkan lagi menjadi lebih baik.

BAB II

PORT KNOCKING

2.1 Definisi

Untuk mendapatkan suatu system yang seratus persen aman dari gangguan dari luar ialah dengan memutuskan hubungan dengan system eksternal. Tapi dengan menutup system tersebut dari system luar, maka system tersebut tidak akan ada lagi gunanya sebab system tersebut tidak dapat lagi diakses dan dipergunakan bahkan oleh yang berhak dan membutuhkan.

Sebuah system yang baik harus memiliki keseimbangan antara keamanan dan fleksibilitas dari system tersebut. Salah satu cara mencapai system komputer seperti demikian ialah dengan menggunakan firewall. Dengan menggunakan firewall, maka kita dapat mendefinisikan *user* yang dapat dipercaya dan yang tidak dapat dipercaya dengan menggunakan alamat IP sebagai kriteria filter.

Kelemahan dari *firewall* ialah bahwa *firewall* tidak mampu membedakan *user* yang dapat dipercaya. *Firewall* hanya mampu membedakan alamat IP yang diasumsikan digunakan oleh orang yang tidak dapat dipercaya.

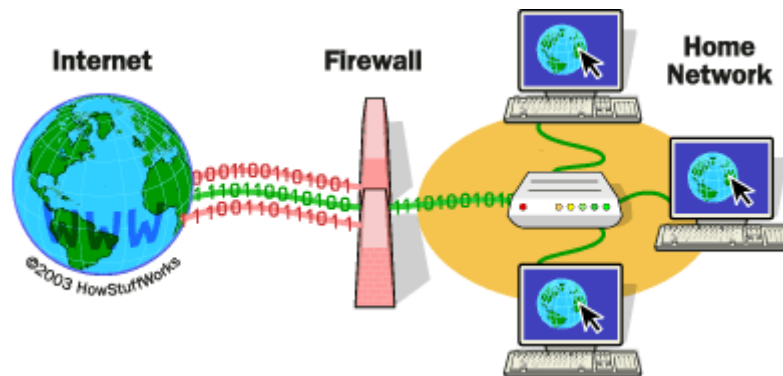
Untuk mendapatkan keamanan yang diperlukan dan kemampuan untuk mengizinkan *user* yang bisa dipercaya untuk mengakses server maka diperlukan suatu metoda yang memenuhi kedua kriteria tersebut. Salah satu metoda baru yang dianggap memiliki kemampuan untuk memenuhi kedua kriteria tersebut adalah *port knocking*.

Port knocking adalah suatu metoda dimana komputer *remote (client)* berkomunikasi dengan sebuah server melalui port yang tertutup. Komunikasi ini sendiri berlangsung secara satu arah, yaitu dari *client* menuju server. Server tidak memberikan respons apapun terhadap *client*.

2.2 Perlengkapan yang Dibutuhkan

Implementasi dari *port knocking* membutuhkan beberapa perlengkapan tambahan, yaitu:

- *Firewall* yang mampu mencatat setiap koneksi ke dalam port tertutup, memonitor logfile secara *real time*, dan mampu merubah aturannya secara dinamis



Gambar 2.1

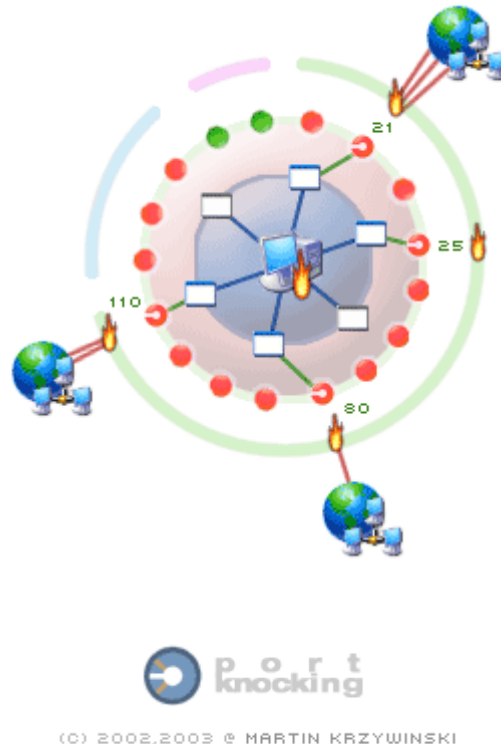
Firewall berfungsi salah satunya untuk melindungi home network

(Diambil dari computer.<http://howstuffworks.com/firewall>)

- *IPChains* atau *IPTables* akan sangat membantu
- Program utama yang menjalankan algoritma *port knocking*

2.3 Cara Kerja *Port Knocking*

Untuk mengaplikasikan metoda *port knocking* ini, pertama-tama semua port yang ada ditutup terlebih dahulu. Meskipun port ditutup, layanan yang disediakan tetap berjalan. Hal ini akan menyebabkan tidak ada orang dari luar yang mampu mengakses layanan tersebut.



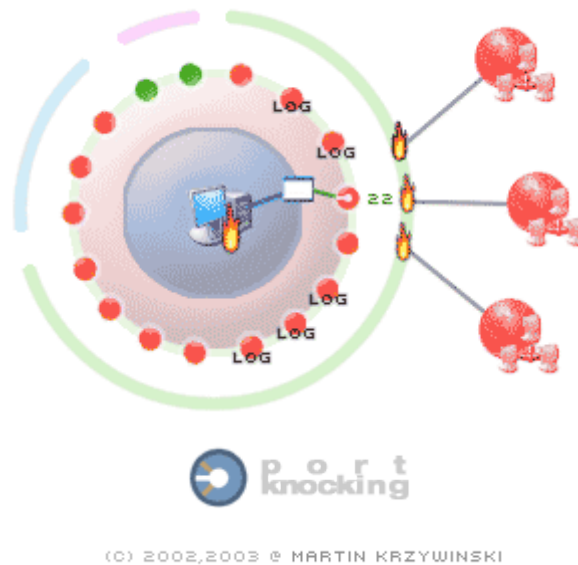
Gambar 2.2

Seluruh port ditutup sehingga tidak ada yang dapat mengakses

(Diambil dari <http://www.portknocking.org/>)

Setelah semua port ditutup, maka beberapa port dicatat segala aktivitas yang terjadi pada port tersebut, termasuk percobaan yang dilakukan untuk mengakses port-port tersebut.

Oleh karena ditutupnya seluruh port ini, metoda *port knocking* ini tidak cocok untuk melindungi semua jenis layanan. Hal ini disebabkan karena koneksi yang dibangun menuju layanan yang dilindungi membutuhkan semacam *'password'* untuk mengaksesnya. Sementara itu, koneksi menuju layanan-layanan publik dapat datang dari mana saja sehingga metoda ini kurang cocok untuk melindungi layanan bersifat publik.

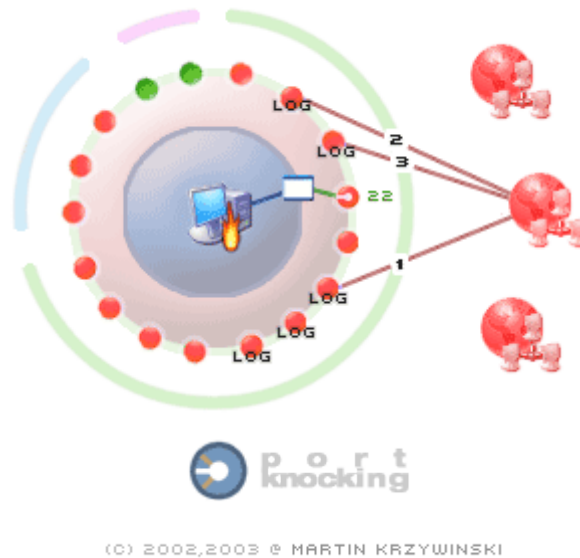


Gambar 2.3

Contoh port 22 yang dilindungi dengan metoda port knocking

(Diambil dari : <http://www.portknocking.org/>)

Apabila ada suatu saat ada seorang *user* yang ingin mengakses layanan tertentu, misalnya SSH, *user* tersebut akan mengawali koneksi dengan melakukan percobaan koneksi ke port-port tertentu yang dimonitor dengan urutan tertentu juga. Urutan ‘ketukan’ ini hanya diketahui oleh orang-orang tertentu yang dipercaya (menjadi semacam *password* untuk menggunakan layanan tertentu). Dengan menggunakan ketukan ini, maka alamat IP seseorang yang oleh *firewall* sebelumnya dianggap sebagai *user* yang tidak dapat dipercaya akan berubah status menjadi *user* yang dapat dipercaya, sehingga *user* akan diizinkan untuk mengakses atau melakukan koneksi terhadap layanan yang disediakan.



Gambar 2.4

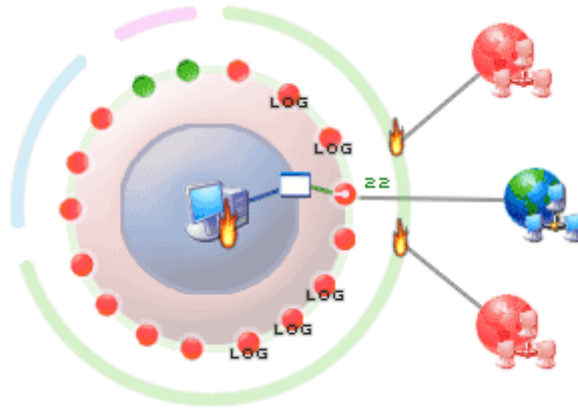
Seorang *user* melakukan ketukan rahasia

(Diambil dari <http://www.portknocking.org/>)

Fase mengetuk ini dapat dianalogikan sebagai ketukan pada pintu yang tidak terlihat. Fase ini disebut demikian karena *user* tidak dapat mengetahui port-port mana yang sedang dimonitor dan dicatat, dan demikian juga dengan orang lain (terutama yang tidak dipercaya).

Setelah *firewall* mendeteksi adanya ketukan dengan urutan ketukan yang benar pada port yang benar pula, maka aturan (*rule*) dari *firewall* akan diubah sehingga alamat IP yang sesuai dengan yang ada pada log dari *firewall* akan diizinkan untuk melakukan koneksi dengan port tertentu. Seperti terlihat pada gambar 2.4.





Gambar 2.5

Seorang *user* diizinkan melakukan koneksi

(Diambil dari <http://www.portknocking.org/>)

Meskipun sebuah port telah dibuka untuk melakukan koneksi dengan *user* tertentu, alamat IP yang lain tetap diblok sehingga tidak dapat mengakses port tersebut seperti terlihat pada gambar di atas. Setelah *user* selesai dalam melakukan koneksi dan menyelesaikan sesinya, maka *user* tersebut melakukan lagi ketukan pada port yang dimonitor untuk memberi tanda pada *firewall* untuk menutup port yang sebelumnya dibuka.

2.4 Format Ketukan

Format ketukan yang digunakan dalam metoda *port knocking* ini bergantung pada seberapa banyak informasi yang ingin disampaikan melalui ketukan kepada *server*. Berikut ini adalah beberapa contoh ketukan yang dapat diaplikasikan berdasarkan keperluan tertentu:

- Port tunggal dengan pemetaan tetap (*fixed*)

Apabila koneksi yang diperlukan (layanan yang disediakan) hanya bersumber pada satu port saja (misalkan port 22 untuk SSH), maka format ketukan yang diperlukan hanya melibatkan tiga port saja. Sebagai contoh, port 100, 101, dan 102 dimonitor. Ketukan yang dapat dipakai dan fungsinya adalah sebagai berikut:

Port yang diketuk	Aksi yang dilakukan
100,102,101	Membuka port 22 untuk alamat IP yang bersangkutan
101,102,100	Menutup port 22 untuk alamat IP yang bersangkutan
101,100,102	Menutup port 22 dan mengabaikan ketukan berikutnya dari alamat IP yang bersangkutan.

Tabel 1

Ketukan untuk port tunggal dengan pemetaan tetap

Dari tabel di atas dapat dilihat contoh ketukan yang dipakai untuk membuka dan menutup port 22. Sedangkan port ketiga dapat digunakan apabila seorang *user* melakukan *port knocking* dari komputer yang bukan miliknya sendiri dan tidak dipercaya. Ketukan ketiga ini berfungsi untuk penduplikasian *port knocking* oleh administrator dari *host* tersebut. Hal ini dengan asumsi bahwa ketukan tidak di-‘tangkap’ oleh orang lain dan diduplikasikan sebelum sesi berakhir. Hanya saja, kelemahan dari ketukan ketiga ini ialah bahwa orang masih mampu mengetuk dengan menggunakan alamat IP yang lain sehingga ketukan ketiga tersebut akan menjadi tidak berguna.

Kelemahan dari ketukan ini ialah port yang digunakan sangat sedikit, sehingga apabila ada orang yang telah mengetahui port yang dimonitor maka akan mudah bagi orang tersebut untuk mencoba ketukan yang tepat.

- Multiple port dengan pemetaan dinamis

Apabila sebuah *server* menyediakan beberapa jenis layanan, maka ketukan seperti pada point pertama sudah tidak dapat lagi digunakan. Salah satu bentuk ketukan yang dapat diaplikasikan adalah sebagai berikut:

Header	Payload	Checksum	Footer
--------	---------	----------	--------

Tabel 2

Format ketukan multiple port

Sebagai contoh, digunakan port 500 sampai 510 memonitor ketukan, maka format ketukan dapat ditulis menjadi:

510,503,507	50a, 50b, 50c, 50d	$50\{(a+b+c+d) \bmod 10\}$	510,501,509
-------------	--------------------	----------------------------	-------------

Tabel 3

Contoh header dan footer

Bagian *payload* merupakan bagian yang menyatakan port mana yang diminta untuk dibuka, yaitu port abcd. Bagian *checksum* akan memastikan port yang diminta benar dan tidak ada kesalahan dalam pengetukan. Sebagai contoh, seorang *user* ingin mengakses port 150, maka ketukan yang dilakukan adalah sebagai berikut:

510,503,507 500,501,505,500 506 510,501,509

Ketika ketukan ini dideteksi untuk alamat IP tertentu maka port 150 akan dibuka. Apabila port 150 sudah dibuka untuk alamat IP tersebut, maka dengan adanya ketukan ini akan menutup port tersebut untuk alamat IP tersebut.

Beberapa informasi dapat ditambahkan pada ketukan, seperti panjang sesi yang diperkirakan, alamat IP yang diinginkan untuk memiliki akses, dan sebagainya.

Salah satu cara untuk mempersulit peniruan ketukan ini, dapat digunakan beberapa range port secara bersamaan, sebagai contoh port 401-405,506-510. Dengan range port ini dapat diselipkan beberapa port (yang tidak dimonitor) pada ketukan sehingga orang yang berusaha meniru ketukan akan tertipu. Sebagai contoh, pada kasus di atas ialah:

510,403,407 400,501,401,505,405,500,400 406,506 510,401,409

Dapat dilihat pada contoh di atas bahwa jumlah ketukan menjadi bertambah namun informasi yang disampaikan tetap sama. Hal tersebut terjadi karena port-port diluar range yang telah ditentukan tidak dimonitor dan tidak dianggap sebagai suatu ketukan. Meskipun demikian ada sedikit perubahan pada aturan pengetukan, dimana digit 1 sampai 5 menggunakan port 400-405, sedang digit 6 sampai 10 menggunakan port 506 sampai 510. Tentu saja *user* yang dapat dipercaya harus mengetahui port mana saja yang dimonitor.

- Pemetaan dengan menggunakan enkripsi

Untuk menambah keamanan dari *port knocking* maka dapat digunakan enkripsi pada bagian informasinya. Dengan enkripsi, maka ketukan yang dilakukan akan semakin sulit untuk ditebak.

Misalkan akan dikirim 10 nomor (k_1, k_2, \dots, k_{10}) dengan menggunakan *port knocking*. Asumsikan server menyediakan port 400-655 untuk dimonitor. Pertama, nomor tersebut akan dienkrapsikan dengan menggunakan algoritma tertentu (misalkan RSA atau Blowfish), lalu dilakukan *byte encoding*. Kemudian hasilnya dipetakan untuk menghasilkan ketukan yang sebenarnya. Server lalu membalik langkah yang telah dilakukan oleh *user*.

Beberapa metoda yang telah disebutkan di atas hanyalah beberapa ide yang tertuang, mengenai aplikasinya sendiri dalam jaringan akan sangat tergantung dari desain administrator masing-masing jaringan dan kebutuhan dari jaringan tersebut.

2.5 Keunggulan dan Kelemahan *Port Knocking*

Setiap sistem keamanan yang didesain oleh manusia tidak ada yang sempurna. Setiap sistem pasti memiliki kelebihan dan kelemahan masing-masing, demikian juga dengan *port knocking*. Berikut ini akan dibahas beberapa mengenai keunggulan dan kelemahannya.

2.5.1 Keunggulan *Port Knocking*

Ada beberapa hal yang membuat *port knocking* lebih unggul daripada sistem keamanan yang biasa, yaitu:

- *Port knocking* merupakan metoda yang terselubung untuk melakukan otentifikasi dan perpindahan informasi menuju sebuah sistem yang terhubung dengan jaringan, namun tidak memiliki port yang terbuka. Akan sulit bagi siapapun untuk mengetahui apakah port dimonitor atau tidak. Selain itu, percobaan yang dilakukan untuk menebak urutan ketukan yang dilakukan secara 'kasar' akan terdeteksi dan alamat IP yang melakukannya dapat diblok.
- Informasi mengalir dalam bentuk percobaan koneksi dan bukannya paket data. Tanpa mengetahui keberadaan sistem ini dan metoda yang digunakan, maka akan sulit mendeteksi penggunaan otentifikasi dengan memonitor *traffic*.
- Karena otentifikasi dibangun ke dalam urutan ketukan, aplikasi yang ada tidak lagi perlu dirubah.

2.5.2 Kelemahan *Port Knocking*

Ada beberapa hal yang menjadi kelemahan dalam pengaplikasian metoda *port knocking* ini:

- Untuk menggunakan metoda ini diperlukan program tersendiri untuk melakukan ketukan.
- Dalam penggunaan metoda ini dibutuhkan sejumlah port yang perlu dialokasikan secara khusus untuk digunakan oleh sistem sehingga port tersebut tidak dapat digunakan untuk keperluan lain
- Metoda ini memodifikasi *rule* dari *firewall* secara otomatis. Hal tersebut memerlukan penanganan yang hati-hati. Bila terjadi kegagalan dimana sistem tidak mampu mengenali atau mendengar ketukan yang telah dilakukan, maka tidak ada yang mampu melakukan koneksi dari luar.

BAB III

IMPLEMENTASI PORT KNOCKING

3.1 Program yang Berperan dalam Port Knock

Untuk implementasi *port knocking* kali ini penulis menggunakan bahasa C sebagai bahasa pemrograman. Implementasi *port knocking* ini menggunakan dua buah program *executable* agar dapat berfungsi dengan baik, yaitu *daemon* dan *client*. *Daemon* berfungsi untuk memonitor log dari firewall dan melakukan aksi sesuai dengan *rule* yang telah ditetapkan. Program *daemon* sendiri terdiri dari empat program yang saling berkaitan satu sama lain, yaitu file *daemon.c*, *sublist.c*, *list.c*, *baca.c*. Setiap file mempunyai fungsinya masing-masing, yaitu:

- *daemon.c* berfungsi sebagai program utama yang akan menjalankan algoritma *port knocking*.
- *Sublist.c*, *list.c*, dan *apprList.c* berfungsi untuk mendukung struktur data yang akan menyimpan data berupa alamat IP, alamat MAC, port, dan lain-lain.

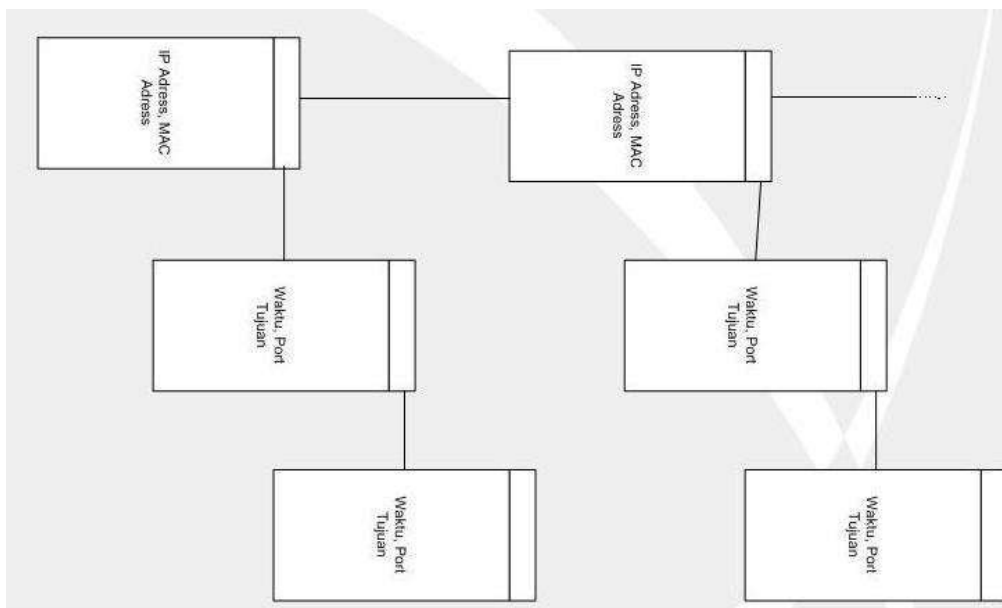
- Baca.c berfungsi untuk menyediakan subrutin yang akan digunakan untuk membaca logfile.

Program *client* berfungsi untuk melakukan ketukan terhadap server yang menyediakan layanan, program ini hanya terdiri dari satu program saja, yaitu *client.c*.

Seperti telah diketahui bahwa log dari firewall akan digunakan sebagai sumber data, logfile dari firewall sendiri terdapat pada file `/var/log/syslog`.

3.2 Struktur Data

Untuk mempermudah dalam pemrosesan data yang diperoleh dari log firewall maka setiap data yang masuk akan dibentuk ke dalam struktur list. Berikut ini adalah gambar dari struktur list yang digunakan pada program ini:



Gambar 3.1

Struktur data list

3.3 Asumsi yang Digunakan dalam Pembuatan Program

Beberapa asumsi digunakan untuk membuat program ini sehingga penulis mampu mensimulasikan algoritma *port knocking*. Asumsi-asumsi tersebut adalah sebagai berikut:

- server menyediakan dua buah layanan, yaitu SSH (port 22) dan smtp (port 25)
- program ini hanya berfungsi untuk membukakan port dan tidak perlu menyediakan layanan yang berhubungan dengan port tersebut
- port yang digunakan untuk memonitor ketukan telah ditentukan dan diketahui baik oleh server maupun oleh *user*
- metoda ketukan yang dilakukan ialah metoda kedua (*multiple port* dengan pemetaan dinamis) dengan beberapa tambahan ketukan pada port yang tidak dimonitor

3.4 Port dan Format Ketukan yang Digunakan

Seperti telah disebutkan di atas, diasumsikan bahwa server menyediakan dua buah layanan. Selain itu port yang digunakan pun telah ditentukan dan diketahui.

Port yang digunakan untuk memonitor ketukan yang masuk dibagi-bagi menjadi 3 bagian, yaitu

<i>Digit</i>	<i>Port</i>
0, 1, 2	400, 401, 402
3, 4, 5	503, 504, 505
6, 7, 8, 9, 10	606, 607, 608, 609, 610

Tabel 4

Port yang dimonitor

Ketukan yang digunakan sebagai syarat untuk membuka port SSH dan smtp pada program ini adalah sebagai berikut:

<i>Header</i>	<i>Payload (SSH)</i>	<i>Checksum</i>	<i>Footer</i>
610,400,503	402,504,505,402,	503	606,505,402

Tabel 5

Implementasi ketukan untuk membuka port 22

<i>Header</i>	<i>Payload (smtp)</i>	<i>Checksum</i>	<i>Footer</i>
610, 400, 503	402, 607, 400, 401	400	606,505,402

Tabel 6

Implementasi ketukan untuk membuka port 25

Kedua ketukan yang terdapat pada tabel 5 dan 6 adalah urutan ketukan yang akan digunakan oleh program *daemon.c* sebagai syarat untuk membuka port yang bersesuaian. Namun, karena adanya pemisahan port yang dimonitor seperti pada tabel 4, maka untuk program *client.c* akan memiliki format ketukan yang berbeda, yaitu dengan tambahan beberapa port yang berada di luar *range* port yang dimonitor. Hal ini bertujuan untuk mengelabui orang yang mencoba mendengarkan ketukan.

Urutan ketukan yang digunakan pada program *client.c* adalah sebagai berikut:

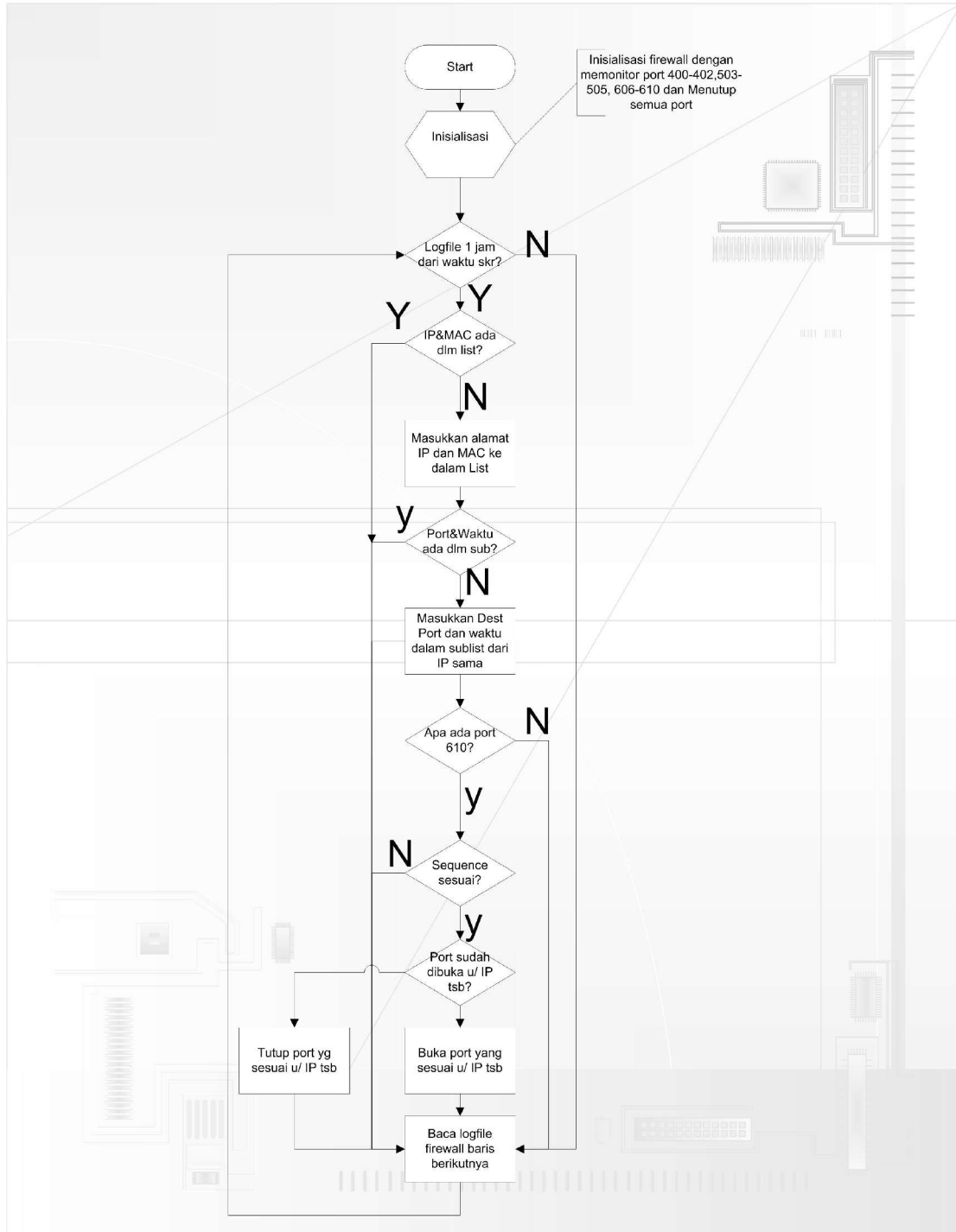
Header	610, 701, 400, 305, 503, 203
Payload (SSH) + Checksum	402, 605, 504, 403, 505, 702, 402, 205, 503, 501
Payload(smtp) + checksum	402, 604, 607, 708, 400, 230, 401, 350, 400, 509
Footer	606, 602, 505, 408, 402, 604

Tabel 7

Implementasi ketukan pada *client.c*

3.5 Algoritma Pemrograman

Algoritma pemrograman yang dipakai dalam program daemon ini adalah sebagai berikut:



Gambar 3.2

Algoritma program daemon.c

Pada intinya, program daemon.c ini akan menginisialisasi semua variabel yang dibutuhkan serta *rule* dari *firewall* yang diperlukan untuk memonitor port yang digunakan dan menolak koneksi dari host manapun. Kemudian logfile dari firewall dibaca, dan diambil data yang waktunya kurang dari 1 jam dari sekarang.

Setelah itu, setiap data IP dan MAC akan dimasukkan ke dalam list, kemudian port dan waktu yang bersesuaian akan dimasukkan ke dalam sublist yang sesuai dengan IP dan MAC tersebut. Kemudian diperiksa ketukan pada port 610 sebagai awal dari *header*. Apabila ditemukan, kemudian akan diperiksa urutan ketukan berikutnya apakah sesuai dengan syarat yang ada. Apabila ketukan sudah sesuai, akan diperiksa apakah port yang bersesuaian dengan ketukan sudah dibuka untuk alamat IP yang bersangkutan. Bila sudah, maka port tersebut akan ditutup sedang bila belum, port tersebut akan dibuka untuk alamat IP yang bersangkutan.

Berikut ini akan disertakan potongan program main dari daemon.c:

```
int main()
{
    time_t now;

    List logList = NULL;

    execCmd("iptables",flush);
    execCmd("iptables",init1);
    execCmd("iptables",init2);
    execCmd("iptables",init3);
    execCmd("iptables",init4);
```

```
execCmd("iptables",init5);

execCmd("iptables",init5);

execCmd("iptables",drop);

for (;;)
{
    time(&now);

    logList = readLog();

    PrintList(logList);

    //printf("Before trace\n");

    traceSurf(logList);

    printf("LogApproved\n-----\n");

    PrintApprlist(LogApproved);

    //printf("Before destroy\n");

    DestroyList (logList);

    //printf("After destroy\n");

    logList = NULL;

    printf("%s-----\n1 cycle of ReadLog\n",ctime(&now));

    sleep(1);

}

return 0;

}
```

Gambar 3.3

Program main dari daemon.c

3.6 Pengujian

Pengujian dilakukan dengan menggunakan sebuah komputer dengan menggunakan *operating system* Linux Mandrake 9.2, dengan bantuan iptables untuk memodifikasi rule dari *firewall*.

3.6.1 Metoda Pengujian

Pengujian dilakukan hanya dengan menggunakan satu komputer karena keterbatasan sarana yang dimiliki oleh penulis dan juga keterbatasan waktu.

Metoda pengujian dilakukan dengan mengirimkan ketukan dari localhost menuju localhost sendiri. Kemudian akan dilihat rule dari *firewall* apakah sudah berubah atau belum. Setelah itu dilakukan port scanning dengan menggunakan nmap menuju diri sendiri.

3.6.2 Hasil Pengujian

Untuk menjalankan program daemon, maka perlu dilakukan *compiling* terlebih dahulu program-program *daemon.c*, *sublist.c*, *list.c*, *apprList.c*, dan *Baca.c*. Kemudian dibuat program *executable* dengan nama *daemon*. Perintah yang digunakan adalah sebagai berikut:

```
[root@bivanh Port Knocking]#gcc -o daemon daemon.c sublist.c list.c apprList.c Baca.c
```

Setelah program daemon dibuat file *executable*-nya maka program client juga perlu dibuat file *executable* juga. Pertama bagian body yang mengkodekan ketukan untuk SSH dijadikan *comment*. Setelah itu di-*compile* dengan menggunakan perintah sebagai berikut:

```
[root@bivanh Port Knocking]#gcc -o clientsmtp client.c
```

Kemudian bagian body yang mengkodekan ketukan untuk SSH dikembalikan menjadi perintah dan body untuk smtp dijadikan *comment*. Setelah itu di-*compile* dengan menggunakan perintah sebagai berikut:

```
[root@bivanh Port Knocking]#gcc -o clientSSH client.c
```

Langkah selanjutnya ialah menjalankan program daemon untuk 'mendengarkan' ketukan. Setelah program daemon dijalankan maka server telah siap mendengarkan dan rule dari *firewall* juga telah berubah. Untuk mengetahui rule dari *firewall* yang saat ini aktif, digunakan perintah:

```
[root@bivanh Port Knocking]#iptables -L > firewall.txt
```

Berikut ini adalah isi dari file *firewall.txt*:

```
Chain INPUT (policy ACCEPT)
target prot opt source destination
LOG tcp -- anywhere anywhere tcp dpts:400:402 LOG level warning
LOG udp -- anywhere anywhere udp dpts:400:402 LOG level warning
LOG tcp -- anywhere anywhere tcp dpts:503:505 LOG level warning
LOG udp -- anywhere anywhere udp dpts:503:505 LOG level warning
LOG tcp -- anywhere anywhere tcp dpts:606:npmp-local LOG level warning
LOG tcp -- anywhere anywhere tcp dpts:606:npmp-local LOG level warning
DROP all -- anywhere anywhere

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination
```

Gambar 3.4

Rule firewall setelah program *daemon.c* dijalankan

Dari isi file *firewall.txt* di atas, dapat dilihat bagaimana rule dari *firewall* telah berubah. Saat ini *firewall* dalam keadaan mendengarkan semua aktivitas yang terjadi

pada port 400-402, 503-505, 606-610, baik itu yang menggunakan protokol tcp maupun udp. Selain itu, *firewall* juga akan menjatuhkan setiap paket yang datang darimanapun menuju manapun. Dengan demikian maka tidak ada yang dapat mengakses satu port pun dari luar. Dan hal tersebut dapat dibuktikan dengan menggunakan nmap. Perintah yang digunakan adalah sebagai berikut :

```
[root@bivanh Port Knocking]# nmap localhost > map.txt
```

Hasil dari *mapping* adalah sebagai berikut:

```
Starting nmap 3.30 ( http://www.insecure.org/nmap/ ) at 2004-06-30 23:28 EDT
Note: Host seems down. If it is really up, but blocking our ping probes, try -P0
Nmap run completed -- 1 IP address (0 hosts up) scanned in 12.077 seconds
```

Gambar 3.5

Hasil mapping saat semua port ditutup

Dari hasil di atas dapat dilihat bahwa nmap menganggap host dalam keadaan down karena seluruh ping yang diberikan diblok. Oleh karena itu, dicoba dengan menggunakan *option -P0*.

```
[root@bivanh Port Knocking]# nmap -P0 localhost > map.txt
```

Hasilnya ditaruh dalam file map.txt. Isi dari file tersebut adalah sebagai berikut:

```
Starting nmap 3.30 ( http://www.insecure.org/nmap/ ) at 2004-06-30 22:47 EDT
All 1644 scanned ports on localhost (127.0.0.1) are: filtered
Nmap run completed -- 1 IP address (1 host up) scanned in 1337.265 seconds
```

Gambar 3.6

Hasil mapping saat semua port ditutup dengan option -P0

Dari hasil *mapping* diketahui ada 1 host yang aktif, namun tidak diketahui apa ada port yang terbuka atau tidak dan apakah ada layanan yang aktif di balik port tersebut.

Setelah dilihat kondisi saat belum ada ketukan, maka ketukan pun dilakukan. Pertama-tama dilakukan ketukan untuk membuka port 25. File `clientsmtp` pun dijalankan.

```
[root@bivanh Port Knocking]# ./clientsmtp
```

Tampilan di layar akan menjadi seperti berikut:

```
localhost::610Connection out...
localhost::701Connection out...
localhost::400Connection out...
localhost::305Connection out...
localhost::503Connection out...
localhost::203Connection out...
localhost::402Connection out...
localhost::604Connection out...
localhost::607Connection out...
localhost::708Connection out...
localhost::400Connection out...
localhost::230Connection out...
localhost::401Connection out...
localhost::350Connection out...
localhost::400Connection out...
localhost::509Connection out...
localhost::606Connection out...
```

```
localhost::602Connection out...
localhost::505Connection out...
localhost::408Connection out...
localhost::402Connection out...
localhost::604Connection out...
```

Gambar 3.7

Ketukan saat file clientsmtp dijalankan

Tampilan di atas menunjukkan localhost sebagai host yang dituju, nomor port yang diketuk dan Connection out yang bertujuan agar program tidak perlu menunggu waktu *timeout*. Ketukan yang dilakukan pun sudah sesuai dengan ketukan yang didefinisikan sebelumnya pada Tabel 7.

Setelah ketukan tersebut, maka rule yang ada akan berubah dan ditambahkan sebuah rule baru yang akan mengizinkan alamat IP tertentu (dalam hal ini localhost 127.0.0.1) untuk mengakses port 25 (smtp). Berikut ini adalah rule dari *firewall* yang sudah dimodifikasi.

```
Chain INPUT (policy ACCEPT)
target  prot opt source      destination
ACCEPT  tcp  -- localhost  anywhere    tcp dpt:smtp
LOG     tcp  -- anywhere  anywhere    tcp dpts:400:402 LOG level warning
LOG     udp  -- anywhere  anywhere    udp dpts:400:402 LOG level warning
LOG     tcp  -- anywhere  anywhere    tcp dpts:503:505 LOG level warning
LOG     udp  -- anywhere  anywhere    udp dpts:503:505 LOG level warning
LOG     tcp  -- anywhere  anywhere    tcp dpts:606:npmp-local LOG level warning
LOG     tcp  -- anywhere  anywhere    tcp dpts:606:npmp-local LOG level warning
DROP    all  -- anywhere  anywhere
```

Chain FORWARD (policy ACCEPT)			
target	prot	opt source	destination
Chain OUTPUT (policy ACCEPT)			
target	prot	opt source	destination

Gambar 3.8

Rule firewall setelah clientsmtp dijalankan

Terlihat pada tabel di atas bahwa port 25 (smtp) memiliki kebijakan ACCEPT untuk SOURCE yang berasal dari *localhost*. Hal tersebut dipertegas lagi dengan penggunaan nmap untuk melihat port yang terbuka.

Perintah digunakan adalah sebagai berikut:

```
[root@bivanh Port Knocking]# nmap localhost > map.txt
```

dan hasilnya adalah sebagai berikut:

```
Starting nmap 3.30 ( http://www.insecure.org/nmap/ ) at 2004-07-01 00:10 EDT
Note: Host seems down. If it is really up, but blocking our ping probes, try -P0
Nmap run completed -- 1 IP address (0 hosts up) scanned in 12.088 seconds
```

Gambar 3.9

Hasil mapping setelah port 25 terbuka

Hasil di atas masih sama dengan hasil sebelumnya dimana tidak terlihat port yang dibuka. Namun apabila menggunakan perintah yang lain, yaitu:

```
[root@bivanh Port Knocking]# nmap -P0 localhost > map.txt
```

maka hasil yang diperoleh menjadi:

```
Starting nmap 3.30 ( http://www.insecure.org/nmap/ ) at 2004-07-01 00:14 EDT

Interesting ports on localhost (127.0.0.1):

(The 1643 ports scanned but not shown below are in state: filtered)

Port      State      Service
25/tcp    open       smtp

Nmap run completed -- 1 IP address (1 host up) scanned in 163.014 seconds
```

Gambar 3.10

Hasil mapping setelah port 25 dibuka dengan option -P0

Dapat dilihat bahwa port 25 sudah terdeteksi dan dalam keadaan open. Hal tersebut menunjukkan bahwa port 25 telah dapat diakses oleh alamat IP localhost.

Apabila program clientSMTP dijalankan sekali lagi maka port akan kembali menutup dan rule yang mengizinkan *localhost* mengakses port 25 akan dihapus.

```
Chain INPUT (policy ACCEPT)

target    prot opt source      destination
LOG      tcp  -- anywhere   anywhere    tcp dpts:400:402 LOG level warning
LOG      udp  -- anywhere   anywhere    udp dpts:400:402 LOG level warning
LOG      tcp  -- anywhere   anywhere    tcp dpts:503:505 LOG level warning
LOG      udp  -- anywhere   anywhere    udp dpts:503:505 LOG level warning
LOG      tcp  -- anywhere   anywhere    tcp dpts:606:npmp-local LOG level warning
LOG      tcp  -- anywhere   anywhere    tcp dpts:606:npmp-local LOG level warning
DROP     all  -- anywhere   anywhere

Chain FORWARD (policy ACCEPT)

target    prot opt source      destination
```

```
Chain OUTPUT (policy ACCEPT)
target  prot opt source      destination
```

Gambar 3.11

Rule firewall setelah clientsmtp dijalankan kembali dan port 25 tertutup

Terlihat bahwa rule yang mengizinkan localhost untuk mengakses port 25 telah dihapus.

Demikian juga dengan file clientSSH, setelah file tersebut dijalankan akan memberikan tampilan:

```
localhost::610Connection out...
localhost::701Connection out...
localhost::400Connection out...
localhost::305Connection out...
localhost::503Connection out...
localhost::203Connection out...
localhost::402Connection out...
localhost::605Connection out...
localhost::504Connection out...
localhost::403Connection out...
localhost::505Connection out...
localhost::702Connection out...
localhost::402Connection out...
localhost::205Connection out...
localhost::503Connection out...
localhost::501Connection out...
```

```
localhost::606Connection out...
localhost::602Connection out...
localhost::505Connection out...
localhost::408Connection out...
localhost::402Connection out...
localhost::604Connection out...
```

Gambar 3.12

Ketukan yang dilakukan file clientSSH

Keterangan mengenai tampilan ini juga sama dengan clientsmtp. Hasilnya adalah bahwa rule dari *firewall* akan ditambahkan pada baris pertama menjadi:

```
Chain INPUT (policy ACCEPT)
target  prot opt source      destination
ACCEPT  tcp  -- localhost anywhere    tcp dpt:ssh
LOG     tcp  -- anywhere anywhere    tcp dpts:400:402 LOG level warning
LOG     udp  -- anywhere anywhere    udp dpts:400:402 LOG level warning
LOG     tcp  -- anywhere anywhere    tcp dpts:503:505 LOG level warning
LOG     udp  -- anywhere anywhere    udp dpts:503:505 LOG level warning
LOG     tcp  -- anywhere anywhere    tcp dpts:606:npmp-local LOG level warning
LOG     tcp  -- anywhere anywhere    tcp dpts:606:npmp-local LOG level warning
DROP    all  -- anywhere anywhere

Chain FORWARD (policy ACCEPT)
target  prot opt source      destination

Chain OUTPUT (policy ACCEPT)
```

target	prot	opt	source	destination
--------	------	-----	--------	-------------

Gambar 3.13

Rule firewall setelah clientSSH dijalankan

Terlihat *rule* yang pertama akan mengizinkan localhost untuk mengakses port 22 dengan protokol tcp. Hasil ini diperkuat dengan penggunaan nmap. Hasil dari nmap akan menunjukkan:

```
Starting nmap 3.30 ( http://www.insecure.org/nmap/ ) at 2004-07-01 00:56 EDT
Interesting ports on localhost (127.0.0.1):
(The 1643 ports scanned but not shown below are in state: filtered)
Port      State  Service
22/tcp    open  ssh

Nmap run completed -- 1 IP address (1 host up) scanned in 152.001 seconds
```

Gambar 3.14

Hasil mapping dengan option -PO setelah clientSSH dijalankan

Terlihat bahwa port 22 sudah dapat terdeteksi oleh localhost dan dapat diakses. Apabila program clietSSH dijalankan sekali lagi maka rule yang mengizinkan localhost mengakses port 22 akan dihapus dan port 22 kembali tertutup.

Chain INPUT (policy ACCEPT)				
target	prot	opt	source	destination
LOG	tcp	--	anywhere	anywhere tcp dpts:400:402 LOG level warning
LOG	udp	--	anywhere	anywhere udp dpts:400:402 LOG level warning
LOG	tcp	--	anywhere	anywhere tcp dpts:503:505 LOG level warning
LOG	udp	--	anywhere	anywhere udp dpts:503:505 LOG level warning
LOG	tcp	--	anywhere	anywhere tcp dpts:606:npmp-local LOG level warning

```
LOG    tcp -- anywhere    anywhere    tcp dpts:606:npmp-local LOG level warning
DROP   all -- anywhere    anywhere

Chain FORWARD (policy ACCEPT)
target  prot opt source      destination

Chain OUTPUT (policy ACCEPT)
target  prot opt source      destination
```

Gambar 3.15

Rule firewall setelah clientSSH dijalankan kembali dan port 22 tertutup

Hasil dari nmap menunjukkan hal yang sama dengan gambar 3.5. Tidak ada port yang aktif dan terdeteksi.

```
Starting nmap 3.30 ( http://www.insecure.org/nmap/ ) at 2004-07-01 01:00EDT
All 1644 scanned ports on localhost (127.0.0.1) are: filtered

Nmap run completed -- 1 IP address (1 host up) scanned in 1337.265 seconds
```

Gambar 3.16

Hasil mapping setelah port 22 ditutup

Hasil pengujian memperlihatkan bahwa program yang mengimplementasikan metoda *port knocking* ini sudah dapat diterapkan dalam komputer pribadi yang memiliki lalu lintas yang tidak terlalu padat. Program ini belum diuji pada jaringan yang memiliki lalu lintas yang padat, namun diperkirakan akan membuat server bekerja lebih berat sebab log file akan membesar sehingga yang harus dibaca oleh program ini juga akan bertambah banyak.

Selain itu, program ini juga belum diujikan pada server dengan *operating system* selain Linux Mandrake 9.2 dan juga belum dicoba dalam jaringan yang sesungguhnya.

BAB IV

KESIMPULAN DAN SARAN

4.1 Kesimpulan

- Metoda port knocking dapat berfungsi dengan baik asalkan komputer tersebut memiliki firewall, iptables atau ipchain, dan koneksi dengan jaringan.
- Metoda port knocking cukup baik diimplementasikan pada jaringan yang lalu lintasnya tidak terlalu padat.
- Dengan metoda port knocking orang lain yang tidak berhak tidak mampu mengetahui apakah ada port yang terbuka dan memberikan layanan.

4.2 Saran

- Program yang dibuat masih sangat sederhana, sehingga perlu dikembangkan lebih lanjut. Misal dengan menambahkan algoritma enkripsi sehingga mempersulit pendeteksian oleh orang yang tidak berhak.

- Ketukan ditambahkan dengan pemilihan port yang tidak dimonitor secara acak. (Pada program ini hanya ditambahkan secara permanen dan tidak berubah-ubah setiap pengetukan)
- Dibuat suatu program yang mampu membaca logfile dengan format yang berbeda-beda (Dalam program yang telah penulis buat hanya mampu membaca format iptables)
- Menguji dan memperbaiki kompatibilitas dengan berbagai *operating system*.

REFERENSI

- Krzywinski, M. 2003. Port Knocking: Network Authentication Across Closed Ports. Sysadmin Magazine 12: 12-17.
- Port Knocking.org. www.portknocking.org
- LeCount, David. Iptables Basic. www.start-linux.com/articles/article-85.php
- Netfilter Log Format. www.logi.cc/linux/netfilter-log-format.php

LAMPIRAN A

PROGRAM UNTUK MENDENGARKAN KETUKAN

```
/*-----File aprList.h-----*/

#ifndef APPRLIST_H
#define APPRLIST_H
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <time.h>

typedef struct tElmtApplist *addrApplist;
typedef struct tElmtApplist
{
    time_t Waktu;
    char *IP;
    char *MAC;
    int Port;
    int Stat;
    addrApplist next;
} elmtApplist;

typedef addrApplist Applist;

addrApplist createElmtApplist (const time_t Waktu, const char *IP, const char *MAC, const int
Port);

/* menghasilkan elmtApplist dengan nilai-nilai diisi dan next = NULL */
```

```
time_t getWaktuAppllist (const elmtAppllist elemen);
/* mengembalikan nilai waktu dari elemen Appllist tersebut */

char *getIPAppllist (const elmtAppllist elemen);
/* mengembalikan nilai IP dari elemen Appllist tersebut */

char *getMACAppllist (const elmtAppllist elemen);
/* mengembalikan nilai MAC sumber dari elemen Appllist tersebut */

int getPortAppllist (const elmtAppllist elemen);
/* mengembalikan nilai port dari elemen Appllist tersebut */

void setWaktuAppllist (elmtAppllist elemen, time_t waktuIn);

void setIPAppllist (elmtAppllist elemen, char *IP_In);

void setMACAppllist (elmtAppllist elemen, char *MAC_In);

void setPortAppllist (elmtAppllist elemen, int Port);

addrAppllist SearchAppllist (const Appllist start, const char *IP_In, const char *MAC_In, const int
Port_In);
/* mencari Appllist sesuai dengan kriteria parameter, bila ditemukan akan mengembalikan
alamat elemen Appllist yang dimaksud, bila tidak akan mengembalikan NULL */

void InsertLastAppllist (Appllist start, addrAppllist elmtIns);
/* menyisipkan elemen elmtIns sebagai elemen terakhir dalam Appllist start */
/* Prekondisi : tidak ada elemen yang sama dengan elmtIns dalam Appllist */

void DeleteAppllist (Appllist *start, Appllist elmtDel);

void InsertAppllist (Appllist start, elmtAppllist elmtIns);
/* menyisipkan elemen elmtIns ke dalam Appllist start dengan menjaga keterurutan waktu */
```

```
/* Prekondisi : tidak ada elemen yang sama dengan elmtIns dalam Apprlist */

void DestroyApprlist (Apprlist start);
/* melakukan dealokasi Apprlist secara rekursif */
void PrintApprlist (Apprlist start);
#endif

/*-----File : apprList.c-----*/
#include "apprList.h"

addrApprlist createElmtApprlist (const time_t Waktu, const char *IP, const char *MAC, const int Port)
/* menghasilkan elmtApprlist dengan nilai-nilai diisi dan next = NULL */
{
    addrApprlist newElmt = (addrApprlist) malloc (sizeof(elmtApprlist));

    (*newElmt).Waktu = Waktu;
    (*newElmt).Port = Port;
    (*newElmt).Stat = 1;

    (*newElmt).IP = (char *) malloc (strlen(IP) * sizeof (char));
    (*newElmt).MAC = (char *) malloc (strlen(MAC) * sizeof (char));

    strcpy((*newElmt).IP,IP);
    strcpy((*newElmt).MAC,MAC);

    (*newElmt).next = NULL;

    return newElmt;
}

time_t getWaktuApprlist (const elmtApprlist elemen)
/* mengembalikan nilai waktu dari elemen Apprlist tersebut */
{
```

```
        return elemen.Waktu;
    }

char *getIPApprlist (const elmtApprlist elemen)
/* mengembalikan nilai IP dari elemen Apprlist tersebut */
{
    return elemen.IP;
}

char *getMACApprlist (const elmtApprlist elemen)
/* mengembalikan nilai MAC sumber dari elemen Apprlist tersebut */
{
    return elemen.MAC;
}

int getPortApprlist (const elmtApprlist elemen)
/* mengembalikan nilai port dari elemen Apprlist tersebut */
{
    return elemen.Port;
}

void setWaktuApprlist (elmtApprlist elemen, time_t waktuIn)
{
    elemen.Waktu = waktuIn;
}

void setIPApprlist (elmtApprlist elemen, char *IP_In)
{
    elemen.IP = (char *) malloc (strlen(IP_In) * sizeof (char));

    strcpy(elemen.IP,IP_In);
}
```

```
void setMACApplist (elmtApplist elemen, char *MAC_In)
{
    elemen.MAC = (char *) malloc (strlen(MAC_In) * sizeof (char));

    strcpy(elemen.MAC,MAC_In);
}

void setPortApplist (elmtApplist elemen, int Port)
{
    elemen.Port= Port;
}

addrApplist SearchApplist (const Applist start, const char *IP_In, const char *MAC_In, const int
Port_In)
/* mencari Applist sesuai dengan kriteria parameter, bila ditemukan akan mengembalikan
alamat elemen Applist yang dimaksud, bila tidak akan mengembalikan NULL */
{
    if (start != NULL)
    {
        if ((strcmp((*start).IP,IP_In) == 0) && (strcmp((*start).MAC,MAC_In) == 0)
            && ((*start).Port == Port_In))
        {
            printf("Sama !!!\n");
            return start;
        }
        else
            return (SearchApplist((*start).next,IP_In,MAC_In,Port_In));
    }
    else
        return NULL;
}
```

```
void InsertLastApplist (Applist start, addrApplist elmtIns)
/* menyisipkan elemen elmtIns sebagai elemen terakhir dalam Applist start */
/* Prekondisi : tidak ada elemen yang sama dengan elmtIns dalam Applist */
{
    if ((*start).next == NULL)
    {
        (*start).next = elmtIns;
        (*elmtIns).next = NULL;
        //printf("Inserted to last\n");
    }
    else
        InsertLastApplist((*start).next,elmtIns);
}

void DeleteApplist (Applist *start, Applist elmtDel)
{
    if ((*start) == elmtDel)
    {
        (*start) = (*elmtDel).next;
        (*elmtDel).next = NULL;
        DestroyApplist(elmtDel);
    }
    else if ((*(*start)).next == elmtDel)
    {
        ((*(*start)).next) = (*elmtDel).next;
        (*elmtDel).next = NULL;
        DestroyApplist(elmtDel);
    }
    else
    {
        Applist pointer = ((*start)).next;

        while ((*pointer).next != elmtDel)
```

```
        pointer = (*pointer).next;

        (*pointer).next = (*elmtDel).next;
        (*elmtDel).next = NULL;
        DestroyApprlist(elmtDel);
    }

}

void InsertApprlist (Apprlist start, elmtApprlist elmtIns);
/* menyisipkan elemen elmtIns ke dalam Apprlist start dengan menjaga keterurutan waktu */
/* Prekondisi : tidak ada elemen yang sama dengan elmtIns dalam Apprlist */

void DestroyApprlist (Apprlist start)
/* melakukan dealokasi Apprlist secara rekursif */
{
    if (start != NULL)
    {
        DestroyApprlist((*start).next);
        free((*start).IP);
        free((*start).MAC);
        free(start);
    }
}

void PrintApprlist (Apprlist start)
{
    if (start != NULL)
    {
        printf("Waktu = %s",ctime(&((*start).Waktu)));
        printf("IP = %s\n",(*start).IP);
        printf("MAC = %s\n",(*start).MAC);
        printf("Stat = %d\n",(*start).Stat);
    }
}
```

```
        printf("Port= %d\n-----\n",(*start).Port);

        if ((*start).next != NULL)
            PrintApprlist((*start).next);
    }
}

/*----- File : baca.h----- */
#ifndef baca_h
#define baca_h
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include "list.h"

typedef struct
{
    time_t waktu;
    char *MAC_SRC;
    char *MAC_DES;
    char *Src_IP;
    char *Des_IP;
    int Src_Port;
    int Des_Port;
    int ID;
} Log_struct;

char *readln(FILE *filein);

Log_struct Extract(char *Extractin);
```

```
time_t Str2Time (char *str);
```

```
#endif
```

```
/*-----File baca.c-----*/
```

```
#include "baca.h"
```

```
char *readln(FILE *filein)
```

```
{
```

```
    char CC;
```

```
    int retval;
```

```
    char *strout;
```

```
    int counter = 0;
```

```
    strout = (char *) malloc (250 * sizeof (char));
```

```
    retval = fscanf(filein,"%c",&CC);
```

```
    while ((retval != EOF) && (CC != '\n'))
```

```
    {
```

```
        //printf("CC = %c\n",CC);
```

```
        *(strout+counter) = CC;
```

```
        counter++;
```

```
        retval = fscanf(filein,"%c",&CC);
```

```
    }
```

```
    *(strout+counter) = '\0';
```

```
    //printf("readln keluar\n");
```

```
    return strout;
```

```
}
```

```
int Str2Int(char *Strin)
```

```
{
```

```
    int hasil = 0;
```

```
    int count = 0;
```

```
    while (count<strlen(Strin))
```

```
    {
        hasil = hasil*10+(int)(* (Strin+count) - '0');
        count++;
    }
    return hasil;
}
```

Log_struct Extract(char *Extractin)

```
{
    char *strtemp = (char *) malloc (500 * sizeof (char));
    char *token;
    Log_struct temp;

    char *waktuTemp = (char *) malloc (15 * sizeof (char));
    temp.MAC_SRC = (char *) malloc (17 * sizeof (char));
    temp.MAC_DES = (char *) malloc (17 * sizeof (char));
    temp.Src_IP = (char *) malloc (14 * sizeof (char));
    temp.Des_IP = (char *) malloc (14 * sizeof (char));

    //printf("extractin = %s\n",Extractin);
    strcpy(strtemp,Extractin);
    //printf("strtemp = %s\n",strtemp);

    strncpy(waktuTemp,strtemp,15);
    //printf("temp.waktu= %s\n",temp.waktu);

    temp.waktu = Str2Time(waktuTemp);

    token = strtok(strtemp," =");
    while (strcmp(token,"MAC") != 0)
        token = strtok(NULL," =");
    token = strtok(NULL," =");
    strncpy(temp.MAC_DES,token,17);
}
```

```
strncpy(temp.MAC_SRC,token+18,17);

while (strcmp(token,"SRC") != 0)
    token = strtok(NULL,"=");
token = strtok(NULL,"=");
strcpy(temp.Src_IP,token);

while (strcmp(token,"DST") != 0)
    token = strtok(NULL,"=");
token = strtok(NULL,"=");
strcpy(temp.Des_IP,token);

while (strcmp(token,"ID") != 0)
    token = strtok(NULL,"=");
token = strtok(NULL,"=");
temp.ID = Str2Int(token);

while (strcmp(token,"SPT") != 0)
    token = strtok(NULL,"=");
token = strtok(NULL,"=");
temp.Src_Port = Str2Int(token);

while (strcmp(token,"DPT") != 0)
    token = strtok(NULL,"=");
token = strtok(NULL,"=");
temp.Des_Port = Str2Int(token);

return temp;
}

int locate(char *locatein) //Kalo ketemu keluar nilai 1
{
    char *lcttemp;
```

```
char *lcttok;
int found = 0;

lcttemp = (char *) malloc (500 * sizeof (char));
strcpy(lcttemp,locatein);
//printf("lcttemp = %s\n",lcttemp);
lcttok = strtok(lcttemp,"=");
while ((lcttok != NULL) && (found != 1))
{
    if (strcmp(lcttok,"MAC") == 0)
        found = 1;
    lcttok = strtok(NULL,"=");
    //printf("lcttok = %s\n",lcttok);
}

return found;
}

time_t Str2Time (char *str)
{
    time_t hasil,now;
    struct tm temp,*now2;
    char *strTemp = (char *) malloc ((strlen(str)+3) * sizeof(char));
    char *tahun = (char *) malloc (3 * sizeof(char));
    int retval;

    time(&now);
    now2 = localtime(&now);

    strftime(tahun,4,"%y \0",now2);
    //printf("tahun = %s\n",tahun);
    strcpy(strTemp,tahun);
}
```

```
strcpy((strTemp+3),str);
//printf("str = %s\n",strTemp);
retval = strptime(strTemp,"%y %b %e %T",&temp);
//printf("retval = %d\n",retval);

hasil = mktime(&temp);

if (difftime(now,hasil) < 0)
{
    // kurangi hasil 1 tahun
    now = hasil;
    now2 = localtime(&now);
    (*now2).tm_year--;
    hasil = mktime(now2);
}
//printf("hasil(%s) = %s\n",str,ctime(&hasil));
return hasil;
}

double selisihWaktu (char *waktu2, char *waktu1)
// Menghasilkan perbedaan waktu antara waktu2 - waktu1
{
    //printf("Waktu2 = %s\n",waktu2);
    //printf("Waktu1 = %s\n",waktu1);
    time_t time2 = Str2Time(waktu2);
    time_t time1 = Str2Time(waktu1);

    return (difftime(time2,time1));
}

/*----- File list.h-----*/
/* Mendefinisikan list yang digunakan untuk menyimpan IP dan MAC.
    Pointer mmenuju elemen list sesudahnya (next). */
```

```
#ifndef LIST_H
#define LIST_H
#include "baca.h"

typedef struct tElmtSublist *addrSublist;
typedef struct tElmtSublist
{
    time_t Waktu;
    int ID;
    int Src_Port;
    int Des_Port;
    addrSublist next;
} elmtSublist;

typedef addrSublist Sublist;

typedef struct tElmtList *addrList;
typedef struct tElmtList
{
    char *Src_IP;
    char *Des_IP;
    char *Src_MAC;
    char *Des_MAC;
    Sublist child;
    addrList next;
} elmtList;

typedef addrList List;

addrSublist createElmtSublist (const time_t Waktu, const int ID, const int Src_port, const int
Des_port);
```

```
/* menghasilkan elmtSublist dengan nilai-nilai diisi dan next = NULL */
```

```
time_t getWaktu (const elmtSublist elemen);
```

```
/* mengembalikan nilai waktu dari elemen sublist tersebut */
```

```
int getID (const elmtSublist elemen);
```

```
/* mengembalikan nilai ID dari elemen sublist tersebut */
```

```
int getSrc_Port (const elmtSublist elemen);
```

```
/* mengembalikan nilai port sumber dari elemen sublist tersebut */
```

```
int getDes_Port (const elmtSublist elemen);
```

```
/* mengembalikan nilai port tujuan dari elemen sublist tersebut */
```

```
void setID (elmtSublist elemen, int ID_In);
```

```
void setWaktu (elmtSublist elemen, time_t waktuIn);
```

```
void setSrc_Port (elmtSublist elemen, int Src_PortIn);
```

```
void setDes_Port (elmtSublist elemen, int Des_PortIn);
```

```
addrSublist SearchSublist (const Sublist start, const time_t Waktu_in, const int ID_In, const int Src_Port_in, const int Des_Port_in);
```

```
/* mencari sublist sesuai dengan kriteria parameter, bila ditemukan akan mengembalikan alamat elemen sublist yang dimaksud, bila tidak akan mengembalikan NULL */
```

```
void InsertLastSublist (Sublist start, addrSublist elmtIns);
```

```
/* menyisipkan elemen elmtIns sebagai elemen terakhir dalam Sublist start */
```

```
/* Prekondisi : tidak ada elemen yang sama dengan elmtIns dalam subList */
```

```
void InsertSublist (Sublist start, elmtSublist elmtIns);
```

```
/* menyisipkan elemen elmtIns ke dalam subList start dengan menjaga keterurutan waktu */
```

```
/* Prekondisi : tidak ada elemen yang sama dengan elmtIns dalam subList */

void DestroySublist (Sublist start);
/* melakukan dealokasi subList secara rekursif */

void PrintSublist (Sublist start, int level);

addrList createElmtList (const char *Src_IP, const char *Des_IP, const char *Src_MAC, const char
*Des_MAC);
/* menghasilkan elmtList dengan nilai-nilai diisi dan next = NULL */

char *getSrc_IP (elmtList elemen);
/* mengembalikan nilai Src_IP dari elemen */

char *getDes_IP (elmtList elemen);
/* mengembalikan nilai Des_IP dari elemen */

char *getSrc_MAC (elmtList elemen);
/* mengembalikan nilai Src_MAC dari elemen */

char *getDes_MAC (elmtList elemen);
/* mengembalikan nilai Des_MAC dari elemen */

Sublist getChild (elmtList elemen);
/* mengembalikan child dari elemen */

void setSrc_IP (elmtList elemen,char *Src_IP_in);

void setDes_IP (elmtList elemen,char *Des_IP_in);

void setSrc_MAC (elmtList elemen,char *Src_MAC_in);

void setDes_MAC (elmtList elemen,char *Des_MAC_in);
```

```
void setChild (elmtList elemen, Sublist child_in);

addrList SearchList (const List start, const char *Src_IP, const char *Des_IP, const char *Src_MAC,
const char *Des_MAC);
/* mencari List sesuai dengan kriteria parameter, bila ditemukan akan mengembalikan
alamat elemen List yang dimaksud, bila tidak akan mengembalikan NULL */

void InsertLastList (List start, addrList elmtIns);
/* menyisipkan elemen elmtIns sebagai elemen terakhir dalam List start */
/* Prekondisi : tidak ada elemen yang sama dengan elmtIns dalam List */

void Insert (List start, elmtList elmtIns);
/* menyisipkan elemen elmtIns ke dalam List start */
/* Prekondisi : tidak ada elemen yang sama dengan elmtIns dalam List */

void DestroyList (List start);
/* melakukan dealokasi List secara rekursif */

void PrintList (List start);

int SearchData (List start, char *Src_IP, char *Des_IP, char *Src_MAC, char *Des_MAC, time_t
Waktu, int ID, int Src_Port, int Des_Port);
// mengembalikan 0 jika tidak ada, dan nilai selain 0 jika ada

List InsertData (List start, char *Src_IP, char *Des_IP, char *Src_MAC, char *Des_MAC, time_t
Waktu, int ID, int Src_Port, int Des_Port);
// Prekondisi : Data tidak ada dalam list

Sublist SearchDesPort(Sublist start, int PortDes);

int panjangSublist (Sublist start);
/* Mengembalikan banyak elemen yang dimiliki oleh sublist start */
```

```
#endif

/*-----File : list.c-----*/
#include "list.h"

addrList createElmtList (const char *Src_IP, const char *Des_IP, const char *Src_MAC, const char
*Des_MAC)
/* menghasilkan elmtList dengan nilai-nilai diisi dan next = NULL */
{
    addrList newElmt = (addrList) malloc (sizeof(elmtList));

    (*newElmt).Src_IP = (char *) malloc (strlen(Src_IP) * sizeof(char));
    (*newElmt).Des_IP = (char *) malloc (strlen(Des_IP) * sizeof(char));
    (*newElmt).Src_MAC = (char *) malloc (strlen(Src_MAC) * sizeof(char));
    (*newElmt).Des_MAC = (char *) malloc (strlen(Des_MAC) * sizeof(char));

    strcpy((*newElmt).Src_IP,Src_IP);
    strcpy((*newElmt).Des_IP,Des_IP);
    strcpy((*newElmt).Src_MAC,Src_MAC);
    strcpy((*newElmt).Des_MAC,Des_MAC);
    (*newElmt).child = NULL;
    (*newElmt).next = NULL;

    return newElmt;
}

char *getSrc_IP (elmtList elemen)
/* mengembalikan nilai Src_IP dari elemen */
{
    char *hasil = (char *) malloc (strlen(elemen.Src_IP) * sizeof(char));
    strcpy(hasil,elemen.Src_IP);
    return hasil;
}
```

```
char *getDes_IP (elmtList elemen)
/* mengembalikan nilai Des_IP dari elemen */
{
    char *hasil = (char *) malloc (strlen(elemen.Des_IP) * sizeof(char));
    strcpy(hasil,elemen.Des_IP);
    return hasil;
}

char *getSrc_MAC (elmtList elemen)
/* mengembalikan nilai Src_MAC dari elemen */
{
    char *hasil = (char *) malloc (strlen(elemen.Src_MAC) * sizeof(char));
    strcpy(hasil,elemen.Src_MAC);
    return hasil;
}

char *getDes_MAC (elmtList elemen)
/* mengembalikan nilai Des_MAC dari elemen */
{
    char *hasil = (char *) malloc (strlen(elemen.Des_MAC) * sizeof(char));
    strcpy(hasil,elemen.Des_MAC);
    return hasil;
}

Sublist getChild (elmtList elemen)
/* mengembalikan child dari elemen */
{
    return elemen.child;
}

void setSrc_IP (elmtList elemen,char *Src_IP_in)
{
```

```
        elemen.Src_IP = (char *) malloc (strlen(Src_IP_in) * sizeof(char));
        strcpy(elemen.Src_IP,Src_IP_in);
    }

void setDes_IP (elmtList elemen,char *Des_IP_in)
{
    elemen.Des_IP = (char *) malloc (strlen(Des_IP_in) * sizeof(char));
    strcpy(elemen.Des_IP,Des_IP_in);
}

void setSrc_MAC (elmtList elemen,char *Src_MAC_in)
{
    elemen.Src_MAC = (char *) malloc (strlen(Src_MAC_in) * sizeof(char));
    strcpy(elemen.Src_MAC,Src_MAC_in);
}

void setDes_MAC (elmtList elemen,char *Des_MAC_in)
{
    elemen.Des_MAC = (char *) malloc (strlen(Des_MAC_in) * sizeof(char));
    strcpy(elemen.Des_MAC,Des_MAC_in);
}

void setChild (elmtList elemen, Sublist child_in)
{
    elemen.child = child_in;
}

addrList SearchList (const List start, const char *Src_IP, const char *Des_IP, const char *Src_MAC,
const char *Des_MAC)
/* mencari List sesuai dengan kriteria parameter, bila ditemukan akan mengembalikan
alamat elemen List yang dimaksud, bila tidak akan mengembalikan NULL */
{
    if (start == NULL)
```

```
        return start;
    else
    {
        if ((strcmp((*start).Src_IP,Src_IP) == 0) && (strcmp((*start).Des_IP,Des_IP) == 0)
            && (strcmp((*start).Src_MAC,Src_MAC) == 0) && (strcmp((*start).
Des_MAC,Des_MAC) == 0))
        {
            return start;
        }
        else
            return (SearchList((*start).next,Src_IP,Des_IP,Src_MAC,Des_MAC));
    }
}
```

```
void InsertLastList (List start, addrList elmtIns)
```

```
/* menyisipkan elemen elmtIns sebagai elemen terakhir dalam List start */
```

```
/* Prekondisi : tidak ada elemen yang sama dengan elmtIns dalam List dan start tidak null*/
```

```
{
    if ((*start).next == NULL)
    {
        (*start).next = elmtIns;
        (*elmtIns).next = NULL;
    }
    else
        InsertLastList((*start).next,elmtIns);
}
```

```
void Insert (List start, elmtList elmtIns)
```

```
/* menyisipkan elemen elmtIns ke dalam List start */
```

```
/* Prekondisi : tidak ada elemen yang sama dengan elmtIns dalam List */
```

```
{
    //Tidak usah
}
```

```
void DestroyList (List start)
/* melakukan dealokasi List secara rekursif */
{
    if (start != NULL)
    {
        DestroyList((*start).next);
        DestroySublist((*start).child);
        free((*start).Src_IP);
        free((*start).Des_IP);
        free((*start).Src_MAC);
        free((*start).Des_MAC);
        free(start);
    }
}

void PrintList (List start)
{
    if (start != NULL)
    {
        printf("Src_IP = %s\n",(*start).Src_IP);
        printf("Des_IP = %s\n",(*start).Des_IP);
        printf("Src_MAC = %s\n",(*start).Src_MAC);
        printf("Des_MAC = %s\n",(*start).Des_MAC);

        PrintSublist((*start).child,5);

        if ((*start).next != NULL)
            PrintList((*start).next);
    }
}
```

```
int SearchData (List start, char *Src_IP, char *Des_IP, char *Src_MAC, char *Des_MAC, time_t
Waktu, int ID, int Src_Port, int Des_Port)
// mengembalikan 0 jika tidak ada, dan nilai selain 0 jika ada
// Prekondisi : start tidak NULL
{
    addrList pointList;

    pointList = SearchList(start,Src_IP,Des_IP,Src_MAC,Des_MAC);

    if (pointList == NULL)
        return 0;
    else
    {
        // Search child
        if (SearchSublist((*pointList).child,Waktu,ID,Src_Port,Des_Port) == NULL)
        {
            return 0;
        }
        else
            return 1;
    }
}
```

```
List InsertData (List start, char *Src_IP, char *Des_IP, char *Src_MAC, char *Des_MAC, time_t
Waktu, int ID, int Src_Port, int Des_Port)
// Prekondisi : Data tidak ada dalam list
{
    addrList pointList;

    //printf("Before SearchList\n");
    pointList = SearchList(start,Src_IP,Des_IP,Src_MAC,Des_MAC);
    //printf("After SearchList\n");
    //printf("Pointlist = %s\n",pointList);
    if (pointList == NULL)
```

```
{
    pointList = createElmtList(Src_IP,Des_IP,Src_MAC,Des_MAC);
    //printf("After Create\n");
    if (start != NULL)
        InsertLastList(start,pointList);
    //printf("After InsertLast\n");
}

Sublist childtmp = (*pointList).child;
if (SearchSublist(childtmp,Waktu,ID,Src_Port,Des_Port) == NULL)
{
    addrSublist elmtIns = createElmtSublist(Waktu,ID,Src_Port,Des_Port);
    //printf("Before SearchSubList\n");
    if (childtmp != NULL)
        InsertLastSublist(childtmp,elmtIns);
    else
        (*pointList).child = elmtIns;
    //printf("After SearchSubList\n");
}

if (start == NULL)
    return pointList;
else
    return start;
}

int panjangSublist (Sublist start)
/* Mengembalikan banyak elemen yang dimiliki oleh sublist start */
{
    if (start != NULL)
        return (panjangSublist((*start).next) + 1);
    else
        return 0;
}
```

```
}

/*-----File : sublist.h -----*/
/* Mendefinisikan sublist yang digunakan untuk menyimpan waktu dan port.
   Pointer mmenuju elemen list sesudahnya (next).
   Insert dilakukan dengan tetap menjaga keterurutan waktu dan keunikan elemen sublist */

#ifndef SUBLIST_H
#define SUBLIST_H
#include "baca.h"

typedef struct tElmtSublist *addrSublist;
typedef struct tElmtSublist
{
    time_t Waktu;
    int Src_Port;
    int Des_Port;
    addrSublist next;
} elmtSublist;

typedef addrSublist Sublist;

addrSublist createElmtSublist (const time_t Waktu, const int Src_port, const int Des_port);
/* menghasilkan elmtSublist dengan nilai-nilai diisi dan next = NULL */

time_t getWaktu (const elmtSublist elemen);
/* mengembalikan nilai waktu dari elemen sublist tersebut */

int getSrc_Port (const elmtSublist elemen);
/* mengembalikan nilai port sumber dari elemen sublist tersebut */

int getDes_Port (const elmtSublist elemen);
/* mengembalikan nilai port tujuan dari elemen sublist tersebut */
```

```
void setWaktu (elmtSublist elemen, time_t waktuIn);

void setSrc_Port (elmtSublist elemen, int Src_PortIn);

void setDes_Port (elmtSublist elemen, int Des_PortIn);

addrSublist SearchSublist (const Sublist start, const time_t Waktu_in, const int Src_Port_in, const int
Des_Port_in);
/* mencari sublist sesuai dengan kriteria parameter, bila ditemukan akan mengembalikan
alamat elemen sublist yang dimaksud, bila tidak akan mengembalikan NULL */

void InsertLastSublist (Sublist start, elmtSublist elmtIns);
/* menyisipkan elemen elmtIns sebagai elemen terakhir dalam Sublist start */
/* Prekondisi : tidak ada elemen yang sama dengan elmtIns dalam subList */

void InsertSublist (Sublist start, elmtSublist elmtIns);
/* menyisipkan elemen elmtIns ke dalam subList start dengan menjaga keterurutan waktu */
/* Prekondisi : tidak ada elemen yang sama dengan elmtIns dalam subList */

void DestroySublist (Sublist start);
/* melakukan dealokasi subList secara rekursif */

void PrintSublist (Sublist start, int level);

#endif

/*----- File : sublist.c -----*/
#include "list.h"

addrSublist createElmtSublist (const time_t Waktu, const int ID, const int Src_port, const int Des_port)
/* menghasilkan elmtSublist dengan nilai-nilai diisi dan next = NULL */
{
```

```
    addrSublist newElmt = (addrSublist) malloc (sizeof(elmSublist));

    (*newElmt).Waktu = Waktu;
    (*newElmt).ID = ID;
    (*newElmt).Src_Port = Src_port;
    (*newElmt).Des_Port = Des_port;
    (*newElmt).next = NULL;

    return newElmt;
}

time_t getWaktu (const elmSublist elemen)
/* mengembalikan nilai waktu dari elemen sublist tersebut */
{
    return elemen.Waktu;
}

int getID (const elmSublist elemen)
/* mengembalikan nilai ID dari elemen sublist tersebut */
{
    return elemen.ID;
}

int getSrc_Port (const elmSublist elemen)
/* mengembalikan nilai port sumber dari elemen sublist tersebut */
{
    return elemen.Src_Port;
}

int getDes_Port (const elmSublist elemen)
/* mengembalikan nilai port tujuan dari elemen sublist tersebut */
{
    return elemen.Des_Port;
}
```

```
}
```

```
void setWaktu (elmtSublist elemen, time_t waktuIn)
```

```
{
```

```
    elemen.Waktu = waktuIn;
```

```
}
```

```
void setID (elmtSublist elemen, int ID_In)
```

```
{
```

```
    elemen.ID = ID_In;
```

```
}
```

```
void setSrc_Port (elmtSublist elemen, int Src_PortIn)
```

```
{
```

```
    elemen.Src_Port = Src_PortIn;
```

```
}
```

```
void setDes_Port (elmtSublist elemen, int Des_PortIn)
```

```
{
```

```
    elemen.Des_Port = Des_PortIn;
```

```
}
```

```
addrSublist SearchSublist (const Sublist start, const time_t Waktu_in, const int ID_In, const int  
Src_Port_in, const int Des_Port_in)
```

```
/* mencari sublist sesuai dengan kriteria parameter, bila ditemukan akan mengembalikan
```

```
alamat elemen sublist yang dimaksud, bila tidak akan mengembalikan NULL */
```

```
{
```

```
    Sublist pointer = start;
```

```
    int selisihID;
```

```
    double selisihWaktu;
```

```
    if (pointer == NULL)
```

```
    {
```

```
        return pointer;
    }
    else
    {
        selisihWaktu = difftime(Waktu_in,(*pointer).Waktu);
        selisihID = ID_In - (*pointer).ID;
        if ((selisihWaktu < 60.1) && (selisihWaktu > -60.1) && (selisihID <= 30) &&
        (selisihID >= -30) &&
        Des_Port_in))
            ((*pointer).Src_Port == Src_Port_in) && ((*pointer).Des_Port ==
        Des_Port_in))
            {
                return pointer;
            }
            else
            {
                return SearchSublist((*pointer).
        next,Waktu_in,ID_In,Src_Port_in,Des_Port_in);
            }
        }
    }
}
```

```
void InsertLastSublist (Sublist start, addrSublist elmtIns)
```

```
/* menyisipkan elemen elmtIns sebagai elemen terakhir dalam Sublist start */
```

```
/* Prekondisi : tidak ada elemen yang sama dengan elmtIns dalam subList dan start tidak null */
```

```
{
    if ((*start).next == NULL)
    {
        // insert
        (*start).next = elmtIns;
    }
    else
        InsertLastSublist((*start).next,elmtIns);
}
```

```
void InsertSublist (Sublist start, elmtSublist elmtIns)
/* menyisipkan elemen elmtIns ke dalam subList start dengan menjaga keterurutan waktu */
/* Prekondisi : tidak ada elemen yang sama dengan elmtIns dalam subList */
{
    // Under Construction
}
```

```
void DestroySublist (Sublist start)
/* melakukan dealokasi subList secara rekursif */
{
    if (start != NULL)
    {
        DestroySublist((*start).next);
        free(start);
    }
}
```

```
void PrintSublist (Sublist start, int level)
{
    int count = 0;
    time_t waktutemp = (*start).Waktu;

    if (start != NULL)
    {
        for (count = 0; count < level; count++)
            printf(" ");
        printf("Waktu : %s",ctime(&waktutemp));

        for (count = 0; count < level; count++)
            printf(" ");
        printf("ID : %d",(*start).ID);
    }
}
```

```
        for (count = 0; count < level; count++)
            printf(" ");
        printf("Src_Port : %d",(*start).Src_Port);
        for (count = 0; count < level; count++)
            printf(" ");
        printf("Des_Port : %d\n",(*start).Des_Port);
        for (count = 0; count < level; count++)
            printf(" ");
        printf("-----\n",(*start).Des_Port);
        //printf("next = %d\n",(*start).next);
        if ((*start).next != NULL)
            PrintSublist ((*start).next,level);
    }
}
```

Sublist SearchDesPort(Sublist start, int PortDes)

```
{
    if (start != NULL)
    {
        if ((*start).Des_Port == PortDes)
        {
            return start;
        }
        else
        {
            return SearchDesPort((*start).next,PortDes);
        }
    }
    else
        return NULL;
}
```

/*-----File daemon.c-----*/

```
#include "baca.h"
#include "apprList.h"
#include <unistd.h>
#include <sys/types.h>

const int Header_Seq[3] = { 610,400,503};
const int Footer_Seq[3] = { 606,505,402};
const int SSH_Seq[5] = { 402,504,505,402,503};
const int ftp_Seq[5] = { 402,607,400,401,400};
const int maxTime = 300; //waktu ketukan harus di bawah 5 menit

char *const init1[10] = {" \0", "-A\0", "INPUT\0", "-p\0", "tcp\0", "--dport\0", "400:402\0", "-j\0",
"LOG\0", NULL};
char *const init2[10] = {" \0", "-A\0", "INPUT\0", "-p\0", "udp\0", "--dport\0", "400:402\0", "-j\0",
"LOG\0", NULL};
char *const init3[10] = {" \0", "-A\0", "INPUT\0", "-p\0", "tcp\0", "--dport\0", "503:505\0", "-j\0",
"LOG\0", NULL};
char *const init4[10] = {" \0", "-A\0", "INPUT\0", "-p\0", "udp\0", "--dport\0", "503:505\0", "-j\0",
"LOG\0", NULL};
char *const init5[10] = {" \0", "-A\0", "INPUT\0", "-p\0", "tcp\0", "--dport\0", "606:610\0", "-j\0",
"LOG\0", NULL};
char *const init6[10] = {" \0", "-A\0", "INPUT\0", "-p\0", "udp\0", "--dport\0", "606:610\0", "-j\0",
"LOG\0", NULL};
char *const drop[6] = {" \0", "-A\0", "INPUT\0", "-j\0", "DROP\0", NULL};
char *const flush[3] = {" \0", "-F\0", NULL};

Apprlist LogApproved = NULL;

int filterLog(Log_struct filterIn)
/*mengembalikan nilai 1 apabila lolos filter dan 0 apabila tidak*/
{
    time_t now;
    time_t filterTemp = filterIn.waktu;
```

```
time(&now);

if (difftime(now,filterTemp)<3600)
    return 1;
else
    return 0;
}

void execCmd(char *cmd, char *const argv[])
{
    //char *arg[1] = {"-A INPUT -p tcp --dport 400:403 -j LOG"};
    /*char *arg[10] = {" \0", "-A\0", "INPUT\0", "-p\0", "tcp\0", "--dport\0", "400:403\0", "-j\0",
"LOG\0", NULL};
    char *arg2[10] = {" \0", "-A\0", "INPUT\0", "-p\0", "udp\0", "--dport\0", "400:403\0", "-j\0",
"LOG\0", NULL};*/
    int pid;

    pid = fork();

    if (pid == 0)
    {
        execvp (cmd,argv);
    }
    else
    {
        wait();
    }
}
```

List readLog()

```
{
    FILE *filein;
    char *line = (char *) malloc (sizeof (char));
```

```
char *token;
Log_struct test;
List newList = NULL;

filein = fopen("/var/log/syslog","r");

while (feof(filein) == 0)
{
    do
    {
        free (line);
        line = readln(filein);

    }
    while ((locate(line)== 0) && (feof(filein) == 0));

    if (strlen(line) > 0)
    {
        test = Extract(line);
        if ((filterLog(test) != 0) &&
            (SearchData
(newList,test.Src_IP,test.Des_IP,test.MAC_SRC,test.MAC_DES,test.waktu,
            test.ID,test.Src_Port,test.Des_Port) == 0))
        {
            newList = InsertData
(newList,test.Src_IP,test.Des_IP,test.MAC_SRC,test.MAC_DES,test.waktu,
            test.ID,test.Src_Port,test.Des_Port);
        }
    }
}
fclose(filein);
//PrintList(newList);
return newList;
```

```
}

int compSublArr(Sublist *SubCheck, const int ArrCheck[], int SizeArr, time_t *wktAwal, time_t
*wktAkhir)
/*Mengembalikan nilai 0 bila ketukan tidak sesuai dan 1 bila sesuai*/
{
    int count = 0;
    int hasil = 1;
    Sublist pointer = (*SubCheck);

    if (panjangSublist(pointer) < SizeArr)
    {
        // sublist tidak cukup panjang
        //printf("Sublist tidak cukup panjang\n");
        (*SubCheck) = NULL;
        return 0;
    }
    else
    {
        while ((hasil == 1) && (count < SizeArr))
        {

            if (pointer == NULL)
                hasil = 0;
            else
            {
                if (count == 1)
                    *wktAwal = (*pointer).Waktu;
                if (count == (SizeArr-1))
                    *wktAkhir = (*pointer).Waktu;

                //printf("%d == %d\n", (*pointer).Des_Port, ArrCheck[count]);
                if ((*pointer).Des_Port == ArrCheck[count])
```

```
        {
            count++;
            pointer = (*pointer).next;
        }
        else
            hasil = 0;
    }

}

if (hasil == 1)
    (*SubCheck) = pointer;
//printf("hasil CompSubArr = %d\n\n",hasil);
return hasil;
}
}

int checkSeq(Sublist *seqList, time_t *waktu_Akhir)
/*mengembalikan nilai -1 apabila sequence tidak sesuai dan nilai port bila sesuai untuk IP dan MAC
tertentu*/
{
    time_t waktuAwal,waktuAkhir, awal, akhir;
    Sublist FirstSeq = (*seqList);
    Sublist tempFirstSeq;

    while (FirstSeq != NULL)
    {
        //printf("Before SearchDesPort\n");
        FirstSeq = SearchDesPort(FirstSeq,610);
        //printf("After SearchDesPort\n");

        if (FirstSeq != NULL)
        {
            if (compSublArr(&FirstSeq, Header_Seq, 3, &waktuAwal, &waktuAkhir)
== 1)
```

```

    {
        //printf("Header cocok\n");
        awal = waktuAwal;
        tempFirstSeq = FirstSeq;
        if (compSublArr(&FirstSeq, SSH_Seq, 5, &waktuAwal,
&waktuAkhir) == 1)
            {
                //printf("SSH cocok\n");
                if (compSublArr(&FirstSeq, Footer_Seq, 3, &waktuAwal,
&waktuAkhir) == 1)
                    {
                        //printf("Footer cocok\n");
                        akhir = waktuAkhir;
                        if(difftime(akhir,awal) <= maxTime)
                            {
                                (*seqList) = FirstSeq;
                                (*waktu_Akhir) = akhir;
                                return 22;
                            }
                    }
            }
        else if (compSublArr(&tempFirstSeq, ftp_Seq, 5, &waktuAwal,
&waktuAkhir) == 1)
            {
                //printf("ftp cocok\n");
                if (compSublArr(&tempFirstSeq, Footer_Seq, 3,
&waktuAwal, &waktuAkhir) == 1)
                    {
                        //printf("Footer cocok\n");
                        akhir = waktuAkhir;
                        if(difftime(akhir,awal) <= maxTime)
                            {
                                (*seqList) = tempFirstSeq;
                                (*waktu_Akhir) = akhir;
                            }
                    }
            }
    }

```

```

return 21;
    }
    }
    FirstSeq = tempFirstSeq;
}
}
if (FirstSeq != NULL)
    FirstSeq = (*FirstSeq).next;
}
}
(*seqList) = FirstSeq;
return -1;
}

void PortAction(int Action,char *IP_Act,char *MAC_Act,int Port_Act)
{
    if(Action == 1)
    {
        if (Port_Act == 22)
        {
            char *openPr22[12]={" \0","-I\0", "INPUT\0","-s\0",IP_Act,
                "-p\0", "tcp\0","--dport\0", "22\0", "-j\0", "ACCEPT\0",NULL};
            execCmd("iptables",openPr22);
        }
        else if (Port_Act == 21)
        {
            char *openPr21[12]={" \0","-I\0", "INPUT\0","-s\0",IP_Act,
                "-p\0", "tcp\0","--dport\0", "21\0", "-j\0", "ACCEPT\0",NULL};
            execCmd("iptables",openPr21);
        }
    }
    else
    {

```

```
    if (Port_Act == 22)
    {
        char *openPr22[12]={ " \0", "-D\0", "INPUT\0", "-s\0", IP_Act,
            "-p\0", "tcp\0", "--dport\0", "22\0", "-j\0", "ACCEPT\0",
            NULL};
        execCmd("iptables",openPr22);
    }
    else if (Port_Act == 21)
    {
        char *openPr21[12]={ " \0", "-D\0", "INPUT\0", "-s\0", IP_Act,
            "-p\0", "tcp\0", "--dport\0", "21\0", "-j\0", "ACCEPT\0",
            NULL};
        execCmd("iptables",openPr21);
    }
}
}
```

```
void traceSurf(List logList)
{
    List point = logList;
    Sublist pointChild;
    int hasil;
    time_t sekarang, waktuAkhir;

    while (point != NULL)
    {
        pointChild = (*point).child;
        while (pointChild != NULL)
        {
            hasil = checkSeq(&pointChild,&waktuAkhir);
            //printf("hasil = %d\n",hasil);
            if (hasil > 0)
            {
```

```
Apprlist srcRes;
if ((srcRes = SearchApprlist(LogApproved,(*point).Src_IP,
(*point).Src_MAC,
    hasil)) == NULL)
{
    printf("Buka port %d untuk IP %s \n", hasil,(*point).
Src_IP);
    PortAction(1,(*point).Src_IP,(*point).Src_MAC,hasil);
    if (LogApproved != NULL)
        InsertLastApprlist
(LogApproved,createElmtApprlist(waktuAkhir,
    (*point).Src_IP,(*point).
Src_MAC,hasil));
    else
        LogApproved = createElmtApprlist(waktuAkhir,
(*point).Src_IP,
    (*point).Src_MAC,hasil);
}
else
{
    //printf("waktuAkhir = %s",ctime(&waktuAkhir));
    //printf("waktuApprove = %s",ctime(&((*srcRes).
Waktu));
    //printf("Selisih = %f\n",difftime(waktuAkhir,(*srcRes).
Waktu));
    if (difftime(waktuAkhir,(*srcRes).Waktu)>0.0)
    {
        if ((*srcRes).Stat == 1)
        {
            // Port sudah dibuka, sekarang ditutup
            printf("Tutup port %d untuk IP %s \n",
hasil,(*point).Src_IP);
            (*srcRes).Stat = 0;
            (*srcRes).Waktu = waktuAkhir;
            PortAction(0,(*point).Src_IP,(*point).
Src_MAC,hasil);
        }
    }
}
```

```

    }
    else
    {
        // Port sudah ditutup, sekarang dibuka
        kembali
        printf("Buka kembali port %d untuk IP
        %s \n",
            hasil,(*point).Src_IP);
        (*srcRes).Stat = 1;
        (*srcRes).Waktu = waktuAkhir;
        PortAction(1,(*point).Src_IP,(*point).
        Src_MAC,hasil);
    }
}
}
}
}
}
point = (*point).next;
}
}

int main()
{
    time_t now;

    List logList = NULL;
    execCmd("iptables",flush);
    execCmd("iptables",init1);
    execCmd("iptables",init2);
    execCmd("iptables",init3);
    execCmd("iptables",init4);
    execCmd("iptables",init5);
    execCmd("iptables",init5);
    execCmd("iptables",drop);
}
```

```
//printf("After execCmd\n");
for (;;)
{
    time(&now);
    logList = readLog();
    PrintList(logList);
    //printf("Before trace\n");
    traceSurf(logList);
    printf("LogApproved\n-----\n");
    PrintApprlist(LogApproved);
    //printf("Before destroy\n");
    DestroyList (logList);
    //printf("After destroy\n");
    logList = NULL;
    printf("%s-----\n1 cycle of ReadLog\n",ctime(&now));
    sleep(1);
}
return 0;
}
```

LAMPIRAN B

PROGRAM UNTUK MENGETUK

```
/*----- File : client.c -----*/
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <signal.h>
#include <fcntl.h>

void ketuk (char* namahost, int port)
{
    int saveflags;
    //int pid = fork();
    //if (pid == 0)
    {
        struct sockaddr_in server;
        int sockfd;
        struct hostent *h;
        char *message="Hello Server";
        //const char *hostname = "localhost";

        if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
        {
            printf("Socket Error...");
            //exit(1);
        }

        if((h=gethostbyname(namahost))==NULL)
        {
```

```
        fprintf(stderr,"Host Name Error...");
        //exit(1);
    }

    server.sin_addr=((struct in_addr*)h->h_addr);
    server.sin_port=htons(port);
    server.sin_family=AF_INET;

    saveflags = fcntl(sockfd,F_GETFL,0);
    if (saveflags < 0)
    {
        perror("Error acquiring flags");
        exit(0);
    }

    /* set non-blocking */
    if (fcntl(sockfd,F_SETFL,saveflags|O_NONBLOCK) < 0)
    {
        perror("Error setting flags");
        exit(0);
    }

    if(connect(sockfd,(struct sockaddr*)&server,sizeof(struct sockaddr))== -1)
    {
        fprintf(stderr,"%s::%dConnection out...\n",namahost,port);
        //exit(1);
    }
    //send(sockfd,message,strlen(message),0);
    //exit(0);
}
/*else
/{
    sleep(0.005);
```

```
        time_t now;
        time(&now);
        printf("%s : kill pid %d\n",ctime(&now),pid);
        //kill(pid,SIGINT);
        wait ();
    }*/
}

int main()
{
    //Header
    ketuk ("localhost",610);
    sleep (1);
    ketuk ("localhost",701);
    sleep (1);
    ketuk ("localhost",400);
    sleep (1);
    ketuk ("localhost",305);
    sleep (1);
    ketuk ("localhost",503);
    sleep (1);
    ketuk ("localhost",203);
    sleep (1);

    //Body SSH
    /*ketuk ("localhost",402);
    sleep (1);
    ketuk ("localhost",605);
    sleep (1);
    ketuk ("localhost",504);
    sleep (1);
    ketuk ("localhost",403);
    sleep (1);
```

```
ketuk ("localhost",505);
sleep (1);
ketuk ("localhost",702);
sleep (1);
ketuk ("localhost",402);
sleep (1);
ketuk ("localhost",205);
sleep (1);
ketuk ("localhost",503);
sleep (1);
ketuk ("localhost",501);
sleep (1);*/

//Body FTP
ketuk ("localhost",402);
sleep (1);
ketuk ("localhost",604);
sleep (1);
ketuk ("localhost",607);
sleep (1);
ketuk ("localhost",708);
sleep (1);
ketuk ("localhost",400);
sleep (1);
ketuk ("localhost",230);
sleep (1);
ketuk ("localhost",401);
sleep (1);
ketuk ("localhost",350);
sleep (1);
ketuk ("localhost",400);
sleep (1);
ketuk ("localhost",509);
```

```
    sleep (1);

    //Footer
    ketuk ("localhost",606);
    sleep (1);
    ketuk ("localhost",602);
    sleep (1);
    ketuk ("localhost",505);
    sleep (1);
    ketuk ("localhost",408);
    sleep (1);
    ketuk ("localhost",402);
    sleep (1);
    ketuk ("localhost",604);

    return 0;
}
```