Metode Pertahanan Web Server Terhadap *Distributed Slow HTTP DoS Attack*

Molavi Arman
AMIK MDP; Jalan Rajawali 14, Palembang - Sumatera Selatan, 30113
Manajemen Informatika, AMIK MDP, Palembang
e-mail: molavi.arman@mdp.ac.id

Abstrak

Meningkatnya kecepatan internet selalu diikuti meningkatnya pula gangguan keamanan dalam jaringan komputer.Hal ini cukup mengganggu layanan yang terhubung ke jaringan LAN maupun internet. Gangguan keamanan ini dikenal dengan DDoS (Distributed Denial of Service). Serangan DoS slow HTTP adalah salah satu metode serangan DoS yang menargetkan server HTTP. Metode ini menghambat layanan dengan membanjiri sehingga menimbulkan kumpulan koneksi dengan permintaan yang lambat dan banyak menuju web server. Diketahui bahwa serangan slow HTTP DoS oleh satu penyerang dapat dicegah secara efektif dengan membatasi jumlah koneksi untuk setiap alamat IP. Di sisi lain, juga diketahui bahwa sulit untuk bertahan dari serangan slow HTTP DoS dari beberapa penyerang. Ancaman serangan DDoS semakin serius, jadi membutuhkan metode pertahanan yang efektif terhadap serangan DoS slow HTTP terdistribusi. Ketika web server berada di jaringan publik atau internet, maka melindungi komputer dan keamanan jaringan adalah masalah penting. Setelah mengidentifikasi dan menganalisis cara kerja serangan slow HTTP, serta deteksi serangannya, tulisan ini menjelaskan sistem kerja, cara mendeteksi, dan cara mempertahankan diri terhadap serangan slow HTTP.

Kata kunci: DDoS, DoS, serangan Slow HTTP, Web Server

Abstract

The increase in internet speed is always followed by the increase in security disturbances in computer networks. This is quite disturbing to the services connected to the LAN network or the internet. This security disturbance is known as DDoS (Distributed Denial of Service). Slow HTTP DoS attack is one of the DoS attack method that targets HTTP servers. This method inhibits service by flooding, causing a collection of connections with slow and many requests to the web server. It is known that a slow HTTP DoS attack by one attacker can be effectively prevented by limiting the number of connections for each IP address. On the other hand, it is also known that it is difficult to survive a slow HTTP DoS attack from some attackers. The threat of DDoS attacks is getting more serious, so it requires an effective defense method against distributed slow HTTP DoS attacks. When a web server is on a public network or the internet, protecting your computer and network security is an important issue. After identifying and analyzing the workings of slow HTTP attacks, as well as detection of attacks, this paper explains the working system, how to detect, and how to defend against slow HTTP attacks.

Keywords : *DDoS, DoS, serangan Slow HTTP, Web Server*

1. PENDAHULUAN

Terus meningkatnya kecepatan internet dari waktu kewaktu selalu diikuti meningkatnya pula serangan *cybercriminals* dari tahun ketahuan. Tahun 2018 yang lalu jumlah serangan DDoS terus menurun, membuat para ahli Kaspersky Lab berasumsi bahwa penjahat *cyber* yang telah melakukan serangan DDoS untuk keuntungan finansial telah mengalihkan perhatian mereka ke sumber pendapatan lain (seperti penambangan *crypto*). Namun, statistik untuk Q1 2019 bertentangan dengan tren ini dan menunjukkan bahwa jumlah serangan DDoS yang diblokir oleh Kaspersky DDoS Protection telah tumbuh dengan mengejutkan 84%, jika dibandingkan dengan Q4 2018. [1].

Serangan *Denial of Service* umumnya dilakukan membanjiri *server* atau *host* sehingga *host* korban kehabisan sumber daya (memori, CPU, lalu lintas). Kondisi ini membuatnya tidak dapat melayani pengguna lain. *Flooding*atau membanjiri sulit diatasi, tidak cukup hanya dengan me-*reboot*, seperti serangan lainnya. Ada beberapa varian serangan DoS, tidak kurang dari 35 varian serangan ini. Setiap varian memiliki perbedaan karakteristik dalam hal serangannya,tetapi mereka memiliki hal yang efek yang sama, menimbulkan penolakan layanan. [2]

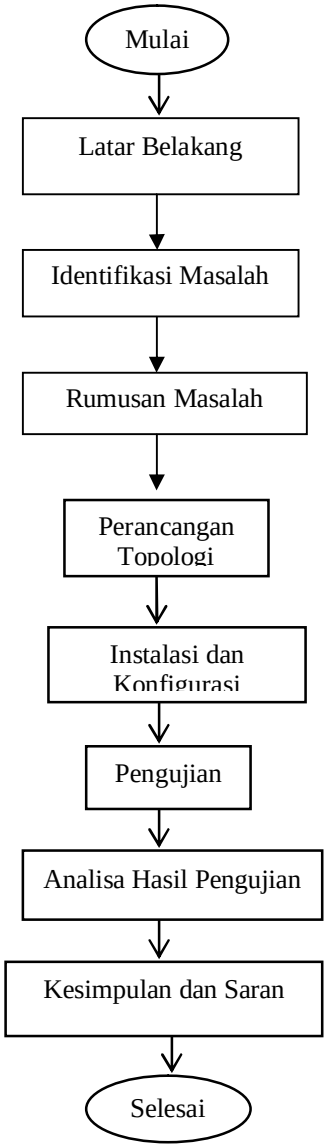
Perangkat *Internet of Things* (*IoT*) yang rentan juga membawa ancaman.*IoT* adalah sistem yang menghubungkan *smart devices* ke *Internet*. Perangkat IoT cenderung memiliki karakteristik berikut [3]: Selalu aktif, *bandwidth* cukup tinggi, kata sandi lemah, standar tertinggal, siklus pembaruan panjang. Karakteristik ini membuat perangkat IoT yang lemah mudah dibajak dan disalahgunakan untuk meluncurkan serangan DDoS.

Serangan HTTP DoS yang lambat adalah salah satu metode serangan DoS yang menargetkan *server* HTTP. Metode ini menghambat layanan dengan menjenuhkan kumpulan koneksi dengan permintaan yang lambat dan banyak. Laporan bahwa serangan DoS "*slow read*" semacam serangan HTTP DoS dari hanya satu penyerang dapat dicegah secara efektif dengan membatasi jumlah koneksi untuk setiap alamat IP, [4] disisi lain juga bahwa sulit untuk bertahan terhadap serangan *slow*HTTP DoS dari beberapa penyerang (*Distributed Slow HTTP DoS Attack*).

2. METODE PENELITIAN

2.1 Framework Penelitian

Metodologi penelitian ini melalui beberapa tahapan yang direpresentasikan dalam suatu kerangka kerja penelitian/*framework*. Kerangka kerja penelitian secara sistematis menjelaskan langkah penelitian yang meliputi perancangan,pengujian,dan proses analisis.

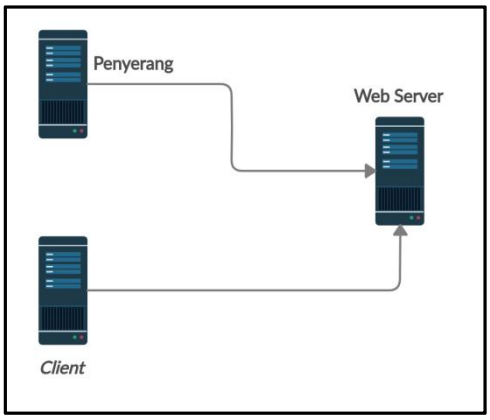


Gambar 1. Diagram Alir *Framework* Penelitian

Tahapan awal penelitian adalah merancang, membangun topologi kemudian melakukan instalasi dan konfigurasi *environment* yang dibutuhkan dalam membangun *web server*.

Penjelasan diagram alir framework adalah sebagai berikut:

a. Menjelaskan latar belakang dengan menngumpulkan penelitian terdahulu, sehingga menjadi landasan dalam penentuan tema.
b. Melakukan indetifikasi masalah yang terjadi dalam menghadapi serangan DDoS.
c. Perancangan topologi. Topologi yang dirancang untuk pengujian dibuat sedemikian rupa sehingga memudahkan dalam pengujian terhadap serangan slowloris atau slow HTTP.



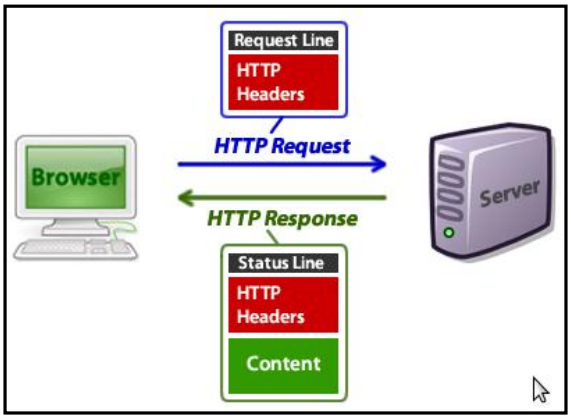
Gambar 2. Topologi Skenario Pengujian

a. Instalasi dan konfigurasi server menggunakan sistem operasi Debian 64 bit dan menggunakan memori sebesar 2GB, dengan kecepatan processor 3.3GHz.
b. Melakukan pengujian dengan menggunakan *tools*slowloris.
c. Pengambilan data diperoleh dari *ouput* pengujian yang dihasilkan *apachectl*.
d. Analisa pengujian memisahkan hasil sebelum konfigurasi dan setelah konfigurasi supaya menunjukkan adanya perbedaan terhadap *web server*yang diserang menggunakan slowloris dan setelah dilindungi.
e. Kesimpulan dari percobaan bahwa *web server* menunjukkan respon perubahan setelah dilakukan konfigurasi.

2.2 *HTTP*

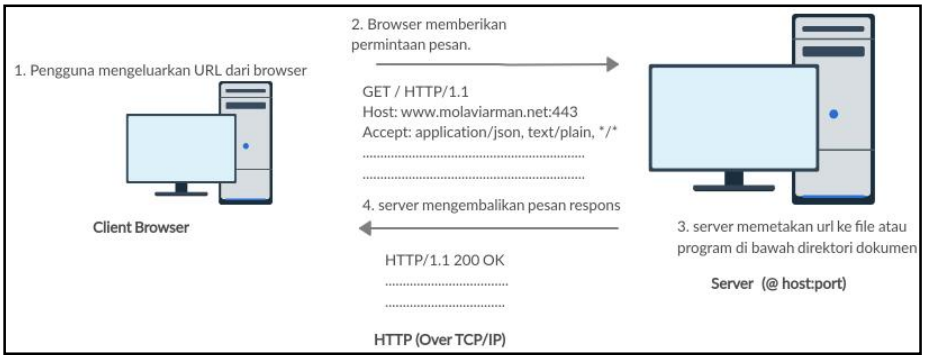
Salah satu protokol aplikasi paling populer yang digunakan di *Internet* adalah HTTP. HTTP adalah singkatan dari "*Hypertext Transfer Protocol*." HTTP adalah protokol aplikasi yang berjalan di atas protokol TCP / IP. *Web* menggunakan protokol ini, ketika membuka halaman *web*, *browser* mungkin telah mengirim lebih dari 40 permintaan HTTP dan menerima tanggapan HTTP untuk masing-masing. *Header* HTTP adalah bagian inti dari permintaan dan respon HTTP, dan proses membawa informasi tentang *browser* klien, halaman yang diminta, *server*, dan lainnya.

Seperti yang diilustrasikan dalam gambar 3, klien HTTP mengirim pesan permintaan ke *server* HTTP.*Server*, pada gilirannya, mengembalikan pesan respon. [5].



Gambar 3. HTTP *Request* dan HTTP *Response*

Program di sisi klien, yang disebut *'browser'*akan melakukan permintaan HTTP ke *server*. *Browserweb* adalah klien HTTP. Mesin *server Web* apa pun berisi *file* halaman *web* (teks, gambar grafik, suara, video, dan *file* multimedia lainnya) dan juga *daemon* HTTP, program yang dirancang untuk menunggu permintaan HTTP dan menanganinya. Klien perlu mengetik *Uniform Resource Locator* yang benar (URL) alamat dalam program *browser* atau mengklik tautan *hypertext* untuk mendapatkan halaman atau *file web*, misalnya., https://www.molaviarman.net/index.php. Kemudian browser mengonversi URL menjadi pesan permintaan dan mengirimkannya ke *server* HTTP. Server daemon HTTP menerima dan menginterpretasikan pesan permintaan, dan mengembalikan *file* yang disamakan atau *file* yang terkait dengan permintaan. Proses ini diilustrasikan dalam gambar 4 di bawah ini:



Gambar 4. Proses Komunikasi Antara *Client* dan *Web Server*

Seperti yang disebutkan di atas, ketika klien memasukkan URL di kotak alamat *browser*, *browser* menerjemahkan URL menjadi pesan permintaan HTTP dan mengirimkannya ke *server*. Misalnya, *browser* menerjemahkan URL https://www.molaviarman.net/index.php menjadi pesan permintaan, seperti yang ditunjukkan gambar 5 di bawah ini:

```
GET / HTTP/1.1
Host: www.molaviarman.net:443
Accept: application/json, text/plain, */*
Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.
User-Agent: Mozilla/5.0 (X11; Linux x86_64)
```

Gambar 5. Pesan Permintaan HTTP GET dari Terjemahan URL

Ketika pesan permintaan ini diterima dan ditafsirkan oleh *server*, *server* akan melakukan salah satu dari tiga tindakan:

- Server, *server* akan melakukan salah satu dari tiga tindakan: *Server* mencari *file* di bawah dokumen *server* direktori dan mengembalikan *file* yang diminta.
- *Server* menjalankan program yang diminta dan mengembalikan *output* program ke klien.
- *Server* mengembalikan pesan kesalahan, karena permintaan tidak bisa dipenuhi.

Contoh pesan respons HTTP adalah seperti yang ditunjukkan gambar 6 di bawah ini:

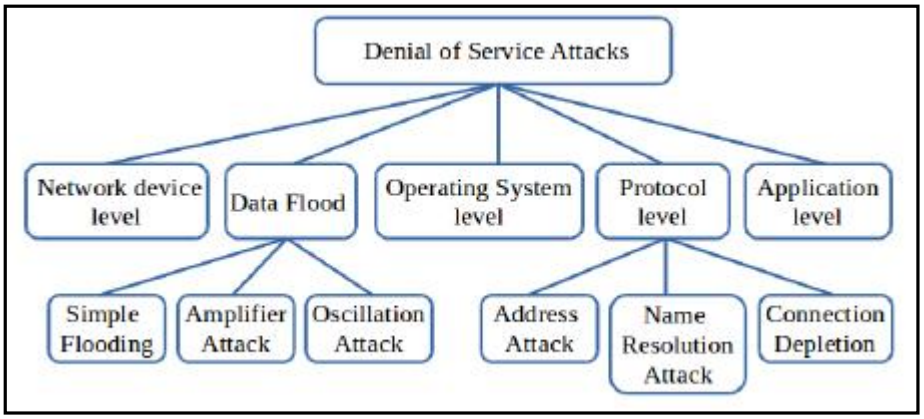
```
HTTP/1.1 200
cache-control: public
cf-cache-status: DYNAMIC
cf-h2-pushed: </wp-content/cache/minify/3a712.js>
cf-ray: 55cbec3bf887e48f-CGK
content-encoding: br
content-type: text/html; charset=UTF-8
date: Wed, 29 Jan 2020 14:33:43 GMT
expect-ct: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
last-modified: Wed, 29 Jan 2020 14:01:28 GMT
link: </wp-content/cache/minify/3a712.js>; rel=preload; as=script
pragma: public
referrer-policy:
server: cloudflare
status: 200
strict-transport-security: max-age=63072000; includeSubdomains;
strict-transport-security: max-age=31536000
vary: Accept-Encoding,Cookie
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-xss-protection: 1; mode=block
```

Gambar 6. Pesan Respon HTTP

Program *browser* menerima, menafsirkan, dan menampilkan isi pesan respon di jendela browser sesuai dengan tipe konten. Contoh di atas, konten - jenisnya adalah teks / html. Ada banyak jenis konten, seperti "teks / teks", "teks / html", "*audio* / mpeg", "*video* / mpeg", "*image* / gif", "*image* / png", "*image* / jpeg", "*application* / pdf", dan lain-lain.

2.3 Denial of Service

Serangan DoS adalah ancaman keamanan di mana penyerang mengirimkan sejumlah besar permintaan palsu ke *host* atau *server*, sehingga *host* target menolak akses dari pengguna yang berwenang sehinga layanan dari *host* menjadi tidak tersedia, oleh karena itu serangan mengganggu ketersediaan sistem. Serangan *denial of service* (DoS) yang bertujuan untuk pengguna yang sah dalam mengakases internet dan menjadi tantangan yang cukup bagi keamanan jaringan Jika serangan penolakan layanan diluncurkan dari beberapa komputer, sering disebut serangan *Distributed Denial of Service* (DDoS). Serangan DoS yang paling umum digunakan sekarang adalah serangan DDoS di mana sejumlah besar komputer mengirim ribuan permintaan ke sistem yang sedang diserang. Serangan DDoS terjadi ketika beberapa *host* terinfeksi dengan *malware* yang memungkinkan *host* diambil alih oleh penyerang; kemudian program penyerang memerintahkan mereka untuk mengakses target situs *web*. Biasanya, *host* yang menjadi target serangan DoS adalah sistem berbasis *web* atau *server web*. Secara umum, program *server web*, seperti *Apache*, IIS memiliki kemampuan untuk menangani atau menerima banyak koneksi dari pengguna. Penyerang mendapat keuntungan dari hal tersebut. [6] [7].



Gambar 7. Klasifikasi Serangan DoS

2.4 Slow HTTP DoS Attack

Slow HTTP *Denial of Service* (DoS) adalah serangan DoS lapisan aplikasi di mana sejumlah besar permintaan HTTP tidak lengkap dikirim. Ini adalah layer 7 DoS. Serangan aplikasi DoS adalah kelas yang baru dalam serangan DoS yang mengeksploitasi kelemahan dalam desain aplikasi atau implementasinya. Serangan ini lebih sulit dilacak dari pada serangan DoS klasik karena, sebagai berikut:

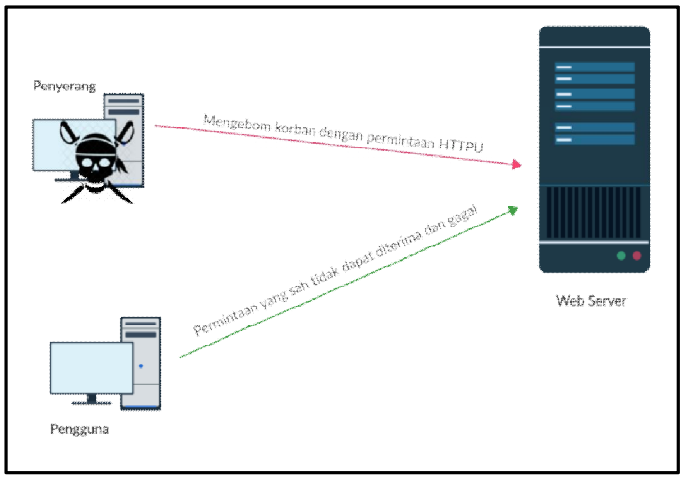
1. Serangan ini tidak memakan banyak *bandwidth*.
2. Target ini untuk menciptakan kemacetan dan sumber daya pembatasan dalam aplikasi dengan berfokus pada tautan terlemah dalam aplikasi.
3. Serangan ini biasanya menggunakan https sebagai transportasi mereka untuk menyembunyikan asal mereka yang sebenarnya.

Serangan HTTP lambat terutama dari tiga jenis [8] sebagai berikut:

1. *Slow Headers* (Slowloris): Dalam serangan *Slow Header*, seorang penyerang meluncurkan aksinya dengan bantuan alat yang disebut Slowloris atau sejenisnya. Alat ini membuka koneksi, kemudian mengirim *header* HTTP, menambah tetapi tidak pernah menyelesaikan permintaan. Ribuan koneksi HTTP *POST* dibuat dan mengirimkan HTTP *Header* dengan sangat lambat untuk memaksa *serverweb* target untuk menjaga koneksi tetap terbuka. Koneksi ini akan tetap hidup, tidak terputus dari *server* target. Slowloris akan mengambil semua sumber daya dari *server web* target, sehingga memblokir permintaan dari klien yang sah atau klien yang ingin mengakses *server web* tersebut.
2. *Slow Body* (R-U-*Dead-Yet*): Serangan *Slow Body* bekerja seperti *Slow Header*. Penyerang dengan bantuan alat yang disebut R-U-*Dead-Yet* atau sejenisnya mengirim *POST Body* yang tidak akan berakhir. Tahap serangan dimulai dengan membuat koneksi TCP awal ke *server web* target. Kemudian mengirimkan *headerPOST* HTTP terlebih dahulu seperti koneksi normal. *Header* berisi informasi ukuran tubuh paket data yang akan dikirim berikutnya. Penyerang mengirim isi pesan dengan kecepatan sangat rendah. Tetapi koneksi tetap hidup, membuat *server web* korban menunggu cukup lama. Koneksi baru dan serupa dibuat dalam jumlah besar, menggunakan semua sumber daya *server* dan membuat koneksi yang sah menjadi tidak mungkin mengakses.
3. *Slow Read*: Dalam Serangan *Slow Read*, penyerang mengirim paket TCP-SYN yang *valid* untuk membuka koneksi dengan *server* target. Kemudian sesi yang *valid* dibuat di antara mereka. Selanjutnya, ia mulai meminta dokumen dari *server* target. Setelah unduhan dimulai, *host* penyerang mulai memperlambat pembacaan paket yang diterima. Kondisi ini akan berlanjut dan mengambil semua sumber daya dari *server* target. *Slow Read Attacks* selalu menggunakan paket *non-spoofed* dalam rangka untuk menahan sesi terbuka untuk jangka waktu yang lama.

2.5 Bagaimana Slow Read Attack Bekerja

Seperti dijelaskan pada bagian di atas, serangan dilakukan dengan bantuan skrip program, yang mampu mengirimkan permintaan sebagian dari paket data, menjaga beberapa koneksi ke *server web* korban tetap terbuka. Secara berkala, ia mengirim *header* HTTP berikutnya, tetapi sengaja tidak pernah selesai. Itu memicu *server web* korban untuk menyediakan semua sumber dayanya bagi penyerang, yang akhirnya menyangkal koneksi dari pengguna yang sah. Seorang penyerang tidak membutuhkan *bandwidth* besar untuk menjatuhkan *server web* korban, tetapi hanya membuat sejumlah besar koneksi [9]. Gambar 8 menggambarkan serangan.



Gambar 8. Ilustrasi Serangan Slow HTTP

Pesan 'permintaan' dari klien ke *server* mencakup, dalam baris pertama pesan itu, metode yang akan diterapkan pada sumber daya, pengidentifikasi sumber daya, dan versi protokol yang digunakan. Protokol HTTP mendefinisikan seperangkat metode permintaan. Metode yang digunakan adalah *HEAD, GET, POST, PUT, DELETE, TRACE, CONNECT* dan *OPTION*. Kami dapat melakukan analisis permintaan HTTP *GET* dari klien ke *serveweb server*. Analisis ini akan membantu dalam penjelasan lebih lanjut tentang permintaan GET HTTP. Alat, seperti *Firebug*, *Live Header* HTTP Langsung diperlukan untuk membantu analisis [11]. Ekstrak melengkapi permintaan GET HTTP seperti yang ditunjukkan gambar 9 di bawah ini:

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 [CRLF]
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3 [CRLF]
Accept-Encoding: gzip,deflate,sdch [CRLF]
Accept-Language: en-US,en;q=0.8 [CRLF]
Connection: keep-alive [CRLF]
Host: localhost:8080 [CRLF]
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 [CRLF] [CRLF]
```

Gambar 9. Contoh *Header* HTTP *request*

Contoh di atas adalah *Header* GET normal. Setiap baris pesan berakhir dengan CRLF karakter. CRLF adalah kependekan dari CR (*Carriage Return*) dan LF (*Line Feed*). CRLF adalah karakter yang tidak dapat dicetak. Pesan permintaan berakhir dengan baris kosong. Ada dua karakter CRLF di baris bawah. Mereka bersama digunakan untuk menunjukkan suatu baris kosong. [CRLF] di akhir pesan permintaan menarik perhatian penyerang.

Dalam serangan *slow* HTTP, garis kosong tidak akan pernah ada. Penyerang sengaja tidak mengirim karakter CRLF, di akhir permintaan. Pesan permintaan sebagai berikut ini akan menghasilkan serangan *slow* HTTP, karena keberadaan *tag* CRLF tunggal di akhir menunjukkan *header* tidak lengkap, dan *server* perlu menunggu *header* lengkap. Sampel *header* yang tidak lengkap ditunjukkan pada gambar 10 di bawah ini:

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 [CRLF]
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3 [CRLF]
Accept-Encoding: gzip,deflate,sdch [CRLF]
Accept-Language: en-US,en;q=0.8 [CRLF]
Connection: keep-alive [CRLF]
Host: localhost:8080 [CRLF]
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 [CRLF]
```

Gambar 10. *Header* Permintaan HTTP Tidak Lengkap Dengan *Slow* HTTP *Attack*

2.6 Identifikasi dan Pengujian Serangan Slow HTTP

Efek dari serangan *Slow* HTTP adalah bahwa semua klien tidak dapat terhubung ke *server web*. Situs tidak mau memuat dan klien tidak akan pernah bisa melihat konten halaman *web*. Jika *server web* diserang, akan melihat banyak koneksi pada *port* 80 dari IP sumber. Perintah *netstat* dapat menampilkan daftar koneksi seperti gambar 11 berikut:

```
#   perl slowloris.pl -dns 192.168.1.90
Welcome to Slowloris - the low bandwidth, yet greedy and poisonous HTTP client by Laera Loris
Defaulting to port 80.
Defaulting to a 5 second tcp connection timeout.
Defaulting to a 100 second re-try timeout.
Defaulting to 1000 connections.
Multithreading enabled.
Connecting to 192.168.1.90:80 every 100 seconds with 1000 sockets:
                    Building sockets.
                    Building sockets.
                    Building sockets.
                    Sending data.
Current stats:      Slowloris has now sent 383 packets successfully.
This thread now sleeping for 100 seconds...
                    Sending data.
Current stats:      Slowloris has now sent 589 packets successfully.
This thread now sleeping for 100 seconds...
                    Building sockets.

# netstat -nalt | grep :80
tcp6       0      0 192.168.1.90:80         192.168.1.91:39354      ESTABLISHED
tcp6       0      0 192.168.1.90:80         192.168.1.91:39390      ESTABLISHED
tcp6       0      0 192.168.1.90:80         192.168.1.91:39394      ESTABLISHED
tcp6       0      0 192.168.1.90:80         192.168.1.91:39174      FIN_WAIT2
tcp6       0      0 192.168.1.90:80         192.168.1.91:39220      ESTABLISHED
tcp6       0      0 192.168.1.90:80         192.168.1.91:39202      FIN_WAIT2
tcp6       0      0 192.168.1.90:80         192.168.1.91:39188      FIN_WAIT2
tcp6       0      0 192.168.1.90:80         192.168.1.91:39148      FIN_WAIT2
tcp6       0      0 192.168.1.90:80         192.168.1.91:39298      ESTABLISHED
tcp6       0      0 192.168.1.90:80         192.168.1.91:39154      FIN_WAIT2
tcp6       0      0 192.168.1.90:80         192.168.1.91:39326      ESTABLISHED
tcp6       0      0 192.168.1.90:80         192.168.1.91:39362      ESTABLISHED
tcp6       0      0 192.168.1.90:80         192.168.1.91:39360      ESTABLISHED
tcp6       0      0 192.168.1.90:80         192.168.1.91:39184      FIN_WAIT2
tcp6       0      0 192.168.1.90:80         192.168.1.91:39438      ESTABLISHED
```

Gambar 11. Pengujian dan Serangan Koneksi *Slow* HTTP pada *Web Server*

Pada *log server* akan menunjukkan jumlah koneksi. Contoh pada gambar 12 di *server Apache*, statusnya tampak seperti berikut:

```
# apachectl status
CPU Usage: u4.2 s.23 cu.26 cs.35 - 1.28% CPU load
  .242 requests/sec - 140 B/second - 580 B/request - .111111 ms/request
  150 requests currently being processed, 0 idle workers
```

Gambar 12. Status pada *Web Server* Apache pada Saat Serangan

Informasi di atas menunjukkan penggunaan CPU yang sangat rendah, banyak proses pada apache, sangat sedikit permintaan baru, penyerang bekerja dengan membuat banyak permintaan dan lebih banyak lagi hingga mencapai batas *MaxClients*apache. Jika kita melihat pada gambar 13 *log*apache, itu akan seperti ini:

```
# cat /var/log/apache2/error.log
[mpm_prefork:error] [pid 1431] AH00161: server reached MaxRequestWorkers
setting, consider raising the MaxRequestWorkers setting
```

Gambar 13. *Log Error* pada Apache

Perintah ini akan menyaring semua IP yang terhubung ke *server*, agar menghitung setiap kejadian yang sering terjadi. *Output*nya seperti gambar 14:

```
# netstat -ntu -4 -6 | awk '/^tcp/{ print $5 }' | sed -r 's/:[0-9]+$//' | sort | uniq -c | sort –n
  1   192.168.1.3
280  192.168.1.91
```

Gambar 14. Perintah Memfilter IP *Address* yang Banyak Mengakses *Web Server*

Setelah mengetahui IP penyerang, pemblokiran dapat dilakukan pada alamat IP. Pada sistem operasiLinux, tersedia aplikasi IPTABLE. Perintah berikut dapat digunakan terilihat pada gambar 15 :

```
#  iptables -A INPUT -i eth0 -s 192.168.1.91  -j DROP
```

Gambar 15. Firewall *Iptables* untuk Pemblokiran

Arti dari baris perintah di atas, jika ada koneksi masuk melalui antarmuka eth 0, dari IP sumber 192.168.1.91 lalu dilakukan (*DROP*). Akan tetapi hal seperti ini cukup merepotkan jika penyerang lebih banyak.

2.7 Penelitian Terkait

Risiko ancaman keamanan komputer oleh serangan *slow* HTTP telah memicu ahli keamanan jaringan komputer untuk mengembangkan teknik pertahanan. Banyak metode dan teknik pertahanan yang berbeda tersedia untuk *server web*. Metode-metode ini bervariasi tergantung pada objek yang dilindungi dan seperangkat aturan dalam operasi. Peneliti dari akademisi juga telah melakukan banyak penelitian untuk mendeteksi dan menangani serangan

slow HTTP. Sejumlah besar literatur tersedia dalam serangan ini. Berikut rangkuman model atau metode yang berfokus pada penanganan serangan *slow* HTTP.

Referensi [6] sistem deteksi yang diusulkan adalah *anomaly* Sistem deteksi yang mengukur *Hellinger Distance* antara dua distribusi probabilitas yang dihasilkan dalam fase pelatihan dan pengujian. Pengujian sistem deteksi yang diusulkan dengan mengumpulkan lalu lintas HTTP yang disimulasikan di LAN dan *Internet*. Hasil menunjukkan bahwa sistem yang diusulkan mendeteksi serangan *Slow Header* dan *Slow Message Body* dengan akurasi tinggi.

Referensi [12] menganalisis serangan DoS *slow read*. Hasil serangan yang efisien dapat direalisasikan ketika *bandwidth* lebih dari 500 Kbps. Juga, peneliti menemukan pengaturan *server web* yang aman terhadap serangan DoS *Slow Read*.

Referensi [13] mempelajari serangan *Slow Read*, menganalisis secara rinci ancaman saat ini dan menyajikan definisi dan kategorisasi yang tepat untuk serangan tersebut. Penelitian ini bertujuan untuk memberikan kerangka kerja yang bermanfaat untuk studi bidang ini, dan untuk proposal metodologi *intrusion detection.*

Referensi [14] menganalisis efektivitas Serangan *Slow Read* oleh lingkungan jaringan virtual. Penelitian menyimpulkan bahwa menyerang oleh penyerang tunggal tidak begitu efisien. Modul *security* untuk *server web, ModSecurity* dapat membatasi panjang status keberhasilan serangan.

Referensi [15] mengusulkan dan mengevaluasi metode pertahanan terhadap serangan DoS HTTP *Slow* Terdistribusi dengan memutuskan koneksi serangan secara selektif dengan memfokuskan pada jumlah koneksi untuk setiap alamat IP dan durasi waktu, letak perbedaan dalam penelitan ini adalah belum menggunakan pemasangan modul *antiloris*pada *web server* apache dan belum mengaktifkan modul *reqtimeout*.

3. HASIL DAN PEMBAHASAN

3.1 Data Hasil Pengujian pada Serangan Slow HTTP / Slowloris

Data hasil pengujian bisa dilihat pada gambar 11. Pengujian dan serangan koneksi *Slow* HTTP pada *web server*, terlihat bahwa banyak koneksi yang tidak mampu dilayani oleh *web server* sehingga *client* yang sah tidak bisa dilayani dengan baik. Gambar 16 menunjukkan koneksi yang bertubi-tubi request ke *web server*,

```
# netstat -nalt | grep :80
tcp6       0      0 192.168.1.90:80         192.168.1.91:39354      ESTABLISHED
tcp6       0      0 192.168.1.90:80         192.168.1.91:39390      ESTABLISHED
tcp6       0      0 192.168.1.90:80         192.168.1.91:39394      ESTABLISHED
tcp6       0      0 192.168.1.90:80         192.168.1.91:39174      FIN_WAIT2
tcp6       0      0 192.168.1.90:80         192.168.1.91:39220      ESTABLISHED
tcp6       0      0 192.168.1.90:80         192.168.1.91:39202      FIN_WAIT2
tcp6       0      0 192.168.1.90:80         192.168.1.91:39188      FIN_WAIT2
tcp6       0      0 192.168.1.90:80         192.168.1.91:39148      FIN_WAIT2
tcp6       0      0 192.168.1.90:80         192.168.1.91:39298      ESTABLISHED
tcp6       0      0 192.168.1.90:80         192.168.1.91:39154      FIN_WAIT2
tcp6       0      0 192.168.1.90:80         192.168.1.91:39326      ESTABLISHED
tcp6       0      0 192.168.1.90:80         192.168.1.91:39362      ESTABLISHED
```

Gambar 16. Menunjukkan Serangan Koneksi *Slow* HTTP / *Slowloris*

3.2 Pertahanan Terhadap Serangan Slow HTTP

Pertahanan terhadap serangan *Slow* HTTP dilakukan dengan cara tertentu konfigurasi, sehingga serangan dapat dicegah atau dikurangi. Pencegahan bisa dilakukan secara spesifik

konfigurasi dengan mengaktifkan *mod_antiloris*. Dalam *web server* apache ada modul*mod_antiloris* digunakan untuk mengantisipasi serangan yang perlu diaktifkan didalam apache, berikut cara mengaktifkannya pada gambar 17.

```
# git clone https://github.com/Deltik/mod_antiloris.git
# cmake .
# make
# apxs -i -a -n antiloris mod_antiloris.so
# /etc/init.d/apache2 restart
```

Gambar 17. Instalasi mod_antiloris

Melakukan pemeriksaan apakah *mod_antiloris* telah terpasang, dengan perintah sebagai berikut :

```
# apachectl -M | grep antiloris
antiloris_module (shared)
```

Gambar 18. *mod_antiloris* Telah Aktif

Jika dilakukan percobaan kembali penyerangan terhadap *web server* ketika mod_antiloris telah aktif bisa dilihat hasilnya pada gambar 19. Terlihat dari hasil percobaan bahwa serangan *slow* HTTP mampu ditangani dengan baik.

```
CPU Usage: u0 s.05 cu.08 cs.13 - .0762% CPU load
.0469 requests/sec - 48 B/second - 1024 B/request - 3.625 ms/request
16 requests currently being processed, 4 idle workers
```

Gambar 19. Hasil Serangan Ketika *mod_antiloris* Telah Aktif

Modul mod_reqtimeout perlu ditambahkan *kendalam web server* untuk menjatuhan / menolak (DROP) koneksi yang terlalu lambat tetapi klien membuka banyak koneksi. Modul *mod_antiloris* bekerja paling baik dalam kombinasi dengan *mod_reqtimeout,* karena:

a. *mod_reqtimeout* menjatuhkan koneksi jika terlalu lambat tetapi rentan ketika klien membuka banyak koneksi dan mencegah koneksi yang sah kehabisan waktu selama periode waktu.
b. *mod_antiloris* mencegah klien dari memonopoli *slot* koneksi tetapi tidak menjatuhkan koneksi yang lambat.

Menggunakan kedua modul pada saat yang sama melindungi terhadap kedua jenis pelaku DoS. Berikut isi file konfigurasi *reqtimeout.conf* dan hasil pengujian pada gambar 18 dan gambar 19.

```
<IfModule reqtimeout_module>
    IPTotalLimit 16
    LocalIPs    127.0.0.1 ::1
    RequestReadTimeout header=20-40,minrate=500
    RequestReadTimeout body=10,minrate=500
</IfModule>
```

Gambar 20. Isi *File Reqtimeout.conf*

Pada *option IP Total Limit* bermaksud koneksi simultan maksimum dalam kondisi apa pun per alamat IP. Jika diatur ke 0, batas ini tidak berlaku. Pada *option Request Read Timeout header* = 20-40, *minrate* = 500 bermaksud tunggu maksimal 20 detik untuk *byte* pertama dari baris ditambah permintaan *header* sejak saat itu, memerlukan kecepatan data minimum 500 *byte* /s, tetapi jangan menunggu total lebih dari 40 detik. Hal yang perlu di ingat batas waktu lebih rendah mungkin masuk akal pada *host virtual non-ssl* tetapi dapat menyebabkan masalah dengan *host virtual ssl* yang diaktifkan, waktu habis ini mencakup waktu yang mungkin diperlukan *browser* untuk mengambil CRL untuk sertifikat. Jika *server* CRL tidak dapat dijangkau, mungkin diperlukan lebih dari 10 detik hingga *browser* menyerah. Pada *option Request Read Timeout body* = 10, minrate = 500 bermaksud tunggu maksimal 10 detik untuk *byte* pertama dari badan permintaan (jika ada) sejak saat itu, memerlukan kecepatan data minimum 500 *byte*/s.

```
CPU Usage: u.67 s1.61 cu.63 cs1.23 - .00525% CPU load
  .00052 requests/sec - 0 B/second - 924 B/request - 2.41463 ms/request
  14 requests currently being processed, 5 idle workers
```

Gambar 21. Hasil Percobaan Setelah *mod_slowloris* dan *mod_reqtimeout* Diaktifkan

Pada gambar 21 menunjukkan semakin baiknya *web server* menghalau serangan *slow* HTTP atau *slowloris* dengan hanya melayani pengguna yang sah dan menolak koneksi-koneksi yang hanya membebani terlihat dengan 14 *requests* selama proses berjalan.

4. KESIMPULAN

Serangan *slow* HTTP bisa dikatakan sangat merusak sumber daya dan sangat mengganggu pada layanan yang berbasis koneksi *web server*, jika tidak ditanggulangi dengan benar. Ada beberapa *options* tambahan penanganan *web server* dalam konfigurasi, setiap jenis *web server* tentu berbeda cara menanggulanginya.

Berdasarkan hasil pengujian dengan mengaktifkan *mod_antiloris* dan *mod_reqtimeout* mampu menanggulangi serangan *slow* HTTP. Akhirnya, diketahui bahwa serangan *slow* HTTP dapat dicegah dan bahkan dihilangkan, jika *serverweb* diinstal sistem keamanan yang tepat.

5. SARAN

Saran untuk penelitian kedepan perlu dilanjutkan dari sisi keamanan traffic jaringan yang terhubung ke sumber daya database.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada AMIK MDP yang telah memberikan izin melakukan percobaan pada laboratorium jaringan komputer.

DAFTAR PUSTAKA

[1] *Kaspersky.com*, 21-May-2019. [Online]. Available: https://www.kaspersky.com/about/press-releases/2019_a-ddos-storm-has-come-number-of-attacks-grows-after-long-period-of-decline. [Accessed: 29-jan-2020].

[2] S. Suroto, 2017, "*A Review of Defense Against Slow HTTP Attack*," *JOIV: International Journal on Informatics Visualization*, Vol. 1, No. 4, p. 127, Apr.

[3] R. Agarwal, "Defending *the Network from Real IoT Threats*," *Security Magazine RSS*, 28-Jul-2015. [Online]. Available: https://www.securitymagazine.com/articles/86545-defending-the-network-from-real-iot-threats. [Accessed: 30-Jan-2020].

[4] J. Park, 2015, "*Analysis of Slow Read DoS Attack and Countermeasures on Web Servers*," *International Journal of Cyber-Security and Digital Forensics*, Vol. 4, No. 2, pp. 339–353.

[5] Burak, B. Guzel, B. Guzel, B. G. B. Guzel, and B. Guzel, "*HTTP Headers for Dummies*," *Code Envato Tuts*, 02-Dec-2009. [Online]. Available: https://code.tutsplus.com/tutorials/http-headers-for-dummies--net-8039. [Accessed: 05-Feb-2020].

[6] N. Tripathi, N. Hubballi, and Y. Singh, 2016, "*How Secure are Web Servers? An Empirical Study of Slow HTTP DoS Attacks and Detection*," *2016 11th International Conference on Availability, Reliability and Security (ARES)*.

[7] D Sai Krishna et al, 2012, "*Application Denial of Service Attacks Detection using Group Testing Based Approach*". *International Journal of Computer Science & Communication Networks*, Vol 2(2), pp. 167- 171, Feb.

[8] "*How Slow HTTP Can Knock Down A Server*?," *Geeksfor Geeks*, 23-May-2017. [Online]. Available: https://www.geeksforgeeks.org/slow-http-can-knock-server/. [Accessed: 05-Feb-2020].

[9] "*Analyzing The Anatomy of a DoS Attack Using Slowloris*," *The Official Admin-Ahead Blog*, 12-Apr-2016. [Online]. Available: https://admin-ahead.com/blog/analyzing-the-anatomy-of-a-dos-attack-using-slowloris/. [Accessed: 05-Feb-2020].

[10] D. Senecal, "*The Akamai Blog Subscribe," Slow DoS on the Rise - The Akamai Blog*. [Online]. Available: https://blogs.akamai.com/2013/09/slow-dos-on-the-rise.html. [Accessed: 05-Feb-2020].

[11] S. Mendon, "*Slow DOS Attack: Why It Is Dangerous and How to Detect Using a SIEM," Cyber Security & Information Security Services*. [Online]. Available:

https://www.paladion.net/blogs/how-to-detect-slow-dos-attack-using-siem. [Accessed: 05-Feb-2020].

[12] S. Tayama and H. Tanaka, 2017, "*Analysis of Effectiveness of Slow Read DoS Attack and Influence of Communication Environment*," *2017 Fifth International Symposium on Computing and Networking (CANDAR)*, pp. 350–359

[13] E. Cambiaso, G. Papaleo, G. Chiola, and M. Aiello, 2013, "*Slow DoS Attacks: Definition and Categorisation*," *International Journal of Trust Management in Computing and Communications*, Vol. 1, No. 3/4, pp. 300–319.

[14] J. Park, 2015, "*Analysis of Slow Read DoS Attack and Countermeasures on Web Servers*," *International Journal of Cyber-Security and Digital Forensics*, Vol. 4, No. 2, pp. 339–353.

[15] T. Hirakawa, K. Ogura, B. B. Bista, and T. Takata, 2016, "*A Defense Method Against Distributed Slow HTTP DoS Attack*," *2016 19th International Conference on Network-Based Information Systems (NBiS)*, pp. 152–158.