BAB 2
DASAR TEORI

2.1 Kajian Pustaka

Penelitian [4] (Fiddin dkk, 2018) meneliti mengenai penyerangan *Denial of Service* (DoS) *Attack* yang berjenis SYN *flooding*, dari serangan tersebut dapat mengakibatkan klien tidak bisa merespon paket *server*, sehingga *port* koneksi antara klien dan *server* tidak dapat terbuka. Adapun parameter perhitungan *responsive server* diantaranya CPU, RAM, *IOzone write*, dan *IOzone read*. Untuk hasil pengujiannya dapat disimpulkan bahwa serangan *Denial of Service* memberikan dampak penurunan performansi dari sisi *overall performance* dan layanan *web server*, baik mesin *native* maupun mesin yang mengalami penyerangan, menggunakan teknik virtualisasi *container Docker*. Pada serangan yang relatif ringan didapatkan hasil bahwa pada parameter *request per second* menyebabkan penurunan performansi sebesar 40,22% pada *native* dan 37,65% pada *Docker*. Pada parameter *response time web server*, serangan DoS menyebabkan peningkatan *response time* pada *server native* sebesar 40,80%, sedangkan pada *Docker* sebesar 38,38%.

Pada penelitian [6] (Adrienda, 2021) menjelaskan tentang perbandingan kinerja dan respon dari *server virtual machine* (VM) dengan server kontainerisasi *docker* terhadap serangan *Denial of Service* (DoS) diatas *software* Ubuntu 20.04. Tipe serangan DoS yang digunakan yaitu TCP SYN *Flooding* dengan menggunakan *tools Low Orbit Ion Cannon* (LOIC) dan HTTP *Unbearable Load King* (HULK). Parameter yang diuji seperti CPU, *memory*, beban *web server*, *bandwidth*, dan *log* baik di *server* VM maupun *server docker*. Untuk penyerangannya sendiri akan dihitung berdasarkan *real time*, dan akan dilakukan pengecekan secara berkala menggunakan *tools Apache Benchmarking* sebagai *software* perhitungannya. Untuk hasilnya dapat disimpulkan dengan pengujian CPU *processing speed, virtual machine* lebih unggul dibandingkan dengan *docker*. Sedangkan untuk pengujian *memory bandwidth* pada *variabel copy* dan *scale*, hasilnya yaitu *docker* lebih unggul dibandikan dengan *virtual machine*. Selanjutnya dipengujian parameter *request per second* server *Docker* mempunyai *respond time*

yang lebih baik dibandingkan dengan *virtual machine*. Selanjutnya hasil pengujian *throughput bandwidth* pada *virtual machine* lebih unggul dengan perbedaan yang sangat ketat dibandingkan dengan *Docker*. Dapat disimpulkan dari hasil pengujian performa saat diserangan DoS, yang mempunyai dampak yang lebih besar adalah server *virtual machine* dan *server* jika dibandingkan, *server docker* lebih aman dari serangan DoS.

Lalu dipenelitian [5] (Bhatia dkk, 2018) membahas mengenai fitur baru Docker yaitu *Docker Swarm* juga rentan terhadap serangan *Denial of Service* (DoS). *Docker Swarm* merupakan *clustering* terdiri dari tiga node yang saling terkait dan saling bekerja sama. Hal pertama yang dilakukan yaitu membuat tiga *node cluster Docker Swarm* dengan kontainer *Nginx* menggunakan *software* Cent OS-7/Ubuntu 16.04. Selanjutnya dilakukan serangan DoS menggunakan *tool Low Orbit Ion Cannon* (LOIC), *High Orbit Ion Cannon* (HOIC) dan *Hping3* dengan mengalihkan jaringan dan *security* sementara, pada saat yang sama penyerangan dilakukan dengan memasukkan *malware* ke dalam jaringan yang bertujuan mencuri IP atau informasi. Untuk parameter yang dihitung diantaranya CPU *Usage, Memory Usage* dan *Response Time*. Pada penelitian dapat disimpulkan jika CPU *Usage, Memory Usage* dan *Response Time* mengalami masalah setelah dilakukan penyerangan DoS. Contohnya penggunaan CPU *usage* pada 0s penyerangan kapasitas CPU 0 *percentage,* lalu pada 60s penggunaan CPU menjadi 14.97 *percentage* dan pada 330s penggunaan CPU mencapai 42.48%. Untuk penggunaan *memory* pada 0s sebesar 1.344Mb, pada 60s sebesar 1.469Mb dan pada 330s 2.066Mb. Dan yang terakhir *response time* pada 0s yaitu 3.14/ms, pada 60s 4.02/ms dan pada 317s ada 16115.11/ms.

Tabel 2.1 Simpulan dan Perbedaan Antar Penelitian

Peneliti	**Teknologi**	**Skenario Penyerangan**	**Parameter Pengujian**	**Hasil**
Chrisna Fiddin, Munadi, Dr.	*container Docker*.	DoS SYN *flooding*	CPU, RAM, *IOzone write, IOzone read,*	Pada *request per second* menyebabkan

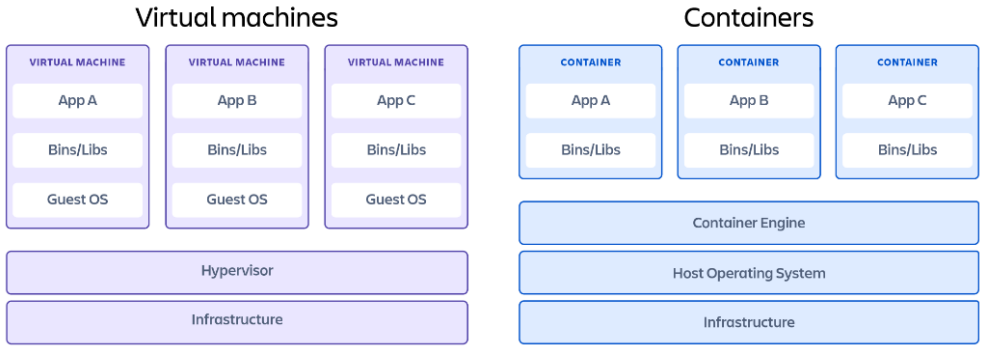
Peneliti	Teknologi	Skenario Penyerangan	Parameter Pengujian	Hasil
Rendy, Ratna Mayasari, S.T, M.T. [4]	Aplikasi web server.		*request per second dan respond time web server*	penurunan performansi 40,22%.
Sabrina Anisa Adrienda [6]	Membandingkan *server virtual machine* (VM) dengan server kontainerisasi *docker.* aplikasi HTTP	TCP dan SYN *Flooding*	CPU, *memory*, beban *web server*, *bandwidth*, dan *log* baik di *server* VM maupun *server docker*	Docker lebih aman dari penyerangan DoS.
Gaurav Bhatia, Arjun Choudhary, Krati Dadheech [5]	*Docker Swarm.* Aplikasi *Nginx*	Mengalihkan jaringan dan *security* sementara	CPU *Usage, Memory Usage* dan *Response Time*	Swarm cluster dapat dijatuhkan dalam beberapa menit dengan DoS
Rizki Desi Pamuji	*Container* Kubernetes. Aplikasi *web server, Nginx*	TCP *Flood*, UDP *Flood* dan *Smurf Attack*	*Respond time, throughput*, CPU *usage*, dan *memory usage*	Untuk penurunan performansi *container* pada penyerangan TCP flood sebesar 86,72%, UDP flood sebesar

Peneliti	Teknologi	Skenario Penyerangan	Parameter Pengujian	Hasil
				22,22% dan penurunan pada smurft attack sebesar 30,78%.

2.2 Dasar Teori

2.2.1 *Container*

Container merupakan sistem yang memungkinkan untuk membuat layanan dan sumber daya baru, dari suatu *operating system* yang sudah ada. Kontainerisasi ini dapat mengurangi jumlah sumber daya yang terbuang, karena hanya menjalankan dan menyediakan sesuai kebutuhan dari *binary* atau *library* sesuai dengan aplikasi atau layanan yang dijalankan [4]. Gambar 2.1 merupakan perbedaan kontainer dan *virtual machine* [9].



Gambar 2.1 Perbedaan *Virtual Mechine* dan Kontainer

Perbedaan antara *Virtual Mechine* (VM) yaitu pada VM terdapat *hypervisor*, sedangkan pada arsitektur *container* terdapat *docker* yang dapat digunakan untuk menjalankan *docker container*. Berbeda dengan *virtual machines*, *container* berbagi *kernel host operating system* dengan *container* yang lain, dari hal inilah yang mengakibatkan *container* lebih efektif penggunaannya dibandingkan dengan VM. Jadi dapat disimpulkan jika satu *server* VM atau satu *operating system*

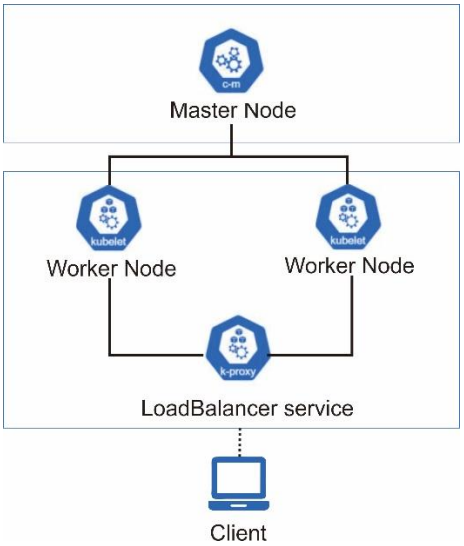
hanya dapat digunakan untuk satu aplikasi, sedangkan satu *container docker* dapat digunakan untuk lebih dari satu aplikasi [10].

2.2.2 *Containerd*

Containerd atau *container runtime* adalah *software* yang bertanggung jawab untuk mengelola atau membuat *container* dan *image container* pada *node* Kubernetes. *Containerd* ini dianggap lebih hemat sumber daya dan aman daripada *docker engine* untuk Kubernetes. Hingga Kubernetes versi 1.20, *docker engine* adalah *runtime container* yang utama. Tetapi, *dockershim* yang merupakan kode integrasi *docker engine* di Kubernetes, tidak digunakan lagi pada Kubernetes 1.20 dan sudah dihapus pada Kubernetes 1.24 [11]. *Containerd* merupakan *daemon* untuk Linux dan Windows. *Containerd* mengelola siklus hidup kontainer lengkap dari sistem *host*, mulai dari transfer, penyimpanan hingga eksekusi. *Containerd* juga melakukan pengawasan pada kontainer hingga penyimpanan tingkat rendah sampai ke lampiran jaringan [12].

2.2.3 Kubernetes

Kubernetes merupakan suatu *platform opensource* yang berfungsi untuk mengelola kumpulan kontainer dalam suatu *cluster server* [13]. Kubernetes bertugas menyediakan konfigurasi dan otomatisasi. Kubernetes juga menyediakan manajemen *environment* yang berpusat pada kontainer. Kubernetes melakukan pengolahan terhadap *computing*, *networking*, dan infrastruktur penyimpanan. Fitur tersebut juga membuat konsep menjadi lebih sederhana [14].



Gambar 2.2 Kubernetes *Cluster*

Gambar 2.2 merupakan *cluster* dari Kubernetes. *Unit* yang di *deploy* dan diatur pada Kubernetes disebut dengan *pod*. *Container* yang dijalankan dalam suatu *pod* bisa satu atau lebih dari satu *container*. Di dalam arsitekstur Kubernetes membagi menjadi dua bagian diantaranya *Master node* dan *Worker node*. Disetiap bagian mempunyai tugas masing-masing seperti *Master Node* berfungsi untuk mengatur *pod*. Sedangkan *Worker Node* berfungsi sebagai *host* dari *pod* yang dijalankan. Komunikasi antara *cluster* Kubernetes dengan *client* menggunakan jenis komunikasi *loadbalancer* yang di pasangkan dengan *web server Nginx*.

Fitur-fitur yang menjadi kelebihan dari Kubernetes diantaranya :

1) *Service Discovery*

Semakin banyak jumlah *container* pembuat aplikasi maka akan semakin sulit untuk dikelola, tetapi dengan menggunakan IP *address* atau nama DNS maka dapat melakukan pelacakan *container* secara *otomatis*.

2) *Load Balancing*

Fungsi *load balancing* yaitu penyeimbang *trafik*/beban. *Load balancing* dapat mengelola *trafik* sehingga ketika terdapat lonjakan *trafik* yang cukup besar, maka Kubernetes mampu membagi *load* yang ada agar aplikasi dapat berjalan lebih *stabil*.

3) Orkestrasi Ruang Penyimpanan

Kubernetes memungkinkan untuk melakukan *mount* pada *storage* yang diinginkan, bisa pada ruang penyimpanan lokal atau yang berbasis *cloud storage.*

4) *Rollout* dan *Rollback Otomatis*

Rollout dan *Rollback Otomatis* memungkinkan untuk melakukan *deployment* menggunakan *file* YAML, sehingga dapat mendeskripsikan *deployment* sesuai dengan kebutuhan. *File* YAML dapat digunakan untuk melakukan *deployment* baru atau mengubah *deployment* yang sudah ada. Fitur ini juga dapat melakukan *rollback* jika terjadi kendala pada *deployment*. Semua data tersimpan secara otomatis.

5) *Bin Packing Otomatis*

Fitur *bin packing otomatis* memungkinkan untuk bisa mengatur kapasitas CPU dan RAM yang spesifik disetiap kontainer. Ketika limit kapasitas sudah ditentukan, maka aplikasi tidak akan berebut *resource* yang ada dan dapat menghemat sumber daya.

6) *Self-healing*

Kubernetes dapat melakukan *self-healing* dengan melakukan pemeriksaan, *restart error*, ataupun mengganti dan memastikan container yang sudah tidak bisa menanggapi *request*. Kubernetes tidak akan memberikan trafik ke *container* sampai benar-benar siap untuk menerima *request (ready to serve)*. Bisa disimpulkan jika *container* akan saling melakukan *back up otomatis* jika *container* yang lain tidak berfungsi [15].

2.2.4 *Load Balancer Service*

Load balancer bertugas sebagai *management traffic* yaitu pendistribusi tugas kepada masing-masing sumber daya, agar sumber daya yang tersedia dapat bekerja dengan optimal. Dan sumber daya digunakan secara seimbang pemanfaatannya, tidak ada yang kelebihan atau kekurangan. Serta memastikan semua sumber daya dapat bekerja dengan efisien.

Kubernetes mempunyai dua jenis *load balancer*, diantaranya yaitu :

1. *Load balancer internal* : bertanggung jawab untuk merutekan permintaan antar *container* dari *Virtual Private Cloud* yang sama. *Load balancer* juga melakukan penyeimbangan beban dalam *cluster*.
2. *Load balancer eksternal* : bertanggung jawab untuk menetapkan *request* masuk ke HTTP *web server*. Berdasarkan sumber daya yang ada pada *container*,

request HTTP *web server* dibagikan ke masing-masing *container*, yang diidentifikasi oleh alamat IP *container* [16].

Load balancer mempunyai beberapa manfaat utama jika ditambahkan pada *service* Kubernetes yaitu meliputi:

1. Menangani lalu lintas tinggi selama jam sibuk. Dengan cara ini permintaan ditangani dengan cara yang paling efisien dan latensi tidak memengaruhi pengguna akhir.
2. Menghindari kemacetan di titik mana pun penyeimbang beban memungkinkan perpindahan lalu lintas.
3. Memprediksi perilaku lalu lintas berdasarkan data historis, di mana beberapa penyeimbang muatan juga dapat menyarankan tindakan korektif.
4. Menugaskan server alternatif setiap kali server tertentu mati atau operasi pemeliharaan terjadi.
5. Mempertahankan kecepatan sistem bahkan menjalankan dua versi berbeda dari aplikasi yang sama [16].

2.2.5 *Denial of Service* (DoS)

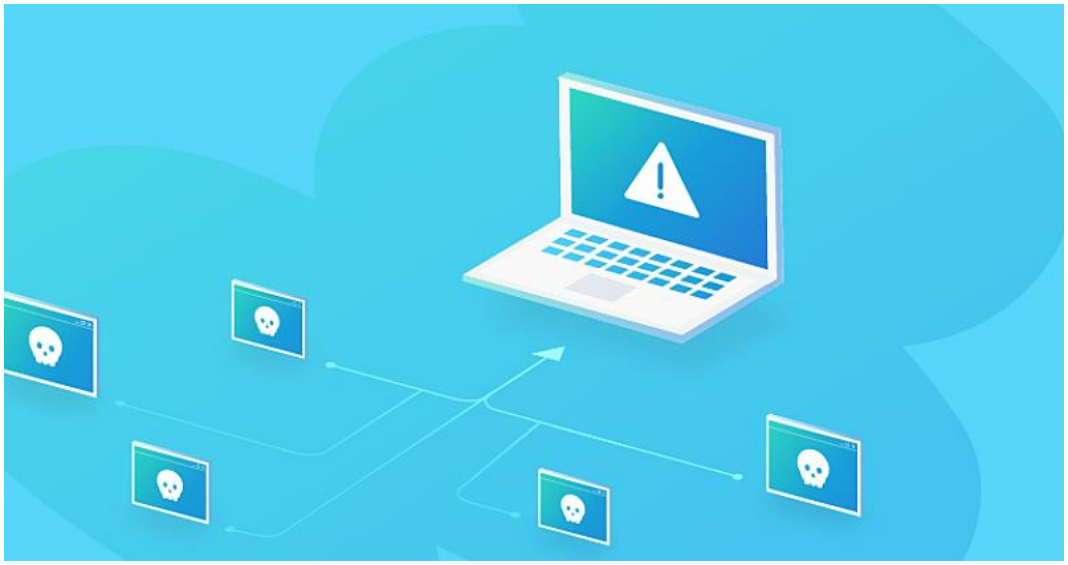
Serangan *Danial of Service* (DoS) adalah adalah jenis serangan terhadap suatu mesin yang mengganggu *traffic* dengan tujuan agar mesin tidak berfungsi dengan baik. Serangan DoS biasanya bekerja dengan cara membanjiri mesin yang ditargetkan dengan mengirim *traffic* dalam jumlah besar, hingga mesin tidak dapat me*respons* semua *traffic* yang masuk [17], secara tidak langsung juga mencegah pengguna lain untuk memperoleh akses layanan dari komputer yang diserang tersebut [4] hal ini cukup berbahaya karena dapat mengakibatkan mesin *down*. Serangan DoS dapat dilakukan menggunakan dua mesin, yaitu mesin sebagai penyerang dan satu mesin sebagai korban yang diserang [17].

Bentuk serangan *Denial of Service* (DoS) pada awalnya hanya ada serangan SYN *Flooding Attack*, yang pertama kali ada pada tahun 1996, dimana SYN *Flooding Attack* bekerja dengan cara melakukan eksploitasi terhadap kelemahan yang berada di dalam protokol *Transmission Control Protocol* (TCP). Setelah berkembangnya serangan pada *protocol* TCP tersebut, mulai berkembang serangan lain yaitu dengan melakukan eksploitasi kelemahan pada sistem operasi, layanan jaringan atau aplikasi untuk menjadikan sistem, layanan jaringan, atau

aplikasi tersebut tidak dapat melayani pengguna, atau bahkan merasakan *crash*. Beberapa *tools* yang digunakan untuk menjalankan serangan DoS banyak dikembangkan setelah munculnya serangan pada protokol TCP, bahkan beberapa *tools* dapat diperoleh secara bebas termasuk pada serangan Bonk, LAND, *Smurf*, *Snork*, WinNuke, dan *Teardrop*.

Meskipun banyak bermunculan jenis serangan *Denial of Service* (DoS) yang baru, serangan terhadap protokol TCP merupakan serangan *Denial of Service* (DoS) yang sering diterapkan. Dikarenakan serangan selain pada protokol TCP seperti halnya memenuhi ruangan hard disk dalam sistem, mengunci salah salah satu akun pengguna yang valid, atau memodifikasi tabel routing dalam sebuah router memerlukan akses yang harus ditembus pada jaringan terlebih dahulu, yang kemungkinan terakses pada jaringan tersebut kecil, terutama jika sistem jaringan yang akan diserang telah diperkuat menggunakan *security* [18].

Cici-ciri mesin yang mengalami serangan DoS yaitu peningkatan *traffic* pada *bandwith* yang padat dan terjadi dalam waktu singkat. Ciri kedua yaitu alamat IP *client* mempunyai *profile* atau perilaku yang sama. Ciri lain yaitu penggunaan CPU yang terus meningkat padahal tidak ada aktifitas dan yang terakhir yaitu koneksi internet yang melambat [19].



Gambar 2.3 Penyerangan *Denial of Service* (DoS)

Pada gambar 2.3 terdapat simulasi penyerangan *Denial of Service* (DoS), dimana pada kondisi tersebut server mendapat serangan oleh penyerang secara terus menerus dengan banyak *user attacker*.

Cara Mengatasi Serangan DoS :

1. Menggunakan *firewall* untuk menghindari serangan yang bertujuan untuk menyerang data – data yang ada di komputer Anda.
2. Melakukan *blocking* terhadap IP yang terlihat mencurigakan. Jika *port* telah dimasuki, maka komputer Anda akan terkuasai oleh penyerang. Cara untuk mengatasinya yaitu dengan menggunakan *firewall* yang di kombinasikan dengan IDS (*Instrusion Detection System*).
3. Menolak paket data dan mematikan service UDP (*User Datagram Protocol*).
4. Menggunakan anti virus yang dapat menangkal serangan data seperti Kapersky.
5. Melakukan *filtering* pada permintaan ICMP (*Internet Control Message Protocol*) *echo* pada *firewall* [20].

2.2.5.1 *TCP Flood*

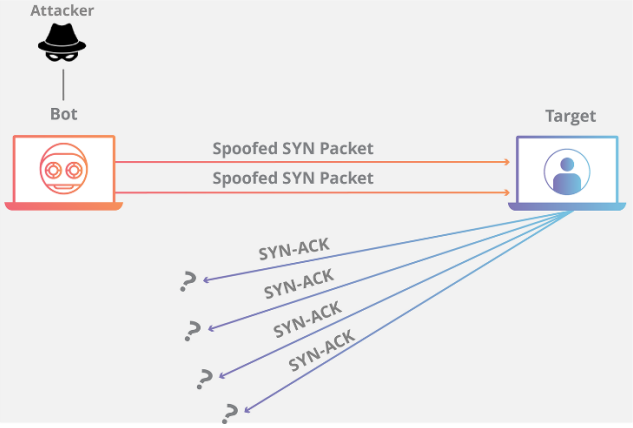
TCP *flood* adalah salah satu serangan *Denial of Service* (DoS) tertua namun masih sangat populer. Serangan yang paling umum melibatkan pengiriman banyak paket SYN ke korban. Serangan dalam banyak kasus akan memalsukan IP SRC yang berarti bahwa balasan (paket SYN+ACK) tidak akan kembali ke sana [21].

Ketika klien dan server membuat TCP normal "*three-way handshake*", pertukaran terlihat seperti ini:

1. Klien meminta koneksi dengan mengirimkan pesan SYN (*synchronize*) ke server.
2. Server mengakui dengan mengirimkan pesan SYN-ACK (*synchronize-acknowledge*) kembali ke klien.
3. Klien merespons dengan pesan ACK (*acknowledge*), dan koneksi dibuat [22].

Serangan TCP *flood* mengeksploitasi kerentanan yang diketahui dalam 3-*way-handshaking* yang memulai koneksi TCP. Tujuan dari TCP *flooding* adalah untuk menghabiskan *backlog* dari *Transmission Control Block* (TCB) yang menyimpan semua informasi tentang koneksi. Hal ini dilakukan dengan mengirimkan sejumlah besar permintaan SYN ke *server*. *Server* membalas permintaan dengan mengirimkan paket SYN + ACK dan menunggu *respons* ACK

dari klien. Pengguna jahat tidak mengirim paket ACK dan *server* menunggu pesan ACK yang tidak ada. Karenanya, antrian *buffer server* menjadi penuh dan permintaan valid yang masuk dibatalkan [23].

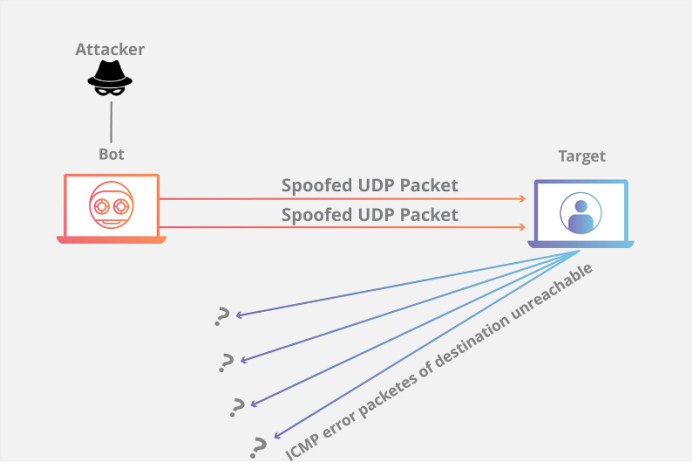


Gambar 2.4 DoS TCP *Flood Attack* [24]

Pada gambar 2.4 skenario penyerangan TCP flood, attacker akan mengirimkan data menuju ke container, lalu container akan memberikan acknowledge atau respon bahwa paket tersebut telah sampai. Tetapi dengan attacker yang mengirimkan pesanan dengan sangat banyak, sehingga container tidak dapat merespons semua permintaan yang masuk hingga dapat menyebabkan server down.

2.2.5.2 *UDP Flood*

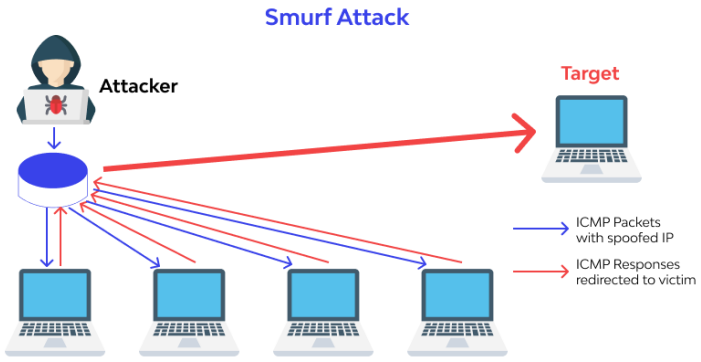
UDP *flood* merupakan serangan yang bersifat *connectionless*, yaitu serangan yang tidak memperhatikan apakah paket yang dikirim diterima semua atau tidak, karena paket pada UDP tidak dimintai respon oleh pengirim atau penerimanya. Cara kerja dari UDP *flood* sendiri yaitu dengan menempel pada *service* UDP *chargen* di salah satu mesin, yang digunakan untuk penyerangan, yang di program untuk meng-*echo* setiap kiriman karakter yang di terima melalui *service chargen*. Hal ini menyebabkan paket UDP di *spoofing* antara ke dua mesin tersebut, sehingga terjadi banjir tanpa henti kiriman karakter yang tidak berguna antara ke dua mesin tersebut [25].



Gambar 2.5 DoS UDP *Flood Attack* [26]

2.2.5.3 *Smurf Attack*

Smurf attack merupakan penyerangan pada fitur spesifikasi IP yang dikenal sebagai *direct broadcast addressing*. *Smurf hacker* biasanya membanjiri *router* dengan paket permintaan *echo Internet Control Message Protocol* (ICMP) yang dikenal dengan *ping*. Alamat IP yang diserang adalah alamat *broadcast* jaringan, dengan cara router akan mengirimkan permintaan ICMP *echo* ini ke semua mesin yang ada di jaringan. Jika terdapat banyak *host* maka akan terhadi trafik ICMP *echo respons* dalam jumlah yang sangat besar. Akibat dari serangan *smurf attack* ini yaitu jika *hacker* memilih untuk men-*spoof* alamat IP sumber permintaan ICMP, akibatnya ICMP trafik akan membuat kemacetan pada *traffic* mesin yang diserang [25].



Gambar 2.6 DoS *Smurf Attack* [26]

2.2.6 *Web Server*

Web server merupakan *software* yang dapat digunakan untuk menyediakan layanan data dan media pada jaringan komputer atau *internet* dengan menggunakan protokol HTTP dan HTTPS serta mengirimkannya dalam bentuk halaman *web* dalam bentuk file HTML. Fungsi utama dari *web server* yaitu untuk mengeksekusi *file* permintaan dari *client* melalui protokol komunikasi seperti *file* teks, video, gambar, dll [27].

Salah satu jenis web server yang banyak digunakan yaitu *Nginx*. Adanya *Nginx* dalam *container* dapat membuat *web server* menjadi lebih kuat, fleksibel, dan mampu memutuskan pada *web server* mana yang akan diadopsi tergantung dari kebutuhan pengguna. Pada *web server*, *cocker container* juga digunakan untuk mendefinisikan baris kode pembuatan *image Docker*. Kemudian mengatur direktori *default* aplikasi dalam *container* pada *web server*. Hubungan antara *web server* dengan *client*, dihubungkan melalui mesin *server* yang dapat dikomunikasingan melalui jarangin *internet* [28].

2.2.7 Parameter *Benchmark*

Benchmark merupakan proses membandingkan suatu aspek dari sebuah organisasi dengan aspek yang sebanding milik organisasi yang dianggap terbaik di industri yang sama atau pasar yang lebih luas [29].

Benchmarking dapat diklasifikasikan berdasarkan dua kategori, yaitu berdasarkan subjeknya dan berdasarkan objek yang diamati.

1. *Benchmarking* Berdasarkan Subjeknya :
   a. *Benchmarking* Internal

Benchmarking internal adalah melakukan perbandingan antara operasi atau proses yang sejenis dalam korporasi. Umumnya hal ini diterapkan pada perusahaan yang memiliki cabang atau anak perusahaan agar memiliki standar yang sama dengan perusahaan utama.

   b. *Benchmarking* Eksternal

Bentuk *benchmarking* yang dilakukan dengan membandingkan suatu organisasi dengan organisasi lain di industri yang sama. *Benchmarking* eksternal dapat dibagi 2, yaitu :

1) *Competitive benchmarking*, artinya suatu perusahaan membandingkan dirinya dengan perusahaan lain yang merupakan pesaing utama.
2) *Non-competitive benchmarking*, artinya suatu perusahaan membandingkan dirinya dengan perusahaan lain yang bukan pesaing dan dari industri yang berbeda. *Benchmarking* ini dibagi dua, yaitu; *Functional*, yaitu membandingkan fungsi yang sama dari organisasi yang berbeda pada berbagai industri. Serta *Generic*, yaitu membandingkan proses bisnis dasar yang cenderung sama pada setiap industri.

2. *Benchmarking* Berdasarkan Objeknya
   a. *Strategic Benchmarking*, proses mengamati bagaimana strategi orang atau perusahaan lain dapat mengungguli para pesaingnya di indutri tertentu.
   b. *Process Benchmarking*, proses mengamati dan membandingkan proses-proses kerja atau sistem operasi tertentu (misal pembayaran, rekrutmen, komplain pelanggan, dan lainnya).
   c. *Functional Benchmarking*, proses mengamati dan membandingkan fungsi kerja sejenis pada industri yang sama untuk meningkatkan operasional pada fungsional tersebut.
   d. *Performance Benchmarking*, proses mengamati dan membandingkan kinerja pada produk atau jasa, seperti; fitur produk, harga, kualitas teknis, dan lainnya.
   e. *Product Benchmarking*, proses mengamati dan membandingkan produk sendiri dengan produk pesaing untuk mengetahui apa kekuatan *(strength)* dan kelemahan *(weakness)* produknya.
   f. *Financial Benchmarking*, proses mengamati dan membandingkan kekuatan finansial untuk mengetahui daya saing suatu organisasi terhadap pesaing [30].

2.2.6.1 *Response Time*

Response time merupakan waktu rata-rata yang diperlukan untuk menanggapi setiap permintaan *client* yang disimulasikan. Waktu respon diambil dari total waktu dibagi dengan jumlah respon menggunakan satuan (*second*) pada

setiap pengujian benchmark. Persamaan 2.1 merupakan persamaan perhitungan *response time*.

$$Response\ time = \frac{\text{total waktu (s)}}{\text{jumlah respon (n respon)}} \quad \textbf{(2.1)}$$

2.2.6.2 *Throughput*

Throughput adalah istilah sebenarnya dari *bandwidth*, yang diukur menggunakan satuan waktu tertentu dan pada kondisi jaringan tertentu yang digunakan untuk melakukan transfer data dengan ukuran tertentu [31]. *Throughput* menunjukan seberapa banyak kapasitas *bandwidth* yang sebenarnya terpakai. Jadi *throughput* merupakan jumlah data yang benar-benar terkirim dalam satu waktu tertentu [32]. Dapat dianalogikan dengan jalan tol, di mana jika kapasitas jalan tol adalah 30 mobil diwaktu tertentu, itu menunjukan *bandwidth*. Dan dalam waktu tertentu tersebut, yang ada hanya ada 3 mobil saja [32].

$$Throughput = \frac{\text{jumlah paket data yang dikirim (bit)}}{\text{waktu pengiriman paket (s)}} x8 \quad \textbf{(2.3)}$$

2.2.6.3 *CPU Usage*

CPU *usage* adalah penggunaan yang tinggi pada CPU yang tinggi [33]. CPU dilakukan didalam pemrosesan komputasi dan ditentukan dalam unit CPU Kubernetes [34]. Semakin kecil penggunaan CPU menunjukkan bahwa mesin berjalan dengan lebih efisien dan menggunakan *resource cluster* secara menyeluruh [35].

2.2.6.4 *Memory*

Memory usage menampilkan jumlah memori yang tersedia di sistem *user*, serta memori yang saat ini digunakan oleh semua aplikasi, termasuk *Windows* itu sendiri. Memori yang tersedia di sistem *user* adalah jumlah dari semua memori fisik yang terpasang di sistem *user* dan *file* halaman di *hard disk*, yang digunakan untuk melengkapi memori fisik [36].