Sistem Deteksi dan Pengendalian Serangan *Denial of Service* pada Server Berbasis Snort dan Telegram-API

Detection and Control System of Denial of Service Attack on Server Based on Snort and Telegram-API

Daniel Desma Mahendra[1], Fransiska Sisilia Mukti[2]
[1,2]Fakultas Teknologi dan Desain, Institut Teknologi dan Bisnis Asia Malang
E-mail: [1]danielbebek07@gmail.com, [2]ms.frans@asia.ac.id

Abstrak

Melihat pentingnya fungsi *server* sebagai penyedia informasi dan layanan dalam sebuah jaringan, maka penting untuk memastikan bahwa *server* selalu dalam keadaan aman dan dapat diakses dengan lancar. *Server* dituntut untuk memiliki tingkat realibilitas dan keamanan yang baik, karena banyak ancaman yang mungkin saja terjadi untuk mengganggu kinerja server, seperti adanya virus, serangan *brute force, denial of service (DoS)* dan sebagainya. Melakukan pemantauan secara manual terhadap kinerja *server* tentunya menjadi hal yang tidak memungkinkan, sehingga dibutuhkan sebuah sistem yang dapat menggantikan posisi manusia untuk melakukan pemantauan secara kontinu. Penelitian ini mengusulkan mekanisme sistem pemantauan dan pengendalian terhadap serangan DoS melalui pengkolaborasian metode *intrusion detection system* (IDS) dan *intrusion prevention system* (IPS). Pengintegrasian *Snort* sebagai IDS dan Telegram-API sebagai IPS diusulkan untuk meningkatkan keamanan pada lingkungan *server*. Serangkaian pengujian dilakukan untuk mengetahui efektivitas dari metode yang diusulkan melalui pengujian *nmap scanning*, *syn-flood attack,* serta pemblokiran IP. Hasil pengujian menunjukkan bahwa *snort* mampu mendeteksi serangan dalam kurun waktu 4,8 hingga 7 detik sementara notifikasi Telegram tersedia dalam kurun waktu 5,8 sampai 8,6 detik. Hasil pemblokiran IP melalui pesan teks Telegram juga telah berhasil memblokir IP penyerang sehingga tidak dapat lagi masuk ke dalam sistem (terbukti melalui adanya 100% *packet loss* dan tidaka da port yang terdeteksi pada proses *nmap scanning*).

Kata kunci: IDS, IPS, Server, Snort, Telegram

Abstract

Seeing the importance of the server's function as a provider of information and services in a network, it is important to ensure that the server is always safe and can be accessed smoothly. Servers are required to have a good level of reliability and security, because there are many threats that might occur to disrupt server performance, such as viruses, brute force attacks, denial of service (DoS) and so on. Manually monitoring server performance is certainly not possible, so we need a system that can replace humans for continuous monitoring. This study proposes a mechanism for monitoring and controlling systems against DoS attacks by collaborating the intrusion detection system (IDS) and intrusion prevention system (IPS) methods. The integration of Snort as IDS and Telegram-API as IPS is proposed to increase security in the server environment. A series of tests were conducted to determine the effectiveness of the proposed method through testing nmap scanning, syn-flood attacks, and IP blocking. The test results show that snort is able to detect attacks within 4.8 to 7 seconds while Telegram notifications are available within 5.8 to 8.6 seconds. The results of IP blocking via Telegram text messages have also succeeded in blocking the attacker's IP so that it can no longer enter the system (proven through 100% packet loss and no ports detected in the nmap scanning process).

Keywords: IDS, IPS, Server, Snort, Telegram

1. PENDAHULUAN

Cepatnya perkembangan teknologi di era revolusi industri 4.0 juga harus diimbangi dengan kemampuan dari para administrator dalam mengelola sistem yang dibangun. Tidak hanya mampu menyediakan informasi yang bervariatif dan *uptodate*, namun layanan informasi tersebut juga harus bersifat *reliable* dan *secure*. Sistem komunikasi dan penyebaran informasi yang saat ini tidak lagi terbatas ruang dan waktu justru menjadi sebuah peluang bagi para pelaku kejahatan di dunia internet, atau yang sering dikenal dengan istilah *cyber crime*.

Untuk itu, faktor keamanan dalam sebuah jaringan menjadi sebuah parameter penting yang harus dipersiapkan dengan baik untuk mencegah adanya usaha-usaha illegal dari oknum tidak bertanggungjawab, yang bertujuan untuk mencuri data atau menghancurkan sistem yang ada. Bahkan tidak jarang ditemui, hilangnya data karena *server* yang mengalami *down* atau *hang* akibat serangan yang datang secara eksplosif [1].

Berbagai serangan yang sering menjadi ancaman bagi jaringan seperti virus, *sniffing, spoofing, phreaking, remote attack* menjadi resiko-resiko yang harus dihadapi oleh para administrator sehari-hari. Bahkan lebih lanjut, serangan-serangan seperti *brute force, scanning, malware,* dan *Denial of Service* (DoS) menjadi masalah yang paling signifikan dalam jaringan karena yang paling sering muncul dan mengganggu kinerja layanan *server* [2]. Serangan-serangan ini umumnya diantisipasi melalui penyediaan *firewall rules*, baik secara langsung di *server* atau pengaktifan fitur di *router*. Namun ternyata, masih sering didapati kasus serangan-serangan yang tidak terdeteksi oleh *firewall* dan pihak administrator baru mengetahui adanya serangan setelah terjadi kelumpuhan sistem.

Mengingat krusialnya keamanan dalam lingkungan jaringan, maka melakukan pemantauan sistem selama 24 jam menjadi hal yang tidak memungkinkan jika harus dilakukan secara manual. Dibutuhkan bantuan sistem yang dapat menggantikan posisi manusia untuk melakukan pemantauan secara kontinu, yang pada akhirnya juga diharapkan mampu mencegah atau melakukan pengendalian terhadap serangan-serangan yang terjadi pada jaringan atau *server*. *Intrusion Detection System* (IDS) menjadi sebuah alternatif bagi para administrator dalam melakukan pemantauan jaringan dan mendeteksi setiap aktivitas yang mencurigakan atau cenderung mengarah kepada serangan. IDS akan merekam setiap aktivitas dalam jaringan dan memberikan sebuah peringatan kepada administrator apabila ditemukan potensi-potensi serangan pada *server* [3].

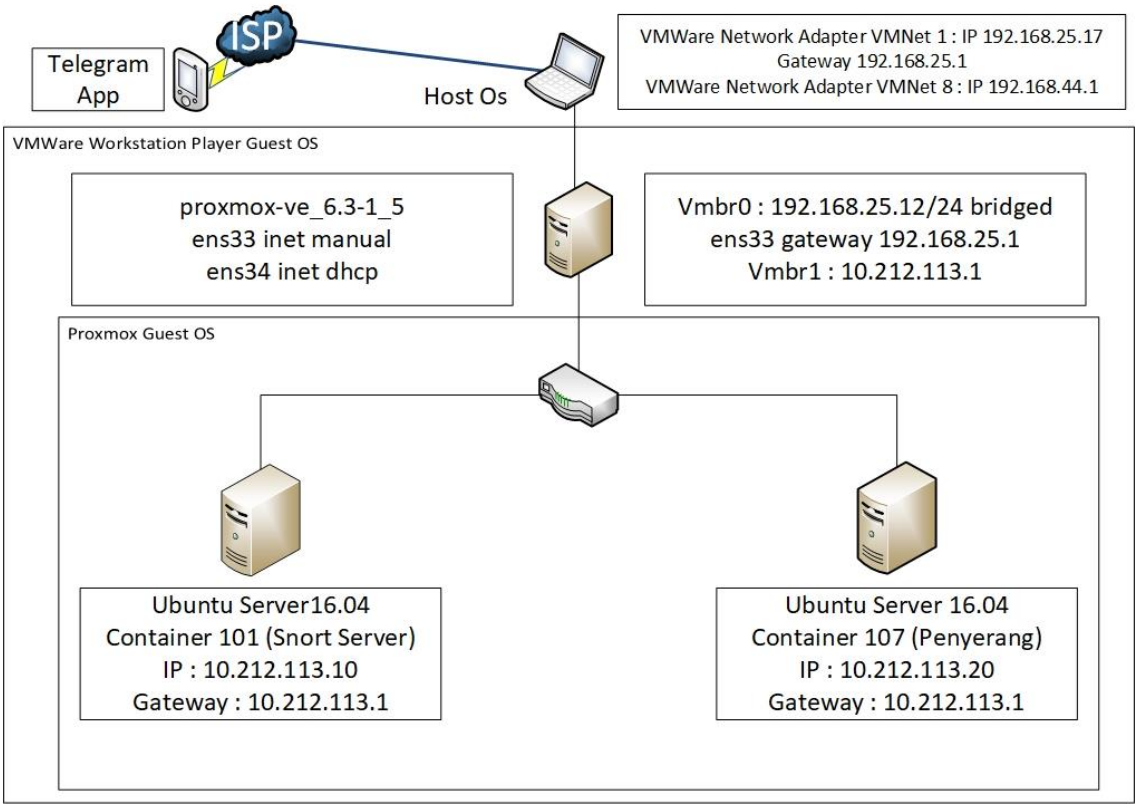
Beragam penelitian telah dilakukan dengan mengadopsi mekanisme IDS untuk mengamankan jaringan. Atmojo [4] memanfaatkan bot Telegram API sebagai pengiriman *alert* serangan kepada administrator dengan bantuan Raspberry Pi pada *web server*. Namun ada keterbatasan *resource* membuat sistem tidak dapat mengirimkan notifikasi untuk serangan-serangan besar. Bot yang sama juga digunakan oleh Fahana dkk [5] untuk keperluan forensic jaringan, sementara Syafari dan Panjaitan [6] dalam mendeteksi serangan pada *server* dengan tipe serangan *brute force,* DDoS, SSH dan *port scanning*. IDS dan Bot Telegram yang diajukan telah berhasil mendeteksi setiap serangan menggunakan aplikasi Hping3 dan Hydra. Telegram membantu dalam memberikan notifikasi (*alert)* dalam sistem deteksi serangan [7] maupun sebagai sistem pemantauan dalam sebuah jaringan [8]. Selain itu, mekanisme IDS juga dikorelasikan dengan *open source Snort* pada penelitian [9]–[11]. *Snort* membantu dalam melakukan deteksi serangan dan menyimpan aktivitasnya dalam sebuah *log* [12]. Hasil deteksi tersebut akan dilanjutkan dalam bentuk pemberian notifikasi serangan kepada administrator melalui *email* dan SMS, ataupun juga bisa melalui *cloud* [11], [13] maupun aplikasi *Firebase Cloud Messaging* [14]

Penggabungan mekanisme IDS, *Snort* dan Telegram API menjadi penelitian yang paling banyak dilakukan dalam mendeteksi serangan pada *server*. Namun masih belum didapati aktivitas lanjutan setelah serangan ditemui, atau lebih dikenal dengan istilah *Intrusion Prevention System* (IPS). Oleh karena itu, dalam penelitian ini, diusulkan sebuah penggabungan antara mekanisme IDS dan IPS dalam sebuah sistem deteksi dan pengendalian serangan pada *server* yang secara khusus ditujukan pada serangan *Denial of Service* (DoS).

2. METODE PENELITIAN

2.1 Gambaran Sistem

Kolaborasi mekanisme IDS dan IPS dalam penelitian ini diusulkan melalui skema pengintegrasian *Snort* sebagai perangkat lunak deteksi serangan dan Telegram-API sebagai perangkat lunak untuk notifikasi serangan dan pengendalian serangan dalam bentuk pemblokiran IP. Mekanisme kerja dari sistem yang diusulkan adalah sebagai berikut: *Snort* akan memantau aktivitas lalu lintas dalam jaringan dan merekamnya ke dalam sebuah *log file*. Ketika ditemukan aktivitas yang mencurigakan (berpotensi sebagai sebuah serangan), maka *snort* akan memberikan notifikasi kepada administrator melalui Telegram-API. Ketika trafik yang dipantau terbukti merupakan upaya serangan dari *hacker*, maka administrator dapat melakukan perubahan *firewall rules* pada *server* secara langsung dengan cara mengirimkan pesan melalui *bot* Telegram. Keseluruhan sistem akan dijalankan secara simulai dengan menggunakan *network simulator* VMWare seperti yang ditunjukkan melalui Gambar 1.



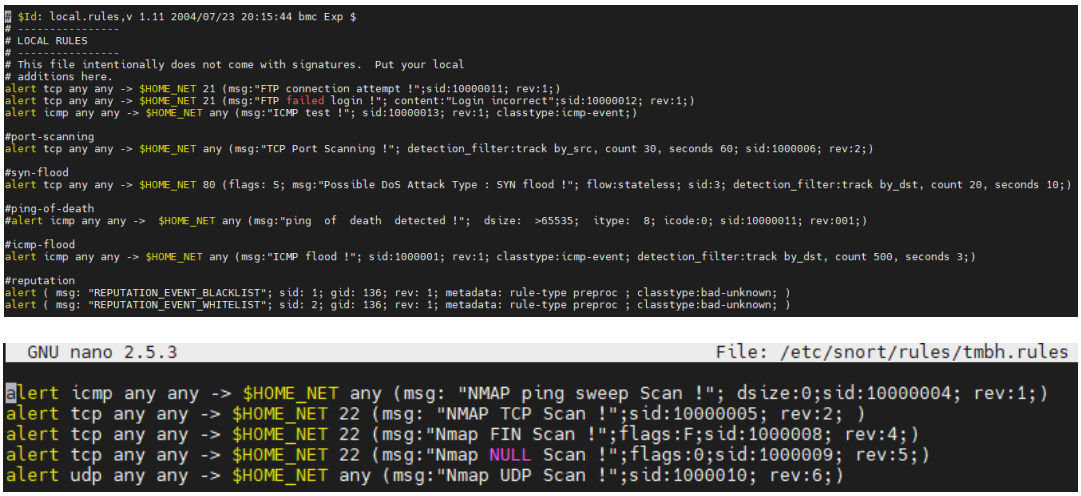
Gambar 1 Gambaran Sistem yang Diusulkan

2.2 Mekanisme Deteksi Serangan Berbasis Snort

Snort merupakan jenis IDS *singlethrading* yang menggunakan *rules* dalam bentuk teks untuk menjalankan perintah, melakukan penanganan serangan dan memberikan aksi atau eksekusi terhadap setiap *event* yang terdeteksi. *Snort* tidak hanya diterapkan pada aliran trafik masuk namun juga setiap trafik yang keluar dari jaringan. *Snort* sendiri dapat bekerja dalam 3 mode, yaitu sebagai penyadap (*sniffer*), penyimpan data (*packet logger*), dan sebagai pendeteksi serangan (*network intrusion detection*) [3].

Pada penelitian ini, *Snort* ditempatkan pada *server* utama (IP: 10.212.113.10) yang bertanggungjawab dalam merekam setiap aktivitas lalu lintas pada jaringan. Beberapa konfigurasi yang dilakukan pada *server Snort* antara lain konfigurasi dasar *snort* sebagai IDS, pembuatan *rules* dan konfigurasi *file bpf* sebagai *database* IP yang dianggap aman (*trusted IP*=bukan IP

penyerang). Gambar 2 menunjukkan beberapa bagian konfigurasi yang dilakukan pada *server Snort*.



```
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# ----------------
# LOCAL RULES
# ----------------
# This file intentionally does not come with signatures.  Put your local
# additions here.
alert tcp any any -> $HOME_NET 21 (msg:"FTP connection attempt !";sid:10000011; rev:1;)
alert tcp any any -> $HOME_NET 21 (msg:"FTP failed login !"; content:"Login incorrect";sid:10000012; rev:1;)
alert icmp any any -> $HOME_NET any (msg:"ICMP test !"; sid:10000013; rev:1; classtype:icmp-event;)

#port-scanning
alert tcp any any -> $HOME_NET any (msg:"TCP Port Scanning !"; detection_filter:track by_src, count 30, seconds 60; sid:1000006; rev:2;)

#syn-flood
alert tcp any any -> $HOME_NET 80 (flags: S; msg:"Possible DoS Attack Type : SYN flood !"; flow:stateless; sid:3; detection_filter:track by_dst, count 20, seconds 10;)

#ping-of-death
#alert icmp any any ->  $HOME_NET any (msg:"ping  of  death  detected !";  dsize:  >65535;  itype:  8; icode:0; sid:10000011; rev:001;)

#icmp-flood
alert icmp any any -> $HOME_NET any (msg:"ICMP flood !"; sid:1000001; rev:1; classtype:icmp-event; detection_filter:track by_dst, count 500, seconds 3;)

#reputation
alert ( msg: "REPUTATION_EVENT_BLACKLIST"; sid: 1; gid: 136; rev: 1; metadata: rule-type preproc ; classtype:bad-unknown; )
alert ( msg: "REPUTATION_EVENT_WHITELIST"; sid: 2; gid: 136; rev: 1; metadata: rule-type preproc ; classtype:bad-unknown; )
```

```
  GNU nano 2.5.3                                    File: /etc/snort/rules/tmbh.rules

alert icmp any any -> $HOME_NET any (msg: "NMAP ping sweep Scan !"; dsize:0;sid:10000004; rev:1;)
alert tcp any any -> $HOME_NET 22 (msg: "NMAP TCP Scan !";sid:10000005; rev:2; )
alert tcp any any -> $HOME_NET 22 (msg:"Nmap FIN Scan !";flags:F;sid:1000008; rev:4;)
alert tcp any any -> $HOME_NET 22 (msg:"Nmap NULL Scan !";flags:0;sid:1000009; rev:5;)
alert udp any any -> $HOME_NET any (msg:"Nmap UDP Scan !";sid:1000010; rev:6;)
```

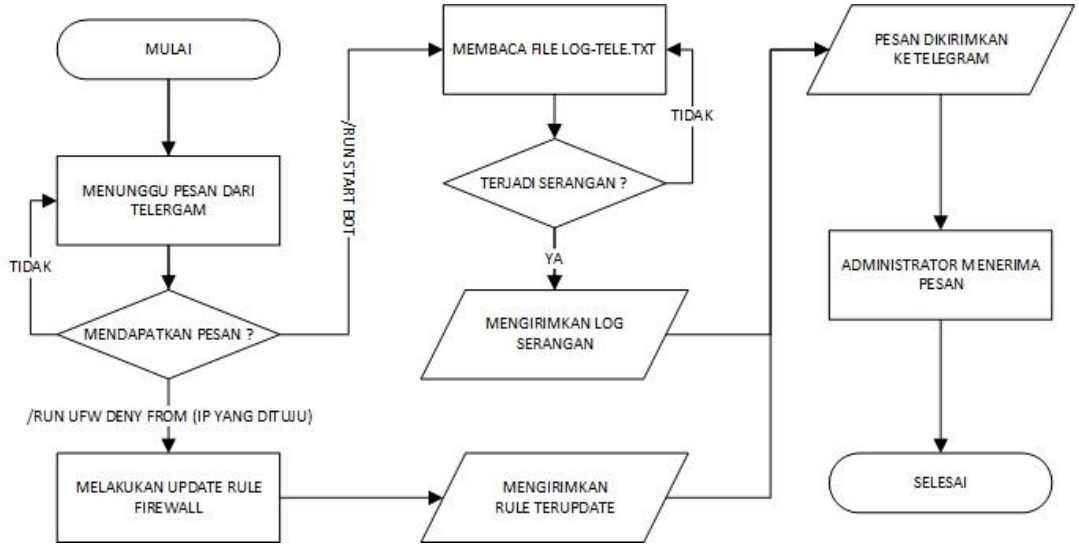
Gambar 2 Konfigurasi *Server Snort*

2.3 Mekanisme Pengendalian Serangan via Telegram-API

Telegram Bot-API merupakan sebuah perangkat lunak yang memungkinkan adanya interaksi antara *bot* dengan pengguna (administrator). *Bot* ini secara khusus dalam kepentingan keamanan jaringan memiliki kemampuan untuk mengirimkan perintah jarak jauh serta memberikan peringatan terhadap serangan [15].

Dalam implementasinya melalui penelitian ini, *bot* Telegram diintegrasikan dengan sistem *shell-bash* sehingga *bot* akan otomatis *stand by* saat *server* mulai dinyalakan. Ketika *bot* menerima pesan dalam bentu "*/run start bot"* maka *bot* akan otomatis menjalankan perintah untuk mencatat setiap aktivitas dalam jaringan dan merekamnya ke dalam file *log-tele.txt*. Namun jika tidak ada perintah apapun yang dijalankan, maka *bot* akan tetap dalam keadaan *stand by*. Prosedur ini merupakan prosedur yang dijalankan oleh *bot* Telegram sebagai mekanisme deteksi serangan.

Selanjutnya, ketika didapati bahwa ada serangan yang masuk ke dalan *server*, maka *Snort* akan mengirimkan notifikasi serangan kepada administrator melalui *bot* Telegram. Ketika administrator menerima pesan tersebut, maka administrator dapat memeriksa kondisi *server* terlebih dahulu atau dapat juga langsung melakukan eksekusi pemblokiran serangan dengan mengetikkan perintah *"/run ufw deny from (ip penyerang)"*. Ketika *bot* menerima pesan tersebut, maka *bot* akan segera melakukan *update firewall rules* sesuai dengan instruksi administrator. Prosedur ini merupakan prosedur yang dijalankan oleh *bot* Telegram sebagai mekanisme pengendalian serangan. Rangkuman prosedur deteksi dan pengendalian serangan melalui Telegram ditunjukkan melalui Gambar 3.



Gambar 3 Mekanisme Sistem via Telegram-API

Beberapa konfigurasi yang dilakukan untuk mengaktivasi mekanisme deteksi dan pengendalian serangan melalui Telegram-APi diuraikan sebagai berikut:

1) Pembuatan *bot* Telegram: proses ini dimulai dengan pembuatan nama *bot* melalui @BotFather, jika berhasil akan mendapatkan notifikasi HTTP API. Melalui HTTP API inilah akan dikirimkan chat id dari *bot* Telegram yang akan digunakan
2) Konfigurasi *shell-bot*: merupakan proses pengkonfigurasian perintah untuk melakukan pengendalian *server* melalui pesan di Telegram. Proses konfigurasi dilakukan melalui pengunduhan repository yang dilanjutkan dengan pendaftaran token bot.
3) Konfigurasi *bot sender*: merupakan proses pengkonfigurasian perintah untuk memulai *bot* mengirimkan notifikasi serangan maupun layanan perubahan *firewall rules* sesuai dengan instruksi administrator. Beberapa hal yang dilakukan di sini adalah pembuatan perintah di *shell Linux*, pembuatan *file log* penyimpan pesan dari Telegram, dan pembuatan *service* untuk pengendalian *server* via Telegram.

3. HASIL DAN PEMBAHASAN

a. Skenario Pengujian

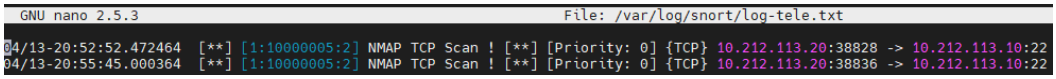
Proses pengujian yang dilakukan secara komprehensif dan bertahap mulai dari proses penyerangan, deteksi aktivitas dan potensi serangan, hingga pengendalian serangan melalui pesan Telegram. Prosedur pengujian sistem dibedakan menjadi 5 tahapan besar yang diuraikan sebagai berikut:

1) Pengujian *Snort system*: penyerang melakukan percobaan akses *server* melalui SSH pada port 22. Percobaan ini dilakukan guna mengetahui apakah server *Snort* berhasil mendeteksi aktivitas jaringan pada *server* tersebut dan menyimpannya pada log file yang telah dibuat.
2) Pengujian *bot* Telegram: prosedur pengujian yang dilakukan untuk mengetahui apakah *bot* Telegram berhasil mengirimkan notifikasi serangan pada *server*, menjalankan perintah pemblokiran dan menyimpan setiap rekaman aktivitas ke dalam *file log*.
3) Pengujian *SYN-Flood*: prosedur pengujian yang dilakukan untuk mensimulasikan serangan DoS terhadap server melalui port 80. Pada pengujian ini dilakukan apakah *snort* dan *bot* Telegram mampu mendapatkan notifikasi sesuai dengan serangan yang dilakukan.
4) Pengujian *nmap scanning*: prosedur pengujian untuk mensimulasikan cara penyerang dalam mendeteksi port – port yang tersedia. Pengujian ini dilakukan apakah *snort* dan *bot* telegram dapat mengenali jenis serangan yang telah dilakukan.

5) Pengujian pemblokiran IP: merupakan prosedur untuk pengendalian serangan melalui pesan Telegram. Prosedur ini dialkukan untuk menguji perintah yang dikirim melalui Telegram dapat terbaca di *server snort* dan melakukan *update firewall rules*. Pengujian akan dilakukan dua kali yaitu sebelum dan sesudah ip penyerang di blokir.

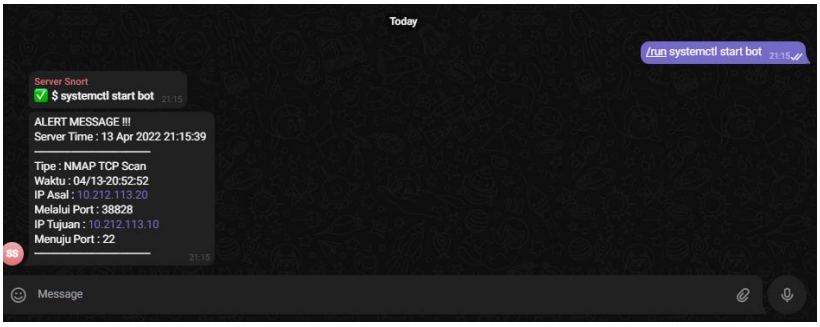
b. Hasil Pengujian Sistem

Sesuai dengan scenario pengujian yang telah dijabarkan sebelumnya, maka hasil pengujian sistem akan diuraikan sesuai dengan 5 tahapan pengujian yang telah dibuat. Pertama, pengujian akan difokuskan pada mekanisme IDS melalui *Snort.* Pengujian dilakukan dengan menggunakan perintah *ssh root@10.212.113.10 -p 22* dari *container* penyerang. Hasil pengujian menunjukkan bahwa *server Snort* dapat berjalan dengan baik, yaitu dengan memberikan informasi serangan yang terjadi tersimpan pada file log-tele.txt, seperti yang ditunjukkan pada Gambar 4.



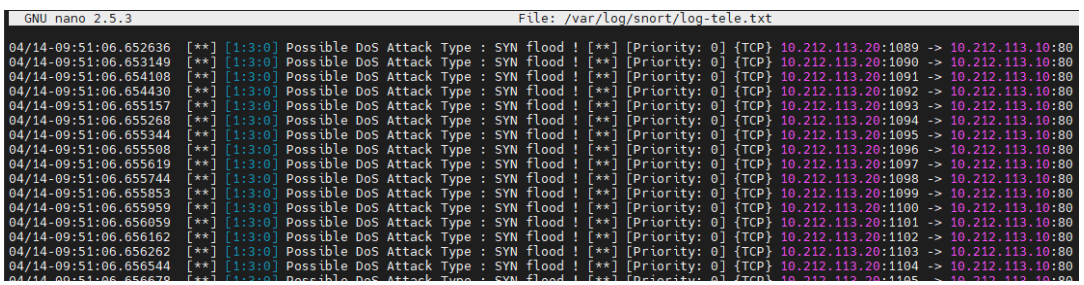
Gambar 4 Hasil Pengujian *Server Snort*

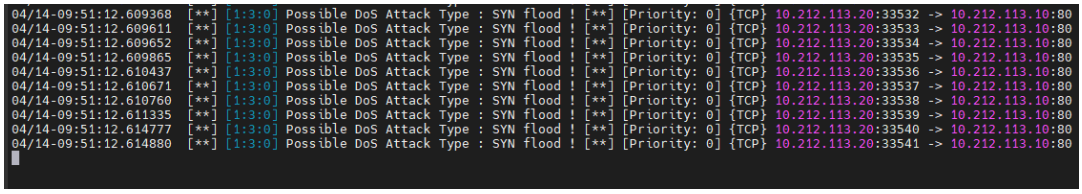
Selanjutnya, pengujian kedua dilakukan untuk mengetahui efektivitas *bot* Telegram dalam memberikan notifikasi dan pengendalian serangan melalui pesan teks. Pengujian dilakukan dengan mengetikkan perintah *"/run systemctl start bot"*. Gambar 5 menunjukkan bahwa *bot* Telegram dapat berfungsi dengan baik dan mengirimkan notifikasi terkait lalu lintas jaringan pada *server*.



Gambar 5 Hasil Pengujian *Bot* Telegram

Prosedur pengujian selanjutnya adalah proses untuk melakukan simulasi serangan. Pengujian dilakukan dari ip penyerang menuju *server snort* melalui port 80, dengan perintah *"hping3 -i u1 -S -p 80 10.212.113.10"*. Setelah dilakukan penyerangan dan penyerang berhasil mengirimkan 32492 paket pada waktu 09:51:01, sistem snort dapat memantau lalu lintas jaringan dan menyimpan log aktivitas pada *file log-tele.txt* dengan memiliki *delay* rata – rata sebesar 5 detik, dari mulai penyerangan pada waktu 09:51:01 namun pada *log-tele.txt* tercatat mulai dari 09:51:06, seperti yang ditunjukkan pada Gambar 6.

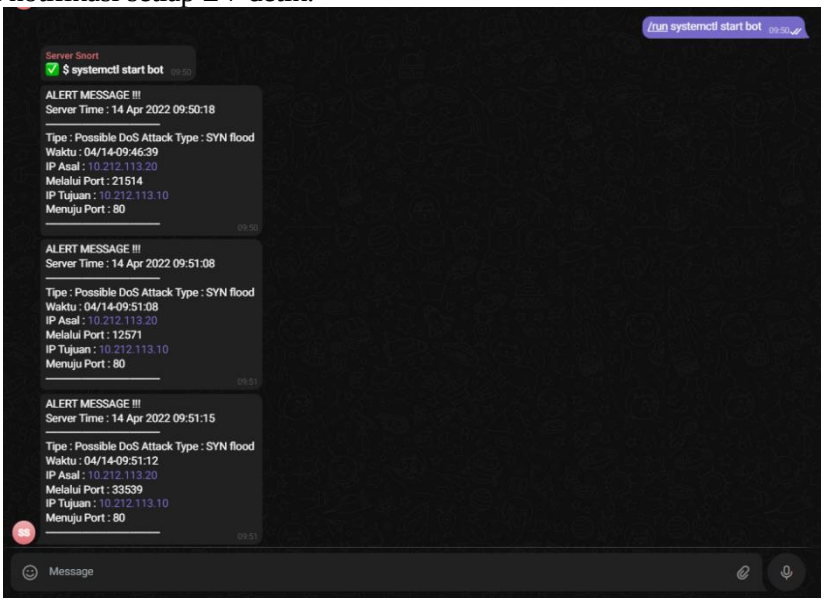




Gambar 6 Hasil Rekam Aktivitas Serangan pada *Snort*

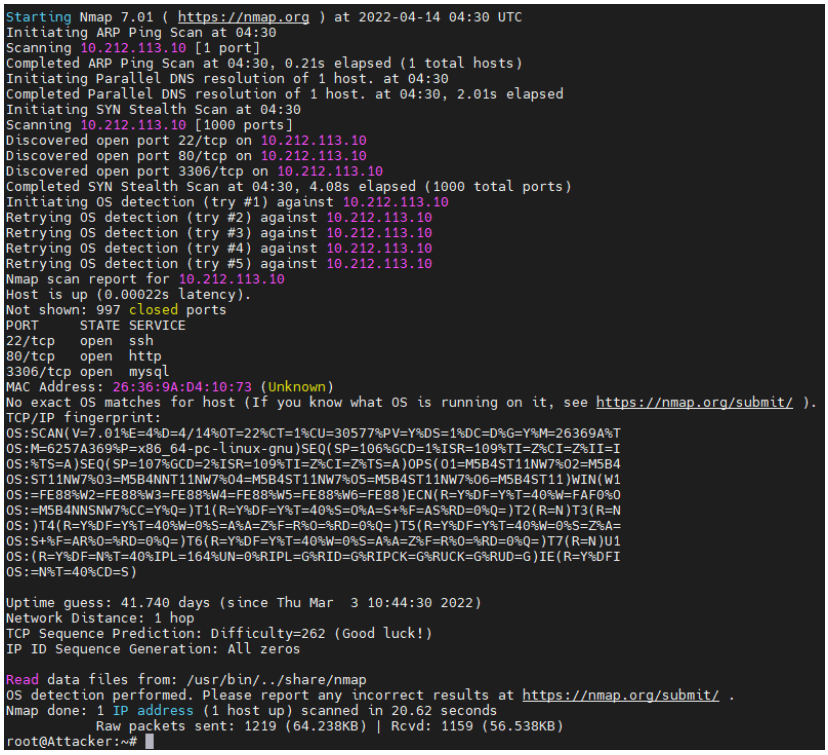
Log pada file *log-tele.txt* tersebut selanjutnya akan dibaca oleh implementasi *trigger bot* telegram, yang selanjutnya akan dikirimkan menuju telegram administrator server. Administrator akan menerima detail penyerangan yang terjadi sesuai dengan log serangan pada file *log-tele.txt.* Pesan tersebut berisi jenis serangan, waktu serangan, ip penyerang dan port yang di tuju oleh penyerang, yang dapat dilihat pada Gambar 7.

Dapat dilihat bahwa pesan yang diterima memiliki selisih waktu dan jumlah yang berbeda dengan log file yang disimpan oleh sistem snort, dari waktu log-file penyerangan pada waktu 09:51:06 sampai dengan 09:51:12 yang terkirimkan ke telegram hanya 2 paket yaitu serangan pada waktu 09:51:08 dan waktu 09:51:12. Dapat disimpulkan bahwa telegram api yang dibangun dapat mengirimkan informasi serangan dengan delay sebanyak 7 detik dari mulainya penyerangan, delay sebanyak 2 detik pengiriman notifikasi dari mulainya tercatat pada log file snort, dan dari lamanya serangan yang dilakukan bot telegram tersebut hanya mampu mengirimkan notifikasi setiap ± 7 detik.



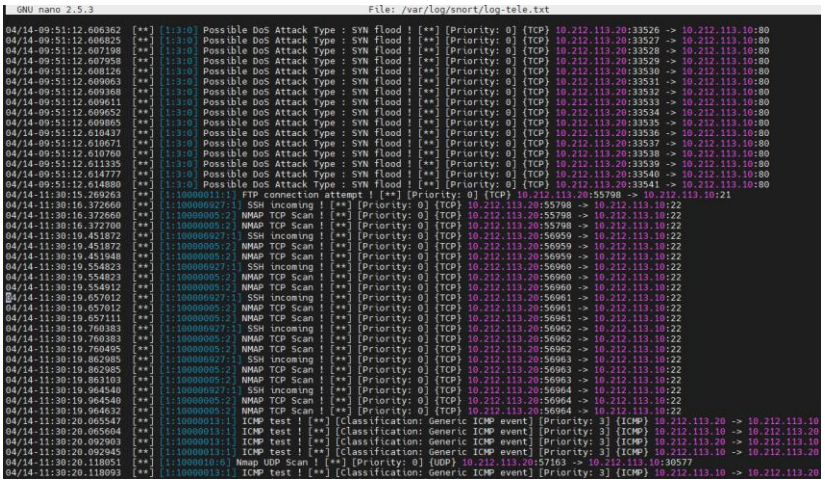
Gambar 7 Notifikasi Telegram Serangan SYN-Flood

Setelah berhasil dilakukan pengujian serangan syn-flood *server snort*, selanjutnya akan dilakukan pengujian serangan dengan menggunakan metode *"nmap scanning"*. Pengujian dilakukan untuk mengetahui port – port terbuka pada server snort, serta untuk menguji apakah sistem snort dapat mengetahui jenis serangan ini dan mengirimkannya ke telegram administrator, serangan dilakukan dengan perintah *"nmap -v -O 10.212.113.10"* dari penyerang, sebagaimana yang terlihat pada Gambar 8.



Gambar 8. Pengujian *Nmap Scanning*

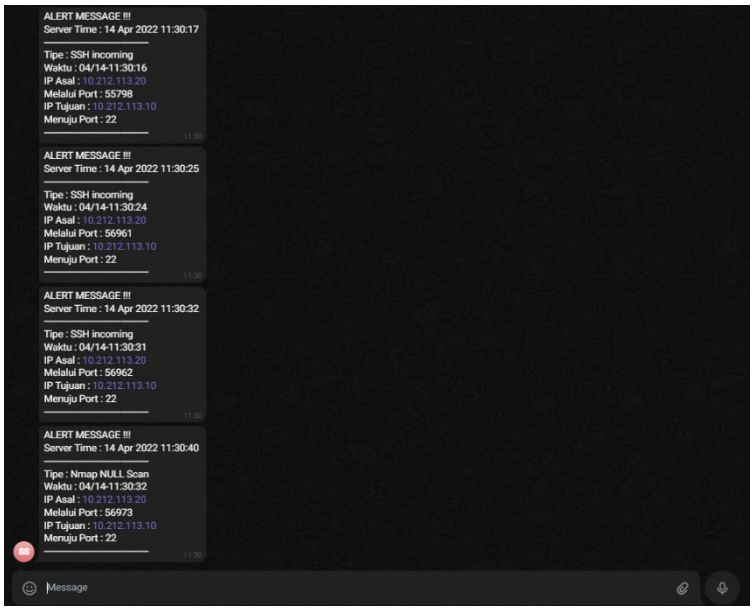
Setelah dilakukan penyerangan dan penyerang berhasil melakukan *scanning nmap* pada waktu 11:30:08 sampai dengan waktu 11:30:27, dapat dilihat bahwa penyerang berhasil mendapatkan informasi port yang terbuka pada *server snort.* Namun dapat dilihat juga bahwa sistem snort menangkap lalu lintas jaringan dan menyimpan log aktivitas pada file *log-tele.txt* pada waktu 11:30:15 sampai 11:30:32 yang berarti deteksi tersebut memiliki *delay* sebanyak 7 detik dari mulainya penyerangan dan *delay* sebanyak 5 detik terhadap berhentinya penyerangan, yang dapat dilihat pada Gambar 9.



Gambar 9 Log File *log-tele.txt* pada Serangan *Nmap Scanning*

Pada log file tersebut sistem snort berhasil menangkap lalu lintas jaringan dari ip 10.212.113.10 yang berupa *ftp connection attempt*, *nmap tcp scan*, *nmap udp scan*, *ssh incoming*, *nmap null scan* dan *icmp test* yang terjadi mulai waktu 11:30:15 sampai waktu 11:30:32. Log file tersebut akan mentrigger bot telegram sehingga telegram administrator mendapatkan pesan dari

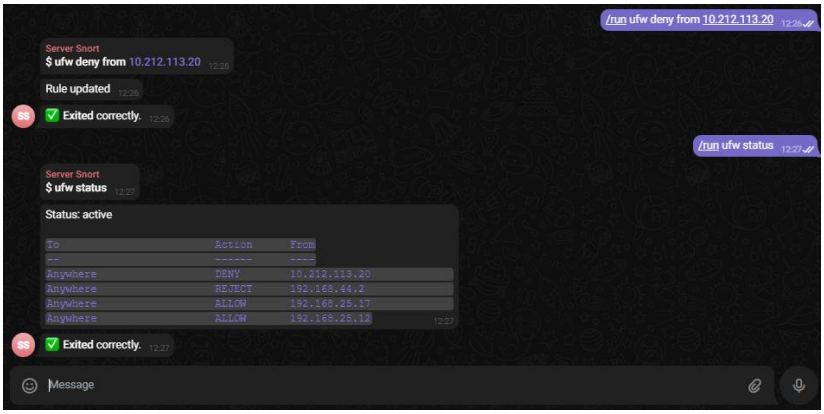
server snort terkait penyerangan yang telah terjadi pada *server* seperti yang terlihat pada Gambar 10.



Gambar 10 Notifikasi Telegram Serangan *Nmap Scanning*

Dapat dilihat bahwa pesan yang diterima memiliki selisih waktu dan jumlah yang berbeda dengan log file yang disimpan oleh sistem snort, dari 6 jenis serangan yang dapat dikirmkan hanya 2 jenis serangan yang dapat dikirimkan yaitu jenis serangan ssh incoming melalui port 22 dan nmap null scan melalui port 22. Serta dapat disimpulkan juga bahwa pengiriman notifikasi pada telegram tersebut juga memiliki delay sebanyak 9 detik dari mulainya penyerangan pada 09:30:08 dan notifikasi diterima pada 09:30:17. Bot telegram juga hanya mampu mengirimkan notifikasi setiap ± 7 detik dari berlangsungnya serangan.

Setelah dilakukan pengujian serangan – serangan dan mendapatkan pesan melalui telegram. Selanjutnya adalah pengujian untuk memblokir ip asal dari detail pesan yang telah diterima. Pemblokiran dilakukan dengan cara mengirimkan pesan *"/run ufw deny from (ip asal)"* seperti yang terlihat pada Gambar 11.

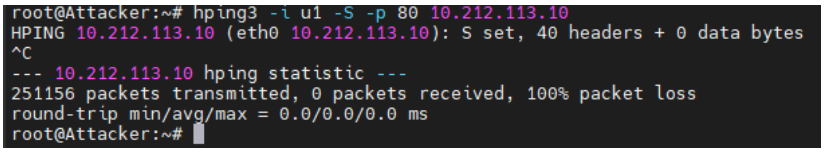


Gambar 11 Pengujian Pengendalian Serangan via Telegram

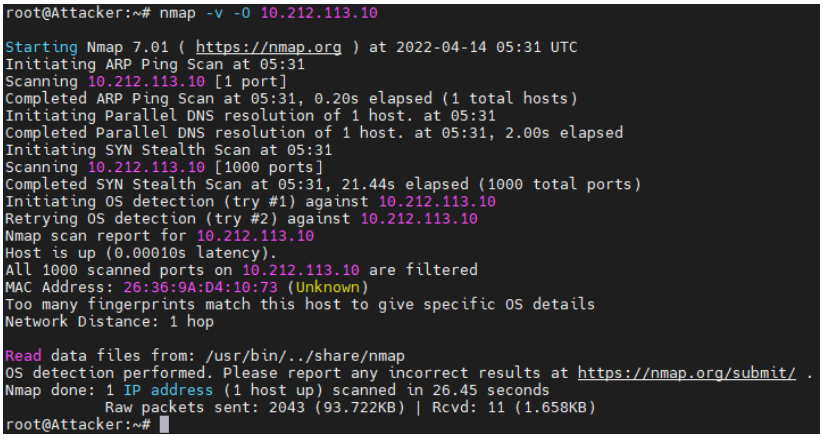
Terlihat bahwa sistem bot telegram dapat mengenali perintah untuk melakukan pemblokiran terhadap ip yang telah dikirimkan melalui pesan telegram dan mengupdate rule firewallnya. Sehingga setiap kali administrator menerima pesan tentang serangan yang terjadi, administrator dapat langsung mengirimkan pesan untuk melakukan pemblokiran pada ip asal

yang telah diterima pada pesan notifikasi serangan dari server tersebut.

Setelah berhasil mengupdate rule, dilakukan pengujian serangan kembali dari client penyerang yang telah di lakukan pemblokiran pada alamat ipnya. Gambar 12 menunjukkan bahwa ketika dilakukan penyerangan dengan menggunakan *syn-flood* terdapat informasi bahwa 100% *paket loss*, yang berarti dimana 251156 paket yang dicoba dikirimkan melalui ip penyerang tersebut gagal dan tidak dapat mengakses *server snort* lagi. Percobaan serangan *nmap scanning* juga gagal dilakukan dimana *nmap* tidak menemukan satupun port terbuka dari 1000 percobaan memindai port pada *server snort* seperti yang terlihat pada Gambar 13.



Gambar 12 Pengujian SYN-Flood Tidak Berhasil



Gambar 13 Pengujian *Nmap Scanning* Tidak Berhasil

c. Analisis Hasil Pengujian

Analisis hasil pengujian dilakukan dengan melakukan resume terhadap keseluruhan pengujian yang telah dilakukan yang berisi informasi waktu mulai dan berhentinya percobaan serangan, waktu pendeteksian oleh log snort, dan waktu diterimanya notifikasi pada aplikasi telegram.

Tabel 1 Hasil Pengujian *Nmap Scanning*

Percobaan	Serangan		Log Snort		Notif Telegram			
	Mulai	Berhenti	Mulai	Berhenti	1	2	3	4
1	11.44.20	11.44.36	11.44.27	11.44.41	11.44.28	11.44.35	11.44.42	-
2	11.47.14	11.47.30	11.47.21	11.47.35	11.47.22	11.47.31	11.47.38	-
3	11.50.47	11.51.04	11.50.54	11.51.09	11.50.55	11.51.02	11.51.09	11.51.16
4	11.54.50	11.55.07	11.54.57	11.55.11	11.55.00	11.55.07	11.55.14	-
5	11.59.53	12.00.10	12.00.00	12.00.15	12.00.02	12.00.09	12.00.16	12.00.23
Delay (Detik)			7	4,8	8,6			

Pada Tabel 1 pengujian serangan menggunakan metode *nmap scanning log snort* memiliki selisih waktu rata – rata sebanyak 7 detik dari mulainya penyerangan dan memiliki selisih 4,8 detik dari berhentinya serangan. Pada kolom notif telegram pengujian nmap scanning telegram memiliki delay rata – rata sebanyak 8,6 detik. Dan dalam setiap pengujian serangan dari mulai sampai berhentinya serangan, notif ke – 1, ke – 2 , sampai dengan seterusnya dapat disimpulkan bahwa bot telegram yang telah dibuat hanya mampu mengirimkan notifikasi setiap ± 7 detik.

Tabel 2 Hasil Pengujian *SYN-Flood*

Percobaan	Serangan		Log Snort		Notif Telegram				
	Mulai	Berhenti	Mulai	Berhenti	1	2	3	4	5
1	12.27.13	12.27.25	12.27.18	12.27.30	12.27.19	12.27.26	12.37.34	-	-
2	12.32.02	12.32.16	12.37.07	12.32.22	12.32.08	12.32.15	12.32.23	-	-
3	12.36.38	12.37.04	12.36.43	12.37.09	12.36.43	12.36.50	12.36.58	12.37.05	12.37.12
4	12.43.10	12.43.25	12.43.15	12.43.30	12.43.17	12.43.24	12.43.31	-	-
5	12.46.42	12.46.59	12.46.47	12.47.04	12.46.47	12.46.55	12.47.03	12.47.10	-
Delay (Detik)			5	5,2	5,8				

Tabel 2 menunjukkan kesimpulan hasil serangan menggunakan metode *syn-flood log snort* memiliki selisih waktu rata – rata sebanyak 5 detik dari mulainya penyerangan dan memiliki selisih 5,2 detik dari berhentinya serangan. Pada kolom notif telegram pengujian syn-flood memiliki delay rata – rata sebanyak 5,8 detik. Dan dalam setiap pengujian serangan dari mulai sampai berhentinya serangan, notif ke – 1, ke – 2 , sampai dengan seterusnya dapat disimpulkan bahwa bot telegram yang telah dibuat hanya mampu mengirimkan notifikasi setiap ± 7 detik.

4. KESIMPULAN DAN SARAN

Keamanan *server* merupakan sebuah isu krusial untuk menyediakan mekanisme layanan jaringan yang aman dan dapat diandalkan. Keterbatasan administrator dalam melakukan pemantauan *server* secara kontinu menuntut adanya sistem yang dapat menggantikan tugas tersebut. Penelitian ini mengusulkan sebuah sistem deteksi dan pengendalian serangan DoS pada *server* dengan mengkolaborasikan *Snort* dan *bot* Telegram. *Snort* bertanggungjawab dalam melakukan pemantauan aktivitas lalu lintas jaringan pada server baik itu jaringan internal maupun eksternal, sementara *bot* Telegram API mampu mengirimkan *log file* serangan yang tejadi pada *server* dan melakukan pemblokiran IP dengan cara mengirimkan pesan melalui aplikasi Telegram.

Sistem disimulasikan secara virtual dan dilakukan serangkaian pengujian mulai dari SYN-Flood, *nmap scanning* dan pemblokiran IP. Dari pengujian yang telah dilakukan, *snort* memiliki *delay* rata – rata sekitar 5 – 7 detik dari waktu serangan yang dilakukan, sedangkan untuk Telegram memiliki delay rata – rata sekitar 5,8 – 8,6 detik dari waktu serangan dilakukan, dan bot Telegram juga hanya mampu mengirimkan notifikasi setiap ± 7 detik. Nilai *delay* muncul karena keterbatasan perangkat yang digunakan sebagai simulasi (512MB, *singlecore*). Keberlanjutan penelitian dapat diimplementasikan pada lingkungan yang sebenarnya, serta upaya pembuatan *bot* Telegram yang lebih *responsive.*

DAFTAR PUSTAKA

[1] F. S. Mukti and R. M. Sukmawan, "Integration of Low Interaction Honeypot and ELK Stack as Attack Detection Systems on Servers," *Jurnal Penelitian Pos & Informatika*, vol. 11, no. 1, pp. 19–28, 2021, doi: https://doi.org/10.17933/jppi.v11i1.336.

[2] D. Panji Agustino, Y. Priyoatmojo, and N. Wayan Wiwin Safitri, "Implementasi Honeypot Sebagai Pendeteksi Serangan dan Melindungi Layanan Cloud Computing," in *Konferensi Nasional Sistem dan Informatika*, 2017, pp. 196–201.

[3] M. R. I. Wahyudi and I. R. Widiasari, "Penerapan SMS Alert Menggunakan Snort Pada Ubuntu Server 16.04 Untuk Monitoring Jaringan," Universitas Kristen Satya Wacana, Salatiga, 2019.

[4] Y. P. Atmojo, "Bot Alert Snort dengan Telegram Bot API pada Intrusion Detection System: Studi Kasus IDS pada Server Web," in *Seminar Nasional Sistem Informasi dan Teknologi Informasi*, 2018, vol. 12, pp. 176–180. [Online]. Available: https://api.telegram.org/bot$apiToken/sendMessage

[5] J. Fahana, R. Umar, and F. Ridho, "Pemanfaatan Telegram Sebagai Notifikasi Serangan untuk Keperluan Forensik Jaringan," *QUERY: Jurnal Sistem Informasi*, vol. 1, no. 2, pp. 6–14, 2017.

[6] F. Panjaitan and R. Syafari, "Pemanfaatan Notifikasi Telegram untuk Monitoring Jaringan," *Jurnal SIMETRIS*, vol. 10, no. 2, 2019.

[7] B. Alfiansyah, D. Risqiwati, and K. Person, "Notifikasi Alert Intrusion detection System pada Bot Telegram," in *Seminar Nasional Teknologi dan Rekayasa (SENTRA)*, 2018, pp. 121–127.

[8] R. Artikel, N. Christianto, and W. Sulistyo, "Model Pemantauan Keamanan Jaringan Melalui Aplikasi Telegram Dengan Snort," *Jurnal Teknik Informatika dan Sistem Informasi*, vol. 7, no. 3, pp. 702–714, 2021, doi: 10.28932/jutisi.v7i1.4088.

[9] | Harjono and A. P. Wicaksono, "Sistem Deteksi Intrusi dengan Snort (Intrusion Detection System with Snort)," *JUITA*, vol. 1, pp. 31–34, 2014.

[10] D. Wijanarko, "Sistem Keamanan Jaringan Komputer Menggunakan Snort," *Jurnal Teknologi Informasi dan Terapan*, vol. 2, no. 1, pp. 171–176, 2015.

[11] B. Wijaya and A. Pratama, "Deteksi Penyusupan Pada Server Menggunakan Metode Intrusion Detection System (IDS) Berbasis Snort," *Sistem Informasi dan Komputer)*, vol. 9, no. 1, pp. 97–101, 2020, doi: 10.32736/sisfokom.v9.i1.770.

[12] P. Pratama Putra, "Pengembangan Sistem Keamanan Jaringan Menggunakan Rumusan Snort Rule (Hids) untuk Mendeteksi Serangan Nmap," *Sains dan Teknologi Informasi (SATIN)*, vol. 2, no. 1, 2016, [Online]. Available: http://jurnal.stmik-amik-riau.ac.id

[13] S. Khadafi *et al.*, "SISTEM KEAMANAN OPEN CLOUD COMPUTING MENGGUNAKAN IDS (INTRUSION DETECTION SYSTEM) DAN IPS (INTRUSION PREVENTION SYSTEM)," *Jurnal IPTEK*, vol. 21, no. 2, pp. 67–76, 2017.

[14] B. M. Susanto and A. T. Guritno, "Implementasi Snort IDS Menggunakan Android sebagai Media Notifikasi," in *Seminar Nasional Teknologi Informasi dan Komunikasi*, 2017, vol. 164, pp. 203–212.

[15] D. T. Atmaja, E. Budhy Prasetya, and P. E. Kresnha, "Notifikasi Adanya Serangan pada Jaringan Komputer Dengan Mengirim Pesan Melalui Aplikasi Telegram dan Kontrol Server," in *Seminar Nasional Sains dan Teknologi*, 2018, vol. 17, pp. 1–8.