Register : Jurnal Ilmiah Teknologi Sistem Informasi 4 (2) 2018 103-113
ISSN 2502-3357 (*Online*)
ISSN 2503-0477 (*Print*)

Tersedia *online* di www.journal.unipdu.ac.id
Unipdu

Halaman jurnal di www.journal.unipdu.ac.id/index.php/register

Secure random port list generator pada mekanisme autentikasi dengan menggunakan *Port Knocking* dan *Secure Socket Layer*

Abdul Rauf [a], *Mahar Faiqurahman* [b], *Denar Regata Akbi* [c]

[a,b,c] *Teknik Informatika, Universitas Muhammadiyah Malang, Malang, Indonesia*

email: [a] abdulraufmustakim@gmail.com, [b] maharf@gmail.com, [c] dregata.dosen@gmail.com

INFO ARTIKEL

Sejarah artikel:
Menerima 19 Mei 2018
Revisi 14 Agustus 2018
Diterima 15 Agustus 2018
Online 18 Agustus 2018

Kata kunci:
autentikasi
Port Knocking
SSH
SSL

Keywords:
authentication
Port Knocking
SSH
SSL

Style APA dalam mensitasi artikel ini:
Rauf, A., Faiqurahman, M., & Akbi, D. R. (2018). Secure random port list generator pada mekanisme autentikasi dengan menggunakan Port Knocking dan Secure Socket Layer. *Register: Jurnal Ilmiah Teknologi Sistem Informasi*, 4(2), 103-113.

ABSTRAK

Port Knocking merupakan proses autentikasi yang dilakukan dengan mengetuk *port* tertentu untuk membuka dan menutup koneksi menuju suatu *service*. Pada umumnya, *Port Knocking* memiliki prosedur penetapan daftar port yang bersifat tetap. Hal inilah yang mendorong munculnya penelitian terkait penerapan *Secure Random Port List Generator* (SRPLG), melalui perancangan sebuah sistem yang mampu mengacak daftar *port* yang digunakan untuk *knocking*. Di samping itu, sistem ini juga didesain agar mampu mengirimkan informasi daftar *port* teracak tersebut kepada *client* melalui jalur aman. SRPLG *server* ini akan di integrasikan pada mekanisme autentikasi *Port Knocking*. Penerapan metode ini bertujuan untuk menciptakan sebuah prosedur autentikasi yang dinamis, aman dan efisien dalam mengamankan *Secure Shell server* (SSH *server*). Hasil pengujian yang didapatkan menunjukkan bahwa SRPLG *server* dalam mengacak daftar *Port Knocking* mampu menghasilkan daftar port yang selalu berubah setiap kali ada *request* dari *client*. Kemudian dari hasil *sniffing* yang dilakukan terhadap data yang ditransmisikan oleh SRPLG *server* dan *client* menunjukkan bahwa seluruh informasi yang ditangkap telah dienkripsi oleh *Secure Socket Layer* (SSL). Hasil pengujian peforma SRPLG *server* terhadap jumlah *client* yang melakukan *request*, rata-rata membutuhkan waktu antara 0,01 detik sampai 0,06 detik dalam setiap variasi pengujian peformansi. Pengujian terakhir menunjukkan bahwa SSH *server* telah berhasil diamankan dengan konfigurasi *Port Knocking* dari serangan *port scanning attack*, di mana seluruh informasi *port* yang ditampilkan, tidak ditemukan satupun celah yang dapat eksploitasi.

ABSTRACT

Port Knocking is an authentication process done by tapping a particular port to open and close the connection to a service. In general, the knocking port has a fixed procedure to assign ports list. This is what prompted the emergence of research related to the implementation of Secure Random Port List Generator (SRPLG), through the design of a system capable of scrambling the list of ports used for knocking. In addition, the system is also designed to be able to transmit random ports list information to clients via a secure path. This SRPLG server will be integrated into the Port Knocking authentication mechanism. Implementation of this method aims to create a dynamic authentication procedure, secure and efficient in securing the SSH server. The test results show that the secure random port list generator server in scrambling the list of knocking ports is able to generate an ever-changing port list every time a client requests. Then from the sniffing done to the data transmitted by the SRPLG server and the client indicates that all captured information has been encrypted by secure socket layer or SSL. Performance test marks SRPLG server to the number of clients who make requests, average takes between 0.01 seconds to 0.06 seconds in every variation of performance testing. In the last test shows SSH server has been successfully secured with Port Knocking configuration from attack port scanning attack, where all port information is displayed, not found any fault that can exploit.

A. Rauf dkk. /Register 4 (2) 2018 103-113
ISSN 2502-3357 (*Online*)
ISSN 2503-0477 (*Print*)

1. Pendahuluan

Autentikasi standar layanan *Secure Shell Server* (SSH *Server*) tidak aman bila tidak diikuti dengan *password* yang unik. Hal tersebutlah yang mendorong banyak sekali metode-metode keamanan dibuat pada suatu jaringan untuk meningkatkan keamanan, termasuk salah satunya dengan penggunaan *Port Knocking* (Fajri, Suhatman, & Putra, 2014). Dalam jaringan komputer, *Port Knocking* adalah metode membuka *port* secara eksternal pada *firewall* dengan membuat percobaan koneksi pada satu set *port* tertutup yang telah ditentukan sebelumnya (Mehran, Reza, & Laleh, 2012). *Port Knocking* merupakan suatu teknik yang pertama kali diperkenalkan untuk mencegah *attacker* dari serangan eksploitasi terhadap kerentanan *service host* jaringan, dengan hanya mengizinkan *client* yang terautentikasi untuk dapat mengakses sebuah *service* (Ali, Yunos, & Alias, 2012).

Pada penelitian tentang Implementasi *Remote Server* menggunakan metode *Port Knocking* dengan *Asymmetric Encryption* (Rozi, Ijtihadie, & Anggoro, 2010). Penelitian ini mencoba mengkombinasikan antara metode *Port Knocking* dengan menambahkan metode *asymmetric encryption* pada proses koneksi yang dilakukan oleh *client*. Tujuannya adalah membatasi koneksi menuju *server* dengan hanya memberikan hak akses kepada *client* tertentu yang memiliki kunci *private*. Sehingga, jika *client* yang tidak sah mencoba untuk mengendalikan *server* dari jarak jauh (*remote*), maka usaha koneksinya tidak akan berhasil sekalipun dia mengetahui urutan *knocking* menuju *server* tersebut (Putra, Rumani, & Purwanto, 2012).

Kemudian penelitian selanjutnya adalah penelitian tentang *random Port Knocking* (Kristianto, 2015). Sebagaimana yang diketahui, *Port Knocking* cenderung menerapkan urutan yang tetap pada proses *knocking* menuju sebuah SSH *server*. Namun pada penelitian tersebut, *port* statis pada metode *Port Knocking* dibuat menjadi *random* atau acak. Sehingga administrator yang melakukan pemeliharaan (*maintanance*) sistem tidak perlu lagi rutin melakukan perubahan urutan *knocking* secara manual kemudian menginformasikan kepada *client* urutan *port* yang saat ini dapat diketukkan (*knock*). *Client* hanya perlu mengakses sebuah sistem dengan mengirimkan informasi permintaan daftar *Port Knocking*, kemudian sistem tersebut akan secara otomatis menginformasikan *client* daftar *port* yang telah dihasilkan (*generate*) oleh sistem yang dapat digunakan untuk *knocking* ke SSH *server*.

Permasalahan pada penelitian ini adalah jalur komunikasi antara *client* dengan sistem yang mengirimkan informasi daftar *port* tersebut itu berada pada jalur yang tidak aman (Kristianto, 2015). Sehingga pihak yang tidak sah masih dapat melakukan *intercept* pada proses pengiriman informasi daftar *port* acak tersebut.

Berdasarakan uraian latar belakang tersebut, peneliti telah mengembangkan sebuah sistem *Secure Random Port List Generator* atau SRPLG yang akan diterapkan pada mekanisme autentikasi *Port Knocking*. Dinamakan SRPLG karena mampu mengacak daftar *Port Knocking* pada setiap upaya koneksi ke SSH server dan mampu mengirim informasi daftar *port* teracak tersebut kepada client melalui implementasi protokol SSL atau *Secure Socket Layer* pada jalur komunikasinya. Sehingga, melalui penerapan SRPLG ini, diharapkan tingkat keamanan pada celah keimanan jaringan, khususnya pada aktivitas remote server, dapat mengurangi upaya-upaya eksploitasi terhadap sebuah service yang diamanakan dengan menggunakan metode autentikasi *Port Knocking*.

2. Metode Penelitian

Secure Random Port List Generator adalah sebuah *server* yang dirancang dengan dua fungsi utama, yaitu 1) Mampu menangani proses pengacakan daftar *Port Knocking*; dan 2) Menjamin informasi yang ditransimisikan antara *server* SRPLG dan *client* selama proses *request* dan *response* berada pada jalur yang aman.

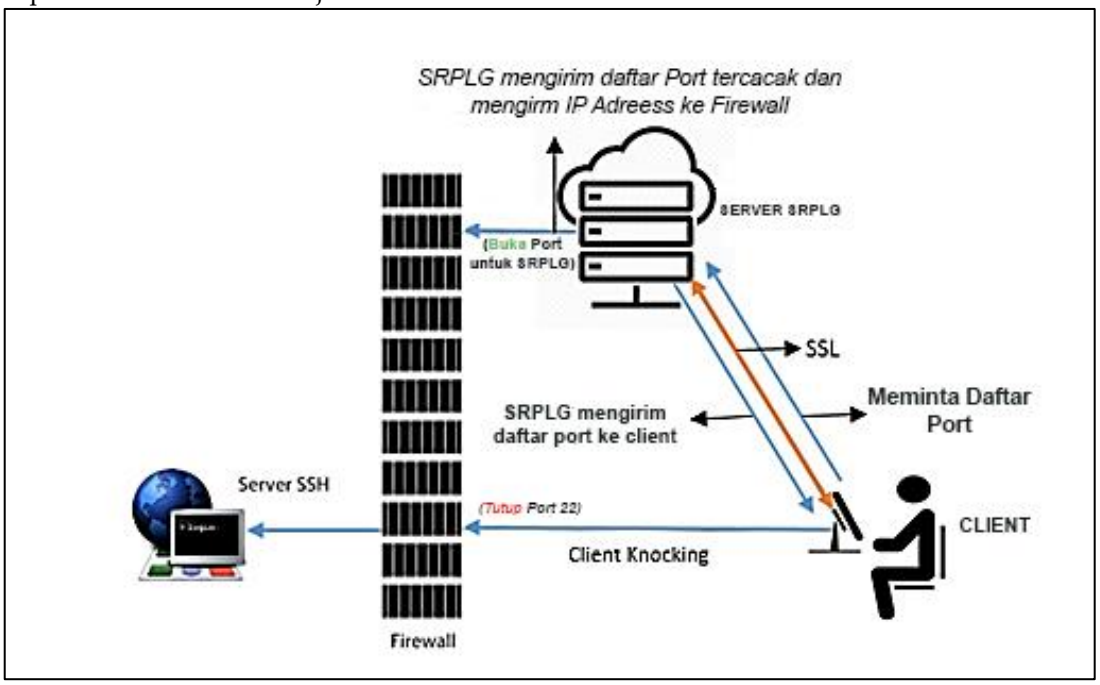
Dalam arsitektur sistem ini yang dapat dilihat pada Gambar 1**,** bahwa akan digunakan dua buah *server* yaitu SSH *server* dan SRPLG *server*. *Client* akan melakukan koneksi pada kedua *server* tersebut dengan terlebih dahulu melakukan *request* daftar *Port Knocking* ke SRPLG *server*. Setelah mendapatkan daftar *Port Knocking* yang telah diacak, *client* akan melakukan koneksi ke SSH *server* dengan melakukan *knocking* sesuai dengan urutan daftar *port* yang diperoleh dari SRPLG *server*.

2.1. Komponen arsitektur sistem

Komponen utama arsitektur sistem pada penelitian ini terdiri dari kombinasi komponen dasar pada metode autentikasi *Port Knocking* konvensional dan implementasi SRPLG *server*. Metode *Port Knocking*

ISSN 2502-3357 (*Online*)
A. Rauf dkk. /Register 4 (2) 2018 103-113 ISSN 2503-0477 (*Print*)

konvensional secara umum sangat bergantung pada konfigurasi *firewall*. Sementara, *firewall* hanya mampu mengkonfigurasi daftar *Port Knocking* yang bersifat tetap. Sehingga melalui penerapan SRPLG ini, *firewall* dapat bekerja lebih optimal dalam menerapkan aturan *Port Knocking* yang lebih dinamis pada proses autentikasi menuju SSH *server*.



Gambar 1. Arsitektur sistem penerapan SRPLG pada mekanisme autentikasi *Port Knocking*

Adapun komponen-komponen utama arsitektur sistem penelitian ini adalah sebagai berikut :

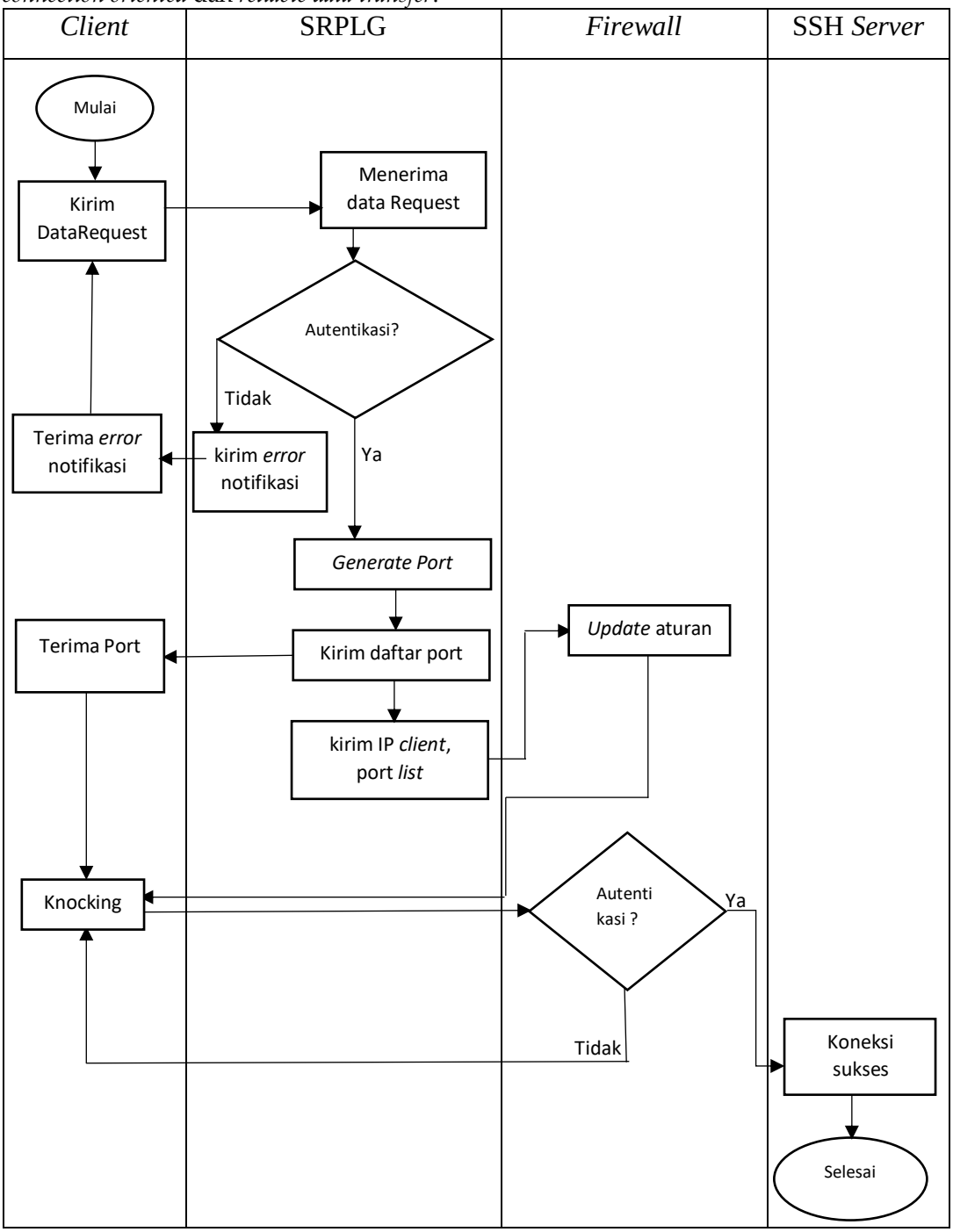
1. *Secure Random Port List Generator Server* (SRPLG *Server*)
   Secure Port List Generator Server (SRPLG *Server*) adalah sebuah *server* yang dibangun dengan menggunakan bahasa Python. SRPLG *Server* ini dirancang untuk dapat mengacak daftar *Port Knocking* dan dapat memberikan informasi kepada *client* melalui jalur aman, serta mampu melakukan pembaruan aturan *knocking* di *firewall*.
2. *Client*
   Client pada penelitian ini memiliki dua aktifitas utama, pertama *client* harus melakukan *request* daftar port teracak. Kemudian, setelah memperoleh daftar port dari *server* SRPLG, *client* dapat melakukan *knocking* berdasarkan urutan port yang diterima, dengan mengetuk SSH *server* menggunakan *tools* yang dapat membuka koneksi dan mengirimkan paket *Transmission Control Protocol* (TCP) ataupun *User Datagram Protocol* (UDP).
3. *Firewall*
   Firewall yang digunakan pada penelitian ini adalah Iptables. Firewall memiliki tanggung jawab untuk mengatur lalu Linas jaringan. Menurut Novrianda (2018), *firewall* ini akan dikoneksikan dengan SRPLG *server* agar dapat mengubah *rule Port Knocking* setiap ada *request* dari *client*. Iptables ini berfungsi untuk merekayasa SSH *server* agak portnya terlihat tertutup bagi *client* yang tidak sah, dan hanya membuka koneksi jika *client* melakukan prosedur *knocking* dengan benar.
4. SSH *Server*
   SSH *server* pada penelitian ini merupakan tujuan utama dari proses *Port Knocking*. Untuk mengakses SSH *server*, *client* perlu melakukan serangkaian proses mulai dari melakukan autentikasi ke SRPLG *server*, melakukan *request Port Knocking* dan setelah itu *client* dapat melakukan autentikasi ke SSH *server* dengan metode *Port Knocking*.

2.2. *Perancangan sistem*

SRPLG dibangun dengan konsep *socket programming*. *Socket* adalah mekanisme komunikasi untuk pertukaran data antar aplikasi yang terdapat di dalam sebuah mesin maupun beda mesin, dan pertukaran ini terjadi pada sebuah jaringan komputer. Ada dua jenis *socket* yang dapat digunakan untuk membangung aplikasi, yaitu TCP *socket* dan UDP *socket*. Pada penelitian ini, TCP *socket* akan

A. Rauf dkk. /Register 4 (2) 2018 103-113
ISSN 2502-3357 (*Online*)
ISSN 2503-0477 (*Print*)

digunakan untuk membangun SRPLG *server* karena lebih sesuai dengan konsep perancangan penelitian yang *connection oriented* dan *reliable data transfer*.



Gambar 2. *Flowchart* metode penelitian

SRPLG *Server* ini di rancang dengan memanfaatkan *modul random* pada bahasa pemrograman Python yang mampu melakukan proses pengacakan angka *integer*. Adapun *range* angka yang akan diacak berkisar dari 1024 hingga 65536. Acuan tersebut berdasar pada *Dynamically Assigned Port*. Angka *integer* tersebut kemudian akan dikonversi menjadi port yang akan dikirimkan ke *firewall* dan ke *client*. *Firewall* dan SRPLG *server* diasumsikan berada pada jaringan *private*. Sementara SRPLG *server* dan *client* berada pada jaringan *public*, sehingga untuk melindungi informasi yang ditransimiskan maka akan digunakan SSL untuk meningkatkan *confidentiality* daftar port yang dikirimkan.

Penjelasan *Flowchart* Gambar 2 adalah sebagai berikut:

1. *User* atau *client* melakukan koneksi ke SRPLG *server* dengan mengirimkan permintaan untuk membentuk koneksi SSL jika permintaan koneksi diterima, maka jalur aman antara *client* dan SRPLG *server* akan terbentuk.

ISSN 2502-3357 (*Online*)
A. Rauf dkk. /Register 4 (2) 2018 103-113 ISSN 2503-0477 (*Print*)

2. Setelah jalur aman terbentuk, SRPLG *server* akan melakukan pengacakan *Port Knocking*.
3. Kemudian SRPLG *server* mengirimkan informasi port yang sudah diacak ke *client* yang melakukan *request* dan mengirimkan informasi port teracak serta IP *address client* ke *Firewall*.
4. Setelah *client* memperoleh daftar *Port Knocking*, aplikasi *client* akan melakukan koneksi ke SSH *server* dengan melakukan *knocking* dengan port yang diperoleh dari SRPLG *server*.
5. Kemudian *firewall* akan melakukan pengecekan terhadap IP *address client* dan *port* yang diketuk oleh *client*.
6. Jika berhasil maka *client* akan mendapatkan hak akses menuju SSH *server*.

2.3. Tahapan pengujian sistem

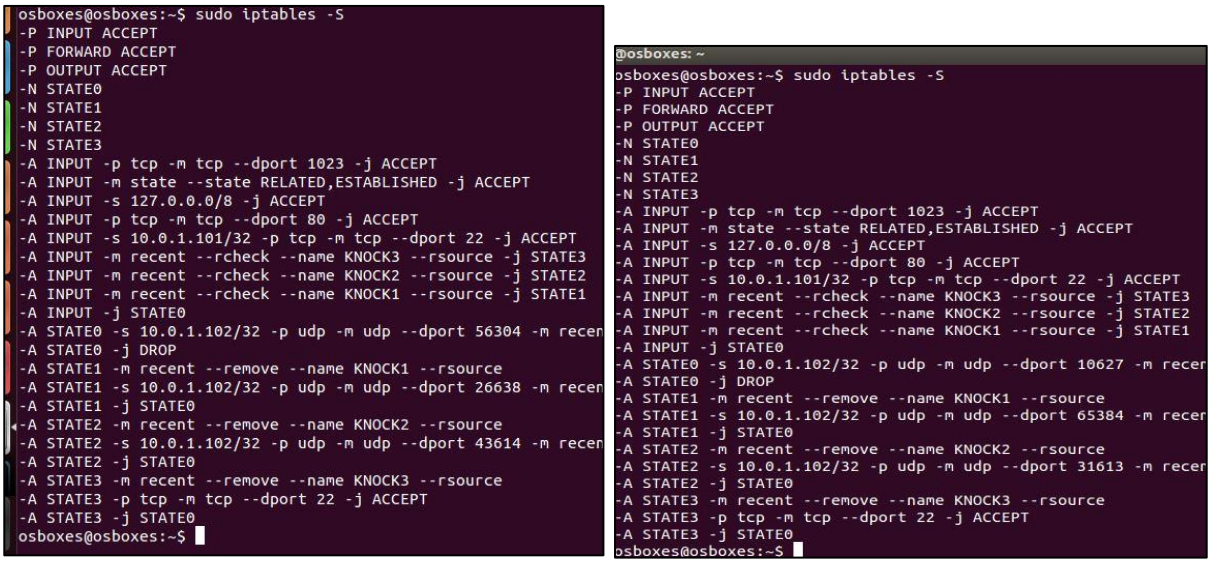
Adapun tahapan pengujian yang dilakukan untuk mengetahui tingkat keberhasilan sistem yang dirancang adalah dengan memastikan bahwa SRPLG *server* mampu melakukan *genarate* daftar port yang akan digunakan untuk melakukan *knocking* ke SSH *server*. Kemudian akan dilakukan pengamatan pada jalur komunikasi antara SRPLG *server* dengan *client* pada saat pertukaran data dengan memanfaatkan utilitas Wireshark untuk melakukan *sniffing*, pengujian ini bertujuan memastikan bahwa implementasi jalur aman dengan SSL telah berhasil mengenkripsi jalur komunikasi SRPLG *server* dan *client* dengan baik.

Selain pengujian tersebut akan dilakukan pula pengujian performa sistem terhadap jumlah *client*. Pengujian ini bertujuan untuk mengetahui tingkat efisiensi sistem dalam menangani banyaknya permintaan dari *client*. Untuk dapat melayani jumlah permintaan daftar port dari *client*, maka SRPLG *server* telah dirancang untuk dapat menjalankan *multiprocessing* dengan memanfaatkan modul *thread* pada pemrograman Python.

Tahapan terakhir dari sistem yang telah dirancang adalah pengujian sistem terhadap port menuju SSH *server*. Pengujian ini dimaksudkan untuk mengetahui tingkat keberhasilan desain sistem yang telah dirancang terhadap mengatasi upaya eksploitasi pada SSH *server*.

3. Hasil dan Pembahasan

Setelah penerapan SRPLG pada mekanisme autentikasi dengan menggunakan *Port Knocking* berhasil diimplementasikan, maka akan dilakukan pengujian terhadap keberhasilan SRPLG dalam menjalankan dua fungsi utama yang diimplementasikan di dalamnya, yaitu mengacak daftar *Port Knocking* dan mampu melindungi kerahasiaan data yang ditransmisikan SRPLG *server* dan *client*. Skenario pengujian yang dilakukan untuk menguji keberhasilan SRPLG *server* mengacak port adalah dengan membandingkan keadaan aturan di *firewall* sebelum menerima *request* dari *client*, dan setelah mendapatkan permintaan daftar port oleh *client*. Pengujian selanjutnya adalah pengujian terhadap jalur komunikasi antara SRPLG *server* dan *client* dengan melakukan pemantauan dan penangkapan *packet* atau *sniffing* terhadap informasi yang dilakukan *capture* pada saat SRPLG *server* dan *client* saling berkomunikasi.

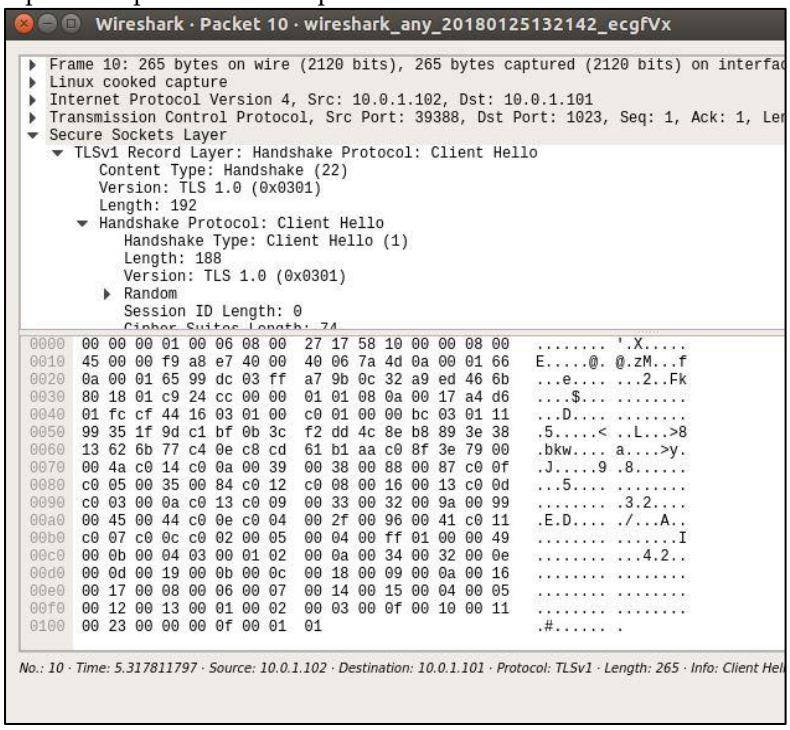


(a) (b)

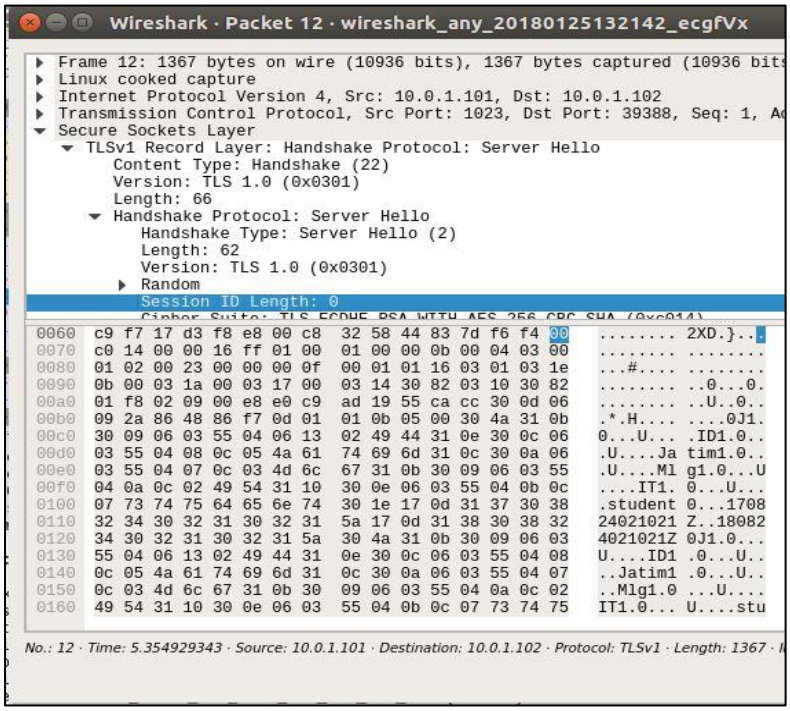
Gambar 3. Perbandingan aturan (a) Port sebelum *generate* pada *firewall*, (b) Port setelah *generate* pada *firewall*

ISSN 2502-3357 (*Online*)
ISSN 2503-0477 (*Print*)

Kemudian pengujian selanjutnya adalah menguji kemampuan SRPLG *server* terhadap jumlah *client* yang melakukan *request* daftar port. Parameter pengujian ini lebih kepada mengukur seberapa efisien kinerja sistem dalam menangani permintaan *client* untuk mendapatkan daftar port yang telah diacak. Skenario pengujian ini adalah dengan melakukan penjadwalan pada beberapa *client* untuk melakukan koneksi secara bersamaan ke SRPLG *server* dengan menggunakan utilitas Crontab pada sistem operasi Ubuntu, *client* akan melakukan koneksi ke SRPLG *server* pada waktu yang sama, parameter pengujian ini adalah seberapa banyak waktu yang dibutuhkan oleh sistem dalam menanggapi setiap permintaan daftar port. Pengujian terakhir pada sistem yang telah dirancang ini adalah dengan melakukan serangan *scanning port*. Tujuan pengujian ini adalah untuk mengamati kerentanan yang dapat dieksploitasi terhadap SSH *server*.



(a)



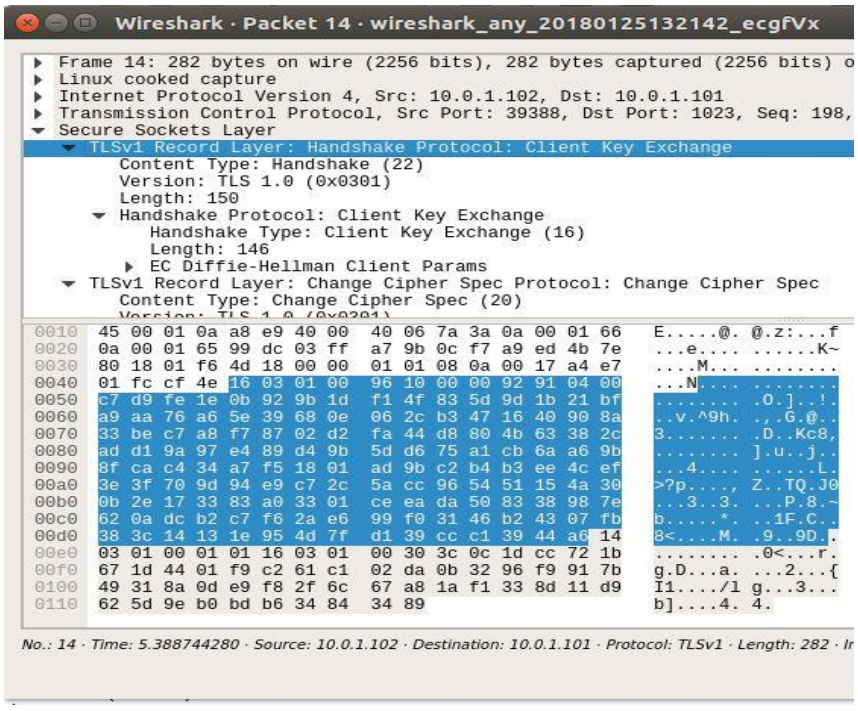
(b)

Gambar 4. (a) Informasi SSL/TLS *Client Hello*; (b) Informasi SSL/TLS *Server Hello*

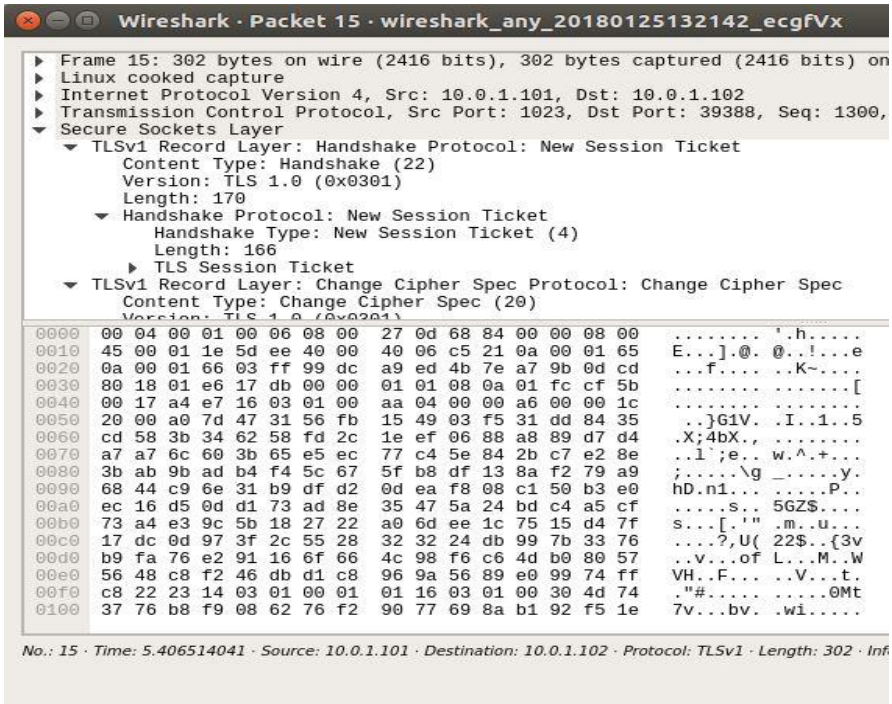
ISSN 2502-3357 (*Online*)
A. Rauf dkk. /Register 4 (2) 2018 103-113 ISSN 2503-0477 (*Print*)

3.1. Pengujian keberhasilan mengacak daftar Port Knocking

Tujuan pengujian keberhasilan mengacak port ini adalah untuk mengetahui apakah sistem yang telah dibangun dapat melakuan pengacakan port yang sesuai dengan perancangan sistem yang telah dilakukan. Skenario pengujian adalah dengan membandingkan kondisi *firewall* sebelum SRPLG *server* menerima permintaan *request* daftar *Port Knocking* dari *client*, dan setelah daftar *Port Knocking* diperbaharui.



(a)



(b)

Gambar 5. (a) Informasi SSL/TLS *Client Key Exchange*; (b)) Informasi SSL/TLS *New Session Ticket*

Gambar 3 menunjukkan keadaan *rule* atau aturan *firewall* pada kondisi Gambar 3 (a) sebelum dan Gambar 3 (b) setelah *port* di *generate* oleh *server* SRPLG. Pada kondisi awal sebelum adanya permintaan

A. Rauf dkk. /Register 4 (2) 2018 103-113
ISSN 2502-3357 (*Online*)
ISSN 2503-0477 (*Print*)

daftar *port* dari *client*, konfigurasi aturan *firewall* untuk proses membuka koneksi ke SSH *server* dengan metode *Port Knocking* adalah 56304, 26638 dan 43614. Kemudian pada aturan *firewall* berikutnya menampilkan kondisi aturan *firewall* telah berubah. Hal ini menunjukkan bahwa aturan daftar *Port Knocking* telah diperbaharui oleh *server* SRPLG setelah menerima permintaan daftar *port* dari *client* dengan IP *address* 10.0.1.102. Adapun urutan daftar *Port Knocking* yang dapat digunakan untuk autentikasi ke SSH *server* setelah proses *generate* tersebut adalah 10627, 65384 dan 31613.

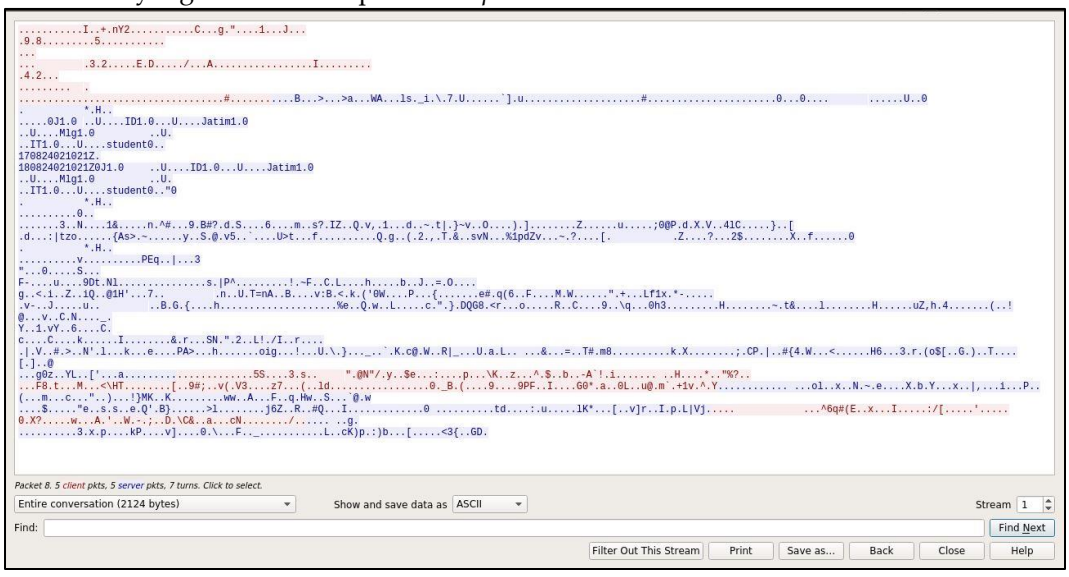
Pada konfigurasi aturan *firewall* pada sistem ini, daftar port yang dilakukan *generate* oleh SRPLG *server* hanya dapat digunakan oleh *client* yang melakukan *request* daftar *Port Knocking*, dalam hal ini yaitu *client* dengan IP *address* 10.0.1.102. Selain dari *client* tersebut, maka setiap upaya koneksinya tidak akan berhasil. Sehingga dengan demikian dapat disimpulkan bahwa dua parameter keberhasilan autentikasi tersebut harus dipenuhi untuk dapat membuka akses menuju SSH *server*.

3.2. Pengujian jalur komunikasi SRPLG server dan client

Tujuan pengujian jalur komunikasi SRPLG *server* adalah untuk mengetahui apakah jalur komunikasi yang dilalui oleh *client* dan SRPLG *server* telah berhasil membentuk jalur aman yang dienkripsi dengan SSL, ketika terjadi proses permintaan daftar port dan pada proses pengiriman daftar port ke *client*. Gambar 4 memperlihatkan IP *address client* adalah 10.0.1.102, sedangkan IP *address* dari SRPLG *server* adalah 10.0.1.101. Hasil *capture* Gambar 4 (a) dapat dilihat bahwa *client* sedang berusaha membentuk koneksi SSL melalui prosedur *handshake* dengan SRPLG *server* dengan mengirimkan *Client Hello*. Kemudian pada Gambar 4 (b) diperlihatkanbahwa SRPLG *server* telah merespon permintaan koneksi SSL dari *client* dengan mengirimkan *Server Hello* yang berisi informasi salinan sertifakat SSL dari SRPLG *server*.

Gambar 5 (a) dapat dilihat bahwa *client* telah menerima salinan sertifikat SSL dari SRPLG *server*. Jika *client* memverifikasi salinan sertifikat tersebut, maka SRPLG *client* akan mengirimkannya kembali untuk ditandatangani oleh SRPLG *server*. Gambar 5 (b) diperlihatkan bahwa *server* telah menyetujui permintaan koneksi SSL, dengan mengirimkan pengakuan yang telah ditandatangani secara digital, kemudian sesi enkripsi SSL sudah dapat dimulai. Dengan demikian telah terjalin komunikasi yang aman antara *client* dan SRPLG *server*.

Gambar 6 adalah hasil *capture* terhadap data yang ditransmsikan oleh SRPLG *server* dan *client*. Protokol SSL yang digunakan untuk menjamin *confidentiality* daftar *Port Knocking* menujukkan bahwa sistem ini telah berhasil membentuk jalur komunikasi yang aman. Sebab setelah dilakukan analisis terhadap data yang dipertukarkan, ditemukan bahwa seluruh informasi yang terdapat didalamnya adalah informasi yang telah terenkripsi bukan *plain-text*.



Gambar 6. Hasil *sniffing Server* SRPLG dan *client*

Dengan demikian SRPLG ini dapat dikategorikan mampu menjamin seluruh informasi yang dikirimkan ke *client* untuk melakukan *knocking* ke SSH *server* adalah informasi yang kerahasiaan telah terjamin keamanan, dan kerahasiaannya melalui implementasi SSL yang diterapkan pada jalur komunikasinya.

ISSN 2502-3357 (*Online*)
A. Rauf dkk. /Register 4 (2) 2018 103-113 ISSN 2503-0477 (*Print*)

3.3. Pengujian peforma SRPLG server terhadap jumlah client

Tujuan dari pengujian ini adalah untuk mengetahui tingkat efisiensi sistem dengan menghitung total waktu yang dibutuhkan oleh *client* untuk mendapatkan daftar *port* dari SRPLG *server*. Skenario pengujian performa *request* dan *response* antara *client* ke SRPLG *server* dilakukan dengan cara menjadwalkan sejumlah *client* untuk melakukan permintaan daftar port secara bersamaan ke SRPLG *server* dengan menggunakan Crontab. Kemudian untuk mengetahui peforma *request* dan *response* akan dilakukan pengecekan terhadap *log* masing-masing *client* yang telah dijadwalkan untuk melakukan proses *request port* ke SRPLG *server* secara bersamaan.

Tabel 1 dapat dilihat bahwa pada percobaan ini dilakukan pengujian dengan variasi jumlah *client* 2, 4, 6 dan seterusnya hingga 16 *client*. Pengujian ini telah dilakukan pencatatan *log* waktu yang dihabiskan oleh SRPLG *client* untuk mendaptkan *daftar port* dari SRPLG *server* di setiap variasi pengujiannya. Waktu yang ditampilkan merupakan total waktu yang diperoleh dari waktu *request response* masing-masing *client*.

Tabel 1. Hasil pengujian peformansi request response ke SRPLG *server*

Jumlah *Client*	**Waktu *Request Response* (Detik)**
2	0,05
4	0,0475
6	0,0533
8	0,055
10	0,056
12	0,054
14	0,0608
16	0,0525

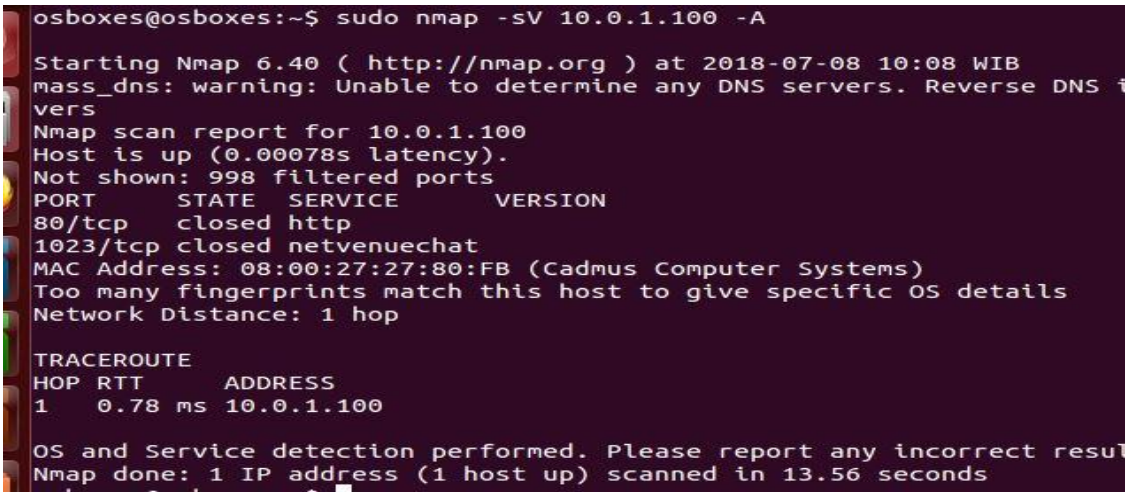
Dari hasil pengujian pada Tabel 1 dapat dilihat bahwa waktu *request response* pada pengujian autentikasi dengan variasi pengujian hingga 16 *client* menampilkan informasi yang dinamis. Seperti pada pengujian pertama dan kedua dapat dilihat bahwa pada pengujian dengan 2 *client* tampak lebih banyak menghabiskan waktu dibandingkan pada pengujian dengan 4 *client*. Hal ini disebabkan karena waktu *request response* setiap *client* itu berbeda-beda, sehingga memungkinkan *client* untuk memperoleh daftar port lebih cepat atau bahkan lebih lambat pada setiap kali proses pengujian. Faktor utama yang menentukan *client* mendapatkan daftar port lebih cepat atau lebih lambat adalah kemampuan *processing* komputer *client* dalam menjalankan *request*. Perlu diketahui bahwa pada pengujian ini, *client* dijalankan pada mesin *virtual* dan peforma setiap PC yang digunakan dalam menjalankan sistem operasi berbeda-beda pula, ada yang lebih cepat dan ada pula yang sedikit lebih lambat. Namun jika diperhatikan pada Tabel 1, rata-rata waktu yang dihabiskan pada setiap variasi pengujian relatif sangat cepat, hal ini menunjukkan bahwa SRPLG *server* bekerja cukup responsif dalam menangani banyaknya permintaan daftar port oleh SRPLG *client*. Disamping itu, kemampuan SRPLG *server* dalam menangani banyak *client* yang melakukan *request* daftar *Port Knocking* dapat ditangani dengan baik dengan implementasi *thread* di dalamnya. Kemampuan *multiprocessing* yang dirancang untuk melayani banyaknya *request* dapat bekerja dengan baik. Sehingga dengan demikian, SRPLG *server* dapat merespon setiap permintaan daftar port dari *client* dalam satu waktu sekaligus.

3.4. Pengujian keamanan port SSH server

Tujuan pengujian keamanan port SSH *server* ini adalah mengetahui apakah celah eksploitasi terhadap SSH *server* dengan desain sistem yang telah dirancang masih rentan terhadap serangan dari luar. Dalam pengujian ini akan dilakukan serangan *scanning port* untuk melihat celah mana saja yang dapat dieksploitasi dari konfigurasi sistem yang telah dibangun.

A. Rauf dkk. /Register 4 (2) 2018 103-113
ISSN 2502-3357 (*Online*)
ISSN 2503-0477 (*Print*)

Gambar 7 merupakan hasil *scanning port attack* yang telah dilakukan pada IP *address* 10.0.1.100 yang merupakan alamat SSH *server*. Dengan menggunakan *tools* Nmap telah dilakukan proses *scanning* terhadap seluruh port pada mesin *server*. Kemudian ditemukan bahwa hanya port 80 yang merupakan port *Hypertext Transfer Protocol* (HTTP) dan port 1023 yang *merupakan port* yang digunakan pada penelitian ini yang berhasil ditampilkan dari hasil *scanning*. Jika diperhatikan port 22 menuju SSH *server* tidak dapat ditemukan pada hasil *port scanning*, ini menunjukkan bahwa konfigurasi aturan *firewall* telah bekerja dengan baik dalam merekayasa *port* SSH *server* agar tidak dapat dilalui dengan mekanisme autentikasi konvensional.



Gambar 7. Hasil *scanning port attack* dengan Nmap

Selain itu, dari informasi port yang ditemukan semuanya berada dalam status *closed*, yang berarti tidak dapat dilalui. Serangan seperti *Brute Force Attack* tentunya dapat dihindari, sebab untuk melakukan serangan tersebut dengan port tidak tersedia tentunya adalah sia-sia atau tidak dapat berlangsung. Sehingga dapat disimpulkan bahwa celah eksploitasi terhadap SSH *server* dapat ditangani dengan baik, sebab untuk dapat memperoleh akses menuju SSH *server* hanya dapat dilakukan dengan metode *Port Knocking* yang telah disempurnakan dengan penerapan SRPLG dalam menciptakan kedinamisan sistem dalam menentapkan aturan *Port Knocking* di *firewall*.

4. Kesimpulan

Hasil implementasi dan pengujian yang telah dilakukan dari penelitian ini dapat disimpulkan bahwa SRPLG yang dirancang dengan menggunakan model pemrograman *socket* berbasis TCP/ IP, dapat mengacak daftar *Port Knocking* sesuai dengan perancangan sistem. Kemudian penerapan SSL bekerja dengan sangat baik dalam mengamankan informasi daftar port yang ditransmisikan antara SRPLG *server* dan *client* dengan berhasil mengenkripsi jalur komunikasinya agar dapat terhindar dari serangan *attacker* seperti *sniffing*.

Melalui penerapan SRPLG pada mekanisme autentikasi ini telah terbukti mampu memberikan dampak keamanan yang lebih tinggi dalam mengamankan SSH *server*, dengan tidak sama sekali memberikan akses atau peluang kepada *attacker* untuk berkomunikasi dengan SSH *server* dengan metode autentikasi konvensional, sebab port menuju *service* tersebut telah dikonfigurasi agar hanya dapat diakses oleh *client* yang memiliki daftar *Port Knocking* yang telah dilakukan *generate* dan dikirimkan oleh SRPLG *server*.

Peneliti ini sangat berharap agar sekiranya ada pengembangan penelitian terkait penerapan metode ini pada mekanisme autentikasi *Port Knocking*. Dalam hal ini, penelitian selanjutnya disarankan agar dilakukan penelitian terkait penerapan SRPLG dengan implementasi SRPLG *server* dan SSH *server* berada pada jaringan *public*. Tujuan dari saran ini adalah untuk melihat peforma penerapan metode penelitian ini jika diimplementasikan pada jaringan yang lebih luas.

Kemudian hal lain yang bisa dikembangkan terkait penelitian ini adalah penambahan parameter keberhasilan autentikasi. Tujuan saran penambahan parameter autentikasi ini adalah untuk meningkatkan keamanan proses autentikasi, sehingga indikator keberhasilan *attacke*r untuk melakukan eksploitasi terhadap SSH *server* semakin sulit.

ISSN 2502-3357 (*Online*)
A. Rauf dkk. /Register 4 (2) 2018 103-113 ISSN 2503-0477 (*Print*)

5. Referensi

Ali, F. H., Yunos, R., & Alias, M. A. (2012). Simple *Port Knocking* method: Against TCP replay attack and port scanning. *Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)* (hal. 247-252). Kuala Lumpur: IEEE.

Fajri, M. S., Suhatman, R., & Putra, Y. E. (2014). Analisa *Port Knocking* Pada Sistem Operasi Linux Ubuntu Server 12.04 LTS. *Jurnal Teknik Elektro dan Komputer, 2*(1), 59-67.

Kristianto, D. Y. (2015). *Keamanan Jaringan Menggunakan Firewall Dengan Metode Random Port Knocking Untuk Koneksi SSH.* Badung: Universitas Udayana.

Mehran, P., Reza, E. A., & Laleh, B. (2012). SPKT: Secure Port Knock-Tunneling, an enhanced port security authentication mechanism. *2012 IEEE Symposium on Computers & Informatics (ISCI)* (hal. 145-149). Penang: IEEE.

Novrianda, R. (2018). Implementasi authentication Captive Portal pada Wireless Local Area Network PT. Rikku Mitra Sriwijaya. *Register: Jurnal Ilmiah Teknologi Sistem Informasi, 4*(2), 67-80.

Putra, A. E., Rumani, R., & Purwanto, Y. (2012). *Implementasi Remote Server dengan Metode Port Knocking Menggunakan Bahasa Phyton.* Bandung: Telkom University.

Rozi, M. F., Ijtihadie, R. M., & Anggoro, R. (2010). Implementasi Remote Server Menggunakan Metode *Port Knocking* Dengan Asymmetric Encryption. *Makalah Seminar Tugas Akhir, 2010*(Januari), 1-5.