BAB II
TINJAUAN PUSTAKA

2.1 PUSH TECHNOLOGY

2.1.1 Pengertian

Push technology merupakan sarana yang relatif baru untuk mengotomatisasi pengiriman berita dan informasi ke perangkat telpon genggam melalui internet [20]. *Push technology* dapat mendukung aliran data yang dapat diterima langsung oleh penerima [18]. Dalam segi waktu, pengiriman informasi oleh *push technology* lebih *real-time* daripada *pull technology* [28].

Pembagian *push technology* menurut mekanisme pengiriman informasi [21] terbagi atas.

1. *Client Pull vs. Server Push*

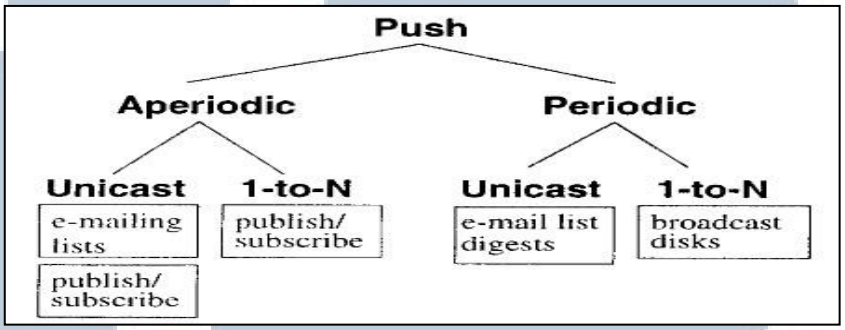
   Client Pull melakukan request terhadap *server* mengenai suatu informasi, kemudian *server* akan mencari dimana informasi tersebut sebelum dikirim ke *client*. Sedangkan *server push*, *server* akan menginisiasi informasi yang akan dikirim sesuai keinginan *client* dan mengirimnya ketika informasi tersebut dibutuhkan dengan inisiasi oleh *server*.

2. *Aperiodic vs. Periodic*

   Aperiodic adalah ketika terdapat sebuah *event* baik *data request* maupun *transmission* telah di-*trigger* oleh *event request* atau *data update*. Sebaliknya, *periodic* bekerja sesuai jadwal yang telah ditentukan sebelumnya.

3. *Unicast vs. 1-to-N*

   Unicast adalah ketika pengiriman data melibatkan satu mesin (sumber) ke mesin lainnya (penerima). Sedangkan 1-to-N mengijinkan banyak penerima meskipun data berasal dari satu sumber.



Gambar 2.1 Pembagian *push* berdasarkan pengiriman informasi [21]

Dengan teknik pengiriman informasi secara *aperiodic, client* dapat melakukan *subscribe* terhadap suatu *topic*. Saat terdapat *update* (*event-trigger)* pada *server*, *server* melakukan *publish* mengenai *update* tersebut pada *topic* sehingga *client* dapat menerima *update* dari *server*.

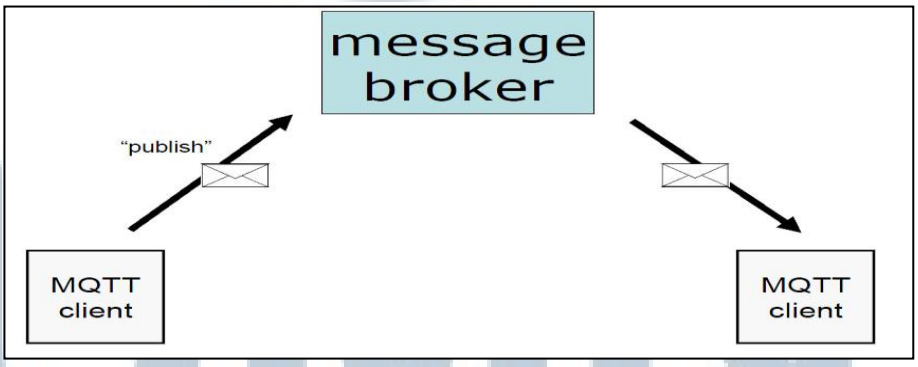
2.1.2 MQ Telemetry Transport (MQTT)

MQTT adalah protokol konektivitas *machine-to-machine* (M2M) yang didesain sebagai *publisher/subscriber* yang mengirim pesan dengan sangat ringan. MQTT ditemukan oleh Dr. Andy Stanford-Clark dari IBM dan Arlen Nipper dari Arcom (sekarang Eurotech), pada 1999 [23]. MQTT menggunakan metode *publish/subscribe* dalam pengiriman informasinya.

Pada MQTT terdapat tiga entitas yang berperan dalam pengiriman informasi yaitu, *server* (penyedia informasi), *message broker*, dan *client*

(penerima informasi). Langkah yang ditempuh untuk mengirim *informasi* adalah sebagai berikut [7]

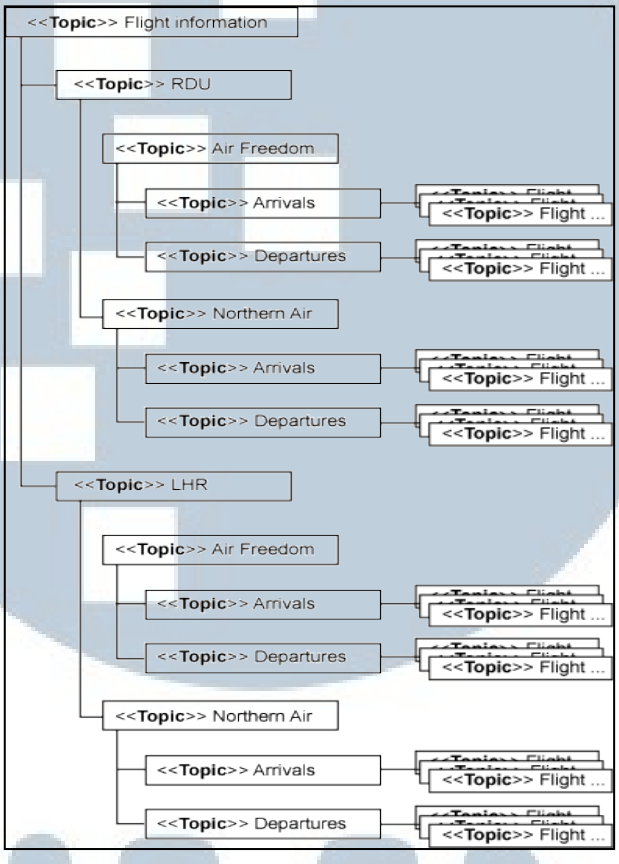
1. *Client* pengirim dan penerima harus melakukan *subscribe* terhadap *topic* yang sama agar pesan dapat terkirim,
2. *Client* pengirim informasi melakukan *publish* informasi kepada *broker* memuat *payload* berupa informasi yang ingin dikirim ke *client* penerima dan *topic* yang ingin di*publish*,
3. *Broker* akan meneruskan dengan cara *broadcast payload* tersebut sesuai dengan *topic* yang sudah ditentukan oleh *client pengirim.*
4. *Client penerima* yang *subscribe* terhadap *topic* akan menerima pesan berupa informasi terbaru mengenai *topic* yang di-*subscribe* nya.



Gambar 2.2 Proses pengiriman pesan antar *client* oleh MQTT [7]

MQTT membutuhkan informasi bahwa *client* yang *subscribe* pada suatu *topic* masih aktif atau tidak. Hal tersebut menuntut *subscriber* untuk terus mengabarkan kepada *broker* bahwa dirinya masih hidup atau tersambung. Untuk

itu dibutuhkan mekanisme yang berfungsi me-*maintain* koneksi dengan cara melakukan *publish* ke *topic* tertentu secara periodik [3].



Gambar 2.3 Hierarki *topic* [14]

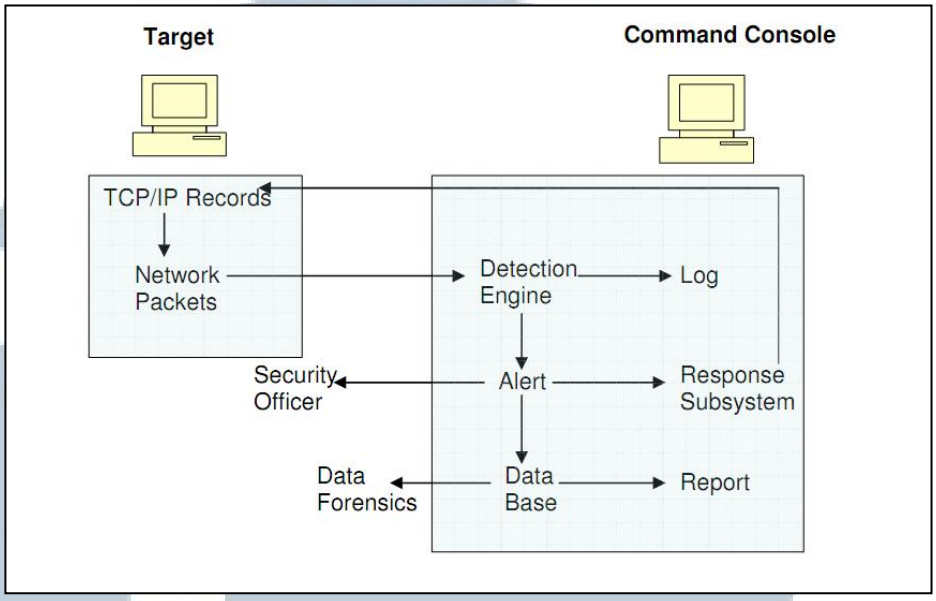
Topic dapat digunakan sebagai media penghubung antara *publisher* dan *subscriber*. *Topic* sendiri dapat digunakan dengan sistem *tree*. Namun *topic* yang memiliki hierarki lebih tinggi tidak dapat melihat pesan yang di-*publish* pada *topic* dibawahnya [14]. Dengan demikian, *topic* dapat digunakan untuk mengirim pesan ke *subscriber* yang bersifat unik.

2.2 Snort

2.2.1 Pengertian

Intrusion Detection System (IDS) adalah sebuah *device* atau *software* yang berfungsi memantau jaringan atau sebuah sistem kemudian menganalisanya untuk menemukan tanda adanya kemungkinan insiden keamanan, *malicious activities* atau pelanggaran aturan dan menghasilkan laporan [17]. Selanjutnya, Snort dikembangkan oleh Sourcefire dengan Martin Roesch sebagai *founder*nya. Sebanyak empat ratus ribu pengguna terdaftar menggunakan Snort dan mendapat pengakuan *de facto* sebagai standar IDS [27].

IDS merupakan kumpulan teknik dan metode yang digunakan untuk mendeteksi aktivitas yang mencurigakan di bagian *network* maupun *host* [16]. IDS terbagi atas dua kategori dasar yaitu, berdasarkan *signature* pada paket data (*signature-based* IDS) dan berdasarkan keanehan pada sistem (*anomaly detection systems*) karena penyusup memiliki *signature* layaknya *virus* pada computer yang dapat dideteksi [16].



Gambar 2.4 Skema penanganan *alerts* NIDS [17]

IDS membaca paket jaringan kemudian menjalankan mesin pendeteksi. Mesin pendeteksi akan memindai paket tersebut yang menghasilkan *Log* dan *Alert*. *Log* merupakan hasil pemindaian yang menjadi catatan tersendiri dan disimpan pada direktori tertentu. *Alert* akan diinformasikan kepada petugas keamanan dalam hal ini *network administrator* dan menghasilkan *response subsystem*. *Alert* tersebut disimpan ke dalam *database* Snort yang dapat digunakan untuk menghasilkan laporan oleh Data Forensics.

2.2.2 Cara Kerja

Snort berkerja sebagai *packet sniffer* untuk membaca dan menganalisa paket pada sistem dan jaringan [16]. Snort menggunakan *alerts* untuk merekam penemuannya dan mengkomunikasikannya kepada pengguna. *Alerts* adalah pesan yang didapat ketika mekanisme pendeteksi (*preprocessor* atau Snort *rule*) cocok

dengan *event* yang terjadi. Berdasarkan tujuan penganalisaan data [17], dapat dikelompokkan sebagai berikut.

1. *Real-time Alerting*

   Proses yang membutuhkan komponen *real*-time untuk menganalisa *alert stream* yang berfungsi untuk meningkatkan analisa berdasarkan kombinasi yang spesifik. Analisa *alert* pada Snort bersifat kompleks dan membutuhkan cara tertentu untuk mendapatkan *alert* yang bersifat *real-time*.

2. *Attack detection and verification*

   Snort tidak mendeteksi serangan melainkan mendetasi tiap-tiap aktivitas yang terjadi pada system, dengan memantau *traffic* pada mesin atau *services*. Hal tersebut merupakan *attack detection* yang memungkinkan dideteksinya serangan dengan menerapkan *rules* tertentu sehingga ketika *rules* tersebut cocok dengan hasil pantauan maka dapat diindikasikan adanya serangan yang telah terjadi. Proses selanjutnya adalah *attack verification* yang berfungsi mencari *false positives* dan meyakinkan laporan penyerangan tersebut mengandung serangan yang benar – benar terjadi.

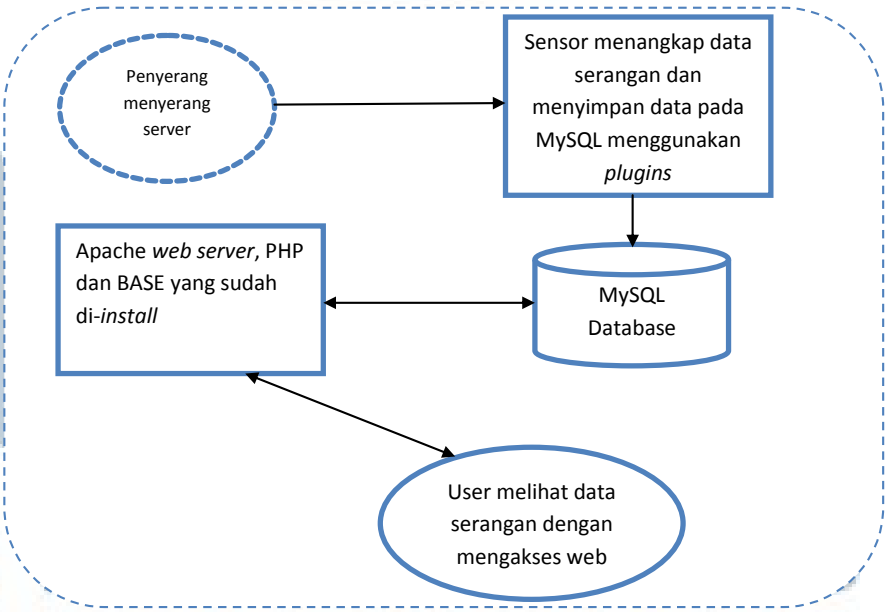
3. *Incident Analysis*

   Ketika sistem mengalami serangan, Snort akan melakukan analisa insiden selama mencari tahu apa yang sebenarnya terjadi. Analisa insiden mengandung informasi apa yang terjadi, kapan kejadian tersebut terjadi, bagaimana hal tersebut dapat terjadi, apakah termasuk *isolated events*,

sistem apa yang terlibat, darimana serangan tersebut datang, dan bagaimana *impact* nya terhadap sistem.

4. *Reporting*

   Pada akhirnya dibutuhkan laporan penemuan indikasi serangan beserta informasi mengenai serangan tersebut baik bagi entitas internal maupun eksternal. Laporan tidak hanya mengenai insiden yang terjadi, melainkan laporan mengenai serangan yang sering terjadi, *alert* yang sering di-*trigger* dan sistem apa yang sering diserang (*hole*). Selain itu, dibutuhkan pengaturan untuk laporan agar dapat diakses dengan mudah dan dimengerti seperti penggunaan *web-based front end* (BASE) untuk mempermudah membaca laporan.



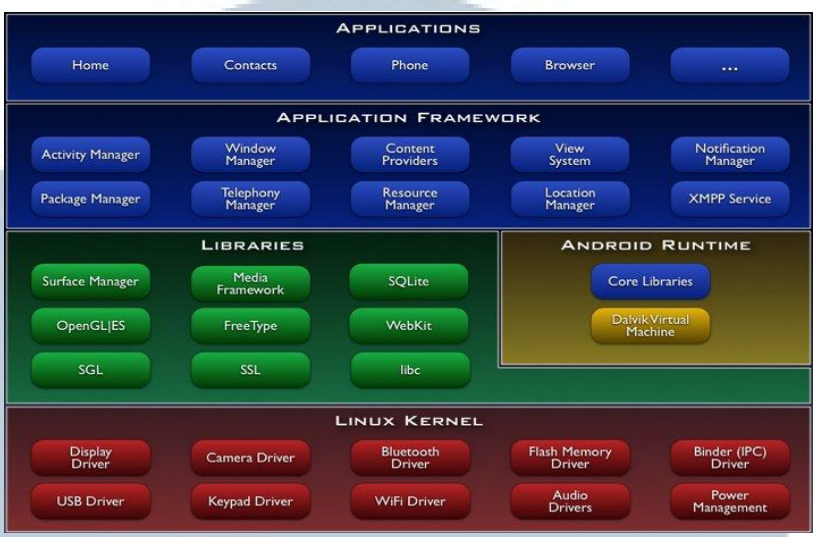
Gambar 2.5 Bagan diagram IDS Snort dan BASE.

Ketika penyerang melakukan serangan terhadap *server*, Snort akan membaca paket data yang diterima. Paket data tersebut akan diolah dan dimasukkan ke dalam *database* Snort menggunakan MySQL *plugins* yang telah dikonfigurasi saat instalasi Snort. Kemudian dari *database* tersebut, aplikasi seperti BASE akan mengolah data yang ada dan menampilkannya melalui *web* sehingga *network administrator* dalam hal ini sebagai pengguna, dapat melihat hasil *capture* Snort melalui *browser* dengan mengakses *web* BASE tersebut.

2.3 ANDROID

2.3.1 Pengertian

Android merupakan sebuah *Java-based operating system* yang bersifat *open source* dengan Google sebagai pemiliknya [10]. Android menyajikan tampilan antarmuka atau *Graphical User Interface* (GUI) dengan format XML [11] yang menjembatani *user* dan sistem pada Android *application* sehingga memungkinkan Android untuk mempresentasikan informasi melalui aplikasi kepada *user* dengan baik. Arsitektur pada Android memiliki empat komponen besar seperti yang digambarkan pada gambar 1.6 berikut.



Gambar 2.6 Arsitektur Android [11]

Setiap aplikasi Java pada Android berjalan di dalam instansi dari *Dalvik Virtual Machine*. Setiap instansi tersebut berjalan di dalam prosesnya yang bersifat unik dalam rangka mengisolasi aplikasi-aplikasi di dalam *platform* [11]. *Middleware* Android menyediakan kumpulan Java dan *native libraries* untuk melakukan pengembangan aplikasi seperti *management* daur hidup aplikasi. *Libraries* tersebut dikenal dengan *system service* berfungsi membantu fungsional dari sistem utama [11]. *Middleware* tersebut selalu memantau setiap aplikasi yang berjalan di *background* maupun di *foreground.*

2.3.2 Activity

Pada Android, *Activity* menggambarkan *visual user interface* untuk sebuah pekerjaan yang diakses [9]. *Activity* digolongkan kepada dua buah contoh sebagai berikut.

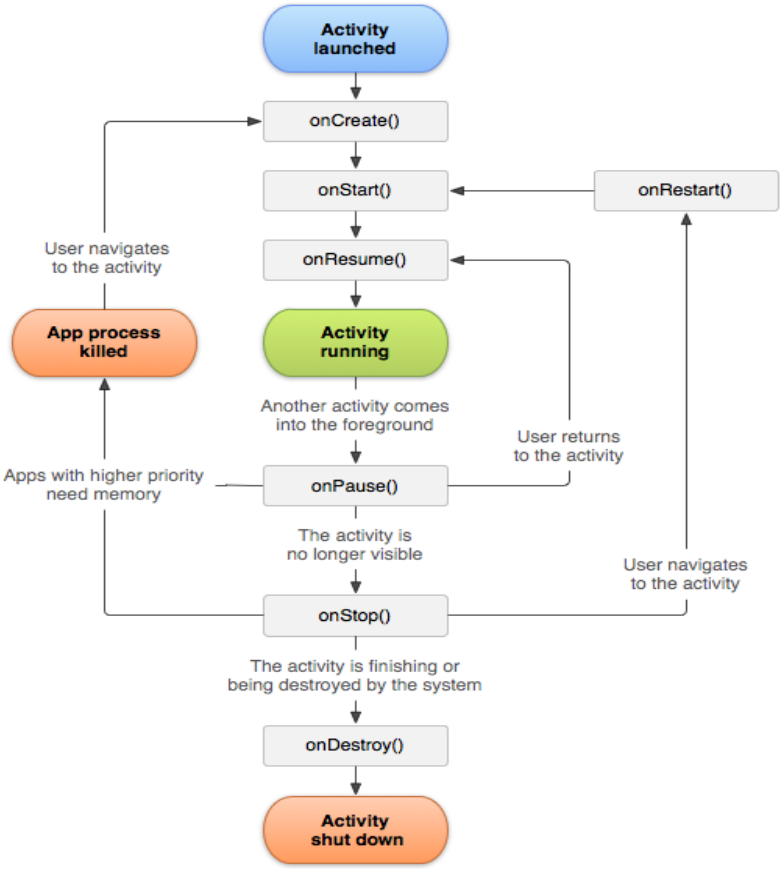
1. *Individual Activity*

Merupakan *Activity* yang berfungsi menampilkan *data* langsung tanpa mendapat *input* dari *Activity* lainnya. Seperti menampilkan informasi mengenai suatu hal saat aplikasi tersebut dibuka (tanpa menerima *input* dari *Activity*), dan

2. *Sequence Activity*

Merupakan *Activity* yang berjalan sekuensial dan menerima *input* dari *Activity* sebelumnya atau dipanggil dari *Activity* sebelumnya.

Setiap *Activity* memiliki proses hidup (*life-cycle process*) yang mana setiap proses tersebut dapat *overwrite* sesuai kebutuhan. Baik saat *Activity* tersebut dipanggil, saat berpindah ke *Activity* lainnya, maupun saat *Activity* tersebut ditutup [9]. Adapun proses tersebut digambarkan pada Gambar 2.6 sebagai berikut.



Gambar 2.7 Activity Life Cycle [9]

2.3.3 Service

Service merupakan komponen dari aplikasi yang dapat berjalan dalam kurun waktu yang lama di *background* dan tidak menyediakan *user interface* [11]. Selain itu, komponen dapat mengikat *service* untuk berinteraksi dengannya melalui *interprocess communication* (IPC).

Service memiliki dua buah form [11] sebagai berikut

1. *Started*

Sebuah *service* berada dalam kondisi *started* ketika komponen aplikasi (seperti sebuah Activity) memanggil fungsi `startService()`. Sekali *service* berjalan, *service* tersebut dapat berjalan di *background* meskipun komponen yang memanggilnya ditutup (*destroyed*). Contoh penggunaannya adalah ketika dibutuhkan sebuah komponen untuk melakukan *download* atau *upload* sebuah *file* yang dapat berjalan meskipun aplikasi yang menginisialisasi nya ditutup atau hilang.

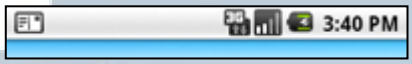
2. *Bound*

Service berada pada kondisi *bound* ketika komponen aplikasi mengikatnya dengan memanggil fungsi `bindService()`. *Service* dalam kondisi demikian, menawarkan *client-server interface* yang mengijinkan komponen berinteraksi dengan *service*, mengirimkan *request*, mendapatkan hasilnya bahkan melakukan pemanggilan proses melalui *interprocess communication* (IPC). *Service* seperti ini berjalan selama komponen aplikasi terikat padanya. Banyak komponen yang dapat terikat kepada satu *service*, jika komponen tersebut sudah tidak terikat, maka *service* akan dihilangkan.

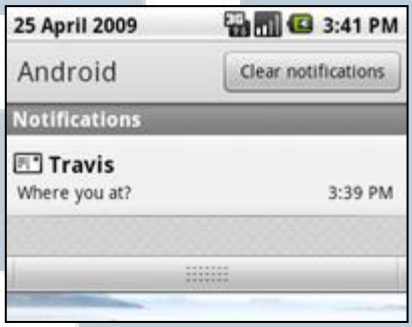
2.3.4 Notification

Android memiliki tempat tersendiri untuk menampung notifikasi yang dimiliki oleh aplikasi – aplikasinya. Tempat tersebut dinamakan *Status Bar Notification*. Pada tempat ini, notifikasi yang masuk digambarkan dengan logo

aplikasi. Di dalam nya terdapat *Notification Window* yang dapat berfungsi sebagai jendela tampilan terhadap notifikasi yang masuk. Hal tersebut seperti yang digambarkan oleh Gambar 2.8 dan 2.9.



Gambar 2.8 *Status Bar* pada Android



Gambar 2.9 *Notification Window* pada Android

Sebuah *activity* atau *services* pada Android dapat menginisiasi notifikasi pada *Status Bar* dengan memanfaatkan *Notification Manager*. Namun *activity* hanya berjalan saat aplikasi diakses oleh *user* sehingga penggunaan notifikasi dilakukan oleh *service* yang dapat berjalan pada *background*.

Pembuatan notifikasi didasari oleh beberapa hal sebagai berikut

1. Icon yang akan ditampilkan pada *Status Bar*,
2. Judul dan pesan yang akan ditampilkan pada *Notification Window*, dan
3. Sebuah `PendingIntent` yang akan dipanggil ketika notifikasi tersebut diakses atau dipilih oleh pengguna.

Langkah – langkah pembuatan notifikasi pada Android sebagai berikut

1. Memanggil *class* `NotificationManager`, membuat *instance* dan menentukan *icon*, teks, dan kapan notifikasi akan tampil seperti yang dijabarkan dalam *code* berikut

```
String ns = Context.NOTIFICATION_SERVICE;
NotificationManager mNotificationManager = (NotificationManager)
getSystemService(ns);

int icon = R.drawable.notification_icon;
CharSequence tickerText = "Testing";
long when = System.currentTimeMillis();

Notification notification = new Notification(icon, tickerText, when);
```

Gambar 2.10 Contoh kode (1)

2. Menentukan `PendingIntent` yang akan dipanggil saat notifikasi diakses atau diklik oleh pengguna

```
Context context = getApplicationContext();
CharSequence contentTitle = "My notification";
CharSequence contentText = "Hello World!";
Intent notificationIntent = new Intent(this, MyClass.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
notificationIntent, 0);
notification.setLatestEventInfo(context, contentTitle, contentText,
contentIntent);
```

Gambar 2.11 Contoh kode (2)

3. Melakukan *notify* kepada aplikasi sehingga *user* dapat menerima notifikasi.

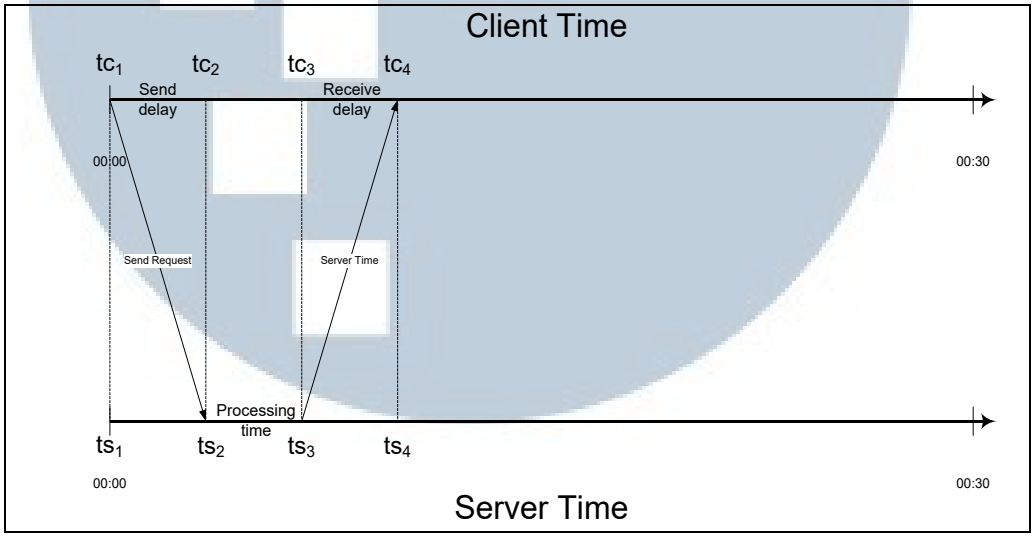
```
private static final int HELLO_ID = 1;
mNotificationManager.notify(HELLO_ID, notification);
```

Gambar 2.12 Contoh kode (3)

2.4 PARAMETER PENGUJIAN

2.4.1 Menghitung perbedaan waktu

Untuk melakukan pengujian dibutuhkan besar perbedaan waktu antara *client* dan *server*. Tujuan perhitungan perbedaan waktu adalah untuk menyamakan waktu *server* dengan waktu *client*. Hal tersebut dilakukan guna mengetahui berapa besar jeda waktu antara insiden dan waktu terima laporan insiden lebih akurat.



Gambar 2.13 Proses pengambilan waktu dari *server* [2]

Sesuai Gambar 2.13 skema pengambilan waktu pada *server* adalah sebagai berikut. *Client* melakukan *request* ke *server* pada tc_1. Kemudian *server* menerima *request* dari *client* dan mengambil waktu pada *server* ts_2 dan menampilkannya ts_3. Hasil tampilan ts_3 diterima oleh *client* ketika tc_4. Dengan demikian besarnya perbedaan waktu antara *client* dan *server* dapat dihitung seperti Gambar 2.14.

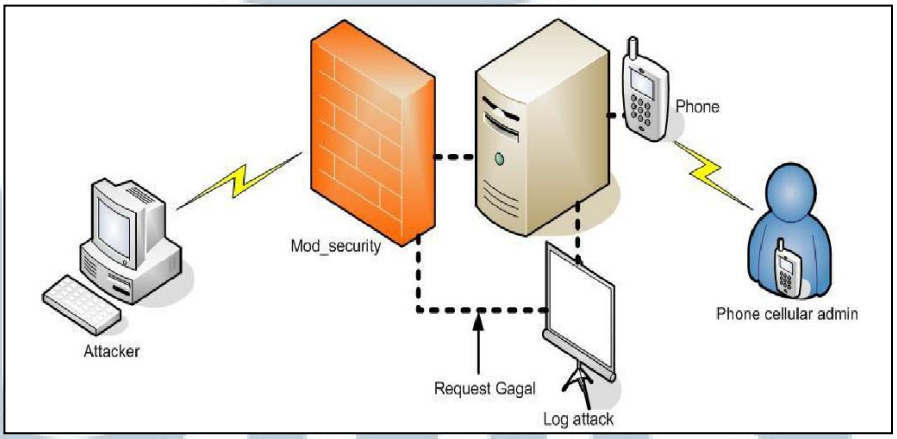
$$\Delta t = ts_i - tc_i$$

Δt = perbedaan waktu antara server dan *client*
ts_i = waktu server
tc_i = waktu *client*

Gambar 2.14 Perhitungan selisih waktu *client* dan *server*

2.5 PENELITIAN TERKAIT

Penelitian terkait mengenai pelaporan intrusi adalah penelitian yang dilakukan oleh Agus Priyo Utomo dan Isbat Uzzin Nadhori [1]. Penelitian menggunakan SMS Gateway sebagai jembatan komunikasi antara *server* dan ponsel pengguna dalam melaporkan intrusi yang terjadi. Gambar 2.15 merupakan diagram umum yang digunakan oleh penelitian sebelumnya.



Gambar 2.15 Diagram Umum Pelaporan Intrusi [1]

Pendeteksian intrusi menggunakan `mod_security` yang merupakan modul milik *webserver* Apache. Ketika pengguna melakukan *request* kepada *server*, *request* tersebut diperiksa berdasarkan *ip address*, *user agent*, *file* yang di-*request* dan *input* yang dikirim kepada *server* [1]. Jika *request* tersebut dianggap

berbahaya oleh `mod_security` maka paket tersebut akan ditolak dan dibuang. Bersamaan dengan itu, informasi mengenai *request* akan disimpan ke dalam *log* dan informasi tersebut akan dikirimkan ke ponsel milik *network administrator* melalui SMS Gateway. Disisi lain, jika request tersebut dianggap tidak berbahaya oleh `mod_security`, maka *request* akan diijinkan untuk dijalankan.