

TUGAS AKHIR

**ANALISA PROSES KILL BOTS DALAM
MEMPERTAHANKAN DARI SERANGAN
DISTRIBUTED DENIAL OF SERVICE
DENGAN MENGGUNAKAN STRUKTUR
DATA BLOOM FILTER**



Oleh :

Ian Febrian Reza Mulia Yulianto

209115870

**JURUSAN TEKNIK INFORMATIKA
SEKOLAH TINGGI TEKNIK SURABAYA
SURABAYA
2013**

TUGAS AKHIR

**ANALISA PROSES KILL BOTS DALAM
MEMPERTAHANKAN DARI SERANGAN
DISTRIBUTED DENIAL OF SERVICE
DENGAN MENGGUNAKAN STRUKTUR
DATA BLOOM FILTER**

Diajukan Guna Memenuhi Sebagian Persyaratan

Untuk Memperoleh Gelar

Sarjana Teknik Informatika

pada

Sekolah Tinggi Teknik Surabaya

S u r a b a y a

Mengetahui/Menyetujui

Dosen Pembimbing

(Yosi Kristian, S.Kom., M.Kom.)

SURABAYA

SEPTEMBER, 2013

ABSTRAK

Distributed Denial of Service atau DDoS adalah salah satu serangan yang mengancam sekuritas dari server. Serangan tersebut mempunyai jenis yang beraneka ragam, contohnya adalah Flash Crowd. Serangan Flash Crowd adalah serangan yang mengandalkan banyaknya agent untuk mengakses server yang ingin diserang dan tidak adanya eksploitasi dari segi protokol maupun dari segi port yang ada. Flash Crowd mempunyai perilaku hampir sama seperti pengguna pada umumnya, dengan demikian penanganannya harus berdasarkan perilaku dari pengguna yang sebenarnya. Berdasarkan masalah tersebut terciptalah Kill Bots, sistem pertahanan terhadap serangan Flash Crowd. Pada tugas akhir ini, Kill Bots akan dianalisa dan diimplementasi ulang pada Apache web server dengan rupa modul Apache.

Kill Bots mempunyai dua buah status yang menandakan kondisi dari server saat itu, yaitu NORMAL dan SUSPECTED. Kill Bots memulai proses pembedaan antara pengguna yang sah dengan zombie dimulai pada kondisi server mencapai status SUSPECTED, di mana kondisi server pada saat itu mempunyai load server lebih dari 70%. Proses pembedaan atau autentikasi yang berjalan dengan menggunakan bantuan captcha dilakukan pada semua request yang baru masuk pada saat status SUSPECTED. Perilaku yang diharuskan untuk menjawab sebuah tes akan membedakan apakah itu pengguna yang sah atau tidak. Perilaku request yang berbau zombie, maka akan dimasukkan pada sebuah tabel zombie yang bersifat temporer. Ketika request yang telah dikenali sebagai zombie masuk, maka server akan langsung memberikan nilai kembalian bahwa request tersebut tidak dapat melanjutkan requestnya.

Pengujian yang dilakukan adalah pengujian performa antara tiga kondisi pertahanan dari server. Hasil dari pengujian performa akan disajikan dalam bentuk grafik garis di mana nilai yang ditunjukkan adalah nilai respond time dari server tanpa pertahanan apa pun, server dengan pertahanan mod_evasive, dan server dengan pertahanan Kill Bots.

ABSTRACT

DDoS is an attack that threaten the security of a server. It has many kind of variation, one of them is the Flash Crowd. Flash Crowd is an attack that relies on the number of agent for attacking the server and no exploitation on the protocol and port. Flash Crowd's behavior is similar to a normal user so the handling has to be based on a real user. This problem leads to the creation of Kill Bots, a defense system for the Flash Crowd. In this final project, Kill Bots will be analyzed and implemented on Apache web server using Apache module.

Kill Bots has two kind of status showing the server's condition, NORMAL and SUSPECTED. Kill Bots will start the separation of a normal user and a zombie when the server status is SUSPECTED (a condition where the server's load is more than 70%). The separation process (also known as the authentication process) that use the captcha is applied to all coming requests when the status is SUSPECTED. Behavior shown by the requests will differentiate between a real user and a zombie. Zombie users will be put in a temporary zombie table and the server will return a value result that the request can't proceed with its request.

The examination performed is a performance examination between three server conditions. The result of the examination will be shown by using the line chart where the value shown represents respond time from three kind of server : server without a defense system, server with mod_evasive defense system and server with Kill Bots defense system.

KATA PENGANTAR

Puji syukur kepada Tuhan Yang Maha Esa atas berkat yang telah diberikannya, sehingga Tugas Akhir ini dapat terselesaikan dengan baik dan tepat pada waktunya. Selama penyusunan hingga selesainya Tugas Akhir ini, penulis telah banyak menerima dorongan, bantuan, dan perhatian dari berbagai pihak. Oleh karena itu pada kesempatan ini, penulis ingin mengucapkan terima kasih kepada:

1. Kedua orang tua, yang telah memberikan dukungan dan motivasi baik berupa material maupun spiritual dalam penyelesaian Tugas Akhir ini.
2. Bapak Yosi Kristian, S.Kom., M.Kom., selaku dosen pembimbing yang banyak memberikan pengarahan, koreksi, serta nasehat kepada penulis selama penyusunan Tugas Akhir ini.
3. Semua dosen pengajar yang telah memberikan bimbingan dan pengajaran selama masa perkuliahan di Sekolah Tinggi Teknik Surabaya.
4. Semua teman yang telah senantiasa membantu dan mendukung penulis dalam mengerjakan Tugas Akhirnya hingga selesai.

Pada kesempatan ini pula, penulis juga ingin mengucapkan terima kasih kepada semua pihak yang belum disebutkan sebelumnya, atas segala bentuk bantuan dan dukungan yang telah dilakukan. Tanpa bantuan dan dukungan dari pihak-pihak tersebut, penulis tidak akan mampu menyelesaikan Tugas Akhir ini dengan baik.

Ibarat *tiada gading yang tak retak*, begitu pula pengerjaan Tugas Akhir ini tentu tidak luput dari kekurangan-kekurangan yang menyertai. Oleh karena itu penulis sangat mengharapkan kritik dan saran dari pembaca. Akhir kata, semoga Tugas Akhir ini memberikan manfaat bagi para pembaca.

Surabaya, September 2013

Penulis

DAFTAR ISI

HALAMAN JUDUL	i
HALAMAN PENGESAHAN	ii
ABSTRAK	iii
ABSTRACT	iv
KATA PENGANTAR	v
DAFTAR ISI	vi
DAFTAR GAMBAR	ix
DAFTAR SEGMENT PROGRAM	x
 BAB I PENDAHULUAN	 1
1.1 Latar Belakang	1
1.2 Tujuan	2
1.3 Ruang Lingkup	2
1.4 Langkah-Langkah Penyelesaian Tugas Akhir	6
1.5 Sistematika Pembahasan	7
 BAB II DISTRIBUTED DENIAL OF SERVICE	 9
2.1 Denial of Service	9
2.1.1 Metode Serangan	9
2.2 Arsitektur Serangan DDoS	12
2.2.1 Model Agent-Handler	12
2.2.2 Internet-Relay Chat	13
2.3 Pengelompokan Serangan DDoS	14
2.3.1 Bandwidth Depletion Attack	15
2.3.2 Resource Depletion	18
2.4 Serangan DDoS dalam Uji Coba Tugas Akhir	20
 BAB III PEMBUATAN MODUL APACHE	 22
3.1 Sejarah	23
3.2 Fitur dan Performa	24
3.3 Arsitektur Apache	26
3.3.1 Two-Phase Operation	27

3.3.2	Struktur dan Konsep Dasar	29
3.4	Modul	31
3.4.1	Programming Module Apache	32
3.5	Serangan DDoS Terhadap Apache Web Server.....	39
BAB IV	MODULE EVASIVE PADA APACHE WEB SERVER	41
4.1	Cara Kerja Mod_evasive.....	42
4.2	Konfigurasi Mod_evasive	43
4.3	Named Timestamp Tree.....	45
4.4	Pemeriksaan Request Baru.....	51
BAB V	ANALISA DESAIN SISTEM KILL BOTS	55
5.1	Pengertian Kill Bots.....	55
5.2	Sistem Kill Bots.....	56
5.3	Daftar Zombie	58
5.3.1	Bloom Filter	59
5.3.2	Hash Table	62
5.4	Autentikasi	65
5.4.1	Stages.....	65
5.4.1.1	Stage SUSPECTED: CAPTCHA-Based Authentication	67
5.4.1.2	Stage SUSPECTED: Pengguna yang Tidak Menjawab CAPTCHA	69
BAB VI	IMPLEMENTASI SISTEM KILL BOTS PADA APACHE.....	71
6.1	Hash Table	71
6.2	Bloom Filter	74
6.3	Load Server	81
6.4	Tabel Puzzle	82
6.5	Content Handler	85
6.6	Captcha	87
6.7	Token	89
6.8	Cookie	94
6.9	Query String	96

6.10	Penentuan Status Server.....	97
6.11	Pemberian Status Pada Content Handler	98
6.12	Proses Pada Saat Start-Up.....	99
BAB VII UJI COBA PERFORMA		102
7.1	JMeter	102
7.2	Uji Coba Perbandingan Performa.....	105
7.2.1	Skenario 1	106
7.2.2	Skenario 2	107
7.2.3	Skenario 3	108
7.2.4	Skenario 4	110
BAB VIII PENUTUP		112
8.1	Kesimpulan	112
8.2	Saran	114
DAFTAR PUSTAKA		115
RIWAYAT HIDUP		116

DAFTAR GAMBAR

Gambar	Halaman
1.1 Flow Diagram Kill Bots.....	3
2.1 Arsitektur Agent-Handler.....	13
2.2 Model Arsitektur IRC	14
2.3 Three-ways Handshake	18
2.4 Serangan TCP SYN	19
3.1 Data Statistik Web Server yang Aktif Agustus 2000 – Mei 2012.....	22
3.2 Logo Apache Web Server	23
3.3 Arsitektur Apache.....	26
3.4 Apache Module Lifecycle	32
4.1 Proses Mod_evasive	42
4.2 Proses Autentikasi Mod_evasive.....	43
5.1 Graphical Tes dari Google	56
5.2 Flow Diagram Kill Bots.....	57
5.3 Skema Contoh Bloom Filter.....	59
5.4 Skema Proses Bloom Filter Terhadap Tempat Penyimpanan.....	60
5.5 Contoh Proses Hash Table	63
5.6 Transisi Status Server	65
5.7 Contoh Captcha Kill Bots	67
5.8 Segmen Token.....	68
7.1 Apache JMeter.....	101
7.2 Laporan Jenis Tabel Pada JMeter.....	102
7.3 Hasil Uji Coba Skenario 1.....	105
7.4 Hasil Uji Coba Skenario 2.....	106
7.5 Hasil Uji Coba Skenario 3.....	108
7.6 Hasil Uji Coba Skenario 4.....	110

DAFTAR SEGMENT PROGRAM

Segment Program	Halaman
3.1 Struktur Data Module Apache 2.x	33
3.2 Register Hook.....	34
4.1 Konfigurasi Mod_evasive	44
4.2 Struktur Data dari Mod_evasive.....	45
4.3 Inisialisasi Named Timestamp Tree	46
4.4 Fungsi Pembuatan Node Baru.....	47
4.5 Fungsi Penambahan Node Baru	48
4.6 Fungsi Pencarian Node	49
4.7 Fungsi Untuk Menghapus NTT.....	50
4.8 Proses Menghapus Satu Node Pada NTT	51
4.9 Pemeriksaan Alamat IP Tahap 1	52
4.10 Pemeriksaan Alamat IP Tahap 2	52
6.1 Inisialisasi Hash Table Pada Apache	71
6.2 Menulis dan Mendapatkan Data Pada hash Table.....	72
6.3 RSHash.....	75
6.4 JSHash.....	76
6.5 Inisialisasi Hash Function	77
6.6 Pengisian Data ke Dalam Bloom Filter	77
6.7 Pencarian Data dari Bloom Filter	79
6.8 Inisialisasi Bloom Filter	80
6.9 Mendapatkan Load Server	81
6.10 Inisialisasi Tabel Puzzle.....	83
6.11 Content Handler Captcha	85
6.12 Content Handler Setelah Captcha.....	86
6.13 Pembuatan Respon Konten Captcha.....	87
6.14 Pembuatan Token	90
6.15 Validasi Token.....	91
6.16 Daur Ulang Token	93

6.17 Mengambil Cookie yang Dikirmkan	95
6.18 Mengambil Query String	96
6.19 Penentuan Status Server.....	97
6.20 Pemberian Status Pada Content Handler	99
6.21 Proses Inisialisasi Pada Proses Start-Up.....	99
6.22 Register Hook.....	100

BAB I

PENDAHULUAN

Dalam bab ini akan dijabarkan tentang latar belakang pengerjaan Tugas Akhir, tujuan yang diharapkan, ruang lingkup yang menjadi batasan dalam pengerjaan Tugas Akhir, metodologi yang digunakan, serta sistematika pembahasan yang akan menjelaskan isi dari setiap bab yang ada dalam buku tugas akhir ini secara ringkas.

1.1 Latar Belakang

Seiring dengan perkembangan teknologi, banyak serangan DoS (Denial of Service) yang dilancarkan oleh para profesional dengan menggunakan Botnets pada ribuan mesin. Untuk menghindari deteksi, zombie (penyerang DoS) mulai meninggalkan teknik bandwidth flooding dan beralih ke teknik serangan yang mencontoh gerak gerik web browser dengan client yang sangat banyak dan mempunyai tujuan ke arah sumber yang lebih berharga pada layer yang mempunyai tingkat lebih tinggi seperti CPU, database dan disk bandwidth.

Serangan DoS yang dibantu dengan adanya virus worm, dapat menginfeksi sampai dengan 30.000 mesin baru setiap harinya. Virus worm ini membawa sebuah mesin yang nantinya setelah diinfeksi ke korban, maka mesin tersebut akan meluncurkan serangan DoS dan serangan yang lain seraca terus menerus. Ketika teknik SYN flood gagal dijalankan, maka mereka akan meluncurkan teknik HTTP flood, mengunduh banyak gambar yang berukuran besar dari korban. Dalam contoh yang lain, zombie meluncurkan permintaan melalui mesin pencari dengan jumlah yang sangat besar, sehingga membuat server rusak.

Pemberantasan serangan DoS ini adalah sebuah tantangan, karena permintaan yang dilakukan oleh zombie mempunyai ciri-ciri yang sama dengan pengguna yang sedang meminta data. Ada perbedaan yang cukup terlihat pada zombie ini, yaitu gerak geriknya. Permintaan yang mencurigakan datang dari berbagai belahan dunia, sehingga tidak dapat difilter dari awalan IP mereka. Disamping itu juga banyak site yang tidak menggunakan password atau

otentikasi, jika pun ada, password cukup mudah dicuri. Untuk mencegah serangan, biasanya pengguna diberikan sebuah teka-teki, tetapi untuk memproses teka-teki ini membutuhkan proses yang cukup berat, sehingga kekurangan ini dapat digunakan untuk melakukan serangan yang lain. Serangan yang paling sulit untuk dideteksi adalah serangan yang bertujuan untuk mendapatkan data yang berasal dari layer yang lebih atas, seperti CPU, database, disk karena sistem operasi yang biasa tidak menyediakan monitoring sumber-sumber tersebut secara detil.

1.2 Tujuan

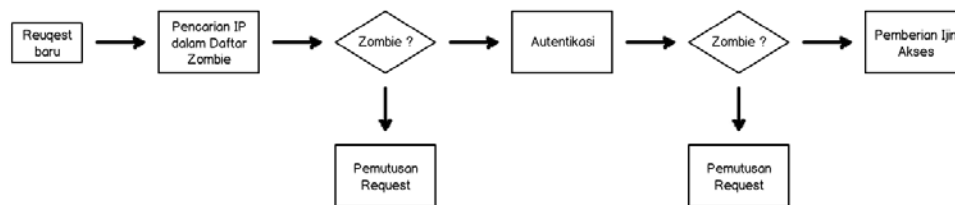
Tujuan dari tugas akhir ini adalah untuk menganalisa kerja dari sistem Kill Bots yang digunakan untuk mengoptimisasi kinerja web server dengan cara mengidentifikasi zombie dan mengurangi serangan-serangan DoS (Denial of Service) yang masuk pada web server agar dapat bekerja dengan lebih optimal.

1.3 Ruang Lingkup

Ruang lingkup yang akan dibahas pada tugas akhir ini akan dijelaskan sebagai berikut,

1. Arsitektur Sistem

Kill Bots mempunyai sistem yang hampir sama dengan sistem load balancer. Kalkulasi utama yang dilakukan adalah kalkulasi tentang load server. Pengertian dari load server itu sendiri adalah persentase dari berapa banyak service atau daemon (sinonim kata dari service yang sering digunakan dalam sistem operasi Linux) yang sedang bekerja. Load server adalah hal yang berbeda dengan load dari CPU atau yang sering disebut dengan CPU *usage*. Terkadang banyak yang mengartikan sama dengan hal tersebut. Sistem yang dipakai oleh Kill Bots adalah sebuah sistem yang cukup sederhana sehingga jika dilihat secara sepintas maka akan mudah dimengerti. Kesederhanaan dari Kill Bots mempunyai keunggulan tertentu, yaitu proses yang dipakai menjadi minimum. Proses yang minimum dan cepat itulah inti dari pengembangan web server untuk bekerja lebih efisien dan cepat dalam menangani request-request yang masuk.



Gambar 1.1
Flow Diagram Kill Bots

Gambar 1.1 menggambarkan tentang alur kerja dari sistem Kill Bots dengan permulaan proses adalah dengan pencarian pada daftar zombie di mana pencarian didasarkan oleh alamat IP dari request yang masuk. Daftar zombie merupakan gabungan dari dua buah komponen, yaitu filter yang digunakan untuk mengurangi proses pencarian yang langsung diproses dalam sebuah tempat penyimpanan, dalam tugas akhir ini filter yang akan digunakan adalah *Bloom Filter*, dan tempat penyimpanan itu sendiri yang dalam tugas akhir ini tempat penyimpanan yang dipakai adalah hash table yang telah disediakan oleh Apache. Alamat IP yang masuk dan akan dicari dalam daftar zombie tidak dengan begitu saja dicari dalam daftar zombie, tetapi sebelumnya, proses *hashing* akan dilakukan pada alamat IP tersebut dan kemudian dilakukan prosedur pencarian dalam daftar zombie tersebut. Proses hashing tersebut menandakan bahwa data-data yang ada pada daftar zombie tersebut ada data hasil proses hashing yang mulanya adalah sebuah alamat IP.

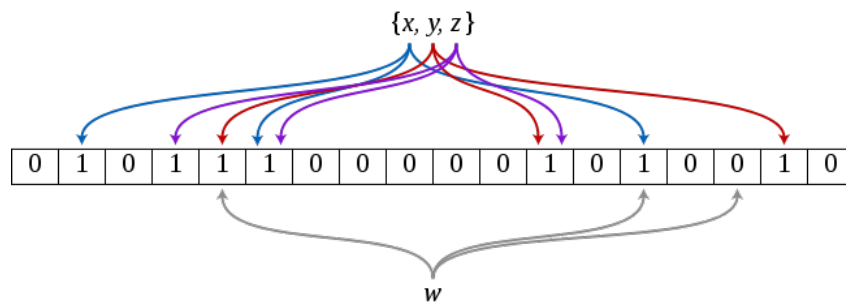
Jika alamat IP yang masuk ternyata ada dalam daftar zombie, maka server akan mengirimkan sebuah sinyal di mana request tersebut tidak dapat diteruskan ke tujuan request tersebut atau dalam bahasa singkatnya, request tersebut akan dihentikan untuk mendapatkan tujuan semulanya. Penghentian request tersebut tidak semudah memotong sebuah tali. Dalam lifecycle dari Apache, tidak adanya prosedur yang dapat memotong request yang masuk, tetapi Apache mempunyai prosedurnya sendiri, yaitu mengirimkan sinyal kepada setiap fasenya untuk tidak memproses request tersebut lebih lanjut dan terakhir pada bagian content handler, sinyal tersebut akan mengembalikan sebuah nilai pada sumber request sebuah kode status html, contohnya kode status 4xx. Kode status 4xx adalah kode status

mempunyai arti client error, di mana kesalahan ada pada client. Pada tugas akhir ini kode status untuk menghentikan request agar tidak diproses lebih lanjut adalah 403, yaitu *Forbidden*.

Pada proses selanjutnya, yaitu akan dilakukan proses autentikasi terhadap request yang masuk tersebut. Autentikasi adalah nama sebuah proses yang mana di dalam proses tersebut terdapat beberapa proses kecil yang bertujuan untuk mengidentifikasi apakah request tersebut layak untuk mendapatkan ijin untuk melakukan prosesnya secara normal atau tidak. Jika tidak, sama seperti sebelumnya, akan dilakukan prosedur untuk menutup request tersebut dengan tanpa melakukan proses lebih lanjut pada request tersebut.

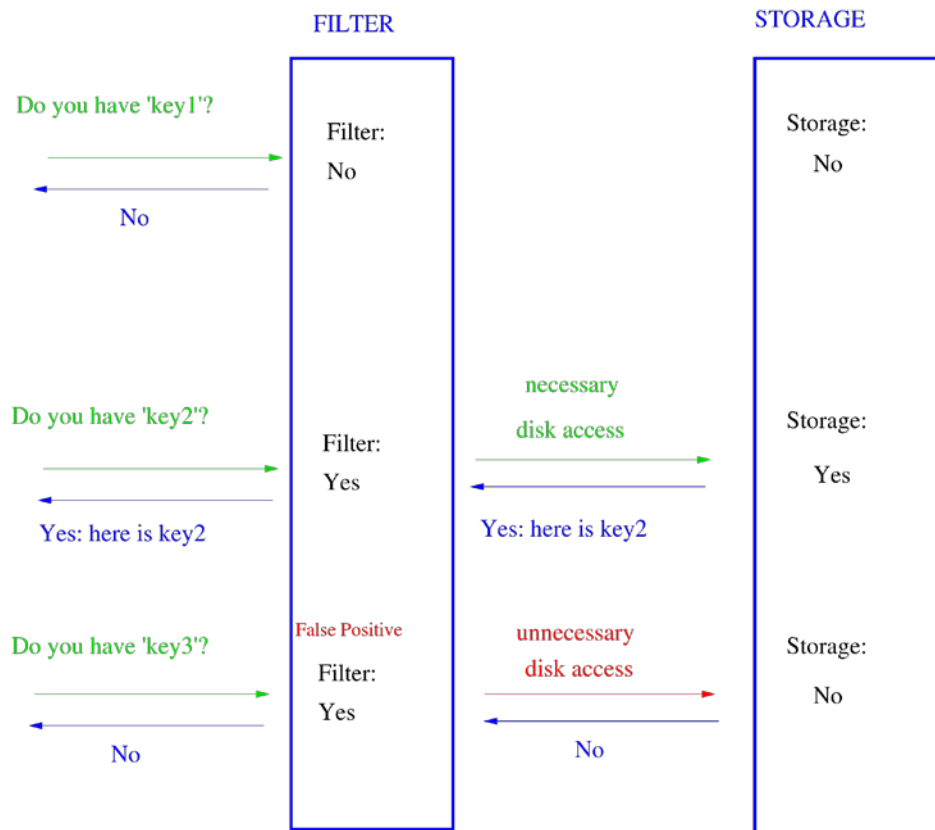
2. Penggunaan Struktur Data Bloom Filter

Bloom filter yang digunakan pada Kill Bots adalah bloom filter yang belum terjadi perubahan-perubahan yang seperti sekarang telah banyak bermuculan. Beberapa contoh bloom filter yang telah dimodifikasi adalah Bloomier Filters, Stable Bloom Filter, Scalable Bloom Filter, Attenuated Bloom Filter, dan lain-lain.



Gambar 1.2
Ilustrasi Struktur Data Bloom Filter

Gambar 1.2 adalah sebuah ilustrasi skema struktur data bloom filter di mana w adalah elemen yang akan dicari pada himpunan $\{x, y, z\}$. Ternyata elemen w tidak berada pada himpunan $\{x, y, z\}$ karena salah satu bit array dari elemen w bernilai 0. Sebuah elemen yang dicari akan bernilai false positive jika bit array yang ditunjuk mempunyai nilai 1.



Gambar 1.3
Skema Proses Kerja Bloom Filter

Pada gambar 1.3 adalah ilustrasi skema proses kerja dari pada bloom filter. Bloom filter digunakan untuk mempercepat pencarian sebuah identifier yang sedang dicari sebelum masuk ke arah storage, sehingga waktu proses pencarian akan lebih cepat dari pada langsung mencari pada sebuah storage. Selain mempunyai keunggulan pada kecepatan waktu proses, bloom filter juga membantu untuk mengurangi akses yang tidak penting pada storage, sehingga kerja dari storage itu sendiri akan lebih ringan.

3. Poin Analisa

Sistem Kill Bots ini akan diuji dan dibandingkan dengan sistem web server yang tidak diimplementasikan sistem Kill Bots maupun modul pertahanan lainnya dan juga dengan server yang menggunakan modul `mod_evasive` sebagai modul tambahan untuk pertahanan. Poin yang akan dianalisa hanyalah respond time,

karena server yang digunakan hanya menyediakan sebuah halaman web yang bersifat statik sehingga tidak menggunakan banyak proses dibelakangnya.

4. Uji Coba

Uji coba dilakukan dengan menggunakan sepuluh komputer sebagai penyerang, satu buah komputer yang mempunyai peran sebagai orak dari serangan, dan juga satu komputer sebagai server yang akan diserang. Total komputer yang akan digunakan dalam uji coba ini adalah 12 unit komputer. Dengan menggunakan sumber daya yang cukup banyak, hasil uji coba tugas akhir ini pun dapat mencapai batasan yang diinginkan.

Jumlah maksimal dari koneksi yang dibentuk dalam uji coba ini adalah 30000 dengan parameter yang berbeda agar mendapatkan hasil yang bervariasi. Dengan menggunakan aplikasi Apache JMeter sebagai web server benchmarking tools, uji coba terhadap server yang siap sebagai korban akan dilancarkan. Skenario yang digunakan berjumlah empat buah. Dua diantaranya akan dirancang menyerupai lingkungan server dengan tingkatan normal padat, satu skenario digunakan sebagai skenario awalan untuk penyerangan yang sesungguhnya, dan satu skenario sebagai skenario penyerangan yang sesungguhnya. Uji coba telah dilakukan dalam waktu satu bulan. Dalam waktu satu bulan tersebut, banyak data yang telah didapat, tetapi hasil laporan yang akan disajikan berupa data yang terlihat perbedaan antara kondisi-kondisi yang diujikan.

5. Batasan

Adanya batasan-batasan yang dibuat agar penelitian pada tugas akhir ini menjadi terarah. Batasan-batasan tersebut adalah sebagai berikut,

- Sistem Kill Bots diimplementasikan pada media modul Apache web server.
- Pengujian berlangsung pada lingkungan jaringan local.
- Serangan DDoS yang dipakai adalah bertipe Flash Crowd.
- Server akan berada pada sistem operasi linux yang dijalankan pada virtual machine.
- Bloom filter yang digunakan adalah Bloom filter dengan struktur dasar.

1.4 Langkah-Langkah Penyelesaian Tugas Akhir

Dalam pengerjaan tugas akhir ini, ada beberapa metodologi yang dilakukan untuk mencakup semua ruang lingkup yang dikerjakan. Poin-poin metodologi adalah sebagai berikut,

1. Mencari referensi yang mempunyai lingkup sama sebagai bahan pembelajaran lebih lanjut.
2. Menganalisa kerja dari web server dengan tidak adanya serangan yang masuk.
3. Mempelajari Apache lifecycle.
4. Mempelajari Apache Module Development.
5. Mempelajari perilaku dari DDoS Flash Crowd secara teori dengan sumber jurnal dan bahan bacaan yang lain.
6. Mempelajari akibat dari serangan DDoS Flash Crowd pada web server yang tidak mempunyai sistem pertahanan apa pun.
7. Mendokumentasi hasil dari serangan yang telah dilakukan.
8. Mempelajari struktur data Bloom filter.
9. Menerapkan sistem Kill Bots ke dalam modul Apache.
10. Mendokumentasi serangan DDoS Flash Crowd pada web server yang ditambahkan mod_evasive sebagai modul pertahanan.
11. Mendokumentasi serangan DDoS Flash Crowd pada web server yang ditambahkan sistem Kill Bots.
12. Menyusun hasil uji coba dan dokumentasi.
13. Menyusun buku tugas akhir.

1.5 Sistematika Pembahasan

Sistematika pembahasan yang digunakan pada buku Tugas Akhir ini terbagi menjadi delapan bab. Isi pembahasan masing-masing bab akan mencakup hal-hal seperti berikut ini:

- Bab I : PENDAHULUAN

Pada bagian ini dijelaskan mengenai latar belakang, tujuan pembuatan Tugas Akhir, ruang lingkup yang menjelaskan batasan-

batasan dalam pembuatan program, metodologi yang digunakan dan sistematika pembahasan tiap bab.

- Bab II : DISTRIBUTED DENIAL OF SERVICE

Pada bagian ini akan dijelaskannya mengenai apa itu serangan Distributed Denial of Service. Selain penjelasan mengenai apa itu DDoS, ada juga penjabaran jenis-jenis dari serangan DDoS.

- Bab III : PEMBUATAN MODUL APACHE

Pada bagian ini akan dijelaskan mengenai dasar teori Tugas Akhir ini mengenai modul Apache. Mulai dari cara memuat modul sampai dengan membuat dari awal.

- Bab IV : MODUL EVASIVE PADA APACHE WEB SERVER

Pada bagian ini akan dijelaskan mengenai modul pertahanan DDoS pada Apache. Penjelasan modul evasive ini bermula dari struktur prosesnya sampai dengan komponen apa saja yang dipakai.

- Bab V : ANALISA DESAIN SISTEM KILL BOTS

Pada bagian ini akan dijelaskan mengenai struktur arsitektur proses Kill Bots.

- Bab VI : IMPLEMENTASI SISTEM KILL BOTS PADA APACHE

Pada bagian ini akan dijelaskan mengenai baris-baris kode pada fungsi-fungsi yang ada pada Kill Bots.

- Bab VII : UJI COBA PERFORMA

Pada bagian ini akan ditunjukkan mengenai uji coba yang dilakukan terhadap tiga buah kondisi server terhadap lingkungan server normal sampai dengan serangan yang sesungguhnya.

- Bab VIII : PENUTUP

Pada bagian ini berisi kesimpulan dari sistem Kill Bots. Pada bab ini juga disertakan saran mengenai kemungkinan pengembangan pada percobaan yang sudah dilakukan.

BAB II

DISTRIBUTED DENIAL OF SERVICE

Pada tahun 2000, sebuah serangan DDoS dilancarkan pada website yang terkenal, yaitu Yahoo.com, mengakibatkan website tersebut tidak dapat diakses, atau sering disebut dengan istilah down, selama kurang lebih 120 menit atau dua jam lamanya. Selain itu, serangan yang memakan waktu yang cukup lama tersebut menyebabkan kehilangannya pendapatan pada bidang periklanan mereka. Setelah kejadian tersebut, banyak serangan-serangan mulai menyerang perusahaan yang bekerja dalam bidang menyediakan sebuah jasa untuk anti-spam.

Sebuah serangan *Distributed Denial of Service* atau yang biasa dikenal dengan singkatan DDoS, adalah serangan yang terkoordinasi oleh seorang penyerang yang ingin menjatuhkan sebuah service yang ada. Serangan DDoS biasanya mempunyai skala yang cukup luas, sehingga dampak yang dikenakan pada korban cukup signifikan. Terlepasnya besar skala yang dilancarkan, DDoS sendiri adalah sebuah serangan yang terjadi karena akumulasi serangan yang dinamakan Denial of Service atau yang sering disebut dengan DoS. Serangan DoS ini sendiri hanya memakai sebuah sumber, yaitu komputer sebagai agen yang diutus untuk menyerang sebuah target serangan.

2.1 Denial of Service

Serangan DoS adalah sebuah serangan yang menggunakan hanya sebuah sumber, yaitu sebuah komputer yang terkoneksi dengan internet. DoS mempunyai sebuah tujuan umum, yaitu untuk menghalangi seorang pemakai yang mempunyai tujuan ingin mengakses sebuah *service* agar tidak dapat memakainya kembali.

2.1.1 Metode Serangan

Ada dua macam pengelompokan serangan DoS, yaitu serangan di mana bertujuan untuk menghancurkan sebuah service dan serangan yang bertujuan

hanya untuk membanjiri sebuah service, dalam serangan ini tidak adanya akibat yang fatal terjadi pada service tersebut. Serangan DoS mempunyai lima macam tipe serangan dasar, yaitu:

- Konsumsi sumber daya komputasi, contohnya *bandwidth*, alokasi ruang pada tempat penyimpanan, waktu yang dibutuhkan oleh *processor*.
- Gangguan informasi pada sebuah konfigurasi, contohnya informasi routing pada sebuah router.
- Gangguan informasi pada sebuah status koneksi, contohnya teratur ulangnya sebuah *TCP session*.
- Gangguan terhadap komponen pada *physical network*.
- Menghalangi sebuah komunikasi antara pengguna dan korban sehingga mereka tidak dapat berkomunikasi secara memadai.

Sebuah serangan DoS dapat juga terdapat sebuah tindakan atau eksekusi dari sebuah *malware* yang mempunyai tujuan untuk:

- Menghabiskan sumber daya dari sebuah processor agar processor tersebut tidak dapat melakukan sebuah pekerjaan kembali, atau yang sering disebut dengan *overload*.
- Memicu kesalahan pada *Microcode* sebuah mesin. Microcode adalah sebuah layer pada instruksi layer hardware atau struktur data yang ikut serta dalam implementasi dari instruksi pada *high level machine code* yang berada di *central processing unit* atau CPU.
- Memicu kesalahan pada proses intruksi pengurutan sehingga memaksa komputer menjadi tidak stabil dan terjadi *lock-up*.
- Memanfaatkan kesalahan pada sistem operasi sehingga tidak adanya sebuah proses baru yang dapat dikerjakan.

Ada banyak macam teknik serangan DoS yang telah beredar di internet dan terus berkembang dengan berjalannya waktu dan perkembangan teknologi yang semakin pesat. Teknik-teknik yang sering digunakan adalah sebagai berikut,

1. ICMP Flood

ICMP adalah singkatan dari *Internet Control Message Protocol*. Serangan ini juga dikenal dengan *Ping-Attack*. ICMP Flood adalah serangan di mana penyerang mengirimkan paket data ICMP pada server dengan skala yang besar secara terus menerus sehingga server akan sibuk untuk membalas paket-paket yang datang dan tidak dapat merespon paket data yang lain. Pada serangan ini, seringkali penyerang menyisipkan sebuah malware yang akan dieksekusi secara diam-diam.

2. SYN Flood

SYN Flood adalah sebuah serangan yang mengacu pada paket SYN. Paket SYN ini adalah sebuah paket yang digunakan untuk menjalin sebuah koneksi TCP antara pengguna dan server. Serangan ini memodifikasi proses komunikasi TCP atau yang lebih dikenal dengan nama lain, yaitu *Three-Ways Handshake*, di mana penyerang hanya mengirimkan paket SYN saja, ketika server mengirimkan SYN-ACK, pengguna tidak mengirimkan respon balik yang sesuai, yaitu paket ACK. Serangan ini membuat server membukan koneksi mereka untuk menunggu balasan respon dari pengguna. Dengan demikian, server harus meninggalkan beberapa *listener* untuk menangani koneksi tersebut.

3. Nuke

Nuke ada sebuah serangan yang berumur cukup tua di mana cara serangannya mengacu pada ICMP. Ada pula perbedaan dengan ICMP Flood, yaitu pada bagian paket ICMP itu sendiri. Pada ICMP Flood, paket yang dikirimkan adalah paket yang valid, sedangkan pada Nuke, paket ICMP yang dikirimkan adalah paket yang tidak valid dan memakai sebuah ping yang telah dimodifikasi. Serangan seperti ini membuat komputer mulai menurun proses kerjanya sampai dengan berhentinya kerja dari komputer secara total.

4. R-U-Dead-Yet?

Serangan ini adalah serangan yang targetnya langsung pada aplikasi web sehingga akan mengakibatkan tidak tersedianya aplikasi tersebut untuk diakses secara normal. R-U-Dead-Yet sering disebut juga dengan sebutan

RUDY. RUDY mempunyai kesamaan dengan Slowloris dengan perbedaan pada *Method Request* yang digunakan. Pada RUDY, Method Request yang dipakai adalah POST sedangkan pada Slowloris adalah GET.

5. Slow Read

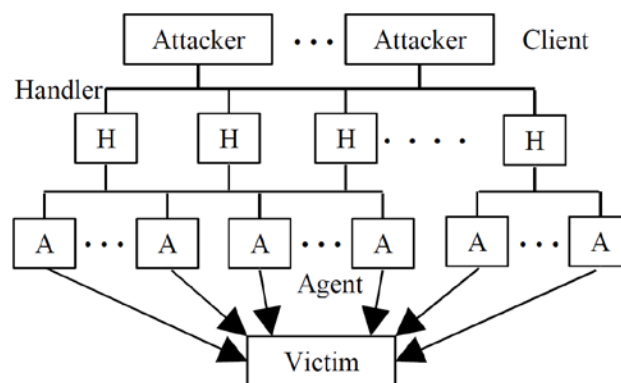
Serangan ini mempunyai kesamaan tingkah laku dengan pengguna yang normal. Perbedaannya ada pada pembacaan respon dari server. Pembacaan respon yang sangat lambat membuat server akan membuka koneksinya terus menerus dengan pengguna tersebut dan mengakibatkan koneksinya menjadi half-open.

2.2 Arsitektur Serangan DDoS

Serangan DDoS sendiri mempunyai dua macam arsitektur serangan yang telah dikelompokkan, yaitu model Agent-Handler dan model Internet Relay Chat (IRC).

2.2.1 Model Agent-Handler

Model *Agent-handler* ini mempunyai beberapa komponen dasar, yaitu *client*, *handler*, *agent*. Client adalah sebuah pos di mana penyerang dapat berkomunikasi dengan seluruh sistem serangan DDoS. Handler adalah sebuah program yang yang ditempatkan pada internet yang mempunyai tujuan agar client dari penyerang ini dapat berkomunikasi dengan para agent. Agent sendiri adalah sebuah komputer yang telah diretas sistemnya sehingga dapat dikontrol untuk meluncurkan serangan-serangan yang telah dikoordinasi. Penyerang akan berkomunikasi dengan handler-handler yang telah disebar di internet dan memastikan apakah para agent-nya aktif dan bekerja secara normal ketika adanya jadwal serangan atau ketika meningkatkan kemampuan dari agent tersebut. Pemilik komputer yang telah diretas tidaklah tahu jika komputer mereka telah diretas dan siap menjadi pelaku sebuah serangan DDoS.



Gambar 2.1
Arsitektur Agent-Handler

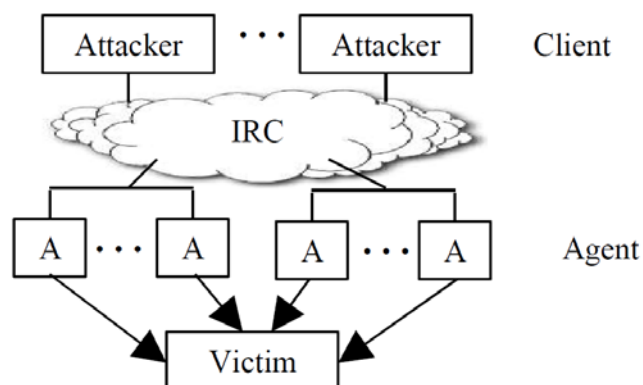
Tergantung bagaimana penyerang meretas dan mengatur sebuah serangan DDoS pada sebuah jaringan, agent dapat diperintahkan untuk menggunakan sebuah handler atau langsung dengan beberapa handler. Penyerang mempunyai kebiasaan untuk meletakkan handler mereka pada sebuah router yang telah diretas atau sebuah server jaringan yang menangani jaringan yang mempunyai volume arus yang besar. Tindakan ini menyulitkan untuk mendeteksi pesan apa yang sedang dikirimkan mulai dari client ke handler dan dari handler ke agent. Handler dan agent mempunyai nama lain yang sering juga dipakai, yaitu master dan daemon.

2.2.2 Internet-Relay Chat

Internet-Relay Chat atau yang sering disebut dengan IRC adalah salah satu arsitektur serangan dari DDoS. Model arsitektur ini mempunyai kemiripan dengan model Agent-Handler, perbedaannya pada penempatan handler program, jika model Agent-Handler ditempatkan pada sebuah network server, model serangan ini memanfaatkan kegunaan IRC itu sendiri untuk berkomunikasi.

Pemakaian IRC mempunyai keuntungan tersendiri seperti menggunakan port IRC yang normal untuk mengirimkan sebuah perintah ke para agent. Tindakan ini membuat pelacakan jejak dari penyerang cukup sulit. Selain itu, IRC sendiri mempunyai tingkat kepadatan jaringan yang besar sehingga penyerang menjadi semakin mudah untuk melenyapkan keberadaannya. Ada pula kegunaan yang menjadi keuntungan dari penyerang dengan menggunakan IRC ini, yaitu

penyerang tidak perlu untuk membuat sebuah daftar siapa saja agent-agent yang penyerang punya, namun dengan masuk ke IRC dan melihat siapa saja agent yang dapat dipakai atau tersedia. Sebuah program yang telah ditempatkan pada IRC server itulah yang biasanya akan berkomunikasi ke semua agent dan menanyakan mereka apakah mereka dapat dipakai atau tidak.



Gambar 2.2
Model Arsitektur IRC

Pada model arsitektur ini, para agent sering disebut dengan panggilan lain, yaitu “Zombie Bots” atau “Bots” saja. Pada kedua model, IRC atau Agent-Handler, sebuah agent juga biasa disebut dengan “Secondary Victim” atau “Zombie”, dan target utama dalam penyerangan sering disebut juga dengan “Primary Victim”. Program yang mengontrol para agent menggunakan hanya sedikit porsi dari sumber daya, sehingga “secondary victim” tidak akan merasakan akibat atau perbedaan performa yang signifikan ketika sistem mereka berada pada garis penyerangan DDoS.

2.3 Pengelompokan Serangan DDoS

Sampai sekarang, serangan DDoS telah berkembang dan memperanakan banyak jenis-jenisnya, tetapi dengan banyaknya jenis metode dan teknik yang digunakan, dalam dasarnya, serangan DDoS mempunyai dua macam tipe, yaitu penyerangan yang bertujuan untuk menghabiskan bandwidth yang ada, “Bandwidth Depletion Attack” dan penyerangan yang bertujuan untuk

menghabiskan sumber daya dari korban sehingga tidak dapat memproses apa pun yang baru masuk, “Resource Depletion Attack”.

2.3.1 Bandwidth Depletion Attack

Dalam serangan yang tertuju pada penghabisan bandwidth ini dapat dikategorikan menjadi dua macam, yaitu Flood dan Amplification. Serangan Flood ini mengupayakan sejumlah zombie dengan skala yang sangat besar untuk sebuah korban. Dengan banyaknya jumlah zombie yang menyerang, kepadatan jaringan dari korban akan meningkat secara pesat sehingga mengakibatkan sistem dari korban akan mulai melambat, terjadinya kesalahan fatal, atau terjadinya kejenuhan pada bandwidth jaringan tersebut sehingga membuat pengguna yang secara normal ingin mengakses jaringan tersebut dikeluarkan dari jaringan tersebut dan tidak dapat kembali dalam jangka waktu tertentu. Ada beberapa tujuan protokol atau paket data yang sering menjadi lahan penyerangan Flood ini, yaitu UDP dan ICMP. Ketiga hal inilah yang seringkali menjadi target serangan dari penyerang.

Pada serangan yang mempunyai target UDP, penyerang akan mengirimkan paket UDP sebanyak-banyaknya kepada korban, entah pada sebuah port yang telah ditargetkan atau penyerangan pada port yang diacak secara tidak beraturan. Korban yang diserang akan mencoba mendeteksi paket data yang masuk, jika korban tidak menjalankan sebuah prosedur pengesahan terhadap port yang menjadi target serangan maka akan dikirimkannya sebuah paket ICMP dengan pesan di mana port tersebut tidak dapat dicapai. Pada kasus yang lebih rumit dan sulit, serangan DDoS ini juga diikuti dengan serangan spoofing, di mana penyerang mengubah sumber pengiriman data yang dikirimkan kepada korban, sehingga korban akan mengembalikan paket data balasan bukan kepada zombie, tetapi kepada alamat yang tertera pada paket data yang telah diterima, di mana alamat tersebut adalah alamat pihak lain atau alamat kosong yang mana tidak adanya pihak yang memakai alamat tersebut. Serangan yang mengacu pada protocol UDP ini mempunyai kemungkinan berada pada koneksi bandwidth yang berlokasi

disekitar sistem korban. Kemungkinan ini juga dapat mengakibatkan sebuah kerusakan pada sistem yang berada disekitar korban.

Sebuah paket data ICMP_ECHO_REPLY yang dikirimkan oleh para zombie dengan skala yang besar adalah sebuah teknik serangan Flood yang mengacu pada protokol ICMP. Internet Control Message Protocol adalah kepanjangan dari singkatan ICMP tersebut. ICMP ini sering disebut dengan “ping” dan protokol ini sering sekali dipakai oleh para pengguna untuk mengetahui apakah sebuah target sedang aktif atau tidak, sedangkan serangan yang mengacu pada protokol ICMP ini sering dibut dengan “Ping of Death”. Cara kerja dari serangan ICMP ini adalah membuat sistem yang diserang mengirimkan terus menerus balasan dari para zombie ini dan mengakibatkan bandwidth koneksi tersebut mulai menjadi penuh dan dengan berjalannya waktu dan bertambahnya para zombie yang dikerahkan, maka akan penuhlah bandwidth koneksi tersebut.

Kategori kedua adalah serangan *Amplification*. Serangan ini cukup berbeda cara penyerangannya dengan serangan Flood. Jika Flood menyerang langsung kepada korban, Amplification menyerang melalui bantuan jaringan sekitar korban. Pada Amplification, penyerang akan mengirimkan sebuah paket data secara broadcast pada jaringan korban, membuat semua sistem yang sedang berjalan dan mempunyai jaringan yang sama dengan korban akan mengirimkan sebuah paket balasan ke korban. Serangan ini memanfaatkan sebuah fitur pada sebuah router, di mana router akan mereplika dan mengirimkan paket yang diterima secara broadcast ke seluruh sistem yang ada pada jaringan tersebut untuk mendapatkan tujuan dari paket data ditujukan. Serangan semacam ini memanfaatkan sistem broadcast untuk menguatkan serangan terhadap kepadatan jaringan tersebut dan mengurangi bandwidth sistem yang sedang diserang.

Amplification dapat dilakukan hanya dengan penyerangnya saja, dengan kata lain, serangan ini dapat menggunakan banyak sumber atau hanya menggunakan sebuah sumber saja. Jika penyerang ingin langsung menyerang korban tanpa perantara siapa pun, penyerang akan dinamakan sebagai zombie. Pada langkah ini, penyerang tidak perlu menggunakan tenaga ekstra untuk

mencari dan meretas sumber-sumber yang lain dan menjadikan mereka menjadi agent-agent penyerang untuk menyerang sebuah korban.

Smurf adalah salah satu contoh dari serangan Amplification. Target serangan dari Smurf ini adalah sistem jaringan yang mempunyai fitur broadcast. Penyerang akan mengirimkan sebuah paket data ke jaringan tersebut dan mengganti alamat ip dari pengirim sehingga paket data yang akan dikirimkan, dengan tujuan sistem yang menjadi korban akan membalas paket data yang pertama kali dikirimkan ke alamat yang bukan sebenarnya. Pada serangan ini, paket data yang sering dipakai untuk penyerangan adalah paket data ICMP_ECHO_REQUEST, di mana paket ini hampir sama dengan perintah “Ping”¹. Router pada jaringan yang sedang diserang akan menjadi alat bantu dalam penyerangan ini, disebut juga dengan istilah Amplifier, akan mereplikasi paket ICMP_ECHO_REQUEST yang didapat dari penyerang dan mengirimkannya ke seluruh sistem yang terhubung dalam satu jaringan dengan cara broadcast, dan setiap sistem yang menerima paket ICMP_ECHO_REQUEST, akan membalaskan sebuah paket data yang disebut dengan ICMP_ECHO_REPLY ke alamat korban yang menjadi target penyerangan. Tipe serangan semacam ini dapat memperkuat serangan yang dilancarkan sampai dengan seratus kali lipat dari hasil awalnya.

Selain Smurf, ada pula serangan *Fraggle*, serangan Fraggle ini hampir sama dengan Smurf, dengan perbedaan pada paket data yang dikirimkan. Pada Smurf, paket data yang dikirimkan adalah ICMP_ECHO, Fraggle mengirimkan paket data UDP_ECHO². Adanya beberapa variasi yang terdapat pada serangan Fraggle, di mana paket data UDP_ECHO dikirimkan ke port yang menyediakan “Character Generation” atau yang sering disebut dengan “Chargen”, yaitu pada port 19 pada UNIX, dengan alamat pengirim yang telah dimodifikasi ke echo service pada korban. Echo service adalah port 7 pada sistem operasi berbasis UNIX. Serangan tersebut akan mengakibatkan infinite loop atau perulangan yang terus menerus

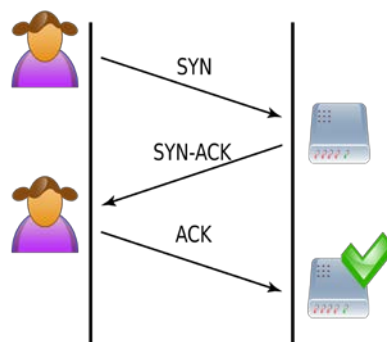
¹ TFreak, “smurf.c”, 11 Oktober 1997, <http://www.rs-labs.com/papers/tactics/ircutils/smurf.html> (3 May 2013).

² TFreak, “fraggle.c”, <ftp://ftp.ntua.gr/mirror/technotronic/denial/fraggle.c> (6 Mei 2013).

tanpa henti³. Fraggle, dengan sistem serangan berbasis paket data UDP_ECHO tersebut, mempunyai tingkat kerusakan yang lebih besar dari pada Smurf.

2.3.2 Resource Depletion

Serangan DDoS dengan tujuan untuk menghabiskan sumber daya dari korban mengharuskan penyerang mengirimkan paket-paket data yang nantinya akan menyalahgunakan protokol komunikasi jaringan dari korban. Sumber daya dari jaringan akan disibukkan, sehingga tidak ada lagi yang tersisa untuk dipakai oleh pengguna yang normal. Dengan demikian, pengguna yang ingin mengakses server tersebut dengan tujuan yang benar akan tidak dapat tercapai tujuannya karena semua proses yang ada telah dihabiskan akibat serangan yang dilancarkan.



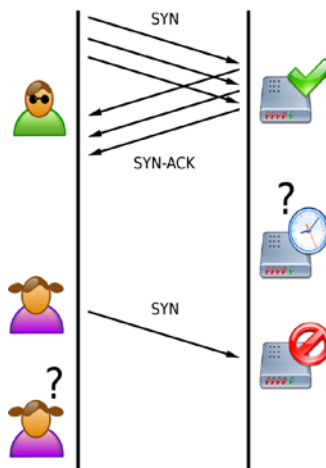
Gambar 2.3
Three-ways Handshake

Dalam serangan yang bertujuan untuk menghabiskan sumber daya dari korban ini mempunyai dua kategori, yaitu serangan yang berdasarkan pada penyalahgunaan protokol dari korban (Protocol Exploit) dan serangan yang menggunakan paket yang tidak valid (Malformed Packet). Penyalahgunaan protokol TCP SYN dan protokol PUSH+ACK adalah dua teknik penyerangan yang berbasiskan protokol dari korban (Protocol Exploit). Pada penyalahgunaan

³ Martin, Michael J., "Router Expert: Smurf/Fraggle Attack Defense Using SACLs", Networking Tips and Newsletters, <http://searchnetworking.techtarget.com/tip/Router-Expert-Smurf-fraggle-attack-defense-using-SACLs> (6 Mei 2013).

TCP SYN, penyerang memerintahkan para zombie untuk mengirimkan sebuah permintaan TCP SYN yang palsu kepada server korban.

Tujuan dari pengiriman sebuah permintaan yang palsu adalah untuk menghabiskan sumber daya dari processor server tersebut dan membuat server tidak dapat memberi timbal balik pada pengguna yang normal. Serangan terhadap TCP SYN ini memanfaatkan sistem untuk menjalin sebuah koneksi, yaitu Three-way handshake. Penyerang akan mengirimkan permintaan TCP SYN kepada server dengan jumlah yang besar dengan mengganti alamat ip dari pengirim, sehingga dengan penggantian alamat tersebut, setelah server memproses permintaan TCP SYN tersebut, akan dikembalikannya sebuah paket balasan berupa SYN+ACK yang berarti bahwa server siap menjalin koneksi, tetapi dengan pergantian alamat ip pengirim, maka pengirim tidak akan mendapat balasan SYN+ACK tersebut dari server.



Gambar 2.4
Serangan TCP SYN

Pada bagian pengiriman paket balasan dari server di mana paket balasan tersebut berupa SYN+ACK, server akan menugaskan sebuah listener yang akan menunggu dan menangani balasan dari pengirim, yaitu berupa paket ACK. Proses menunggu ini sering disebut dengan istilah *Half-open Connection*. Dengan zombie yang berjumlah besar, maka sumber daya dari server akan habis dikarenakan oleh prosedural dari server sendiri dalam proses menjalin koneksi. Kondisi demikian membuat semua koneksi yang masuk harus diletakkan pada

sebuah antrian atau *queue* yang mana akan segera ditangani setelah adanya proses yang kosong.

Kategori kedua adalah penyerangan di mana penyerang akan memerintahkan para zombie untuk mengirimkan paket yang mempunyai informasi tentang alamat ip pengirim dengan tidak tepat dengan maksud untuk menggagalkan sistem tersebut. Ada dua tipe yang arah serangannya mengacu pada kesalahan informasi, yaitu alamat ip dan paket ip. Pada tipe alamat ip, penyerang akan membuat sebuah paket data di mana isi dari paket tersebut mempunyai alamat sumber dan tujuan yang sama, sehingga paket tersebut akan membingungkan sistem korban dan dapat membuat sistem tersebut gagal. Pada tipe kedua, yaitu paket ip, paket data yang jelek mungkin dapat mengacak bagian yang opsional, mulai dari paket ip sampai mengatur semua kualitas bit dari service menjadi satu, sehingga korban harus menambahkan sebuah proses untuk menganalisa semua service terlebih dahulu. Jika penyerangan ini digandakan, maka akan menghabiskan tenaga kemampuan memproses dari sistem korban.

2.4 Serangan DDoS dalam Uji Coba Tugas Akhir

Serangan DDoS mempunyai banyak macam metode yang dapat dipakai dalam menyerang sebuah korban seperti yang telah dijelaskan. Dalam pengerjaan tugas akhir ini, metode serangan DDoS yang akan digunakan mempunyai sifat yang mirip dengan pengguna yang sah. Serangan ini sudah jarang sekali terdengar di berita maupun di forum-forum *underground*. Forum underground adalah forum di mana para hacker dari yang masih tingkat pembelajaran sampai dengan tingkat profesional berbagi info dan mencari pekerjaan sampingan.

Forum underground ini bukan seperti namanya yang terkesan gelap dan menakutkan, sebaliknya forum ini cukup mudah dicari dan tidak ada unsur berbahaya sama sekali bagi yang tidak tahu tempat apa itu. Hackforum.net adalah salah satu contoh forumnya. Dalam forum tersebut ada berbagai macam hal yang dapat dicari, mulai dari materi tentang hacking sampai pencarian jasa untuk menyerang sebuah korban. Forum tersebut masih aktif sampai dengan sekarang. Hacker bukan berarti mereka menutupi keberadaan mereka, sebaliknya mereka

adalah kumpulan orang yang ingin membagikan ilmu-ilmu mereka kepada orang yang ingin belajar lebih lanjut tentang hacking.

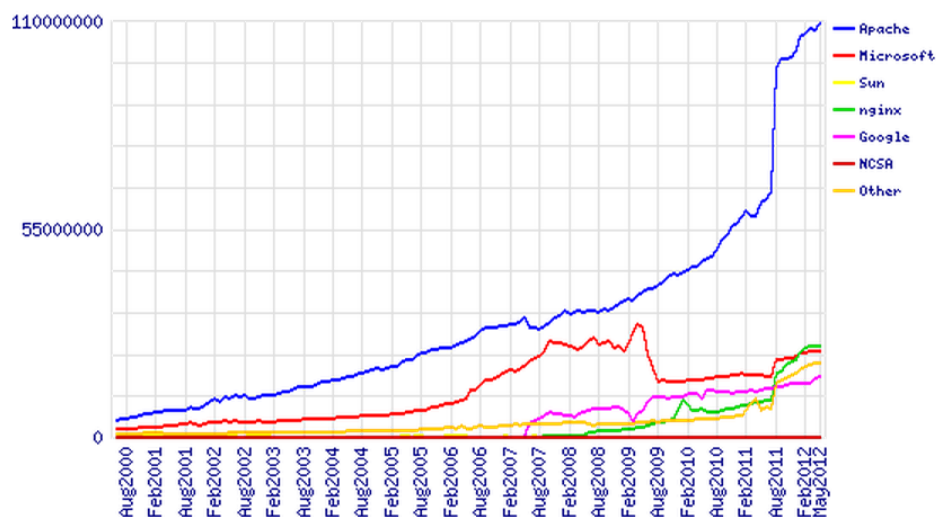
Serangan yang akan digunakan dalam tugas akhir ini disebut dengan *Mimic Flash Crowd DDoS*. Sesuai dengan namanya, serangan ini meniru kondisi sebuah jaringan di mana tingkat kepadatan jaringan tersebut telah mencapai batas maksimalnya secara normal dikarenakan banyak yang mengakses korban serangan. Serangan ini mempunyai sifat yang sama dengan metode *Resource Depletion*. Protokol bukanlah tujuan utama serangan ini, melainkan menyerang dengan menggunakan sifat sebuah pengguna yang sah di mana akses yang dilakukan adalah akses yang dilakukan secara normal. Terlihat cukup sederhana dan tidak menakutkan, tetapi lihat efek samping yang dapat diakibatkan oleh serangan tersebut. Jika korban adalah sebuah web yang bersifat dinamik dan database ikut serta dalam prosesnya, maka ada tiga hal yang terpengaruh, yaitu disk bandwidth, database bandwidth, dan worker process. Ketiga hal ini, jika dipadatkan terus menerus proses kerjanya, maka lama kelamaan, server tersebut dapat down sehingga dibutuhkan proses restart sistem yang mana waktu yang dibutuhkan dalam proses tersebut cukup lama, sehingga dapat menghilangkan pendapatan dikarenakan tidak tersedia untuk diakses atau dengan nama lain sistem tersebut tidak online.

Sebuah proses dalam lingkungan komputer pasti memerlukan adanya program untuk menangani proses tersebut. Apache JMeter adalah sebuah program benchmarking yang fungsinya dapat mereplika sebuah request yang dilancarkan ke server dalam sebuah komputer. Apache JMeter dapat digunakan secara distributed, yang berarti terdapat satu komputer dapat memberikan sinyal untuk memberikan request ke pada request secara distribusi. Hasil dari Apache JMeter ini akan diproses ulang sehingga didapatkan data yang dapat direpresentasikan menjadi sebuah grafik atau *chart*.

BAB III

PEMBUATAN MODUL APACHE

Apache adalah sebuah web server yang telah mendunia dan mempunyai catatan reputasi yang tidak dapat dianggap sebelah mata. Web server Apache ini bersifat *open source*, di mana para pemakai, sampai para pengembang yang ingin mengembangkan Apache yang mereka anggap kurang menjadi lebih baik.



Developer	April 2012	Percent	May 2012	Percent	Change
Apache	107,686,403	56.66%	109,278,620	57.02%	0.36
nginx	24,253,806	12.76%	23,938,754	12.49%	-0.27
Microsoft	22,813,215	12.00%	22,803,442	11.90%	-0.10
Google	15,671,026	8.25%	15,855,806	8.27%	0.03

Gambar 3.1
Data Statistik Web Server yang Aktif
Agustus 2000 – Mei 2012

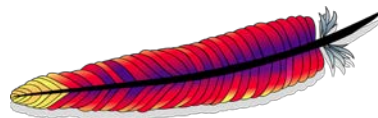
Gambar 3.1 adalah gambar yang diambil dari NetCraft.com di mana menunjukkan data statistic web server yang masih aktif dari bulan Agustus tahun 2000 sampai dengan bulan Mei tahun 2012⁴. Pada pengaturan standar, ada 167 module yang menyokong jalannya web server yang diminati jutaan pengguna.

⁴ NetCraft, “May 2012 Web Server Survey”, NetCraft.com, 2 Mei 2012, <http://news.netcraft.com/archives/2012/05/02/may-2012-web-server-survey.html>, (14 Mei 2013) .

Masih ada banyak lagi module-module yang dikembangkan secara terpisah dan tersedia secara gratis di internet.

3.1 Sejarah

Bermula pada National Center for Supercomputing Applications (NCSA) di University of Illinois, Urbana-Champaign, Rob McCool mengerjakan sebuah program yang berhubungan dengan *Hyper Text Protocol* atau sering disingkat dengan singkatan HTTP yang implementasinya pada sebuah server yang dinamakan Apache. Pada tahun 1994, pengerjaan program ini mulai mengalami kendala, karena Rob McCool meninggalkan NCSA. Pengembangan Apache tidak kandas, melainkan banyak pengembang-pengembang yang berkecimpung pada dunia web dan server unjuk gigi dan memulai penelitian mereka pada Apache. Mereka mulai membangun ekstensi-ekstensi mereka sendiri dan mulai membenahi *bug* yang terlihat.



Gambar 3.2
Logo Apache Web Server

Para pengembang Apache disatukan dan dikoordinasi akan perkembangan mereka terhadap Apache. Brian Behlendorf dan Cliff Skolnick adalah pencetus pertama dari koordinasi pengembangan Apache. Mereka mulai membangun sebuah mailing list untuk membagikan berita dari satu sama lain webmaster. Webmaster adalah seorang pengembang yang berkecimpung di dalam pengembangan web dan server. Proses pengelompokan ini mempunyai donatur yang membantu dari belakang, yaitu HotWired. HotWired adalah sebuah penyedia majalah secara online bersifat komersial yang berskala besar. Pada tahun 1995, mulailah Apache ini dipublikasikan dan digunakan menjadi web server yang sampai tahun 2012 tercatat bahwa Apache-lah yang menduduki peringkat pertama pada kategori web server dengan catatan angka 109.278.260 pada bulan mei 2012.

3.2 Fitur dan Performa

Apache web server mempunyai kemampuan untuk mendukung berbagai macam fitur. Fitur-fitur tersebut diimplementasikan ke dalam sebuah modul yang telah melalui tahap kompilasi yang bertujuan untuk memperluas fungsionalitas inti Apache sendiri. Perluasan fungsi ini mulai dari pemrograman yang berbasis server atau server-side programming sampai dengan skema autentikasi sebuah request yang masuk. Apache mendukung pemrograman yang berbasis server dengan menggunakan beberapa bahasa, yaitu Perl, Python, TCL dan PHP. TCL adalah sebuah bahasa pemrograman yang dikembangkan oleh John Ousterhout pada tahun 1990. TCL atau yang sering disebut dengan *tickle* sering digunakan untuk *Rapid Application Development* (RAD).

Modul-modul terkenal berbasis autentikasi yang ada pada Apache adalah `mod_access`, `mod_auth`, `mod_digest` dan `mod_auth_digest`. Selain modul-modul berbasis autentikasi ada pula modul-modul yang mendukung jalannya Apache web server, yaitu module *Secure Socket Layer* (`mod_ssl`), *Proxy* (`mod_proxy`), penggantian ulang URL (`mod_rewrite`), *Log* (`mod_log_config`) dan modul filter (`mod_include` dan `mod_ext_filter`). Metode kompresi yang digunakan pada Apache menggunakan module `mod_gzip` yang diimplementasikan untuk mengurangi ukuran dari halaman web site yang akan disediakan untuk pengguna. *Virtual Host* adalah salah satu fitur yang disediakan oleh Apache. Virtual Host adalah suatu metode di mana sebuah web server dapat mempunyai lebih dari satu website, contohnya adanya website `www.example.com`, `testing.edu`, `www.nothing-happen.org`.

Modul Apache adalah sesuatu yang dapat ditemukan dengan mudah di internet. Pengaturan Apache dapat bervariasi tergantung dengan kebutuhan dari penyedia dan pengguna itu sendiri. Selain itu, bagi para pengembang yang ingin mengembangkan modul-modulnya, Apache memberikan layanan penuh secara gratis karena modul-modul Apache bersifat *open-source*. Dengan kemudahan dan bersifat cuma-cuma, perkembangan modul-modul Apache naik secara signifikan. Perkembangan yang signifikan pada Apache akan membuat tantangan bisnis di dunia web server menjadi memanas dan membuat penyedia web server yang

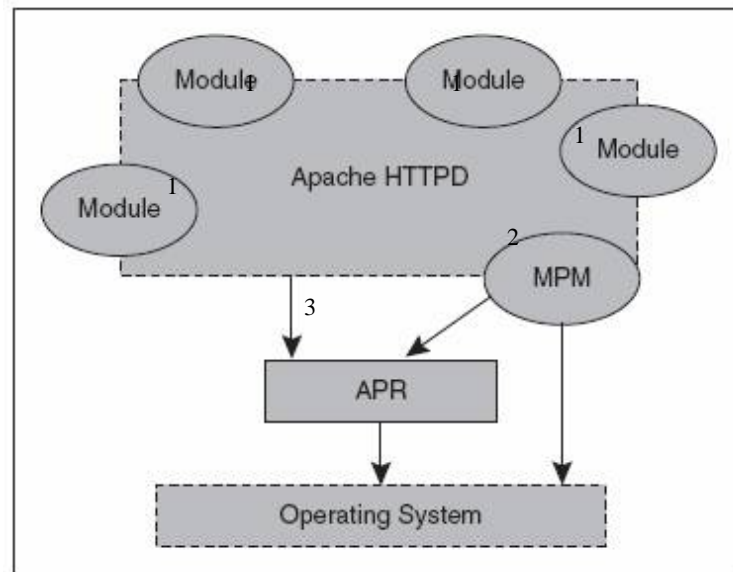
bersifat komersil harus selalu melangkah lebih jauh agar tidak kalah dengan Apache.

Meski Apache dirancang dengan tujuan untuk menjadi web server yang paling cepat, Apache mempunyai performa yang hampir serupa dengan web server dengan performa yang tinggi lainnya. Apache menyediakan fitur MPMs yang akan mendukung Apache untuk menjadi web server yang paling cepat. Multi-Processing Modules adalah kepanjangan dari singkatan MPMs yang mana membuat Apache dapat menjalankan *Process-Based*, *Hybrid Process and Thread* atau bahkan *Event-Hybrid* ke arah yang lebih baik dan menyamakan dengan permintaan setiap infrastruktur. Dengan demikian, pemilihan atas MPM dan pengaturan konfigurasi adalah sesuatu yang harus diperiksa dan dirancangkan dengan baik dan benar. Pemilihan tersebut akan berpengaruh pada beberapa aspek, yaitu pengurangan hambatan atau latensi dan menaikkan angka *throughput* atau keberhasilan paket data sampai pada tujuan, melayani sebuah request yang masuk dan waktu yang dibutuhkan untuk memproses request yang masuk.

Pada Apache 2.2 muncul beberapa kelemahan yang ada pada bagian proses sebuah halaman website yang bersifat statik. Kelemahan ini dikerjakan dengan maksimal oleh web server yang bersifat open-source, yaitu *Nginx*. Tidak sama dengan pemrosesan halaman yang bersifat dinamik. Apache dapat menangani kategori ini dengan kecepatan yang luar biasa. Dengan adanya kelemahan Apache tersebut, *Apache Foundation*, sebuah perkumpulan dengan beranggotakan para pengembang modul-modul Apache, mencetuskan ide multi-thread versi yang baru, yaitu menggabungkan beberapa proses dengan beberapa thread pada setiap prosesnya. Arsitektur multi-thread yang baru ini diimplementasikan dan dikokohkan pada Apache 2.4 di mana metode tersebut telah menambah performa dan mengurangi waktu yang dibutuhkan untuk memproses request-request yang masuk dan juga peningkatan performa dengan terobosan pada multi-thread mempunyai performa yang lebih baik dari pada web server yang bersifat event-based.

3.3 Arsitektur Apache

Apache menjalankan beberapa macam tugas yang pengerjaannya bersifat *background* atau pengerjaan yang tidak terlihat oleh pemakainya. Pada sistem operasi UNIX proses background ini dinamakan *Daemons* dan pada sistem operasi Windows dinamakan *Services*.



Gambar 3.3
Arsitektur Apache

Pada gambar 3.3 tergambar skema arsitektur dari proses kerja Apache web server. Seperti yang terlihat, Apache tergantung dengan adanya modul-modul yang mendukung kinerja prosesnya. Poin-poin yang telah tertera pada gambar 3.3 mempunyai penjelasan singkat sebagai berikut,

1. Module, adalah sebuah objek yang mempunyai peranan penting dalam proses kerja Apache, yaitu mengatur seluruh bagian proses yang sedang dilakukan. Module di sini dapat dikatakan sebagai inti dari Apache web server.
2. MPM, adalah sebuah metode pengaturan dan teknik mengolah modul-modul yang terdapat pada Apache, sehingga proses yang dikerjakan menjadi lebih cepat dan lebih efisien. MPM adalah singkatan dari *Multi-Processing Module*.

3. APR, adalah singkatan dari *Apache Portable Runtime* mempunyai kegunaan dalam berkomunikasi dengan sistem operasi. APR ini bertujuan untuk memudahkan penggunaan dan pengembangan dalam berbagai macam sistem operasi.

Start-Up atau proses yang bertujuan untuk menjalankan sebuah operasi membutuhkan waktu yang cukup lama dan juga sumber daya yang tidak sedikit, sehingga untuk server yang bersifat operasional, Apache mempunyai kebiasaan jika proses untuk menjalankan web server tersebut ikut pada awal mula sistem operasi berjalan atau pada saat *Booting* dan akan berakhir ketika sistem operasi tersebut berhenti secara keseluruhan. Web server Apache menawarkan ukuran inti yang cukup kecil dan bekerja bersama dengan beberapa macam modul. Modul-modul tersebut disusun secara static dan dijalankan secara dinamik pada sebuah *runtime*. Selain menitik beratkan pada modul-modulnya, Apache Portable Runtime atau yang sering disingkat dengan sebutan APR juga mempunyai peranan yang penting dalam Apache. APR menyediakan sebuah perlengkapan yang dapat digunakan untuk sistem operasi yang berbeda atau yang sering disebut dengan *Cross-Platform Operating System*. Tujuan khusus diadakannya modul adalah untuk menggunakan metode Multi-Processing Module (MPM). MPM adalah metode yang mempunyai kewajiban berkomunikasi dengan sistem operasi dari pada APR sendiri.

3.3.1 Two-Phase Operation

Apache mempunyai dua operasi yang harus dilakukan, yaitu start-up dan operational. Operasi start-up bertempat sebagai *root*, dan berisi proses penguraian file-file konfigurasi, memuat modul-modul, dan menginisialisasi sumber daya sistem, contohnya logs file, shared memory segment, dan koneksi database. Pada keadaan normal, Apache akan menjalankan bagian-bagian yang tidak mempunyai hak-hak istimewa dari pada langsung menjalankan bagian-bagian yang khusus dan mempunyai hak-hak istimewa secara tersendiri sebelum menerima dan memproses koneksi dari klien pada sebuah jaringan. Metode

keamanan yang bersifat dasar ini membantu Apache untuk mencegah *bug* yang kecil di mana akan mengarah ke penghancuran sistem yang rentan.

Operasi dua fase ini mempunyai beberapa implikasi untuk asitektur aplikasi. Pertama, semua hal yang memerlukan sebuah hak istimewa dari sistem harus dijalankan mulai dari saat sistem berjalan atau start-up. Kedua, penginisialisasian modul sebanyak-banyaknya pada saat sistem start-up adalah sesuatu tindakan yang baik, sehingga pada tahap memproses request-request yang masuk akan dibutuhkan sumber daya yang lebih sedikit dan minimal karena modul-modul tersebut telah dimuat pada awal sistem dijalankan dan hanya butuh proses pemanggilan saja untuk mengakses modul-modul tersebut. Sebaliknya, dengan waktu yang dibutuhkan dan sumber daya yang dibutuhkan pada proses start-up, menjadi kesalahan besar jika mencoba menjalankan Apache dari server yang umum seperti *inetd* atau *tcpserver*.

Fase start-up adalah fase di mana adanya proses untuk membaca dan memuat data konfigurasi, memuat modul-modul dan library, dan menginisialisasi sumber daya. Setiap modul dapat mempunyai sumber dayanya masing-masing dan mempunyai kesempatan untuk menginisialisasi sumber daya tersebut. Pada fase ini, Apache menjalankan sebuah program *single-process* dan *single-thread* dan mempunyai hak istimewa secara penuh.

Fase yang kedua adalah fase operasional. Pada akhir dari fase start-up, kontrol Apache akan melemparkannya ke Multi-Processing Module. MPM mempunyai kewajiban untuk mengatur operasi dari Apache pada tingkatan sistem. Pengaturan tersebut khususnya dalam hal memelihara dan menjaga pool yang berisi proses kerja dari para pekerja atau thread sebagai penyesuaian dan penyediaan sumber daya untuk sistem operasi dan batasan batasan yang berlaku. Proses yang asli akan menjadi *master*, yang mempunyai kewajiban untuk menjaga dan memelihara pool yang berisi *worker children*. Thread ini mempunyai tanggung jawab untuk melayani koneksi yang masuk ketika thread yang di atasnya sedang memproses dan membuat thread-thread yang serupa, menghancurkan thread anaknya, dan melayani sinyal komunikasi seperti *shut down* atau *restart*.

3.3.2 Struktur dan Konsep Dasar

Dalam modul Apache, terdapat empat konsep dan struktur yang harus dipahami terlebih dahulu. Keempat struktur ini adalah bagian inti dari Apache yang akan mengatur dan mengolah informasi dari request, server, koneksi dan proses itu sendiri. Inti-inti tersebut adalah sebagai berikut,

- `request_rec`
- `server_rec`
- `conn_rec`
- `process_rec`

Struktur `request_rec` dan `server_rec` adalah bagian yang sering digunakan untuk menangani dan berhubungan dengan pengembangan modul Apache. Sedangkan pada `conn_rec` dan `process_rec` cukup jarang para pengembang merambah ke bagian tersebut.

Sebuah `request_rec` akan dibuat ketika Apache menerima sebuah HTTP request dari client dan akan dihancurkan segera ketika Apache telah selesai memproses HTTP request tersebut. Objek ini akan dilewatkan ke berbagai modul yang bertujuan untuk memproses request yang masuk. Sesuai dengan namanya, `request_rec` adalah sebuah objek yang menyimpan data-data yang mengandung informasi tentang HTTP request yang masuk ke dalam Apache. Selain itu, juga mengandung beberapa kategori yang digunakan untuk menjaga status dan informasi dari client oleh Apache. Kategori-kategori tersebut adalah,

- Request pool, digunakan untuk mengatur alokasi sumber daya ketika sedang memproses sebuah request. Pengaturan ini berlangsung selama request tersebut ada.
- Sebuah vector konfigurasi yang dibertugas untuk mencatat tentang konfigurasi request yang bersifat static.
- Tabel dari HTTP input, output, dan error header.
- Tabel dari Apache environment variable, dan sebuah tabel yang berisi note tentang request data yang tidak boleh diketahui secara tertulis.

Environment yang dimaksudkan adalah sebuah ekstensi penulisan seperti SSI, CGI, mod_rewrite dan PHP).

- Sebuah pointer yang menuju ke semua objek yang bersangkutan, termasuk dengan koneksi, server dan objek request apa pun yang bersangkutan.
- Pointer yang menuju ke deretan filter input dan output.
- URI (Uniform Request Identifier) yang diminta oleh client dan penguraian secara internal yang mewakili objek tersebut, termasuk dengan handler dan filesystem mapping. URI adalah gabungan dari URL (Uniform Resource Locator) dan URN (Uniform Resource Name).

Web server yang bersifat logis adalah pengartian dari `server_rec`. Jika fitur virtual host digunakan, maka setiap virtual host akan mendapatkan `server_rec` mereka sendiri dan tidak ada keterikatan maupun hubungan dengan satu sama lain. `Server_rec` akan dijalankan sesaat sewaktu server mulai dijalankan dan tidak pernah hancur kecuali server tersebut dimatikan secara total. Tidak seperti `request_rec`, `server_rec` tidak mempunyai pool, tetapi sumber daya dari server perlu dialokasikan dari process pool yang mana disebarkan oleh semua server. Objek tersebut mempunyai vector konfigurasi yang mana berisi tentang nama server dan definisinya, sumber daya dan keterbatasan server, dan informasi tentang log server tersebut.

Pada Apache, `conn_rec` adalah sebuah representatif internal yang menangani tentang protokol TCP. Objek tersebut akan dibuat ketika Apache menerima koneksi baru dari client dan nantinya akan dihancurkan setelah koneksi tersebut ditutup oleh Apache. Alasan khusus untuk koneksi dibentuk adalah pelayanan satu atau lebih HTTP request, jadi satu atau lebih `request_rec` akan dijalankan pada setiap `conn_rec`, tetapi modul yang menangani tentang protokol dan filter yang berada pada level koneksi akan membutuhkan kegunaan `conn_rec` dan juga kegunaan dari beberapa modul yang bertujuan untuk mengoptimalkan performa dalam memproses data dalam kurun waktu tertentu atau selama waktu yang diberikan untuk request tersebut.

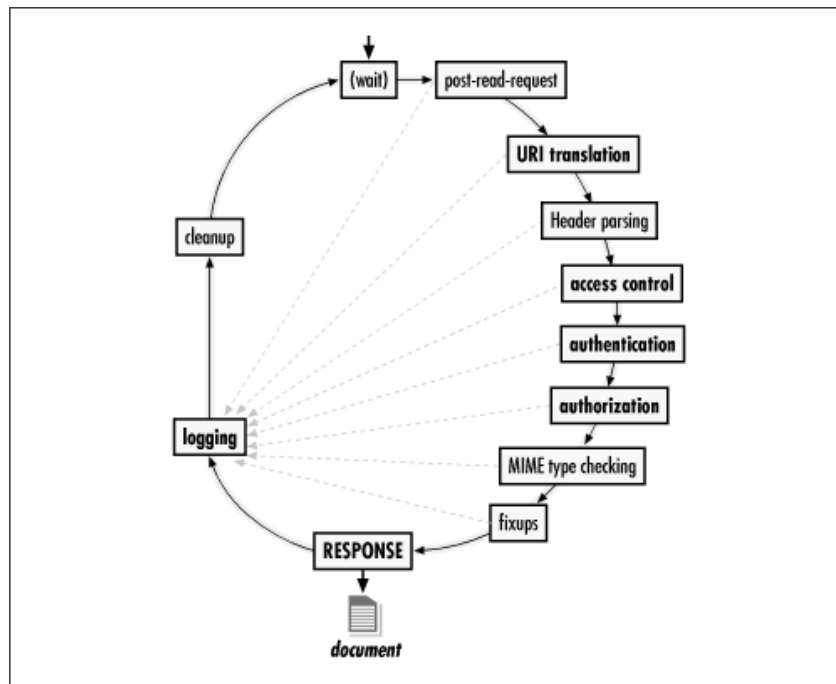
Struktur `conn_rec` tidak mempunyai informasi tentang konfigurasi apa pun, tetapi mempunyai sebuah vector konfigurasi yang berhubungan tentang koneksi yang terjadi dan juga terdapat pool untuk sumber daya koneksi. Selain itu, juga terdapat deretan dari filter input dan output koneksi ditambah dengan data-data yang mendeskripsikan protokol TCP yang masuk.

Struktur inti terakhir yang ada pada Apache adalah `process_rec`. Tidak seperti objek inti yang lainnya, `process_rec` ini termasuk pada bagian sistem operasi, bukan termasuk pada objek arsitektur web. Struktur ini biasanya digunakan sebagai shared memory atau pool, sehingga terdapat komunikasi antar modul jika diperlukan.

3.4 Modul

Apache adalah web server yang performa kinerja yang dikerjakan berdasarkan dari modul-modul yang ada. Sifat ketergantungan terhadap modul tersebut mempunyai keuntungan-keuntungan tersendiri salah satunya, yaitu Apache dapat diatur dengan sedemikian rupa sehingga dapat memenuhi keinginan dan kebutuhan para pamakai. Apache termasuk web server yang pengaturannya bersifat fleksibel.

Modul Apache adalah sesuatu yang dapat dikembangkan sendiri. Bahasa C adalah bahasa pemrograman yang mendasari modul-modul Apache dibentuk. Ada pula bahasa Perl yang dapat digunakan untuk mengembangkan modul Apache, tetapi karena Perl bukanlah bahasa standar yang dipakai oleh Apache, maka harus adanya modul lain yang digunakan untuk menyediakan pemrograman modul Apache dalam bahasa Perl, yaitu `mod_perl`. Bahasa Perl dalam pengembangan modul Apache mempunyai tingkat kepopuleran yang jauh di bawah bahasa C. Kebanyakan modul-modul yang tersebar di internet menggunakan bahasa C daripada bahasa Perl.



Gambar 3.4
Apache Module Lifecycle

Variabel-variabel yang ada pada sebuah modul tidak bersifat dapat dipakai secara terus menerus. Dalam hal ini yang dimaksudkan adalah variabel yang ada pada setiap modul hanya bekerja di dalam lingkup modul mereka sendiri-sendiri dan akan dihapus ketika Apache telah menyelesaikan periode *cycle* modulnya. Dengan data demikian, untuk dapat membagikan data yang ada, data tersebut harus diletakkan pada sebuah pool. Apache mempunyai pool yang beraneka ragam dan mempunyai jangka hidup yang berbeda-beda, sehingga pemilihan pool yang akan membantu penyebaran data ini adalah hal yang penting. Selain pemilihan pool, di dalam pool sendiri sudah mempunyai data-data yang telah ada akibat proses Apache secara umum, dengan demikian, jika ingin menyisipkan sebuah data pada pool, haruslah mempunyai key yang cukup unik sehingga tidak menumpuk dengan data yang sudah ada di dalam pool tersebut.

3.4.1 Programming Modul Apache

Apache menggunakan bahasa C sebagai pemrograman utama dalam pengembangan modul-modulnya. Bersamaan dengan runtime dari Apache sendiri,

maka pemrograman modul Apache sendiri tidak memerlukan baris kode yang panjang ketika tidak memakai APR.

Kebanyakan, pengembangan modul Apache berada pada lingkungan UNIX, dikarenakan pada sistem operasi yang berbasis UNIX sajalah, Apache menyediakan development pack. Meskipun Apache menggunakan bahasa pemrograman C, tetapi ketika proses compile, Apache menggunakan compiler-nya sendiri, yaitu APXS singkatan dari APache eXtensionS tool. Selain berfungsi sebagai compiler, APXS juga dapat difungsikan untuk memuat modul yang ingin dimasukkan ke dalam Apache httpd.conf. File tersebut digunakan sebagai sarana pencatatan konfigurasi dan modul-modul apa saja yang akan diinisialisasi ketika Apache memulai proses start-up. Script yang digunakan untuk menjalankan serta memuat modul ada sebagai berikut,

```
# sudo /opt/lampp/bin/apxs -i -a -c /home/user/mod_example.c
```

Setiap modul Apache bekerja dengan cara mengeluarkan sebuah struktur data modul. Pada umumnya, Apache mempunyai struktur data modul yang sama berapa pun versi Apache yang dipakai. Perbedaan hanya terdapat pada pemberian tanda pengenalan pada struktur data modul tersebut.

Segmen Program 3.1 Struktur Data Modul Apache 2.x

```
1: module AP_MODULE_DECLARE_DATA some_module = {
2:     STANDARD20_MODULE_STUFF,
3:     some_dir_cfg,
4:     some_dir_merge,
5:     some_svr_cfg,
6:     some_svr_merge,
7:     some_cmds,
8:     some_hooks
9: };
```

Struktur data modul yang dipakai oleh Apache dapat dilihat pada segmen program 3.1. Sebuah modul pasti harus diberi nama yang sesuai dengan kegunaannya. Penamaan tersebut dapat ditulis pada bagian baris pertama, yaitu pada `some_module`. `STANDARD20_MODULE_STUFF` adalah penanda apakah modul

ini berjalan di Apache versi berapa, dalam segemen program 3.1, menunjukkan bahwa modul ini akan digunakan pada Apache versi 2. Jika ingin mengembangkan versi pertamanya, maka penanda tersebut dapat ditulis dengan `STANDARD_MODULE_STUFF`. Perbedaan mereka hanya terdapat pada angka yang ada setelah kata `STANDARD`. Baris ketiga, yaitu `some_dir_cfg` adalah bagian di mana akan digunakan untuk melemparkan sebuah fungsi untuk membuat struktur konfigurasi setiap direktori pada server. Lanjut pada baris keempat, `some_dir_merge`, adalah bagian di mana akan dilemparkannya sebuah fungsi yang menangani tentang penggabungan struktur konfigurasi setiap direktorinya. Baris kelima dan keenam mempunyai fungsi yang hampir sama dengan baris ketiga dan keempat, perbedaannya hanya pada tempat yang dituju. Jika pada baris ketiga dan baris keempat, tempat yang dituju adalah setiap direktori dalam server, pada baris kelima dan keenam, tempat yang dituju adalah setiap host yang ada dalam server. Pada `some_cmds`, mempunyai fungsi untuk menunjukkan konfigurasi global yang akan dipakai oleh modul tersebut. Pada baris kedelapan, menangani tentang di mana modul ini akan dipasangkan pada Apache. Pada Apache, banyak bagian yang dapat dipasangkan atau disisipi sebuah modul yang baru yang mempunyai fungsi masing-masing di setiap tempatnya. Pada bagian ini pula adalah bagian yang harus diisi, jika tidak, sewaktu modul disusun oleh APXS, maka akan langsung terjadi error dikarenakan tidak adanya informasi tentang di mana modul ini akan disisipkan.

Segmen Program 3.2 Register Hook

```

1: static void helloworld_hooks(apr_pool_t *pool) {
2:     ap_hook_handler(helloworld_handler, NULL, NULL,
3:                     APR_HOOK_MIDDLE);
4: }
5: module AP_MODULE_DECLARE_DATA helloworld_module = {
6:     STANDARD20_MODULE_STUFF,
7:     NULL,
8:     NULL,
9:     NULL,
10:    NULL,
11:    NULL,
12:    helloworld_hooks
13: };

```

Segmen program 3.2 menunjukkan bahwa modul yang bernama `helloworld_module` akan disisipkan pada bagian *content handler*. Baris ke dua adalah penanda di mana modul `helloworld_module` ini akan disisipkan. Fungsi dengan awalan `ap_hook_` adalah sebuah fungsi yang digunakan untuk menyisipkan modul dan akhirnya adalah penentu di mana modul tersebut akan disisipkan, yaitu `_handler`. Ini menandakan bahwa modul akan disisipkan pada bagian *content handler*, yang mana bertugas dalam memproses content apa yang akan dikeluarkan oleh Apache ketika handler ini diakses. Fungsi untuk menyisipkan sebuah modul tersebut mempunyai empat parameter yang harus diisi. Parameter pertama adalah nama fungsi yang akan dipanggil, yaitu `helloworld_handler`. Parameter kedua digunakan untuk memanggil sebuah fungsi yang mana akan dieksekusi sebelum masuk ke bagian yang dituju, dalam kasus ini adalah *content handler*. Pada parameter ketiga, adanya kemiripan dengan parameter yang kedua, perbedaan ada pada waktu eksekusi, jika parameter kedua akan dieksekusi pada waktu sebelum masuk pada bagian yang dituju, parameter ketiga akan dieksekusi ketika modul ini telah dijalankan secara menyeluruh. Pada parameter terakhir, bagian ini adalah bagian yang mana modul akan ditentukan prioritasnya. Segmen program 3.2 menunjukan bahwa prioritasnya `APR_HOOK_MIDDLE` yang berarti bahwa modul ini mempunyai prioritas yang normal. Fungsi `APR_HOOK_` mempunyai beberapa macam jenisnya, yaitu:

- `APR_HOOK_REALLY_FIRST`
- `APR_HOOK_FIRST`
- `APR_HOOK_MIDDLE`
- `APR_HOOK_LAST`
- `APR_HOOK_REALLY_LAST`

Prioritas-prioritas yang akan ditempatkan adalah berdasar dengan nama belakang. Pada `APR_HOOK_REALLY_FIRST`, modul akan dijalankan dengan prioritas yang paling pertama pada bagian mereka. Adanya kemiripan dengan `APR_HOOK_FIRST`, prioritas ini menempatkan modul pada prioritas pertama, tetapi

ada pula perbedaannya, yaitu pada `APR_HOOK_REALLY_FIRST` akan menempatkan modul tersebut pada kalangan yang paling pertama dan pada `APR_HOOK_FIRST` akan dijalankan ketika semua modul pada prioritas `APR_HOOK_REALLY_FIRST` telah selesai dijalankan. Konsep yang sama juga dipunyai oleh kedua poin yang terakhir, hanya saja mereka berada pada prioritas yang terakhir.

Modul Apache bekerja berdasarkan bagian-bagian yang telah ditempatkan bagi modul tersebut. Ada beberapa macam bagian yang ada pada Apache, content handler adalah salah satu bagian yang menangani bagian konten yang akan dilemparkan kepada client. Bagian-bagian pada Apache, ada yang harus dilakukan pengaturan terlebih dahulu, ada juga yang tidak perlu. Pengaturan tersebut adalah pengaturan pada file `httpd.conf` yang terdapat pada Apache. File tersebut adalah file yang menangani tentang modul-modul yang akan dimuat pada Apache ketika dijalankan. Bagian-bagian pada Apache adalah sebagai berikut,

- `ap_hook_pre_config`, bagian ini akan dipanggil ketika sebelum server memproses file konfigurasi.
- `ap_hook_post_config`, digunakan untuk membenarkan atau memodifikasi header request yang masuk.
- `ap_hook_open_logs`, dipanggil ketika sesaat sebelum file log apapun diakses.
- `ap_hook_child_init`, akan dijalankan ketika sebuah process atau daemon akan diinisialisasi.
- `ap_hook_handler`, bagian ini adalah bagian yang menangani tentang konten yang akan diberikan pada client (content handler).
- `ap_hook_pre_connection`, dipanggil ketika sesaat setelah Apache menerima sebuah koneksi.
- `ap_hook_process_connection`, akan dijalankan ketika koneksi telah diterima. Bagian ini hanya diperbolehkan untuk implementasi berbasis protokol.

- `ap_hook_post_read_request`, pembacaan dan memproses sebuah request yang baru saja masuk sebelum fase-fase yang lainnya dijalankan adalah tugas pada bagian ini.
- `ap_hook_translate_name`, adalah bagian di mana proses penerjemahan URI sering dikerjakan. Misalnya untuk menerjemahkan URI menjadi sebuah nama file yang akan diakses oleh client, contohnya dapat dilihat pada storage online seperti *mediafire*.
- `ap_hook_header_parser`, member kesempatan untuk memeriksa ulang header yang datang dan menjalankan sebuah tindakan yang khusus, misalnya jika ada sebuah header yang bermaksud untuk menghabiskan sumber daya server dengan meminta file secara terus menerus.
- `ap_hook_check_user_id`, digunakan untuk mengecek informasi request yang datang dengan tujuan sebagai proses autentikasi.
- `ap_hook_fixups`, bekerja hampir sama dengan bagian `ap_hook_header_parser`.
- `ap_hook_type_checker`, adalah bagian di mana akan adanya pemeriksaan pada tipe content yang keluar.
- `ap_hook_access_checker`, dipanggil ketika adanya tindakan untuk mengakses sebuah file.
- `ap_hook_auth_checker`, akan dipanggil ketika sebuah file yang diminta oleh client membutuhkan autentikasi lebih lanjut.
- `ap_hook_insert_filter`, digunakan untuk menambahkan sebuah filter.

Dalam sebuah modul, dapat mempunyai banyak bagian yang dikerjakan tergantung dengan keperluan tentang tujuan pengembangan modul tersebut. `APR_HOOK_` dan `AP_HOOK_` adalah sebuah prefix fungsi yang sangat penting dalam pengembangan modul Apache. Selain kedua fungsi tersebut, masih banyak lagi fungsi-fungsi yang dapat digunakan sesuai dengan kebutuhan dan tujuan pengembangan modul Apache itu sendiri. Untuk mengadakan fungsi-fungsi

tersebut, tidak lupa harus adanya satu atau lebih file yang diikutkan dalam modul. File-file tersebut adalah sebagai berikut,

- `httpd.h`
- `http_core.h`
- `http_config.h`
- `http_connection.h`
- `http_log.h`
- `http_main.h`
- `http_protocol.h`
- `http_request.h`
- `http_vhost.h`

Poin pertama pada list file yang harus diikutkan adalah file yang memegang peranan paling penting dalam pengembangan modul Apache ini. File `httpd.h` mencakup informasi tentang `server_rec`, `request_rec`, `process_rec`, `conn_rec` dan objek-objek yang lain di mana mempunyai peranan penting untuk memudahkan pengembangan. Pada poin kedua, `http_core.h`, adalah file yang mempunyai fungsi dalam penyediaan fungsi-fungsi inti tentang beberapa hal, yaitu tentang koneksi, protocol yang digunakan, autentikasi, konfigurasi, remote, dan lainnya.

Untuk prosesnya, ada `http_config.h` yang menangani tentang proses konfigurasi pada Apache, `http_connection.h` yang bertugas menyediakan fungsi-fungsi yang menangani tentang proses koneksi yang terjadi, dan seterusnya sesuai dengan nama belakang file-file tersebut, Apache memberikan kemudahan bagi penggunaanya dan pengembang modul dalam membaca fungsi-fungsi dan nama file. File `http_main.h` mempunyai fungsi yang dapat dikatakan sebagai file yang menyediakan informasi tentang proses kerja dari Apache web server dan file ini sering dipakai dalam pengembangan untuk bagian cache, proxy, dan ssl.

3.5 Serangan DDoS Terhadap Apache Web Server

Ada pepatah mengatakan, “Tidak ada gading yang tidak retak”, meski pun terkenal sebagai web server yang populer, Apache web server masih mempunyai beberapa kelemahan. Dini ini, telah banyak metode-metode serangan yang dapat mengancam proses kerja dari sebuah web server. Metode-metode itu tidak hanya beristirahat diingatan pembuatnya saja, melainkan disebarkan sebagai bahan pengajaran baru yang semua orang dapat mempelajarinya. Pengajaran tersebut terletak pada internet, di mana semua orang dengan bebas dapat mengaksesnya kapanpun dan di mana pun. Salah satunya adalah serangan yang bernama Denial of Service atau yang sering disingkat dengan singkatan DoS. Seperti yang telah dibahas pada bab sebelumnya, DoS ini tidaklah berbahaya, karena hanya menggunakan sebuah komputer sebagai media serangannya. Jika berbicara tentang jaringan dan internet, di sana pasti ada sebuah materi tentang sistem distribusi. Dengan adanya teori tentang sistem distribusi, DoS pun menerapkan sistem tersebut dan menjadikan DoS yang sebelumnya hanya bekerja sendiri atau solo, sekarang menjadi serangan yang terkoordinasi dengan sebuah kepala serangan yang merancang dan melancarkan serangan. Dengan tambahan sistem distribusi yang telah diterapkan, menghasilkan sebuah kata yang disisipkan di depan kalimat Denial of Service, yaitu Distributed, menjadikan nama serangan yang mempunyai tingkat yang cukup serius tersebut menjadi Distributed Denial of Service.

Koneksi-koneksi yang tidak aktif, tidak dibuang begitu saja oleh Apache, melainkan disisihkan dalam periode tertentu. Metode tersebut memberikan jangka waktu untuk melayani koneksi-koneksi lain yang masuk. Jika periode penyisihan koneksi tersebut habis, maka akan ada service yang akan melayani kembali koneksi tersebut. Perkiraan waktu penyisihan koneksi yang tidak aktif tersebut adalah kurang lebih tiga detik. Setelah tiga detik, sebuah service akan dibentuk untuk melayani koneksi tersebut. Hasil tersebut didapatkan dengan bantuan modul standar dari Apache, yaitu `mod_status`. Modul tersebut dapat memperlihatkan berapa koneksi yang masuk, nomor identitas koneksi, waktu koneksi masuk, status koneksi, dan lain-lain.

Pada pengerjaan tugas akhir ini akan digunakannya Apache versi 1.7.1 sebagai media dalam mengimplementasikan sistem Kill Bots. Pemilihan versi 1.7.1 ini dikarenakan versi tersebut adalah versi peralihan di mana fungsi-fungsinya dapat digunakan pada versi 2 ke atas, tidak seperti versi di bawah dari versi 1.7, di mana banyak fungsi-fungsinya telah tidak dapat dipakai kembali atau *deprecated*. Implementasi pada Apache sendiri tidak merubah pada tingkat protocol, tetapi pada bagian penanganan request-nya, sehingga metode serangan DDoS yang akan digunakan adalah metode yang mengarahkan sebuah koneksi yang benar-benar mengakses web server tetapi dengan skala yang besar, kondisi ini disebut pula *Flash Crowd*.

BAB IV

MODULE EVASIVE PADA APACHE WEB SERVER

Ketahanan dari sebuah serangan adalah suatu hal yang dibutuhkan sebuah server untuk bertahan dari kejahatan-kejahatan di dunia internet. Kejahatan-kejahatan tersebut tidak selalu terlihat hebat atau besar, tetapi kejahatan yang kecil pun dapat merepotkan kinerja server yang diserang, dengan demikian, proses kerja server akan berkurang dan tidak dapat melayani para clientnya secara maksimal dikarenakan adanya serangan yang bersarang di dalamnya.

Mod_evasive adalah salah satu modul pertahanan yang dikembangkan pada Apache web server. Mod_evasive mempunyai kegunaan sebagai sebuah pertahanan untuk mengelak dari serangan HTTP DoS atau DDoS, atau serangan yang menyerang HTTP atau web yang dilakukan dengan menggunakan metode *brute force*. Brute force adalah sebuah metode penyerangan di mana penyerang akan mencoba semua cara secara satu per satu. Mod_evasive melakukan proses deteksinya dengan pembuatan sebuah hash table yang bersifat internal di mana isi dari hash table tersebut adalah alamat ip dan URI (Uniform Resource Identifier). URI adalah sebuah kesatuan dari URL (Uniform Resource Locator) dan URN (Uniform Resource Name). Penyimpanan alamat ip berguna untuk proses penolakan request yang dilakukan oleh sebuah alamat ip dengan sifat sebagai berikut,

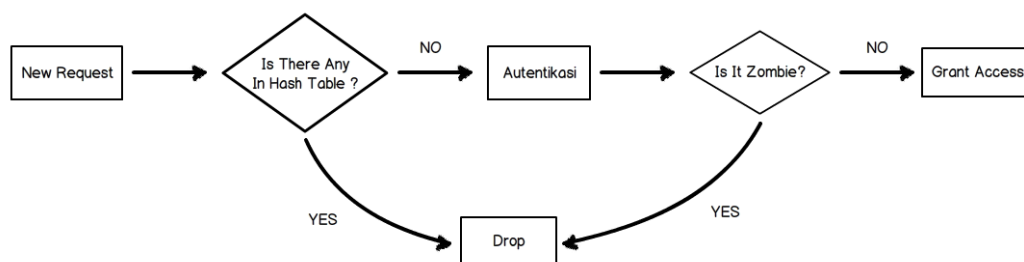
- Melakukan request secara terus menerus pada sebuah halaman yang sama dalam waktu satu detik.
- Membangun lebih dari 50 request secara bersamaan pada child yang sama dalam kurun waktu satu detik.
- Melakukan request ketika tercantum pada daftar alamat ip yang tidak diijinkan masuk.

Metode ini telah bekerja secara baik pada serangan single-server script dan juga serangan yang dilakukan secara distribusi, tetapi seperti alat bantu untuk

mengelak lainnya, metode ini hanya dapat berfungsi ke arah pengaturan bandwitdh dan konsumsi prosesor. Contoh konsumsi bandwitdh prosesor adalah ketika menerima, memproses, dan merespon balik kepada request yang tidak valid. Dengan demikian, pemakaian mod_evasive ini dan juga mengatur firewall dan router yang sesuai dengan data dari mod_evasive sendiri adalah sebuah tindakan yang baik.

4.1 Cara Kerja Mod_evasive

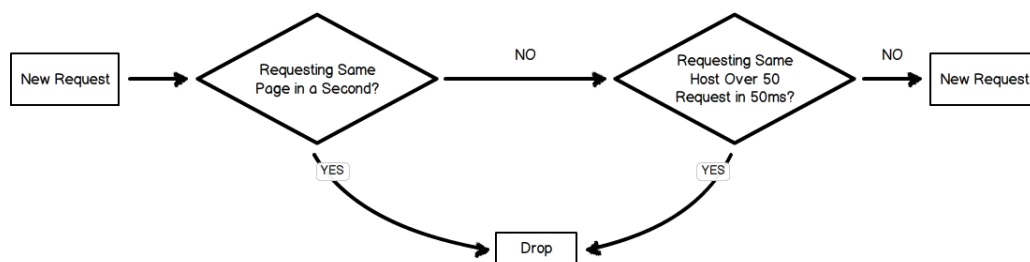
Mod_evasive mempunyai cara kerja yang cukup sederhana. Hash table adalah inti dari kinerja mod_evasive. Alamat ip dan URI yang dipakai serta kurun waktu adalah elemen-elemen penting dalam proses deteksi sebuah serangan yang mungkin terjadi. Ketika mod_evasive mendeteksi adanya kemungkinan serangan dari sebuah alamat ip, maka alamat ip tersebut akan dimasukkan dalam hash table yang menjadikan alamat ip dan URI sebagai key. Jika alamat ip tersebut melakukan kegiatan yang sesuai dengan batasan dari mod_evasive, maka mod_evasive akan meminta server agar memberikan kode HTTP respon 403 atau dengan kata lain, web tersebut tidak ditemukan (Not Found).



Gambar 4.1
Proses Mod_evasive

Gambar 4.1 memperlihatkan proses deteksi yang dilakukan oleh mod_evasive. Ketika sebuah request masuk, alamat ip request tersebut akan dicari pada hash table temporer. Setelah pencarian pada hash table temporer, proses dilanjutkan pada autentikasi. Dalam proses autentikasi terdapat lebih dari satu macam pemeriksaan. Mulai dengan jumlah akses per detik sampai dengan pemeriksaan tentang banyaknya request yang masuk secara bersamaan dalam

kurun waktu setengah detik atau 500ms. Pertama tama, alamat ip dan URI yang sedang dicoba untuk mengakses akan dibungkus dengan menggunakan sebuah fungsi hash dan menjadikan paket tersebut sebagai sebuah key pada hash table. Key tersebut akan mempunyai elemen nilai waktu dan jumlah key tersebut masuk. Jika jumlahnya melebihi satu kali dalam 1000ms atau per detik, maka alamat ip tersebut akan dimasukkan pada hash table temporer yang berisi semua alamat ip yang mempunyai potensi sebagai sebuah serangan.



Gambar 4.2
Proses Autentikasi Mod_evasive

Tahap terakhir pada proses deteksi adalah membungkus alamat ip tersebut dengan menggunakan fungsi hash. Paket tersebut mempunyai elemen nilai jumlah yang masuk. Jika jumlah tersebut melebihi angka 50 dalam waktu satu detik, maka alamat ip tersebut akan masuk ke dalam daftar alamat ip yang mempunyai sifat sebagai serangan. Ketika alamat ip tersebut mencoba untuk mengakses kembali, maka mod_evasive akan meminta server untuk mengantarkan respon 403, yaitu HTTP Not Found. Dengan demikian, server akan menghemat kerja yang dilakukannya sampai batas minimal.

4.2 Konfigurasi Mod_evasive

Sebuah modul Apache web server dapat mempunyai pengaturan konfigurasi tambahan menurut keperluan mereka sendiri-sendiri atau tidak mempunyai sama sekali. Begitu juga dengan modul yang bernama mod_evasive. Konfigurasi tambahan dapat diberikan kepada modul ini, dengan menambahkan pengaturan konfigurasi, batas-batas deteksi dari mod_evasive dapan diganti menurut konfigurasi yang diubah. Konfigurasi yang dapat diubah adalah sebagai berikut,

- *DOSHashTableSize*, konfigurasi yang mengatur tentang besar ukuran dari hash table yang dipakai. Nilai yang diberikan akan dengan secara otomatis mengikuti angka bilangan prima yang terdekat dan di atasnya.
- *DOSPageCount*, adalah konfigurasi yang mengatur batas deteksi pada bagian brapa banyak sebuah alamat ip mengakses halaman yang sama dalam kurun waktu yang ditentukan atau *PageInterval*.
- *DOSSiteCount*, adalah batasan yang mengatur tentang jumlah request yang dilakukan oleh client yang sama pada listener yang sama dalam halaman yang sama pula.
- *DOSPageInterval*, kurun waktu yang membatasi pada bagian *DOSPageCount*. Jika tidak diberikan sebuah nilai, maka nilai awalnya akan bernilai 1000ms atau satu detik.
- *DOSSiteInterval*, adalah waktu yang diberikan untuk membatasi pada bagian *DOSSiteCount*. Jika tidak diberikan sebuah nilai, maka nilai awalnya akan bernilai 1000ms atau satu detik.
- *DOSBlockingPeriod*, alamat ip yang telah dimasukkan dalam daftar alamat ip yang diduga sebagai serangan akan dapat dilepaskan dari daftar tersebut menurut batas waktu yang telah ditentukan oleh konfigurasi ini. Jika tidak diberikan sebuah nilai, maka nilai awalnya akan bernilai 10000ms atau 10 detik.

Konfigurasi-konfigurasi yang telah dijelaskan di atas adalah konfigurasi yang paling penting dalam mengubah batas-batas proses deteksi serangan. Sesuai dengan kebutuhan web server yang dipakai, *mod_evasive* dapat juga disesuaikan agar dapat bekerja dengan maksimal. Implementasi konfigurasinya sendiri dalam Apache cukup mudah, hanya perlu mencantumkan beberapa baris kode yang ingin ditambahkan pada file *httpd.conf* di dalam Apache web server.

Segmen Program 4.1 Konfigurasi Mod_evasive

```
1: <ifmodule mod_evasive20.c="">
2:     DOSHashTableSize    3097
3:     DOSPageCount        2
```

Segmen Program 4.1 (lanjutan)

```

4:      DOSSiteCount          50
5:      DOSPageInterval      1
6:      DOSSiteInterval      1
7:      DOSBlockingPeriod    10
8:  </ifmodule>

```

Pada segmen program 4.1 terlihat konfigurasi yang ditujukan kepada `mod_evasive`. Untuk bagian yang menangani tentang interval dan period, nilai yang diberikan adalah berupa detik atau second. Pada konfigurasi di atas tertulis bahwa interval untuk `DOSPageInterval` dan `DOSSiteInterval` adalah sebanyak satu detik, dan pada bagian `DOSBlockingPeriod` nilai yang diberikan adalah sebanyak 10 detik.

4.3 Named Timestamp Tree

Seperti modul Apache web server seperti pada umumnya, `mod_evasive` dikembangkan dengan menggunakan bahasa pemrograman C sebagai dasar pengembangannya. Struktur data yang dinamakan *Named Timestamp Tree* (NTT) adalah metode penyimpanan dan pembuatan hash table yang dipakai. Fungsi hash yang dipakai adalah fungsi hash yang mengandalkan perhitungan modulus, perkalian, dan penambahan untuk membungkus sebuah key yang akan digunakan sebagai alamat pencarian dalam hash table.

Segmen Program 4.2 Struktur Data dari Mod_evasive

```

1:  /* ntt root tree */
2:  struct ntt {
3:      long size;
4:      long items;
5:      struct ntt_node **tbl;
6:  };
7:
8:  /* ntt node */
9:  struct ntt_node {
10:      char *key;
11:      time_t timestamp;
12:      long count;
13:      struct ntt_node *next;
14:  };
15:
16:  /* ntt cursor */

```


Segmen Program 4.2 (lanjutan)

```

17: struct ntt_c {
18:     long iter_index;
19:     struct ntt_node *iter_next;
20: };

```

Segmen program 4.2 memperlihatkan struktur data yang digunakan oleh `mod_evasive`, yaitu Named Timestamp Tree (NTT). Pada bagian awal, yaitu pada `ntt`, variabel yang bernama `tbl` di mana variabel itu adalah sebuah array dengan adanya penanda dua buah pointer dibelakang nama variabel tersebut dan variabel tersebut mempunyai tipe sebagai `ntt_node`. Variabel `tbl` adalah sebuah array yang digunakan untuk menampung node-node yang nantinya akan diisikan. Pada bagian `ntt_node`, terlihat bahwa objek `ntt_node` mempunyai sebuah kaki yang dinamakan dengan `next` yang mempunyai tipe objek itu sendiri, yaitu `ntt_node`. Sepintas, node ini terlihat seperti node yang digunakan oleh metode struktur data Linked List dengan kaki `next` sebagai penunjuk objek yang selanjutnya. Seperti yang telah dijelaskan pada bagian pengantar, `mod_evasive` akan menyimpan key, jumlah, dan catatan waktu, terlihat juga pada elemen yang dimiliki oleh `ntt_node`. Pada objek terakhir yang ditunjukkan pada segmen program 4.2 adalah `ntt_c`. Objek ini digunakan sebagai alat bantu untuk proses pencarian yang dilakukan oleh struktur data NTT.

Segmen Prgoram 4.3 Inisialisasi Named Timestamp Tree

```

1: struct ntt *ntt_create(long size) {
2:     long i = 0;
3:     struct ntt *ntt = (struct ntt *) malloc(sizeof(struct
   ntt));
4:
5:     if (ntt == NULL)
6:         return NULL;
7:     while (ntt_prime_list[i] < size) { i++; }
8:     ntt->size = ntt_prime_list[i];
9:     ntt->items = 0;
10:    ntt->tbl = (struct ntt_node **) calloc(ntt->size,
   sizeof(struct ntt_node *));
11:    if (ntt->tbl == NULL) {
12:        free(ntt);
13:        return NULL;
14:    }
15:    return(ntt);
16: }

```

Sebuah struktur data pasti mempunyai sebuah fungsi yang digunakan untuk inisialisasi jika ingin dipakai. Segmen program 4.3 memperlihatkan fungsi inisialisasi yang dilakukan oleh NTT sebelum struktur data tersebut siap dipakai. Sebuah fungsi inisialisasi yang mempunyai parameter size sebagai inputan masuknya. NTT tidak menggunakan input parameter size sebagai ukuran seberapa banyak elemen yang harus dialokasikan, tetapi NTT menggunakan parameter size tersebut sebagai bahan perhitungan lebih lanjut. Pada baris ke tujuh, terlihat adanya perbandingan parameter size dengan `ntt_prime_list`. Array dengan nama `ntt_prime_list` adalah sebuah tempat untuk menampung angka-angka bilangan prima yang digunakan oleh NTT. Ketika `ntt_prime_list` mengembalikan nilai yang lebih dari parameter size, maka nilai tersebutlah yang akan dijadikan sebagai nilai alokasi memori yang dibutuhkan.

Segmen Program 4.4 Fungsi Pembuatan Node Baru

```

1:  struct ntt_node *ntt_node_create(const char *key) {
2:      char *node_key;
3:      struct ntt_node* node;
4:
5:      node = (struct ntt_node *) malloc(sizeof(struct
ntt_node));
6:      if (node == NULL) {
7:          return NULL;
8:      }
9:      if ((node_key = strdup(key)) == NULL) {
10:         free(node);
11:         return NULL;
12:     }
13:     node->key = node_key;
14:     node->timestamp = time(NULL);
15:     node->next = NULL;
16:     return(node);
17: }
```

Node pada NTT menyimpan tiga data, yaitu key sebagai identitas yang dapat dicari, timestamp yang mencatat waktu ketika node tersebut dibuat, dan next sebagai penunjuk node setelah node yang baru dibuat, yaitu NULL, karena node yang baru mempunyai tempat paling belakang. Tidak ada sesuatu yang berbeda dalam fungsi ini, hanya fungsi pembuatan node baru saja.

Segmen Program 4.5 Fungsi Penambahan Node Baru

```

1:  struct ntt_node *ntt_insert(struct ntt *ntt, const char *key,
    time_t timestamp) {
2:      long hash_code;
3:      struct ntt_node *parent;
4:      struct ntt_node *node;
5:      struct ntt_node *new_node = NULL;
6:
7:      if (ntt == NULL) return NULL;
8:
9:      hash_code = ntt_hashcode(ntt, key);
10:     parent     = NULL;
11:     node       = ntt->tbl[hash_code];
12:
13:     while (node != NULL) {
14:         if (strcmp(key, node->key) == 0) {
15:             new_node = node;
16:             node = NULL;
17:         }
18:
19:         if (new_node == NULL) {
20:             parent = node;
21:             node = node->next;
22:         }
23:     }
24:
25:     if (new_node != NULL) {
26:         new_node->timestamp = timestamp;
27:         new_node->count = 0;
28:         return new_node;
29:     }
30:
31:     /* Create a new node */
32:     new_node = ntt_node_create(key);
33:     new_node->timestamp = timestamp;
34:     new_node->timestamp = 0;
35:     ntt->items++;
36:
37:     /* Insert */
38:     if (parent) { /* Existing parent */
39:         parent->next = new_node;
40:         return new_node; /* Return the locked node */
41:     }
42:
43:     /* No existing parent; add directly to hash table */
44:     ntt->tbl[hash_code] = new_node;
45:     return new_node;
46: }

```

Pada fungsi pemasukan node yang baru pada NTT, ada serangkaian pengecekan yang dilakukan. Pertama kali, akan dilakukan pengecekan apakah ada sebuah nilai yang ada pada node yang dirujukan oleh key yang diberikan, jika

ternyata ada, maka nilai timestamp dan counter pada node tersebut akan diubah menjadi catatan waktu yang sekarang dan mengembalikan nilai counter menjadi nol. Setelah pengecekan tentang node yang telah ada, dilanjutkan dengan pengecekan node terakhir yang ada dalam NTT. Pembuatan node baru dilakukan setelah dua pengecekan tersebut dijalankan. Ketika ditemukanya node yang lain pada NTT, dalam segmen program 4.5 menyatakan bahwa node yang terakhir dalam NTT disebut dengan parent, maka next pada parent akan dirujukkan kepada node yang baru sehingga terbuatlah sebuah link pada kedua node tersebut. Jika tidak adanya node yang lain atau parent bernilai NULL, maka node yang baru akan diisikan kepada tbl yang dipunyai oleh root.

Segmen Program 4.6 Fungsi Pencarian Node

```

1:  struct ntt_node *ntt_find(struct ntt *ntt, const char *key) {
2:      long hash_code;
3:      struct ntt_node *node;
4:
5:      if (ntt == NULL) return NULL;
6:
7:      hash_code = ntt_hashcode(ntt, key);
8:      node = ntt->tbl[hash_code];
9:
10:     while (node) {
11:         if (!strcmp(key, node->key)) return(node);
12:
13:         node = node->next;
14:     }
15:     return((struct ntt_node *)NULL);
16: }
```

Proses pencarian pada NTT tidaklah terlalu rumit. Seperti yang terlihat pada segmen program 4.6, dalam proses pencarian node pada NTT dibutuhkan objek rootnya sendiri di mana diberikan parameter ntt untuk menrujukkan root yang sedang dipakai, nilai kembalian dari fungsi hash di mana yang dirujukkan pada fungsi hash adalah key dan root NTT, dan juga node yang ada pada tbl dari root. Ketika node dengan key yang dibutuhkan ditemukan, maka fungsi pencarian tersebut akan mengembalikan node yang ditemukan tersebut, jika tidak ditemukan maka fungsi pencarian akan mengembalikan sebuah nilai NULL yang akan menandakan bahwa node yang dicari tidak ada.

Segmen Program 4.7 Fungsi Untuk Menghapus NTT

```

1:  int ntt_destroy(struct ntt *ntt) {
2:      struct ntt_node *node, *next;
3:      struct ntt_c c;
4:
5:      if (ntt == NULL) return -1;
6:
7:      node = c_ntt_first(ntt, &c);
8:      while(node != NULL) {
9:          next = c_ntt_next(ntt, &c);
10:         ntt_delete(ntt, node->key);
11:         node = next;
12:     }
13:
14:     free(ntt->tbl);
15:     free(ntt);
16:     ntt = (struct ntt *) NULL;
17:
18:     return 0;
19: }

```

Pada segmen program 4.7 seperti judul yang diberikan, fungsi tersebut digunakan untuk menghapus NTT. Dengan proses penghapusan setiap node yang dipunya oleh NTT yang dilemparkan, proses untuk menghapus NTT pun dimulai. Setelah menghapus setiap anak node yang dipunyai oleh NTT yang dilemparkan, pelepasan memori data dari NTT yang dilemparkan pun dilakukan, agar memori tersebut dapat dipakai oleh data yang lain. Pelepasan memori ini harus dilakukan karena pada proses inisialisasi sebelumnya, NTT menempatkan alokasi data-datanya pada memori heap komputer server, bukan di dalam memori heap server. Untuk menghapus setiap anak node yang dipunya oleh NTT tersebut, dipanggilnya fungsi untuk menghapus per node anak, nama fungsi tersebut adalah `ntt_delete`.

Segmen Program 4.8 Proses Menghapus Satu Node Pada NTT

```

1:  int ntt_delete(struct ntt *ntt, const char *key) {
2:      long hash_code;
3:      struct ntt_node *parent = NULL;
4:      struct ntt_node *node;
5:      struct ntt_node *del_node = NULL;
6:
7:      if (ntt == NULL) return -1;
8:
9:      hash_code = ntt_hashcode(ntt, key);
10:     node      = ntt->tbl[hash_code];
11:

```

Segmen Program 4.8 (lanjutan)

```

12:     while (node != NULL) {
13:         if (strcmp(key, node->key) == 0) {
14:             del_node = node;
15:             node = NULL;
16:         }
17:
18:         if (del_node == NULL) {
19:             parent = node;
20:             node = node->next;
21:         }
22:     }
23:
24:     if (del_node != NULL) {
25:
26:         if (parent) {
27:             parent->next = del_node->next;
28:         } else {
29:             ntt->tbl[hash_code] = del_node->next;
30:         }
31:
32:         free(del_node->key);
33:         free(del_node);
34:         ntt->items--;
35:
36:         return 0;
37:     }
38:
39:     return -5;
40: }

```

Setelah berbicara untuk menghapus semua node beserta NTT itu sendiri, dalam segmen program 4.8 memperlihatkan proses untuk menghapus sebuah node saja pada NTT. Fungsi yang membutuhkan dua buah parameter, yaitu NTT yang ingin salah satu nodenya dihapus, dan parameter kedua adalah key dari node yang akan dihapus dari NTT yang telah dilemparkan. Seperti proses penghapusan sebelumnya, adanya proses untuk pelepasan alokasi memori pun harus dilakukan dengan sebab alokasi memori dari node NTT ini berada pada memori heap komputer server.

4.4 Pemeriksaan Request Baru

Pada bagian ini akan dijelaskan mengenai proses pemeriksaan proses dari request yang masuk. Dengan proses awal mencocokkan pada dua buah tempat penyimpanan, modul evasive memulai proses pemeriksaannya.

Segmen Program 4.9 Pemeriksaan Alamat IP Tahap 1

```

1: ... access_checker
2: /* Check whitelist */
3: if (is_whitelisted(r->connection->remote_ip))
4: return OK;
5:
6: /* First see if the IP itself is on "hold" */
7: n = ntt_find(hit_list, r->connection->remote_ip);
8:
9: if (n != NULL && t-n->timestamp<blocking_period) {
10:
11:     /* If the IP is on "hold", make it wait longer in 403 land
        */
12:     ret = HTTP_FORBIDDEN;
13:     n->timestamp = time(NULL);
14: }
15: ... access_checker

```

Segmen program 4.9 memperlihatkan proses awal untuk memeriksa alamat IP yang masuk. Pada baris ketiga proses di atas adalah proses pemeriksaan alamat IP yang masuk pada tabel yang berisi alamat-alamat IP yang berstatus diboleh masuk dan mengakses server. Jika alamat IP yang masuk ada dalam tabel tersebut, maka sistem akan langsung memperbolehkan alamat IP tersebut mengakses server secara bebas tanpa perijinan tambahan apa pun. Berlanjut pada baris kode ketujuh adalah proses pencarian alamat IP yang masuk apakah ada dalam NTT atau tidak. Pada baris ke 11 dari proses di atas, adanya proses pengecekan apakah data yang dicari ada atau tidak, dan apakah waktu yang ada dalam data tersebut masih valid atau tidak.

Segmen Program 4.10 Pemeriksaan Alamat IP Tahap 2

```

1: ... access_checker
2: else {
3:
4:     /* Has URI been hit too much? */
5:     snprintf(hash_key, 2048, "%s_%s", r->connection->remote_ip,
        r->uri);
6:     n = ntt_find(hit_list, hash_key);

```

Segmen Program 4.10 (lanjutan)

```

7:   if (n != NULL) {
8:
9:       /* If URI is being hit too much, add to "hold" list and
403 */
10:      if (t-n->timestamp<page_interval && n->count>=page_count)
11:      {
12:          ret = HTTP_FORBIDDEN;
12:          ntt_insert(hit_list,          r->connection->remote_ip,
time(NULL));
13:      } else {
14:
15:          /* Reset our hit count list as necessary */
16:          if (t-n->timestamp>=page_interval) {
17:              n->count=0;
18:          }
19:      }
20:      n->timestamp = t;
21:      n->count++;
22:  } else {
23:      ntt_insert(hit_list, hash_key, t);
24:  }
25:
26:  /* Has site been hit too much? */
27:  snprintf(hash_key,      2048,      "%s_SITE",      r->connection-
>remote_ip);
28:  n = ntt_find(hit_list, hash_key);
29:  if (n != NULL) {
30:
31:      /* If site is being hit too much, add to "hold" list and
403 */
32:      if (t-n->timestamp<site_interval && n->count>=site_count)
33:      {
34:          ret = HTTP_FORBIDDEN;
34:          ntt_insert(hit_list,          r->connection->remote_ip,
time(NULL));
35:      } else {
36:
37:          /* Reset our hit count list as necessary */
38:          if (t-n->timestamp>=site_interval) {
39:              n->count=0;
40:          }
41:      }
42:      n->timestamp = t;
43:      n->count++;
44:  } else {
45:      ntt_insert(hit_list, hash_key, t);
46:  }
47:  }
48: ... access_checker

```

Pada proses penanganan alamat IP yang baru datang, proses pertama yang dilakukan adalah mencari alamat IP tersebut pada tabel `hit_list` dapat dilihat

pada baris kode keenam berlanjut ke baris kode ketujuh. Setelah pencarian pada tabel `hit_list`, dilakukannya proses pemeriksaan apakah alamat IP tersebut telah mengakses server berapa kali, apakah melebihi batas diberikan atau tidak, proses pemeriksaan tersebut dapat dilihat pada baris kode ke 10. Jika alamat IP tersebut telah melewati batas jumlah pengaksesan, maka alamat IP tersebut akan dimasukkan pada tabel `hit_list` dan proses tersebut akan langsung menemui akhirnya dengan nilai kembalian `HTTP_FORBIDEN`.

BAB V

ANALISA DESAIN SISTEM KILL BOTS

Pada bab ke lima ini, sesuai dengan judulnya, akan dijelaskan tentang desain sistem dari Kill Bots. Penjelasan yang akan dijelaskan pada bab ini mulai dengan pengertian umum dari apa itu Kill Bots. Setelah pengertian umum dari Kill Bots tersebut, mulai akan dijelaskannya proses apa saja yang terjadi dalam sistem tersebut. Metode-metode dan struktur yang dipakai juga akan dijelaskan pada bab ke lima ini.


5.1 Pengertian Kill Bots

Kill Bots adalah sebuah sistem yang merupakan kernel module modifikasi yang ada pada sebuah web server. Fungsi dari Kill Bots itu sendiri adalah untuk menangani serangan Distributed Denial of Service atau yang sering disingkat dengan DDoS. Seperti yang telah dijelaskan sebelumnya, DDoS adalah sebuah serangan yang penyerangan yang memakai lebih dari sebuah komputer yang digunakan untuk menyerang. Serangan tersebut merupakan serangan yang terpusat kepada sebuah korban dan juga dikoordinasi oleh sebuah komputer sebagai otak dari serangan tersebut.

Sistem Kill Bots ini menangani serangan DDoS yang dinamakan Flash Crowd DDoS. Serangan ini adalah sebuah serangan yang bersifat Resource Depletion, di mana korban akan disibukkan dengan request-request yang datang dari penyerang secara terus menerus dan mempunyai skala yang besar. Serangan ini menyerupai request-request normal pada umumnya. Sehingga membuat sistem Kill Bots ini seperti sebuah *load balancer* pada web server. Load balancer adalah sebuah sistem yang digunakan untuk menyeimbangkan load server. Load balancer sering digunakan ketika sebuah web server mempunyai angka *traffic* yang cukup tinggi, sehingga load yang dipakai akan diseimbangkan sesuai dengan kemampuan server saat itu.

Module yang serupa terdapat pada perusahaan yang saat ini menjadi perusahaan yang paling dikenal dalam dunia internet dan telah menjadi brand pencarian dalam lingkungan sekitar, yaitu Google. Dipastikan perusahaan sebesar google tidak mungkin hanya menggunakan modul yang sangat sederhana ini, tetapi jika dilihat sepintas, google juga mempunyai konsep yang sama.

To continue, please type the characters below:



About this page

Our systems have detected unusual traffic from your computer network. This page checks to see if it's really you sending the requests, and not a robot. [Why did this happen?](#)

IP address: 202.91.25.41
 Time: 2013-05-17T16:15:32Z
 URL: <http://www.google.com/search?q=check&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:en-US:official&client=firefox-beta>

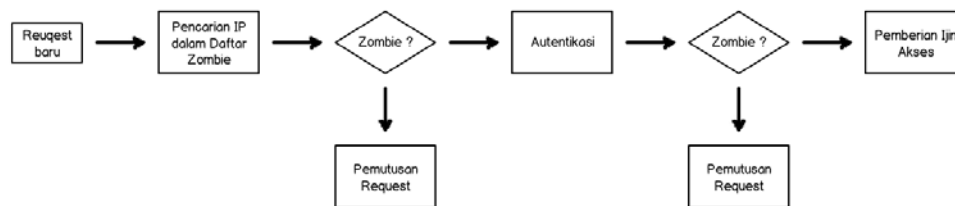
Gambar 5.1
Graphical Tes dari Google

Penanganan load server yang dipunyai google, jika mendeteksi adanya kelakuan dari request yang masuk berdasarkan load server dari google saat itu berbau adanya serangan, maka google akan melemparkan sebuah tes yang berupa gambar captcha yang harus dijawab oleh pengguna yang bertujuan untuk memetakan apakah request yang masuk tersebut adalah request yang benar atau tidak.

5.2 Sistem Kill Bots

Kill Bots mempunyai sistem yang hampir sama dengan sistem load balancer. Kalkulasi utama yang dilakukan adalah kalkulasi tentang load server. Pengertian dari load server itu sendiri adalah persentase dari berapa banyak

service atau daemon (sinonim kata dari service yang sering digunakan dalam sistem operasi Linux) yang sedang bekerja. Load server adalah hal yang berbeda dengan load dari CPU atau yang sering disebut dengan CPU *usage*. Terkadang banyak yang mengartikan sama dengan hal tersebut.



Gambar 5.2
Flow Diagram Kill Bots

Sistem yang dipakai oleh Kill Bots adalah sebuah sistem yang cukup sederhana sehingga jika dilihat secara sepintas maka akan mudah dimengerti. Kesederhanaan dari Kill Bots mempunyai keunggulan tertentu, yaitu proses yang dipakai menjadi minimum. Proses yang minimum dan cepat itulah inti dari pengembangan web server untuk bekerja lebih efisien dan cepat dalam menangani request-request yang masuk.

Gambar 5.1 menggambarkan tentang alur kerja dari sistem Kill Bots dengan permulaan proses adalah dengan pencarian pada daftar zombie di mana pencarian didasarkan oleh alamat IP dari request yang masuk. Daftar zombie merupakan gabungan dari dua buah komponen, yaitu filter yang digunakan untuk mengurangi proses pencarian yang langsung diproses dalam sebuah tempat penyimpanan, dalam tugas akhir ini filter yang akan digunakan adalah *Bloom Filter*, dan tempat penyimpanan itu sendiri yang dalam tugas akhir ini tempat penyimpanan yang dipakai adalah hash table yang telah disediakan oleh Apache. Alamat IP yang masuk dan akan dicari dalam daftar zombie tidak dengan begitu saja dicari dalam daftar zombie, tetapi sebelumnya, proses *hashing* akan dilakukan pada alamat IP tersebut dan kemudian dilakukan prosedur pencarian dalam daftar zombie tersebut. Proses hashing tersebut menandakan bahwa data-data yang ada pada daftar zombie tersebut ada data hasil proses hashing yang mulanya adalah sebuah

alamat IP. Jika alamat IP yang masuk ternyata ada dalam daftar zombie, maka server akan mengirimkan sebuah sinyal di mana request tersebut tidak dapat diteruskan ke tujuan request tersebut atau dalam bahasa singkatnya, request tersebut akan dihentikan untuk mendapatkan tujuan semulanya. Penghentian request tersebut tidak semudah memotong sebuah tali. Dalam lifecycle dari Apache, tidak adanya prosedur yang dapat memotong request yang masuk, tetapi Apache mempunyai prosedurnya sendiri, yaitu mengirimkan sinyal kepada setiap fasenya untuk tidak memproses request tersebut lebih lanjut dan terakhir pada bagian content handler, sinyal tersebut akan mengembalikan sebuah nilai pada sumber request sebuah kode status html, contohnya kode status 4xx. Kode status 4xx adalah kode status mempunyai arti client error, di mana kesalahan ada pada client. Pada tugas akhir ini kode status untuk menghentikan request agar tidak diproses lebih lanjut adalah 403, yaitu *Forbidden*.

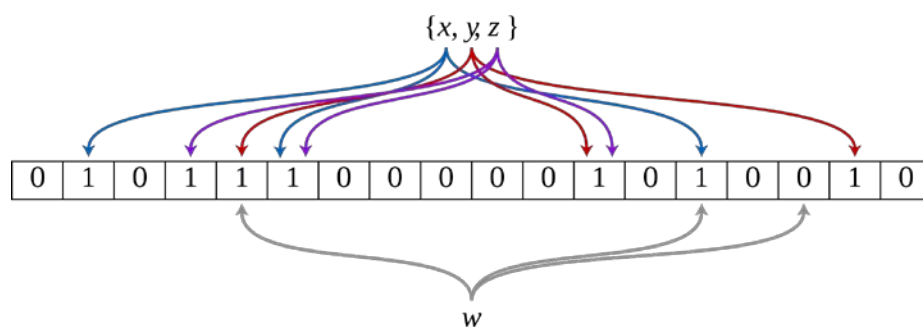
Pada proses selanjutnya, yaitu akan dilakukan proses autentikasi terhadap request yang masuk tersebut. Autentikasi adalah nama sebuah proses yang mana di dalam proses tersebut terdapat beberapa proses kecil yang bertujuan untuk mengidentifikasi apakah request tersebut layak untuk mendapatkan izin untuk melakukan prosesnya secara normal atau tidak. Jika tidak, sama seperti sebelumnya, akan dilakukan prosedur untuk menutup request tersebut dengan tanpa melakukan proses lebih lanjut pada request tersebut.

5.3 Daftar Zombie

Bagian 5.3 ini akan menjelaskan tentang komponen-komponen apa saja yang digunakan dalam sistem Kill Bots dalam pencarian alamat IP untuk pertama kalinya dalam daftar zombie. Seperti yang telah dijelaskan pada bagian sebelumnya, daftar zombie mempunyai dua buah komponen yang menunjang pencarian sebuah alamat IP, yaitu sebuah filter yang bernama Bloom Filter, dan tempat penyimpanan itu sendiri yang berupa hash table di mana hash table tersebut telah disediakan oleh Apache.

5.3.1 Bloom Filter

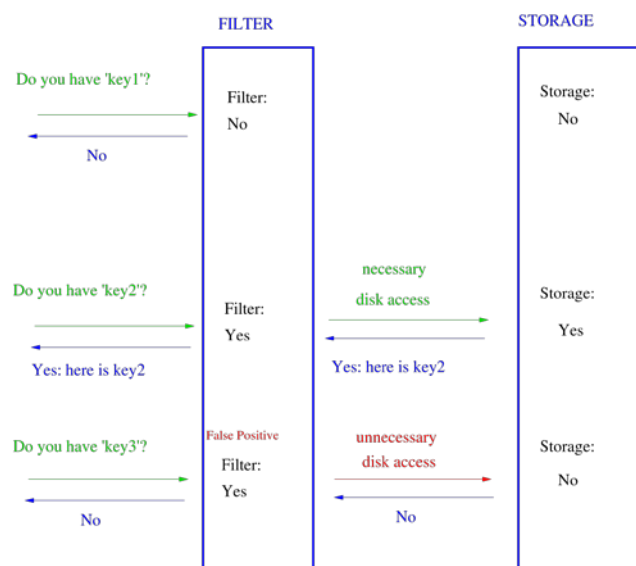
Pada tahun 1970, Burton Howard Bloom menciptakan temuannya tentang sebuah struktur data yang baru, di mana struktur data tersebut sering dikenal dengan *space-efficient probabilistic* yang bertujuan untuk mengetahui apakah sebuah elemen ada dalam sebuah kumpulan elemen (set) atau tidak. Dalam Bloom Filter, sesuai dengan sifat yang dipunyainya, yaitu probabilistic atau artian dalam bahasa Indonesia adalah probabilitas, mempunyai dua nilai yang dikembalikan, yaitu *False Positive*, dan *False Negative*. *False Positive* adalah nilai kembalian yang menandakan bahwa sebuah elemen yang dicari ada dalam set tersebut. *False Negative*, sesuai nama belakangnya, nilai kembalian tersebut menandakan bahwa elemen yang dicari tidak ada dalam set. Pada inti dari struktur data ini, yaitu probabilitas, membuat *False Positive* menjadi sesuatu yang tidak pasti atau mungkin. Jika dalam sebuah set tersebut terdapat elemen dengan jumlah yang besar atau mendekati batas penyimpanan dari filter itu sendiri, maka kemungkinan untuk mendapatkan nilai kembalian *False Positive* pun akan menjadi lebih besar, sehingga membuat Bloom Filter menjadi tidak efisien.



Gambar 5.3
Skema Contoh Bloom Filter

Fungsi yang ada pada Bloom Filter hanya berjumlah dua buah, yaitu fungsi query/test dan fungsi add. Fungsi query adalah fungsi yang digunakan untuk mencari sebuah elemen apakah ada dalam kumpulan elemen yang sedang ditunjuk atau set. Sedangkan untuk fungsi add adalah fungsi yang digunakan untuk memasukkan sebuah elemen pada kumpulan elemen yang telah dibuat atau disediakan.

Bloom filter, selayaknya sama seperti struktur data yang lain, mempunyai komponen-komponen penyusun. Dalam penyusunan Bloom filter, *hash function* adalah sebuah komponen penyusun yang sangat penting atau dapat dikatakan dengan komponen utama. Banyaknya hash function yang digunakan pada Bloom filter ini dapat lebih dari satu hash function, tergantung dengan kebutuhan dari berapa banyak data yang ingin disimpan dalam Bloom filter. Pada tugas akhir ini, banyaknya hash function yang dipakai adalah sebanyak dua buah, yaitu RS hash dan JS hash. Kedua hash function tersebut termasuk hash function sederhana, di mana proses yang digunakan adalah proses matematika dan pergeseran bit semata. Proses yang sederhana dan tidak memakan sumber daya yang banyak inilah yang cocok untuk digunakan dalam lingkup proses yang besar.



Gambar 5.4
Skema Proses Bloom Filter
Terhadap Tempat Penyimpanan

Kedua fungsi yang dipunyai oleh Bloom filter, kenyataannya di dalamnya, mereka mempunyai proses yang hampir sama. Seperti yang telah dijelaskan, Bloom filter mempunyai komponen utama dalam hal hash function, dengan komponen tersebut, maka Bloom filter ini mempunyai langkah untuk menambahkan elemen ke dalam sebuah kumpulan elemen sebagai berikut,

1. Melakukan proses hashing sebanyak hash function yang dimiliki.

2. Setiap hasil kembalian proses hashing tersebut akan dijadikan index untuk menunjuk sebuah elemen dalam bit array.
3. Elemen yang ditunjuk tersebut, nilainya akan dirubah menjadi 1.

Sebagai visualisasi langkah-langkah tersebut, dapat dilihat pada gambar 5.2. proses penambahan/add, dan proses query/pencarian mempunyai kemiripan proses di dalamnya. Pada proses pencarian, langkah ke tiga akan diubah menjadi memeriksa apakah nilai dari elemen yang ditunjuk tersebut adalah 1. Jika tidak, maka proses akan dihentikan dan fungsi pencarian akan mengembalikan nilai False Negative secara langsung tanpa menunggu proses lain dalam Bloom filter untuk dijalankan.

Pada dunia praktisnya, Bloom filter dipakai dalam beberapa proses yang membutuhkan proses yang sangat cepat. Beberapa mempunyai lingkup dalam internet dan ada juga yang tidak. Proses-proses tersebut adalah sebagai berikut,

- *Web Cache Sharing*⁵. Penggunaan Bloom filter dalam sebuah web cache adalah untuk merepresentasikan kumpulan dari data cache yang disimpan secara local di dalam client. Setiap cache, dalam periode tertentu akan mengirimkan hasil laporannya masing-masing ke cache yang lain dalam lingkup yang sama. Menggunakan semua laporan yang diterima, sebuah node dari cache akan mempunyai gambaran dari kumpulan data yang tersimpan dalam semua cache. Contoh dari aplikasi ini adalah *Squid Web Proxy Cache* menggunakan proses yang dinamakan “cache digest” berdasarkan sistem tersebut.
- *Query filtering and routing*⁶. Bloom filter digunakan sebagai sebuah hasil laporan untuk kumpulan pelayanan yang ditawarkan dari sebuah node. Query adalah sebuah deskripsi untuk pelayanan tertentu, juga direpresentasikan sebagai Bloom filter. Dengan demikian, setiap node yang menerima data atau yang mengirimkan data, mempunyai cukup

⁵ <http://www.squid-cache.org/>

⁶ Hodes, T.D., Czerwinski, S.E., Zhao, B.Y., Joseph, A.D. and Katz, R.H. An Architecture for Secure Wide-Area Service Discovery. *Wireless Networks*.

data untuk dikirimkan ke node sebelum dan sesudahnya. Aplikasi dari sistem ini adalah *OceanStore*.

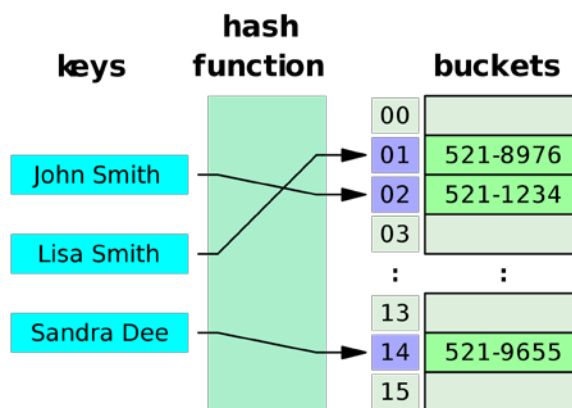
- *Compact representation of a differential file*⁷. Sebuah data deferensial mengandung sekumpulan database data yang harus selalu diperbaharui. Alasan performa, database akan diperbaharui pada tengah malam atau ketika data deferensial berkembang melebihi batas yang telah ditentukan. Integritas adalah objek yang harus dicapai oleh sistem ini, sehingga setiap query atau referensi yang melayang ke database harus melihat data deferensial terlebih dahulu untuk mengetahui apakah data tersebut dijadwalkan untuk diperbaharui atau tidak. Bloom filter digunakan untuk menaikkan kecepatan proses kerja dari proses ini dengan cara data deferensial tersebut direpresentasikan sebagai Bloom filter itu sendiri.
- *Free Text Searching*⁸. Pada dasarnya, kumpulan kata-kata yang tampak dalam sebuah teks adalah representasi dari Bloom filter yang dibungkus secara ringkas.

5.2.2 Hash Table

Hash table atau nama lain yang cukup dikenal adalah hash map, adalah sebuah struktur data yang mana digunakan untuk membuat sebuah array asosiatif. Array asosiatif, sesuai nama depannya, komponen ini mempunyai kemiripan dengan kebanyakan array yang ada. Perbedaannya adalah pada array asosiatif menggunakan key sebagai index dari data yang akan didapatkan. Key tersebut biasanya bertipe teks atau string. Hash table menggunakan bantuan hash function untuk mengkalkulasi indek yang akan didapatkan sebagai penunjuk data yang ada di dalam tempat penyimpanan atau *bucket*, sehingga akan dapat di ambil nilainya dengan menggunakan key tersebut.

⁷ Mullin, J.K. A second look at Bloom filters. Communications of the ACM, 26 (8). 570-571.

⁸ Ramakrishna, M.V. Practical performance of Bloom filters and parallel free-text searching. Communications of the ACM, 32 (10). 1237-1239.



Gambar 5.5
Contoh Proses Hash Table

Pada gambar 5.5 terlihat contoh proses dari hash table. Gambar 5.5 memperlihatkan proses sederhana hash table dari buku telepon. Seperti yang dapat dilihat, setelah menerima sebuah key, hash table akan menggunakan fungsi dari hash function untuk melakukan kalkulasi indek dari data yang ada berdasarkan key yang diberikan. Indeks tersebut akan dicocokkan pada *bucket* tempat penyimpanan data, jika bucket yang ditunjuk kosong, maka akan dikembalikannya nilai NULL.

Hash table pada sistem Kill Bots ini mempunyai peran untuk menyimpan data dari request-request yang masuk di mana request tersebut gagal menjawab tes yang telah diberikan dari server karena kondisi server yang menandakan bahwa server dalam kondisi siaga. Sebelum pencarian langsung ke dalam hash table, pencarian mula-mula dilakukan pada Bloom filter. Dengan proses yang cepat dan akurat, jika ditemukannya data pada Bloom filter, maka akan dilanjutkan pengaksesan pada hash table, sehingga proses pada hash table menjadi minimal dan kembali lagi, proses pencarian menitik beratkan pada Bloom filter. Hash table sendiri hanya digunakan untuk tempat penyimpanan, dikarenakan pada dasarnya, Bloom filter adalah sebuah struktur data yang digunakan untuk mengindek data-data yang ada, sehingga dalam proses yang besar, tidak diperlukannya pengaksesan kepada tempat penyimpanan secara berlebih.

Pada praktiknya, memang penyimpanan dengan menggunakan hash table dan bloom filter sebagai garis depannya, menciptakan proses yang maksimal dengan proses yang minimal, sehingga membuat kolaborasi tersebut menuai waktu proses yang sangat cepat. Di mana ada kelebihan, di sanalah terdapat kelemahan. Dalam proses pembuatan tugas akhir ini, tidak ditemukannya cara untuk menyimpan data-data yang masuk secara *persistent* atau terus menerus. Dalam jangka waktu tertentu, data-data yang ada dalam bloom filter dan hash table akan terbangun atau tertumpuk dengan data yang lain. Masalah ini telah dikaji sedemikian rupa sehingga mendapatkan dua buah hipotesa, yaitu dikarenakan bloom filter menggunakan tipe status array yang mana alokasi array tersebut berada pada jalur memori dari Apache, sehingga mempunyai kemungkinan data tersebut tertumpuk dengan data dari Apache, dan hipotesa yang kedua adalah memori yang telah dialokasi untuk bloom filter mengikuti pola pembersihan memori dari Apache, sehingga membuat isi dari bloom filter terhapus. Masalah ini belum dapat dibuktikan lebih lanjut karena tidak adanya sumber yang dapat dijadikan referensi dan panutan untuk mengkaji masalah tersebut lebih dalam.

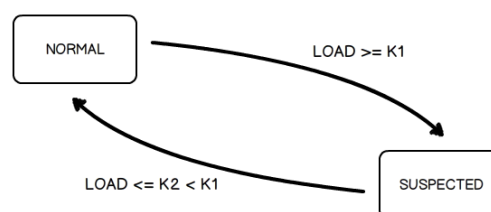
Sebelumnya, ada yang mengatakan bahwa jika ingin mendapatkan sebuah struktur data yang persistent dan tidak menyangkutpautkan dengan proses kerja Apache adalah dengan cara mengalokasikan memori tersebut ke bagian *heap* sistem komputer. Setelah dilakukan ujicoba, memang data yang ada di dalam struktur data yang alokasi memorinya diarahkan pada heap sistem komputer mempunyai data yang persistent, dan tidak adanya kehilangan data selama komputer tidak mati, tetapi proses untuk menuliskan data ke dalam memori tersebut memakan waktu yang cukup lama. Dibutuhkan waktu selama 10 request yang masuk dengan rate request sebesar 100 request per detik. Waktu yang diperlukan ini cukup lama jika berbicara proses dalam server. Dengan demikian proses alokasi yang digunakan adalah alokasi yang berdasarkan pada Apache, karena proses yang dilakukan membutuhkan waktu yang sangat cepat walaupun dalam jangka waktu tertentu data tersebut dapat hilang.

5.4 Autentikasi

Sistem Kill Bots menggunakan proses autentikasi setiap request yang masuk ketika sebuah server dalam status siaga, atau dalam sistem ini dinamakan mode *SUSPECTED*. Dalam Kill Bots sendiri terdapat dua buah mode yang menandakan status dari server dan masing-masing mempunyai proses yang berbeda. Seperti yang telah dikatakan salah satunya, mode tersebut adalah *SUSPECTED* dan *NORMAL*. Proses pembagian status server ini dimasukkan pada sebuah bagian sendiri, yaitu bagian *Stages*. *Stages* ini adalah proses yang dapat dikatakan paling penting dalam autentikasi serangan DDoS ini.

5.4.1 Stages

Proses stages ini mengembalikan dua buah status yang mana status ini akan menandakan dari performa server. Kedua status tersebut adalah *SUSPECTED* dan *NORMAL*. Sesuai dengan namanya, *NORMAL* adalah status server yang menandakan bahwa server berada pada performa yang baik di mana server dapat melayani request-request yang datang secara normal dan tidak ada kendala apa pun. Sebaliknya, pada status server *SUSPECTED* menandakan bahwa server mempunyai kondisi yang sangat sibuk sehingga untuk kedepannya mungkin akan ada kesalahan proses atau koneksi yang terbentuk. Setelah server menandakan bahwa sekarang adalah berstatus *SUSPECTED*, maka akan diikuti serangkaian proses untuk mengambil alih request-request yang datang.



Gambar 5.6
Transisi Status Server

Proses pembedaan status server di sini menggunakan sebuah fungsi yang mana menggunakan kalkulasi sederhana. Kalkulasi yang digunakan berdasarkan oleh load dari server. Load server ini berbeda dengan CPU usage yang ada dalam

komputer. Kadang, beberapa sumber mengatakan bahwa load server sama dengan load CPU atau CPU usage, dalam kasus ini, hal tersebut cukup berbeda. Load server sendiri adalah kumpulan service atau daemon yang tersisa atau kosong dalam waktu tersebut.

Contohnya, jika sebuah server mempunyai jumlah maksimal daemon untuk melayani request-request yang datang sebesar 256, pada saat kalkulasi sedang dilakukan, terdapat 100 request yang masuk dan sedang diproses, dengan demikian menyisakan 156 daemon dalam keadaan *idle ready*. Load server yang sekarang didapatkan dari kalkulasi pembagian antara daemon yang sedang melakukan proses terhadap request yang datang dengan jumlah maksimal daemon yang dapat menangani request dalam server, dengan demikian didapatkan nilai 0.39, dan load server merupakan nilai presentase, maka nilai tersebut dikalikan 100, sehingga didapatkan nilai 39% untuk load server.

Pada awal mula server dijalankan, server akan berstatus NORMAL dengan tingkat load yang sangat rendah atau dapat dikatakan hampir nol karena pada kondisi tersebut server baru saja dijalankan. Setelah beberapa lama, dengan meningkatnya load dari server sehingga load dapat melebihi batas yang telah diberikan, yaitu K1, maka status server akan terganti dengan status SUSPECTED.

Untuk kembali ke status NORMAL, load harus mempunyai nilai kurang dari K2, di mana K2 bernilai lebih rendah dari pada K1. Gambar 5.6 menggambarkan transisi dari status server. Dalam tugas akhir ini, penilaian variabel K1 dan K2 adalah berdasar percobaan dan logika dasar. Nilai untuk variabel K1 adalah sebesar 70% dan untuk K2 adalah sebesar 50%.

Jika penyerang mempunyai ide untuk mengganti status server ini terus menerus, dari NORMAL ke SUSPECTED dan sebaliknya, proses tersebut tidaklah menguntungkan sama sekali, karena penentuan status tersebut memakan sumber daya yang sangat kecil sekali, hanya ada pengambilan load server saat ini dan akhirnya ditunjuk status mana yang cocok untuk kondisi load yang telah didapat.

5.4.1.1 Stage SUSPECTED: CAPTCHA-Based Authentication

Setiap request yang masuk ke dalam server ketika status server adalah SUSPECTED, akan diberikan sebuah *graphical test* atau *captcha* di mana pengguna harus menjawab pertanyaan dari tes tersebut untuk melanjutkan proses untuk mendapatkan data dari server secara normal kembali. Pemberian captcha ini melepaskan pengguna dari perlakuan yang langsung ke arah data dari server, sehingga dalam hal ini server belum melemparkan data apa pun yang ada. Captcha itu sendiri adalah hasil pembuatan dari Kill Bots. Pembuatan captcha ini berlangsung ketika server sedang dijalankan untuk pertama kali atau nama lain dari proses ini adalah start-up, maka ketika start-up server telah selesai, captcha akan sudah siap untuk diakses secara langsung tanpa harus membuat kembali.

Our Website is Experiencing Unusually High Load.

To Restrict Automated Access We Require Code Verification.

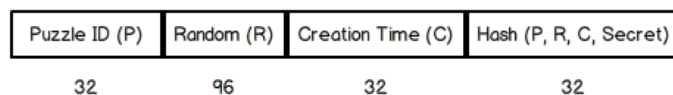
Please Enter The Code (Number Only) Below.

D0N188K5G2

Gambar 5.7
Contoh Captcha Kill Bots

Jika sebuah pengguna yang sah sedang mencoba untuk mengakses server, di mana server sedang berada pada status SUSPECTED, kemudian pengguna tersebut dapat menjawab captcha yang telah diberikan, akan tidak baik jika pengguna yang telah menjawab dengan benar sebuah tes yang diberikan dan setelah itu pengguna tersebut mencoba mengakses server untuk kedua kalinya, dan diminta untuk menjawab sebuah captcha kembali. Kill Bots menggunakan sistem *one test per session*. Sistem tersebut memberikan kelonggaran pada pengguna yang telah dapat menjawab tes yang diberikan kepada mereka dengan memberi sebuah token yang berupa sebuah cookie yang menandakan bahwa pengguna tersebut dapat mengakses server tanpa harus diberikan tes sebelumnya.

Perijinan tersebut mempunyai batas waktu yang telah ditentukan. Dalam tugas akhir ini perijinan tersebut bernilai 30 menit.



Gambar 5.8
Segmen Token

Berbicara tentang perijinan, berarti menandakan bahwa adanya pengamanan data agar data tersebut tidak dapat dibuat ulang dari bagian pengguna. Token yang mana menandakan perijinan tersebut terbagi menjadi empat bagian, seperti yang telah digambarkan pada gambar 5.7, yaitu segmen token. Terlihat bahwa segmen pertama pada token adalah *puzzle id*. Puzzle id ini merepresentasikan puzzle yang diberikan kepada pengguna untuk dijawab. Sebelah kanan dari puzzle id adalah sebuah rangkaian karakter *random*. Rangkaian karakter random pada Kill Bots ini berupa karakter angka. Selanjutnya, pada kolom ketiga terdapat *creation time* yang menandakan kapan token ini dibuat. Kolom terakhir, yaitu kolom yang memerlukan ketiga segmen sebelumnya dan sebuah *secret key* yang ada pada server. Kolom ini akan membungkus keempat komponen tersebut dengan menggunakan fungsi RS hash. RS hash akan mengembalikan nilai berupa rangkaian angka dan digabungkan dengan ketiga komponen sebelumnya dan terbentuklah sebuah token yang berfungsi menjadi sebuah perijinan agar pengguna yang dapat menjawab tes yang diberikan tidak mendapat tes kembali.

Pengguna yang mengakses server dan dalam akses tersebut ada token yang dimaksudkan, maka server akan memproses token tersebut terlebih dahulu sebelum ada proses yang lebih lanjut ke dalam server. Proses yang pertama kali dilakukan adalah memeriksa apakah hash yang ada pada token tersebut sama dengan hash yang dikalkulasi dengan berdasarkan komponen-komponen dalam token tersebut dan komponen secret key yang ada dalam server. Setelah cocok, maka pemeriksaan berlanjut pada creation time. Pada proses ini terdapat dua buah alur. Sebelum berlanjut kepada penjelasan alur, perlu diketahui bahwa token diselipkan pada captcha yang diberikan kepada pengguna. Alur pertama, yaitu

ketika pengguna menjawab sebuah captcha. Pengguna diberikan waktu sebanyak empat menit untuk menjawab captcha yang telah diberikan. Jika pengguna menjawab lebih dari waktu yang diberikan, meski pun jawaban yang diberikan tepat, pengguna tersebut akan diberikan captcha dengan isi yang berbeda. Berlanjut pada alur yang kedua, yang mana alur kedua ini telah dijelaskan sebelumnya. Alur kedua ini adalah ketika pengguna telah menjawab benar tes yang diberikan. Pada tahap ini token tersebut tersimpan pada cookie pengguna. Pengguna yang telah menjawab benar tes, berhak mengakses secara bebas server tersebut tanpa harus diberikan tes ulang. Kebebasan ini mempunyai batas waktu tertentu, yaitu 30 menit. Terlepas dari waktu tersebut, pengguna akan diberikan tes ulang ketika mengakses server untuk lebih lanjut.

5.4.1.2 Stage SUSPECTED: Pengguna yang Tidak Menjawab CAPTCHA

Mekanisme yang menitik beratkan pada captcha, mempunyai dua buah kerugian. Pertama, penyerang akan membuat server untuk terus memberikan captcha agar server selalu dalam keadaan sibuk. Kedua dan juga poin yang cukup penting adalah bagi para pengguna yang tidak dapat menjawab captcha yang diberikan, maka pengguna tersebut tidak dapat mengakses server sama sekali.

Poin-poin kerugian tersebut dapat diselesaikan dengan mengidentifikasi perilaku dari kedua belah pihak ini. Pada pihak pengguna, terdapat dua poin yang sering mereka lakukan pada kondisi mereka tidak dapat menjawab captcha tersebut. Poin pertama, pengguna tersebut akan menjawab captcha yang diberikan setelah beberapa kali me-reload halaman web yang berisi tes tersebut. Pada poin kedua, pengguna menyerah dalam menjawab tes yang diberikan dan meninggalkan tes tersebut dalam kurun waktu tertentu. Perilaku ini terjadi dikarenakan pengguna tersebut telah menjoba untuk menjawab tes yang diberikan beberapa kali tetapi jawaban yang diberikan tersebut adalah salah. Pada perilaku yang dilakukan oleh penyerang juga mempunyai dua buah poin yang dapat menandakan bahwa perilaku tersebut adalah perilaku dari penyerang. Pertama, penyerang mungkin akan mengikuti perilaku dari pengguna pada poin pertama,

dan meninggalkan tes tersebut setelah beberapa kali mencoba, dalam poin ini, tidak adanya perilaku yang mencurigakan, karena ketika penyerang meninggalkan tes, maka penyerang tersebut menandakan bahwa serangan telah dihentikan. Poin kedua, penyerang akan mengirimkan request terus menerus, meski pun mereka tidak dapat menjawab tes yang diberikan. Berdasarkan poin kedua yang ada pada pihak penyerang, membuat penyerang sangat terlihat bahwa mereka mempunyai keinginan untuk menyerang server.

Sebuah batasan diberikan dengan dasar dari poin-poin yang telah didapatkan dari kedua belah pihak. Batasan tersebut memerlukan bantuan dari bloom filter dan hash table untuk menanganinya. Kesalahan dalam proses menjawab sebuah tes akan dicatat kedalam hash table secara terus menerus. Pemakaian bloom filter dalam kondisi ini adalah untuk mempercepat proses dan juga meminimalisasikan proses yang harus dikerjakan. Setiap request yang masuk, berdasarkan alamat IP mereka akan diberikan sejumlah kesempatan atas kesalahan menjawab. Ketika alamat IP yang mengirim request melebihi batas kesalahan yang diberikan, maka request untuk alamat IP tersebut akan dihentikan, dan tidak adanya tes yang diberikan. Hash table dan bloom filter akan dikosongkan ketika server masuk kepada status NORMAL, di mana server dapat bekerja melayani request-request yang masuk dengan maksimal.

BAB VI

IMPLEMENTASI SISTEM KILL BOTS PADA APACHE

Bab VI, sesuai dengan judul babnya, akan mengulas tentang implementasi dari sistem Kill Bots. Pembahasan implementasi meliputi tentang penjelasan fungsi-fungsi yang dipakai, bagaimana sebuah objek atau struktur data diinisialisasi, baris-baris kode yang ada dalam pembuatan Kill Bots yang lain, dan juga tentang masalah-masalah pembungkusan sistem Kill Bots dalam sebuah module dalam Apache web server.

6.1 Hash Table

Seperti yang telah dijelaskan pada bab sebelumnya, yaitu bab analisa desain sistem Kill Bots, Hash table digunakan untuk mencatat request-request yang gagal untuk menjawab tes yang diberikan pada saat server mengembalikan kondisi siaga atau SUSPECTED, di mana kondisi tersebut berarti kondisi server sedang cukup sibuk, sehingga harus siaga pada setiap request yang masuk ke dalam server.

Segmen Program 6.1 Inisialisasi Hash Table Pada Apache

```
1: apr_hash_t *puzzle_dump_tab;  
2: puzzle_dump_tab = apr_hash_make(pconf);
```

Segmen program 6.1 adalah segmen program untuk menginisialisasi sebuah hash table. Pada praktiknya, kedua baris tersebut tidak benar-benar berada dalam satu pasang baris yangurut seperti yang terlihat. Baris pertama mengartikan bahwa adanya pembuatan sebuah variabel yang bernama `puzzle_dump_tab` yang bertipe `apr_hash_t`. Tipe tersebut adalah sebuah hash table yang telah disediakan oleh Apache, sehingga dapat dipakai dengan bebas. Selayaknya sebuah variabel yang ada dalam bahasa pemrograman C, mengalokasikan memori ada sebuah langkah yang utama, dengan demikian meneruskan pada baris kedua. Sebuah fungsi yang bernama `apr_hash_make` adalah fungsi yang digunakan untuk

mengalokasikan memori yang diperuntukkan bagi hash table. Seperti yang terlihat, fungsi tersebut membutuhkan sebuah parameter, yaitu sebuah pool memori yang nantinya, memori dari hash table ini dialokasikan. Memori yang dibutuhkan ini adalah memori pool yang telah ada pada Apache itu sendiri. Pool dari Apache adalah bertipe `apr_pool_t`. Dalam contoh segmen program 6.1, pool yang digunakan adalah `pconf`. Pool tersebut adalah pool yang berada pada konfigurasi server. Pool tersebut mempunyai batas waktu selama server berjalan dan terbentuk ketika server melakukan proses start-up atau proses di mana server pertama kali dijalankan.

Dalam keseluruhan baris kode pada Kill Bots, baris pertama tidak berada tepat di atas baris program ke dua. Baris program pertama berada di luar fungsi-fungsi yang ada dalam Kill Bots, atau sering dikenal sebagai daerah *global*, sehingga fungsi-fungsi yang ada dapat mengakses atau memanggil variabel tersebut secara leluasa. Sedangkan baris program kedua berada pada fungsi yang mana akan dijalankan pada saat server melakukan proses start-up, yaitu pada fungsi `x_posh_config`. Dari fungsi tersebutlah didapatkan pool yang berada pada bagian server, sehingga hash table tersebut dapat selalu ada saat server berjalan.

Segmen Program 6.2 Menulis dan Mendapatkan Data Pada Hash Table

```
1: apr_hash_set(puzzle_dump_tab,
2:     apr_pstrdup(r->server->process->pconf, hashed_ip),
3:     APR_HASH_KEY_STRING, apr_pstrdup(r->server->process->pconf,
4:     apr_psprintf(r->server->process->pconf, "%i", counter)));
5: apr_hash_get(puzzle_dump_tab, hashed_ip, APR_HASH_KEY_STRING);
```

Sebuah tabel, pasti mempunyai proses untuk menuliskan sebuah data pada tabel tersebut dan juga mendapatkan data yang diinginkan. Segmen program 6.2 memperlihatkan fungsi yang harus dipakai untuk melakukan kedua proses tersebut. Penulisan data pada hash table ini tidaklah terlalu sukar, hanya dengan menggunakan fungsi pada baris pertama, yaitu `apr_hash_set`, data dapat dituliskan pada hash table yang diinginkan. Fungsi untuk menuliskan data tersebut mempunyai empat buah parameter yang harus diisi. Parameter pertama adalah

nama hash table yang akan menjadi tempat penyimpanan data. Kedua adalah key yang digunakan untuk mencari data yang ingin dicari. Berlanjut pada parameter ketiga adalah panjang dari key yang diberikan, dalam Kill Bots, key yang digunakan adalah sebuah teks atau string, sehingga tidak ada pemikiran untuk mencari panjang key, tetapi dengan hanya menggunakan komponen `APR_HASH_KEY_STRING` hash table tersebut akan menyelaraskan panjang dari key tersebut menjadi key yang bertipe string. Parameter yang terakhir adalah nilai yang akan disimpan pada hash table.

Sampai pada tahap ini, telah diketahuinya cara untuk menuliskan sebuah data pada hash table, tetapi ada beberapa prosedur yang perlu diketahui. Hash table tidak menduplikasi memori yang masuk, melainkan hanya menunjuk tempat memori tersebut, sehingga membuat data yang ada dalam hash table tidak terpercaya, maka dari itu, dalam proses penulisan pada hash table, key mau pun nilai data yang akan disimpan harus diduplikasi terlebih dahulu pada memori yang cukup aman. Berbicara tentang memori yang aman, berarti menunjuk pada memori server yang selalu ada ketika server hidup. Fungsi untuk menduplikasi memori dalam Apache adalah menggunakan fungsi `apr_pstrdup`, yang mempunyai dua parameter, yaitu parameter pool memori yang akan digunakan dan nilai data yang akan diduplikasi.

Pada baris segmen keempat, tertulis fungsi `apr_psprintf` yang mempunyai kegunaan untuk menuliskan sebuah data ke dalam bentuk teks atau string. Dalam kondisi ini, sebuah data integer akan dituliskan sebagai sebuah string dengan menggunakan bantuan fungsi tersebut. Terdapat tiga parameter yang harus diisi, yaitu parameter pertama adalah parameter yang berisi tentang pool memori yang mempunyai tujuan sebagai tempat untuk menyelesaikan proses ini. Parameter kedua adalah format string yang akan dituliskan. Format `%i` adalah format untuk data integer. Parameter terakhir adalah parameter untuk data yang akan diubah menjadi string. Fungsi ini membutuhkan satu tambahan parameter lagi yang berguna untuk menghentikan pencarian data yang akan diubah, parameter tersebut harus diisi dengan nilai `NULL`, sehingga fungsi tersebut dapat digunakan dengan

benar, jika tidak, maka proses tersebut akan mengembalikan error karena tidak dapat berhenti mencari data yang akan diubah.

Setelah menuliskan sebuah data, maka pasti ada keinginan untuk mendapatkan data yang telah disimpan. Proses mendapatkan data pada hash table ini sangatlah mudah. Fungsi yang digunakan mempunyai parameter yang hampir sama pada fungsi penulisan data. Fungsi yang digunakan dapat dilihat pada segmen program 6.2 pada baris ke lima, nama fungsi tersebut adalah `apr_hash_get`. Fungsi yang mempunyai tiga buah parameter dan parameter-parameter yang dibutuhkan sama persis dengan fungsi untuk menuliskan data, kecuali parameter terakhir pada penulisan data. Parameter tersebut tidak dibutuhkan pada proses mendapatkan data. Ada satu lagi perbedaan pada saat mendapatkan data pada hash table, yaitu key yang dicari tidak perlu diduplikasi sama sekali.

6.2 Bloom Filter

Bloom filter, sebagai baris depan tahap proses pencarian zombie, adalah komponen yang dapat dikatakan sebagai komponen atau fungsi yang utama. Kecepatan proses yang cepat, dan dengan proses yang minimal, Bloom filter dapat mendapatkan data yang dicari dalam waktu yang singkat, dengan demikian, dalam proses pencarian tidaklah memakan waktu yang lama dan proses yang kompleks, melainkan hanya membutuhkan waktu yang singkat dan proses yang sederhana.

Dalam bagian pembahasan implementasi dari Bloom filter ini, sebelumnya akan dibagi menjadi beberapa bagian. Bagian pertama dari pembahasan Bloom Filter ini adalah bagian pembahasan tentang implementasi dan baris kode yang digunakan pada hash function yang dipakai. RSHash dan JSHash adalah hash function yang dipakai sistem Kill Bots dalam tugas akhir ini. Pemilihan dari kedua hash function ini adalah dengan proses yang sangat sederhana, di mana di dalamnya hanya terdapat proses penambahan, perkalian, dan manipulasi bit dengan menggunakan fungsi XOR, *left shift*, dan *right shift*. RSHash adalah sebuah hash function yang diciptakan oleh seorang profesor pada Princeton University dan juga seorang direktur pada sistem yang pada abad ini adalah sistem

yang sangat terkenal, yaitu Adobe. Lahir pada tahun 1946, beliau mengembangkan hash function ini dengan tujuan untuk membuat sebuah hash function yang sesederhana mungkin dengan performa yang maksimal.

Segmen Program 6.3 RSHash

```

1:  unsigned int RSHash(unsigned char *str, unsigned int len)
2:  {
3:      unsigned int b    = 378551;
4:      unsigned int a    = 63689;
5:      unsigned int hash = 0;
6:      unsigned int i    = 0;
7:
8:      for(i = 0; i < len; str++, i++)
9:      {
10:         hash = hash * a + (*str);
11:         a     = a * b;
12:      }
13:
14:      return (hash);
15:  }
```

Segmen program 6.3 memperlihatkan sebuah prosedur sebuah fungsi dari hash function RSHash. Terlihat bahwa hash function ini mempunyai dua buah parameter yang harus diisikan. Parameter pertama memerlukan data teks apa yang harus dibungkus dalam hash function ini. Tipe data yang harus diberikan adalah unsigned char dan tipe data tersebut harus berupa pointer. Alasan mengapa harus dengan tipe data unsigned, dikarenakan tipe data unsigned mempunyai range nilai angka mulai dari nol sampai dengan 255, dengan demikian, disatukan dengan proses dari RSHash yang berisi tentang penambahan dan perkalian, hasil dari hash function ini tidaklah akan di bawah nol atau angka minus. Pada baris ke tiga sampai ke enam adalah baris yang digunakan untuk menginisialisasi variabel-variabel apa saja yang nantinya akan digunakan dalam proses hashing. Baris ketiga dan baris keempat mempunyai nilai yang paten dalam fungsi ini.

Setelah membahas tentang hash function yang pertama, JSHash pun menjadi bahan bahasan yang kedua. Hash function yang kedua ini diciptakan oleh Justin Sobel. Dalam pencarian tentang sejarah atau catatan tentang nama Justin Sobel, tidak adanya catatan yang menuliskan nama tersebut di dalam jurnal atau hasil riset sejauh pencarian dilakukan.

Segmen Program 6.4 JSHash

```

1:  unsigned int JSHash(unsigned char *str, unsigned int len)
2:  {
3:      unsigned int hash = 1315423911;
4:      unsigned int i     = 0;
5:
6:      for(i = 0; i < len; str++, i++)
7:      {
8:          hash ^= ((hash << 5) + (*str) + (hash >> 2));
9:      }
10:
11:     return hash;
12: }

```

JSHash mempunyai jumlah baris kode yang lebih sedikit dibandingkan dengan hash function yang pertama, yaitu RSHash. Total 12 baris, JSHash akan mengembalikan sebuah nilai yang bertipe data unsigned integer. Integer dalam bahasa pemrograman C bernilai 32bit, sedangkan tipe unsigned adalah tipe yang mengembalikan data selalu lebih dari nol atau angka bukan minus. Terlihat pada segmen program 6.4, fungsi JSHash hanya membutuhkan satu inisialisasi angka yang paten dibandingkan dengan hash function sebelumnya yang membutuhkan dua buah inisialisasi nilai yang paten sebelum melakukan proses hashing. Dalam hash function kali ini, proses yang digunakan adalah proses manipulasi bit dan juga penambahan sederhana.

Proses manipulasi yang dimaksud adalah proses Exclusive OR atau dikenal juga dengan singkatannya, yaitu XOR mempunyai simbol ^= yang mempunyai arti jika ada dua buah bit yang dibandingkan dan kedua bit tersebut berbeda, contohnya 0 dengan 1 atau 1 dengan 0, maka nilai hasil perbandingan tersebut adalah 1, dan sebaliknya, jika dua buah bit tersebut mempunyai nilai yang sama, contohnya 0 dengan 0 atau 1 dengan 1, maka nilai hasil perbandingan tersebut adalah 0. Proses manipulasi bit yang selanjutnya adalah left shift dan right shift bit. Kedua proses ini mempunyai kemiripan di mana perbedaannya hanya dalam orientasi arahnya saja. Pada dasarnya, proses shift bit adalah proses penambahan nilai 0 ke dalam bit array yang dituju sebanyak yang dikehendaki, dan berbicara orientasi arah, left shift yang mempunyai simbol << merupakan proses penggeseran bit array ke kiri sebanyak yang dikehendaki dan pada tempat di

sebelah kanan, di mana tempat tersebut adalah kosong, maka akan digantikan dengan nilai 0. Begitu juga dengan proses right shift yang mempunyai bentuk simbol \gg , bit array yang ditunjuk untuk dilakukannya proses ini akan digeser ke kanan sebanyak yang dikehendaki dan tempat di sebelah kiri, di mana tempat tersebut adalah kosong, akan diisi dengan nilai 0. Kedua proses ini sering sekali dijumpai pada proses masking sebuah bit array yang besar menjadi sebuah bit array yang lebih kecil.

Segmen Program 6.5 Inisialisasi Hash Function

```

1: void filter_get_hash(unsigned int hash[], char *str)
2: {
3:     unsigned char *str_in = (unsigned char *)str;
4:     int pos = strlen(str_in);
5:     hash[0] = RSHash (str_in, pos);
6:     hash[1] = JSHash (str_in, pos);
7: }
```

Setelah membahas tentang hash function apa saja yang digunakan dalam Bloom Filter ini, inisialisasi hash function tersebut adalah topik pembahasan selanjutnya. Nama prosedur atau fungsi untuk inisialisasi hash function dalam tugas akhir ini adalah `filter_get_hash`. Pada fungsi ini akan dikembalikannya sebuah array yang bertipe unsigned integer. Setiap elemen pada array tersebut berisi nilai kembalian dari hash function yang telah disiapkan. Seperti yang dapat dilihat pada segmen program 6.5, pada indeks pertama, yaitu indeks nol, akan diisi nilai kembalian dari RSHash dengan parameter string yang akan dibungkus oleh hash function tersebut dan juga panjang dari string tersebut. Berlanjut pada indeks ke dua, yaitu indeks satu, diisi dengan nilai kembalian dari hash function JSHash yang mempunyai parameter yang sama seperti hash function sebelumnya, string yang akan dibungkus dan panjang dari string tersebut. Dengan berakhirnya fungsi ini, hash function pun telah siap untuk dijadikan acuan indeks pada array Bloom filter yang sesungguhnya.

Segmen Program 6.6 Pengisian Data ke Dalam Bloom Filter

```

1: void filter_insert(unsigned char filter[], char *str)
2: {
3:     unsigned int hash[NUM_HASHES];
```


Segmen Program 6.6 (lanjutan)

```

4:      int i;
5:
6:      filter_get_hash(hash, str);
7:
8:      for (i = 0; i < NUM_HASHES; i++) {
9:          /* xor-fold the hash into FILTER_SIZE bits */
10:         hash[i] = (hash[i] >> FILTER_SIZE) ^
11:                 (hash[i] & FILTER_BITMASK);
12:         /* set the bit in the filter */
13:         filter[hash[i] >> 3] |= 1 << (hash[i] & 7);
14:
15:     }
16: }

```

Tahap pengisian data ke dalam Bloom filter adalah satu dari tiga tahap yang utama dalam pembuatan Bloom filter. Sebelumnya, inisialisasi hash function adalah salah satu dari ketiga tahap utama dari Bloom filter. Pada tahap ini data akan diisikan ke dalam Bloom filter. Awal fungsi ini berjalan akan ada insialisasi variabel yang mana akan menyimpan data dari nilai kembalian hash function, dan varciabel tersebut diinisialisasikan sebanyak jumlah hash function yang dipakai. Setelah itu penginisialisasian variabel yang digunakan untuk memulai proses perulangan atau looping. Variabel yang akan digunakan telah selesai diinisialisasikan, dan saatnya untuk proses utamanya. Sebelum memasuki proses perulangan, fungsi `filter_get_hash` akan dipanggil lebih dahulu. Seperti yang telah dibahas sebelumnya, fungsi yang dipanggil adalah fungsi yang digunakan untuk menginisialisasikan hash function yang akan digunakan. Dengan parameter yang diminta adalah tempat untuk menyimpan elemen-elemen hasil kembalian dari hash function, dan string yang akan dibungkus oleh hash function yang digunakan. Hash function telah masuk ke tempnya, maka berlanjut pada bagian perulangan.

Proses perulangan pada fungsi pengisian data pada Bloom filter ini akan diulang sebanyak jumlah hash function yang ada. Di dalam perulangan tersebut terdapat dua buah proses, yang pertama dan dapat dilihat pada baris sepuluh adalah proses untuk menyamakan bit yang ada pada nilai kembalian hash function dengan nilai filter dari Bloom filter yang digunakan. Dalam hash function yang digunakan dalam tugas akhir ini, nilai yang dikembalikan

mempunyai besar bit sebesar 32 bit, sedangkan besar bit yang digunakan dalam Bloom filter tugas akhir ini hanya sebesar 2^{20} bit. Angka 20 adalah besar filter dari Bloom filter dan juga angka 20 tersebut adalah nilai dari variabel `FILTER_SIZE`. Jika melihat pada baris ke sebelas, ada variabel yang bernama `FILTER_BITMASK`. Variabel tersebut berisi nilai bit dari besar filter. Dalam tugas akhir ini isi dari variabel tersebut sebesar 1.048.576 bits. Setelah proses penyamaan atau yang sering disebut dengan proses *masking* ini, proses pengisian bit pada filter pun dilakukan.

Segmen Program 6.7 Pencarian Data dari Bloom Filter

```

1:  int filter_is_any(unsigned char filter[], char *str)
2:  {
3:      unsigned int hash[NUM_HASHES];
4:      int i;
5:
6:      filter_get_hash(hash, str);
7:
8:      for (i = 0; i < NUM_HASHES; i++) {
9:          hash[i] = (hash[i] >> FILTER_SIZE) ^ (hash[i] &
FILTER_BITMASK);
10:
11:          if (!(filter[hash[i] >> 3] & (1 << (hash[i] &
7))))) {
12:              return 0;
13:          }
14:      }
15:
16:      return 1;
17:  }

```

Pada segmen program 6.7 dapat dilihat fungsi untuk mencari data dari Bloom filter. Seperti yang telah dilihat sebelumnya, fungsi ini tidak terlalu berbeda pada fungsi yang terdapat pada segmen program 6.6, yaitu fungsi pengisian data ke dalam Bloom filter. Perbedaannya hanya pada baris ke sebelas, di mana pada segmen program ini, fungsi akan bertanya, apakah indek yang ditunjuk dengan proses manipulasi yang sama sebelumnya mempunyai data atau tidak, jika tidak ada, maka fungsi tersebut akan langsung terhenti dan mengembalikan nilai 0, dapat dilihat pada baris ke 13. Jika indek yang ditunjuk

mempunyai data terus menerus hingga perulangan selesai, fungsi akan mengembalikan nilai 1 yang menandakan bahwa data tersebut ada dalam filter.

Segmen Program 6.8 Inisialisasi Bloom Filter

```

1: void filter_init(unsigned char filter[])
2: {
3:     int i;
4:     for (i=0; i < FILTER_BYTE_SIZE; i++){
5:         filter[i] = 0;
6:     }
7: }
```

Penginisialisasian Bloom filter ini tidaklah susah atau membutuhkan proses yang rumit. Hanya dibutuhkan sebuah perulangan sebanyak nilai variabel `FILTER_BYTE_SIZE`, yang mana isi dari variabel tersebut adalah besar filter dalam bentuk satuan byte. Dengan satuan ukuran dari filter yang digunakan adalah 20, maka nilai satuan byte dari angka tersebut adalah 131.072 bytes. Sebelumnya, variabel untuk menyimpan data-data yang ada telah dibuat pada awal program, dengan nilai dari `FILTER_SIZE_BYTE` sebagai acuan besar array yang dibuat. Pada proses inisialisasi Bloom filter ini, semua elemen yang ada di dalam array akan diisi dengan angka nol.

6.3 Load Server

Sering kali beberapa orang berbicara tentang load server. Seperti layaknya sebuah proses, load adalah presentase terpakainya sumber daya yang ada dalam proses tersebut. Berbicara sumber daya, pada server terdapat dua buah sumber daya yang mana mendukung kinerja dari server dalam melayani request-request yang masuk. Service yang dapat menampung berapa banyak request yang masuk adalah sumber daya utama yang perlu diperhatikan, dan yang kedua adalah CPU *usage* dari server tersebut. Jika server yang diakses hanya menyimpan data-data yang statik dan mempunyai proses yang tidak panjang, maka sumber daya kedua tidaklah terlalu penting, tetapi ketika berbicara tentang *web service* dan proses yang sangat panjang dan rumit, maka sumber daya kedua perlu diperhatikan dan prioritasnya tidak kalah dengan sumber daya pertama. Dalam tugas akhir ini,

server yang dipakai adalah server yang mempunyai proses yang menengah ke bawah. Server dengan proses menengah ke bawah adalah server yang memuat halaman web site yang bersifat statik atau halaman web site yang dinamik, tetapi prosesnya cukup sederhana.

Segmen Program 6.9 Mendapatkan Load Server

```

1:  double get_server_load()
2:  {
3:      worker_score *ws_record;
4:      process_score *ps_record;
5:      int i, j, res;
6:      int ready;
7:      int busy;
8:
9:      busy = 0;
10:     ready = 0;
11:
12:     for (i = 0; i < server_limit; ++i) {
13:         ps_record = ap_get_scoreboard_process(i);
14:         for (j = 0; j < thread_limit; ++j) {
15:             ws_record = ap_get_scoreboard_worker(i, j);
16:             res = ws_record->status;
17:
18:             if (!ps_record->quiescing
19:                 && ps_record->pid) {
20:                 if (res == SERVER_READY
21:                     && ps_record->generation ==
22:                     ap_my_generation)
23:                     ready++;
24:                 else if (res != SERVER_DEAD &&
25:                         res != SERVER_STARTING &&
26:                         res != SERVER_IDLE_KILL)
27:                     busy++;
28:             }
29:         }
30:
31:     return (busy * 100)/ (server_limit * thread_limit);
32: }

```

Pada segmen program 6.9 dapat dilihat proses untuk mendapatkan load server dengan acuan sumber daya yang pertama, yaitu banyaknya service atau daemon yang sedang aktif melayani request yang masuk. Baris ketiga sampai dengan baris ke tujuh adalah proses inisialisasi variabel-variabel yang akan digunakan untuk mendapatkan nilai dari load server pada saat itu. Baris ketiga dan baris keempat mempunyai peran yang penting dalam proses ini. Tipe data

`worker_score` mempunyai fungsi untuk mendapatkan status dari daemon-daemon yang dipunyai oleh server, sedangkan `process_worker` mempunyai fungsi mencari tahu tentang info tentang proses dari daemon tersebut. Dapat dilihat pada baris ke 13, adanya pemanggilan sebuah fungsi yang bernama `ap_get_scoreboard_process` dan nilai kembaliannya dimasukkan pada variabel `ps_record`, di mana variabel tersebut mempunyai tipe data `process_worker`.

Status daemon-daemon dapat didapatkan dengan menggunakan proses perulangan sebagai bantuan untuk dapat menjangkau semua daemon server. Ada dua buah proses perulangan yang dipakai, dan perulangan yang kedua berada di dalam proses perulangan yang pertama. Perulangan yang pertama akan diulangi sebanyak `server_limit` yang mana mempunyai nilai banyaknya server yang digunakan. Pada perulangan yang kedua akan diulangi sebanyak `thread_limit` yang bernilai jumlah daemon yang ada pada server. Baris kode 18 dan 19 adalah proses untuk menyatakan bahwa apakah daemon server sedang aktif atau tidak. Properti yang diperiksa adalah `quiescing` dan `pid` yang ada pada `process_worker`. Properti `quiescing` adalah yang menandakan daemon tersebut sedang tidak aktif, dan properti `pid` adalah properti yang bernilai identitas dari proses tersebut. Setelah itu berlanjut pada baris ke 20 sampai ke baris ke 27, yaitu mencari tahu apakah status daemon tersebut sedang sibuk atau tidak. Setelah pemeriksaan selesai, masuklah pada baris ke 31, yaitu proses untuk mengembalikan sebuah nilai. Nilai yang dikembalikan adalah hasil dari perhitungan presentase dari banyaknya daemon yang sibuk per jumlah server dan jumlah daemon yang ada dikalikan 100. Demikian nilai kembalian dari fungsi ini akan dijadikan acuan untuk metode transisi status server yang menandakan apakah server sedang berada pada tahap `NORMAL` atau pada tahap `SUSPECTED`.

6.4 Tabel Puzzle

Tabel puzzle adalah tabel yang berisi dengan puzzle-puzzle yang nantinya akan digunakan sebagai tes untuk semua request yang masuk pada server. Untuk

membuat tabel puzzle, dibutuhkan dua buah table untuk menyimpan jawaban dan soal dari puzzle tersebut.

Segmen Program 6.10 Inisialisasi Tabel Puzzle

```

1: void *init_puzzle_table(apr_table_t **q_tab, apr_table_t
   **a_tab, apr_pool_t *pool, int size)
2: {
3:     if (*q_tab){
4:         apr_table_clear(*q_tab);
5:     }
6:     else{
7:         *q_tab = apr_table_make(pool, size);
8:     }
9:
10:    if (*a_tab){
11:        apr_table_clear(*a_tab);
12:    }
13:    else{
14:        *a_tab = apr_table_make(pool, size);
15:    }
16:
17:    int i, j, count, data, ival, size_puzzle;
18:    unsigned char *name, *value;
19:
20:    size_puzzle = 10;
21:
22:    name = (unsigned char *)apr_palloc(pool,
size_puzzle*sizeof(char));
23:    value = (unsigned char *)apr_palloc(pool,
size_puzzle*sizeof(char));
24:
25:    for(i=0; i<size; i++){
26:        ival = 0;
27:
28:        for (j=0; j<size_puzzle; j++){
29:            name[j] = 0;
30:            value[j] = 0;
31:        }
32:
33:        for (j=0; j<size_puzzle; j++){
34:            srand(apr_time_now());
35:            count = rand()%255;
36:
37:            srand(apr_time_now());
38:            if (count % 2 == 0){
39:                data = rand()%9;
40:                name[j] = '0'+data;
41:                value[ival] = '0'+data;
42:                ival ++;
43:            }
44:            else{
45:                name[j] = (unsigned char *)((rand()%25)+97);
46:            }

```

Segmen Program 6.10 (lanjutan)

```

47:         }
48:
49:         char key[size_puzzle];
50:         sprintf(key, "%i", i);
51:         apr_table_set(*q_tab, (const char *)key, (const char
*)name);
52:         apr_table_set(*a_tab, (const char *)key, (const char
*)value);
53:     }
54: }

```

Segmen program 6.10 memperlihatkan tentang alur proses untuk inisialisasi tabel puzzle yang akan digunakan untuk menguji request yang masuk ketika server sedang berada pada kondisi SUSPECTED. Fungsi yang mempunyai nama `init_puzzle_table` ini mempunyai empat buah parameter yang diminta. Parameter pertama dan kedua mempunyai tipe data yang sama, karena parameter tersebut digunakan untuk media penyimpanan pertanyaan dan jawaban. Parameter selanjutnya digunakan untuk membantu proses inisialisasi ini, yaitu `pool`, di mana `pool` ini berguna untuk proses manipulasi data. Parameter terakhir digunakan sebagai acuan seberapa banyak baris data pada tabel yang akan dipakai.

Mulai dari baris ketiga sampai baris ke 15 digunakan untuk memeriksa apakah tabel yang diberikan telah ada atau belum terinisialisasi. Jika tabel yang diberikan telah terinisialisasi, maka akan dikosongkan terlebih dahulu dengan memanggil fungsi `apr_table_clear`, dengan tujuan untuk mengosongkan tabel tersebut, jika tabel tersebut belum terinisialisasi, maka akan dipanggilnya fungsi `apr_table_make`. Pada baris ke 20 terdapat nama variabel yang mempunyai arti sama seperti dengan parameter terakhir yang dibutuhkan oleh fungsi ini, tetapi pada pemakaiannya, variabel ini mempunyai kegunaan yang cukup berbeda. Dengan nilai sepuluh, variabel ini merupakan jumlah karakter yang akan dibuat sebagai pertanyaan yang akan diajukan kepada pengguna. Baris ke 22 dan ke 23 merupakan proses inisialisasi variabel yang nantinya sebagai tempat penampungan sementara untuk pertanyaan dan jawaban yang telah dibuat. Dengan menggunakan `apr_palloc`, variabel `name` dan `value` dinisialisasi sebanyak karakter yang akan diisikan. Data yang tersimpan pada variabel tersebut

akan disimpan pada memori pool, di mana pool sendiri adalah parameter yang diminta oleh fungsi ini.

Baris ke 25 adalah awal proses perulangan yang digunakan untuk mengisi tabel puzzle. Pada baris ke 28, dilakukannya proses pengosongan elemen yang ada pada variabel `name` dan `value`. Berlanjut pada baris ke 33 sampai dengan baris ke 47 adalah proses membuat karakter-karakter *alphanumeric* secara acak. Pada baris 49 sampai dengan 52, digunakan untuk proses menyimpan data pada tabel yang telah ditunjuk. Tabel puzzle ini adalah contoh mengalokasikan data pada memori secara statik. Berbicara tentang data statik, pool yang digunakan untuk mengalokasikan data statik ini adalah pool yang berada pada server, sehingga pool yang sesuai dengan proses ini adalah *configuration pool* yang mana pool tersebut ada ketika server hidup dan akan dilepaskan ketika server telah selesai dalam proses-prosesnya atau dengan kata lain server tersebut telah mati.

6.5 Content Handler

Ketika server merespon client dengan sebuah tes, pertanyaan, atau captcha, di sanalah tersdapat proses yang disebut dengan content handler. Dengan kondisi tertentu, server dapat memanipulasi konten yang akan diberikan kepada client, sesuai dengan kondisi dan aturan server tersebut.

Segmen Program 6.11 Content Handler Captcha

```

1:  ...
2:  if (apr_strnatcmp(notes, "create_puzzle") == 0){
3:      srand(apr_time_now());
4:      create_puzzle(r, apr_psprintf(r->pool, "%i",
      rand()%QUESTION_COUNT));
5:
6:      if (filter_is_any(bloom_filter, hashed_ip)){
7:          counter = atoi(apr_hash_get(puzzle_dump_tab,
      hashed_ip, APR_HASH_KEY_STRING));
8:      }
9:      else{
10:         filter_insert(bloom_filter, apr_pstrdup(r->server-
      >process->pconf, hashed_ip));
11:
12:     }
13:
14:     counter++;
15:

```


Segmen program 6.11 (lanjutan)

```

16:  //-- Masukkan ke dalam puzzle dump
17:  apr_hash_set(puzzle_dump_tab,
18:              apr_pstrdup(r->server->process->pconf,
19:                          hashed_ip),
20:              APR_HASH_KEY_STRING, apr_pstrdup(r->server->process->pconf,
21:                                                  apr_psprintf(r->server->process->pconf, "%i",
22:                                                  counter))));
21:
22:  return OK;
23:  }
24:  ...

```

Segmen program 6.11 memperlihatkan proses content handler untuk respon captcha. Pertama kali akan diperiksa apakah dari server memerintahkan untuk memberikan captcha atau tidak. Baris keempat dari proses di atas merupakan inti dari proses pemberian respon ini. Pemanggilan fungsi tersebut digunakan untuk membuat html script yang nantinya akan dikirimkan kepada client. Berlanjut pada baris keenam, yaitu baris yang memeriksa apakah request dengan alamat IP ini telah ada pada Bloom filter atau tidak, jika tidak ada maka akan ditambahkan, jika sudah ada, maka data dari hash tabel akan diambil dan dimasukkan kedalam variabel `counter`. Selanjutnya, nilai variabel tersebut akan ditambah satu, dan tahap terakhir adalah memasukkan alamat IP tersebut ke dalam hash tabel. Proses memasukkan ke dalam hash tabel ini mempunyai arti bahwa request dengan alamat IP tersebut telah mendapatkan sebuah soal. Kemudian, setelah proses penyimpanan alamat IP selesai, akan adanya pengembalian nilai oleh proses ini, di mana pesan OK mempunyai arti, pada bagian content handler, tidak perlu untuk memanggil modul-modul yang lain.

Segmen Program 6.12 Content Handler Setelah Captcha

```

1:  ...
2:  else if (apr_strnatcmp((char *)notes, "redirect") == 0){
3:
4:      char *meta = apr_psprintf(r->pool, "<meta http-
5:      equiv=\"refresh\" content=\"5; url=%s\">", r->uri);
6:      ap_set_content_type(r, "text/html; charset=ascii") ;
7:      ap_rputs("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML
8:      4.01//EN\">\n", r);

```

Segmen Program 6.12 (lanjutan)

```

8:     ap_rputs(apr_psprintf(r->pool, "<header>%s</header>",
    meta), r);
9:     ap_rputs("<html>", r);
10:    ap_rputs("<body>", r);
11:    ap_rputs("<p>Your page will be redirected immediately</p>",
    r);
12:    ap_rputs("</body>", r);
13:    ap_rputs("</html>", r);
14:
15:    return OK;
16: }
17: ...

```

Content handler untuk captcha telah dijelaskan, selanjutnya bagian di mana captcha telah dijawab. Segmen program 6.12 memperlihatkan proses penanganan konten setelah captcha berhasil dijawab. Pada awalnya akan ada pemeriksaan apakah dari server mempunyai sebuah pemberitahuan yang berisikan teks redirect atau tidak, jika ternyata ada pemberitahuan untuk redirect, maka server akan melemparkan sebuah respon yang mana halaman tersebut akan dilanjutkan ke web muka dari alamat web yang sedang dikunjungi. Proses pemberian respon tersebut dapat dilihat mulai pada baris keempat sampai dengan 13, dan seperti sebelumnya, sebuah kembalian dengan nilai OK dilemparkan.

6.6 Captcha

Dalam bagian pembahasan content handler, telah disinggung tentang fungsi untuk membuat respon konten yang berisi tentang captcha yang akan dikirimkan, nama fungsi tersebut adalah `create_puzzle`. Fungsi ini mempunyai proses yang hampir sama dengan proses yang ada pada segmen program 6.12, yaitu pemberian respon konten di mana captcha telah berhasil dijawab.

Segmen Program 6.13 Pembuatan Respon Konten Captcha

```

1:  char *xtoken = apr_psprintf(r->pool, "<input type=\"hidden\"
    name=\"token\" value=\"%s\"/>", new_token(puzzle_id, r,
    KEY));
2:  char *puzzle_text = apr_table_get(question_puzzle_tab,
    puzzle_id);
3:  char *js0 = apr_psprintf(r->pool,

```

Segmen Program 6.13 (lanjutan)

```

4:  "<script language=\"javascript\" type=\"text/javascript\"
    src=\"http://%s/jquery.js\"></script>", r->hostname);
5:
6:  char *js1 = apr_psprintf(r->pool, "<script
    language=\"javascript\" type=\"text/javascript\"
    src=\"http://%s/index.js\"></script>", r->hostname);
7:
8:  char *js2 = apr_psprintf(r->pool, "<script
    language=\"javascript\"
    type=\"text/javascript\">$(document).ready(function(){
    $(\"#container\").html(generateCaptcha(\"%s\"))
    });</script>", puzzle_text);
9:
10: ap_set_content_type(r, "text/html; charset=ascii") ;
11: ap_rputs("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML
    4.01//EN\">\n", r) ;
12:
13: ap_rputs("<html>", r);
14: ap_rputs("<head>", r);
15: ap_rputs(js0, r);
16: ap_rputs(js1, r);
17: ap_rputs(js2, r);
18: ap_rputs("</head>", r);
19:
20: ap_rputs("<body>", r);
21: ap_rputs("<form action=\"#\" method=\"get\" style=\"text-
    align:center\">", r);
22:
23: ap_rputs("<h2>Our Website is Experiencing Unusually High
    Load.</h2>", r);
24: ap_rputs("<h2>To Restrict Automated Access We Require Code
    Verification.</h2>", r);
25: ap_rputs("<h3>Please Enter The Code (<i>Number Only</i>)
    Below.</h3>", r);
26:
27: ap_rputs("<input type=\"text\" name=\"answer\" />", r);
28: ap_rputs(xtoken, r);
29: ap_rputs("<input type=\"submit\" name=\"submit\" /><br/>",
    r);
30: ap_rputs("<br/>", r);
31:
32: ap_rputs("<div id=\"container\" style=\"font-family: 'courier
    new'; font-size: 5pt; line-height: 80%;\">", r);
33: ap_rputs("</div>", r);
34: ap_rputs("<br/>", r);
35: ap_rputs("</form>", r);
36:
37: ap_rputs("</body>", r);
38: ap_rputs("</html>", r);

```

Segmen program 6.13 menunjukkan pembuatan respon konten untuk captcha yang akan dikirimkan kepada client. Seperti yang telah dijelaskan sebelumnya,

prose ini mempunyai kemiripan pada proses respon konten ketika pengguna telah menjawab captcha yang diberikan dengan benar. Dalam konten captache ini digunakannya elemen form untuk mengirimkan data yang dijawab oleh user kepada server untuk diperiksa dan divalidasi dengan ketentuan-ketentuan yang ada pada server. Pertanyaan yang ada dalam captcha didapatkan dengan cara mengambil data dari tabel puzzle secara acak. Nilai kembalian dari tabel puzzle tersebut disimpan dahulu pada variable yang telah dibuat, yaitu variabel `puzzle_text`. Pembuatan sebuah respon konten yang akan dikirimkan kepada client sama seperti pembuatan website pada umumnya, dengan menggunakan script-script html yang sama, perbedaannya berada pada penulisan script tersebut agar dapat dikirimkan kepada client. Fungsi yang bernama `ap_rputs` adalah fungsi yang digunakan untuk menulis sebuah konten atau sebuah teks yang nantinya akan dikirimkan kepada client. Sebelumnya, konten yang akan dikirimkan harus dijelaskan terlebih dahulu. Penjelasan tipe konten yang akan dikirimkan dapat dilihat pada baris ketujuh pada segmen program 6.13, dengan menggunakan bantuan fungsi yang bernama `ap_set_content_type` yang akan menetapkan tipe konten yang akan dikirimkan. Setelah menjelaskan tipe konten apa yang dikirimkan, jika konten tersebut sebuah halaman html, maka sebelum menulis tag html, sebelumnya harus dituliskan header dari halaman tersebut. Penulisan header dari halaman sebuah web dapat dilihat pada baris kedelapan pada segmen program 6.13, begitu juga pada baris ketujuh segmen program ke 6.12.

6.7 Token

Kill Bots mempunyai sebuah rahasia dalam respon captcha yang dikirimkan kepada clientnya. Dengan sebuah token yang disisipkan di dalam captcha yang dikirimkan, Kill Bots akan memvalidasi jawaban yang dikirimkan oleh pengguna yang menjawab captcha tersebut. Pada bab sebelumnya telah dijelaskan tentang elemen apa saja yang ada dalam pembentukan token rahasia ini, dengan penggabungan empat elemen yang telah dibuat oleh Kill Bots, jawaban dari captcha yang diberikan pun akan divalidasi menurut token tersebut.

Segmen Program 6.14 Pembuatan Token

```

1:  char *new_token(unsigned char *puzzle_id, request_rec *r,
    char *key)
2:  {
3:      unsigned char *xtime, *xhash, *random, *body;
4:
5:      unsigned char *temp = (unsigned char *)apr_palloc(r-
    >pool, 4*sizeof(char));
6:
7:      temp = (unsigned char *)apr_pstrdup(r->pool,(puzzle_id));
8:
9:      int padding = 4 - strlen(puzzle_id);
10:     if (padding < 0){
11:         return NULL;
12:     }
13:
14:     while (strlen(temp) < 4){
15:         temp = apr_psprintf(r->pool, "0%s", temp);
16:     }
17:
18:     struct tm *ftime;
19:     time_t raw_time;
20:
21:     time(&raw_time);
22:     ftime = localtime(&raw_time);
23:
24:     char *hour, *minute;
25:
26:     hour = apr_psprintf(r->pool, "%i", ftime->tm_hour);
27:     if (strlen(hour) < 2){
28:         hour = apr_psprintf(r->pool, "0%s", hour);
29:     }
30:
31:     minute = apr_psprintf(r->pool, "%i", ftime->tm_min);
32:     if (strlen(minute) < 2){
33:         minute = apr_psprintf(r->pool, "0%s", minute);
34:     }
35:
36:     xtime = (unsigned char *)apr_psprintf(r->pool, "%s%s",
    hour, minute);
37:     random = (unsigned char *)randomize_char(22, r);
38:     body = (unsigned char *)apr_pstrcat(r->pool, temp,
    random, xtime, key, NULL);
39:     xhash = apr_psprintf(r->pool, "%u", RSHash(body,
    strlen(body)));
40:
41:     return apr_pstrcat(r->pool, temp, random, xtime, xhash,
    NULL);
42: }

```

Seperti yang telah dijelaskan, token Kill Bots ini merupakan penggabungan dari empat elemen, sehingga dapat terlihat pada baris ketiga dalam segmen program ke 6.14, tersedianya empat buah variabel yang akan menampung nilai

dari keempat elemen tersebut. Mulai baris kelima sampai ke 16, terlihat bahwa puzzle id dari puzzle yang telah dipilih akan ditulis dengan format yang sesuai dengan token Kill Bots ini. Format dari puzzle id yang akan digabungkan berupa kumpulan karakter berjumlah empat, sehingga jika sebuah puzzle yang terpilih mempunyai puzzle id 10, maka akan disesuaikan dengan format yang sudah ada, yaitu 0010. Berlanjut pada pembuatan elemen selanjutnya, yaitu creation time. Pada elemen ini dibutuhkan fungsi-fungsi yang dapat mendapatkan nilai waktu yang ada di dalam komputer. Proses pembuatan elemen creation time ini dapat dilihat mulai baris ke 18 sampai dengan baris ke 36 pada segmen program ke 6.14.

Pada baris selanjutnya, sesuai dengan nama variabel yang diberikan, yaitu `random`, akan dibuatnya sejumlah karakter random yang mana angka random ini digunakan untuk menambahkan kerumitan dari token ini, sehingga tidak mudah untuk kupas oleh pihak lain. Angka random yang akan digabungkan adalah sebesar 22 karakter banyaknya. Ketiga elemen telah dibuat dan siap untuk digabungkan, tetapi masih tertinggal satu elemen lagi yang menentukan kevalidasian token ini, yaitu elemen hash. Elemen keempat ini adalah gabungan dari ketiga elemen sebelumnya ditambahkan dengan key yang diberikan oleh server sendiri. Setelah puzzle id, creation time, random, dan server key digabung untuk mendapatkan nilai hash ini, keempat elemen ini akan dibungkus oleh hash function, agar data yang didapatkan tidak mudah untuk diretas pihak lain. Hash function yang dipakai untuk membungkus elemen hash ini adalah RSHash, di mana hash function ini akan mengembalikan nilai karakter angka. Keempat elemen pun telah siap untuk digabungkan dan disisipkan kepada captcha yang akan dikirim kepada client.

Segmen Program 6.15 Validasi Token

```

1:  int is_valid_token(char *token, request_rec *r, char *key,
    int puzzle)
2:  {
3:      if (!token){
4:          return 0;
5:      }
6:  }
```

Segmen Program 6.15 (lanjutan)

```

7:     unsigned char *puzzle_id, *xtime, *xhash, *random, *body;
8:
9:     puzzle_id = (unsigned char *)apr_pstrmemdup(r->pool,
&token[0], 4*sizeof(char));
10:     random = (unsigned char *)apr_pstrmemdup(r->pool,
&token[4], 22*sizeof(char));
11:     xtime = (unsigned char *)apr_pstrmemdup(r->pool,
&token[26], 4*sizeof(char));
12:     xhash = (unsigned char *)apr_pstrmemdup(r->pool,
&token[30], (strlen(token)-30)*sizeof(char));
13:
14:     body = (unsigned char *)apr_pstrcat(r->pool, puzzle_id,
random, xtime, key, NULL);
15:
16:     unsigned char *new_hash = apr_psprintf(r->pool, "%u",
RSHash(body, strlen(body)));
17:
18:     if (apr_strnatcmp(new_hash, xhash) == 1){
19:         return 0;
20:     }
21:
22:     //-- Check Waktu Respond Form Data
23:     int hour, minute;
24:
25:     hour = atoi(apr_pstrmemdup(r->pool, (const char
*)&xtime[0], 2*sizeof(char)));
26:     minute = atoi(apr_pstrmemdup(r->pool, (const char
*)&xtime[2], 2*sizeof(char)));
27:
28:     struct tm *ftime;
29:     time_t raw_time;
30:
31:     time(&raw_time);
32:     ftime = localtime(&raw_time);
33:     ftime->tm_hour = hour;
34:     ftime->tm_min = minute;
35:
36:     double sec = difftime(time(NULL), mktime(ftime));
37:
38:     //-- puzzle = 1 -> Token untuk puzzle
39:     //-- puzzle = 0 -> Token untuk autentikasi
40:     int limit_time = 4;
41:     if (puzzle == 0){
42:         limit_time = 30;
43:     }
44:
45:     if (sec / 60 >= limit_time){
46:         return 0;
47:     }
48:
49:     return 1;
50: }

```

Jika pengguna yang dikirimkan sebuah captcha dapat memberi jawaban dari captcha tersebut, maka sebelum memeriksa jawaban dari captcha yang diberikan, Kill Bots akan memeriksa dan melakukan proses validasi terhadap token yang telah disisipkan pada captcha tersebut terlebih dahulu. Pada awal proses validasi token ini, akan dilakukan pembagian elemen-elemen yang menjadi dasar dari token ini. Baris kesembilan sampai dengan baris ke 12 memperlihatkan proses pemotongan token yang didapatkan menjadi bagian-bagian kecil yang mana bagian tersebut adalah elemen-elemen dasar dari token Kill Bots ini. Setelah dipotong dan diduplikatnya elemen-elemen tersebut, dibuatnya elemen hash yang baru sesuai dengan ketiga elemen pertama dari token yang didapatkan. Melalui proses yang sama dengan pembuatan elemen hash sebelumnya, hash yang baru pun dibentuk. Elemen hash yang didapatkan akan dicocokkan dengan elemen hash yang baru saja dibentuk, jika kedua elemen tersebut cocok, maka proses akan dilanjutkan, jika tidak, maka proses akan langsung berhenti dan memberikan nilai balik 0 seperti yang dapat dilihat pada baris ke 19.

Setelah mencocokkan elemen hashnya, masuk pada pengecekan waktu dari token tersebut. Pertama-tama akan diambilnya nilai waktu sekarang dari server, setelah itu akan ada proses perbandingan. Dalam Kill Bots terdapat dua buah batasan untuk masalah waktu ini, yang pertama ketika client menjawab captcha, dan yang kedua adalah ketika pengguna telah mendapatkan izin untuk mengakses server. Untuk poin pertama, perbandingan bernilai benar jika waktu token yang didapatkan adalah kurang dari 4 menit, dan untuk poin kedua, perbandingan bernilai benar jika waktu dari token yang didapatkan adalah kurang dari 30 menit.

Segmen Program 6.16 Daur Ulang Token

```

1: char *get_update_token(char *token, request_rec *r, char
   *key)
2: {
3:     if (!token){
4:         return NULL;
5:     }
6:     unsigned char *puzzle_id, *xtime, *xhash, *random, *body;
7:     puzzle_id = (unsigned char *)apr_pstrmemdup(r->pool,
   &token[0], 4*sizeof(char));
8:     random = (unsigned char *)apr_pstrmemdup(r->pool,
   &token[4], 22*sizeof(char));

```


Segmen Program 6.16 (lanjutan)

```

9:      struct tm *ftime;
10:      time_t raw_time;
11:      time(&raw_time);
12:      ftime = localtime(&raw_time);
13:      char *hour, *minute;
14:      hour = apr_psprintf(r->pool, "%i", ftime->tm_hour);
15:      if (strlen(hour) < 2){
16:          hour = apr_psprintf(r->pool, "0%s", hour);
17:      }
18:
19:      minute = apr_psprintf(r->pool, "%i", ftime->tm_min);
20:      if (strlen(minute) < 2){
21:          minute = apr_psprintf(r->pool, "0%s", minute);
22:      }
23:
24:      xtime = (unsigned char *)apr_psprintf(r->pool, "%s%s",
hour, minute);
25:      body = (unsigned char *)apr_pstrcat(r->pool, puzzle_id,
random, xtime, key, NULL);
26:      xhash = (unsigned char *)apr_psprintf(r->pool, "%u",
RSHash(body, strlen(body)));
27:
28:      return apr_pstrcat(r->pool, puzzle_id, random, xtime,
xhash, NULL);
29:  }

```

Segmen program 6.16 memperlihatkan tentang proses daur ulang token Kill Bots. Seperti arti dari daur ulang, token Kill Bots yang sudah ada akan dipakai kembali menjadi token yang baru dengan beberapa penggantian nilai menjadi yang lebih baru. Proses daur ulang ini akan dipanggil ketika sebuah captcha berhasil diselesaikan oleh pengguna yang mendapati captcha tersebut. Berdasarkan data elemen yang telah ada, sebuah token pun diperbaharui. Nilai elemen yang diperbaharui adalah elemen creation time. Dalam proses daur ulang ini, elemen tersebut akan diperbaharui dengan nilai yang sekarang ada pada server, dengan kata lain, nilai elemen tersebut akan diperbaharui dan menjadi nilai waktu dari server pada saat itu juga, dengan demikian jika ada pemeriksaan elemen creation time, token tersebut menjadi token yang lebih baru. Proses yang dilakukan pada daur ulang token ini tidaklah sepanjang proses pembuatannya. Mengawali proses dengan pemotongan bagian-bagian yang sebagaimana telah diberikan batasan pembagian. Setelah proses pemotongan selesai, maka dilanjutkan dengan mendapatkan nilai waktu yang ada pada saat itu dari sever.

Dengan tergantinya nilai waktu yang lama menjadi nilai waktu yang baru, maka dilanjutkannya dengan proses hashing dengan elemen yang baru. Pada akhir proses, seperti pada pembuatan token ini, proses penggabungan keempat elemen pun terjadi.

6.8 Cookie

Ketika pengguna telah dapat menyelesaikan sebuah captcha yang diberikan oleh server, maka pengguna tersebut akan mendapatkan sebuah perijinan yang didapatkan dari server. Perijinan tersebut berupa sebuah cookie yang diletakkan pada web browser pengguna yang telah menyelesaikan captcha tersebut.

Segmen Program 6.17 Mengambil Cookie yang Dikirimkan

```

1:  const char *get_cookie(const char *cookie_name, char
    *cookies, request_rec *r)
2:  {
3:      if (!cookies){
4:          return NULL;
5:      }
6:      unsigned char *token;
7:      apr_table_t *cookie_tab = apr_table_make(r->pool, 100);
8:
9:      char *pair;
10:     char *last = NULL;
11:     char *del = ";";
12:     for ( pair = apr_strtok(cookies, del, &last) ; pair ;
13:         pair = apr_strtok(NULL, del, &last) ) {
14:
15:         char *xlast = NULL;
16:         char *xcookie = apr_pstrdup(r->pool, trim(pair));
17:         char *xtoken, *key, *value;
18:
19:         xtoken = apr_strtok(xcookie, "=", &xlast);
20:         key = apr_pstrdup(r->pool, xtoken);
21:         xtoken = apr_strtok(NULL, "=", &xlast);
22:         value = apr_pstrdup(r->pool, xtoken);
23:
24:         apr_table_set(cookie_tab, (const char *)key, (const
    char *)value);
25:     }
26:
27:     return apr_table_get(cookie_tab, cookie_name);
28: }
```

Cookie adalah sebuah data yang tempat penyimpanannya ada pada web browser masing-masing pengguna. Cookie akan dikirimkan ke server bersama dengan request yang dikirimkan oleh pengguna ke server. Dengan demikian, server akan dapat membaca cookie yang dikirimkan beserta dengan request yang datang. Ketika sebuah request dikirimkan, cookie berada pada header request tersebut, sehingga untuk membaca cookie yang dikirimkan, haruslah membaca header request yang masuk tersebut. Bentuk dari cookie yang dikirimkan adalah berupa sebuah teks. Jika dalam request tersebut mempunyai lebih dari satu cookie, maka setiap elemen cookienya akan dipisahkan berdasarkan tanda pemisah titik koma. Dengan aturan pemisahan tersebut, maka untuk mendapatkan setiap elemennya, harus dilakukan pemotongan teks berdasarkan simbol titik koma.

6.9 Query String

Query string adalah sebuah elemen yang merupakan bagian dari Uniform Resource Locator atau yang sering dikenal dengan singkatan URL. Tanda yang menandakan bahwa teks yang ada dalam URL tersebut query string adalah simbol tanda tanya. Setelah simbol tanda tanya yang ada pada URL merupakan sebuah query string. Data yang dilemparkan oleh query string adalah berupa data teks. Data teks yang dilemparkan oleh query string mempunyai batasan-batasan karakter mengikuti aturan penulisan URL.

Segmen Program 6.18 Mengambil Query String

```

1:  apr_table_t *get_query_table(request_rec *r)
2:  {
3:      apr_table_t *av = apr_table_make(r->pool, 2);
4:      if(r->parsed_uri.query) {
5:
6:          const char *q = (const char *)apr_pstrdup(r->pool, (const
          char *)r->parsed_uri.query);
7:
8:          while(q && q[0]) {
9:
10:             const char *t = ap_getword(r->pool, &q, '&');
11:             const char *name = ap_getword(r->pool, &t, '=');
12:             const char *value = apr_pstrdup(r->pool, t);
13:
14:             if(name && (strlen(name) > 0)) {
15:                 if(value && (strlen(value) > 0)) {

```

Segmen Program 6.18 (lanjutan)

```

16:         apr_table_add(av, name, value);
17:
18:     }
19:     else if((strlen(name) > 0)) {
20:
21:         apr_table_add(av, name, "");
22:
23:     }
24: }
25: }
26: }
27:
28: return av;
29: }

```

Query string berada dalam request yang masuk ke dalam server. Dengan menggunakan bantuan `request_rec`, query string pun dapat diambil dengan cukup mudah. Nilai query string yang ada pada `request_rec` mempunyai format yang harus dijabarkan kembali agar pengaksesan data tersebut menjadi lebih mudah. Dengan cara yang hampir sama dengan berbeda fungsi yang dipakai, proses penjabaran query string ini dapat dikatakan memiliki konsep proses yang sama dengan penjabaran data cari cookie pada bagian sebelumnya. Fungsi yang terlihat pada segmen program 6.18 ini akan mengembalikan sebuah tabel yang telah disusun secara rapi agar pengambilan data-datanya menjadi lebih mudah.

6.10 Penentuan Status Server

Dalam proses untuk penentuan status ini, seperti yang telah dijelaskan sebelumnya, acuan dasar dari penentuan status server ini adalah load server. Dengan menggunakan load server, maka status server akan dipetakan apakah status pada server sekarang adalah NORMAL atau sedang dalam status SUSPECTED.

Segmen Program 6.19 Penentuan Status Server

```

1:  ...
2:  if (server_status == NORMAL_MODE){
3:      if (get_server_load() >= 70){
4:          server_status = SUSPECTED_MODE;
5:      }

```

Segmen Program 6.19 (lanjutan)

```

6:  }
7:  else if(server_status == SUSPECTED_MODE){
8:
9:      if (get_server_load() <= 50){
10:         server_status = NORMAL_MODE;
11:      }
12:  }
13:  ...

```

Untuk mendapatkan load server yang sekarang, pemanggilan sebuah fungsi pun dilakukan. Pada baris ketiga dapat dilihat pemanggilan fungsi tersebut. Fungsi yang mempunyai nama `get_server_load` adalah fungsi yang digunakan untuk mendapat load server tersebut. Dapat terlihat pada segmen program 6.19 terdapat beberapa macam pemeriksaan. Seperti yang telah dijelaskan sebelumnya, transisi status server adalah berdasarkan angka-angka sebagai batasannya. Jika sebelumnya server berstatus `NORMAL`, maka untuk naik menjadi status `SUSPECTED`, maka load server harus melebihi angka 70, dan sebaliknya, jika sebelumnya server berstatus `SUSPECTED`, maka untuk menurunkan status tersebut menjadi status `NORMAL`, maka angka yang harus dicapai adalah antara nol sampai dengan angka 50.

6.11 Pemberian Status Pada Content Handler

Untuk merespon sebuah konten sesuai dengan keadaan server, maka sebuah pemberitahuan pada awal proses pembacaan request pun harus dilakukan. Dalam Apache, untuk berkomunikasi antar bagian dan tahap yang ada haruslah melalui sebuah pemberitahuan yang diletakkan pada request tersebut, sehingga setiap proses pada setiap tahap yang ada pada Apache dapat mengetahui bahwa ada sebuah pemberitahuan dari tahap sebelumnya. Pemberitahuan tersebut ada dalam request yang nantinya diproses pada setiap tahap pada Apache. Tempat yang lebih detail adalah pada property note dalam `request_rec`. Data yang dapat ditampung oleh note hanya berupa data teks saja.

Segmen Program 6.20 Pemberian Status Pada Content Handler

```

1:  ...
2:  if (server_status == SUSPECTED_MODE && !input && !submit &&
    !token){
3:      cookie = get_cookie((const char *)COOKIE_NAME,
    apr_table_get(r->headers_in, (const char *)"Cookie"), r);
4:      if (cookie == NULL){
5:          apr_table_set(r->notes, note_name, (const char
    *)"create_puzzle");
6:
7:          return DECLINED;
8:      }
9:      if (is_valid_token((char *)cookie, r, KEY, 0) == 0){
10:         apr_table_set(r->notes, note_name, (const char
    *)"create_puzzle");
11:
12:         return DECLINED;
13:     }
14: }
15: ...

```

Pemberian sebuah pemberitahuan kepada sebuah tahap tertentu memerlukan sebuah kata kunci yang hanya pada tahap tersebut dikenali. Seperti pada pemberian pengetahuan yang ditujukan pada content handler ini, kata kunci yang akan dipakai adalah `create_puzzle` untuk memberikan pemberitahuan agar server merespon dengan memberikan captcha, dan redirect untuk melanjutkan pengaksesan ke server setelah pengguna dapat menjawab pertanyaan dari captcha yang diberikan.

6.12 Proses Pada Start-Up Server

Proses start-up server adalah proses yang dilakukan pertama kali dan hanya sekali ketika server pertama kali dijalankan. Kebanyakan pada proses start-up ini adalah proses yang memakan banyak sumber daya atau proses inisialisasi yang nantinya, data dari inisialisasi tersebut akan digunakan terus menerus, sehingga pada saat server sudah siap untuk melayani request-request yang data, data yang dibutuhkan telah siap dan tidak diperlukannya lagi inisialisasi tambahan.

Segmen Program 6.21 Proses Inisialisasi Pada Proses Start-Up

```

1:  static int x_post_config(apr_pool_t *pconf, apr_pool_t *plog,
2:                          apr_pool_t *ptemp, server_rec *s)
3:  {
4:      server_status = NORMAL_MODE;

```

Segmen Program 6.21 (lanjutan)

```

5:     init_puzzle_table(&question_puzzle_tab,
    &answer_puzzle_tab, pconf, QUESTION_COUNT);
6:
7:     puzzle_dump_tab = apr_hash_make(pconf);
8:     ap_mpm_query(AP_MPMQ_HARD_LIMIT_THREADS, &thread_limit);
9:     ap_mpm_query(AP_MPMQ_HARD_LIMIT_DAEMONS, &server_limit);
10:    filter_init(bloom_filter);
11:
12:    return OK;
13: }

```

Dapat dilihat pada segmen program 6.21 proses-proses yang terjadi pada tahap start-up server ini. Bermula dengan melakukan proses insialiasi pada tabel puzzle, setelah itu pada baris kedelapan dengan nama variabel `puzzle_dump_tab` adalah hash function yang digunakan untuk menyimpan data-data dari alamat IP mana saja yang tidak dapat menyelesaikan captcha yang diberikan oleh server. Kemudian pada baris ke 10 dan 11, proses tersebut adalah proses pengambilan data konfigurasi dari server yang isinya tentang banyak server yang ada dan berapa banyak thread yang ada pada Apache saat itu. Pada akhir proses start-up ini, inisialisasi Bloom filter pun dilakukan.

Segmen Program 6.22 Register Hook

```

1: static void register_hooks(apr_pool_t *pool){
2:     ap_hook_post_read_request(post_read_request_handler,
    NULL, NULL, APR_HOOK_FIRST);
3:     ap_hook_post_config(x_post_config, NULL, NULL,
    APR_HOOK_MIDDLE);
4:     ap_hook_handler(content_handler, NULL, NULL,
    APR_HOOK_FIRST);
5:     #ifdef HAVE_TIMES
6:     ap_hook_child_init(status_child_init, NULL, NULL,
    APR_HOOK_MIDDLE);
7:     #endif
8: }

```

Segmen program 6.22 memperlihatkan proses tentang mengaitkan tahap-tahap yang telah dibuat ke dalam tempat fungsi-fungsi tersebut akan ditempatkan. Proses ini telah dijelaskan pada bab yang membahas tentang Apache dan pengembangannya. Pengaitan tahap-tahap ini adalah pada tahap `post_read_request`, `post_config`, `handler`, dan `child_init`. Tahap yang pertama akan dipanggil ketika header request pertama kali masuk dan pertama

kali dibaca oleh server. Kedua adalah tahapan di mana server dijalankan untuk pertama kali atau yang dikenal dengan nama lain, yaitu start-up server. Pada poin ketiga adalah tahapan yang menangani tentang content handler, dan poin terakhir adalah digunakan untuk mendapatkan process id dari server. Proses untuk mendapatkan process id server hanya dibutuhkan satu fungsi, yaitu fungsi `getpid()`, di mana fungsi tersebut dapat digunakan tanpa membuat fungsi sendiri, dengan kata lain Apache telah menyediakan fungsi tersebut sehingga dapat digunakan dengan lebih mudah.

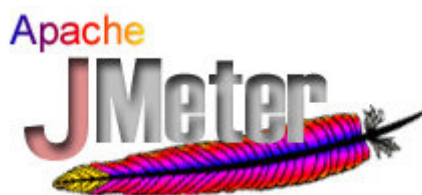
BAB VII

UJI COBA PERFORMA

Bab ketujuh ini akan membahas tentang hasil pengujian sistem Kill Bots. Ada beberapa scenario dalam pengujian sistem Kill Bots ini. Dalam pengujiannya, Kill Bots akan dibandingkan dengan server yang mempunyai konfigurasi standar di mana tidak ada tambahan modul apa pun di dalamnya, hanya modul-modul yang memang sudah ada dalam pengaturan standar tersebut yang akan dimuat. Kemudian pengujian Kill Bots akan dibandingkan dengan modul yang cukup terkenal dalam lingkungan modul tambahan dalam Apache web server yang mana kegunaannya hampir sama seperti Kill Bots, yaitu menangani serangan Distributed Denial of Service yang serangannya berupa request berskala besar datang untuk mengakses server secara bersamaan.

7.1 JMeter

JMeter adalah aplikasi yang merupakan hasil dari proyek Apache. Kegunaannya sebagai alat load testing untuk menganalisa dan menghitung performa dari beberapa macam service. Fokus dari Apache JMeter ini adalah ke arah aplikasi web.




















Gambar 7.1
Apache JMeter

Beberapa macam tes yang dapat dilakukan oleh Apache JMeter adalah JDBC (Java DataBase Connection), FTP (File Transfer Protocol), LDAP (Ligthweight Directory Access), webservice, JMS (Java Message Service), HTTP (Hyper-Text Transfer Protocol), TCP (Transmission Control Protocol), dan proses sistem operasi yang bersifat native. Selain mempunyai kegunaan sebagai alat

untuk memberikan tes performa, Apache JMeter juga dapat digunakan sebagai alat untuk melihat performa sebuah server, nama lain dari proses ini adalah monitoring. Aplikasi ini dapat dipakai bersamaan untuk mengirimkan parameter-parameter yang diinginkan atau yang sering disebut dengan variable parameterization, melakukan validasi respon atau assertions, dapat menerima cookie yang dikirimkan dari server pada setiap thread yang dibuatnya, melakukan konfigurasi variabel-variabel yang akan dikirimkan, dan juga mempunyai laporan yang beraneka ragam.

JMeter juga dapat menerima plug-in yang dibuat untuk aplikasi tersebut. Pengembangan plug-in dari JMeter ini dapat dilakukan oleh siapa pun yang ingin mengembangkannya. Fitur-fitur yang mempunyai kegunaan yang tidak standar, dapat dipastikan fitur tersebut diimplementasikan bersamaan dengan dimuatnya sebuah plug-in. Berbicara tentang pengembangan aplikasi JMeter, Jakarta adalah tempat pengembangan dari aplikasi ini. Huruf J pada nama JMeter adalah berdasarkan huruf J yang ada dalam kata Jakarta. Jakarta adalah sebuah kota yang mempunyai gelar ibu kota dari Indonesia. Jika kedua kata tersebut digabungkan, maka JMeter adalah singkatan dari dua buah kata yaitu Jakarta Meter.

Dalam uji coba perbandingan performa tugas akhir ini akan digunakan laporan yang berjenis tabel. Data yang ditunjukkan secara visual oleh laporan ini adalah berupa nomor sample, kapan request mulai dikirimkan, nama thread, label dari thread, sample time dalam millisecond, status, Bytes, dan latency.

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Latency
1	13:08:49.896	Group 1 1-1	Root Page	3781		7753	3781
2	13:08:50.700	Group 1 1-9	Root Page	5152		7753	5152
3	13:08:49.986	Group 1 1-2	Root Page	6491		7753	6491
4	13:08:50.100	Group 1 1-3	Root Page	7426		7753	7426
5	13:08:50.398	Group 1 1-6	Root Page	15366		7753	15366
6	13:08:50.197	Group 1 1-4	Root Page	15613		7753	15613
7	13:08:50.500	Group 1 1-7	Root Page	16513		7753	16237
8	13:08:50.299	Group 1 1-5	Root Page	17784		7753	17784
9	13:08:50.799	Group 1 1-10	Root Page	17348		7753	17348
10	13:08:50.598	Group 1 1-8	Root Page	17590		7753	17590
11	13:08:55.854	Group 1 1-9	Root Page	12735		20693	12337
12	13:08:53.680	Group 1 1-1	Root Page	16103		20682	15770
13	13:08:56.480	Group 1 1-2	Root Page	16897		20682	16539
14	13:08:57.527	Group 1 1-3	Root Page	21786		20682	21480
15	13:09:07.015	Group 1 1-7	Root Page	15761		20768	15437
16	13:09:05.766	Group 1 1-6	Root Page	17107		20682	16703
17	13:09:05.812	Group 1 1-4	Root Page	18442		21013	18061

Gambar 7.2
Laporan Jenis Tabel Pada JMeter

Gambar 7.2 menunjukkan visualisasi dari laporan berjenis tabel. Sample time adalah waktu yang dibutuhkan sebelum request dikirimkan sampai dengan

JMeter menerima respon terakhir dari server. Dalam bagian ini, JMeter tidak menghitung waktu untuk proses rendering pada web browser atau proses memuat aplikasi client, yaitu javascript. Status pada JMeter mempunyai dua buah nilai, jika hijau maka kode status yang dikirimkan dari server adalah kode status 200, selain kode tersebut, maka akan ditujukkannya tanda merah dengan simbol perintah ditengahnya. Masuk pada bagian Bytes, laporan tersebut menunjukkan berapa besar konten yang dikirimkan dari server sesuai dengan request diterima. Pada poin terakhir, yaitu latency⁹. Poin latency ini mempunyai kemiripan pada sample time. Jika sample time menghitung dari sebelum request dikirimkan sampai dengan respon terakhir dari server diterima, pada poin latency ini, perhitungan awalnya sama seperti sample time, yaitu memulai perhitungan sebelum request dikirimkan, tetapi berbeda pada proses akhirnya. Proses akhir dari perhitungan latency ini adalah ketika JMeter menerima respon pertama. Menurut pengembangnya, perhitungan JMeter dengan perhitungan oalh aplikasi yang cukup terkenal seperti Wireshark mempunyai perbedaan yang tipis, karena perbedaan yang tipis tersebut, perhitungan JMeter ini menjadi poin yang dapat dijadikan acuan dalam membuat laporan performa dari server.

```
1374740891161,12,HTTP Request,200,OK,Group 1 1-1,text,true,5982,4
```

Laporan dari JMeter ini dapat didapatkan dalam bentuk file teks, di mana di dalamnya mempunyai data yang sama seperti laporan yang terlihat pada gambar di atas. Baris kode di atas adalah contoh dari laporan dalam bentuk teks untuk sebuah request. Dalam laporan dalam bentuk teks, ada beberapa poin yang tidak ada dalam laporan pada gambar di atas, yaitu jenis request yang dikirimkan, kode status dan status yang didapat dari respon server atas request yang dikirimkan. Kode status dalam laporan tersebut adalah kode respon html yang dikirimkan dari server, contoh dari kode status tersebut adalah 200, 404, 403, dan lain-lain. Sedangkan untuk poin status adalah keterangan dari kode status yang telah diterima. Dengan demikian untuk mendapatkan nilai-nilai yang dibutuhkan untuk membuat laporan performa dari server yang sedang diuji coba, data dari laporan

⁹ <http://jmeter.apache.org/usermanual/glossary.html#Latency>

dalam bentuk file teks tersebut harus dipotong-potong sedemikian rupa agar mendapatkan data yang lebih mudah untuk dibaca dan dikelompokkan.

Apache JMeter ini digunakan karena mempunyai fitur yang mana dapat membuat sebuah komputer membuat lebih dari satu buah thread untuk melakukan request kepada server, sehingga dengan menggunakan banyak komputer, kondisi serangan DDoS Flash Crowd pun dapat dicapai dengan mengerahkan jumlah thread per komputernya agar lebih besar.

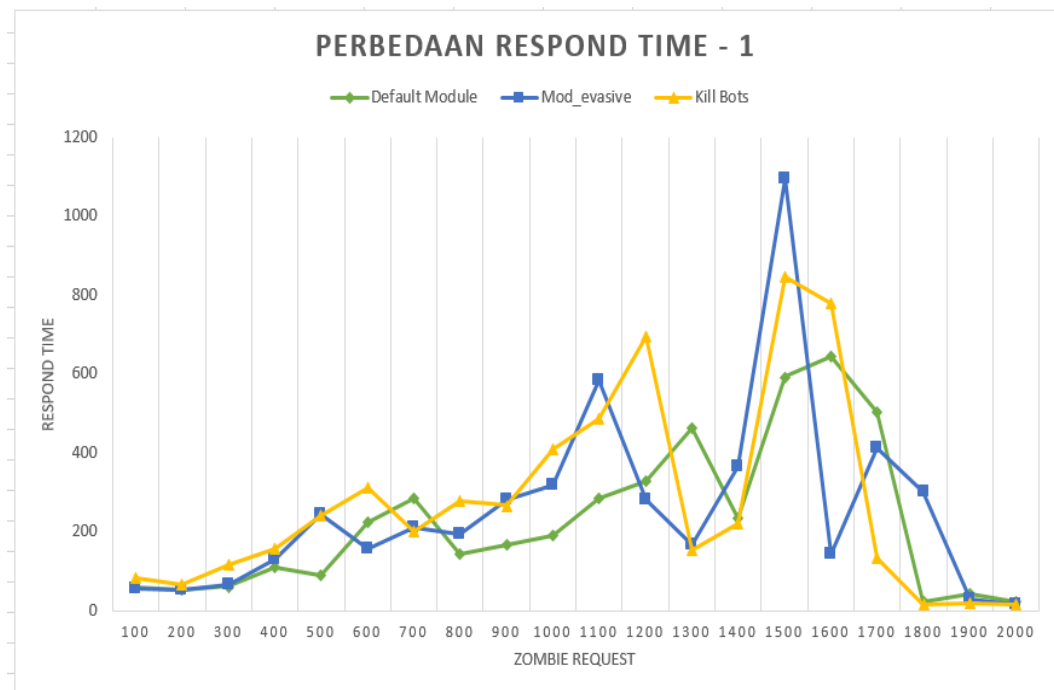
7.2 Uji Coba Perbandingan Performa

Bagian ini akan membahas dan memperlihatkan hasil dari uji coba performa dari server. Terdapat tiga buah kondisi pengujian dalam uji coba performa ini. Kondisi pertama server dengan default module, yang dimaksud dengan default module adalah tidak adanya penambahan modul lainnya pada server tersebut, server akan memuat modul-modul standar dari server itu sendiri. Kondisi kedua adalah kondisi di mana server akan diberi modul tambahan sebagai penanganan serangan DDoS, modul tambahan yang dipakai adalah `mod_evasive`. Terakhir adalah kondisi di mana server akan diberi modul yang telah diimplementasikan sistem Kill Bots di dalamnya. Terdapat empat skenario yang akan dilakukan terhadap ketiga kondisi tersebut. Setiap scenario akan dilakukan, server akan dijalankan ulang agar mendapat performa yang sama setiap skenarionya. Dalam pengujian ini, hasil kembalian yang akan diperhitungkan adalah respond time dari server ketika sebuah request masuk ketika gelombang serangan sedang dilancarkan kepada server.

7.2.1 Skenario 1

Dalam skenario pengujian performa server ini akan digunakannya sepuluh unit komputer sebagai agent atau worker aktif di mana kesepuluh komputer ini akan melakukan request terus menerus kepada server dengan parameter jumlah thread per komputer dan berapa kali komputer-komputer tersebut melakukan request per threadnya sesuai dengan skenario yang akan dijalankan. Pada skenario pertama ini, parameter-parameter yang akan disiapkan adalah sebagai berikut,

- Jumlah komputer: 10
- Thread per komputer: 20
- Request per thread: 10
- Total request: 2000



Gambar 7.3
Hasil Uji Coba Skenario 1

Pada gambar 7.3 dapat dilihat hasil pengujian performa dengan menggunakan skenario pertama. Pengujian dengan menggunakan skenario pertama ini tidaklah mendapatkan hasil yang memperlihatkan bahwa antara default module, mod_evasive, dan kill bots mempunyai perbedaan yang besar. Skenario pertama dirancang agar request yang dilancarkan bersifat lingkungan server yang normal. Meski hasil pengujian performa dari Kill Bots terlihat mempunyai respond time yang lebih tinggi dari default module, tetapi jarak perbedaan tersebut tidaklah tinggi dan cukup mendekati respond time dari default module. Begitu juga dengan server yang diberikan modul mod_evasive sebagai modul tambahan.

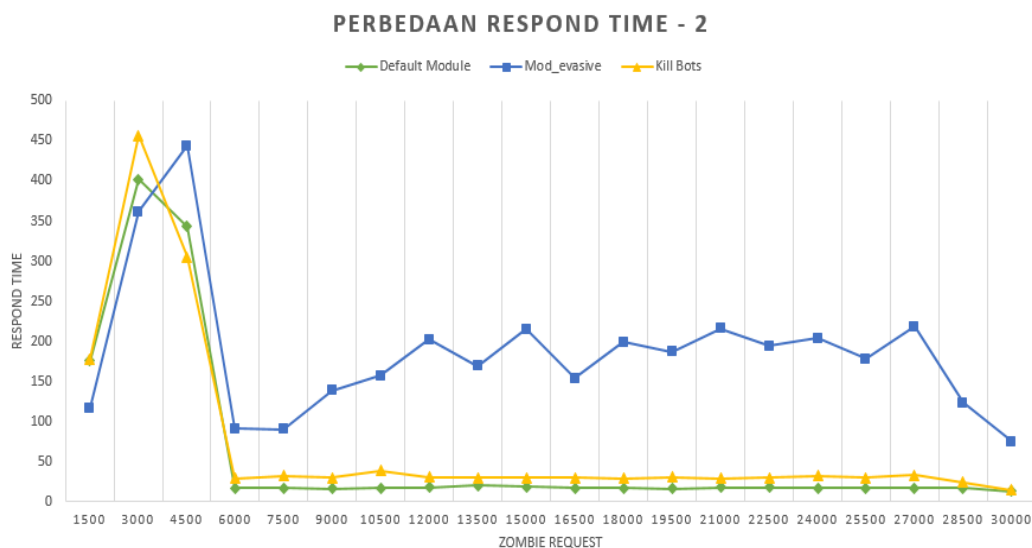
Berdasarkan data parameter yang diberikan, dengan delay per request yang dikirimkan adalah sebesar satu detik, maka akan didapatkan data rate request per

dtiknya adalah 200 request. Kill Bots memiliki respond time yang lebih tinggi karena adanya proses autentikasi yang berjalan di dalamnya, sehingga menaikkan waktu yang diperlukan untuk mengembalikan respon dari request yang diterima.

7.2.2 Skenario 2

Setelah melakukan pengujian berdasarkan lingkungan server yang cukup normal, kemudian pengujian berlanjut dengan menambahkan angka pada parameter request per threadnya. Dengan penambahan angka pada parameter request per thread ini, diharapkan adanya hasil yang cukup terlihat dari perbandingan ketiga kondisi tersebut. Berikut rincian dari parameter-parameter yang diberikan pada skenarion kedua,

- Jumlah komputer: 10
- Thread per komputer: 20
- Request per thread: 150
- Total request: 30000



Gambar 7.4
Hasil Uji Coba Skenario 2

Hasil yang cukup menakjubkan terlihat pada gambar 7.4 yang telah disajikan. Harapan untuk mendapatkan hasil yang memperlihatkan perbedaan dari ketiga kondisi ini pun sedikit tercapai. Hasil skenario kedua ini memperlihatkan

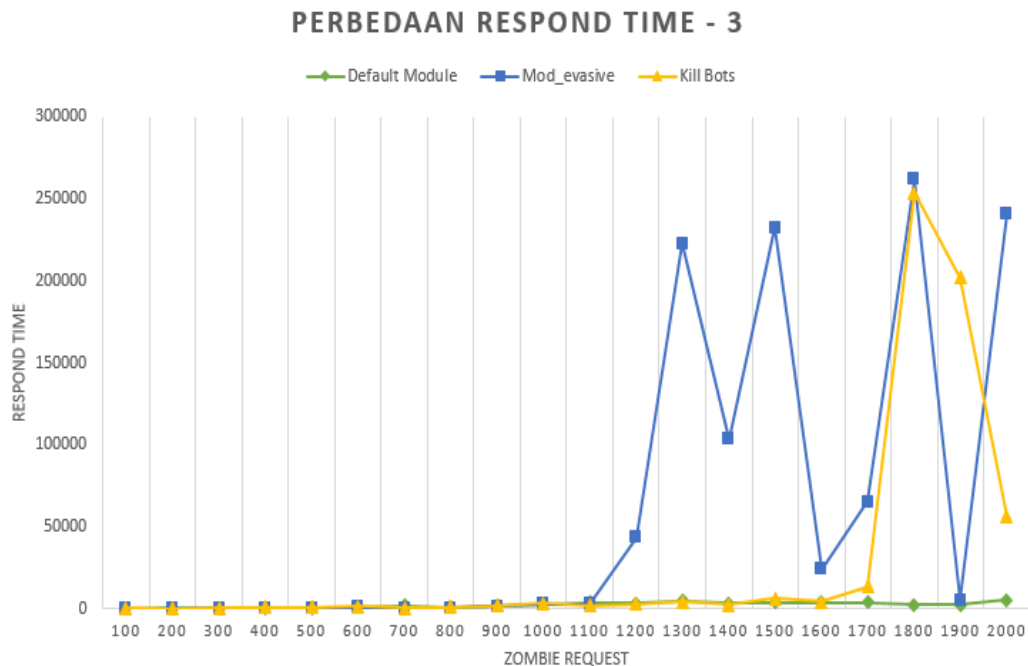
bahwa dalam rate request yang sama seperti skenario pertama, tetapi dengan total request yang berbeda, mempunyai dampak yang cukup terlihat pada kondisi server diberi modul tambahan berupa `mod_evasive`. Dalam skenario ini tampak sangat jelas `mod_evasive` memberikan respond time yang sangat jauh berbeda dengan kondisi default module. Berbeda dengan kondisi di mana server diberi modul tambahan berupa implementasi dari Kill Bots. Sistem Kill Bots dapat menjaga jarak performanya dengan server yang tidak diberikan modul tambahan apa pun. Seperti yang sebelumnya telah dijelaskan, alasan mengapa Kill Bots mempunyai respond time di atas default module adalah adanya tambahan proses transisi status pada server di mana proses yang berlangsung adalah pengambilan data load server saat itu juga. Perbedaan sekecil itu tidaklah terasa, karena perbedaannya tidak lebih dari 50 milisecond, dengan demikian, performa dari Kill Bots dapat dikatakan sama dengan performa server tanpa adanya modul tambahan pada rate request 200 request per detik. Berdasarkan hasil pengujian performa diantara ketiga kondisi tersebut, dan dengan perbedaan yang sangat terlihat, kira-kira hampir 100milisecond, `mod_evasive` telah mulai mengundurkan posisinya dari kedua kondisi yang masih bersaing antar satu sama lain.

7.2.3 Skenario 3

Pada skenario ketiga ini ada sedikit perbedaan dari dua skenario sebelumnya, di mana dua skenario sebelumnya memainkan angka request per threadnya, dalam skenario ketiga ini, parameter yang akan dimainkan adalah thread per komputernya. Pada sebelumnya, dengan perbedaan request per threadnya, rate request dari skenarionya tidak berubah sama sekali. Berbeda dengan skenario ini, dengan angka yang diubah adalah thread per komputernya dan request per threadnya dikurangi sehingga menjadi cukup kecil. Dengan demikian diharapkan dengan berbedanya rate request yang diberikan, hasil yang didapat pun akan beragam, sehingga dapat memberikan pengetahuan tentang keunggulan dari sistem Kill Bots. Berikut adalah rincian parameter yang diberikan pada skenario ketiga ini,

- Jumlah komputer: 10

- Thread per komputer: 100
- Request per thread: 2
- Total request: 2000



Gambar 7.5
Hasil Uji Coba Skenario 3

Hasil uji coba dengan skenario ketiga ini memberikan kejutan yang cukup besar. Dapat dilihat pada gambar 7.5, kondisi default module berlinggang dengan sangat mudahnya dengan nilai respond time dari uji coba performanya hampir mendekati nilai nol dibandingkan dengan dengan kondisi-kondisi yang lain. Pada awal skenario sampai dengan angka 1700 request, Kill Bots masih dapat mengikuti performa dari default module. Kejadian yang cukup mengherankan terjadi pada angka lebih dari 1700 request. Tiba-tiba performa dari Kill Bots melambung tinggi sampai dengan angka 250000. Untuk kondisi mod_evasive mulai terlihat kekurangannya pada angka awal, yaitu angka 1100 request.

Dengan hasil pada skenario ketiga ini, mod_evasive menurunkan kualitasnya sebagai modul penanganan serangan DDoS untuk Apache web server. Sampai dengan akhir pengerjaan tugas akhir ini, belum diketahuinya penyebab tentang melambungnya respond time pada saat di mana akhir skenario tercapai.

Dengan hasil yang terlihat pada skenario ketiga ini, dapat dinilai bahwa Kill Bots mempunyai kualitas yang sedikit dibawah default modul pada skenario dengan rate request 1000 request per detik.

7.2.4 Skenario 4

Skenario keempat ini adalah skenario yang diambil dengan menggunakan pergantian thread per komputernya. Dengan mengganti thread per komputer, maka rate requestnya pun akan berbeda. Berbeda dengan skenario pertama dan skenario yang kedua, di mana perbedaan keduanya hanya pada berapa kali thread itu mengakses server bukan perbedaan berdasarkan rate request.

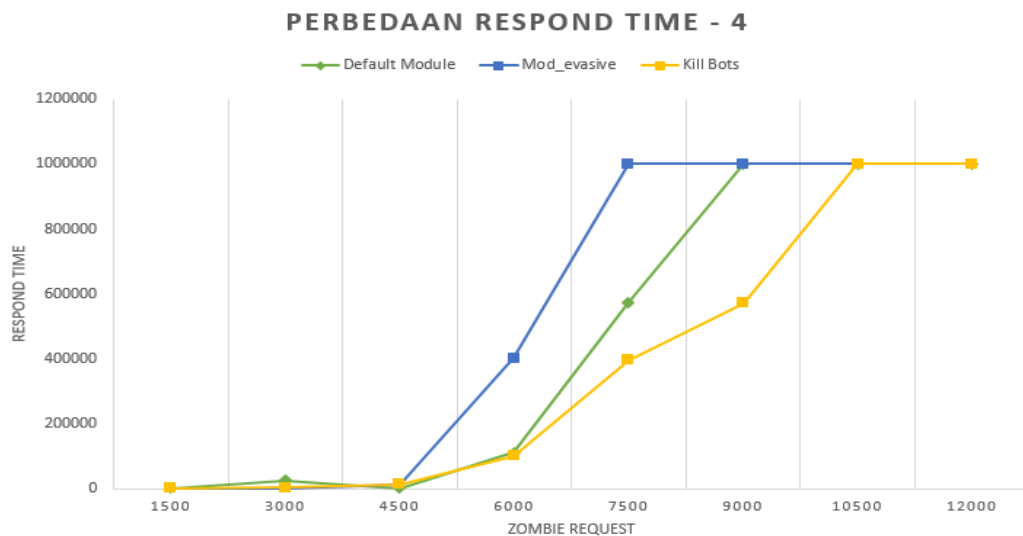
Dalam skenario keempat ini, rate request yang akan diujikan mempunyai perbedaan yang sangat jauh bila dibandingkan dengan skenario sebelumnya, yaitu skenario ketiga. Perbedaannya hampir 15 kali lebih tinggi dari pada rate request dari skenario ketiga. Jika sebelumnya rate requestnya bernilai 1000 request per detik, pada skenario keempat ini akan dinaikkan nilai rate requestnya sampai dengan angka 15000 request per detik.

Pada skenario keempat ini parameter yang digunakan akan dirancang berdasarkan lingkungan dari serangan DDoS Flash Crowd. Serangan Flash Crowd sendiri memberikan kondisi lingkungan yang sangat berat kepada server sehingga server diharapkan dapat memberikan hasil server tidak berkecil atau tidak merespon request yang masuk dari mana pun dan isi request apa pun. Dengan jumlah rate request per detiknya yang mempunyai nilai yang sangat tinggi, karena sangat tingginya rate request yang dilakukan, server akan menemui keadaan di mana server tidak dapat merespon request apa pun.

Ketika kondisi server tidak dapat merespon apa pun, data yang diambil seketika itu juga dihentikan, sehingga dapat dipetakan perbandingan antar kondisi yang diuji performanya. Berikut adalah rincian parameter yang digunakan dalam skenario keempat ini,

- Jumlah komputer: 10
- Thread per komputer: 1500
- Request per thread: 2

- Total request: 30000



Gambar 7.6
Hasil Uji Coba Skenario 4

Hasil uji coba skenario keempat yang harusnya mempunyai data sampai dengan 30000 request dipotong sampai dengan nilai 12000, karena pada angka lebih dari 12000 server telah menunjukkan bahwa server tidak dapat diakses atau server sedang tidak dapat merespon request-request yang masuk. Pada sebelumnya, server yang diberikan modul dengan sistem Kill Bots di dalamnya menunjukkan performa yang kurang dari default modul, pada skenario keempat ini Kill Bots mulai menunjukkan keunggulannya. Nilai 100000 pada bagian respond time menunjukkan bahwa server tidak dapat diakses.

Kondisi pertama yang menunjukkan server tidak dapat merespon request adalah mod_evasive pada nilai request 7500, dilanjutkan dengan default module pada angka 9000 request, dan yang terakhir adalah Kill Bots pada angka 10500 request. Pada gambar 7.6 juga dapat dilihat bahwa kenaikan respond time dari Kill Bots tidak secara langsung menanjak ke angka 100000, tetapi bertahap sehingga lebih banyak request yang dapat dilayani sebelum server sampai pada batasnya, sehingga tidak dapat merespon request-request yang masuk. Pada rate request yang sangat tinggi ini, Kill Bots menunjukkan kemampuannya dalam menangani serangan DDoS Flash Crowd.

BAB VIII

PENUTUP

Bagian ini adalah bagian terakhir yang berisi mengenai kesimpulan dan saran bagi pembaca buku ini. Kesimpulan dan saran ini diharapkan akan berguna bagi pembaca agar dapat lebih memahami dan dapat melakukan pengembangan pada penelitian lebih lanjut tentang Apache Module Development serta mengenai serangan Distributed Denial of Service dengan jenis Flash Crowd maupun jenis yang lainnya.

8.1 Kesimpulan

Bagian ini adalah bagian di mana kesimpulan-kesimpulan atas hasil penelitian dari Apache Module Development dan tentang serangan Distributed Denial of Service Flash Crows disusun. Kesimpulan-kesimpulan tersebut adalah sebagai berikut,

1. Apache merupakan web server yang berdasarkan keintegritasan dari modul-modulnya membuat pengembangan modul-modulnya lebih mudah untuk dipelajari. Arti dari kemudahan pengembangan modul-modulnya adalah tertatanya proses-proses apa saja yang dapat dikembangkan, sehingga ketika melakukan pengembangan modul Apache, tidak sulit untuk mencari proses atau tahapan apa yang harus dikembangkan.
2. Kintegritasan dari modul-modul Apache adalah berkat kerja dari proses MPM atau Multi Processing Modules dari Apache. MPM mengelompokkan proses-proses pada tahap yang sama dan menjalankannya pada secara bersamaan sesuai dengan prioritas-prioritas yang telah diberikan pada modul tersebut, dan dengan proses tersebut membuat proses kerja Apache berjalan lebih cepat.
3. MPM juga mempunyai kekurangan atas proses kerjanya. Kekurangan dari MPM adalah pelepasan pool yang tidak dipakai secara berkala,

sehingga membuat pembentukan tempat penyimpanan yang bersifat dinamik cukup sukar.

4. Pada serangan Distributed Denial of Service dengan tipe Flash Crowd, adanya perilaku yang dapat membedakan apakah itu zombie atau pengguna yang sah. Perbedaan dari kedua pihak tersebut dapat dilihat pada perilaku mereka terhadap tes yang diberikan kepada mereka.
5. Jika adanya rencana untuk membuat sebuah penyimpanan data yang bersifat statik, maka alokasi dari data tersebut harus diletakkan pada configuration pool, di mana untuk mengakses configuration pool dapat dilakukan ketika server pertama kali dijalankan atau sedang melakukan proses start-up.
6. Sebelum mulai berkecimpung dalam Apache Module Development, ada baiknya untuk mengaji tentang perilaku Apache Module lifecycle dan MPM terlebih dahulu, karena kedua komponen tersebut adalah sesuatu yang sangat penting dalam mengembangkan sebuah modul untuk Apache web server.
7. Bloom filter dapat mengikuti kecepatan proses server, dengan demikian menandakan bahwa proses pencarian kata kunci dalam Bloom filter mempunyai kecepatan yang sangat cepat.
8. Kill Bots mempunyai kemampuan untuk melayani jumlah request yang lebih banyak dari pada server dengan tanpa modul pertahanan dan server yang diberi pertahanan mod_evasive.
9. Performa Kill Bots dalam lingkungan rate request yang rendah, terkadang terjadi anomali yang membuat respond time dari server meningkat cukup tinggi. Kenaikan respond time hanya berlangsung sekali saja, setelah kenaikan tersebut, respond time server menurun kembali.
10. Bloom Filter mempunyai performa yang lebih cepat dari pada struktur data NTT dari mod_evasive di mana struktur data Bloom Filter merupakan struktur data yang menyerupai hash table sedangkan NTT merupakan struktur data yang berupa sebuah tree.

Diharapkan kesimpulan-kesimpulan yang telah didapatkan ini dapat bermanfaat bagi para pengembang modul Apache dan bagi pengembang sekuritas server berdasarkan serangan Distributed Denial of Service dengan tipe Flash Crowd.

8.2 Saran

Selain kesimpulan pada sub bab 8.1, dari proses autentikasi dari Kill Bots terdapat beberapa saran yang mungkin berguna bagi pembaca untuk mengembangkan topik ini lebih lanjut. Saran-saran tersebut adalah sebagai berikut,

1. Adanya penelitian lebih lanjut tentang Bloom filter yang dapat menyimpan data tidak hanya keynya saja, tetapi juga dapat menyimpan value dari key tersebut. Bloom filter yang dapat melakukan hal tersebut adalah Invertible Bloom filter.
2. Untuk menaikkan kualitas proses autentikasi Kill Bots, sebuah pemeriksaan berdasar performa server sebelumnya dapat diimplementasikan. Proses pemeriksaan tersebut dapat disebut dengan proses admission control, di mana acuan performa server dapat diambil dengan metode microbench di mana pengukuran performa server dapat diukur sampai nilai yang cukup kecil, yaitu sampai satuan microsecond.

Diharapkan saran-saran yang diberikan dapat berguna bagi pembaca yang ingin mengembangkan topik ini lebih lanjut dan ingin meningkatkan kinerja dari Kill Bots agar kualitas dari proses autentikasinya lebih mengarah ke request-request yang butuh untuk diautentikasi.

DAFTAR PUSTAKA

A. Goel dan P. Gupta. *Small Subset Queries and Bloom Filters Using Ternary Associative Memories, with Applications*. ACM SIGMETRICS, 2010.

Bloom, B. *Space/time Trade-offs in Hash Coding with Allowable Errors*. Communications of the ACM, 13 (7). 422-426.

CERT. *Incident Note IN-2004-01 W32/Novarg.A Virus*, 2004.

<http://billmill.org/bloomfilter-tutorial/>.

http://en.wikipedia.org/wiki/Bloom_filter.

J. Leyden. *East European Gangs in Online Protection Racket*, 2003. www.theregister.co.uk/2003/11/12/east_european_gangs_in_online/.

Jung., Jaeyeon. Krishnamurthy., Balachander. Rabinovich., Michael. *Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites*. Mei 2002.

K. Poulsen. *FBI Busts Alleged DDoS Ma_a*, 2004. <http://www.securityfocus.com/news/9411>.

Kandula., Srikanth. Katabi., Dina. Jacob., Matthias. Berger., Arthur. *Botz-4-Sale: Surviving Organized DDoS Attack That Mimic Flash Crowds*. 2005.

Kew., Nick. *The Apache Modules Book*. Januari 2007.

Specht., Stephen M. dan Lee., Ruby B. *Distributed Denial of Service: Taxonomies of Attack, Tools, and Countermeasure*. Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems, 2004 International Workshop on Security in Parallel and Distributed Systems, pp. 543-550, September 2004.

RIWAYAT HIDUP



Nama : Ian Febrian Reza Mulia
Yulianto
Alamat : Jl. Ngagel Madya X / 32,
Surabaya
Tempat/Tanggal Lahir : Rembang, 13 Febuari 1992

Jenjang Pendidikan:

- 1997 – 2003 SDK Santa Maria, Rembang
- 2003 – 2006 SMP Negeri 2, Rembang
- 2006 – 2009 SMA Negeri 1, Rembang
- Sejak 2009 Sekolah Tinggi Teknik Surabaya, Surabaya
(Program Studi S1 Jurusan Teknik Informatika)

Pengalaman Kerja:

- Januari 2009 – Juli 2012 Asisten Honorer Laboratorium Komputer STTS