UNIVERSITAS INDONESIA

IMPLEMENTASI DAN ANALISIS KINERJA WIDS (*WIRELESS INTRUSION DETECTION SYSTEM*) UNTUK *MONITORING* KEAMANAN JARINGAN *WIRELESS* TERDISTRIBUSI BERBASIS KISMET

SKRIPSI

RIKA FEBITA
NPM 0806339300

FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK KOMPUTER
DEPOK
JULI 2012

UNIVERSITAS INDONESIA

IMPLEMENTASI DAN ANALISIS KINERJA WIDS (*WIRELESS INTRUSION DETECTION SYSTEM*) UNTUK *MONITORING* KEAMANAN JARINGAN *WIRELESS* TERDISTRIBUSI BERBASIS KISMET

SKRIPSI

Diajukan sebagai salah satu syarat untuk memperoleh gelar sarjana

RIKA FEBITA
NPM 0806339300

FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK KOMPUTER
DEPOK
JULI 2012

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri,

dan semua sumber baik yang dikutip maupun dirujuk

telah saya nyatakan dengan benar.

Nama	**: Rika Febita**
NPM	**: 0806339300**
Tanda Tangan	**:** 
Tanggal	**: 6 Juli 2012**

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh :

Nama	: Rika Febita
NPM	: 0806339300
Program Studi	: Teknik Komputer
Judul Skripsi	: Implementasi dan Analisis Kinerja WIDS (*Wireless Intrusion Detection System*) untuk *Monitoring* Keamanan Jaringan *Wireless* Terdistribusi berbasis Kismet

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Teknik pada Program Studi Teknik Komputer, Fakultas Teknik Universitas Indonesia.

DEWAN PENGUJI

Pembimbing : Muhammad Salman S.T., MIT ( )

Penguji : Yan Maraden ST. MSc. ( )

Penguji : I Gde Dharma Nugraha ST. MT. ( )

Ditetapkan di : Depok
Tanggal : 5 Juli 2012

UCAPAN TERIMA KASIH

Puji syukur saya panjatkan ke khadirat Allah SWT, karena atas segala rahmat dan hidayah-Nya saya dapat menyelesaikan skripsi ini. Saya menyadari bahwa skripsi ini tidak akan terselesaikan tanpa bantuan dari berbagai pihak. Mulai dari proses pembelajaran dan proses penyusunan dari buku skripsi ini, saya ingin mengucapkan terima kasih kepada:

1. Muhammad Salman S.T., MIT, selaku pembimbing telah menyediakan waktu, tenaga, dan pikiran untuk mengarahkan saya dalam penyusunan skripsi ini.
2. Seluruh keluarga besar sivitas akademik Departemen Teknik Elektro Fakultas Teknik Universitas Indonesia yang tidak dapat saya sebutkan satu persatu.
3. Orang tua dan keluarga saya yang telah memberikan bantuan dukungan materil, moral serta do'a.
4. Boma Anantasatya Adhi dan Mega Oktafiani selaku teman yang membimbing dalam pengerjaan skripsi ini serta Ainun Jariyah dan Wenni Indriyani yang membimbing dalam penulisan skripsi ini.
5. Teman-teman satu bimbingan dan satu angkatan saya.

Akhir kata, semoga Allah SWT berkenan membalas segala kebaikan semua pihak yang telah membantu. Semoga skripsi ini bermanfaat bagi perkembangan ilmu pengetahuan.

Depok, 6 Juli 2012

Penulis

Rika Febita

NPM. 0806339300

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini :

Nama	: Rika Febita
NPM	: 0806339300
Program Studi	: Teknik Komputer
Departemen	: Teknik Elektro
Fakultas	: Teknik
Jenis Karya	: Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty Free Right*)** atas karya ilmiah saya yang berjudul :

IMPLEMENTASI DAN ANALISIS KINERJA WIDS (*WIRELESS INTRUSION DETECTION SYSTEM*) UNTUK *MONITORING* KEAMANAN JARINGAN *WIRELESS* TERDISTRIBUSI BERBASIS KISMET

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pengkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok

Pada tanggal : 6 Juli 2012

Yang menyatakan

(Rika Febita)

ABSTRAK

Nama : Rika Febita
Program Studi : Teknik Komputer
Judul : Implementasi dan Analisis Kinerja WIDS (*Wireless Intrusion Detection System*) untuk *Monitoring* Keamanan Jaringan *Wireless* Terdistribusi berbasis Kismet

Banyak pihak yang berusaha memanfaatkan kerentanan dari jaringan WLAN sehingga dibutuhkan suatu WIDS yang *user friendly* dapat mendeteksi adanya serangan dalam jaringan ini. Implementasi WIDS menggunakan Kismet sebagai aplikasi WIDS, Sagan sebagai penghubung Kismet dengan Snorby, dan Snorby sebagai *frontend*. Metode pengujian menggunakan *functionality test* untuk *spoofed* AP, *brute force* WPS, dan *de-authentication flood* dan *response time* untuk *de-authentication flood* saja. Pengujian *de-authentication flood* akan dilakukan 10 kali untuk membandingkan nilai *alert*, *frame*, dan *response time* berdasarkan banyaknya serangan dan peletakan sensor terhadap penyerang.
Untuk penyerang1 pada banyaknya serangan, pada 1, 2, dan 3 serangan, rata-rata *alert* adalah 12 *alert*, 3,8 *alert*, dan 2,3 *alert*, persentase *false negative frame* de-otentikasi yang mengacu kepada 1 serangan adalah 28,43% (2 serangan) dan 44,47% (3 serangan), dan *response time* adalah 0,015 detik, 0,056 detik, dan 0,087 detik. Untuk peletakan sensor, pada ruang yang sama (ruang 1), ruang yang berbeda 1 ruangan (ruang 2), dan ruang yang berbeda 2 ruangan (ruang 3) dari penyerang, rata-rata *alert*-nya adalah 10,6 *alert*, 7,9 *alert*, dan 7,8 *alert*, persentase *false negative frame* de-otentikasi yang mengacu kepada *frame* de-otentikasi yang terdeteksi pada ruang 1 adalah 72,48% dan 77,17%, dan rata-rata *response time* adalah 0,018 detik, 0,046 detik, dan 0,111 detik.
Seiring bertambahnya serangan dan semakin banyak dinding pembatas, *alert* penyerang1 semakin sedikit, dan *false negative frame* de-otentikasi dan *response time* penyerang1 semakin banyak. Oleh karena itu, banyaknya trafik dan peletakan sensor berpengaruh terhadap kinerja WIDS. WIDS dapat bekerja optimal jika berada dalam 1 ruangan dengan AP yang ingin dimonitor dan tidak terlalu banyak trafik. Hal ini untuk menghindari adanya interferensi dan terlalu banyaknya frame yang lalu lalang di udara.

Kata kunci :
WLAN, WIDS, vulnerabilitas, Kismet, Sagan, Snorby, *functionality test*, *spoofed* AP, *brute force* WPS, *de-authentication flood*, *alert*, *frame*, *response time*

ABSTRACT

Name : Rika Febita
Study Program : Computer Engineering
Title : Implementation and Performance Analysis of WIDS (Wireless Intrusion Detection System) for Kismet based Distributed Wireless Network Security Monitoring

Many people that try to exploit the vulnerability of WLAN so it is needed a user friendly WIDS that can detect attacks in these networks. WIDS implementation is using Kismet as WIDS application, Sagan which connects Kismet and Snorby, and Snorby as a frontend. Method of testing for functionality test is using spoofed AP, WPS brute force, and de-authentication flood and the response time for the de-authentication flood. De-authentication flood testing will be performed 10 times to compare the value of alerts, frames, and response time based on the number of attacks and the laying of the sensor against the attacker.
For attacker1 on the number of attacks, at 1, 2, and 3 attacks, the average alert is 12 alerts, 3,8 alerts, and 2,3 alerts, the percentage of de-authentication frame false negative that refers to 1 attack is 28,43 % (2 attacks) and 44,47% (3 attacks), and response time is 0,015 seconds, 0,056 seconds and 0,087 seconds. For sensor placement, in the same room (room 1), a different 1 room (room 2), and different 2 rooms (room 3) from the attacker, the average alert is 10,6 alert, 7, 9 alerts, and 7,8 alerts, the percentage of de-authentication frame false negative are referring to the de-authentication frame that are detected in the room 1 is 72,48% and 77,17%, and the average response time is 0,018 seconds, 0,046 seconds and 0,111 seconds.
As we get more and more attacks and the dividing wall, the less alert from attacker1, and de-authentication frames's false negative and response time from attacker1 is bigger than before. Therefore, the amount of traffic and the placement of the sensors affect the performance of WIDS. WIDS can work optimally if it is in a room with the AP would like to be monitored and not too much traffic. This is to avoid interference and that too many frames passing through the air.

Key words:
WLAN, WIDS, vulnerability, Kismet, Sagan, Snorby, functionality test, spoofed AP, brute force WPS, de-authentication flood, alert, frame, response time

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi sangatlah pesat dan tidak dapat kita hindari. Betapa terasa perbedaannya teknologi informasi di tanah air dari sepuluh tahun silam dengan sekarang. Setiap inovasi teknologi ini tentunya dibuat dengan tujuan positif agar masyarakat lebih merasa nyaman, mudah, hemat, dan ramah terhadap lingkungan.

Salah satu teknologi yang banyak dipakai sekarang adalah jaringan internet. Jaringan internet bisa diakses melalui jaringan kabel maupun nirkabel (*wireless*). Jaringan kabel dapat membatasi pengguna dalam mengakses internet, tetapi dengan jaringan *wireless*, pengguna dapat bebas bergerak asalkan masih dalam batas area yang memungkinkan untuk mengaksesnya. Jenis jaringan yang memakai jaringan *wireless* ini diantaranya adalah *Wireless Local Area Network* (WLAN).

Penggunaan jaringan *wireless* yang paling populer adalah jaringan WLAN. Jaringan ini banyak digunakan di rumah, *cafe*, restoran, pusat perbelanjaan, perpustakaan, universitas, dan tempat-tempat berkumpul lainnya. Namun, jaringan *wireless* tidak mempunyai batasan yang jelas dan dengan penggunaan gelombang radio ini dapat membuat pihak-pihak tertentu untuk melewati celah keamanan dan memanfaatkannya untuk melakukan hal-hal negatif, seperti pencurian *password, e-mail* penting, dan data-data rahasia lainnya. Oleh karena itu, dibutuhkan suatu sistem deteksi yang disebut *Intrusion Detection System* (IDS) untuk mengantisipasi bahaya tersebut. Dalam jaringan WLAN, IDS ini disebut *Wireless Intrusion Detection System* (WIDS).

WIDS dapat mendeteksi serangan pada *frame* 802.11 pada *layer* dua dari model *Open Systems Interconnection* (OSI). Istilah 802.11 adalah standar dalam jaringan WLAN. Ada tiga tipe *frame* pada jaringan *wireless*, yaitu *data frame*, *control frame*, dan *management frame*. Mayoritas serangan *wireless* menjadikan *management frame* sebagai targetnya karena *frame* ini bertugas untuk melakukan otentikasi, asosiasi, disosiasi, *beacon*, dan *probe request/response*. Serangan *wireless* seperti *Man-in-the-Middle* (MIM), *Rogue Access Point* (RAP), *war*

driver, dan *Denial-of-Service* (DoS) berjalan pada *frame* 802.11. IDS biasa (pada jaringan kabel) tidak dapat menerima *frame* ini.

Implementasi ini dilakukan dengan menggunakan *tools* WIDS yang menggunakan sensor yang ditempatkan pada titik tertentu pada jaringan sehingga lebih terstruktur. *Tools* yang digunakan adalah Kismet. Kismet mempunyai *interface* yang dapat menampilkan jaringan apa saja yang berada dalam jangkauannya, *client* mana saja yang terhubung dengan jaringan tersebut serta *alert* yang dihasilkan oleh *tools* ini sebagai WIDS. Kismet mempunyai program Kismet *drone* yang ditempatkan ke sensor yang berupa *Wireless Router* (WR), Kismet *server* yang mengolah data-data yang dikumpulkan Kismet *drone*, dan Kismet *client* yang menampilkan data-data yang telah diolah oleh Kismet *server*. Kismet *server* dan Kismet *client* umumnya ditempatkan pada PC. Hal ini dikarenakan memori *Wireless Router* yang terbatas.

Kismet masih menyimpan datanya berupa *log file* yang disimpan sepanjang Kismet itu dijalankan. Ketika Kismet dijalankan lagi, Kismet akan menyimpan *log*-nya dalam *file* yang baru dan hanya berbeda penamaan tanggal dan waktunya saja. Hal ini membuat administrator sulit dalam mencari *alert* berdasarkan AP mana yang ingin dilihat ataupun pada saat administrator ingin melihat apa yang terjadi di jaringannya kemarin. Selain itu, dengan Kismet *client* yang menampilkan *alert* berupa *rolling log* di *interface*-nya membuat administrator kesulitan dalam menganalisis jaringannya. Oleh karena itu, dibutuhkan adanya pemakaian *database* dan *interface* yang *user friendly*.

Implementasi ini dilakukan dengan menggunakan beberapa *tools* seperti Hostapd, Aircrack-ng, dan Reaver sebagai penyerang. Hostapd untuk *spoofed* AP, Aircrack-ng untuk serangan *de-authentication flood*, dan Reaver untuk serangan *brute force Wi-Fi Protected Setup* (WPS). Sedangkan untuk mendeteksi apabila terjadi serangan, maka dilakukan implementasi dengan menggunakan Sagan dan Snorby. Sagan merupakan sistem *event log monitoring* yang diproses secara *real-time*. Apabila Sagan mendeteksi adanya '*bad thing*', *event* tersebut dapat disimpan di *database* (MySQL) dan Sagan akan mengkorelasikan *event* tersebut dengan *alert* Kismet. Snorby merupakan *frontend website* untuk Sagan. Selain itu, Kismet

juga menyimpan datanya ke *file* pcap yang dapat dibaca Wireshark. *File* ini berisi *frame* atau paket yang tertangkap oleh sensor.

1.2 Tujuan Penulisan

Tujuan umum dari penulisan skripsi ini adalah untuk mengimplementasikan sebuah aplikasi *Wireless Intrusion Detection System* (WIDS) untuk mendeteksi adanya serangan atau anomali yang terjadi di jaringan *wireless*. Dari tujuan umum tersebut, dapat dirinci kembali menjadi beberapa tujuan khusus sebagai berikut:

1. Mengimplementasikan *Wireless Intrusion Detection System* (WIDS) pada jaringan *Wireless Local Area Network* (WLAN) yang dapat mendeteksi adanya serangan atau anomali di dalam jaringan.
2. Menghitung jumlah *alert* dan *frame* yang terdeteksi, dan waktu yang dibutuhkan untuk mendeteksi *frame* serangan.
3. Mengukur performansi sistem WIDS terhadap faktor banyaknya serangan dan peletakan sensor.

1.3 Batasan Masalah

Untuk menghindari meluasnya materi pembahasan skripsi ini, maka penulis membuat batasan masalah sebagai berikut:

1. Mengimplementasikan sebuah sistem WIDS dengan menggunakan aplikasi Kismet yang memiliki fungsi untuk melakukan deteksi terhadap serangan dan memberikan *alerting* pada jaringan infrastruktur WLAN.
2. Apabila terjadi serangan maka Kismet *drone* akan mendeteksi serangan tersebut dan Kismet *server* akan memberikan *alerting* yang kemudian akan dibaca oleh Sagan. Kemudian Sagan akan menyimpan ke *database* Snorby agar Snorby dapat menampilkannya dengan *interface website*.
3. *Frame* yang tertangkap sensor (Kismet *drone*) akan dianalisis dengan *file* pcap yang dihasilkan Kismet yang dapat dibaca oleh Wireshark.
4. Mengukur performansi sistem.

1.4 Metode Penelitian

Metode penelitian yang digunakan pada skripsi ini adalah :

1. Studi literatur dan pustaka, yaitu melakukan berbagai diskusi pembahasan dengan dosen pembimbing serta dari pustaka yang mendukung.
2. Pendefinisian masalah dan kebutuhan sistem.
3. Analisis dan perancangan sistem, yang meliputi tahapan terstruktur sebagai berikut :
   a. Perancangan sistem *Wireless Intrusion Detection System* (WIDS).
   b. Implementasi dan uji coba
4. Implementasi perancangan perangkat lunak, sistem yang akan diimplementasikan adalah *Wireless Intrusion Detection System* (WIDS), yaitu sistem yang dapat mendeteksi adanya serangan yang masuk ke dalam jaringan *Wireless Local Area Network* (WLAN).
5. Uji coba dan evaluasi sistem, melakukan uji coba dan evaluasi sistem yang telah diimplementasikan.
6. Mengambil kesimpulan, pengujian WIDS yang dapat disimpulkan dari hasil *log* yang ada.

1.5 Sistematika Penulisan

Sistematika penulisan skripsi ini dibagi menjadi beberapa bab yang meliputi :

BAB 1 Pendahuluan

Bab ini berisi tentang latar belakang, tujuan penulisan, batasan masalah, metologi penelitian yang dipakai dalam penelitian, dan sistematika penulisan yang memuat susunan penulisan skripsi ini.

BAB 2 Pengenalan Jaringan WLAN dan Sistem WIDS

Bab ini membahas definisi-definisi dan konsep-konsep dasar yang digunakan dalam penelitian ini, meliputi jaringan WLAN, serangan-serangan pada jaringan WLAN, dan sistem WIDS yang digunakan untuk mendeteksi adanya serangan atau anomali dalam jaringan *wireless*.

BAB 3 Perancangan Sistem WIDS

Bab ini berisi pembahasan tentang disain dari sistem WIDS, korelasi *tools* yang digunakan dan juga perancangan dari sistem *interface website* Snorby serta gambaran tahapan penginstalannya.

BAB 4 Pengujian dan Analisis

Bab ini berisi tentang analisis dan pengujian dari sistem WIDS tersebut yang hasil *alert*-nya dapat ditampilkan di Snorby serta pengukuran *frame* dan waktu yang dibutuhkan untuk mendeteksi serangan yang dilihat dari Wireshark agar dapat diketahui bagaimana performansi sistem ini jika dikondisikan terhadap faktor banyaknya serangan dan peletakan sensor.

BAB 5 Kesimpulan

Bab ini berisi tentang kesimpulan yang di dapat dari hasil penelitian yang dilakukan.

BAB 2
PENGENALAN JARINGAN WLAN DAN SISTEM WIDS

Pada bab ini akan dijelaskan jaringan *Wireless Local Area Network* (WLAN), komponen dan mode dalam jaringan WLAN, dan teknologi WLAN itu sendiri. Selain itu juga serangan-serangan yang umumnya terjadi sehingga dibutuhkan suatu *Wireless Intrusion Detection System* (WIDS) agar jaringan menjadi lebih aman.

2.1 *Wireless Local Area Network* (WLAN)

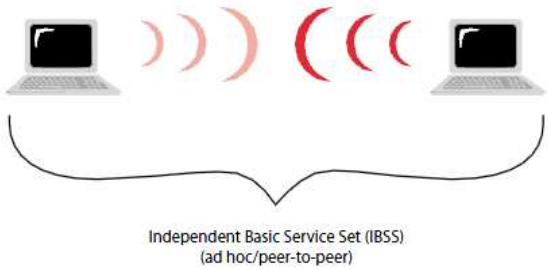
Jaringan internet bisa diakses melalui jaringan kabel maupun nirkabel (*wireless*). Jaringan kabel dapat membatasi pengguna dalam mengakses internet, tetapi dengan jaringan *wireless,* pengguna dapat bebas bergerak asalkan masih dalam batas area yang memungkinkan untuk mengaksesnya. Jenis jaringan yang memakai jaringan *wireless* ini diantaranya adalah *Wireless Local Area Network* (WLAN).

2.1.1 Mode Jaringan WLAN

Spesifikasi 802.11 mendefinisikan dua tipe mode jaringan WLAN, yaitu mode *ad-hoc* (*peer-to-peer*) dan mode infrastruktur.

1. Mode *ad-hoc*

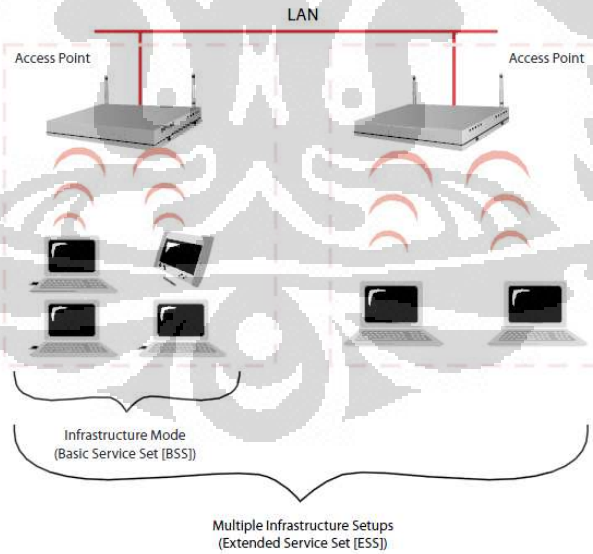
   Jaringan *ad-hoc* juga kadang dinamakan *Independent Basic Service Set* (IBSS). Pada jaringan *ad-hoc*, jaringan *wireless* sangat sederhana. Jaringan ini menghubungkan beberapa komputer yang menggunakan NIC radio ke dalam sebuah jaringan tanpa menggunakan peralatan tambahan seperti *Access Point* (AP) melalui gelombang radio. [1]



Gambar 2.1. Mode *ad-hoc* [1]

2. Mode infrastruktur

Pada mode infrastruktur, semua perangkat *client* berkomunikasi dengan AP yang menghubungkan *client wireless* dengan jaringan kabel yang telah ada. AP mengubah paket 802.11 ke paket *ethernet* LAN 802.3. Semua *client wireless* dan perangkat dalam jangkauan dapat menerima paket, tetapi hanya *client* yang cocok dengan alamat tujuan yang akan menerima dan memproses paket. Infrastruktur *wireless* sederhana dengan satu AP disebut *Basic Service Set* (BSS). Jaringan dimana lebih dari satu AP yang membentuk satu sub-jaringan disebut *Extended Service Set* (ESS). [1]



Gambar 2.2. *Basic Service Set* (BSS) dan *Extended Service Set* (ESS) [1]

Di dalam jaringan *wireless* terdapat *Service Set Identifier* (SSID). SSID adalah 32 karakter pengenal unik yang terdapat pada *header* atau paket yang dikirim pada jaringan yang bertindak sebagai *password* ketika perangkat

mobile mencoba untuk membuat koneksi dengan BSS. SSID membedakan WLAN yang satu dengan yang lain, jadi semua AP dan perangkat yang mencoba untuk membuat koneksi ke WLAN tertentu harus menggunakan SSID yang sama. SSID juga dianggap sebagai nama jaringan karena SSID merupakan nama yang dapat mengidentifikasi suatu jaringan *wireless*.

Pada jaringan ESS, jaringan-jaringan BSS tidak harus menggunakan SSID yang sama namun tanpa SSID yang sama, pengguna tidak akan dapat menggunakan fungsi *roaming*. *Roaming* adalah fitur yang memungkinkan *client* berpindah dari sebuah jaringan BSS ke jaringan BSS yang lain secara otomatis tanpa terputus koneksinya. Untuk menggunakan *roaming*, harus terdapat *overlapping area* atau area dimana sinyal dari kedua BSS bisa diakses. [2]

2.1.2 *Channel Wireless*

Sebelum membahas lebih jauh tentang WLAN, ada komponen utama yang menyebabkan WLAN dapat bekerja, yaitu *channel*. Frekuensi yang digunakan untuk radio adalah FM dan AM. Di dalam frekuensi FM, terdapat ratusan *channel* tempat dimana masing-masing stasiun radio menggunakannya. Jaringan *wireless* menggunakan konsep yang sama dengan stasiun radio, dimana saat ini terdapat dua alokasi frekuensi yang digunakan, yaitu 2,4 GHz dan 5 GHz. Frekuensi 2,4 GHz digunakan oleh 802.11b/g/n. Frekuensi ini dibagi menjadi *channel-channel* seperti pembagian frekuensi untuk stasiun radio. *International Telecommunication Union* (ITU) membagi frekuensi ini menjadi 14 *channel* namun setiap negara mempunyai kebijakan tertentu terhadap pembagian *channel* ini. Amerika hanya mengijinkan penggunaan *channel* 1 – 11, Eropa hanya menggunakan *channel* 1 – 13, dan Jepang diperbolehkan menggunakan semua *channel*, yaitu 1 – 14. [2]

Tabel 2.1. Alokasi pembagian *channel* pada frekuensi 2,4 GHz [2]

Channel	**Frekuensi (GHz)**	***Range***	***Channel Range***
1	2,412	2,401 – 1,423	1 – 3
2	2,417	2,406 – 2,428	1 – 4
3	2,422	2,411 – 1,433	1 – 5
4	2,427	2,416 – 2,438	2 – 6
5	2,432	2,421 – 1,443	3 – 7
6	2,437	2,426 – 2,448	4 – 8
7	2,442	2,431 – 1,453	5 – 9
8	2,447	2,436 – 2,458	6 – 10
9	2,452	2,441 – 1,463	7 – 11
10	2,457	2,446 – 2,468	8 – 11
11	2,462	2,451 – 1,473	9 – 11
12	2,467	2,456 – 2,478	tidak ada di US
13	2,472	2,461 – 1,483	tidak ada di US
14	2,484	2,473 – 2,495	tidak ada di US

2.1.3 *Management Frame*

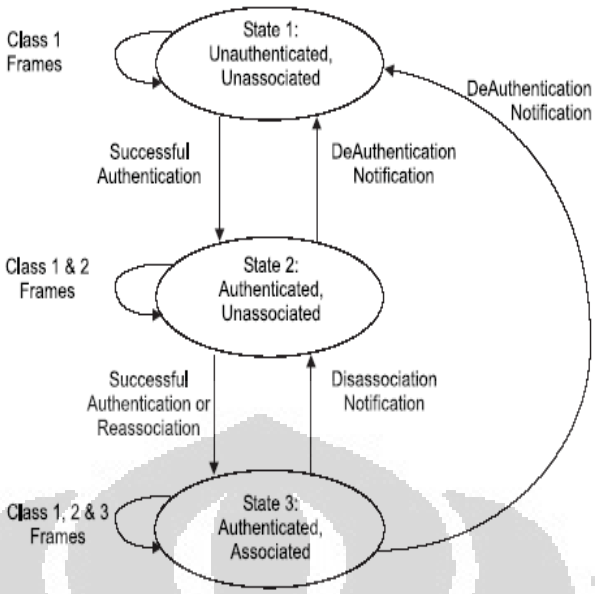
Standar 802.11 mendefinisikan beberapa tipe *frame* yang digunakan oleh *station* (NIC dan AP) untuk berkomunikasi, mengatur, dan mengontrol hubungan di jaringan *wireless*. *Management frame* membuat *station* dapat membangun dan mempertahankan komunikasi. Berikut merupakan subtipe dari *management frame* 802.11 [3]:

1. Otentikasi: *frame* ini digunakan oleh *client* untuk membuat AP mengindentifikasi mereka sebagai *client* yang sah. *Client* mengirim *authentication request* dan AP menjawabnya dengan *authentication response*, yang bisa berarti AP menolak atau menerima *client*.
2. De-otentikasi: sebuah *station* mengirim paket de-otentikasi ke *station* lain jika *station* tersebut ingin mengakhiri komunikasi.
3. *Association request*: digunakan pada saat terjadinya proses asosiasi antara NIC radio dengan AP. NIC radio akan memulai proses asosiasi dengan mengirimkan *frame association request* ke AP.
4. *Association response*: *frame* ini akan dikirim oleh AP ke NIC radio yang berisi persetujuan atau penolakan terhadap NIC radio yang meminta asosiasi.
5. *Reassociation request*: digunakan oleh NIC radio saat ingin melakukan asosiasi dengan AP yang baru ketika NIC berpindah tempat dari satu AP ke AP yang lain.

6. *Reassociation response*: dikirim oleh AP yang berisi pemberitahuan persetujuan atau penolakan terhadap NIC radio yang meminta proses asosiasi kembali.
7. Disosiasi: digunakan pada saat sebuah *station* ingin menghentikan proses asosiasi. Setelah menerima *frame* ini, AP akan melepaskan alokasi memori dan menghapus NIC radio di tabel asosiasi.
8. *Beacon*: merupakan *frame* yang akan dikirim oleh AP untuk memberitahukan keberadaannya serta informasi *relay* kepada setiap NIC radio *station*. Melalui *frame* ini, NIC kemudian akan menentukan AP yang terbaik untuk melakukan asosiasi.
9. *Probe request*: digunakan oleh *station* untuk mengetahui atau memperoleh informasi dari *station* yang lain. Dengan *frame* ini, NIC radio akan melakukan pemeriksaan dan menentukan AP mana yang ada pada jarak *range* penerimaannya.
10. *Probe response*: *station* akan merespon dengan menggunakan *frame* ini dengan menyertakan kapabilitas informasi dan dukungan *data rate* setelah menerima *probe frame request*.

2.1.4 Proses Otentikasi dan Asosiasi

Dalam melakukan koneksi ke AP, *client* melakukan beberapa tahapan, yaitu otentikasi dan asosiasi. Skema tahapannya dapat dilihat pada Gambar 2.3. berikut. [4]



Gambar 2.3. Tahapan otentikasi dan asosiasi [4]

1. Otentikasi

   Otentikasi adalah proses untuk membuktikan identitas *station* ke *station* lain ataupun ke AP. Dalam sistem otentikasi terbuka, semua *station* akan diotentikasi tanpa pengecekan apapun. Sebuah *station* A mengirim sebuah *management frame* otentikasi yang berisi identitas A ke *station* B. *Station* B menjawab dengan sebuah *frame* yang berisi pengakuan ke *station* A. Dalam arsitektur jaringan tertutup, *station* harus mengetahui SSID dari AP agar dapat membuat koneksi ke AP. Selain itu, *shared key authentication* juga digunakan sebagai *password*.

2. Asosiasi

   Pertukaran data antara AP dan *station* hanya dapat terjadi ketika *station* sudah berasosiasi dengan AP dalam mode infrastruktur ataupun dengan *station* lain dalam mode *ad-hoc*. Semua AP mentransmisikan *beacon frame* beberapa kali tiap detiknya yang berisi SSID, waktu, kapabilitas, kecepatan, dan informasi lainnya. Asosiasi mempunyai dua proses. Sebuah *station* yang belum diotentikasi dan belum diasosiasi mendengarkan *beacon frame*. *Station* memilih BSS. *Station* dan AP bersama-sama mengotentikasi dengan bertukar *frame* otentikasi. *Client* sekarang sudah diotentikasi tetapi belum diasosiasi. Pada tahap kedua, *station* mengirim *frame association request* dan AP akan

merespon dengan *frame association response* yang terdiri dari ID asosiasi ke *station. Station* tersebut sekarang sudah diotentikasi dan diasosiasi.

2.1.5 *Wi-Fi Protected Setup*

Wi-Fi Protected Setup (WPS) adalah program sertifikasi tambahan dari Wi-Fi Alliance yang didisain untuk memudahkan mengatur dan mengkonfigurasi keamanan dalam jaringan WLAN. Diluncurkan oleh Wi-Fi Alliance pada awal tahun 2007. Program ini menyediakan solusi untuk lingkungan rumah dan kantor kecil terutama bagi mereka yang tidak begitu paham mengenai keamanan jaringan. [5]

Ada dua metode dalam otentikasi WPS, yaitu:

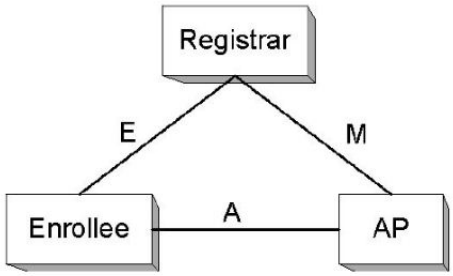
1. *Personal Information Number* (PIN)

   Metode PIN membuat *user* harus menggunakan PIN (dari label stiker atau *utility screen* atau *website-based control panel*) dan memasukkannya ke AP atau perangkat *client* WPS untuk membuat koneksi. Metode ini merupakan metode utama untuk semua perangkat yang bersetifikasi WPS .
2. *Push Button Configuration* (PCB)

   Metode *Push Button Configuration* (PCB) membuat *user* menekan tombol, bisa secara virtual ataupun yang sebenarnya pada kedua perangkat WPS untuk membuat koneksi.

Pada kedua metode, *user* WPS tidak diminta untuk memasukkan SSID dan keamanan *wireless*. Keseluruhan operasi ini dilakukan tidak lebih dari 2 menit. Ada tiga komponen utama secara logika dalam WPS, yaitu [5]:

1. *Enrollee*: sebuah perangkat yang tidak mempunyai pengaturan untuk jaringan *wireless.*
2. *Registrar*: menyediakan pengaturan *wireless* untuk *enrollee*.
3. *Access Point*: menyediakan jaringan *wireless* dan juga pesan *proxy* antara *enrollee* dan *registrar*.



Gambar 2.4. Skema WPS [6]

Interface E secara logika berada di antara *enrollee* dan *registrar* (secara fisik, AP dapat bekerja sebagai *proxy* untuk menyampaikan pesan). *Interface* E bertujuan untuk mengaktifkan *registrar* agar mengetahui dan mengurus otentikasi WLAN untuk *enrollee*. *Interface* M berada di antara AP dan *registrar*. *Interface* ini mengaktifkan *registrar* eksternal agar mengatur AP WPS. WPS menggunakan protokol yang sama untuk mengatur *interface* manajemen AP dan juga untuk melakukan otentikasi ke perangkat *enrollee*. *Interface* A berada di antara *enrollee* dan AP. *Interface* ini berfungsi untuk mengaktifkan komunikasi antara *enrollee* dan IP (hanya *registrar*). [6]

Protokol registrasi WPS secara umum adalah rangkaian pertukaran pesan *Extensible Authentication Protocol* (EAP) seperti berikut [6]:

- *Enrollee -> Registrar*: M1 = Versi || N1 || Deskripsi || PKE
- *Enrollee* <- *Registrar*: M2 = Versi || N1 || N2 || Deskripsi || PKR [|| ConfigData] || HMAC_AuthKey(M1 || M2*)
- *Enrollee -> Registrar*: M3 = Versi || N2 || E-Hash1 || E-Hash2 || HMAC_AuthKey(M2 || M3*)
- *Enrollee* <- *Registrar*: M4 = Versi || N1 || R-Hash1 || R-Hash2 || ENC_KeyWrapKey(R-S1) || HMAC_AuthKey (M3 || M4*)
- *Enrollee -> Registrar*: M5 = Versi || N2 || ENC_KeyWrapKey(E-S1) || HMAC_AuthKey (M4 || M5*)
- *Enrollee* <- *Registrar*: M6 = Versi || N1 || ENC_KeyWrapKey(R-S2) || HMAC_AuthKey (M5 || M6*)
- *Enrollee -> Registrar*: M7 = Versi || N2 || ENC_KeyWrapKey(E-S2 [||ConfigData]) || HMAC_AuthKey (M6 || M7*)

- *Enrollee* <- *Registrar*: M8 = Versi || N1 || [ENC_KeyWrapKey(ConfigData)] || HMAC_AuthKey (M7 || M8*)

Berikut penjelasan untuk skema alur protokol registrasi di atas [6]:

- || adalah simbol rangkaian parameter untuk membuat sebuah pesan.
- Mn* adalah pesan Mn tidak termasuk HMAC-SHA-256.
- Versi mengindentifikasikan tipe pesan protokol registrasi.
- N1 adalah nomor acak 128 bit yang dispesifikasikan oleh *enrollee.*
- N2 adalah nomor acak 128 bit yang dispesifikasikan oleh *registrar*.
- Deskripsi berisi deskripsi perangkat pengirim pesan (UUID, manufaktur, nomor model, MAC *address*, dan lain-lain) dan kapabilitas perangkat, seperti algoritma, *channel* I/O, peran protokol registrasi, dan sebagainya. Deskripsi juga ada di pesan *probe request* dan *probe response*.
- HMAC_AuthKey(...) mengindikasikan atribut otentikator yang berisi HMAC *keyed hash* yang berada dalam tanda kurung dan menggunakan kunci AuthKey. Fungsi *keyed hash* adalah HMAC-SHA-256 per FIPS 180-2 dan RFC-2104. Untuk mengurangi ukuran pesan, hanya 64 bit dari 256 bit keluaran HMAC yang termasuk dalam atribut otentikator.
- ENC_KeyWrapKey(...) mengindikasikan enkripsi simetris dari nilai dalam tanda kurung, menggunakan kunci KeyWrapKey. Algoritma enkripsinya adalah AES-CBC per FIPS 197, dengan lapisan PKCS#5 v2.0.
- PKE dan PKR adalah *public key* Diffie-Hellman dari *enrollee* dan *registrar*.
- AuthKey adalah kunci otentikasi yang diambil dari rahasia Diffie-Hellman, N1 dan N2, dan MAC *address enrollee*. Jika M1 dan M2 keduanya dikirim di *channel* yang rentan terhadap serangan *Man-in-the-Midddle, password* perangkat *enrollee* mungkin dapat bocor.
- E-Hash1 dan E-Hash2 adalah persetujuan sebelumnya yang dibuat oleh *enrollee* untuk membuktikan bahwa *enrollee* mengetahui dua bagian dari *password* perangkatnya.
- R-Hash1 dan R-Hash2 adalah persetujuan sebelumnya yang dibuat oleh *registrar* untuk membuktikan bahwa *enrollee* mengetahui dua bagian dari *password* perangkatnya.

- R-S1 dan R-S2 adalah rahasia 128 bit N1 dan N2 yang bersama-sama dengan R-Hash1 dan R-Hash2, dapat digunakan oleh *enrollee* untuk mengkonfirmasi apakah *registrar* mengetahui bagian pertama dan kedua dari *password* perangkat *enrollee*.
- E-S1 dan E-S2 adalah rahasia 128 bit N1 dan N2 yang bersama-sama dengan E-Hash1 dan E-Hash2, dapat digunakan oleh *registrar* untuk mengkonfirmasi apakah *enrollee* mengetahui bagian pertama dan kedua dari *password* perangkat *enrollee*.
- ConfigData berisi pengaturan WLAN dan info-info penting untuk *enrollee*. Pengaturan tambahan untuk jaringan lainnya dan aplikasi-aplikasi dapat berada juga di ConfigData. Walaupun ConfigData ditunjukkan selalu dienkripsi, enkripsi hanya utama untuk kunci dan pengikatan kunci. Enkripsi hanya tambahan untuk konfigurasi data lainnya. Hal ini merupakan keputusan pengirim apakah akan mengenkripsi atau tidak.

2.1.6 Serangan-serangan pada Jaringan WLAN

Dibalik kenyamanannya jaringan WLAN, terdapat ancaman yang lebih rawan, khususnya terhadap serangan-serangan. Targetnya bisa data yang tidak penting sampai pada yang sangat penting, khususnya rekening dan data rahasia lainnya. Semakin banyaknya penggunaan jaringan *wireless*, semakin banyak pula vulnerabilitasnya.

1. MAC *address spoofing*

   Istilah ‘MAC *address spoofing’* dalam konteks ini berarti penyerang mengubah MAC *address* keluaran dari pabrik menjadi nilai yang lain. MAC *address spoofing* secara konsep berbeda dengan IP *address spoofing* di mana penyerang mengirim data dari sumber alamat yang sembarang dan tidak mengharapkan adanya respon ke sumber IP *address* yang sebenarnya. MAC *address spoofing* lebih akurat disebut sebagai peniruan atau penyamaran MAC *address* karena penyerang tidak membuat data dengan sumber yang berbeda dengan alamat transmisinya. Ketika penyerang mengganti MAC *address*-nya, penyerang terus menggunakan *wireless card* untuk

mengirimkan *frame*, mengirim dan menerima dari sumber MAC yang sama. [7]

2. *Denial-of-Service* (DoS)

   a. *Jamming*

      Jamming bekerja dengan membuat *noise Radio Frequency* (RF) pada jangkauan frekuensi yang digunakan oleh perangkat jaringan *wireless* (2,4 GHz dan 5 GHz). *Jamming* dapat terjadi secara tidak disengaja karena perangkat-perangkat tertentu dapat beroperasi pada jangkauan frekuensi ini, seperti gelombang mikro, monitor bayi, radar, dan lain-lain. Dengan penggunaan perangkat-perangkat ini setiap harinya, perangkat-perangkat ini dapat mengganggu jaringan *wireless* yang ada di sekitarnya. Selain itu, *jamming* dapat disengaja dengan perangkat yang dapat mengeluarkan *noise* untuk mengganggu jaringan WLAN. [8]

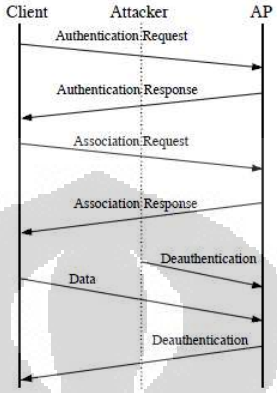
   b. *Authentication*/*association flooding*

      Serangan ini mencoba membanjiri AP dengan *frame* otentikasi dan *frame* asosiasi. Untuk melakukan *association flood*, perangkat penyerang akan meng-*spoof* MAC *address wireless client*, kemudian secara cepat dan berulang berusaha berasosiasi dengan AP. Setiap percobaan asosiasi, penyerang akan mengganti MAC *address*, menyamar menjadi sekian banyaknya *client*. Hal ini menyebabkan memori dan kemampuan pemrosesan AP berkurang, yang membuat AP menolak *client* yang asli. [8]

   c. *De-authentication flooding*

      Serangan ini bekerja dengan mengeksploitasi kelemahan dalam jaringan *wireless* dan biasanya digunakan sebagai metode untuk melancarkan *Rogue Access Point* (RAP). Penyerang mengirim paket-paket de-otentikasi ke *client* yang terkoneksi pada sebuah AP. *Client* yang menerima *frame* de-otentikasi akan melakukan disosiasi dengan AP secepatnya. Serangan ini berhasil karena pengontrolan trafik AP tidak diproteksi sehingga *frame* de-otentikasi dikirim dengan tidak menggunakan enkripsi sehingga sangat mudah untuk di-*spoof*. Serangan ini dapat menjadikan seluruh AP sebagai targetnya, dengan mengirim *frame-frame* de-otentikasi ke semua MAC

address secara *broadcast* dan meng-*spoof* sumbernya sebagai AP. Serangan dapat juga ditargetkan ke satu *client*, dengan mengirim *frame* de-otentikasi ke satu *client*. [8]



Gambar 2.5. Skema serangan *de-authentication flood* [9]

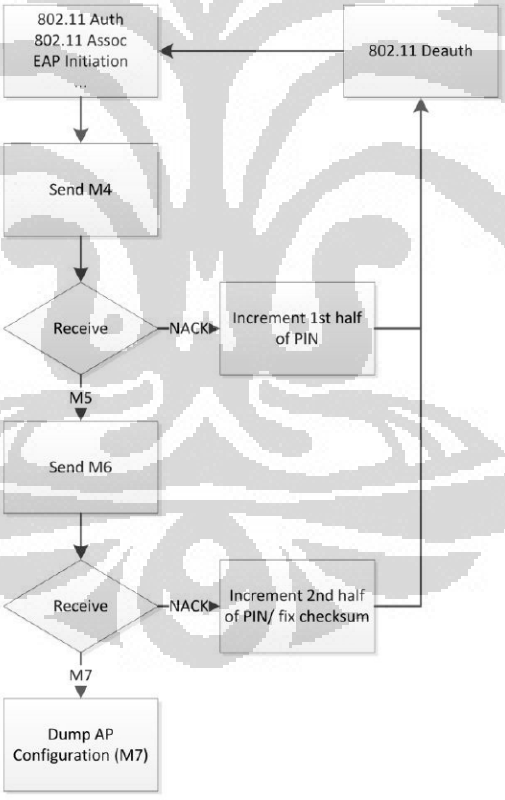
3. *Man-in-the-Middle*

   Penyerang menggunakan serangan *Man-in-the-middle* (MIM) untuk menangkap aktivitas dalam jaringan. Serangan MIM dimulai dengan mengkonfigurasi *Rogue Access Point* (RAP) untuk menyamar menjadi AP asli. Lalu penyerang memaksa *client* untuk membuat koneksi ke RAP dengan melakukan serangan DoS ke AP asli atau dengan menyediakan sinyal yang lebih kuat dibandingkan AP asli. *Client* biasanya akan membuat koneksi dengan AP yang sinyalnya lebih kuat. Agar dapat membohongi *client* dengan pasti, RAP membuat koneksi ke jaringan lainnya. Jika berhasil, penyerang akan dapat mendapat mengambil hak kontrol terhadap koneksi jaringan *wireless client*. Agar serangan sukses dijalankan, AP MIM harus paling tidak berbeda 5 *channel* dari *channel* target AP untuk menghindari serangan DoS. Dengan MIM, penyerang dapat memalsukan halaman *website* otentikasi *wireless* untuk mengumpulkan ID dan *password*; dan mengumpulkan *website*, *username* dan *password* yang digunakan pada *website* tersebut. [10]

4. *Brute force* WPS

   Disain *Wi-Fi Protected Setup* (WPS) mempunyai kelemahan. Penyerang memanfaatkan kelemahan ini untuk mengungkap PIN dalam waktu

beberapa jam saja. Seorang penyerang dapat memperoleh apakah bagian dari PIN itu benar atau tidak dari respon AP. Jika penyerang mendapat sebuah pesan EAP-NACK setelah mengirim M4, hal ini berarti bahwa bagian setengah awal PIN sudah benar. Jika penyerang mendapat pesan EAP-NACK setelah mengirim M6, hal ini berarti bahwa setengah bagian akhir PIN sudah benar. PIN terdiri dari 8 angka, sehingga kemungkinan banyaknya nilai PIN adalah 10^8 atau 100.000.000. Dengan adanya skema serangan ini, percobaan jumlah serangan maksimum akan berkurang menjadi 10^4+10^4, yaitu 20.000 kali. Karena bilangan ke-8 PIN merupakan *checksum* dari bilangan ke-1 sampai ke-7, sehingga percobaan penyerangan akan berkurang menjadi 10^4+10^3, yaitu menjadi 11.000 kali. Gambar 2.6. merupakan skema metode serangan *brute force* WPS. [11]



Gambar 2.6. *Flowchart* serangan *brute force* PIN WPS [11]

2.2 *Wireless Intrusion Detection System*

Penggunaan jaringan *wireless* berkembang dengan sangat cepat. Dengan adanya jaringan *wireless*, *user* dapat lebih mudah dan nyaman mengakses internet.

Penggunaan *client* tidak terbatas dengan adanya kabel seperti layaknya jaringan kabel. Untuk menjaga jaringan *wireless* dari hal-hal yang tidak diinginkan, administrator harus memonitor kejadian-kejadian apa saja yang terjadi di jaringannya dan mendeteksi adanya serangan atau tidak. Untuk melakukannya, administrator harus menginstal sebuah *Wireless Intrusion Detection System* (WIDS).

2.2.1 Cara Kerja WIDS

Secara umum, *Instrusion Detection System* (IDS) didisain dan dibuat untuk memonitor dan melaporkan aktivitas yang berlangsung dalam jaringan kepada administrator. WIDS membutuhkan sensor untuk mengumpulkan data-data dalam jaringan, *server* untuk mengolah data-data yang telah dikumpukan, dan *client* untuk menampilkan hasil dari pengolahan data yang telah dikumpulkan. *Interface* WIDS ditempatkan pada *client* WIDS.

WIDS dapat mendeteksi serangan pada *frame* 802.11 pada lapisan dua dari jaringan *wireless.* Ada tiga tipe *frame* MAC 802.11, yaitu *data frame*, *control frame*, dan *management frame.* Mayoritas serangan *wireless* menjadikan *management frame* sebagai targetnya karena *frame* ini bertugas untuk melakukan otentikasi, asosiasi, disosiasi, *beacon*, dan *probe request/response*. Serangan *wireless* seperti MIM, RAP, *war driver*, dan *de-authentication flood* berjalan pada *frame* 802.11 dan tidak bisa dideteksi pada lapisan tiga. IDS biasa (pada jaringan kabel) tidak dapat menerima *frame* ini, karena *management frame* tidak dapat diteruskan ke lapisan di atasnya.

WIDS membutuhkan *interface* khusus. *Interface wireless* ini harus dioperasikan pada mode monitor, yang dikenal juga sebagai mode RFMON. Mode ini membolehkan perangkat untuk menerima semua lalu lintas yang masuk. *Interface* yang bertugas memonitor harus terus berganti *channel*, dikenal dengan *channel hopping*, yang tersedia pada jaringan tersebut. Beberapa serangan *wireless* bekerja dengan menggunakan RAP pada *channel* yang berbeda. Sebagai contoh, serangan MIM menggunakan RAP yang paling sedikit berbeda lima *channel* dari target AP [12].

Untuk mendeteksi serangan pada *range* tertentu, WIDS mencocokkan serangan dengan metodologi deteksi:

1. *Signature-based.*

   Deteksi *signature-based* menggunakan *signature* statik untuk mencocokkan lalu lintas yang mencurigakan. Tipe pencocokan ini bekerja dengan baik untuk serangan-serangan yang telah dikenal sebelumnya dan cocok dengan pola tertentu. Sebagai contoh, untuk mendeteksi RAP, WIDS menggunakan daftar AP yang sah dan memberikan *alert* jika ada AP yang terdeteksi tidak cocok dengan daftar.

2. *Knowledge-based.*

   Deteksi *knowledge-based* menggunakan acuan dasar historis dan memberikan *alert* ketika lalu lintas jaringan berbeda dari acuan dasar historis. Beberapa serangan *wireless* tidak cocok dengan *signature* tetapi serangan ini dapat menyebabkan anomali lalu lintas jaringan.

3. Analisis protokol *stateful.*

   Analisis protokol *stateful* merupakan proses membandingkan profil yang sudah ditentukan sebelumnya. Analisis protokol *stateful* bergantung pada profil yang dikembangkan vendor secara universal yang menspesifikasikan bagaimana protokol tertentu seharusnya digunakan. Kata "*stateful*" berarti IDS mampu memahami dan mengikuti keadaan protokol *network*, *transport*, dan *application*. Sebagai contoh, ketika *user* memulai sesi *File Transfer Protocol* (FTP), sesi pertama-tama berada pada keadaan yang belum sah. *User* yang tidak sah seharusnya hanya dapat melakukan perintah-perintah tertentu pada keadaan ini, seperti melihat informasi bantuan. Bagian penting memahami keadaan adalah mencocokkan *request* dengan *response*, jadi ketika otentikasi FTP terjadi, IDS dapat menentukan apakah sukses atau tidak dalam menemukan kode status dalam respon yang sesuai. Ketika *user* sudah berhasil diotentikasi, sesi ada pada keadaan sah, dan *user* dapat melakukan beberapa perintah lainnya. Melakukan hampir semua perintah ketika masih dalam keadaan tidak sah dapat dipertimbangkan sebagai keadaan yang mencurigakan. [13]

2.2.2 Kismet WIDS

Kismet merupakan sebuah aplikasi analisis jaringan *open source*. Kismet mengidentifikasi jaringan dengan cara mengumpulkan paket dan mendeteksi jaringan secara pasif. Kismet dapat mendeteksi nama SSID yang tersembunyi dan keberadaan jaringan *non-beaconing* melalui lalu lintas data. Kismet dapat dijalankan dengan *wireless card* apapun yang mendukung mode *raw monitoring*, dan dapat men-*sniff* lalu lintas 802.11a, 802.11b, 802.11g, dan 802.11n. Kismet juga mendukung arsitektur *plugin* yang memperbolehkan protokol selain 802.11 untuk di-*decode*. Kismet dapat diintegrasikan dengan alat GPS untuk menggambarkan koordinat jaringan yang terdeteksi. [14]

Konfigurasi Kismet dapat diubah untuk mengontrol *channel* dan pola loncatan untuk menangkap *source* di Kismet. Kismet *plugin* dapat melakukan hampir semua hal yang dapat dilakukan oleh Kismet asli. Hal ini termasuk menambah kapabilitas *logging*, menambah *alert* IDS, mendefinisikan *capture source* yang baru, dan menambah fitur baru ke Kismet *User Interface* (UI). Kismet dapat diintegrasikan dengan perangkat *Global Positioning System* (GPS) untuk memberikan koordinat jaringan yang telah terdeteksi. Kismet membolehkan 1 *alert* per detik dan maksimum 5 *alert* per menit. Kismet dapat membaca dengan metode *fingerprint* (serangan paket tunggal spesifik) dan *trend* (*probe* yang tidak biasa, *disassociation floods*, dll). Kismet dapat diintegrasikan dengan aplikasi lain menggunakan tun/tap *export* untuk menyediakan *interface* virtual jaringan dari lalu lintas *wireless*.

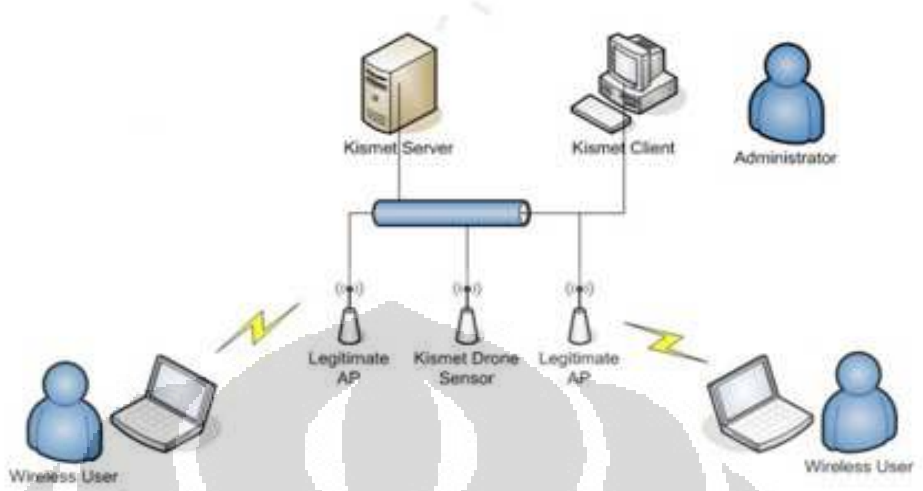
Kismet juga tersedia untuk versi Windows, yaitu KisWin [15], tetapi fitur-fitur yang ditawarkan untuk Linux lebih banyak dan lebih *update* daripada Windows.

2.2.2.1 Arsitektur Kismet WIDS

Kismet *wireless* telah dilengkapi dengan Kismet *drone* yang membuatnya menjadi aplikasi WIDS terdistribusi. Arsitektur Kismet terdiri dari [10]:

1. Kismet *drone*: mengumpulkan data-data dalam jaringan dan mengirimkannya ke Kismet *server*.
2. Kismet *server*: mengolah data-data yang telah dikumpukan.

3. Kismet *client*: menampilkan hasil dari pengolahan data yang telah dikumpulkan.



Gambar 2.7. Arsitektur Kismet WIDS

Perangkat-perangkat yang dibutuhkan agar dapat menjalankan Kismet WIDS ini yaitu [10]:

1. Komputer (Kismet *server* dan Kismet *client* dapat menggunakan 1 komputer).
2. *Wireless adapter* (jika ingin menggunakan *drone*, *wireless adapter* berada pada *Wireless Router*, jika tidak ingin menggunakan *drone*, *wireless adapter* bisa dari *build-in* PC atau USB *wireless adapter*).
3. *Wireless Router* (dikonfigurasi program Kismet *drone*).

WIDS ini dapat menggunakan lebih dari satu sensor sehingga penggunaannya dapat dikembangkan sendiri tergantung kebutuhan administrator.

2.2.2.2 *Alert* Kismet WIDS

Kismet secara *default* membolehkan 1 *alert* per detik dan maksimum 5 *alert* per menit. Kismet dapat membaca dengan metode *fingerprint* (serangan paket tunggal spesifik) dan *trend* (seperti *probe* yang tidak biasa, *disassociation floods*, dll). Kismet mendukung *alert-alert* di bawah ini [14]:

1. ADVCRYPTO (*trend/stateful*)

   Sebuah AP 802.11 mengubah enkripsinya. Hal ini bisa berarti perubahan konfigurasi biasa atau dapat mengindikasikan *spoofed* AP yang tidak benar-benar sama dengan AP yang sah.

2. AIRJACKSSID (*fingerprint*)

   Airjack adalah aplikasi *hacking* 802.11. Airjack membuat SSID-nya menjadi 'airjack' ketika mulai berjalan. Airjack telah lama tidak dilanjutkan pengembangannya, sehingga bisa dikatakan *alert* ini sudah tidak relevan lagi.

3. APSPOOF (*fingerprint*)

   Jika respon dari *beacon* atau *probe* untuk SSID yang dilihat dari MAC *address*-nya tidak ada di daftar yang diperbolehkan mengakses AP, *alert* ini akan diaktifkan. *Alert* ini dapat digunakan untuk mendeteksi AP yang konflik, *spoofed* AP, atau serangan seperti Karma/Airbase dimana serangan ini merespon ke semua *probe request*. Dengan Karma/Airbase, *client* dapat menjadi target dengan membuat RAP. Karma sekarang dikembangkan dengan nama Karmetasploit.

4. BEACONRATE (*trend/stateful*)

   Sebuah AP 802.11 mengubah kecepatan keluaran *beacon*-nya. Hal ini dapat berupa konfigurasi biasa atau mungkin mengindikasikan *spoofed* AP yang tidak benar-benar sama dengan AP yang sah.

5. BSSTIMESTAMP (*trend/stateful*)

   Invalid/out-of-sequence timestamp BSS dapat mengindikasikan AP *spoofing*. AP dengan *timestamp* BSS yang berfluktuasi dapat mengindikasikan serangan "*evil twin*" *spoofing*, karena beberapa aplikasi tidak dapat mengsinkronkan *timestamp* BSS sama sekali, sehingga *timestamp* BSS ini sulit untuk disamakan. *Timestamp* bertugas mengsinkronkan antar *station* dalam BSS.

   Evil twin adalah *wireless* AP palsu yang menyamar menjadi AP asli dengan menyebarkan nama WLAN-nya, yaitu ESSID dan SSID. *Evil twin* dapat menggunakan Karma, sebuah aplikasi untuk memonitor *probe station*, mengamati SSID yang biasa digunakan dan mengadopsi salah satunya. Penyerang juga dapat menggunakan SSID yang umum digunakan seperti

SSID *default* dari vendor. Bahkan AP yang tidak mengirim SSID di *beacon*-nya dapat menjadi target, selama *user* asli dapat dimonitor dengan Wireshark, Kismet, atau WLAN *analyzer* yang lain. [16]

6. CHANCHANGE (*trend/stateful*)

   AP yang sudah terdeteksi tiba-tiba berganti *channel*. Hal ini dapat mengindikasikan serangan *spoofing*. Dengan *spoofing* AP asli di *channel* yang berbeda, penyerang dapat menarik *client* ke *spoofed* AP. Sebuah AP yang berganti *channel* selama operasi normal dapat mengindikasikan serangan ini sedang dalam proses.

7. CRYPTODROP (*trend/stateful*)

   Spoofing sebuah AP dengan enkripsi yang kurang aman dapat membodohi *client* agar membuat koneksi. Satu-satunya situasi dimana sebuah AP harus mengurangi keamanan enkripsinya adalah pada saat AP dikonfigurasi ulang.

8. DEAUTHFLOOD dan BCASTDISCON (*trend/stateful*)

   Dengan *spoofing* paket *disassociation* dan *de-authentication*, seorang penyerang bisa memutuskan *client* dari jaringan, menyebabkan DoS dimana hanya berlangsung selama penyerang bisa mengirim paket.

9. DHCPCLIENTID (*fingerprint*)

   Client yang mengirim paket DHCP DISCOVER berisi *tag* Client-ID (*Tag* 61), dimana tidak cocok dengan sumber MAC dari paket, dapat melakukan DHCP DoS untuk menghabiskan *resource* DHCP *pool*.

10. DHCPCONFLICT (*trend/stateful*)

    Client yang sudah menerima DHCP *address* dan melanjutkan untuk menggunakan IP *address* yang berbeda dapat mengindikasikan kesalahan konfigurasi atau *spoofed client*.

11. DISASSOCTRAFFIC (*trend/stateful*)

    Client yang putus dari jaringan seharusnya tidak secepatnya melanjutkan pertukaran data. Hal ini dapat mengindikasikan *spoofed client* mencoba untuk memasukkan data ke jaringan dengan cara yang tidak benar, atau dapat mengindikasikan *client* sedang menjadi korban dari serangan DoS.

12. DISCONCODEINVALID dan DEAUTHCODEINVALID (*fingerprint*)
    Spesifikasi 802.11 mendefinisikan *valid reason codes* untuk kejadian *disconnect* dan *deauthenticate*. Berbagai *driver client* dan AP telah dilaporkan salah dalam memperlakukan *invalid/undefined reason code*.
13. DHCPNAMECHANGE dan DHCPOSCHANGE (*trend/stateful*)
    Protokol konfigurasi DHCP membolehkan *client* secara optimal memasukkan sistem operasi/vendor *hostname* dan DHCP *client* dalam paket DHCP Discover. Nilai ini hanya berubah jika *client* tiba-tiba baru saja mengganti sistem (sistem *dual-boot*). Mengganti nilai ini sering mengindikasikan serangan *spoofing*/MAC *cloning*.
14. DOT11D (*trend/stateful*)
    Sebuah AP 802.11 mengubah pengaturan 802.11d. Hal ini dapat berarti perubahan konfigurasi normal atau dapat mengindikasikan *spoofed* AP yang tidak benar-benar sama dengan AP yang sah. Perubahan 802.11d juga dapat mengindikasikan serangan aktif untuk membingungkan *client*.
15. LONGSSID (*fingerprint*)
    Spesifikasi 802.11 membolehkan maksimum 32 *byte* untuk SSID. Kelebihan nilai SSID mengindikasikan serangan dimana penyerang berusaha untuk mengeksploitasi vulnerabilitas dari beberapa *driver*.
16. LUCENTTEST (*fingerprint*)
    Lucent Orinoco *card* lama yang dalam mode *scanning test* tertentu menghasilkan paket yang bisa diidentifikasi.
17. MALFORMMGMT (*fingerprint*)
    Management frame yang strukturnya salah dapat mengindikasikan eror di *capture source* (seperti tidak menghapus paket yang rusak), tetapi dapat juga mengindikasikan serangan.
18. MSFBCOMSSID (*fingerprint*)
    Beberapa versi *driver wireless* Broadcom Windows tidak dapat memperlakukan SSID *field* secara benar lebih lama daripada spesifikasi 802.11, menyebabkan *system compromise* dan *code execution*. *System compromise* adalah penggunaan sistem secara ilegal oleh yang bukan pemilik sistem tersebut. *Code execution* adalah mengeksekusi perintah apapun yang

diinginkan penyerang pada targetnya. Vulnerabilitas ini dieksploitasi oleh Metasploit Framework.

19. MSFDLINKRATE (*fingerprint*)

    Beberapa versi *driver wireless* D-Link Windows tidak dapat secara benar mengurus 802.11 *valid rate field* yang sangat panjang, menyebabkan *system compromise* dan *code execution*. Vulnerabilitas ini dieksploitasi oleh Metasploit Framework.

20. MSFNETGEARBEACON (*fingerprint*)

    Beberapa versi *driver wireless* netgear Windows tidak dapat secara benar mengurus kelebihan *frame* dari *beacon*, menyebabkan *system compromise* dan *code execution*. Vulnerabilitas ini dieksploitasi oleh Metasploit Framework.

21. NETSTUMBLER (*fingerprint*)

    Versi lama dari Netstumbler (3.22, 3.23, 3.30) menghasilkan, dalam kondisi tertentu, paket khusus.

22. NULLPROBERESP (*fingerprint*)

    Paket *probe response* dengan komponen SSID IE *tag* dengan panjang 0 dapat menyebabkan *card* lama (Prism2, Orinoco, Airport-classic) gagal.

23. WPSBRUTE (*trend/stateful*)

    WPS rentan terhadap serangan *brute force* yang dapat mengungkapkan kunci WPS dalam waktu beberapa jam. Sebuah AP yang diserang akan mengirim jumlah respon WPS yang tidak normal. Vulnerabilitas ini didokumentasikan dalam *paper* oleh Stefan Viehbock dan mengimplementasikannya. Salah satu *tools* untuk melakukan serangan ini yang paling populer adalah Reaver.

Kismet WIDS dapat digunakan untuk mendeteksi *spoofed* AP, serangan *de-authentication flood*, *spoofed client*, dan *active scanning*.

2.2.2.3 *Supported Driver*

Kismet mendukung beberapa *driver* berikut [14]:

1. Mac80211 General (Linux)
2. Madwifi (Linux)

3. RT28xx (Linux)
4. Rt73-k2wrlz (Linux)
5. WL (Linux, Intel)
6. OTUS (Linux)
7. Nokia ITT (Linux)
8. Orinoco (Linux)
9. NDISWrapper (Linux)
10. BSD (BSD Generic)
11. Windows (Generic)
12. Airpcap (Windows)
13. Perangkat USB (OSX)

2.2.2.4 Kelemahan Kismet WIDS

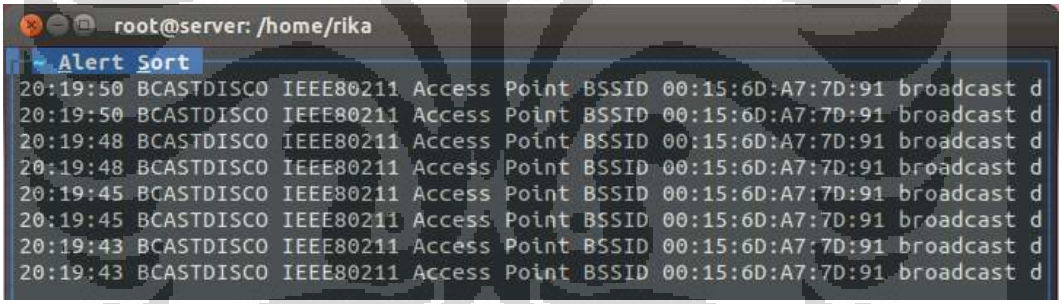
Kismet WIDS dapat digunakan untuk serangan *de-authentication flood*, *spoofing* AP, dan *active scanning*. WIDS ini dapat menggunakan lebih dari satu sensor sehingga penggunaannya dapat dikembangkan sendiri tergantung kebutuhan administrator. Kismet WIDS mempunyai kekurangan yang sangat menyulitkan administrator, yaitu dalam hal pembacaan *alert.* Kismet tidak menulis *alert* IDS-nya ke *log file*, tetapi menampilkannya melalui *rolling log* dalam Kismet *client*. Hal ini membuat administrator harus terus berada di depan Kismet *client* untuk membaca arus *alert* yang masuk. Oleh karena itu, dibutuhkan perangkat lunak lain yang dapat menampilkan *alert-alert* dari Kismet WIDS ini. [17]

BAB 3
PERANCANGAN SISTEM WIDS

Wireless Intrusion Detection System (WIDS) merupakan suatu sistem yang mampu mendeteksi, menganalisis data, mencatat dan memberikan *alerting* kepada administrator secara *real-time*. Suatu WIDS berfungsi untuk memonitor jaringan WLAN dalam suatu ruang lingkup dan memberitahukan apabila ada anomali-anomali yang terjadi dalam jaringannya.

3.1 Perancangan Sistem

Kismet merupakan WIDS *open source* yang terdiri dari beberapa komponen yang saling bekerja sama. Namun, tampilan *alert* dalam *Graphical User Interface* (GUI) untuk Kismet sendiri masih kurang interaktif dan masih belum mempunyai *database* sendiri.



Gambar 3.1. Tampilan *alert* Kismet

Perancangan sistem yang akan digunakan untuk membangun sistem WIDS ini bertujuan untuk mengintegrasikan komponen-komponen yang dibutuhkan dan membuatnya bekerja sama agar dapat membuat WIDS ini menjadi terintegrasi dengan *database* dan menampilkannya dengan tampilan *website* yang *user friendly*.

Sebelumnya WIDS ini sudah dapat mendeteksi anomali-anomali dan serangan dalam jaringan hanya menggunakan *tools* Kismet saja, tetapi untuk penyimpanan, pencarian, dan tampilan datanya masih mempunyai kelemahan. Penyimpanan data masih terbatas *log* pada saat *tools* Kismet dinyalakan saja dan tidak terorganisir dengan baik sehingga jika ingin mencari data-data yang telah

diambil sebelumnya, administrator harus mencarinya secara manual. Hal ini sangat merepotkan dan memakan waktu karena administrator harus mencari data satu-persatu di antara sekian banyaknya data. Selain itu, tampilan Kismet-nya sendiri menggunakan GUI terminal (*console*) sehingga kurang sesuai jika digunakan sebagai WIDS. Oleh karena itu, untuk mendapatkan WIDS yang sesuai dengan kebutuhan administrator dibutuhkan komponen-komponen sebagai berikut:

1. Kismet *drone*

   Kismet *drone* merupakan program Kismet yang berfungsi sebagai sensor. Program ini didisain untuk mengubah Kismet menjadi WIDS terdistribusi. *Drone* menangkap data dan meneruskannya ke Kismet *server* melalui koneksi kabel. *Drone* tidak melakukan *decoding* paket apapun sehingga kebutuhan perangkat kerasnya pun minimal. [14]

   Kismet *drone* dijalankan dengan sistem operasi OpenWRT. OpenWRT adalah sistem operasi *open source* berbasis Linux yang diperuntukkan untuk perangkat *Wireless Router*. Kismet *drone* dapat diunduh dari *repository* OpenWRT.

2. Kismet *server*

   Kismet *server* merupakan program Kismet yang berfungsi untuk mengolah paket-paket data yang telah ditangkap Kismet *drone*. [14]

   Kismet *server* merupakan komponen utama dalam sistem WIDS ini. Kismet dapat diunduh dari https://www.kismetwireless.net/code/svn/trunk/.

3. Sagan

   Sagan adalah sistem *event log monitoring* yang *multi-threaded* yang diproses secara *real-time.* Apabila Sagan mendeteksi adanya '*bad thing*', *event* tersebut dapat disimpan di *database* (Mysql) dan Sagan akan mengkorelasikan *event* tersebut dengan Kismet. Sagan ditulis dengan bahasa C. Sagan dapat mencegah *blocking Input/Output* (I/O). Sebagai contoh, pemrosesan data tidak berhenti ketika sebuah SQL *query* dibutuhkan. [18]

   Sagan dapat diunduh dari https://github.com/beave/sagan.

4. MySQL

   Database MySQL (*Management System Structured Query Language*) merupakan sistem manajemen *database* yang paling populer, yang dikembangkan, didistribusikan, dan didukung oleh MySQL AB. MySQL AB adalah sebuah perusahaan komersial yang didirikan oleh para pengembang MySQL. [19]

   MySQL dapat diunduh dari *repository* Ubuntu.

5. Apache *Web Server*

   Apache adalah *web server* yang mengikuti standar HTTP (*Hypertext Transfer Protocol*).

   Apache dapat diunduh dari *repository* Ubuntu.

6. Snorby

   Snorby adalah aplikasi *frontend* berbasis *Ruby on Rails* yang digunakan untuk memonitor keamanan jaringan. *Ruby on Rails* adalah sebuah kerangka Ruby berbasis *open source* untuk pengembangan aplikasi *website*. Ruby adalah bahasa pemrograman yang dibuat oleh Yukihiro Matsumoto, yang merupakan campuran dari bahasa pemrograman Perl, Smalltalk, Eiffel, Ada, dan Lisp. [20]

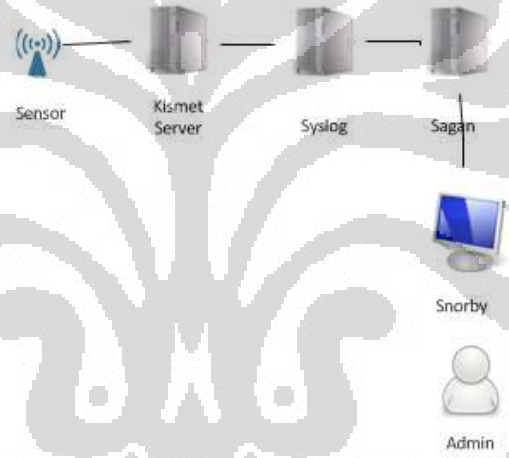
   Snorby merupakan aplikasi *open source* yang dapat menampilkan *log* dari aplikasi Snort, Sagan, dan Suricata. Snort dan Suricata termasuk aplikasi IDS (*Intrusion Detection System*). [21]

   Snorby dapat diunduh di http://github.com/Snorby/snorby.git

Sekarang, Kismet sudah dapat mengalirkan keluaran dari Kismet *server* ke syslog dengan adanya *plugin* syslog yang merupakan bagian dari paket Kismet. Keluaran ini tentunya dapat ditampilkan pada aplikasi *log analyzer* atau *log monitoring*, seperti LogAnalyzer, EventLog Analyzer, Splunk, dan *tools* lainnya. Tetapi dengan aplikasi-aplikasi ini, *log* dari Kismet hanya dapat ditampilkan pada satu grafik saja, yaitu untuk program Kismet. Selain itu juga bercampur dengan info-info dari syslog lainnya, sehingga tidak dapat diatur sesuai keinginan administrator. Dengan Sagan, administrator dapat mengkonfigurasi *rules* Kismet sendiri agar dapat sesuai dengan keinginannya dan dapat memilih hanya syslog

Kismet saja yang diolah dan disimpan dalam *database*. Dengan *frontend* Snorby, grafik dapat ditampilkan menurut masing-masing *signature* yang terdapat dalam *file* kismet.rules. Selain itu, dengan Snorby, dapat dilakukan pencarian *alert* menurut jangka waktunya, sehingga hal ini dapat memudahkan administrator dalam memonitor jaringannya.

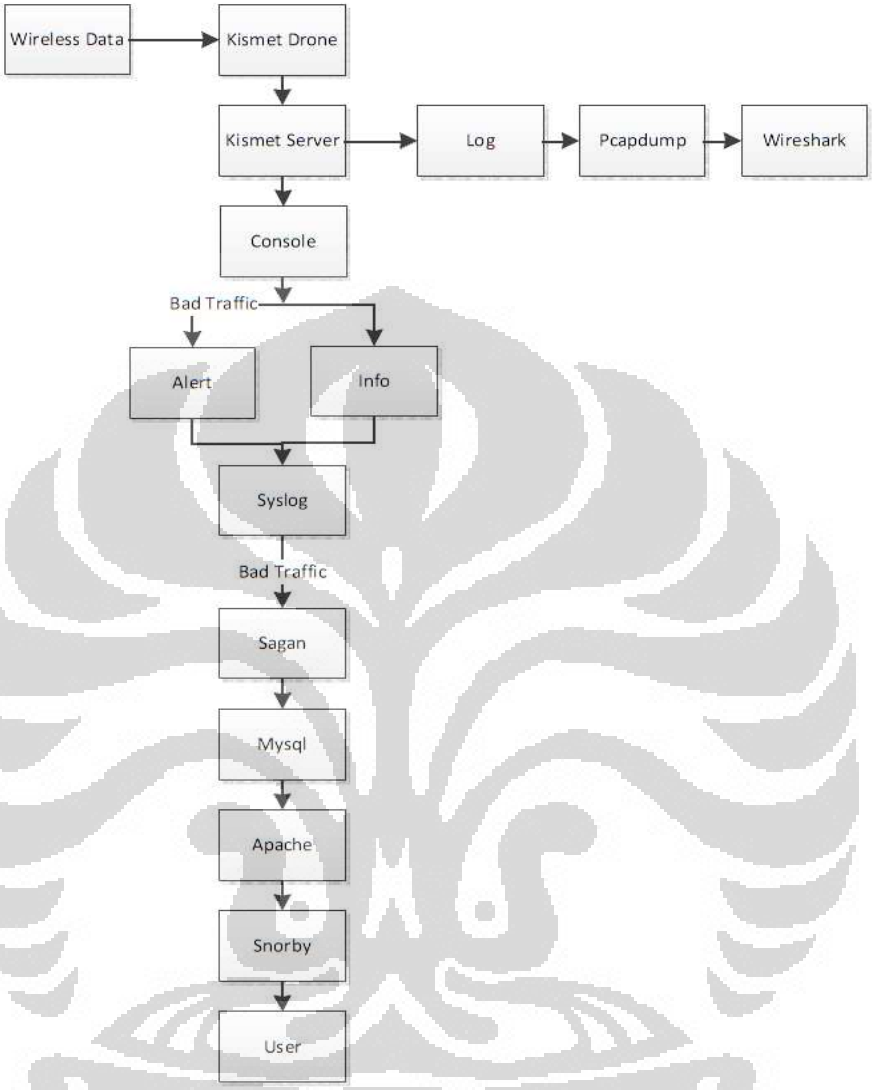
Jika sudah melakukan instalasi dan konfigurasi dasar dengan benar, maka sistem WIDS ini dapat dikonfigurasi sesuai dengan keinginan. Konfigurasi dilakukan di Kismet, Sagan dan di Snorby. Sagan akan memilih *event* yang ada di syslog secara *real-time*. Sagan akan menyimpan data yang sudah diolah ke *database* Snorby, dan Snorby akan menampilkannya di *website*. Gambar 3.2. merupakan rancangan sederhana dari WIDS ini.



Gambar 3.2. Arsitektur WIDS

Pada jaringan tersebut digunakan perangkat *Wireless Router* sebagai sensor. Pada sistem WIDS ini semua menggunakan Linux Ubuntu sebagai sistem operasi utama. Sensor dengan mode monitor akan mengumpulkan atau menangkap data-data yang lalu lalang di udara dan mengirimkannya ke Kismet *server* untuk diproses dan *plugin* syslog akan membuat semua keluaran Kismet dikirimkan ke syslog. Sagan akan memilih info dari syslog, mana yang sesuai kriteria di *rules*-nya. Setelah itu, Sagan akan memasukkan data yang sudah diprosesnya ke *database* agar dapat digunakan Snorby sebagai *database*-nya. Kemudian data-data tersebut akan disajikan pada sebuah *interface* Snorby yang

akan menampilkan data dalam bentuk grafik dan fungsi *query* yang akan memudahkan administrator dalam memonitor jaringannya.



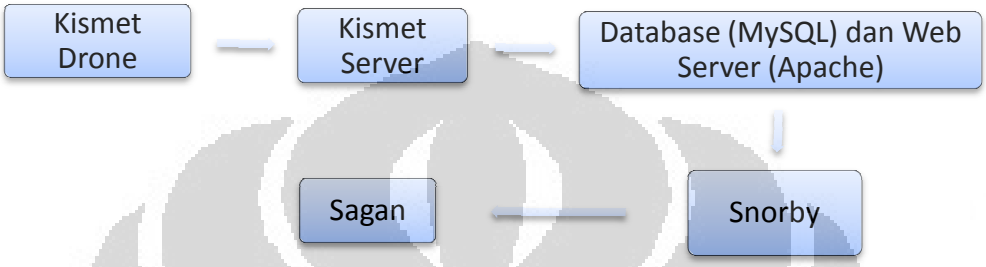
Gambar 3.3. *Work flow* diagram WIDS

Sistem WIDS yang akan dibangun menggunakan komponen Kismet, Sagan, MySQL *database*, Apache, dan Snorby. Data *wireless* ditangkap oleh Kismet *drone* untuk kemudian dikirimkan ke Kismet *server* untuk diolah. Kismet *server* kemudian menggunakan *plugin* syslog untuk mengalirkan *alert* dan info dari Kismet *server* ke syslog. Sagan kemudian mencari info syslog mana yang sesuai dengan ciri-ciri dalam kismet.rules dan menyimpannya ke dalam *database* MySQL. Kemudian Apache akan menampilkan data dalam *database* ke Snorby.

Lalu administrator dapat membuat berbagai jenis *query* pada *form* yang ditampilkan di halaman *website* untuk menganalisis data *alert*.

3.2 Tahapan Instalasi dan Konfigurasi *Tools* WIDS

Berikut gambar dan tahapan instalasi dan konfigurasi yang dilakukan untuk membangun sistem WIDS.



Gambar 3.4. Tahapan instalasi dan konfigurasi sistem WIDS

1. Kismet *drone*

   Kismet *drone* dalam pengujian ini menggunakan perangkat *Wireless Router* TP-LINK TL-WR741ND yang mendukung sistem operasi OpenWRT. Pertama, *firmware* bawaan TP-LINK akan diganti dengan sistem operasi OpenWRT agar program Kismet *drone* dapat diinstal. Setelah *firmware* telah diganti, Kismet *drone* akan diinstal dari *repository* OpenWRT. Setelah OpenWRT diinstal, kode negara dalam pengaturan sistem harus diganti agar sesuai dengan regulasi *channel* yang diperbolehkan. Untuk Indonesia, *channel* yang diperbolehkan beroperasi adalah *channel* 1 sampai 13.

2. Kismet *server*

   Kismet *server* diinstal dan dikonfigurasi di *server*. *Server* menggunakan *laptop* yang mempunyai sistem operasi Linux. Dalam pengujian ini, penulis menggunakan Linux Ubuntu 11.10. Kismet *server* merupakan program utama, karena program ini merupakan *tools* WIDS yang akan digunakan dalam pengujian. Di dalam Kismet *server* terdapat *alert-alert* yang

menunjukkan bahwa terdapat serangan atau anomali dalam suatu jaringan *wireless*.

3. *Database* dan *web server*

   Module report yang digunakan dan telah terintegrasi dengan baik dengan Sagan yaitu Snorby yang digunakan untuk mengelola data-data *security event*. Berikut ini adalah beberapa keuntungan dari Snorby yaitu:

   - *Log-log* yang sulit untuk dibaca akan menjadi mudah untuk dibaca.
   - Data-data dapat dicari sesuai dengan kriteria tertentu.

   Namun, untuk menginstal Snorby ini dibutuhkan beberapa *software* pendukung yaitu MySQL sebagai *database* dan Apache sebagai *web server*. Kedua *tools* ini diinstal di *server*.

4. Snorby

   Tahap selanjutnya adalah menginstal Snorby. Snorby adalah aplikasi *frontend* berbasis *Ruby on Rails*. *Ruby on Rails* adalah sebuah kerangka Ruby berbasis *open source* untuk pengembangan aplikasi *website*. [20]

   Snorby nantinya akan diintegrasikan dengan Apache agar dapat menampilkannya di *website*. Snorby akan diinstal di *server*.

5. Sagan

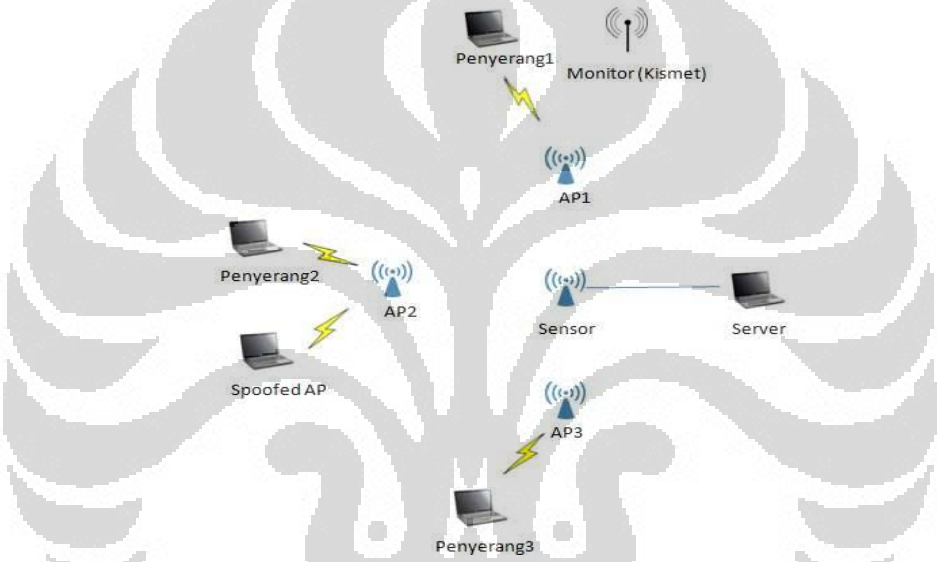
   Sagan juga akan diinstal di *server*. Sagan berfungsi sebagai penghubung antara Kismet dengan Snorby. Sagan akan dikonfigurasi agar menggunakan *database* Snorby yang nantinya keluaran *alert*-nya akan ditampilkan di *frontend* Snorby. Sagan membutuhkan syslog dalam memproses *alert-alert* yang masuk. Sehingga dibutuhkan konfigurasi syslog yang benar agar Sagan dapat bekerja dengan baik.

Jadi, selain Kismet *drone*, program yang lain akan diinstal dan dikonfigurasi di *server*. Detail penginstalan masing-masing *tools* dapat dilihat di lampiran.

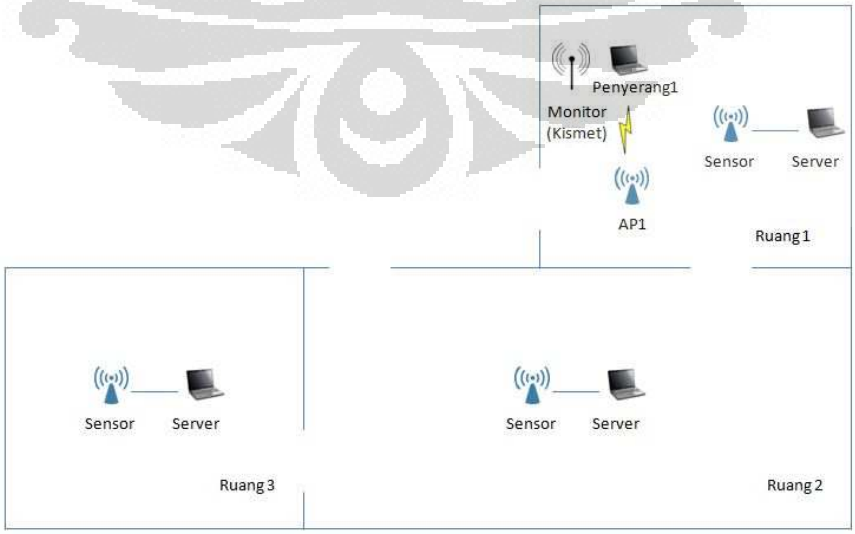
3.3 Disain Jaringan

Pada skripsi ini, jaringan yang akan digunakan adalah jaringan infrastruktur sederhana yang terdiri dari AP yang sah, *spoofed* AP, dan penyerang.

Access Point yang sah akan dinamakan AP1, AP2, dan AP3. *Spoofed* AP akan dinamakan SpoofedAP, dan AP yang mempunyai keamanan WPS adalah AP1. Skenario yang akan digunakan adalah mengubah parameter banyaknya serangan dan peletakan sensor. Kismet akan digunakan pada sisi penyerang juga agar dapat memonitor penyerang1 untuk dapat diketahui kapan waktunya serangan diluncurkan, sehingga dapat diketahui perbedaan waktu dengan yang terdeteksi di Kismet. Kismet pada sisi penyerang1 akan berfungsi sebagai *sniffer* dan dikonfigurasi agar tetap memonitor pada *channel* di mana penyerang1 melakukan serangan. Berikut ini adalah gambar dari disain jaringan WIDS.



Gambar 3.5. Disain Jaringan WIDS (1)



Gambar 3.6. Disain jaringan WIDS (2)

Untuk parameter banyaknya serangan, akan digunakan AP1, AP2, dan AP3. Untuk parameter peletakan sensor, akan digunakan salah satu AP saja, yaitu AP1. Selain itu, akan digunakan *spoofed* AP dan serangan *brute force* WPS untuk menguji performa dari sistem WIDS ini.

BAB 4
PENGUJIAN DAN ANALISIS

Pada bab ini akan dilakukan pengujian sistem yang sudah dibuat berdasarkan perancangan pada bab 3. Pengujian sistem dilakukan dengan melakukan beberapa serangan untuk mengetahui apakah WIDS dapat bekerja dengan baik.

4.1 Metode dan Skenario Pengujian

Pengujian WIDS pada skripsi ini dilakukan dengan dua metode untuk menguji apakah sistem WIDS ini dapat berfungsi dengan baik dan juga bagaimana daya tahan dan keakuratan sistem ini dalam mendeteksi adanya serangan. Dua metode tersebut, yaitu *functionality test* dan penghitungan *response time*.

Skenario yang dilakukan pada skripsi ini adalah menguji dengan beberapa serangan bagaimana sistem WIDS ini dapat mendeteksi serangan tersebut. Selain itu, memilih satu serangan yang dapat diukur untuk kemudian topologi jaringan akan diubah agar dapat diketahui bagaimana performansi sistem WIDS jika dihadapkan pada kondisi-kondisi tertentu. Kondisi-kondisi ini diantaranya bagaimana jika satu sensor mendeteksi 1 sampai 3 serangan dan bagaimana jika sensor diletakkan di ruangan yang sama dan diletakkan di ruangan yang berbeda dari penyerang. Dari penjelasan tersebut dapat dirinci metode dan skenarionya sebagai berikut:

1. *Functionality test*

   Functionality test bertujuan untuk menguji apakah sistem WIDS ini dapat berfungsi dengan baik dan sesuai dengan kriteria IDS pada umumnya. WIDS diharapkan dapat mendeteksi ketika ada serangan atau anomali di dalam jaringan dan kemudian *alert* dari serangan atau anomali tersebut dikirimkan ke Snorby. Selain itu, Kismet juga menyimpan *file log*-nya dengan format pcap yang dapat dibaca Wireshark. Sehingga dapat diketahui apa yang sebenarnya terjadi pada jaringan. Pada *functionality test* ini akan dilakukan beberapa pengujian yang menggunakan tipe serangan yang berbeda, yaitu:

a. *Spoofing*

Serangan ini akan menggunakan *spoofed* AP yang meniru MAC *address* AP yang sah sehingga dapat mengelabui *client* agar dapat terhubung dengan *spoofed* AP tersebut.

b. *Brute force*

Serangan akan dilakukan dengan menyerang AP yang mempunyai sistem keamanan WPS (*Wi-Fi Protected Setup*) dengan mencoba melakukan otentikasi dengan PIN yang acak secara langsung.

c. *Flooding*

Serangan akan dilakukan dengan menginjeksi paket de-otentikasi dengan sumber MAC *address* AP yang di-*spoof* dengan tujuan ke semua MAC *address* (*broadcast*). Serangan ini merupakan serangan yang dapat dihitung, sehingga serangan ini akan digunakan untuk menguji performansi WIDS jika dihadapkan pada kondisi-kondisi tertentu, seperti banyaknya serangan dan peletakan sensor terhadap penyerang. *Functionality test* pada serangan ini dapat dihitung dari jumlah *alert* dan *frame* yang terdeteksi. Dari nilai-nilai ini dapat diketahui berapa *false negative* yang dihasilkan sistem WIDS ini.

2. *Response time*

Response time adalah waktu yang dibutuhkan sistem untuk mendeteksi serangan. Karena jaringan WLAN menggunakan udara sebagai media transmisinya, akan ada perbedaan waktu yang lebih besar dalam pendeteksian serangan dibandingkan dengan dengan media kabel karena media udara dapat dipengaruhi oleh banyak faktor. Waktu dalam mendeteksi adanya serangan dapat dihitung dari nilai perbedaan waktu yang dibutuhkan sistem untuk mendeteksi adanya serangan dibandingkan pada saat dikirim. Pada penghitungan *response time* akan digunakan serangan *flooding*, yaitu *de-authentication flood* karena serangan ini dapat diukur. Skenario akan sama seperti pada *functionality test*, yaitu menguji performansi WIDS jika dihadapkan pada kondisi-kondisi tertentu, seperti banyaknya serangan dan peletakan sensor dengan penyerang.

Dapat disimpulkan bahwa skenario-skenarionya adalah sebagai berikut:

1. Melakukan analisis pendeteksian *spoofed* AP
2. Melakukan analisis pendeteksian serangan *brute force* WPS
3. Melakukan penghitungan data dan analisis serangan *de-authentication flood* dengan parameter banyaknya serangan (trafik)
4. Melakukan penghitungan data dan analisis serangan *de-authentication flood* dengan parameter peletakan sensor

4.2 Penghitungan dan Analisis

Pembahasan penghitungan dan analisis akan dibagi berdasarkan metode pengujian, yaitu berdasarkan *functionality test* dan penghitungan *response time.*

4.2.1 *Functionality test*

Functionality test bertujuan untuk menguji apakah sistem WIDS ini dapat berfungsi dengan baik dan sesuai dengan kriteria IDS pada umumnya. Serangan yang akan dilakukan adalah *spoofed* AP, serangan *brute force* WPS, dan serangan *de-authentication flood.*

4.2.1.1 *Spoofed* AP

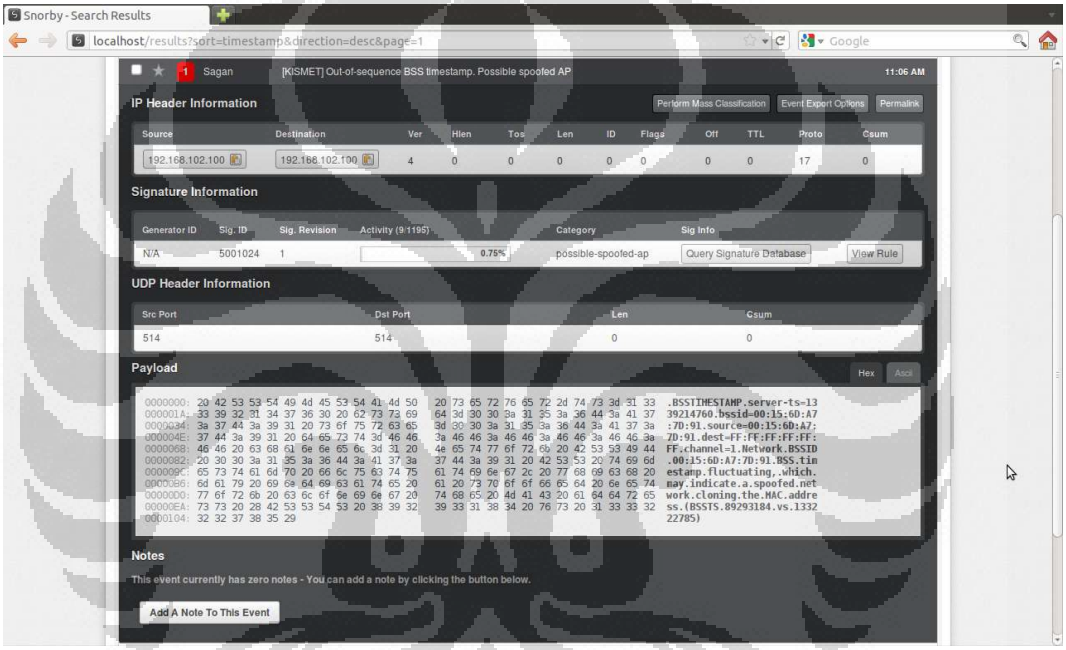
Pengujian pendeteksian *spoofed* AP ini dilakukan dengan menggunakan *tools* Hostapd yang merupakan *tools* untuk sistem operasi Linux yang berfungsi untuk membuat AP sendiri dengan menggunakan *laptop*. Dengan Hostapd, penyerang dapat menggunakan keamanan otentikasi dan enkripsi sesuai dengan keinginan. Pemilihan *channel* dan penamaan SSID juga dapat diatur. Pada pengujian ini, *spoofed* AP akan menyamar menjadi AP asli.

Pada pengujian ini akan dilakukan *spoofing* AP dari AP yang sah. *Spoofed* AP dideteksi oleh Kismet kemudian Kismet akan mengirimkan *alert* ke syslog. Sagan akan mencari info syslog yang sesuai dengan kriteria *spoofed* AP. Sagan akan menyimpan *alert* ini ke *database* sehingga Snorby dapat menampilkannya.

Spoofed AP ini akan terdeteksi oleh Kismet dengan menganalisis *timestamp* dari *beacon frame* dari AP. Kismet membandingkan *beacon frame* yang dikirimkan AP yang sah dan *spoofed* AP. Pada *beacon frame* terdapat

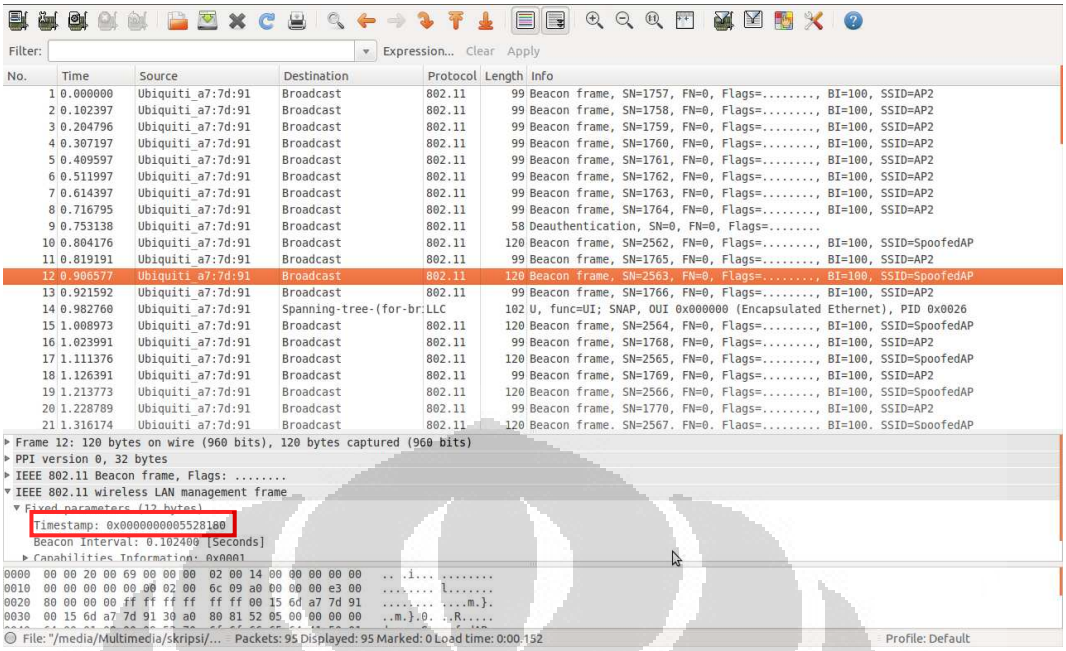
timestamp yang berguna untuk menyinkronkan transaksi antara *client* dan AP. BSS *timestamp* dikirim bersamaan dengan *beacon frame* dan beberapa *probe frame* dan tidak dapat di-*spoof* dengan *firmware* standar ataupun *driver*, bahkan ketika memalsukan *frame*. BSS yang tidak cocok merupakan upaya untuk *spoofing* SSID dan BSSID dari AP. *Alert* ini dapat meminimalkan *false positive*. [14]

Pada Gambar 4.1. dapat dilihat detil dari *alert* BSSTIMESTAMP yang ditampilkan di Snorby.



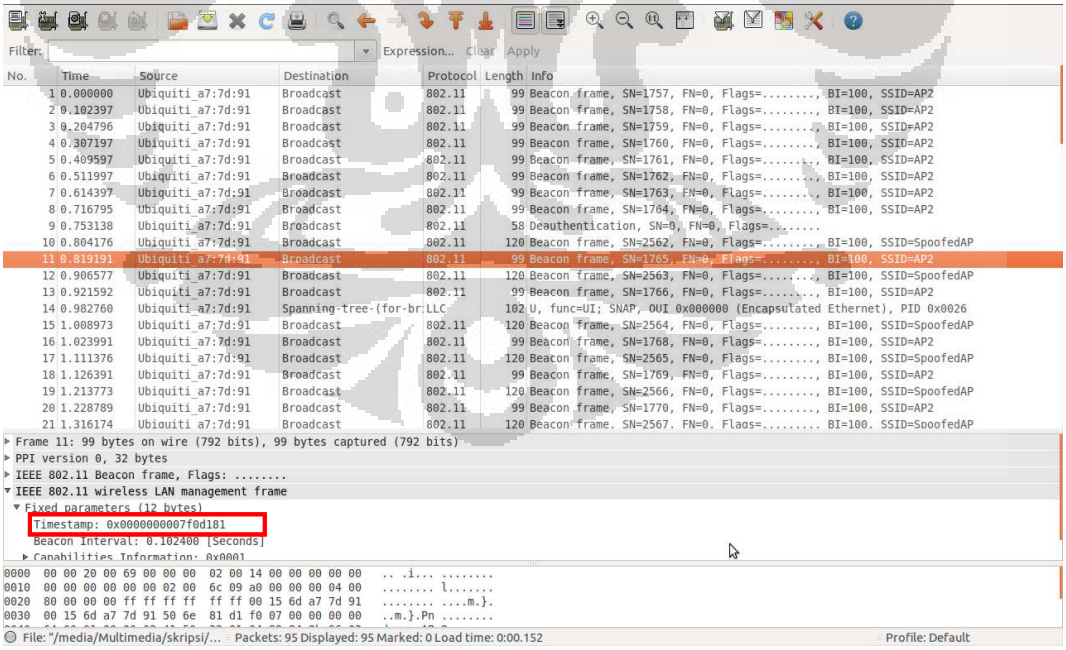
Gambar 4.1. Detil *alert spoofed* AP (BSSTIMESTAMP)

Terlihat di akhir *payload* ada BSSTS 89293184 vs 133222785. Kismet juga menyimpan *file* pcap yang dapat dibaca oleh Wireshark. *Beacon frame* yang berhasil ditangkap sensor dapat dilihat pada tampilan Wireshark pada Gambar 4.2.



Gambar 4.2. *Timestamp* 89293184 untuk *spoofed* AP

89293184 merupakan nilai desimal dari *timestamp* yang mempunyai bilangan heksadesimal 0x000000005528180.



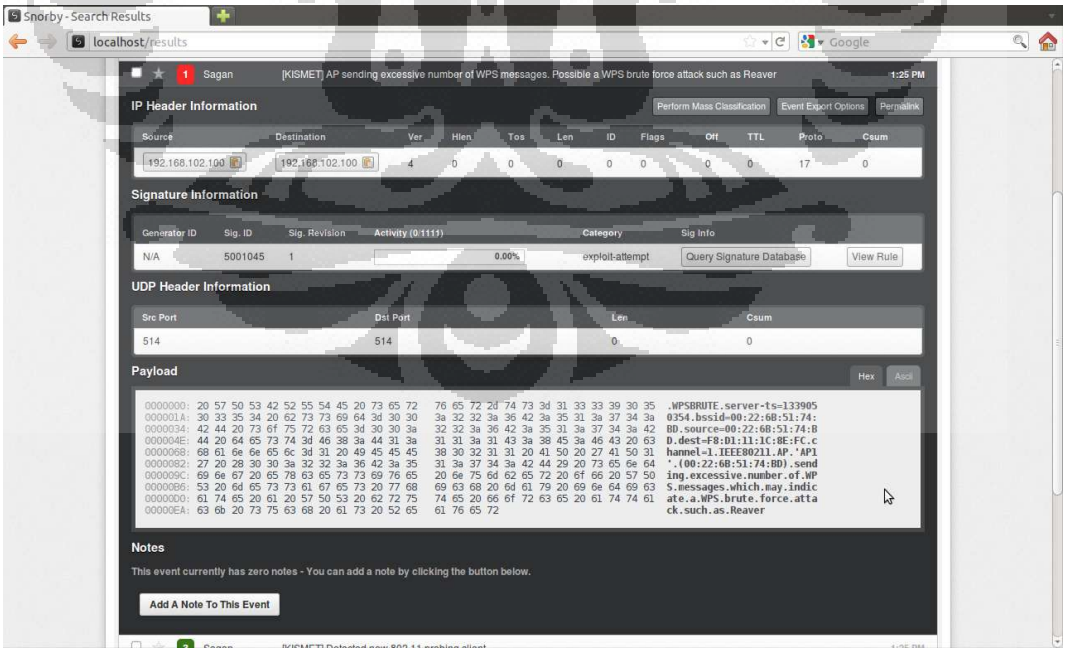
Gambar 4.3. *Timestamp* 133222785 untuk *spoofed* AP

133222785 merupakan nilai desimal dari *timestamp* yang mempunyai nilai heksadesimal 0x000000007F0D181. Nilai *timestamp* ini dapat dilihat pada Gambar 4.3. *Spoofed* AP dapat mengakibatkan data-data dalam jaringan tersebut tidak aman karena *spoofed* AP menangkap data yang seharusnya menuju ke AP yang sah. Oleh karena itu, serangan ini dapat dikategorikan serangan yang perlu diperhatikan.

4.2.1.2 Serangan *Brute Force* WPS

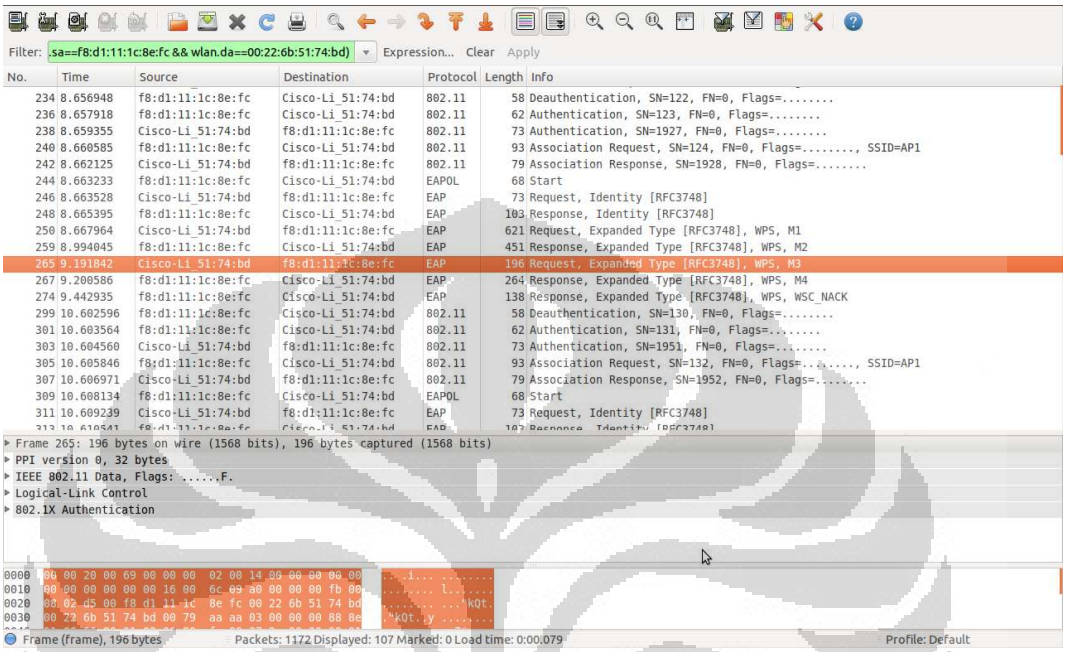
Pengujian akan dilakukan dengan menggunakan *tools* bernama Reaver. Reaver merupakan salah satu *networking tools* yang dapat membuat penyerang mengetahui PIN WPS (*Wi-Fi Protected Setup*) sekaligus mengetahui keamanan yang digunakan AP tersebut. Penyerangan akan dilakukan ke AP1 yang mempunyai sistem keamanan WPS.

Serangan ini terdeteksi oleh Kismet. Pada saat mendeteksi serangan ini, Kismet dikonfigurasi agar hanya memonitor *channel* 1 agar paket transaksi WPS ini tidak terlewat. Tampilan *alert* untuk serangan ini ditampilkan di Snorby dapat dilihat pada Gambar 4.4.



Gambar 4.4. Detil *alert brute force* WPS di Snorby

Kismet menyimpan *log*-nya dalam format pcap yang dapat dibaca Wireshark. Gambar 4.5. memperlihatkan pendeteksian *brute force* WPS pada Wireshark.



Gambar 4.5. Pendeteksian *brute force* WPS di Wireshark

Dengan mem-*filter* data-data yang terdeteksi dengan perintah "(wlan.sa == 00:22:6b:51:74:bd && wlan.da == f8:d1:11:1c:8e:fc) || (wlan.sa == f8:d1:11:1c:8e:fc && wlan.da == 00:22:6b:51:74:bd)" dapat diketahui transaksi yang dilakukan antara penyerang dan AP. MAC *address* AP adalah 00:22:6B:51:74:BD dan MAC *address* penyerang adalah F8:D1:11:1C:8E:FC.

AP mengirim paket M1 setelah penyerang memberitahu AP bahwa dia ingin melakukan otentikasi dan asosiasi. Setelah itu, penyerang mengirim paket M2 dan dibalas AP dengan paket M3. Kemudian penyerang membalas lagi dengan mengirimkan paket M4. Setelah PIN yang dikirimkan penyerang ternyata salah, AP mengirimkan *frame* de-otentikasi dan memutuskan koneksi dengan penyerang. Berikutnya penyerang kembali mencoba PIN baru dan seterusnya. Kismet mendeteksi adanya serangan jika paket M3 ada lebih dari 1 dalam 5 menit [22].

Vulnerabilitas WPS ini membuat *user* mematikan fungsi WPS di *Wireless Router*-nya. Tetapi, banyak *firmware* yang tidak mempunyai pilihan untuk mematikan WPS dalam sistemnya, seperti *Wireless Router* yang digunakan dalam pengujian ini., yaitu Linksys WR54G2.

4.2.1.3 Serangan *De-authentication Flood*

Pengujian *flooding* dilakukan dengan menggunakan *tools* bernama Aircrack-ng. Aircrack-ng merupakan salah satu *networking tools* yang mempunyai banyak fungsi yang bertujuan untuk mengaudit suatu jaringan *wireless*, yang dapat digunakan untuk meretas WEP dan WPA-PSK *key*, melakukan serangan *authentication* dan *de-authentication flood*, dan menginjeksi paket ARP *request*. Dalam pengujian ini akan dilakukan serangan *de-authentication flood* dengan skenario memutuskan semua *client* yang terhubung dengan AP korban. Serangan ini merupakan serangan awal untuk serangan *Man-in-the-middle* agar *client* yang terputus dengan AP yang sah membuat koneksi dengan AP palsu. Serangan akan dilakukan dengan membandingkan kinerja WIDS dalam mendeteksi serangan *de-authentication flood* dengan parameter banyaknya serangan dan peletakan sensor.

1. Banyaknya Serangan

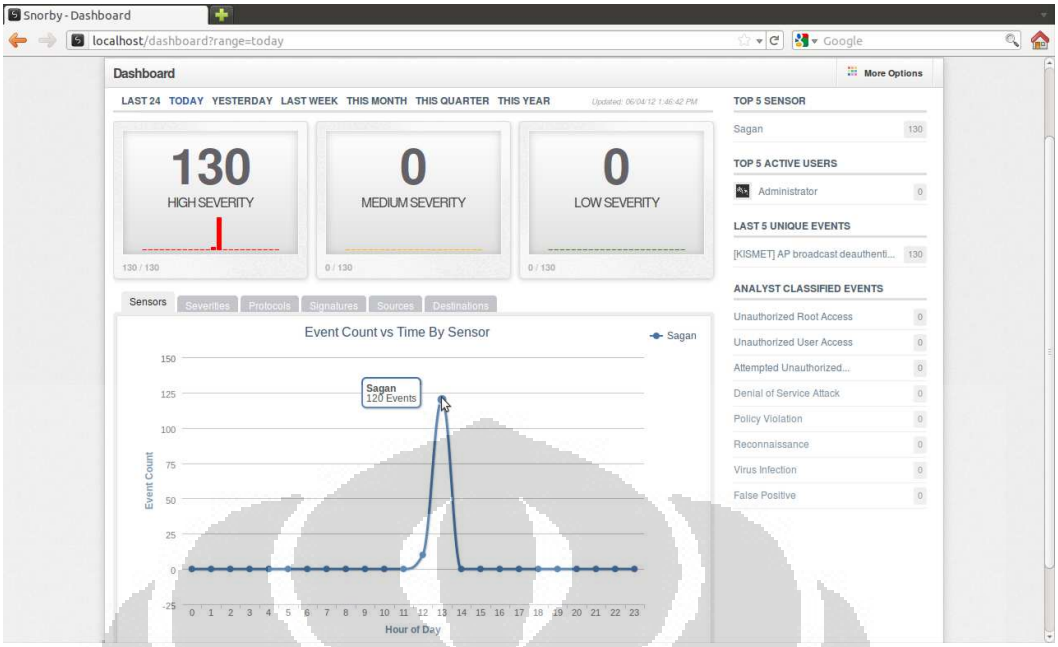
   Pada pengujian ini, serangan akan ditambahkan sampai 3 serangan. Serangan dilakukan oleh penyerang1 dengan menyamar sebagai AP1 dengan MAC *address* 00:22:6B:51:74:BD, penyerang2 dengan menyamar sebagai AP2 dengan MAC *address* 00:15:6D:A7:7D:91, dan penyerang3 dengan menyamar sebagai AP3 dengan MAC *address* 00:80:C8:AD:38:00.

   Sensor akan diletakkan pada ruangan yang sama dengan penyerang1, penyerang2, dan penyerang3. Pada awalnya, serangan hanya akan dilakukan oleh penyerang1, kemudian akan ditambah dengan serangan dari penyerang2, dan terakhir akan ditambah dengan serangan dari penyerang3. Semua penyerang akan ditempatkan pada ruangan yang sama. Penyerang2 dan penyerang3 akan bertugas sebagai pengganggu WIDS dalam mendeteksi serangan dari penyerang1 sehingga nantinya akan dihitung berapa nilai yang terdeteksi dari penyerang1 jika diganggu oleh penyerang2 dan penyerang3.

Masing-masing pengujian akan dilakukan 10 kali. Penyerang akan menyamar sebagai AP yang nantinya akan mengirim 20 paket de-otentikasi secara *broadcast*. Satu paket berisi 128 *frame* de-otentikasi. Parameter yang dihitung pertama adalah *functionality test*. Pada pengujian ini, *channel hopping* 1:3,7,13,2,8,3,9,4,10,5,11,6,12 diaktifkan pada konfigurasi Kismet. 1:3 berarti Kismet akan memonitor *channel* tersebut 3 kali lebih lama dibandingkan *channel* lainnya. Pada umumnya, administrator jaringan akan mengkonfigurasi WIDS-nya agar dapat memonitor *channel* di mana AP-nya berada. Pada pengujian ini, AP akan berada pada *channel* 1. Karena penyerang menyamar sebagai AP asli, penyerang juga melakukan aksinya pada *channel* 1. Pada skenario ini akan dilakukan serangan *de-authentication flood* dengan menggunakan *tools* Aircrack-ng pada sistem operasi BackTrack 5 R2 yang berbasis Linux Ubuntu 10.04.

a. Satu Serangan

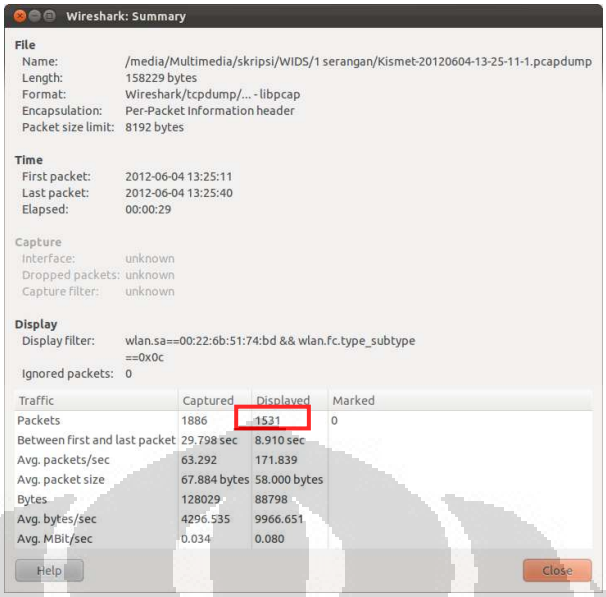
   Sensor akan diletakkan dalam ruangan yang sama dengan penyerang dengan mengirimkan serangan sebanyak 20 paket. Pada skenario ini, yang akan melakukan penyerangan hanyalah penyerang1. Serangan dilakukan selama kurang lebih 9 detik antara jam 13:00:00 sampai jam 13:59:59. Gambar 4.6. merupakan *dashboard* yang menampilkan berapa *event* atau *alert* yang terjadi pada jam 13:00:00 sampai 13:59:59. Hasil *capture dashboard* dari Snorby menunjukkan jumlah atau banyaknya serangan yang masuk pada jaringan yang merupakan keluaran dari Sagan. Sagan mengolah *alert* dari Kismet yang masuk ke syslog.



Gambar 4.6. *Dashboard* Snorby untuk 1 serangan setelah 10 pengujian

Dapat dilihat bahwa jumlah *alert* yang dihasilkan berjumlah 120 *alert.* Karena pengujian dilakukan 10 kali, rata-rata *alert* yang dihasilkan berjumlah 12 *alert.*

Kismet juga menghasilkan keluaran *file* pcap yang dapat dibaca Wireshark. *Frame* de-otentikasi yang berhasil ditangkap sensor dapat dilihat pada tampilan Wireshark pada Gambar 4.7. Agar Wireshark hanya menampilkan *frame* de-otentikasi dari penyerang saja, di isian 'Filter' diisi dengan "wlan.sa== 00:22:6b:51:74:bd && wlan.fc.type_subtype==0x0c". 00:22:6B:51:74:BD merupakan MAC *address* AP1. 0x0c merupakan kode subtipe dari *management frame*, yaitu *frame* de-otentikasi.



Gambar 4.7. *Summary* Wireshark untuk 1 serangan pengujian pertama

Dari gambar di atas dapat dilihat bahwa Kismet mendeteksi 1531 *frame* de-otentikasi yang berasal dari penyerang1. Tabel 4.1. memperlihatkan jumlah *frame* de-otentikasi yang dilihat dari Wireshark untuk 10 kali pengujian.

Tabel 4.1. Jumlah *frame* de-otentikasi untuk 10 pengujian untuk 1 serangan

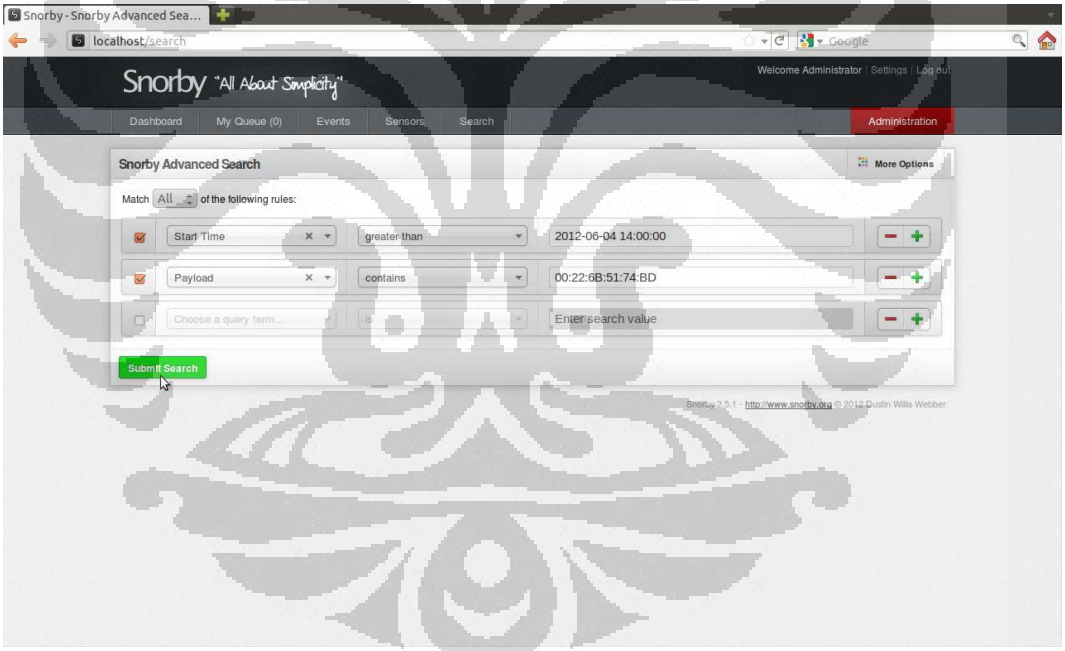
Pengujian ke-	***Frame* De-otentikasi yang Terdeteksi (*frame*)**
1	1531
2	1326
3	1250
4	1445
5	1404
6	1436
7	1271
8	1235
9	1445
10	1411
Rata-rata	1375,4

Rata-rata *frame* de-otentikasi yang dikirimkan oleh penyerang1 yang terdeteksi oleh sensor berjumlah 1375,4 *frame*.

b. Dua Serangan

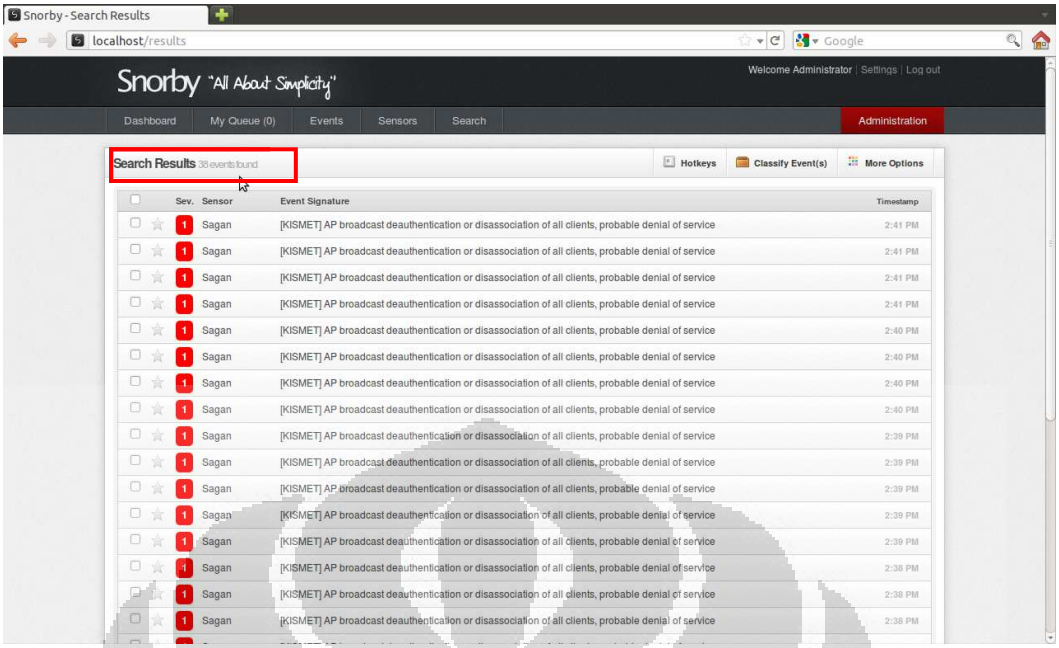
Pada skenario ini, yang akan melakukan penyerangan adalah penyerang1 dan penyerang2 secara bersamaan.

Serangan dilakukan selama kurang lebih 9 detik antara jam 14:00:00 sampai jam 14:59:59. Karena ada 2 serangan, perlu dicari serangan yang hanya berasal dari penyerang1. Pengambilan data *alert* tidak dapat dilihat dari grafik di *dashboard* Snorby karena pada grafik tersebut ditampilkan semua *alert* serangan *de-authentication flood*, tidak peduli dari mana sumbernya. Tampilan halaman pencarian di Snorby diperlihatkan pada Pengujian dilakukan 10 kali. Gambar 4.8. Pada *payload* diisi dengan MAC *address* AP1, yaitu 00:22:6B:51:74:BD. Hal ini dilakukan agar Snorby dapat mencari *alert* yang berisi *payload* 00:22:6B:51:74:BD.



Gambar 4.8. Pencarian *alert* penyerang1 untuk 2 serangan

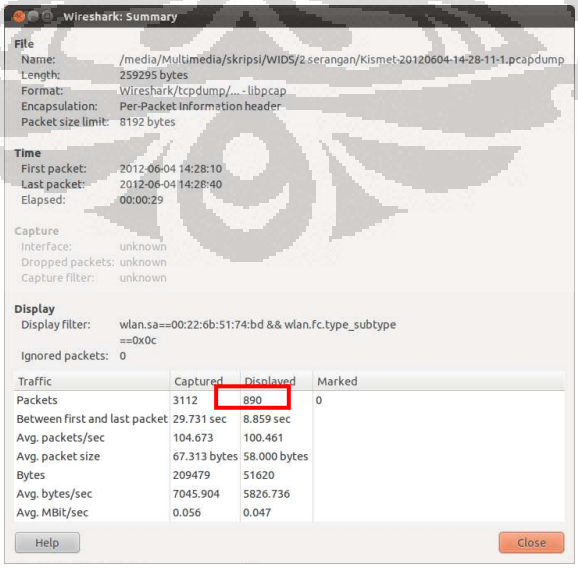
Dari pencarian tersebut dapat diketahui berapa *alert* yang dihasilkan Kismet untuk penyerang1. Hasil pencarian ini dapat dilihat pada Gambar 4.9.



Gambar 4.9. Hasil pencarian *alert* penyerang1 untuk 10 pengujian 2 serangan

Dapat dilihat bahwa jumlah *alert* yang dihasilkan berjumlah 38 *alert.* Karena pengujian dilakukan 10 kali, rata-rata *alert* yang dihasilkan berjumlah 3,8 *alert.*

Gambar 4.10. merupakan keluaran *file* pcap yang sebelumnya telah difilter terlebih dahulu agar hanya memperlihatkan *frame* de-otentikasi dari penyerang1 untuk pengujian pertama.



Gambar 4.10. *Summary* Wireshark untuk 1 serangan pengujian pertama untuk 2 serangan

Dari gambar di atas dapat dilihat bahwa Kismet mendeteksi 890 *frame* de-otentikasi yang berasal dari penyerang1. *Frame* yang terdeteksi ini lebih sedikit dibandingkan dengan pengujian pertama pada 1 serangan yang mendeteksi 1531 *frame*.

Tabel 4.2. memperlihatkan jumlah *frame* de-otentikasi yang dilihat dari Wireshark untuk 10 kali pengujian.

Tabel 4.2. Jumlah *frame* de-otentikasi penyerang1 yang terdeteksi untuk 10 pengujian 2 serangan

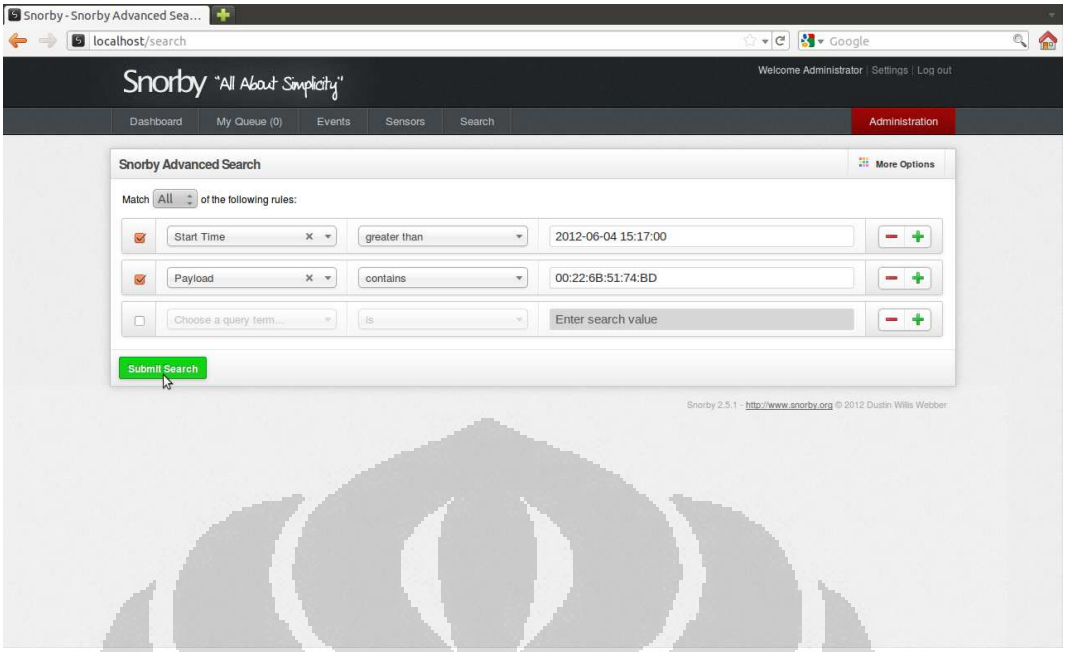
Pengujian ke-	***Frame* De-otentikasi yang Terdeteksi (*frame*)**
1	890
2	910
3	1168
4	917
5	951
6	1052
7	981
8	991
9	958
10	1026
Rata-rata	984,4

Rata-rata *frame* de-otentikasi yang dikirimkan oleh penyerang1 yang terdeteksi oleh sensor berjumlah 984,4 *frame*.

c. Tiga Serangan

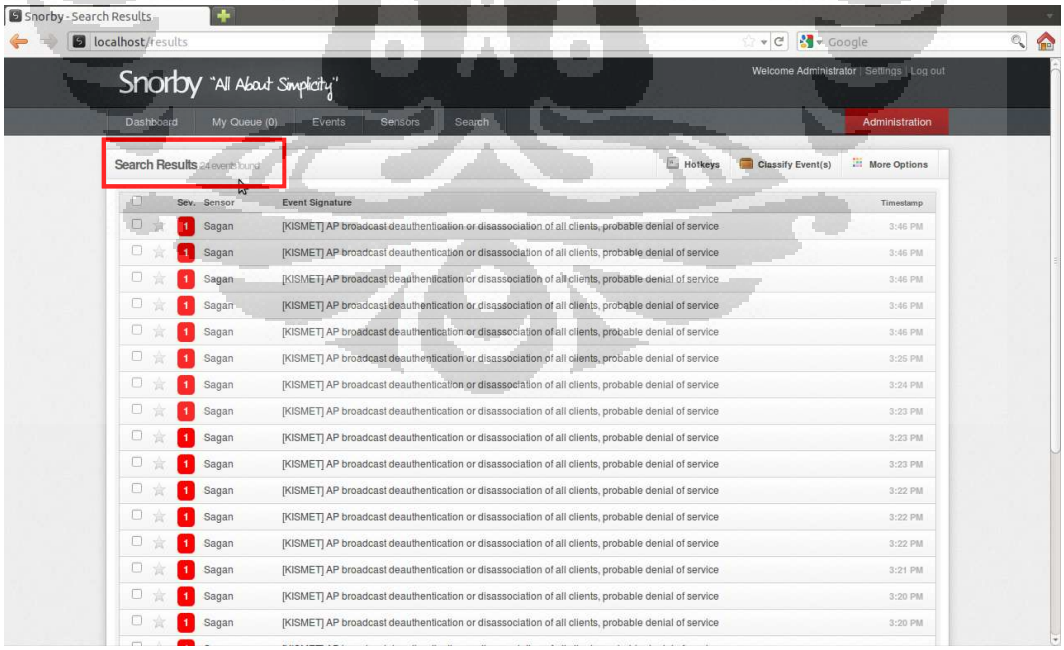
Pada skenario ini, yang akan melakukan penyerangan adalah penyerang1, penyerang2, dan penyerang3 secara bersamaan.

Serangan dilakukan selama kurang lebih 9 detik antara jam 15:17:00 sampai jam 15:59:59. Pengujian dilakukan 10 kali. Karena ada 3 serangan, perlu dicari serangan yang hanya berasal dari penyerang1. Caranya sama seperti untuk 2 serangan. Tampilan halaman pencarian di Snorby diperlihatkan pada Gambar 4.11.



Gambar 4.11. Pencarian *alert* penyerang1 pengujian pertama untuk 3 serangan

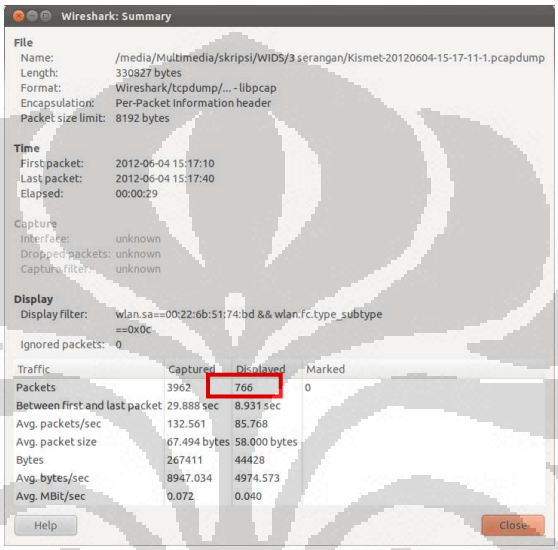
Dari pencarian tersebut dapat diketahui berapa *alert* yang dihasilkan Kismet untuk penyerang1. Hasil pencarian dapat dilihat pada Gambar 4.12.



Gambar 4.12. Hasil pencarian *alert* penyerang1 untuk 10 pengujian 3 serangan

Dapat dilihat bahwa jumlah *alert* yang dihasilkan berjumlah 24 *alert.* Karena pengujian dilakukan 10 kali, rata-rata *alert* yang dihasilkan berjumlah 2,4 *alert.*

Gambar 4.13. memperlihatkan keluaran *file* pcap yang sebelumnya telah difilter terlebih dahulu agar hanya memperlihatkan *frame* de-otentikasi dari penyerang1 untuk pengujian pertama.



Gambar 4.13. *Summary* Wireshark untuk penyerang1 pengujian pertama untuk 3 serangan

Dari gambar di atas dapat dilihat bahwa Kismet mendeteksi 766 *frame* de-otentikasi yang berasal dari penyerang1. Jumlah *frame* ini lebih sedikit dibandingkan dengan *frame* yang terdeteksi pada 2 serangan yang berjumlah 890 *frame*.

Tabel 4.3. memperlihatkan jumlah *frame* de-otentikasi yang dilihat dari Wireshark.

Tabel 4.3. Jumlah *frame* de-otentikasi penyerang1 yang terdeteksi untuk 10 pengujian pada 3 serangan

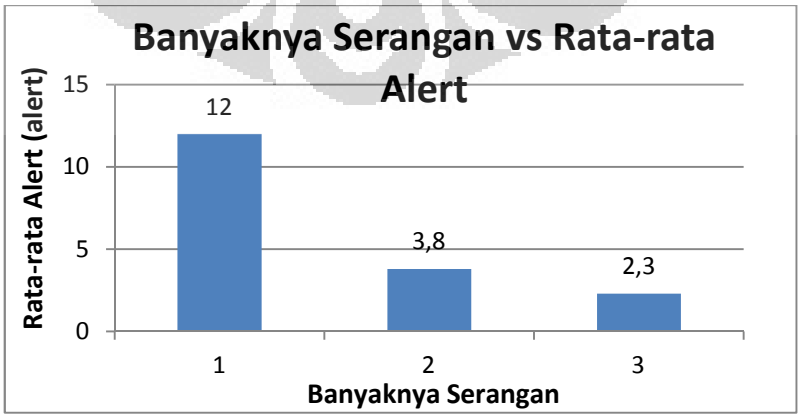
Pengujian ke-	*Frame* De-otentikasi yang Terdeteksi (*frame*)
1	766
2	834
3	595
4	752
5	762
6	576
7	792
8	783
9	1001
10	776
Rata-rata	763,7

Rata-rata *frame* de-otentikasi yang dikirimkan oleh penyerang1 yang terdeteksi oleh sensor berjumlah 763,7 *frame*.

Untuk mengetahui dengan jelas perbedaan banyaknya serangan terhadap *alert* dan *false negative*-nya, dapat dilihat melalui tabel dan grafik. Untuk *alert* dapat dilihat pada Tabel 4.4. dan Gambar 4.14.

Tabel 4.4. Rata-rata *alert* penyerang1 untuk banyaknya serangan

Banyaknya Serangan	Rata-rata *Alert* Penyerang1 (*alert*)
1	12
2	3,8
3	2,3



Gambar 4.14. Grafik rata-rata *alert* untuk banyaknya serangan

Persentase *false negative* dapat dihitung dengan persamaan:

$$\%FNS = \frac{FS1-FSn}{FS1}\ x\ 100 \qquad (4.1)$$

Keterangan:

%FNS = Persentase *False Negative frame* de-otentikasi untuk banyaknya serangan (%)

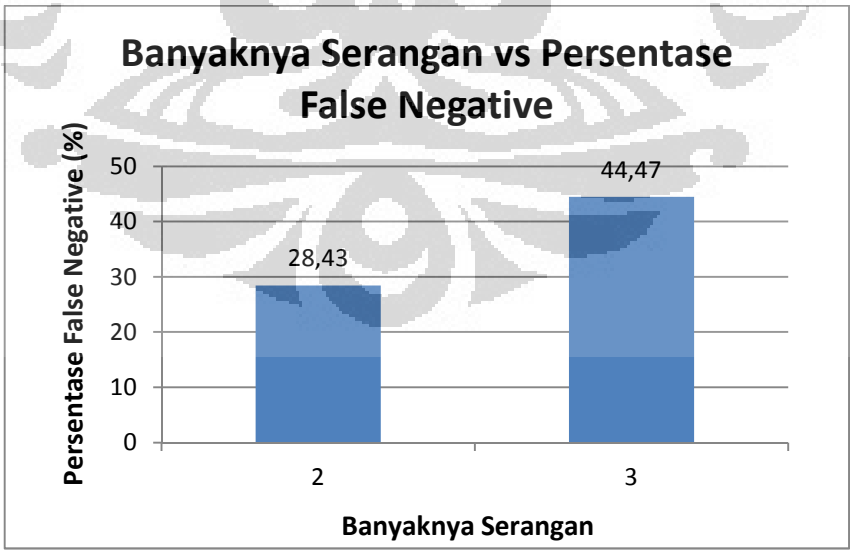
FS1 = Rata-rata *frame* de-otentikasi dari penyerang1 pada 1 serangan (*frame)*

FSn = Rata-rata *frame* de-otentikasi dari penyerang1 pada 2 atau 3 serangan (*frame*)

Persentase *false negative frame* de-otentikasi yang terdeteksi oleh sensor pada 1, 2, dan 3 serangan dapat dilihat pada Tabel 4.5. dan Gambar 4.15.

Tabel 4.5. Jumlah *frame* dan persentase *false negative* untuk banyaknya serangan

Rata-rata *Frame* De-otentikasi dari Penyerang1 yang Terdeteksi (*frame*)			**Persentase *False Negative* (%)**	
1 Serangan	2 Serangan	3 Serangan	2 Serangan	3 Serangan
1375,4	984,4	763,7	28,43	44,47



Gambar 4.15. Grafik persentase *false negative* untuk banyaknya serangan

Pada Tabel 4.4. dan Gambar 4.14. menunjukan perbandingan antara *alert* terhadap serangan yang dideteksi sensor dari penyerang1 pada 1, 2, dan 3 serangan. Terjadi penurunan *alert* terhadap bertambahnya serangan. Selain itu, pada Tabel 4.5. dan Gambar 4.15. dapat dilihat peningkatan *false negative* yang dihasilkan ketika ditambah oleh penyerang2 dan penyerang3. Adanya nilai *false negative* ini dikarenakan adanya kepadatan trafik yang disebabkan oleh *frame* de-otentikasi yang dikirimkan oleh penyerang2 dan penyerang3 membebani performansi sensor dalam menangkap *frame* dari penyerang1. Sistem WIDS akan mengolah data yang lebih banyak dari sebelumnya, sehingga sistem pada saat tertentu akan mengolah *frame* dari penyerang lain dan tidak mengolah *frame* dari penyerang1. Padatnya trafik juga akan menyebabkan sensor akan men-*drop* beberapa *frame,* sehingga sistem tidak medeteksi adanya *frame-frame* tersebut.

2. Peletakan Sensor

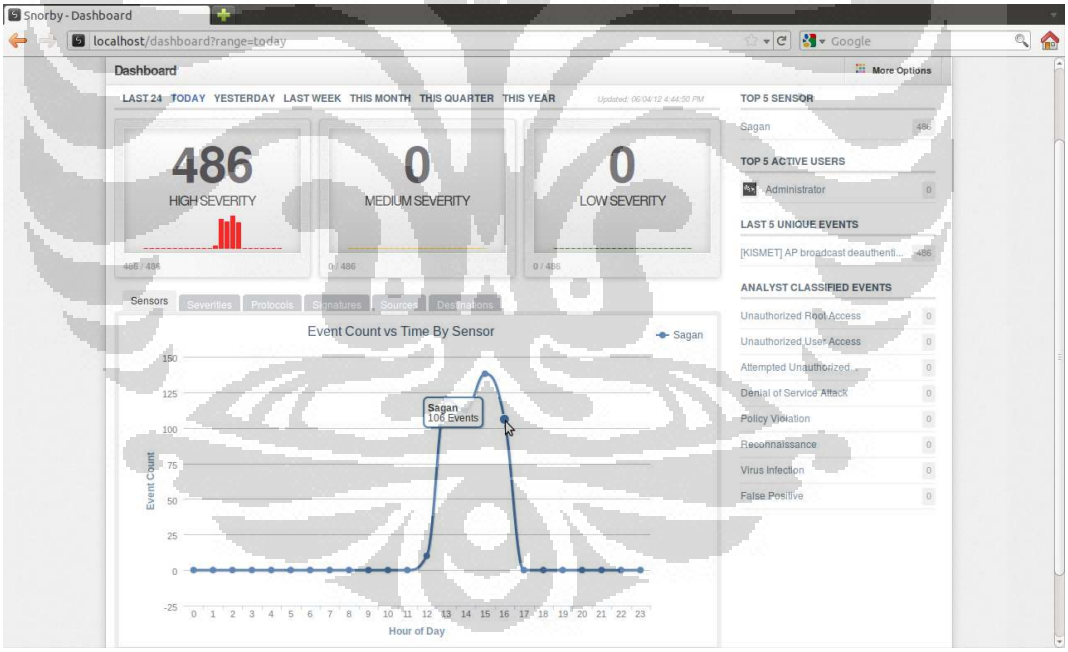
   Salah satu parameter kehandalan dari suatu sistem WIDS juga harus dapat memonitor perangkat *wireless* atau *frame wireless* pada batasan ruang tertentu, sehingga untuk menguji ini, sensor ditempatkan pada ruangan yang sama dan ruangan yang berbeda terhadap serangan.

   Serangan dilakukan dengan menyamar sebagai AP1 dengan MAC *address* 00:22:6B:51:74:BD. Sensor akan diletakkan pada ruangan yang sama (ruang 1) dan berikutnya sensor akan diletakkan pada ruangan di sampingnya (ruang 2), dan terakhir, sensor akan diletakkan pada ruangan di samping ruang 2 atau berada 2 ruangan yang berbeda dengan penyerang. Jadi, semakin lama, sensor akan berada makin jauh dari penyerang. Penyerang akan menyamar sebagai AP yang nantinya akan mengirim 20 paket de-otentikasi secara *broadcast*. Satu paket berisi 128 *frame* de-otentikasi. Parameter yang dihitung pertama adalah *functionality test*. Pada pengujian ini, *channel hopping* 1:3,7,13,2,8,3,9,4,10,5,11,6,12 diaktifkan pada konfigurasi Kismet. 1:3 berarti Kismet akan memonitor *channel* 1 tiga kali lebih lama dibandingkan *channel* lainnya. Pada umumnya, administrator jaringan akan mengkonfigurasi WIDS-nya agar dapat memonitor *channel* di mana AP-nya berada. Pada pengujian ini, AP akan berada pada *channel* 1. Karena

penyerang menyamar sebagai AP yang sah, penyerang juga melakukan aksinya pada *channel* 1. Pada skenario ini akan dilakukan serangan *de-authentication flood* dengan menggunakan *tools* Aircrack-ng pada sistem operasi BackTrack 5 R2 yang berbasis Linux Ubuntu 10.04.

a. Ruang 1

Pertama, sensor akan diletakkan dalam ruangan yang sama dengan penyerang.Serangan dilakukan selama kurang lebih 9 detik antara jam 16:00:00 sampai jam 16:59:59. Hasil *capture dashboard* dari Snorby menunjukkan jumlah atau banyaknya serangan yang masuk pada jaringan yang merupakan keluaran dari Sagan. Sagan mengolah *alert* dari Kismet yang masuk ke syslog. Pengujian dilakukan 10 kali. Gambar 4.16. merupakan *dashboard* yang menampilkan berapa *event* yang terjadi pada jam 16:00:00 sampai 16:59:59.

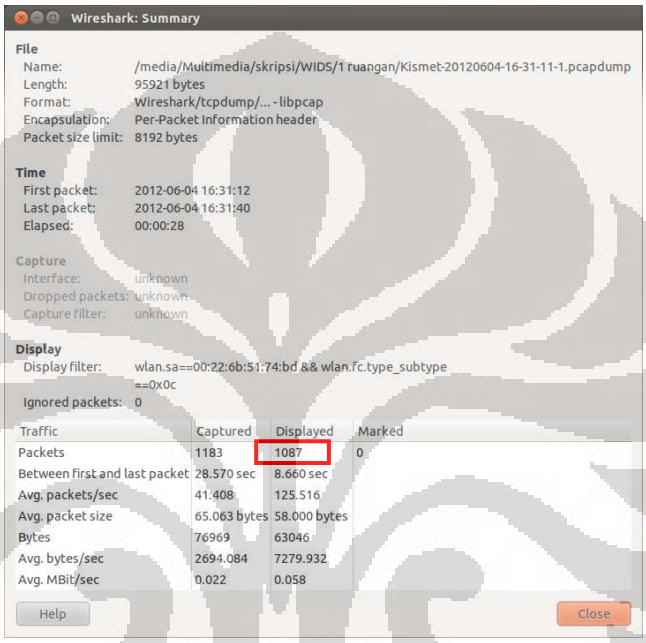


Gambar 4.16. *Dashboard* Snorby untuk ruang 1 setelah 10 pengujian

Dapat dilihat bahwa jumlah *alert* yang dihasilkan berjumlah 106 *alert.* Karena pengujian dilakukan 10 kali, rata-rata *alert* yang dihasilkan berjumlah 10,6 *alert.*

Kismet juga menghasilkan keluaran *file* pcap yang dapat dibaca Wireshark. *Frame* de-otentikasi yang berhasil ditangkap sensor pada

pengujian pertama pada ruang 1 dapat dilihat pada tampilan Wireshark pada Gambar 4.17. Agar Wireshark hanya menampilkan *frame* de-otentikasi dari penyerang saja, di isian 'Filter' diisi dengan "wlan.sa== 00:22:6b:51:74:bd && wlan.fc.type_subtype==0x0c". 00:22:6B:51:74:BD merupakan MAC *address* AP1. 0x0c merupakan kode subtipe dari *management frame*, yaitu *frame* de-otentikasi.



Gambar 4.17. *Summary* Wireshark untuk ruang 1 pengujian pertama

Dari Gambar 4.17. dapat dilihat bahwa Kismet mendeteksi 1087 *frame* de-otentikasi yang berasal dari penyerang1. Tabel 4.6. memperlihatkan jumlah *frame* de-otentikasi yang dilihat dari Wireshark untuk 10 kali pengujian untuk ruang 1.

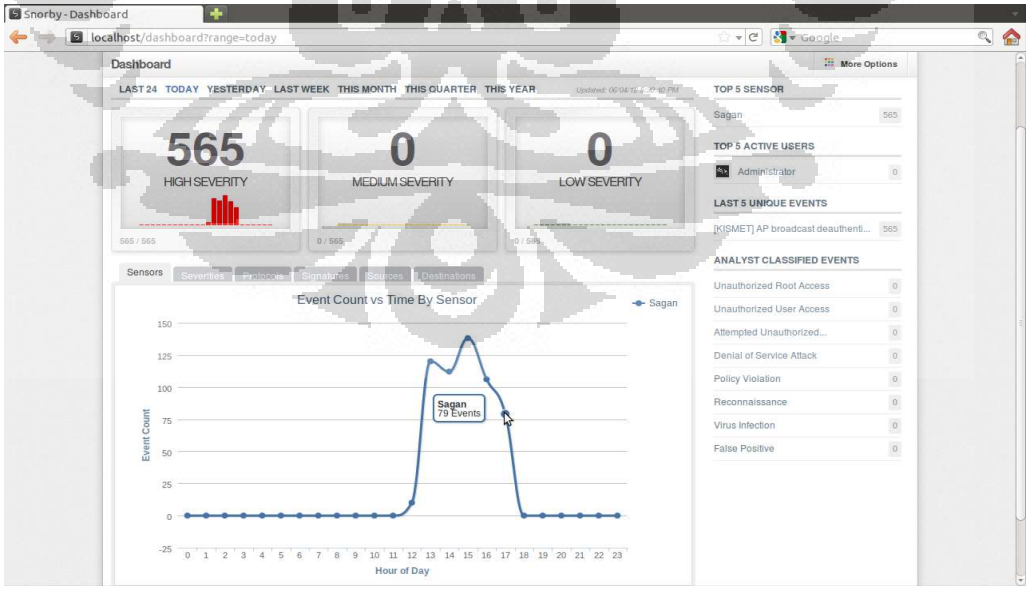
Tabel 4.6. Jumlah *frame* de-otentikasi untuk 10 pengujian pada ruang 1

Pengujian ke-	*Frame* De-otentikasi yang Terdeteksi (*frame*)
1	1087
2	1479
3	1361
4	1590
5	1458
6	1539
7	1664
8	1634
9	1437
10	1558
Rata-rata	1480,7

Rata-rata *frame* de-otentikasi yang dikirimkan oleh penyerang1 yang terdeteksi oleh sensor berjumlah 1480,7 *frame*.

b. Ruang 2

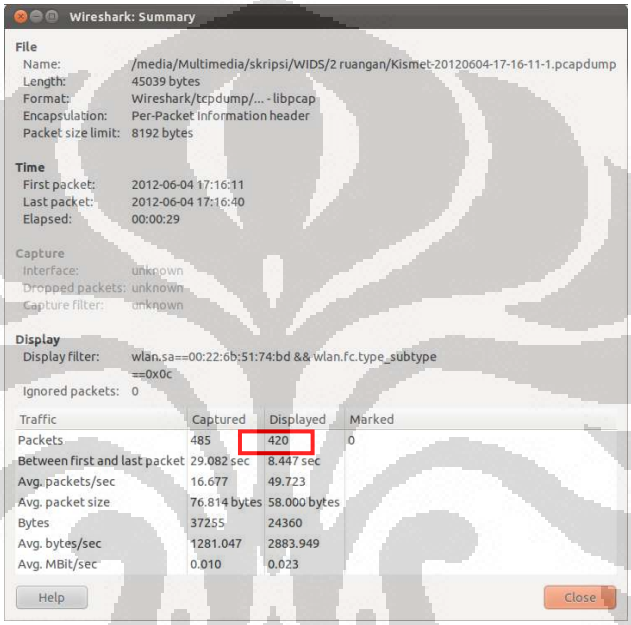
Untuk ruang 2, sensor akan diletakkan dalam ruangan yang berbeda 1 ruangan dari penyerang. Serangan dilakukan selama kurang lebih 9 detik antara jam 17:00:00 sampai jam 17:59:59. Pengujian dilakukan 10 kali. Gambar 4.18. merupakan *dashboard* yang menampilkan berapa *event* atau *alert* yang terjadi pada jam 17:00:00 sampai 17:59:59.



Gambar 4.18. *Dashboard* Snorby untuk ruang 2 setelah 10 pengujian

Dapat dilihat bahwa jumlah *alert* yang dihasilkan berjumlah 79 *alert.* Karena pengujian dilakukan 10 kali, rata-rata *alert* yang dihasilkan berjumlah 7,9 *alert.*

Berikutnya akan dihitung berapa *frame* de-otentikasi yang berhasil dideteksi sensor untuk ruang 2. Gambar 4.19. merupakan keluaran *file* pcap yang sebelumnya telah di-*filter* terlebih dahulu agar hanya memperlihatkan *frame* de-otentikasi seperti pada ruang 1.



Gambar 4.19. *Summary* Wireshark untuk ruang 2 pengujian pertama

Dari Gambar 4.19. dapat dilihat bahwa Kismet mendeteksi 420 *frame* de-otentikasi yang berasal dari penyerang1. Jumlah *frame* ini lebih sedikit dibandingkan dengan ruang 1 pengujian pertama yang mendeteksi 1087 *frame*.

Tabel 4.7. memperlihatkan jumlah *frame* de-otentikasi yang dilihat dari Wireshark untuk 10 kali pengujian pada ruang 2.

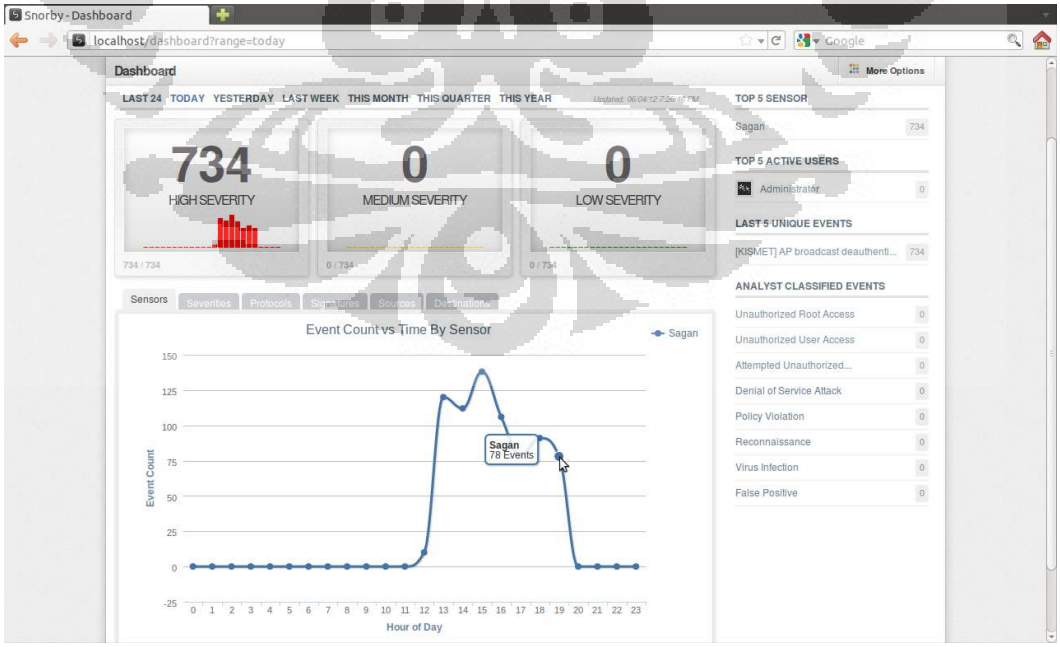
Tabel 4.7. Jumlah *frame* de-otentikasi untuk 10 pengujian pada ruang 2

Pengujian ke-	*Frame* De-otentikasi yang Terdeteksi (*frame*)
1	420
2	546
3	538
4	378
5	321
6	416
7	361
8	300
9	346
10	449
Rata-rata	407,5

Rata-rata *frame* de-otentikasi yang dikirimkan oleh penyerang1 yang terdeteksi oleh sensor berjumlah 407,5 *frame*.

c. Ruang 3

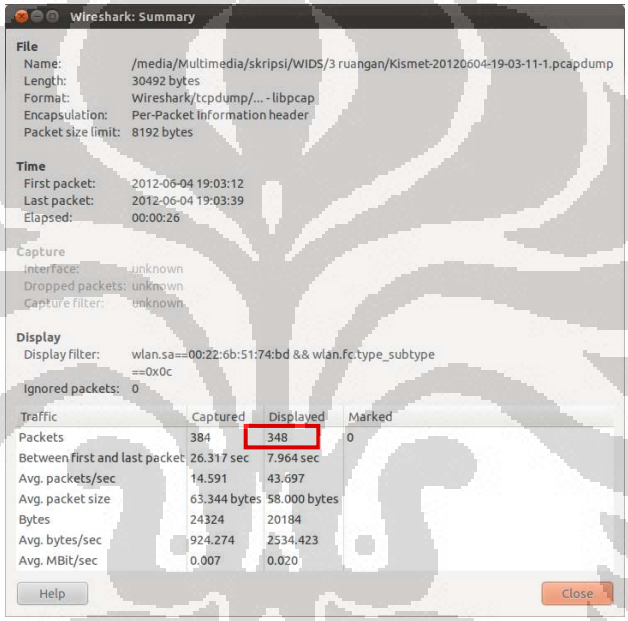
Untuk ruang 3, sensor akan diletakkan dalam ruangan yang berbeda 2 ruangan dari penyerang. Serangan dilakukan selama kurang lebih 9 detik antara jam 19:00:00 sampai jam 19:59:59. Pengujian dilakukan 10 kali. Gambar 4.20. merupakan *dashboard* yang menampilkan berapa *event* atau *alert* yang terjadi pada jam 19:00:00 sampai 19:59:59.



Gambar 4.20. *Dashboard* Snorby untuk ruang 3 setelah 10 pengujian

Dapat dilihat bahwa jumlah *alert* yang dihasilkan berjumlah 78 *alert.* Karena pengujian dilakukan 10 kali, rata-rata *alert* yang dihasilkan berjumlah 7,8 *alert.*

Berikutnya akan dihitung berapa *frame* de-otentikasi yang berhasil dideteksi sensor untuk ruang 3. Gambar 4.21. memperlihatkan jumlah *frame* de-otentikasi yang terdeteksi oleh sensor yang sebelumnya telah difilter terlebih dahulu agar hanya memperlihatkan *frame* de-otentikasi pada Wireshark.



Gambar 4.21. *Summary* Wireshark untuk ruang 3 pengujian pertama

Dari gambar di atas dapat dilihat bahwa Kismet mendeteksi 348 *frame* de-otentikasi yang berasal dari penyerang1. Jumlah *frame* ini pun lebih sedikit dibandingkan dengan pengujian pertama pada ruang 2, yaitu 420 *frame*.

Tabel 4.8. memperlihatkan jumlah *frame* de-otentikasi yang dilihat dari Wireshark untuk 10 kali pengujian pada ruang 3.

Tabel 4.8. Jumlah *frame* de-otentikasi untuk 10 pengujian pada ruang 3

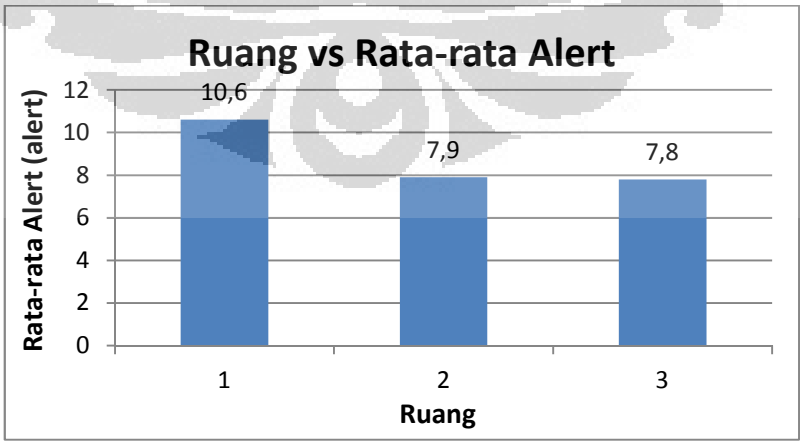
Pengujian ke-	*Frame* De-otentikasi yang Terdeteksi (*frame*)
1	348
2	329
3	300
4	276
5	298
6	434
7	283
8	337
9	405
10	370
Rata-rata	338

Rata Rata-rata *frame* de-otentikasi yang dikirimkan oleh penyerang1 yang terdeteksi oleh sensor berjumlah 338 *frame*.

Untuk mengetahui dengan jelas perbedaan peletakan sensor terhadap pada ruang 1, 2, dan 3 terhadap *alert* dan *false negative*-nya, dapat dilihat melalui tabel dan grafik. Untuk *alert* dapat dilihat pada Tabel 4.9. dan Gambar 4.22.

Tabel 4.9. Rata-rata *alert* untuk peletakan sensor

Ruang	Rata-rata *Alert* (*alert*)
1	10,6
2	7,9
3	7,8



Gambar 4.22. Grafik rata-rata *alert* untuk peletakan sensor

Persentase *false negative* dapat dihitung dengan persamaan:

$$\%FNR = \frac{FR1-FRn}{FR1}\ x\ 100 \qquad (4.2)$$

Keterangan:

%FNR = Persentase *False Negative frame* de-otentikasi untuk peletakan sensor (%)

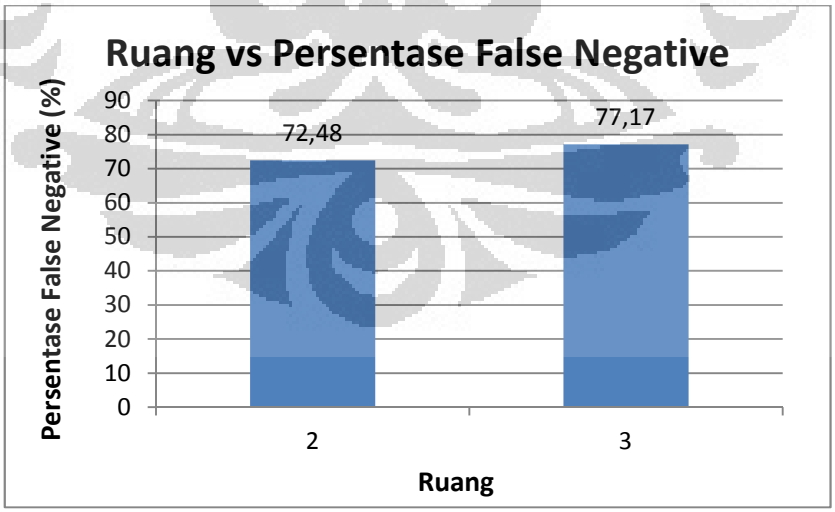
FR1 = Rata-rata *frame* de-otentikasi pada ruang1 (*frame*)

FRn = Rata-rata *frame* de-otentikasi pada ruang2 atau ruang3 (*frame*)

Persentase *false negative frame* de-otentikasi yang terdeteksi oleh sensor pada ruang 1, 2, dan 3 dapat dilihat pada tabel dan grafik yang diperlihatkan pada Tabel 4.10. dan Gambar 4.23.

Tabel 4.10. Jumlah *frame* dan persentase *false negative* untuk peletakan sensor

Rata-rata *Frame* De-otentikasi yang Terdeteksi (*frame*)			**Persentase *False negative* (%)**	
Ruang 1	Ruang 2	Ruang 3	Ruang 2	Ruang 3
1480,7	407,5	338	72,48	77,17



Gambar 4.23. Grafik persentase *false negative* untuk peletakan sensor

Tabel 4.9. dan Gambar 4.22. menunjukan perbandingan antara *alert* terhadap serangan yang dideteksi sensor pada ruang 1, 2, dan 3. Pada table dan gambar

tersebut dapat dilihat terjadi penurunan *alert* terhadap bertambahnya ruangan antara sensor dengan penyerang.

Selain itu, dari Tabel 4.10. dan Gambar 4.23. dapat dilihat *false negative* yang dihasilkan pun mengalami peningkatan yang sangat besar ketika sensor hanya berbeda 1 ruangan dengan penyerang. Besarnya nilai *false negative* ini dikarenakan adanya interferensi yang disebabkan oleh adanya halangan berupa dinding yang menghalangi penyebaran *frame* de-otentikasi. Semakin banyak dinding pembatas, semakin banyak *frame* yang hilang sehingga *alert* yang dihasilkan sistem juga semakin sedikit.

4.2.2 *Response Time*

Penghitungan response time akan menggunakan serangan *de-authentication flood*.

4.2.2.1 Serangan *De-authentication flood*

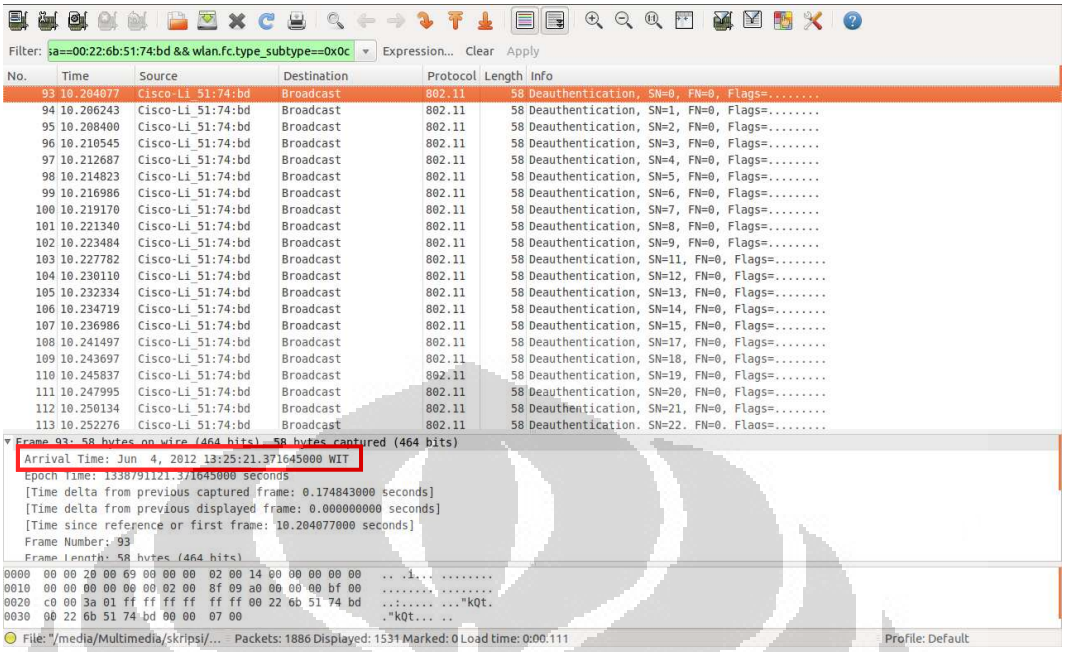
Serangan ini merupakan serangan yang mudah diukur waktu maupun jumlah *frame* dan *alert*-nya. Oleh karena itu, serangan ini dipilih untuk mengukur performansi sistem WIDS dengan menggunakan *response time*-nya. Data-data yang dianalisis diambil dari data serangan *de-authentication flood* pada *functionality test*. Pada pengukuran *response time* ini, akan dibandingkan performansi WIDS dalam mendeteksi serangan *de-authentication flood* dengan parameter banyaknya serangan dan peletakan sensor.

1. Banyaknya Serangan

   Response time akan dihitung dengan menambah serangan sampai 3 serangan secara bertahap.

   a. Satu serangan

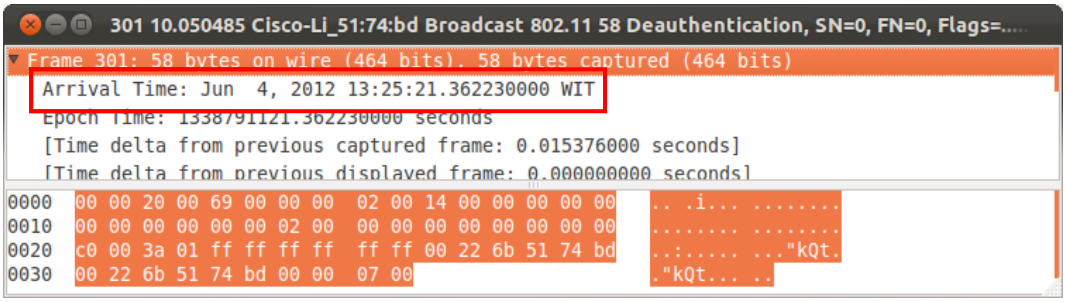
      Penghitungan *response time* ini didapat dengan menghitung perbedaan waktu dari serangan dikirim sampai serangan terdeteksi oleh Kismet di mana serangan hanya dilakukan oleh penyerang1. Untuk melihat waktu kapan serangan terdeteksi dapat dilihat pada *file* pcap yang dihasilkan Kismet. Hasil *capture* Wireshark dari sisi *server* pada pengujian pertama diperlihatkan pada Gambar 4.24.



Gambar 4.24. Waktu awal deteksi serangan untuk 1 serangan

Dapat dilihat bahwa Kismet pertama kali mendeteksi *frame* yang dikirimkan oleh penyerang pada *Sequence Number* (SN) 0 dengan jam 13:25:21.371645000. Hal ini berarti bahwa Kismet mendeteksi SN awal yang dikirim oleh penyerang. Setiap *frame* mempunyai *Sequence Number* yang unik (tidak sama), dimana nilainya akan bertambah satu ketika sebuah *node* mengirimkan *frame* yang sama lagi. Nilai maksimal SN adalah 4096. Setelah mencapai nilai maksimal, SN akan kembali lagi mulai dari 0. [23]

Untuk mengetahui *response time* ini, perlu diketahui kapan serangan dengan SN yang sama dikirimkan oleh penyerang. Pada sisi penyerang juga dijalankan Kismet untuk mengetahui kapan penyerang melakukan serangan. Gambar 4.25. memperlihatkan *capture* Wireshark untuk SN yang sama dari sisi penyerang.



Gambar 4.25. Waktu awal serangan diluncurkan untuk 1 serangan

Penyerang mengirim *frame* de-otentikasi dengan SN 0 pada jam 13:25:21.362230000. Perbedaan waktu dari dikirim sampai terdeteksi Kismet dihitung dari pengurangan waktu pendeteksian dengan waktu pengiriman, yaitu sebesar 0,009415000 detik.

Pengujian dilakukan pengulangan sebanyak 10 kali. Tabel 4.11. memperlihatkan hasil *response time* pada masing-masing pengujian.

Tabel 4.11. *Response time* untuk 10 pengujian pada 1 serangan

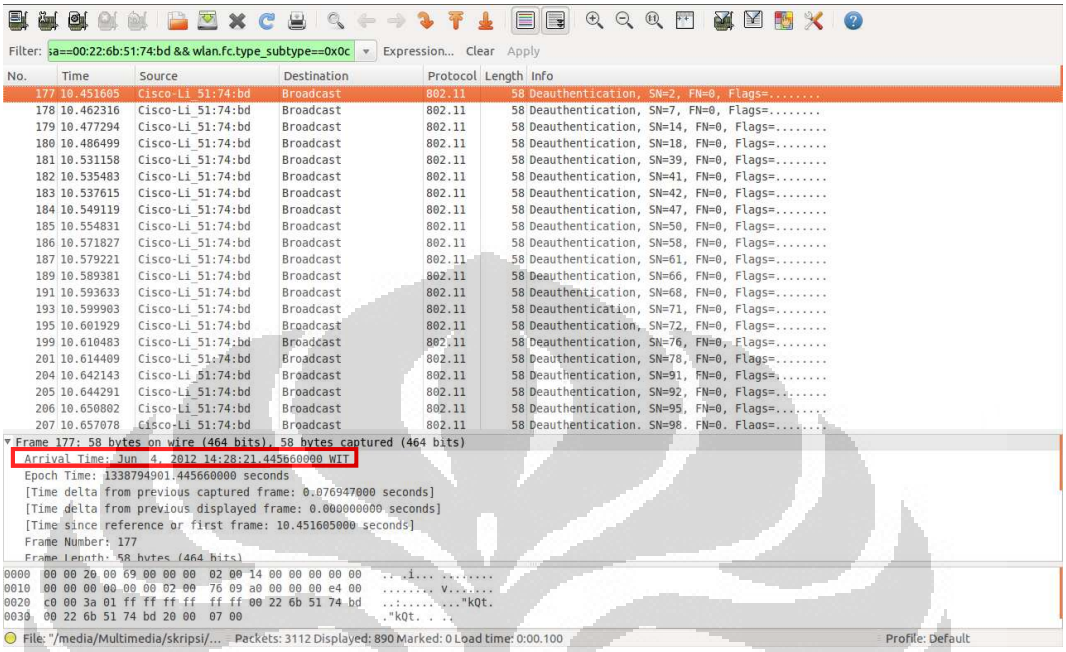
Pengujian ke-	*Response Time* (detik)
1	0,009415000
2	0,012609000
3	0,013209000
4	0,013834000
5	0,014494000
6	0,015043000
7	0,015733000
8	0,016339000
9	0,016948000
10	0,017665000
Rata-rata	0,014528900

Rata-rata *response time* untuk 1 serangan adalah sebesar 0,014528900 detik.

b. Dua serangan

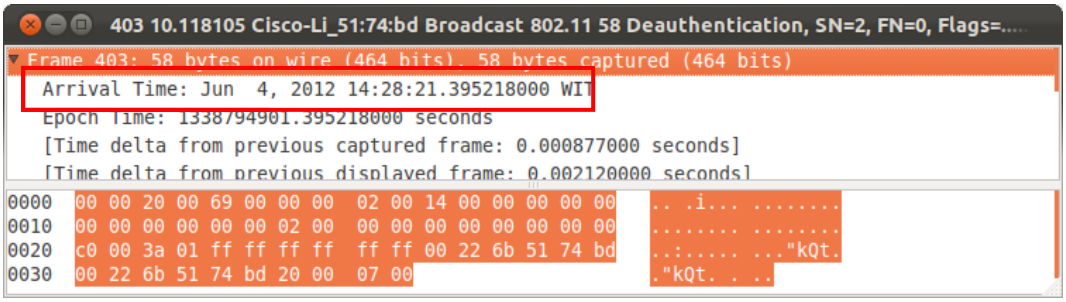
Penghitungan *response time* ini didapat dengan menghitung perbedaan waktu dari serangan dikirim sampai serangan terdeteksi oleh Kismet di mana serangan dilakukan oleh penyerang1 dan penyerang2 secara bersamaan. Pada skenario ini, *response time* yang akan dihitung adalah

untuk penyerang1 saja. Hasil *capture* Wireshark dari sisi *server* pada pengujian pertama diperlihatkan pada Gambar 4.26.



Gambar 4.26. Waktu awal deteksi serangan untuk 2 serangan

Dapat dilihat bahwa Kismet pertama kali mendeteksi *frame* yang dikirimkan oleh penyerang pada *Sequence Number* (SN) 2 dengan jam 14:28:21.445660000. Hal ini berarti bahwa Kismet tidak mendeteksi *frame* dengan SN 0 dan 1. Untuk mengetahui *response time* ini, perlu diketahui kapan serangan dengan SN yang sama dikirimkan oleh penyerang. Pada sisi penyerang juga dijalankan Kismet untuk mengetahui kapan penyerang melakukan serangan. Gambar 4.27. merupakan *capture* Wireshark untuk SN yang sama dari sisi penyerang.



Gambar 4.27. Waktu awal serangan diluncurkan untuk 2 serangan

Penyerang mengirim *frame* de-otentikasi dengan SN 2 pada jam 14:28:21.395218000. Perbedaan waktu dari dikirim sampai terdeteksi Kismet dihitung dari pengurangan waktu pendeteksian dengan waktu pengiriman, yaitu sebesar 0,050442000 detik. Tabel 4.12. memperlihatkan hasil *response time* pada 10 kali pengujian.

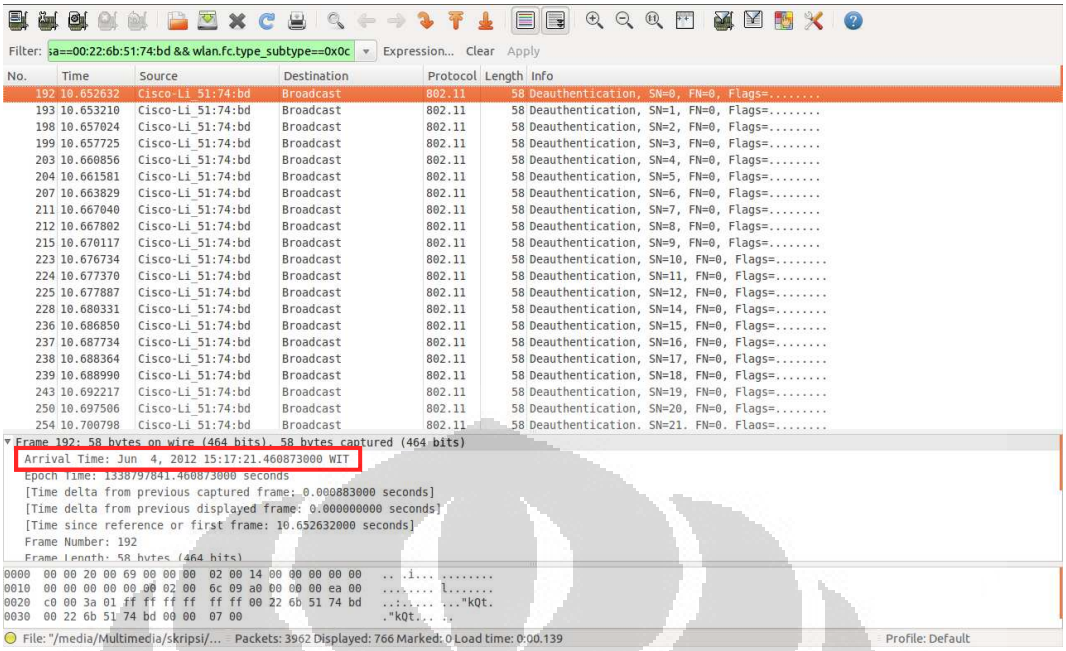
Tabel 4.12. *Response time* untuk 10 pengujian pada 2 serangan

Pengujian ke-	***Response Time* (detik)**
1	0,050442000
2	0,053817000
3	0,054469000
4	0,055137000
5	0,055759000
6	0,056430000
7	0,057031000
8	0,057748000
9	0,058493000
10	0,059129000
Rata-rata	0,055845500

Rata-rata *response time* untuk penyerang1 di mana dilakukan serangan oleh penyerang1 dan penyerang2 secara bersamaan adalah sebesar 0,055845500 detik.

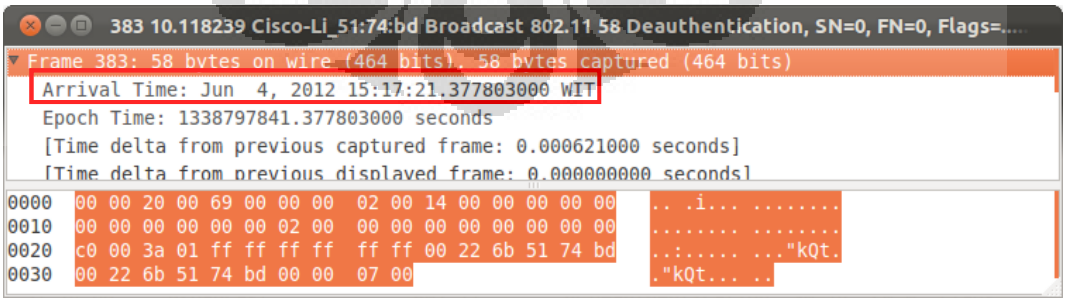
c. Tiga serangan

Penghitungan *response time* ini didapat dengan menghitung perbedaan waktu dari serangan dikirim sampai serangan terdeteksi oleh Kismet di mana serangan dilakukan oleh penyerang1, penyerang2, dan penyerang3 secara bersamaan. Pada skenario ini, *response time* yang akan dihitung adalah untuk penyerang1 saja. Hasil *capture* Wireshark dari sisi *server* pada pengujian pertama diperlihatkan pada Gambar 4.28.



Gambar 4.28. Waktu awal deteksi serangan untuk 3 serangan

Dapat dilihat bahwa Kismet pertama kali mendeteksi *frame* yang dikirimkan oleh penyerang pada *Sequence Number* (SN) 0 dengan jam 15:17:21.460873000. Hal ini berarti bahwa Kismet mendeteksi SN awal penyerang melakukan serangan. Untuk mengetahui *response time* ini, perlu diketahui kapan serangan dengan SN yang sama dikirimkan oleh penyerang. Pada sisi penyerang juga dijalankan Kismet untuk mengetahui kapan penyerang melakukan serangan. Gambar 4.29. merupakan *capture* Wireshark untuk SN yang sama dari sisi penyerang.



Gambar 4.29. Waktu awal serangan diluncurkan untuk 3 serangan

Penyerang mengirim *frame* de-otentikasi dengan SN 0 pada jam 15:17:21.377803000. Perbedaan waktu dari dikirim sampai terdeteksi

Kismet dihitung dari pengurangan waktu pendeteksian dengan waktu pengiriman, yaitu sebesar 0,083070000 detik.

Pengujian dilakukan pengulangan sebanyak 10 kali. Tabel 4.13. memperlihatka hasil *response time* pada masing-masing pengujian.

Tabel 4.13. *Response time* untuk 10 pengujian pada 3 serangan

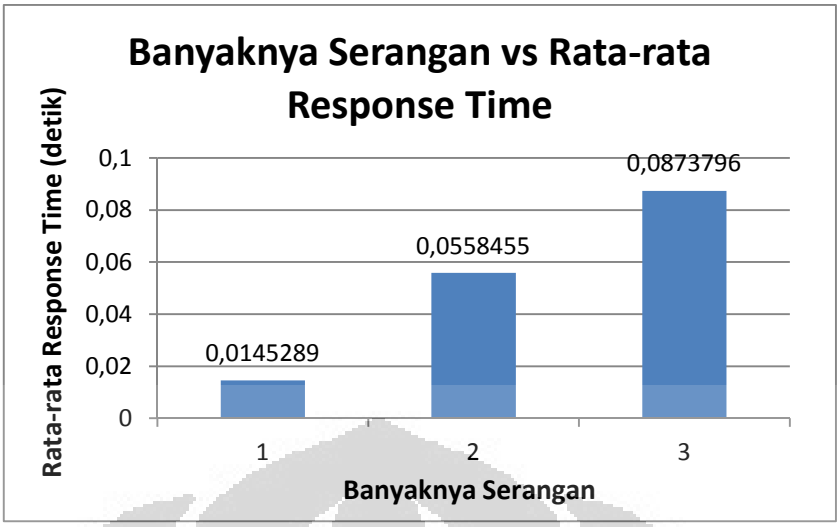
Pengujian ke-	***Response Time* (detik)**
1	0,083070000
2	0,083761000
3	0,084322000
4	0,085066000
5	0,085763000
6	0,086425000
7	0,087057000
8	0,087663000
9	0,088378000
10	0,102291000
Rata-rata	0,087379600

Rata-rata *response time* untuk penyerang1 di mana dilakukan serangan oleh penyerang1, penyerang2, dan penyerang3 secara bersamaan adalah sebesar 0,087379600 detik.

Untuk mengetahui perbedaan rata-rata *response time* terhadap banyaknya serangan, dapat dilihat pada Tabel 4.14 dan Gambar 4.30. yang menunjukkan tabel dan grafik rata-rata *response time* pada 1, 2, dan 3 serangan.

Tabel 4.14. Rata-rata *response time* untuk banyaknya serangan

Banyaknya Serangan	**Rata-rata *Response Time* (detik)**
1	0,014528900
2	0,055845500
3	0,087379600



Gambar 4.30. Grafik *response time* untuk banyaknya serangan

Dari hasil pengujian dapat dilihat bahwa dengan bertambahnya serangan akan berpengaruh terhadap performa sistem WIDS ini. Hal ini ditunjukkan pada Tabek 4.14. dan Gambar 4.30. dengan semakin lamanya *response time* yang dihasilkan oleh sistem WIDS tersebut. Hal ini menunjukkan bahwa banyaknya trafik akan mempengaruhi kinerja WIDS. Terdapat *delay* yang menunjukkan WIDS terlambat dalam mendeteksi serangan yang disebabkan oleh banyaknya *frame* yang harus diproses oleh WIDS terlebih dahulu. Selain itu SN yang terdeteksi tidak selalu mulai dari SN 0 karena adanya *channel hopping* atau dapat disebabkan juga oleh *frame* tidak sampai ke sensor. *Channel hopping* dapat menyebabkan sensor tidak mendeteksi *frame* dari penyerang di waktu-waktu tertentu.

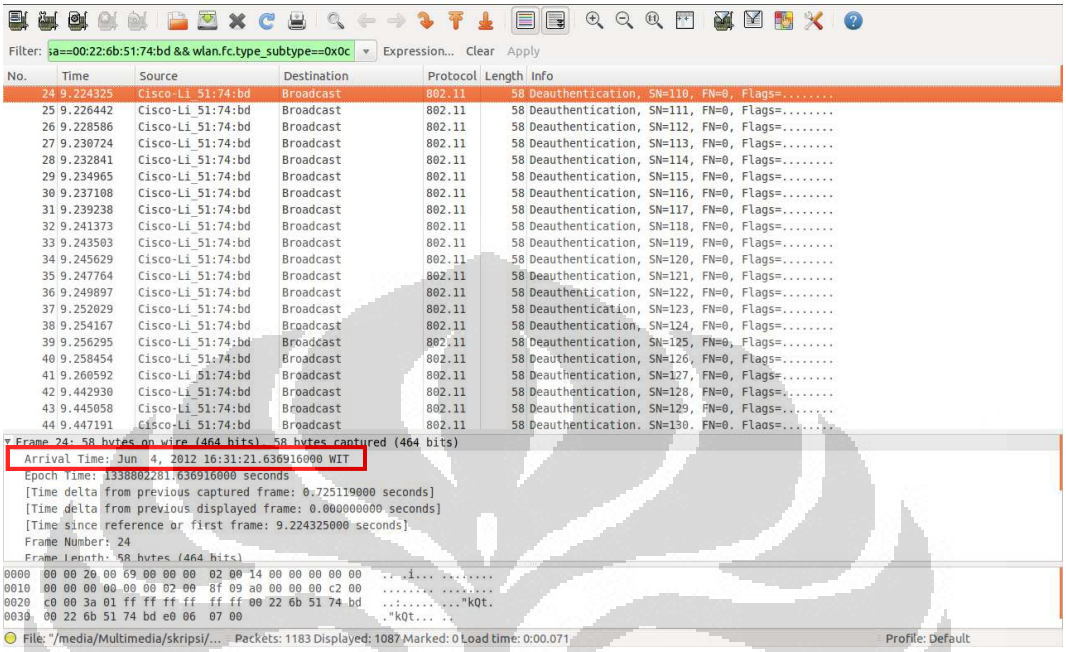
2. Peletakan Sensor

   Penghitungan *response time* yang pertama dilakukan dengan mengubah posisi sensor terhadap penyerang.

   a. Ruang 1

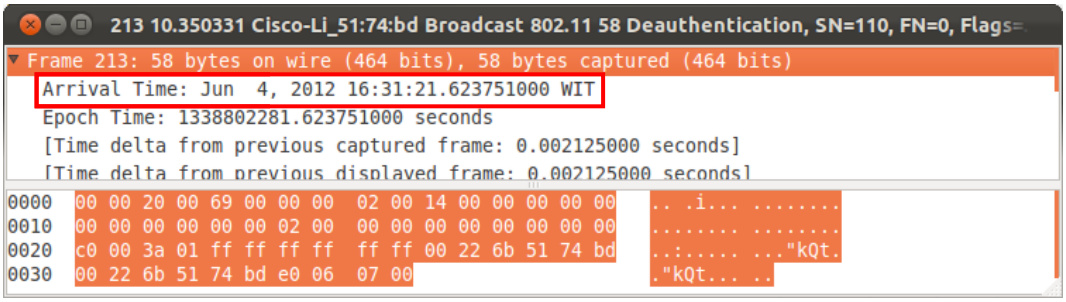
      Penghitungan *response time* ini didapat dengan menghitung perbedaan waktu dari serangan dikirim sampai serangan terdeteksi oleh Kismet di mana sensor berada pada ruangan yang sama dengan penyerang. *Response time* akan dihitung dari *frame* de-otentikasi yang masuk. Kismet menghasilkan *file* pcap yang dapat dibaca oleh Wireshark. Gambar 4.31.

memperlihatkan tampilan Wireshark untuk pengujian pertama di sisi *server*.



Gambar 4.31. Waktu awal deteksi serangan untuk ruang 1

Dapat dilihat bahwa Kismet pertama kali mendeteksi *frame* yang dikirimkan oleh penyerang pada *Sequence Number* (SN) 110 dengan jam 16:31:21.636916000. Hal ini berarti bahwa Kismet tidak mendeteksi SN 0 sampai 109. Untuk mengetahui *response time* ini, perlu diketahui kapan serangan dengan SN yang sama dikirimkan oleh penyerang. Pada sisi penyerang juga dijalankan Kismet untuk mengetahui kapan penyerang melakukan serangan. Gambar 4.32. merupakan *capture* Wireshark untuk SN yang sama dari sisi penyerang.



Gambar 4.32. Waktu awal serangan diluncurkan untuk ruang 1

Penyerang mengirim *frame* de-otentikasi dengan SN 110 pada jam 16:31:21.623751000. Perbedaan waktu dari dikirim sampai terdeteksi Kismet dihitung dari pengurangan waktu pendeteksian dengan waktu pengiriman, yaitu sebesar 0,013165000 detik.

Pengujian dilakukan pengulangan sebanyak 10 kali. Tabel 4.15. memperlihatkan *response time* untuk 10 kali pengujian.

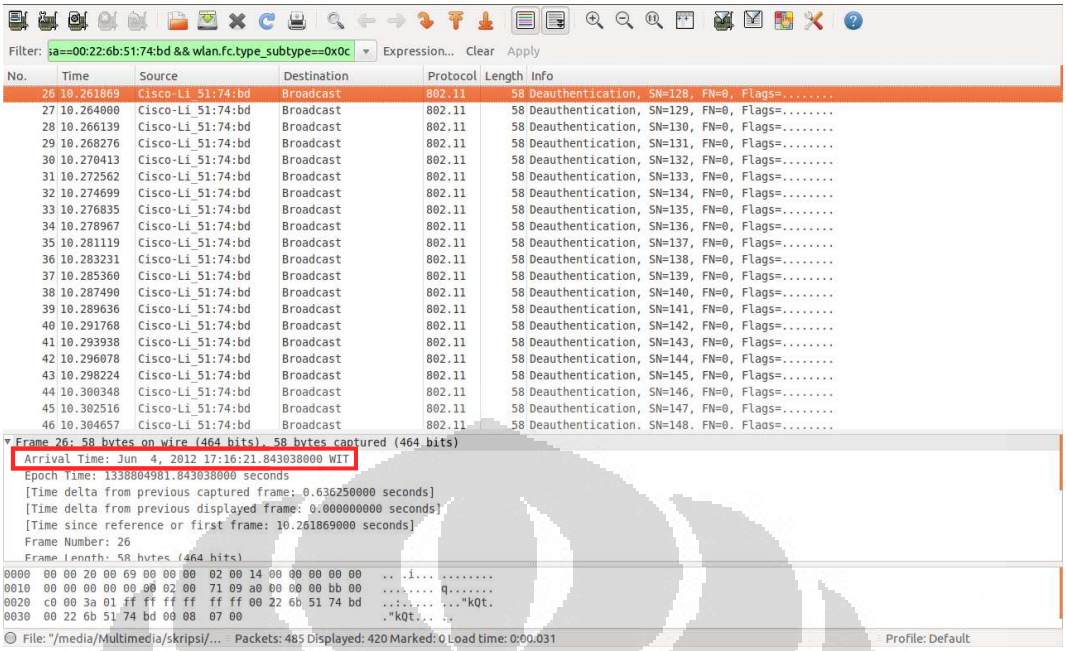
Tabel 4.15. *Response time* untuk 10 pengujian pada ruang 1

Pengujian ke-	***Response Time* (detik)**
1	0,013165000
2	0,015624000
3	0,016189000
4	0,016787000
5	0,017505000
6	0,018123000
7	0,018655000
8	0,019376000
9	0,019923000
10	0,020535000
Rata-rata	0,017588200

Rata-rata *response time* pada ruang 1 di mana peletakan sensor berada pada 1 ruangan yang sama dengan penyerang adalah sebesar 0,017588200 detik.

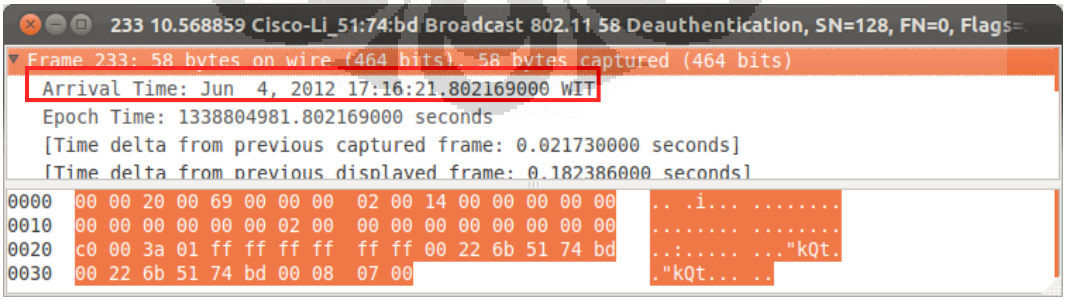
b. Ruang 2

Penghitungan *response time* ini didapat dengan menghitung perbedaan waktu dari serangan dikirim sampai serangan terdeteksi oleh Kismet di mana sensor berada pada ruangan yang berbeda 1 ruangan dengan penyerang. Untuk mengetahui kapan serangan terdeteksi, dapat dilihat pada Wireshark. Gambar 4.33. memperlihatkan tampilan Wireshark untuk pengujian pertama di sisi *server*.



Gambar 4.33. Waktu awal deteksi serangan untuk ruang 2

Dapat dilihat bahwa Kismet pertama kali mendeteksi *frame* yang dikirimkan oleh penyerang pada *Sequence Number* (SN) 128 dengan jam 17:16:21.843038000. Hal ini berarti bahwa Kismet tidak mendeteksi SN 0 sampai 127. Untuk mengetahui *response time* ini, perlu diketahui kapan serangan dengan SN yang sama yang dikirimkan oleh penyerang. Pada sisi penyerang juga dijalankan Kismet untuk mengetahui kapan penyerang melakukan serangan. Gambar 4.34. memperlihatkan *capture* Wireshark untuk SN yang sama dari sisi penyerang.



Gambar 4.34. Waktu awal serangan diluncurkan untuk ruang 2

Penyerang mengirim *frame* de-otentikasi dengan SN 128 pada jam 17:16:21.802169000. Perbedaan waktu dari dikirim sampai terdeteksi

Kismet dihitung dari pengurangan waktu pendeteksian dengan waktu pengiriman, yaitu sebesar 0,040869000 detik.

Pengujian dilakukan pengulangan sebanyak 10 kali. Hasil *response time* pada masing-masing pengujian dapat dilihat pada Tabel 4.16.

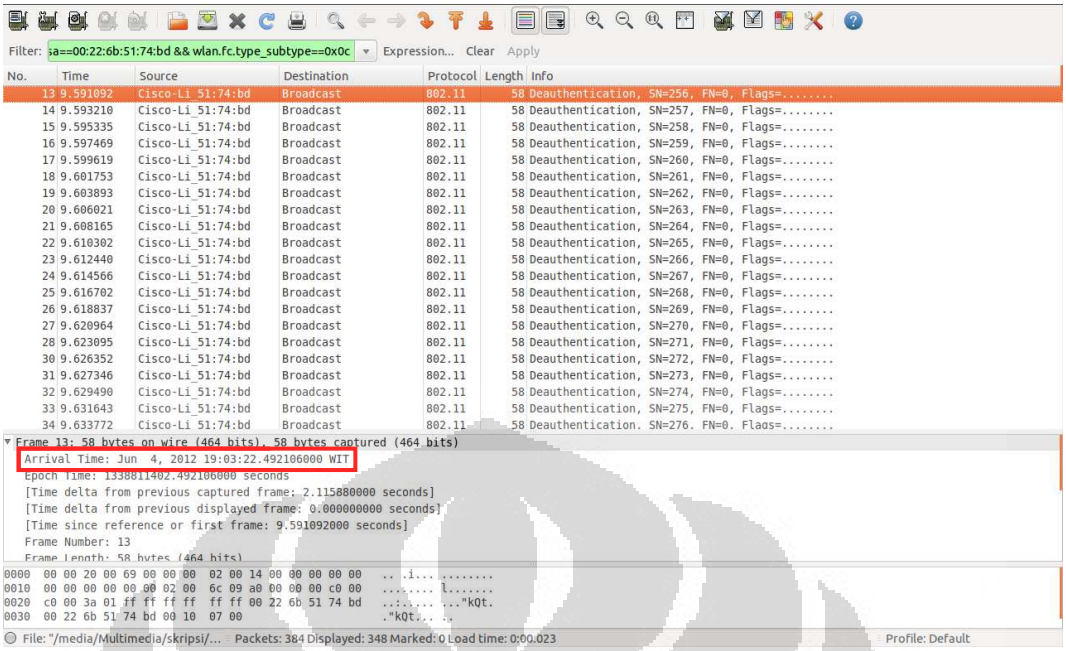
Tabel 4.16. *Response time* untuk 10 pengujian pada ruang 2

Pengujian ke-	***Response Time* (detik)**
1	0,040869000
2	0,043888000
3	0,044569000
4	0,045080000
5	0,045745000
6	0,046326000
7	0,046883000
8	0,047495000
9	0,048073000
10	0,048700000
Rata-rata	0,045762800

Rata-rata *response time* pada ruang 2 di mana peletakan sensor berada pada 1 ruangan yang berbeda dengan penyerang adalah sebesar 0,045762800 detik.

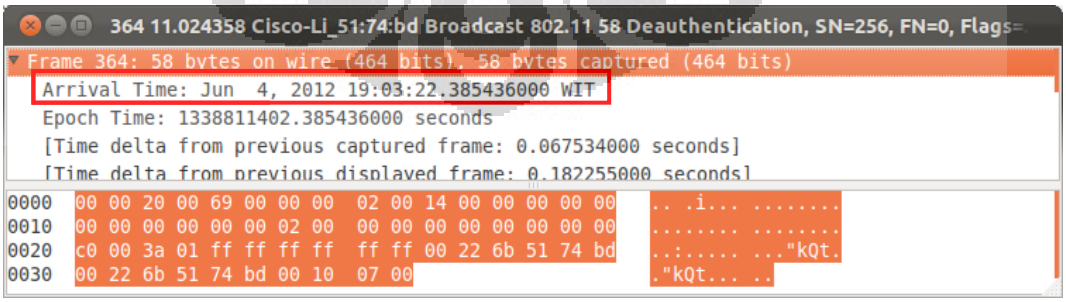
c. Ruang 3

Penghitungan *response time* ini didapat dengan menghitung perbedaan waktu dari serangan dikirim sampai serangan terdeteksi oleh Kismet di mana sensor berada pada ruangan yang berbeda 2 ruangan dengan penyerang. Kismet menghasilkan *file* pcap yang dapat dibaca oleh Wireshark. Hasil *capture* Wireshark dari sisi *server* untuk pengujian pertama diperlihatkan pada Gambar 4.35.



Gambar 4.35. Waktu awal deteksi serangan untuk ruang 3

Dapat dilihat bahwa Kismet pertama kali mendeteksi *frame* yang dikirimkan oleh penyerang pada *Sequence Number* (SN) 256 dengan jam 19:03:22.492106000. Hal ini berarti bahwa Kismet tidak mendeteksi SN 0 sampai 255. Untuk mengetahui *response time* ini, perlu diketahui kapan serangan dengan SN yang sama yang dikirimkan oleh penyerang. Pada sisi penyerang juga dijalankan Kismet untuk mengetahui kapan penyerang melakukan serangan. Gambar 4.36. memperlihatkan *capture* Wireshark untuk SN yang sama dari sisi penyerang.



Gambar 4.36. Waktu awal serangan diluncurkan untuk ruang 3

Penyerang mengirim *frame* de-otentikasi dengan SN 256 pada jam 19:03:22.385436000. Perbedaan waktu dari dikirim sampai terdeteksi

Kismet dihitung dari pengurangan waktu pendeteksian dengan waktu pengiriman, yaitu sebesar 0,106670000 detik.

Pengujian dilakukan pengulangan sebanyak 10 kali. Hasil *response time* pada masing-masing pengujian diperlihatkan pada Tabel 4.17.

Tabel 4.17. *Response time* untuk 10 pengujian pada ruang 3

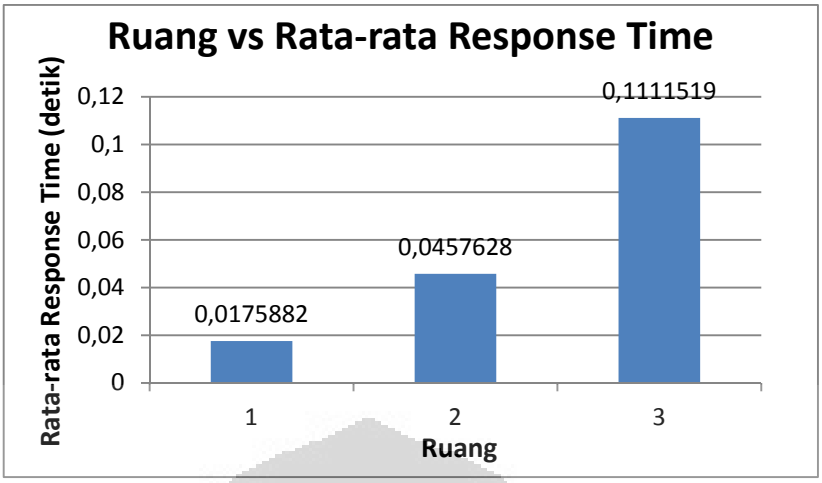
Pengujian ke-	***Response Time* (detik)**
1	0,106670000
2	0,109161000
3	0,109787000
4	0,110367000
5	0,111063000
6	0,111677000
7	0,112267000
8	0,112933000
9	0,113467000
10	0,114127000
Rata-rata	0,111151900

Rata-rata *response time* pada ruang 3 di mana peletakan sensor berada pada 2 ruangan yang berbeda dengan penyerang adalah sebesar 0,111151900 detik.

Untuk mengetahui perbedaan *response time* terhadap peletakan sensor, dapat dilihat pada Tabel 4.18. dan Gambar 4.37. yang menunjukkan dan grafik rata-rata *response time* pada ruang 1, ruang 2, dan ruang 3.

Tabel 4.18. Rata-rata *response time* untuk peletakan sensor

Ruang	**Rata-rata *Response Time* (detik)**
1	0,017588200
2	0,045762800
3	0,111151900



Gambar 4.37. Grafik rata-rata *response time* untuk peletakan sensor

Dari Tabel 4.18. dan Gambar 4.37. dapat dilihat bahwa dengan bertambahnya perbedaan banyaknya ruangan yang membatasi penyerang dengan sensor akan berpengaruh terhadap kinerja sistem WIDS ini. Hal ini ditunjukkan dengan semakin lamanya *response time* yang dihasilkan oleh sistem WIDS tersebut. Hal ini menunjukkan peletakan sensor sangat berpengaruh pada pendeteksian serangan pada WIDS. Hal ini dapat disebabkan karena adanya *delay* yang menyebabkan WIDS terlambat dalam mendeteksi serangan yang disebabkan oleh jauhnya letak sensor dengan penyerang. Selain itu, pada pengujian pertama pada masing-masing ruangan terlihat bahwa sensor tidak langsung mendeteksi SN 0 yang merupakan SN awal dari serangan. Hal ini menyebabkan *alert* pertama yang dihasilkan Kismet menjadi lebih lambat. Nilai SN yang terdeteksi ini juga dapat disebabkan oleh adanya *channel hopping* yang ketika penyerang sedang melakukan serangan, sensor sedang memonitor *channel* lain selain *channel* 1 atau *frame* tidak sampai ke sensor.

BAB 5
KESIMPULAN

Dari pembahasan hasil dan analisis performansi sistem WIDS yang sudah dipaparkan sebelumnya, maka dapat diambil kesimpulan:

1. Pada *functionality test*, WIDS mampu mendeteksi adanya serangan atau anomali baik yang berupa *spoofed* AP, *brute force* WPS, maupun *de-authentication flood*. Kismet menghasilkan *alert* berdasarkan ciri-ciri *frame* yang terdeteksi pada sensor. Untuk *spoofed* AP, Kismet mendeteksinya dengan membandingkan nilai *timestamp* dari *beacon frame* yang dikirimkan oleh AP yang sah dengan *spoofed* AP. Untuk *brute force* WPS, Kismet mendeteksinya dengan *message* M3 dari proses WPS. Untuk serangan *de-authentication flood*, Kismet mendeteksinya dengan *frame* de-otentikasi yang dikirimkan secara *broadcast*.
2. Serangan *de-authentication flood* digunakan untuk skenario banyaknya serangan. Analisis akan dilihat dari penghitungan serangan dari penyerang1. Rata-rata *alert* pada 1, 2, dan 3 serangan berturut-turut semakin menurun, yaitu 12 *alert*, 3,8 *alert*, dan 2,3 *alert*. Persentase *false negative frame* de-otentikasi yang mengacu kepada 1 serangan semakin banyak. Persentase *false negative* penyerang1 pada 2 serangan adalah 28,43% dan pada 3 serangan adalah 44,47%. *Response time* berturut-turut juga semakin lambat, yaitu 0,015 detik, 0,056 detik, dan 0,087 detik. Hal ini menunjukkan padatnya trafik sangat berpengaruh pada pendeteksian serangan pada WIDS.
3. Serangan *de-authentication flood* digunakan untuk scenario peletakan sensor terhadap penyerang. Rata-rata *alert* untuk peletakan sensor pada ruang yang sama dengan penyerang (ruang 1), ruang yang berbeda 1 ruangan dari penyerang (ruang 2), dan ruang yang berbeda 2 ruangan dari penyerang (ruang 3) berturut-turut semakin sedikit, yaitu 10,6 *alert*, 7,9 *alert*, dan 7,8 *alert*. Persentase *false negative frame* de-otentikasi yang mengacu kepada *frame* de-otentikasi yang terdeteksi pada ruang 1 semakin banyak. Untuk ruang 2 adalah 72,48% dan ruang 3 adalah 77,17%. Rata-rata *response time* berturut-turut semakin lambat, yaitu 0,018 detik, 0,046 detik, dan 0,111 detik.

Hal ini menunjukkan peletakan sensor sangat berpengaruh pada pendeteksian serangan pada WIDS.

4. WIDS dapat bekerja optimal jika berada dalam 1 ruangan dengan AP yang ingin dimonitor dan tidak terlalu banyak trafik. Hal ini untuk menghindari adanya interferensi dan terlalu banyaknya *frame* yang lalu lalang di udara.

DAFTAR REFERENSI

[1] "Wireless LAN Networking". (2011). U.S.Robotics. [Online]. Tersedia: http://www.usr.com/download/whitepapers/wireless-wp.pdf. Diakses tanggal 4 Januari 2012.

[2] S'to. *Wireless Kung Fu.* Jakarta: Jasakom. 2009.

[3] Mulyanta, E. S. *Pengenalan Protokol Jaringan Komputer.* Yogyakarta: Andi. 2003.

[4] Prabhaker Mateti, "Hacking Techniques in Wireless Networks," *http://www.cs.wright.edu/~pmateti/InternetSecurity/Lectures/WirelessHacks/Mateti-WirelessHacks.htm*, 2005. Diakses tanggal 17 Mei 2012.

[5] "Wi-Fi Protected Setup," *http://www.Wi-Fi.org/knowledge-center/articles/Wi-Fi-protected-setup%E2%84%A2*, 2012. Diakses tanggal 17 Mei 2012.

[6] Kamel Messaoudi, "Wi-Fi Protected Setup (WPS)," *http://briolidz.wordpress.com/2012/01/10/Wi-Fi-protected-setup-wps/*, 10 Jan 2012. Diakses tanggal 10 Mei 2012.

[7] Wright, J. (2003, Jan.). Detecting Wireless LAN MAC Address Spoofing. Tersedia: http://www.rootsecure.net/content/downloads/pdf/wlan_macspooof_detection.pdf. Diakses tanggal 10 Mei 2012.

[8] Armitage, S. (2011, Okt.). Known Wireless Attacks. JA. [Online]. Tersedia: http://www.ja.net/documents/services/wtas/known-wireless-attacks.pdf. Diakses tanggal 17 Mei 2012.

[9] Makanju, A., Patrick LaRoche, A., & Zincir-Heywood, N, "A Comparison Between Signature and GP-Based IDSs for Link Layer Attacks on WiFi Networks," di *Proc. 2007 IEEE Symposium on Computational*, Honolulu: IEEE. 2007, hal. 213-219.

[10] Murray, J. (2009, Apr.). An Inexpensive Wireless IDS using Kismet and OpenWRT. SANS. [Online]. Tersedia:

http://www.sans.org/reading_room/whitepapers/detection/inexpensive-wireless-ids-kismet-openwrt_33103. diakses tanggal 10 Desember 2011.

[11] Viehbock, S. (2011, Des.). Brute Forcing Wi-Fi Protected Setup. [Online]. Tersedia: http://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf. Diakses tanggal 17 Mei 2012.

[12] Deckerd, G. (2006, Nov.). Wireless Attack from an Intrusion Detection Perspective. SANS. [Online] Tersedia: http://www.sans.org/reading_room/whitepapers/honors/wireless-attacks-intrusion-detection-perspective_1681. Diakses tanggal 10 Desember 2011.

[13] Scarfone, K., & Mell, P. (2007, Feb.). Guide to Intrusion Detection and Prevention System (IDPS). National Institute of Standard and Technology. [Online]. Tersedia: http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf. Diakses tanggal 10 Desember 2011.

[14] Mike Kershaw, "Kismet 2011-03-R2," *https://www.kismetwireless.net/code/svn/trunk/README*, 2011. Diakses tanggal 16 Mei 2012.

[15] "Kismet server/Drone," *http://www.dd-wrt.com/wiki/index.php/Kismet_Server/Drone*, 20 Apr 2010. Diakses tanggal 17 November 2011.

[16] "Defeating Evil twin Attacks," *http://searchsecurity.techtarget.com/feature/Defeating-Evil-Twin-attacks*, 26 Jun 2009. Diakses tanggal 20 November 2011.

[17] "A Wireless Intrusion Detection System Based on Open-Source Software". (2008, Feb.). ELCA. [Online]. Tersedia: http://www.elca.ch/sites/elca.ch/files/resources/downloads/ELCA_white_paper_final_new.pdf. Diakses tanggal 17 November 2011.

[18] "Quadrant InfoSec's: Sagan," *http://sagan.softwink.com/*, 2012. Diakses tanggal 4 April 2012.

[19] "MySQL Commercial License," *http://www.mysql.com/about/legal/licensing//commercial-license.html*, 2012. Diakses tanggal 1 Mei 2012.

[20] "Rails 3.2.3," *http://rubygems.org/gems/rails*, 2012. Diakses tanggal 1 Mei 2012.

[21] "Snorby," *http://snorby.org/*, 2012. Diakses tanggal 1 Mei 2012.

[22] Mike Kershaw, "Reaver/WPS Brute force IDS," *http://blog.kismetwireless.net/2012/01/Reaver-wps-brute-force-ids.html*, 9 Jan 2012. Diakses tanggal 22 April 2012.

[23] Guo, F., & Chiueh, T.-c, "Sequence Number-Based MAC Address Spoof Detection," di *RAID'05 Proc. 8th Int. Conf. on Recent Advances in Intrusion Detection*, Berlin: Heidelberg, 2006, hal. 309-329.

LAMPIRAN

Lampiran 1. Instalasi

Sebagaimana yang disebutkan dalam BAB 3, berikut ini adalah langkah-langkah secara detil bagaimana menginstal dan mengkonfigurasi *tools* yang digunakan dalam skripsi ini.

1. Kismet *drone*
   a. Ganti *firmware* asli dengan OpenWRT, dengan cara memasukkan *file* openwrt-ar71xx-generic-tl-wr741nd-v2-squashfs-factory.bin ke halaman *firmware upgrade*.
   b. Setelah *firmware* diganti, IP *address default* dari OpenWRT adalah 192.168.1.1 dan pada saat pertama kali *booting*, OpenWRT dapat diakses dengan *telnet*. Dari terminal, ketik:
      ```
      rika@server:# telnet root@192.168.1.1
      ```
   c. Setelah berhasil masuk, akan ditampilkan sebagai berikut:
      ```
      BusyBox v1.19.3 (2012-02-18 23:34:43 MST) built-in shell (ash)
      Enter 'help' for a list of built-in commands.

        _______                     ________        __
       |       |.-----.-----.-----.|  |  |  |.----.|  |_
       |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
       |_______||   __|_____|__|__||________||__|  |____|
                |__| W I R E L E S S   F R E E D O M
      ATTITUDE ADJUSTMENT (bleeding edge, r30639) ----------
       * 1/4 oz Vodka      Pour all ingredients into mixing
       * 1/4 oz Gin        tin with ice, strain into glass.
       * 1/4 oz Amaretto
       * 1/4 oz Triple sec
       * 1/4 oz Peach schnapps
       * 1/4 oz Sour mix
       * 1 splash Cranberry juice
      -----------------------------------------------------
      ```
   d. Konfigurasi jaringan OpenWRT terlebih dahulu agar dapat terhubung ke internet. Konfigurasi tergantung pada ISP dan jaringan masing-masing. Pada pengujian ini, penulis menggunakan Speedy sehingga konfigurasi di /etc/config/network menjadi seperti berikut:
      Untuk LAN:
      ```
      option type 'bridge'
      option proto 'static'
      option ipaddr '192.168.102.1'
      option netmask '255.255.255.0'
      ```
      Untuk WAN:
      ```
      option proto 'dhcp'
      ```
   e. Setelah berhasil terhubung ke internet, *update* terlebih dahulu untuk dapat menginstal kismet *drone*.
      ```
      root@OpenWrt:~# opkg update
      ```
   f. Ganti *country code* menjadi Indonesia di /etc/config/wireless dengan menambahkan:
      ```
      option country ID
      ```
      dan mengganti *option mode* menjadi:
      ```
      option mode monitor
      ```
   g. Karena ada *bug* dalam paket OpenWRT untuk *firmware* ini, *file* regulatory.bin harus diganti.

```
root@OpenWrt:~# cd /usr/lib/crda/
root@OpenWrt:/usr/lib/crda~# rm regulatory.bin
root@OpenWrt:/usr/lib/crda~# wget
http://datatomb.de/mirror/stuff/regulatory.bin
root@OpenWrt:/usr/lib/crda~# reboot
```

h. Untuk mengetahui *channel* mana saja yang berfungsi, pada terminal ketik:
```
root@OpenWrt:~# iwlist wlan0 channel
```
i. Jika sudah ada *channel* 12 dan 13 berarti konfigurasi berhasil.
j. Di /etc/config/system ganti *hostname*, *timezone*, dan *list server* menjadi:
```
option hostname 'OpenWrt'
option timezone 'WIT-7'
list server 'asia.pool.ntp.org'
```
k. Setelah *update*, kita bisa menginstal program kismet *drone*
```
root@OpenWrt:~# opkg install kismet-drone
```
l. Setelah itu, mengkonfigurasi *file* kismet_drone.conf yang ada di /etc/kismet_drone.conf.
m. Ganti *dronelisten* menjadi IP *address* LAN yang terhubung ke PC. Pada konfigurasi ini, penulis menggunakan IP 192.168.102.1
```
dronelisten=tcp://192.168.102.1:2502
```
n. Mengkonfigurasi IP *address* berapa saja yang dapat mengakses OpenWRT
```
bindaddress=192.168.0.0/255.255.255.0
droneallowedhosts=192.168.0.0/255.255.255.0
```
o. Konfigurasi *interface* dimana Kismet akan menangkap data. Umumnya, *Wireless Router* menggunakan tipe mac80211.
```
ncsource=wlan0:type=mac80211
```
p. Jika tidak ingin mengaktifkan *channel hopping*, konfigurasinya adalah sebagai berikut:
```
ncsource=wlan0:name= mac80211,hop=false,channel=1
```
q. Aktifkan daftar *channel* yang 802.11b saja.
```
# Users outside the US might want to use this list:
channellist=IEEE80211b:1:3,7,13,2,8,3,14,9,4,10,5,11,6,12
# channellist=IEEE80211b:1:3,6:3,11:3,2,7,3,8,4,9,5,10
# US IEEE 80211a
#channellist=IEEE80211a:36,40,44,48,52,56,60,64,149,153,157,161,165
# Combo
#channellist=IEEE80211ab:1:3,6:3,11:3,2,7,3,8,4,9,5,10,36,40,44,48,52,56,
60,64,1$
```

2. Wireshark

   Pada ubuntu, Wireshark dapat diistal dengan *software management* ataupun lewat terminal. Jika dari terminal, penginstalan dapat dilakukan dengan mengetik:
```
root@server:# apt-get install Wireshark
```
3. Kismet *server*
   a. Instal Subversion
```
root@server:/home/rika# apt-get install subversion
```
   b. Instal Libncurses5 dan Libnlcurses5-dev
```
root@server:/home/rika# apt-get install libncurses libncurses5-dev
```
   c. Instal Libpcap0.8 dan Libpcap0.8-dev
```
root@server:/home/rika# apt-get install libpcap0.8 libpcap0.8-dev
```
   d. Instal Libnl-dev
```
root@server:/home/rika# apt-get install libnl libnl-dev
```
   e. Instal Libpcre3-dev
```
root@server:/home/rika# apt-get install install libpcre3 libpcre3-dev
```
   f. Unduh Kismet versi *development*
```
root@server:/home/rika# exit
rika@server:~$ svn co https://www.kismetwireless.net/code/svn/trunk
kismet-devel
```

g. Instal Kismet
```
rika@server:~$ cd kismet-devel
rika@server:~/kismet-devel$./configure
rika@server:~/kismet-devel$ make
rika@server:~/kismet-devel$ sudo su
root@server:/home/rika/kismet-devel# make install
```
h. Ubah *file* konfigurasi Kismet di /usr/local/etc/kismet.conf
```
ncsource=drone:host=192.168.102.1,port=3501
syslogtype=info,alert
```
i. Instal *plugin* Kismet
```
root@server:/home/rika/kismet-devel# make plugins
root@server:/home/rika/kismet-devel# make plugins-install
```
j. Dengan mengetikkan "`tail –f /var/log/syslog`", info dari syslog dapat dilihat secara *real-time* di terminal. Sebelumnya jalankan Kismet *server* dahulu dengan perintah kismet_server.

4. MySQL
   a. Untuk melakukan instalasi Mysql-*server*, karena Ubuntu sudah mempunyai paket ini di *repository*-nya, dari terminal jalankan perintah berikut:
   ```
   root@server:# apt-get install mysql-server
   ```
   b. Ketika proses instalasi sedang berjalan, *user* akan diminta memasukkan *password* untuk *user* 'root' untuk mengakses MySQL. Masukkan *password*: secret123
   c. Ketika proses instalasi selesai, *server* MySQL harus mulai secara otomatis. Jalankan perintah berikut untuk mengecek apakah *server* MySQL sedang berjalan:
   ```
   root@server:# sudo netstat -tap | grep mysql
   ```
   d. Dari terminal, jalankan perintah berikut untuk melakukan instalasi *library* untuk MySQL, yaitu Libmysqlclient-dev dan Libmysql++-dev:
   ```
   root@server:# apt get install libmysqlclient-dev libmysql++-dev
   ```
5. Apache2

   Untuk melakukan instalasi Apache2, karena Ubuntu sudah mempunyai paket ini di *repository*-nya, dari terminal jalankan perintah berikut:
   ```
   root@server:# apt-get install apache2
   ```
6. Snorby
   a. Instal beberapa prasyarat dari Snorby yang tersedia dari *repository* Ubuntu:
   ```
   root@server:# apt-get install git-core default-jre
   ```
   b. *Reboot server*
   ```
   root@server:# reboot
   ```
   c. Instal paket yang dibutuhkan:
   ```
   root@server:# apt-get install imagemagick libmagickwand-dev wkhtmltopdf
   root@server:# cd /tmp
   root@server:/tmp# wget http://wkhtmltopdf.googlecode.com/files/wkhtmltopdf-
   0.10.0_rc2-static-i386.tar.bz2
   root@server:/tmp# cp wkhtmltopdf /usr/bin
   root@server:# cd
   root@server:#  apt-get  install  gcc  g++  build-essential  libssl-dev
   libreadline-gplv2-dev  zlib1g-dev  linux-headers-generic  libsqlite3-dev
   libxslt1-dev libxml2-dev
   ```
   d. Unduh dan instal Ruby (1.9.2):
   ```
   root@server:# cd /usr/local/src/
   root@server:/usr/local/src# wget http://ftp.ruby-
   lang.org/pub/ruby/1.9/ruby-1.9.2-p290.tar.gz
   root@server:/usr/local/src# tar xvzf ruby-1.9.2-p290.tar.gz
   root@server:/usr/local/src# ln -s ruby-1.9.2-p290 ruby
   root@server:/usr/local/src# rm -rf ruby-1.9.2-p290.tar.gz
   root@server:/usr/local/src# chown root:root -R ruby-1.9.2-p290/
   root@server:/usr/local/src# cd ruby/
   ```

```
root@server:/usr/local/src/ruby# ./configure
root@server:/usr/local/src/ruby# make
root@server:/usr/local/src/ruby# make install
root@server:/usr/local/src/ruby# cd /usr/local/src/ruby/ext/openssl
root@server:/usr/local/src/ruby/ext/openssl# ruby extconf.rb
root@server:/usr/local/src/ruby/ext/openssl# make && make install
```

e. Jalankan "ruby – v" dan pastikan telah menginstal versi Ruby yang benar:
```
root@server:/usr/local/src/ruby/ext/openssl# cd /usr/local/src/ruby && ruby -v
  ruby 1.9.2p290 (2011-07-09 revision 32553) [i686-linux]
```

f. Instal dependensi (termasuk Rails) dengan gem:
```
root@server:/usr/local/src/ruby/ext/openssl# cd /usr/local/src/ruby
root@server:/usr/local/src/ruby/#  gem  install  thor  i18n  bundler  tzinfo
builder memcache-client erubis mail text-format sqlite3
root@server:/usr/local/src/ruby# gem install rack-test
root@server:/usr/local/src/ruby# gem install rack-mount
root@server:/usr/local/src/ruby# gem install rake --version=0.9.2
root@server:/usr/local/src/ruby# gem install rails --version=3.1.0
root@server:/usr/local/src/ruby# gem install rack –version=1.3.5
```

g. Instal Rubygems:
```
root@server:/usr/local/src/ruby# gem install rubygems-update
```

h. Ubah format data (baris 9) dari "2 s.date = %q{2011-08-25 00:00:00.000000000Z}" ke "2011-08-25" di *file* /usr/local/lib/ruby/gems/1.9.1/specifications/tilt-1.3.3.gemspec.

i. *Update* Rubygems:
```
root@server:/usr/local/src/ruby# update_rubygems
```

j. Unduh Snorby ke direktori /var/www
```
root@server:/usr/local/src/ruby# cd /var/www/
root@server:/var/www# git clone git://github.com/Snorby/snorby.git
root@server:/var/www# cd snorby
root@server:/var/www/snorby# bundle install
```

k. Ganti database_example.yml menjadi database.yml
```
root@server:/var/www/snorby# cd config
root@server:/var/www/snorby/config# mv database_example.yml database.yml
```

l. Ganti config_example.yml menjadi snorby_config.yml
```
root@server:/var/www/snorby/config# mv config_example.yml snorby_config.yml
```

m. Ubah informasi *database* dan ubah *password* untuk dapat mengakses Mysql di *file* /var/www/snorby/config/database.yml
Example: password: "s3cr3tsauce" menjadi password: "secret123"

n. Ubah *file* konfigurasi Snorby: /var/www/snorby/config/snorby_config.yml dan ubah *path script* Wkhtmktopdf.
```
root@server:/var/www/snorby/config# sed –i
s/"\/usr\/local\/bin\/wkhtmltopdf"/"\/usr\/bin\/wkhtmltopdf"/g
/var/www/snorby/config/snorby_config.yml
```

o. Instal semua dependensi yang dibutuhkan seperti yang telah dispesifikasikan di Gemfile:
```
root@server:/var/www/snorby/config# cd /usr/local/src/ruby
root@server:/usr/local/src/ruby# bundle install --deployment
```

p. Kemudian instal Snorby
```
root@server:/usr/local/src/ruby# cd /var/www/snorby
root@server:/var/www/snorby# rake snorby:setup
```

q. Agar dapat dijalankan pada *website browser*, digunakan Passenger sebagai penghubung Apache dan Snorby. Sebelumnya *tools dependency* Apache2-prefork-dev Libcurl4-openssl-dev harus diinstal terlebih dahulu.
```
root@server:# apt-get install apache2-prefork-dev libcurl4-openssl-dev
root@server:# chown www-data:www-data /var/www/snorby –R
```

r. Instal modul Passenger untuk Apache2
```
root@server:# cd /usr/local/src/ruby
root@server:/usr/local/src/ruby# gem install passenger
root@server:/usr/local/src/ruby# passenger-install-apache2-module
```

s. Tambahkan kalimat di bawah di bagian akhir *file* /etc/apache2/apache2.conf:

```
LoadModule passenger_module /usr/local/lib/ruby/gems/1.9.1/gems/passenger-3.0.9/ext/apache2/mod_passenger.so PassengerRoot /usr/local/lib/ruby/gems/1.9.1/passenger-3.0.9 PassengerRuby /usr/local/bin/ruby
```

t. Ubah *file* konfigurasi *default site* Apache2 di /etc/apache2/sites-enabled/000-default

`DocumentRoot /var/www` menjadi `DocumentRoot /var/www/snorby/publiC`

`<Directory "/var/www">` menjadi `<Directory "/var/www/snorby/public">`

u. *Restart* Apache2

```
root@server:# /etc/init.d/apache2 restart
```

v. Buka *browser* dan ketik "localhost" pada alamat *url*, maka akan muncul halaman *login*. Masukkan ***e-mail*** dan *password default* di bawah ini.

```
e-mail: snorby@snorby.org
password: snorby
```

7. Sagan

a. Unduh Sagan versi development

```
root@server:# git clone https://github.com/beave/sagan.git
```

b. Unduh *rules* Sagan dan tempatkan di /usr/local/etc

```
root@server:# git clone https://github.com/beave/sagan-rules.git
mv sagan-rules /usr/local/etc/
```

c. Instal Sagan

```
root@server:# cd sagan
root@server:/sagan# ./configure
root@server:/sagan# make
root@server:/sagan# make install
```

d. Buat *user* sagan dengan grup adm

```
root@server:/sagan# useradd –G adm sagan
```

e. Buat *file* fifo di /opt

```
root@server:/sagan# cd /opt
root@server:/opt# mkfifo sagan.fifo
```

f. Ubah *file* permission sagan.fifo menjadi *user* sagan grup adm

```
root@server:/opt# chown sagan:adm sagan.fifo
```

g. Buat folder sagan di /opt

```
root@server:/opt# mkdir sagan
```

h. Ubah *file* permission folder sagan menjadi *user* sagan grup adm

```
root@server:/opt# chown sagan:adm sagan
```

i. Buat folder sagan di /var/log

```
root@server:/opt# cd /var/log
root@server:/var/log# mkdir sagan
```

j. Buat *file* sagan.log dan *alert* di direktori /var/log/sagan

```
root@server:/var/log# cd sagan
root@server:/var/log/sagan# vi sagan.log
root@server:/var/log/sagan# vi alert
```

k. Ubah *file permission* sagan.log dan *alert* menjadi *user* sagan grup adm

```
root@server:/var/log/sagan# chown sagan:adm sagan.log
root@server:/var/log/sagan# chown sagan:adm alert
```

l. Ubah konfigurasi sagan.conf di /usr/local/etc/sagan.conf

`var FIFO /var/run/sagan.fifo` menjadi: `var FIFO /opt/sagan.fifo`

`var LOCKFILE /var/run/sagan/sagan.pid` menjadi: `var FIFO /opt/sagan/sagan.pid`

`sagan_host 192.168.0.1` menjadi: `sagan_host 192.168.102.100`

m. Aktifkan konfigurasi *database* dengan menghilangkan tanda ";" di /usr/local/etc/sagan.conf

```
sagan_hostname sagan
sagan_interface syslog
sagan_filter none
sagan_detail 1
```

```
output database: log, mysql, user=sagan password=secret dbname=snorrt_db
host=192.168.0.1
```

n. Ubah *output database* menjadi:

```
output database: log, mysql, user=sagan password=secret123 dbname=snorby
host=localhost
```

o. Aktifkan kismet.rules saja dengan memberikan tanda "#" ke *rules* selain kismet.rules

p. Agar sesuai dengan versi Kismet yang sekarang, ubah konfigurasi kismet.rules di /etc/sagan-rules/kismet.rules menjadi:

```
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Kismet
starting"; program: Kismet_2009; content: "Started source"; classtype:
info; sid: 5001013; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] BSSID
updated observed data encryption"; program: Kismet_2009; content: "updated
observed data encryption"; classtype: info; sid: 5001014; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Detected
new AP"; program: Kismet_2009; content: "Detected new 802.11 AP SSID";
classtype: info; sid: 5001015; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Detected
new 802.11 network"; program: Kismet_2009; content: "Detected new 802.11
network BSSID"; classtype: info; sid: 5001016; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Detected
new 802.11 probing client"; program: Kismet_2009; content: "Detected new
802.11 probing client"; classtype: info; sid: 5001017; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Found IP
address range"; program: Kismet_2009; content: "Found IP range"; classtype:
info; sid: 5001018; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Kismet
starting to gather packets [Startup]"; program: Kismet_2009; content:
"Found IP range"; classtype: info; sid: 5001019; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Kismet
shutting down"; program: Kismet_2009; content: "Stopped source"; classtype:
info; sid: 5001020; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Previously
AP network, now Ad-hoc.  Possible spoofed AP"; program: Kismet_2009;
content: "ADHOCCONFLICT server-ts"; classtype: possible-spoofed-ap; sid:
5001021; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] AP
broadcasting SSID AirJack, attempt to disrupt networks"; program:
Kismet_2009; content: "AIRJACKSSID server-ts"; classtype: suspicious-
traffic; sid: 5001022; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET]
Unauthorized device matching APSPOOF rule.  Possible spoofed AP"; program:
Kismet_2009; content: "APSPOOF server-ts"; classtype: possible-spoofed-ap;
sid: 5001023; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Out-of-
sequence BSS timestamp.  Possible spoofed AP"; program: Kismet_2009;
content: "BSSTIMESTAMP server-ts"; classtype: possible-spoofed-ap; sid:
5001024; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] AP changed
advertised channel. Possible spoofed AP"; program: Kismet_2009; content:
"CHANCHANGE server-ts"; classtype: possible-spoofed-ap; sid: 5001025; rev:
1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] AP changed
advertised encryption to Open. Possible spoofed AP"; program: Kismet_2009;
content: "CRYPTODROP server-ts"; classtype: possible-spoofed-ap; sid:
5001026; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] AP
broadcast deauthentication or disassociation of all clients, probable
denial of service"; program: Kismet_2009; pcre: "/DEAUTHFLOOD server-
ts|BCASTDISCON server-ts/"; classtype: possible-denial-of-service; sid:
5001027; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] DHCP
request doesn't match DHCP DISCOVER client id. Possible spoofed client";
program: Kismet_2009; content: "DHCPCLIENTID server-ts"; classtype:
suspicious-traffic; sid: 5001028; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET]
Misconfigured or spoofed client [ignoring DHCP]"; program: Kismet_2009;
content: "DHCPCONFLICT server-ts"; classtype: suspicious-traffic; sid:
5001029; rev: 1;)
```

```
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Spoofed client [incorrectly] injecting data"; program: Kismet_2009; content: "DISASSOCTRAFFIC server-ts"; classtype: suspicious-traffic; sid: 5001030; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Invalid disconnect/deauthenticate"; program: Kismet_2009; pcre: "/DISCONCODEINVALID server-ts|DEAUTHCODEINVALID server-ts/"; classtype: suspicious-traffic; sid: 5001031; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Client changed advertised DHCP vendor. Possible spoofed client"; program: Kismet_2009; content: "DHCPOSCHANGE server-ts"; classtype: suspicious-traffic; sid: 5001032; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Client changed advertised DHCP hostname. Possible spoofed client"; program: Kismet_2009; content: "DHCPNAMECHANGE server-ts"; classtype: suspicious-traffic; sid: 5001033; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Over-size SSID.  Possible exploit attempt"; program: Kismet_2009; content: "LONGSSID server-ts"; classtype: exploit-attempt; sid: 5001034; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Older Lucent/Orinico card scanning the network"; program: Kismet_2009; content: "LUCENTTEST server-ts"; classtype: network-scan; sid: 5001035; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Broadcom wireless improper SSID handling"; program: Kismet_2009; content: "MSFBCOMSSID server-ts"; classtype: exploit-attempt; sid: 5001036; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Windows D-Link improper SSID handling "; program: Kismet_2009; content: "MSFDLINKRATE server-ts"; classtype: exploit-attempt; sid: 5001037; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Windows Netgear over-size beacon frame"; program: Kismet_2009; content: "MSFNETGEARBEACON server-ts"; classtype: exploit-attempt; sid: 5001038; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Older version of Netsumbler detected"; program: Kismet_2009; content: "NETSTUMBLER server-ts"; classtype: exploit-attempt; sid: 5001039; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Zero length probe/response packet"; program: Kismet_2009; content: "NULLPROBERESP server-ts"; classtype: attempted-dos; sid: 5001040; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] AP changed 802.11d country. Possible spoofed AP"; program: Kismet_2009; content: "DOT11D server-ts"; classtype: possible-spoofed-ap; sid: 5001041; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] AP changed beacon rate. Possible spoofed AP"; program: Kismet_2009; content: "BEACONRATE server-ts"; classtype: possible-spoofed-ap; sid: 5001042; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] AP changed advertised encryption. Possible spoofed AP"; program: Kismet_2009; content: "ADVCRYPTCHANGE server-ts"; classtype: possible-spoofed-ap; sid: 5001043; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] Malformed management frames may indicate errors in the capture source. Possible an attack"; program: Kismet_2009; content: "MALFORMMGMT server-ts"; classtype: suspicious-traffic; sid: 5001044; rev: 1;)
alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[KISMET] AP sending excessive number of WPS messages. Possible a WPS brute force attack such as Reaver"; program: Kismet_2009; content: "WPSBRUTE server-ts"; classtype: exploit-attempt; sid: 5001045; rev: 1;)
```

q. Ubah *file* /usr/local/classification.config untuk menambahkan klasifikasi baru.

```
config classification: info,Info,3
config classification: possible-denial-of-service,Possible Denial of Service Attack,1
config classification: possible-spoofed-ap, Possible Spoofed Access Point,1
```

r. Tambahkan perintah di bawah ini untuk mengaktifkan Sagan di konfigurasi rsyslog.conf di /etc/rsyslog.conf

```
# The standard "input" template Sagan uses.  Basically the message 'format' Sagan understands.  The template is _one_ line.

$template sagan, "%fromhost-ip%|%syslogfacility-text%|%syslogpriority-text%|%syslogseverity-text%|%syslogtag%|%timegenerated:1:10:date-rfc3339%|%timegenerated:12:19:date-rfc3339%|%programname%|%msg%\n"
```

```
# The FIFO/named pipe location.  This is what Sagan will read.

*.*     |/opt/sagan.fifo;sagan
```

s. Aktifkan udp dengan menghilangkan tanda "#" pada:
```
$ModLoad imudp
$UDPServerRun 514
```
t. *Restart* Rsyslog
```
root@server:/var/log/sagan# /etc/init.d/rsyslog restart
```
u. Memberikan hak akses *user* sagan untuk *database* snorby
```
root@server:/var/log/sagan# mysql –u root –psecret123
mysql> grant all privileges on snorby.* to sagan identified by 'secret123';
mysql> grant all privileges on snorby.sensor to sagan identified by
'secret123';
```
v. Jalankan sagan dengan mengetikkan "sagan" di terminal.
w. Jalankan kismet dengan mengetikkan "kismet_drone" pada Kismet *drone* dan "kismet_server" pada *server*.
x. Aktifkan *snorby worker* di *website interface* Snorby agar Snorby dapat membaca data *cache* yang di-*log* oleh Sagan di *database* snorby.

Lampiran 2. Spesifikasi Alat

1. Spesifikasi *Hardware Server*

Model	*Notebook* HP dv2 1122ax
CPU	AMD Athlon(tm) Neo X2 *Dual Core Processor* L335 1,60 GHz
Memory	2 GB
Ethernet Card	Realtek RTL8102E/RTL8103E Family PCI-E *Fast Ethernet* NIC (NDIS 6.20)

2. Spesifikasi *Software Server*

Sistem Operasi	Ubuntu Edisi Desktop 11.10
Kernel Linux	3.0.0-12
Kismet	Versi development
Libpcap	Libpcap0.8-dev 1.1.1-8
Libnl	Libnl-dev 1.1-6ubuntu1
Libpcre	Libpcre3-dev 8.12-3ubuntu2
Wireshark	Versi 1.6.2-1
MySQL	Mysql-server 5.1.61-0ubuntu0.11
Apache2	Versi 2.2.20-1ubuntu1.2
Sagan	Versi development
Ruby	Versi 1.9.2
Rails	Versi 1.3.5
Rack	Versi 1.3.5
Rake	Versi 0.9.2
Snorby	Versi 2.5.1-0
Ntpdate	1:4.2.6.p2+dfsg-1ubuntu12

3. Spesifikasi *Wireless Drone*

Model	TP-LINK WR741ND
Prosesor	Atheros AR7240, 350 MHz
Flash Memory	4 MiB
RAM *Memory*	32 MiB
Ethernet	4x1

4. Konfigurasi *Wireless Drone*

Kismet *drone*	Versi 2010-07-R1-1
Wireless Card	AR9285
Firmware	OpenWRT
Kernel Linux	2.6.39.2
Software NTP	Ntpclient versi 2007 365-4

5. Spesifikasi AP1

Model	Cisco Linksys Wireless-G Broadband Router (WRT54G2)
Firmware	Versi 1.0.04 (*Build* 5)
Wireless Mode	802.11 b/g
Antena	Internal *Integrated* 2x

6. Spesifikasi AP2

Model	Ubiquiti LiteStation2
Firmware	DD-WRT
Wireless Mode	802.11 b/g
Antena	Eksternal 1x

7. Spesifikasi AP3

Model	D-LINK DWL AP+900
Firmware	D-LINK
Wireless Mode	802.11 b
Antena	Eksternal 1x

8. Spesifikasi Penyerang1

Model	*Netbook* Acer Aspire One 532h-2527
Prosesor	Intel Atom N450 @1,66 GHz, 512 KB *Cache*
RAM	1 GB DDR2 PC2-6400
Sistem Operasi	Ubuntu Edisi Desktop 10.04 BackTrack 5 R2 KDE
Wireless Card	Atheros AR9271 802.11b/g USB *Wi-Fi Adapter*
Aircrack-ng	Versi 1.2-bt0
Reaver	Versi 1.4-bt1
Ntpdate	Versi 1:4.2.4p8+dfsg-1ubuntu2.1

9. Spesifikasi Penyerang2

Model	*Laptop* Acer Aspire 5052
Prosesor	1,60 GHz AMD Turion 64 X2 *Processor* TL50 dengan 512 KB L2 *Cache*
RAM	1 GB DDR2 533/667 MHz SDRAM
Sistem Operasi	Ubuntu Edisi Desktop 10.04 BackTrack 5 R2 KDE
Wireless Card	Atheros AR9271 802.11b/g USB *Wi-Fi Adapter*
Aircrack-ng	Versi 1.2-bt0
Ntpdate	Versi 1:4.2.4p8+dfsg-1ubuntu2.1

10. Spesifikasi Penyerang3

Model	*Netbook* HP Mini 110-3500
Prosesor	Intel (R) Atom(TM) CPU N570 @1,66 GHz 1,67 GHz
RAM	1 GB DDR3 (1x1024 MB)
Sistem Operasi	Ubuntu Edisi Desktop 10.04 BackTrack 5 R2 KDE
Wireless Card	Atheros AR9271 802.11b/g USB *Wi-Fi Adapter*
Aircrack-ng	Versi 1.2-bt0
Ntpdate	Versi 1:4.2.4p8+dfsg-1ubuntu2.1

11. Spesifikasi Spoofed AP

Model	*Notebook* Toshiba Portege T210
Prosesor	Intel Core i3-380UM *Processor* (1,33 GHz, *cache* 3 MB)
RAM	2 GB DDR3 SODIMM PC-8500
Sistem Operasi	Ubuntu Edisi Desktop 11.10
Wireless Card	Atheros AR9271 802.11b/g USB *Wi-Fi Adapter*
Hostapd	Versi 0.7.3-2

Lampiran 3. Data Pengujian

1. Banyaknya Serangan
   a. 1 Serangan (penyerang1)

Alert	*Frame* De-otentikasi (*frame*)	Waktu Pendeteksian (detik)	SN Awal yang Terdeteksi	Waktu Awal Deteksi	Waktu Awal Serangan
10	1531	8,910	0	13:25:21.371645000	13:25:21.362230000
12	1326	8,922	0	13:30:21.467774000	13:30:21.455165000
12	1250	8,463	0	13:31:21.399383000	13:31:21.386174000
12	1445	8,922	0	13:32:21.378650000	13:32:21.364816000
12	1404	8,910	0	13:33:21.385068000	13:33:21.370574000
12	1436	8,917	1	13:34:21.393876000	13:34:21.378833000
14	1271	8,732	84	13:35:21.577818000	13:35:21.562085000
12	1235	8,911	1	13:36:21.509451000	13:36:21.493112000
12	1445	8,908	0	13:37:21.339194000	13:37:21.322246000
12	1411	8,868	2	13:38:21.422446000	13:38:21.404781000

b. 2 Serangan (penyerang1)

Alert	*Frame* De-otentikasi (*frame*)	Waktu Pendeteksian (detik)	SN Awal yang Terdeteksi	Waktu Awal Deteksi	Waktu Awal Serangan
5	890	8,859	2	14:28:21.445660000	14:28:21.395218000
2	910	8,890	3	14:33:21.480976000	14:33:21.427159000
3	1168	8,901	0	14:34:21.481137000	14:34:21.426668000
2	917	8,793	62	14:35:21.621187000	14:35:21.566050000
3	951	8,874	18	14:36:21.430695000	14:36:21.374936000
4	1052	8,911	7	14:37:21.516003000	14:37:21.459573000
6	981	8,924	0	14:38:21.405415000	14:38:21.348384000
5	991	8,831	10	14:39:21.454481000	14:39:21.396733000
4	958	8,910	0	14:40:21.418481000	14:40:21.359988000
4	1026	8,920	1	14:41:21.530019000	14:41:21.470890000

c. 3 Serangan (penyerang1)

Alert	*Frame* De-otentikasi (*frame*)	Waktu Pendeteksian (detik)	SN Awal yang Terdeteksi	Waktu Awal Deteksi	Waktu Awal Serangan
3	766	8,931	0	15:17:21.460873000	15:17:21.377803000
2	834	8,917	1	15:18:21.467447000	15:18:21.383686000
3	595	8,755	15	15:19:21.435044000	15:19:21.350722000
2	752	8,931	0	15:20:21.481031000	15:20:21.395965000
1	762	8,941	0	15:21:21.484987000	15:21:21.399224000
3	576	8,819	3	15:22:21.432161000	15:22:21.345736000
3	792	8,929	0	15:23:21.498303000	15:23:21.411246000
1	783	8,482	0	15:24:21.504926000	15:24:21.417263000
1	1001	8,784	1	15:25:21.514898000	15:25:21.426520000
4	776	8,248	128	15:46:22.004550000	15:46:21.902259000

2. Peletakan Sensor

a. Ruang 1

Alert	*Frame* De-otentikasi (*frame*)	Waktu Pendeteksian (detik)	SN Awal yang Terdeteksi	Waktu Awal Deteksi	Waktu Awal Serangan
10	1087	8,660	110	16:31:21.636916000	16:31:21.623751000
10	1479	8,900	0	16:35:21.426062000	16:35:21.410438000
10	1361	8,901	0	16:36:21.358510000	16:36:21.342321000
10	1590	8,851	0	16:37:21.439122000	16:37:21.422335000
10	1458	8,897	3	16:38:21.452094000	16:38:21.434589000
10	1539	8,910	0	16:39:21.349732000	16:39:21.331609000
10	1664	8,868	0	16:40:21.356276000	16:40:21.337621000
14	1634	8,901	0	16:41:21.362882000	16:41:21.343506000
10	1437	8,904	0	16:42:21.398317000	16:42:21.378394000
12	1558	8,900	0	16:43:21.406941000	16:43:21.386406000

b. Ruang 2

Alert	*Frame* De-otentikasi (*frame*)	Waktu Pendeteksian (detik)	SN Awal yang Terdeteksi	Waktu Awal Deteksi	Waktu Awal Serangan
8	420	8,447	128	17:16:21.843038000	17:16:21.802169000
10	546	8,570	73	17:21:21.577114000	17:21:21.533226000
6	538	7,738	128	17:22:21.815936000	17:22:21.771367000
8	378	7,220	333	17:23:22.508072000	17:23:22.462992000
8	321	7,246	123	17:24:21.705497000	17:24:21.659752000
6	416	7,539	129	17:25:21.906344000	17:25:21.860018000
9	361	8,790	52	17:26:21.496666000	17:26:21.449783000
8	300	8,369	37	17:27:21.539691000	17:27:21.492196000
10	346	8,448	128	17:28:21.846383000	17:28:21.798310000
6	449	6,953	172	17:29:21.936137000	17:29:21.887437000

c. Ruang 3

Alert	*Frame* De-otentikasi (*frame*)	Waktu Pendeteksian (detik)	SN Awal yang Terdeteksi	Waktu Awal Deteksi	Waktu Awal Serangan
6	348	7,964	256	19:03:22.492106000	19:03:22.385436000
8	329	8,281	0	19:07:21.505516000	19:07:21.396355000
6	300	7,940	0	19:08:21.512157000	19:08:21.402370000
8	276	7,090	512	19:09:23.261619000	19:09:23.151252000
8	298	8,161	118	19:10:21.778569000	19:10:21.667506000
10	434	8,413	18	19:11:21.468196000	19:11:21.356519000
8	283	7,374	332	19:12:22.610173000	19:12:22.497906000
8	337	7,989	0	19:13:21.545110000	19:13:21.432177000
10	405	8,277	0	19:14:21.449407000	19:14:21.335940000
6	370	8,314	177	19:15:22.118942000	19:15:22.004815000