MEKANISME KEAMANAN UNTUK MELINDUNGI APLIKASI BERBASIS WEB DARI SERANGAN SESSION HIJACKING

Diajukan Sebagai Salah Satu Syarat Untuk

Memperoleh Gelar Sarjana Strata-1

Teknik Informatika

Disusun Oleh:

Nama : Wahyu Aji Wibawa

NIM : 07523374

JURUSAN TEKNIK INFORMATIKA FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS ISLAM INDONESIA
YOGYAKARTA

2011

LEMBAR PENGESAHAN PEMBIMBING

MEKANISME KEAMANAN UNTUK MELINDUNGI APLIKASI BERBASIS WEB DARI SERANGAN SESSION HIJACKING

TUGAS AKHIR

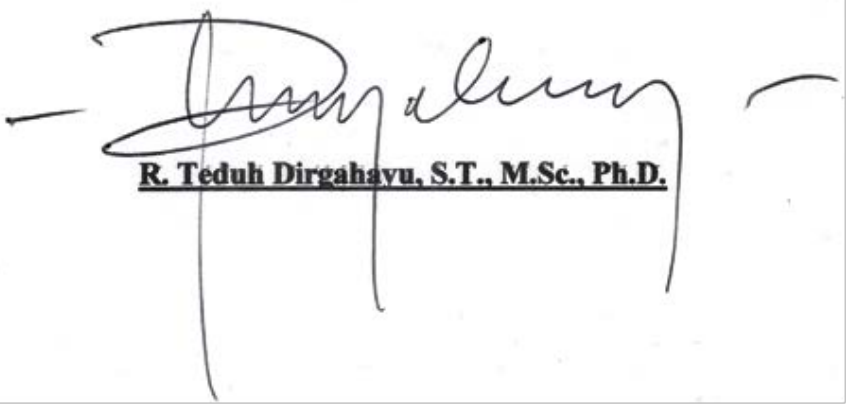
Disusun Oleh:

Nama : Wahyu Aji Wibawa

NIM : 07523374

Yogyakarta, 13 Desember 2011

Dosen Pembimbing,



R. Teduh Dirgahayu, S.T., M.Sc., Ph.D.

LEMBAR PENGESAHAN PENGUJI

MEKANISME KEAMANAN UNTUK MELINDUNGI APLIKASI BERBASIS WEB DARI SERANGAN SESSION HIJACKING

TUGAS AKHIR

Disusun Oleh:

Nama : Wahyu Aji Wibawa

NIM : 07523374

Telah Dipertahankan di Depan Sidang Penguji Sebagai Salah Satu Syarat Untuk Memperoleh Gelar Sarjana Jurusan Teknik Informatika Fakultas Teknologi Industri Universitas Islam Indonrsia

Yogyakarta, 28 Desember 2011

Tim Penguji,

R. Teduh Dirgahayu, S.T., M.Sc., Ph.D.
Ketua

Taufiq Hidayat, ST., M.CS.
Anggota 1

Syarif Hidayat, S.Kom., M.I.T.
Anggota 2

Mengetahui,
Ketua Jurusan Teknik Informatika
Universitas Islam Indonesia

Yudi Prayudi, S.Si., M.Kom.

LEMBAR PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini,

Nama : Wahyu Aji Wibawa

NIM : 07523374

Menyatakan bahwa seluruh komponen isi dalam Laporan Tugas Akhir ini adalah hasil karya saya sendiri. Apabila dikemudian hari terbukti bahwa ada beberapa bagian dari karya ini bukan hasil karya sendiri, maka saya siap menanggung resiko dan konsekuensi apapun. Demikian pernyataan ini saya buat, semoga dapat dipergunakan sebagai mana mestinya.

Yogyakarta, 13 Desember 2011

Wahyu Aji Wibawa

HALAMAN PERSEMBAHAN

Kedua orang tua yang selalu mendukung memberikan semangat, nasehat, doa, cinta dan kasih kepada anak-anaknya, sehingga kami selalu termotivasi menjadi yang terbaik.

Teman, kerabat dan orang terdekat saya yang selalu mendukung dari awal hingga akhir perjuangan pengerjaan tugas akhir ini.

MOTO

“Allah tidak membebani seseorang melainkan sesuai dengan kesanggupannya”

(QS. Al Baqarah:286)

“Everything must be balance”

(Gt. Dika Nugraha Aryadinata)

“Manusia hanya bisa merencanakan, Allah yang menentukan segalanya”

KATA PENGANTAR

Assalamu'alaikum Wr. Wb,

Alhamdulillah, dengan mengucap syukur kehadirat Allah SWT yang senantiasa memberikan petunjuk serta melimpahkan rahmat dan hidayah-Nya sehingga Laporan Tugas Akhir dengan judul "Mekanisme Keamanan Untuk Melindungi Aplikasi Berbasis Web Dari Serangan Session Hijacking" ini dapat diselesaikan. Tak lupa sholawat serta salam senantiasa tercurah kepada Rasulullah Muhammad SAW beserta keluarga dan seluruh sahabatnya atas suri tauladannnya sehingga kita berada di jaman Islamiyah.

Laporan Tugas Akhir ini membahas tentang pembuatan mekanisme manajemen *session* dan autentikasi berbasis *challenge/response* untuk melindungi aplikasi berbasis web dari serangan *session hijacking.* Laporan Tugas Akhir ini merupakan salah satu syarat yang harus dipenuhi untuk memperoleh gelar sarjana di Jurusan Teknik Informatika Fakultas Teknologi Industri Universitas Islam Indonesia.

Dalam pengerjaan tugas akhir ini banyak pihak yang telah memberikan bantuan baik berupa materi maupun non materi sehingga tugas akhir ini dapat diselesaikan. Oleh karena itu penlis mengucapkan terimakasih kepada:

1. Allah SWT atas segala limpahan Rahmat dan Hidayah-Nya sehingga penulis dapat menyelesaikan tugas akhir ini dengan baik dan lancar serta bermanfaat.
2. Rasulullah Muhammad SAW yang telah memberi kita suri tauladan.
3. Kedua orang tua, kakak dan adik yang selalu memberikan kasih sayang, nasehat, doa serta dukungannya.
4. Bapak Ir. Gumbolo, HS, M.SC. selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia.

5. Bapak Yudi Prayudi, S.Si., M.Kom. selaku Ketua Jurusan Informatika Universitas Islam Indonesia.
6. Bapak R. Teduh Dirgahayu, S.T., M.Sc.,Ph.D. selaku dosen pembimbing tugas akhir. Terimakasih atas segala dukungan, bantuan dan ilmu yang telah diberikan.
7. Bapak Hamid, S.T., M.Eng terimakasih atas arahan dan bimbingannya.
8. Seluruh dosen di Jurusan Teknik Informatika Universitas Islam Indonesia yang telah memberikan ilmunya, serta memberikan motivasi dan semangat.
9. Teman-teman jurusan teknik informatika.
10. Semua pihak yang tidak bisa disebutkan satu per Satu.

Kepada segenap pihak yang secara langsung maupun tidak langsung telah membantu serta memudahkan proses selesainya tugas akhir ini, semoga Allah SWT melimpahkan rahmat serta memberikan balasan yang sebaik-baiknya.

Penulis menyadari bahwa dalam penulisan laporan ini masih belum sempurna, karena keterbatasan kemampuan dan pengalaman. Penulis berharap laporan ini dapat menjadi sesuatu yang bermanfaat bagi kita semua untuk menambah wawasan dan pengetahuan.

Wassalamualaikum Wr. Wb,

Yogyakarta, 13 Desember 2011

Wahyu Aji Wibawa

SARI

Aplikasi berbasis web berjalan dengan protokol HTTP (Hyperteks Transfer Protocol) yang bersifat *stateless*, artinya protokol HTTP tidak bisa mempertahankan informasi antara satu permintaan (*request*) dengan permintaan yang lainnya. Untuk mendapatkan situs web yang *stateful* membutuhkan *session.* Dengan *session* aplikasi berbasis web dapat mencatat dan mempertahankan informasi antar *request* yang dilakukan oleh *client.* Penggunaan *session* pada aplikasi berbasis web menimbulkan celah keamanan untuk serangan *session hijacking.* Untuk melindungi aplikasi berbasis web dari serangan *session hijacking* dibuat dua mekanisme keamanan yaitu mekanisme manajemen *session* dan mekanisme autentikasi *challenge/response.* Mekanisme manajemen *session* membangkitkan *session id* baru dan melakukan autentikasi untuk setiap *request* dengan memanfaatkan informasi *user agent browser* dan informasi HTTP *referer*. Sedangkan mekanisme autentikasi *challenge/response* diguanakan untuk autentikasi terhadap transaksi-transaksi yang penting. Setelah melakukan pengujian terhadap mekanisme keamanan tersebut, kedua mekanisme keamanan dinyatakan dapat diimplementasikan dan dapat berjalan dengan baik pada aplikasi berbasis web. Kedua mekanisme keamanan tersebut juga dapat melindungi aplikasi berbasis web dari serangan *session hijacking.*

Kunci: session id, fingerprint, secret key, challenge, response.

TAKARIR

attacker	penyerang yang tujuannya untuk merusak sistem.
challenge	suatu *string* yang dipilih secara acak yang digunakan untuk membangkitkan nilai *response*.
chipertext	pesan yang terenkripsi.
epoh	jumlah waktu dari 1 Januari 1970 00:00:00 GMT (Unix Epoch) sampai waktu sekarang.
hypermedia	dokumen multi media yang mengandung informasi dan saling terhubung serta dapat ditampilkan bersama-sama.
local storage	tempat penyimpanan pada *client* dalam HTML5.
plaintext	pesan biasa yang tidak terenkripsi.
promiscuous	modus baca semuanya pada Ethernet Card. Modus ini biasanya digunakan network administrator untuk analisis trafik jaringan.
proxy	sistem komputer atau aplikasi yang bertindak sebagai perantara permintaan dari *client* kepada *server*.
response	suatu *string* yang digunakan untuk autentikasi terhadap nilai *challenge* tertentu.
salt	*string* rahasia yang digunakan untuk memperkuat hasil enkripsi.
session id	*string* unik yang diberikan *server* kepada *user* yang digunakan untuk menjaga *state user*.
session hijacking	serangan yang mengambil alih koneksi antara *client* dengan *server* dengan memanfaatkan *session id* yang valid.
state	rangkaian aktivitas user pada suatu sistem.
trojan horse	aplikasi yang dibuat dengan tujuan untuk menginfeksi dan memata-matai sistem.
vulnerability	Kelemahan yang terdapat pada suatu sistem yang dapat membuka celah keamanan.

DAFTAR ISI

LEMBAR PENGESAHAN PEMBIMBING ..... **ii**
LEMBAR PENGESAHAN PENGUJI ..... **iii**
LEMBAR PERNYATAAN KEASLIAN TUGAS AKHIR ..... **iv**
HALAMAN PERSEMBAHAN ..... **v**
MOTO ..... **vi**
KATA PENGANTAR ..... **vii**
SARI ..... **ix**
TAKARIR ..... **x**
DAFTAR ISI ..... **xi**
DAFTAR GAMBAR ..... **xiii**
DAFTAR TABEL ..... **xiv**
DAFTAR ALGORITMA ..... **xiv**
1 PENDAHULUAN **1**
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Batasan Masalah 2
1.4 Tujuan Penelitian 3
1.5 Manfaat Penelitian 3
1.6 Metodologi Penelitian 3
1.6.1 Studi Literatur 4
1.6.2 Perancangan dan Implementasi Sistem 4
1.7 Sistematika Penulisan 4
2 LANDASAN TEORI **6**
2.1 WWW 6
2.2 HTTP 6
2.2.1 HTTP Message 7
2.2.2 Manajemen State 10

2.3 Session Hijacking....12
2.3.1 Teknik Session Hijacking....13
2.4 Autentikasi....16
2.5 Kriptografi....17
3 ANALISIS DAN DESAIN....19
3.1 Analisis Sistem....19
3.1.1 Analisis Masalah....19
3.1.2 Perangkat Keras dan Perangkat Lunak Yang Dibutuhkan....21
3.2 Desain Sistem....21
3.2.1 Activity Diagram....21
3.2.2 Algoritma....27
3.3 Rencana Pengujian....32
3.3.1 Implementasi Pada Aplikasi Berbasis Web....32
3.3.2 Penetration Testing....32
4 HASIL DAN PEMBAHASAN....34
4.1 Implementasi Sistem....34
4.1.1 Mekanisme Manajemen Session....34
4.1.2 Mekanisme Autentikasi Challenge/Response....37
4.2 Pengujian Sistem....42
4.2.1 Implementasi Pada Aplikasi Berbasis Web....43
4.2.2 Penetration Testing....47
4.3 Kelemahan Mekanisme Keamanan....59
5 KESIMPULAN DAN SARAN....61
5.1 Kesimpulan....61
5.2 Saran....61
DAFTAR PUSTAKA....63
LAMPIRAN....65

DAFTAR GAMBAR

Gambar 2.1 Komunikasi HTTP (Thomas, 2001:14) 7
Gambar 2.2 HTTP Message (Gourley, 2002:47) 8
Gambar 2.3 HTTP Request-Respose + Cookie 11
Gambar 2.4 Serangan Session Hijacking 13
Gambar 2.5 Sniffing Session Id 14
Gambar 2.6 Serangan Man-in-the-Midle 15
Gambar 2.7 Serangan Session Fixation 16
Gambar 2.8 Proses kriptografi sederhana 18
Gambar 3.1 Activity Diagram Mekanisme Manajemen Session 23
Gambar 3.2 Activity Diagram Autentikasi Challenge/Response 25
Gambar 3.3 Activity Diagram Mobile Token 27
Gambar 4.1 Prompt Secret Key User 42
Gambar 4.2 Halaman Untuk Membangkitkan Nilai Response 42
Gambar 4.3 Autentikasi Challenge/Response Pada Aplikasi Berbasis Web 45
Gambar 4.4 Pesan Kesalahan Autentikasi Challenge/Response 46
Gambar 4.5 Membangkitkan Nilai Response Pada Mobile Token 47
Gambar 4.6 Skenario Sniffing 48
Gambar 4.7 Hasil Sniffing Sistem Tanpa Session 49
Gambar 4.8 Hasil Sniffing Sistem Dengan URL Session 50
Gambar 4.9 Hasil Sniffing Sistem Dengan Cookie Session 51
Gambar 4.10 Hasil Sniffing Sistem Dengan Manajemen Session 52
Gambar 4.11 Hasil Man-in-the-Middle Sistem Tanpa Session 53
Gambar 4.12 Hasil Man-in-the-Middle Sistem Dengan URL Session 53
Gambar 4.13 Hasil Man-in-the-Middle Sistem Dengan cookie Session 54
Gambar 4.14 Hasil Man-in-the-Middle Sistem Dengan manajemen Session 54
Gambar 4.15 Request/Response Sistem Tanpa Session 55
Gambar 4.16 Request/Response Sistem dengan URL Session 56
Gambar 4.17 Request/Response Sistem Dengan Cookie Session 57
Gambar 4.18 Request Response Sistem Dengan Manajemen Session 58

DAFTAR TABEL

Tabel 2.1 HTTP Method (Gourley, 2002:48)..... 8
Tabel 2.2 Status Code Classes (Gourley, 2002:49)..... 9
Tabel 4.1 Hasil Pengujian Mekanisme Manajemen Session Tahap Satu..... 44
Tabel 4.2 Hasil Pengujian Autentikasi Challenge/Response 46
Tabel 4.3 Hasil Pengujian Mekanisme Keamanan Tahap Dua 58

DAFTAR ALGORITMA

Algoritma 3.1 Algoritma Mekanisme Manajemen Session..... 29
Algoritma 3.2 Algoritma Autentikasi Challenge/Response..... 31
Algoritma 3.3 Algoritma Mobile Token..... 32

BAB I

PENDAHULUAN

1.1 Latar Belakang

Web merupakan aplikasi *client-server* yang pertama kali diperkenalkan oleh Tim Benners Lee pada tahun 1991. Situs web awalnya hanya digunakan untuk komunikasi antar peneliti yang tersebar di seluruh dunia. Seiring dengan berjalannya waktu dan perkembangannya, fungsi situs web pun berkembang. Saat ini situs web telah dimanfaatkan dalam berbagai bidang sebagai sarana informasi dan komunikasi. Misalnya dalam dunia bisnis, banyak perusahaan yang telah memanfaatkan situs web, baik sebagai *company profile*, sarana komunikasi antar karyawan maupun antar kantor, serta sebagai sarana komunikasi dengan rekan bisnis dalam rangka kerjasama, membagi informasi bisnis, perencanaan bisnis bersama atau dalam urusan yang lain.

Aplikasi berbasis web berjalan dengan *protokol* HTTP (Hyperteks Transfer Protocol). Protokol HTTP bersifat *stateless*, artinya protokol HTTP tidak bisa mempertahankan informasi antara satu permintaan (*request*) dengan permintaan yang lainnya. Suatu *server* HTTP melayani suatu *request* tanpa tergantung pada *request* yang lain baik sebelum ataupun sesudahnya. Untuk mendapatkan situs web yang *stateful* membutuhkan suatu teknik tersendiri, yaitu dengan memanfaatkan *session.* Fungsi *session* adalah untuk mempertahankan *state*. Dengan *session* suatu sistem berbasis web dapat mencatat dan mempertahankan informasi antar *request* yang dilakukan oleh *client.*

Meskipun dengan pemanfaatan *session* memberikan *statefulness*, aplikasi berbasis web membutuhkan penanganan khusus pada segi keamanan. Karena dengan adanya *session* aplikasi berbasis web rawan dengan serangan yang berbasis *session* (*session-base attacks*), yang salah satunya adalah *session hijacking. Hijacking* adalah suatu teknik serangan yang mengambil alih koneksi

antara *client* dengan *server* dengan memanfaatkan *session key* yang valid yang telah didapatkan penyerang (*attacker*) dengan teknik-teknik tertentu.

Dalam dunia bisnis keamanan data merupakan suatu hal yang sangat penting. Akan tetapi keamanan kurang mendapat perhatian dari pemilik maupun pengelola sistem informasi. Sering kali masalah keamanan berada diurutan kedua, bahkan diurutan terakhir (Rahardjo, 2005:1). Akan menjadi sangat fatal jika aplikasi berbasis web yang digunakan oleh perusahaan terkena serangan *session hijacking* sehingga informasi bisnis bocor keluar perusahaan atau berhasil dicuri oleh pesaing bisnis, tidak hanya kerugian *financial* yang dialami tetapi juga waktu untuk *recovery* data bahkan nama baik dan kepercayaan pelanggan akan hilang. Maka dari itu diperlukan suatu mekanisme keamanan yang dapat melindungi aplikasi berbasis web dari serangan *session hijacking*.

1.2 Rumusan Masalah

Dari uraian latar belakang di atas maka dapat diambil rumusan masalah yaitu:

1. Bagaimana membangun mekanisme manajemen *session* yang dapat mempersulit *attacker* dalam melakukan serangan *session hijacking* pada aplikasi berbasis web.
2. Bagaimana membangun mekanisme autentikasi *challenge/response* yang dapat melindungi data pada aplikasi berbasis web jika terjadi serangan *session hijacking*.

1.3 Batasan Masalah

Dalam suatu penelitian agar pembahasan tidak telalu luas serta agar tidak menyimpang dari hal yang diinginkan maka diperlukan batasan masalah. Batasan masalah dalam penelitian ini adalah sebagai berikut:

1. Mekanisme keamanan diimplementasikan pada aplikasi berbasis web yang dikembangkan menggunakan PHP & MySQL.

2. Pembahasan terbatas pada *session hijacking* di level aplikasi.
3. Mekanisme keamanan hanya diimplementasikan pada aplikasi web yang berjalan pada protokol HTTP, dan tidak mencakup pada penggunaan protokol HTTPS.
4. Sistem informasi yang dibuat hanya digunakan sebagai sarana implementasi dan pengujian mekanisme keamanan, dan bukan menjadi fokus dari pembahasan.

1.4 Tujuan Penelitian

Tujuan penelitian ini adalah membangun mekanisme manajemen *session* untuk mempersulit *attacker* dalam melakukan serangan *session hijacking* serta membangun mekanisme autetikasi *challenge/response* untuk melindungi aplikasi berbasis web jika terjadi serangan *session hijacking*.

1.5 Manfaat Penelitian

Dengan adanya penelitian ini diharapkan dapat memberikan manfaat sebagai berikut:

1. Bagi programmer web atau pengelola sistem informasi untuk mengetahui bagaimana *session hijacking* dilakukan dan bagaimana penangananya, sehingga bisa meningkatkan sistem keamanan mereka.

2. Bagi Pengguna atau pelanggan situs web agar lebih berhati-hati dalam melakukan aktivitas di Internet.

1.6 Metodologi Penelitian

Metodologi penelitian berfungsi untuk menggambarkan bagaimana langkah atau strategi peneliti dalam melakukan penelitian. Dalam penelitian Tugas Akhir ini metodologi yang digunakan penulis terbagi menjadi dua tahap yaitu:

1.6.1 Studi Literatur

Metode studi literatur yang digunakan penulis dalam penelitian ini adalah metode studi pustaka. Pada tahap ini penulis melalukan pengumpulan dan pembacaan literatur yang digunakan sebagai referensi dalam penelitian ini.

1.6.2 Perancangan dan Implementasi Sistem

Metode perancangan dan implementasi sistem disusun berdasarkan hasil yang diperoleh dari tahap sebelumnya yaitu dari metode studi literatur. Metode tersebut meliputi:

a. Analisis Kebutuhan Sistem

   Pada tahap analisis kebutuhan sistem akan dilakukan identifikasi kebutuhan yang dibutuhkan dalam pembuatan, implementasi serta pengujian mekanisme keamanan yang akan dibuat.

b. Desain Sistem

   Pada tahap ini akan membuat desain dari mekanisme keamanan. Desain ini akan menggambarkan alur kerja sistem dan proses yang dilakukan sistem.

c. Implementasi Desain Sistem

   Pada tahap ini akan dilakukan implementasi dari desain sistem yang telah dibuat sebelumnya. Pada tahap ini desain sistem akan diimplementasikan kedalam bahasa pemrograman.

d. Pengujian Sistem

   Setelah tahap implementasi desain sistem selesai, tahap selanjutnya adalah pengujian sistem. Pada tahap ini akan dilakukan pengujian sistem untuk mengetahui apakah sistem yang telah dibuat dapat berjalan sesuai dengan tujuan yang diharapkan.

1.7 Sistematika Penulisan

Sistematika penulisan yang digunakan dalam pembuatan Laporan Tugas Akhir ini adalah sebagai berikut:

BAB I Pendahuluan

Bagian ini membahas tentang latar belakang masalah, batasan masalah, rumusan masalah, tujuan penelitian serta manfaat penelitian dan metodologi penelitian yang digunakan dalam Laporan Tugas Akhir ini.

BAB II Landasan Teori

Bagian ini membahas tentang WWW (World Wide Web), HTTP (Hipertext Transfer Protocol), *session hijacking*, autentikasi dan kriptografi.

BAB III Analisis dan Desain

Bagian ini berisi tentang analisis masalah pada aplikasi berbasis web serta perangkat keras dan perangkat lunak yang dibutuhkan. Bagian ini juga akan menjelaskan tentang desain, alur kerja sistem dan rencana pengujian sistem.

BAB IV Hasil dan Pembahasan

Bagian ini memuat tentang hasil dari mekanisme keamanan yang telah dibuat serta membahas tentang pengujian yang dilakukan terhadap mekanisme keamanan dan kelemahan dari mekanisme keamanan yang telah dibuat.

BAB V Penutup

Bagian ini memuat tentang kesimpulan dari seluruh rangkaian yang telah dilakukan dalam penelitian tugas akhir ini. Bab ini juga memuat saran dari penulis yang ditujukan untuk pengembangan sistem lebih lanjut.

BAB II

LANDASAN TEORI

2.1 WWW

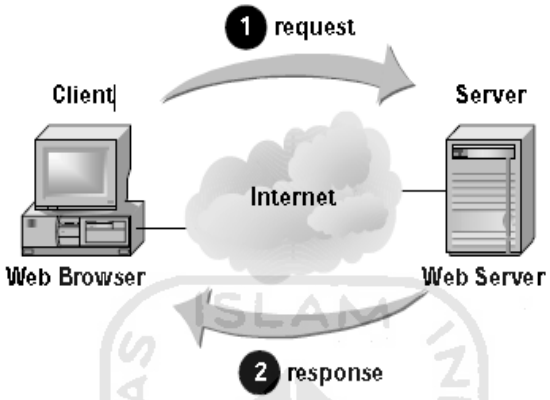
World Wide Web (WWW) atau yang biasa disebut dengan situs web adalah sebuah layanan dari Internet yang saat ini telah digunakan oleh banyak pengguna Internet di seluruh dunia. Awal terbentuknya WWW bermula di salah satu laboratorium fisika di Eropa CERN (Conseil Europeen pour la Recherche Nucleaire), yang dalam bahasa ingris disebut dengan *European Organization for Nuclear Research*. Bermula pada bulan Maret 1989, ketika Tim Berners Lee yang merupakan salah seorang peneliti dari CERN mengajukan proposal yang berjudul "*Information Management: A Proposal*". Proposal tersebut yang kemudian menjadi awal dari penemuan *World Wide Web* (WWW) (Shiflett, 2003).

Tim Berners Lee awalnya memperkenalkan WWW sebagai perpustakaan virtual, yaitu sebuah sistem kontrol dokumen untuk berbagi informasi dan sumber daya diantara para peneliti yang tersebar diseluruh dunia (Shklar, 2003:2). Seiring dengan perkembangannya fungsi dan pengguna situs web berkembang, yang sebelumnya hanya digunakan sebagai perpustakaan virtual oleh kalangan tertentu kini telah digunakan oleh masyarakat umum. Fungsi situs web juga telah merambah ke dalam berbagai bidang kehidupan mulai dari pendidikan, bisnis, pemerintahan, maupun bidang yang lain.

2.2 HTTP

HTTP (Hipertext Transfer Protcol) adalah protokol jaringan lapis aplikasi yang digunakan untuk sistem informasi terdistribusi, kolaboratif dan *hypermedia.* HTTP telah digunakan oleh WWW sejak tahun 1990 (Internet, 1999:7). HTTP merupakan protokol TCP yang menggunakan port standar 80 dan merupakan protokol *client-server* yang berkomunikasi antara HTTP *client* dan HTTP *server.* Protokol HTTP merupakan protokol komunikasi dua arah antara *client* dengan *server* dengan metode *request* dan *response* seperti Gambar 2.1. Pesan yang

dikirim dari *client* (*web browser*) ke *server* (*web server*) yang disebut dengan *request.* Sebaliknya setiap pesan yang dikirimkan dari *server* (*web server*) ke *client* (*web browser*) disebut dengan *response. Response* HTTP merupakan jawaban dari *request* HTTP.



Gambar 2.1 Komunikasi HTTP (Thomas, 2001:14)

HTTP merupakan *stateless* protokol, setiap *request* yang dilakukan HTTP tidak saling terkait satu sama lain atau berdiri sendiri, karena protokol HTTP tidak memiliki mekanisme untuk mengidentifikasi atau mengenali *request* (Shiflett, 2003).

2.2.1 HTTP Message

Dalam berkomunikasi protokol HTTP menggunakan pesan (HTTP *Message*). Setiap pesan yang dipertukarkan antara *client* dengan *server* berisi tiga bagian utama yaitu (Gourley, 2002:44):

1. **Start Line**, mendiskripsikan apakah pesan HTTP berupa pesan *request* atau *response.*
2. **Blok Header**, berisi variable *header* HTTP dan nilainya.
3. **Body Message**, berisi data. *Body Message* boleh kosong atau tidak berisi apa-apa.

2.2.1.1 Message Format

Format pesan *request* dan *response* adalah sebagai berikut (Gourley, 2002:44):

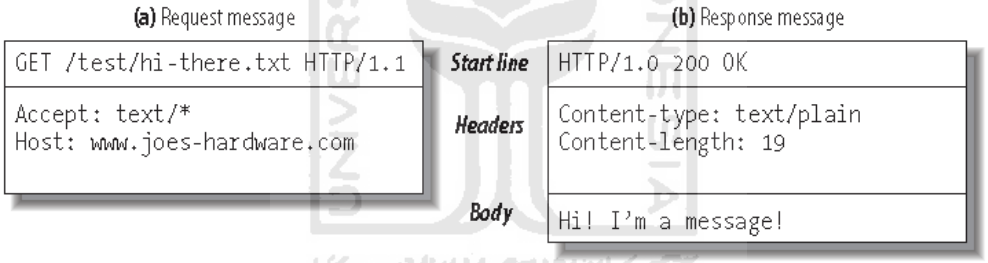
Format *Request Message*

```
<method> <request-URL> <version>
<headers>

<entity-body>
```

Format *Respone Message*

```
<version> <status code> <reason-phrase>
<headers>

<entity-body>
```

Contoh HTTP *message* dapat dilihat pada Gambar 2.2 di bawah ini.



Gambar 2.2 HTTP Message (Gourley, 2002:47)

Penjelasan masing-masing bagian dari format *request* dan *response* HTTP adalah sebagai berikut (Gourley, 2002:44):

Method, perintah yang dikirimkan *client* agar *server* melakukan sesuatu terhadap *resource*. Beberapa *method* yang bisa digunakan dapat dilihat pada Tabel 2.1.

Tabel 2.1 HTTP Method (Gourley, 2002:48)

METHOD	FUNGSI
GET	Mendapatkan dokumen dari *server*
HEAD	Mendapatkan *header* dokumen dari *server*

POST	Mengirim data ke *server* untuk diproses
PUT	Menyimpan *request body* ke *sever*
TRACE	*Trace* pesan (*message*) melalui proxy *server* menuju *server*
OPTIONS	Menentukan metode apa yang dapat beroperasi pada *server*
DELETE	Menghapus sebuah dokumen dari *server*

Request-URL, URL (Uniform Resource Locator) mendiskripsikan lokasi dari sebuah *resource* yang terdapat pada *server*. URL berisi dokumen *root* yang merupakan akar dokumen yang disimbolkan dengan *slash* (/) dan biasanya diikuti dengan *path* atau alamat *directory* dari file yang ingin di buka.

Version, versi HTTP yang digunakan. Misalnya, 1.1 atau 1.0.

Satus Code, merupakan tiga digit angka yang mendefinisikan apa yang terjadi terhadap *request. Status code* dikelompokkan menjadi beberapa kategori seperti pada Tabel **2.2**.

Tabel 2.2 Status Code Classes (Gourley, 2002:49)

Overall range	Defined range	Category
100-199	100-101	Informational
200-299	200-206	Successful
300-399	300-305	Redirection
400-499	400-415	*Client* error
500-599	500-505	*Server* error

Reason-Phrase, status yang bisa dibaca dan dimengerti manusia, biasanya untuk menerangkan *status code*. Contohnya adalah sebagai berikut.

`HTTP/1.0 200 OK` → Reason-Phrase adalah **OK**.

Headers, bisa kosong atau berisi beberapa HTTP *header* yang berupa variabel *header* diikuti dengan nilainya. Contoh HTTP *header* seperti *script* di bawah ini dengan variabel *header* **User-Agent** dan diikuti nilainya yang dipisahkan oleh tanda **":".**

```
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:6.0)
Gecko/20100101 Firefox/6.0
```

Entity-Body, berisi blok data, tidak semua pesan HTTP memiliki *entity body*.

2.2.2 Manajemen State

Pada uraian sebelumnya telah dijelaskan bahwa HTTP merupakan protokol yang *stateless*. Sehingga untuk membuat sistem yang *stateful* diperlukan teknik khusus. Teknik yang biasa digunakan untuk manajemen *state* adalah dengan memanfaatkan *session. Session* merupakan suatu metode yang digunakan untuk menjaga data antar halaman (S'to, 2009:83). Dengan adanya *session* suatu aplikasi berbasis web dapat mempertahankan informasi antar *request* yang dilakukan oleh *client. Session* dapat mencatat serta mempertahankan informasi dari *request* yang dilakukan *client.* Sehingga *server* dapat membedakan *request* yang datang dari beberapa *client* yang berbeda dan mengetahui bahwa *client* yang melakukan *request* adalah *user* yang sama.

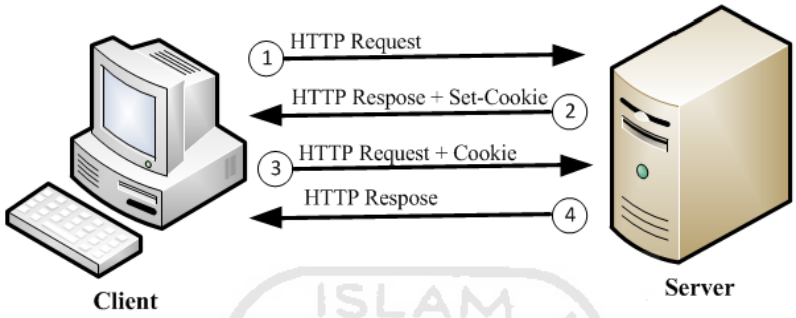
Informasi yang disimpan *session* disebut dengan *session id.* Dalam melakukan tugasnya *session* bisanya memanfaatkan *cookie* sebagai media bentuk penyimpanan data. Akan tetapi *session* tidak harus menggunakan *cookie*, karena masalah *privacy* terkadang *cookie* diblok oleh *user.* Selain menggunakan *cookie, session* dapat memanfaatkan URL (Uniform Resource Locator) sebagai tempat untuk menyimpan *session id.*

2.2.2.1 Cookie

Cookie dikembangkan oleh Netscape untuk pengolahan *state.* Akan tetapi sekarang *cookie* telah di *support* oleh banyak *browser* lain. *Cookie* merupakan perluasan dari protokol HTTP. *Cookie* terdiri dari dua HTTP *header* yaitu "`Set-Cookie`" yang terdapat di pesan *response* pada HTTP *header* dan "`Cookie`" yang terdapat di pesan *request* pada HTTP *header* (Shiflett, 2005:47). Contoh sebagai berikut:

```
Set-Cookie: variable-cookie=value; path=/; domain=contoh.com
```

Cookie terbentuk ketika *client* mengirimkan *request* kepada *server*, kemudian *server* menambahkan *cookie* pada pesan *response. Cookie* yang diberikan server akan ditambahkan pada setiap *request.* Untuk lebih jelasnya tentang *request-response* HTTP menggunakan *cookie* bisa dilihat pada Gambar 2.3.



Gambar 2.3 HTTP Request-Respose + Cookie

Terdapat dua jenis tipe *cookie* yang dibedakan berdasarkan masa berakhirnya. Tipe *cookie* tersebut adalah (Gourley, 2002:264):

1. ***Session cookies***, *cookie* akan hilang sewaktu *browser* ditutup. Contoh: *cookie* yang digunakan untuk *shopping carts* pada *e-commerce.*
2. ***Persistent cookies***, masa berakhir *cookie* ini lebih lama karena *cookie* ini disimpan dalam *disk. cookie* ini tidak akan hilang ketika *browser* ditutup atau komputer di-*restart.* Contoh: opsi ”*Remember Me*” saat *login*.

2.2.2.2 URL Session

Selain menggunakan *cookie, session* juga bisa memanfaatkan media URL untuk menyimpan *session id.* Tidak seperti *cookie* yang menyembunyikan *session id* di dalam HTTP *header,* URL *session* menyimpan *session id* pada parameter URL sehingga terlihat jelas dan tidak bisa disembunyikan. Ketika *session id* disimpan di URL, maka *request* yang mengandung *session id* terdapat dalam variabel “*Referer”* pada HTTP *header*. Contohnya adalah sebagai berikut:

```
Referer: http://www.contoh.com/p.php?PHPSESSID=12345
```

Session id dikirimkan menggunakan *method* GET atau POST. *Method* GET mengirimkan data melalui *query string* atau parameter yang terletak setelah tanda "?" pada URL. *Method* POST mengirimkan data dalam *field* tersembunyi (Alshanetsky, 2005:115). Contoh permintaan GET dan POST adalah sebagai berikut:

Pengiriman data dengan GET

```
GET /index.php?parameter1=value1&paramete2=value2 HTTP/1.1
Host: contoh.com
```

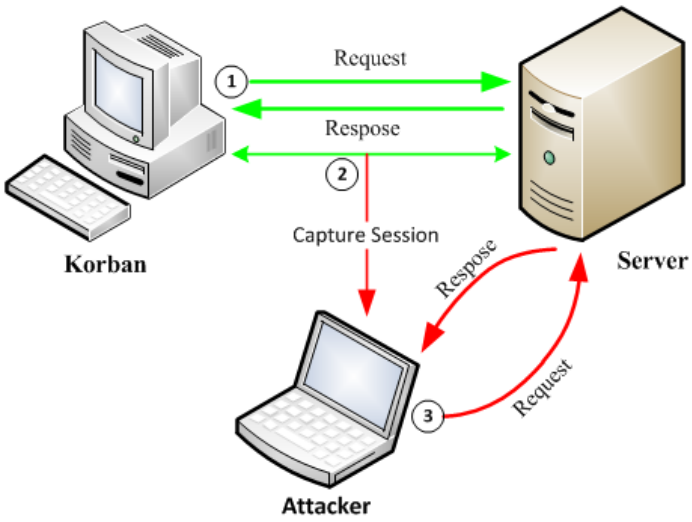
Data yang dikirimkan berupa parameter dan value yang terletak setelah tanda "?"

Pengiriman data dengan POST

```
POST /index.php HTTP/1.1
Host: contoh.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 7

parameter=value → Data yang dikiriman berupa parameter dan nilai
                  yang dilewatkan melalui hide field.
```

2.3 Session Hijacking

Session hijacking merupakan suatu teknik serangan dengan cara melakukan pembajakan terhadap *session id.* Pada level aplikasi (HTTP), serangan *session hijacking* tidak hanya dilakukan dengan membajak *session* yang sudah ada tetapi bisa juga dengan menciptakan *session* baru yang tidak sah menggunakan informasi yang telah didapatkan sebelumnya. Untuk lebih jelasnya tentang *session hijacking* bisa dilihat pada Gambar 2.4 (OWASP, 2009; EC-Council, V7).



Gambar 2.4 Serangan Session Hijacking

Dalam melakukan *session hijacking* biasanya seorang penyerang (*attacker*) berusaha untuk mendapatkan *session id* dari korban. Umumnya *session id* dapat ditemukan pada tiga lokasi sebagai berikut (OWASP, 2009; EC-Council, V7) :

1. Terdapat pada URL, yang diterima aplikasi melalui *request* HTTP GET ketika *client* membuka suatu halaman.
2. Di dalam *fields* dari sebuah *form* yang dikirimkan kepada aplikasi, biasanya *session id* berupa *field* tersembunyi yang disertakan pada HTTP POST.
3. Terdapat pada *Cookie.*

2.3.1 Teknik Session Hijacking

Dalam melakukan *hijacking* untuk mendapatkan *session id attacker* menggunakan beberapa teknik *sebagai* berikut (OWASP, 2009; EC-Council, V7):

a. Predict

Teknik ini kurang efektif dan tingkat keberhasilanya rendah karena teknik ini dilakukan dengan cara menebak *session id.* Tennik ini bisa dilakukan dengan menganalisis pola dari beberapa *session id.* Bila *session id*

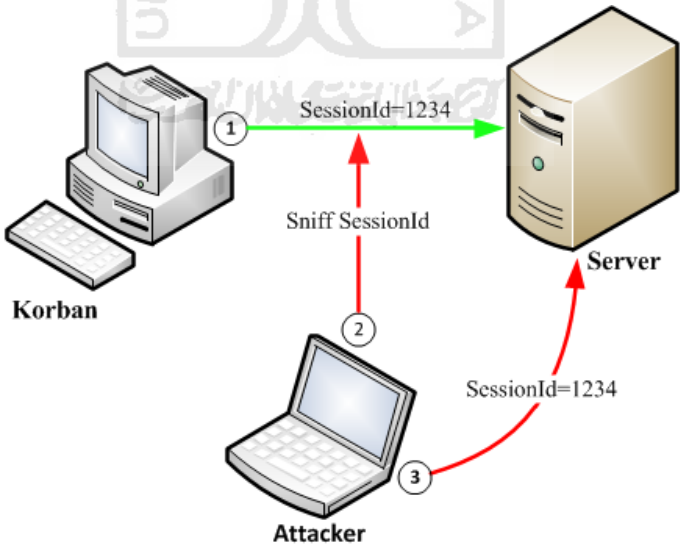
mempunyai pola tertentu dan tidak cukup acak kemungkinan *session id* dapat ditebak.

b. Brute Force

Teknik ini digunakan ketika *session id* sulit ditebak. Teknik ini dilakukan dengan cara mencoba semua kemungkinan *session id* yang mungkin digunakan. Teknik ini biasanya memanfaatkan suatu algoritma dan *tools* tertentu.

c. Pasif Sniffing

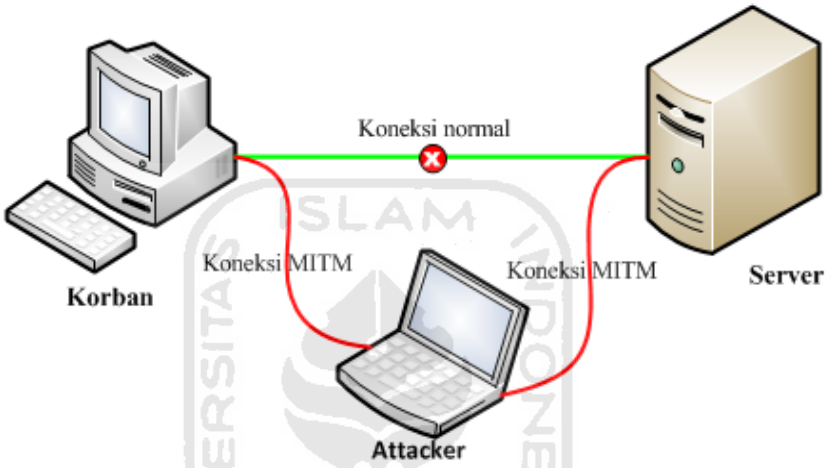
Sniffing merupakan teknik untuk menyadap paket yang lewat pada komunikasi yang dilakukan antara *client* dan *server*. Dalam melakukan *sniffing*, *ethernet card* harus dijalankan pada modus baca semuanya (*promiscuous*). Modus ini biasanya digunakan network administrator untuk analisis trafik jaringan. Dengan modus ini semua paket dapat dilihat baik itu paket dari atau untuk komputer sendiri ataupun komputer lain. Serangan *hijacking* dengan teknik *sniffing* dapat dilihat pada Gambar 2.5.



Gambar 2.5 Sniffing Session Id

d. Man-in-the-Midle (Active Sniffing)

Man-in-the-Midle (MITM) dilakukan dengan cara mengganggu koneksi antar korban dan *server*. Pada serangan MITM, *attacker* berusaha menjadi *proxy* yang terletak diantara korban dan *server*, sehingga semua paket komunikasi antara korban dan *server* akan dipaksa melalui komputer *attacker* terlebih dahulu. Ilustrasi MITM bisa dilihat pada Gambar 2.6.



Gambar 2.6 Serangan Man-in-the-Midle

e. Man-in-the-Browser (MITB)

Teknik ini mirip dengan MITM, hanya saja teknik ini menggunakan *trojan horse* untuk menginfeksi *browser* yang digunakan untuk mencegat, memanipulasi komunikasi dan mencuri informasi antara browser dengan mekanisme keamanan. Saat terjadi serangan MITB sebuah situs web bekerja secara normal sehingga korban tidak akan menyadari bahwa dia telah dimata-matai oleh *trojan* yang telah menginfeksi *browser*-nya.

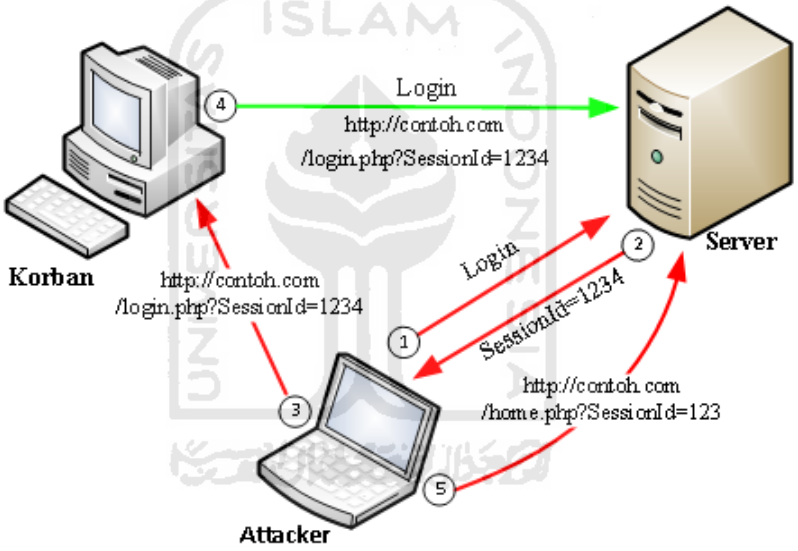
f. Cross Site Scripting (XSS)

XSS merupaka serangan yang bekerja pada sisi *client.* Serangan XSS dilakukan dengan menginjeksi *client side script* seperti *java script*, HTML atau *client script* yang lain. XSS terjadi karena validasi input yang tidak baik pada *form*, yang mengakibatkan *attacker* bisa memasukkan kode jahatnya pada *form*. Dalam *hijacking* serangan XSS biasanya bertujuan

untuk mendapatkan *cookie* yang tersimpan pada server, misalnya dengan memanfaatkan fungsi *Java Script* seperti "`document.cookie`".

g. Session Fixation

Session fixation merupakan serangan yang dilakukan dengan cara memaksa korban untuk menggunakan *session id* yang dipilih oleh *attacker*. Serangan *session fixation* dilakukan sebelum korban login. *Attacker* sebelumnya harus memilih *session id* yang dikenali *server* kemudian memaksa korban untuk menggunakan *session id* tersebut. Untuk ilustrasi *session fixation* bisa dilihat pada Gambar 2.7.



Gambar 2.7 Serangan Session Fixation

2.4 Autentikasi

Autentikasi merupakan metode untuk membuktikan apakah *user* yang melakukan akses terhadap sistem benar-benar *user* yang berhak. Dalam suatu proses autentikasi terdapat syarat mutlak yang harus terpenuhi yaitu *password* yang dikirimkan *user* sama dengan *password* yang terdapat pada *server*.

Terdapat banyak metode yang dapat digunakan untuk proses autentikasi. Berikut ini adalah tiga kategori metode autentikasi yang sering digunakan (Oni, 2010).

1. Something You Know

   Metode ini merupakan metode yang paling sering digunakan. Metode ini sangat tergantung pada kerahasian informasi karena metode ini berasumsi bahwa tidak ada seorangpun yang tahu kecuali seorang itu sendiri. Contoh autentikasi dengan metode ini adalah penggunaan *password* dan PIN.

2. Something You Have

   Metode ini biasanya digunakan sebagai faktor tambahan untuk membuat autentikasi lebih aman. Biasanya dengan memanfaatkan *hardware* tambahan yang bersifat unik seperti *smart card, usb token, hardware token*. Metode ini berasumsi bahwa tidak ada yang memiliki *hardware* tesebut keculi orang tersebut.

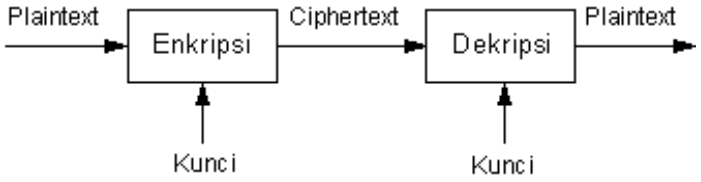
3. Something You Are

   Metode ini berkaitan erat dengan biometric, metode ini memanfaatkan keunikan yang terdapat pada bagian tubuh manusia yang tidak mungkin terdapat pada tubuh orang lain. Contohnya adalah penggunaan sidik jari, retina mata, suara dan wajah untuk autentikasi.

2.5 Kriptografi

Kriptografi berasal dari bahasa Yunani *"criptos"* yang berarti rahasia dan *"graphein"* yang berarti tulisan. Jadi kriptografi berarti tulisan rahasia (Munir, 2006:2). Kriptografi dulu digunakan untuk keamanan komunikasi dikalangan militer. Seiring dengan perkembangannya kriptografi sekarang berkembang menjadi disiplin ilmu tersendiri. Kriptografi tidak hanya dianggap sebagai ilmu akan tetapi juga dianggap sebagai seni untuk menjaga keamanan pesan atau data. Saat ini kriptografi telah banyak diimplementasikan dalam kehidupan sehari-hari. Seperti untuk transaksi di ATM, *Internet banking*, *e-commerce* dan lain sebagainya.

Proses kriptografi diawali dari *plaintext* atau pesan yang terbaca kemudian akan dienkripsi dengan fungsi dan algoritma enkripsi tertentu sehingga menghasilkan *chipertext* atau pesan yang terenkripsi. Agar dapat dibaca kembali tentunya *chipertext* harus diubah menjadi *plaintext*, proses pengubahan tersebut disebut dengan proses dekripsi. Ilustrasi proses kriptografi sederhana bisa dilihat pada Gambar 2.8.



Gambar 2.8 Proses kriptografi sederhana

BAB III

ANALISIS DAN DESAIN

3.1 Analisis Sistem

3.1.1 Analisis Masalah

Dalam analisis sistem ini, beberapa masalah keamanan yang terdapat pada aplikasi berbasis web:

1. Kelemahan mekanisme *session* pada aplikasi berbasis web. Pemakaian *session* hanya dengan fungsi session_start() dalam PHP merupakan kelemahan (*vulnerability*) yang dapat membuka celah untuk serangan *session hijacking*.
2. Kurangnya mekanisme autentikasi untuk membatasi hak akses *attacker* ketika *attacker* berhasil melakukan serangan *hijacking* pada aplikasi berbasis web.

Dari kedua masalah diatas, untuk melindungi aplikasi berbasis web dari serangan *session hijacking* perlu dibuat mekanisme keamanan sebagai berikut:

a. Mekanisme manajemen *session.*

   Mekanisme manajemen *session* memiliki fitur sebagai berikut:

   1. Cek waktu aktifitas terakhir *user*. Ketika *user* tidak melakukan aktivitas dalam jangka waktu yang ditentukan *session user* akan di hapus.
   2. Membangkitkan *session id* baru pada setiap *request.* Sehingga setiap *request* mempunyai *session id* yang berbeda-beda dengan tujuan untuk mempersingkat validitas waktu penggunaan *session id*.
   3. Melakukan autentikasi pada setiap *request* dengan memanfaatkan informasi *user agent browser* dan informasi *host* dari HTTP *referer*

(*fingerprint*). Sehingga ketika *attacker* berhasil mendapatkan *session id*, *session id* tersebut tidak akan bisa digunakan jika HTTP *header* dari *request* yang dilakukan *attacker* tidak memiliki informasi *user agent browser* dan HTTP *referer* yang sama dengan HTTP *header* korban.

b. Mekanisme autentikasi *challenge/response*.

Autentikasi *challenge/response* digunakan sebagai mekanisme keamanan tambahan untuk membatasi hak akses *attacker* terhadap transaksi-transaksi yang penting ketika *attacker* berhasil melakukan *hijacking*. Metode *challenge/response* digunakan karena metode ini memiliki mekanisme keamanan yang kuat. Nilai *response* hanya bisa digunakan untuk autentikasi terhadap nilai *challenge* tertentu dan nilai *challenge* tersebut akan selalu berubah sehingga sangat sulit bagi *attacker* untuk mendapatkan nilai *challenge* atau *response*.

Mekanisme autentikasi *challenge/response* terbagi menjadi dua mekanisme yaitu mekanisme pada sisi *server* yang digunakan untuk membangkitkan nilai *challenge* dan autentikasi nilai *response* dan mekanisme pada sisi *client* (*mobile* token) yang digunakan untuk membangkitkan nilai *secret key* dan nilai *response*. Autentikasi *challenge/response* memiliki fitur sebagai berikut:

1. Untuk membangkitkan nilai *response* menggunakan *software* token yang dibuat berbasis *mobile* (*mobile* token).
2. Cek kesalahan autentikasi nilai *response*. Jika jumlah kesalahan autentikasi mencapai jumlah maksimal yang ditentukan *user* akan diarahkan ke form login.
3. Nilai *response* hanya bisa digunakan satu kali.
4. Pada setiap transaksi nilai *challenge* selalu berubah.

3.1.2 Perangkat Keras dan Perangkat Lunak Yang Dibutuhkan

Dalam penelitian ini perangkat keras dan perangkat lunak yang digunakan sebagai berikut:

a. Kebutuhan perangkat keras

   Penelitan yang dilakukan membutuhkan seperangkat komputer yang akan digunakan sebagai web *server*, komputer *client* atau korban sekaligus komputer *attacker*.

b. Kebutuhan perangkat lunak

   1. Windows 7 dan XP sebagai sistem operasi.
   2. XAMPP yang digunakan sebagi web *server* dan *server* basisdata.
   3. Android SDK, Phonegap dan Eclipse digunakan untuk membuat aplikasi *mobile* token.
   4. VirtualBox digunakan untuk membuat virtual komputer.
   5. Mozilla firefox dan Google crome digunakan sebagai web *browser*.
   6. Wireshark digunakan untuk *sniffing*.
   7. *Add-on* Web Developer dari Mozilla firefox digunakan manajemen *cookie*.
   8. *Add-on* Live HTTP Header dari Mozilla firefox digunakan untuk melihat HTTP *request* dan HTTP *response* pada saat melakukan *hijacking* dengan teknik *session fixation*.

3.2 Desain Sistem

3.2.1 Activity Diagram

Activity diagram merupakan diagram yang menggambarkan aktivitas yang dilakukan *user* dan sistem. *Activity diagram* akan menunjukan bagaimana aktivitas dimulai dan bagaimana aktivitas berakhir.

3.2.1.1 Mekanisme Manajemen Session

Activity diagram mekanisme manajemen *session* yang digambarkan pada Gambar 3.1 menggambarkan aktivitas yang dilakukan *user* dan sistem. Aktivitas yang dilakukan *user* dan sistem sebagai berikut:

a. User

1. *Login*: *user* melakukan autentikasi denga *user name* dan *password* (*log-in*).
2. Halaman *user*: halaman yang hanya boleh diakses oleh *user* tertentu yang mempunyai hak akses.
3. *Request*: *user* melakukan permintaan kepada *server* dari halaman *user.*
4. *Log-out*: *user* menghapus *session* dan keluar dari halaman *user.*
5. Form login: halaman untuk autentikasi *user name* dan *password user* atau *log-in user*.

b. Sistem

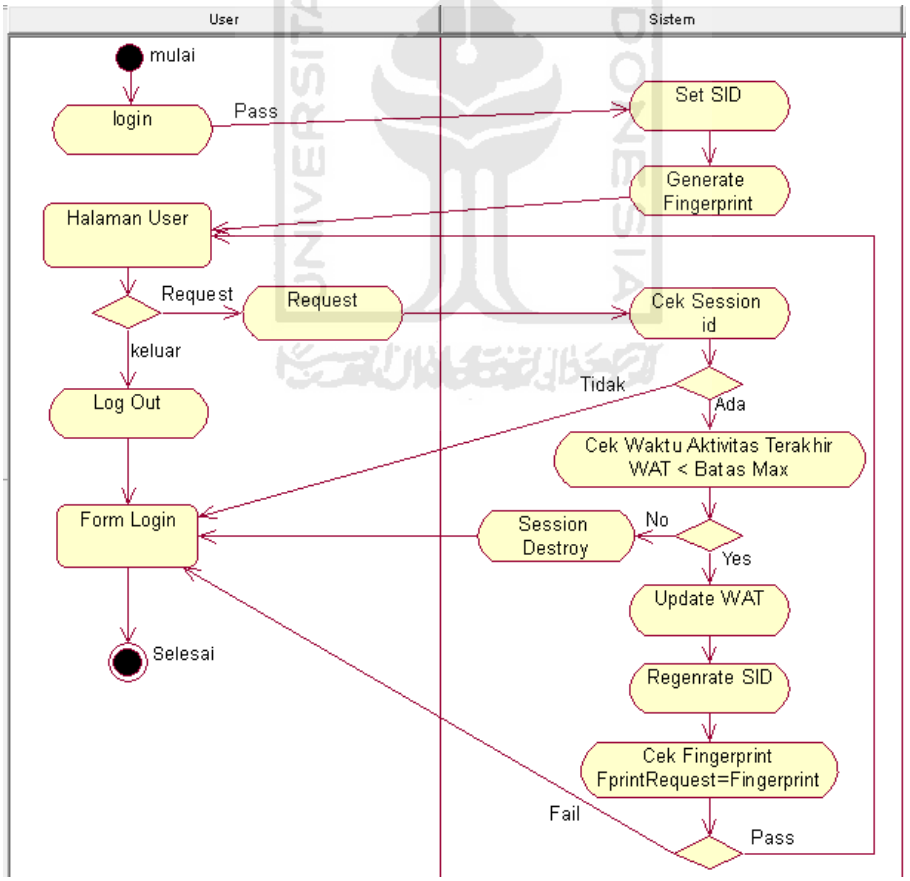
1. Set SID: membangkitkan *session id* dan akan diberikan kepada *user* dalam halaman *user*.
2. *Generate fingerprint*: membangkitkan nilai *fingerprint* yang akan digunakan untuk autentikasi nilai *fingerprint* dari *request.* Nilai *fingerprint* dibangkitkan dari infomasi *user agent* dan *server name* yang akan disimpan dalam variabel *session* untuk autentikasi *request* yang dilakukan *user*.
3. Cek *session id*: cek apakah *request* dari *user* mengandung *session id* atau tidak.

4. Cek waktu aktivitas terakhir: cek waktu aktivitas *user*, ketika *user* tidak melakukan aktivitas selama waktu yang ditentukan (*inactivity*) *session user* akan dihapus (*session destroy*).

5. *Update* WAT: menghitung ulang (*update*) waktu aktivitas terakhir *user*.

6. *Regenerate* SID: membangkitkan *session id* baru untuk *user* dan menhapus *session id* yang lama. Sehingga setiap *request* memiliki *session id* yang berbeda.

7. Cek *fingerprint*: membangkitkan nilai *fingerprint* untuk *request* dari informasi *user agent* dan HTTP *referer* dan melakukan autentikasi dengan nilai *fingerprint* pada poin dua.



Gambar 3.1 Activity Diagram Mekanisme Manajemen Session

3.2.1.2 Mekanisme Autentikasi Challenge/Response

Activity diagram autentikai *challenge/response* yang digambarkan pada Gambar 3.2 menggambarkan aktivitas yang terjadi pada saat *user* melakukan autentikasi nilai *response* yang dihasilkan dari *mobile* token. Aktivitas yang dilakukan *user* dan sistem sebagai berikut:

a. User

   1. *Request*: *request* dari *user* yang membutuhkan autentikasi *challenge/ response.*

   2. Form autentikasi: halaman untuk autentikasi nilai *response* dari *user*.

   3. Input *response*: *user* melakukan autentikasi terhadap nilai *challenge* tertentu dengan memasukan nilai *response* yang didapatkan dari *mobile* token pada form autentikasi.

b. Sistem

   1. *Generate challenge*: sistem membangkitkan nilai *challenge* yang bersifat acak dan diberikan kepada user dalam form autentikasi.

   2. *Generate response server*: sistem membangkitkan nilai *response* dari informasi EPOH, *seckret key* dan *challenge*. Nilai *response server* dibangkitkan secara berulang dalam selang waktu toleransi yang diberikan kepada sistem dan dalam periode 10 detik sekali.

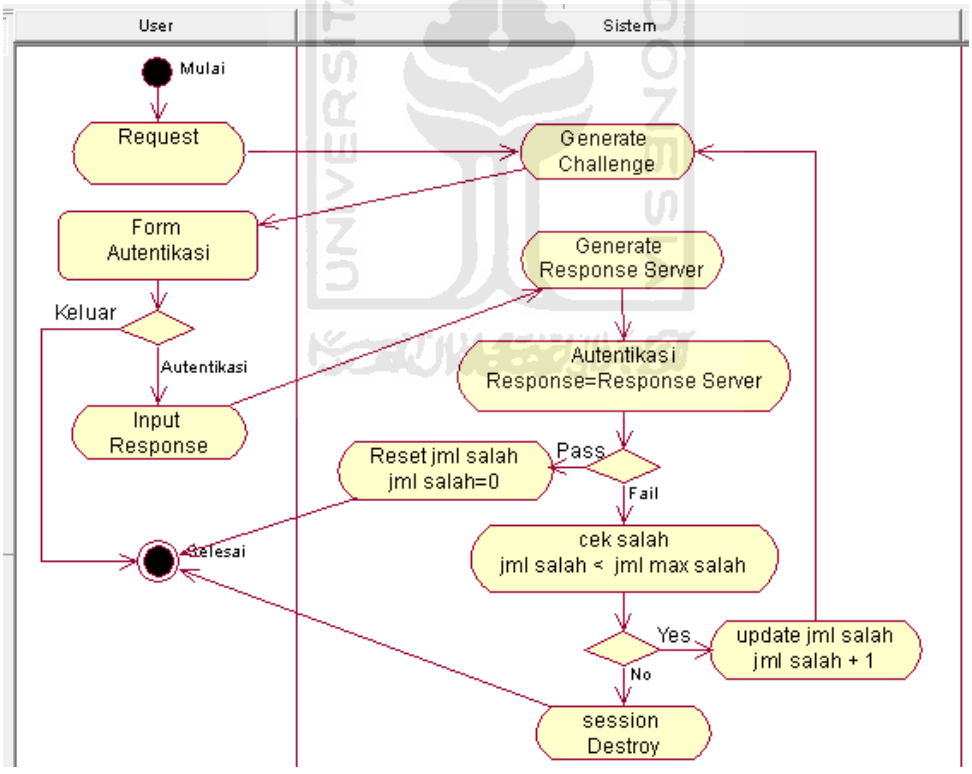
   3. Autentikasi: autentikasi nilai *response* dari *user* (*mobile* token) dengan nilai *response server* pada poin dua. Jika dalam selang waktu toleransi terdapat nilai *response* yang dibangkitkan sistem yang sama dengan nilai *response* dari *user* autentikasi berhasil.

   4. Reset jml salah: jika autentikasi berhasil sistem akan mereset jumlah salah menjadi 0.

5. Cek salah: jika terjadi kesalahan autentikasi, jumlah kesalahan akan di cek. Ketika jumlah kesalahan mencapai batas yang ditentukan *session user* dihapus. Sebaliknya ketika jumlah kesalahan kurang dari batas yang ditentukan, sistem menghitung kembali jumlah kesalahan autentikasi (*update*).

6. Update jml salah: jika terjadi kesalahan autentikasi dan kesalahan autentikasi kurang dari batas yang ditentukan, sistem menghitug kembali jumlah kesalahan autentikasi dengan menambah satu pada jumlah salah.

7. *Session destroy*: Ketika jumlah kesalahan autentikasi mencapai batas yang ditentukan session user akan dihapus (*session destroy*).



Gambar 3.2 Activity Diagram Autentikasi Challenge/Response

3.2.1.3 Mobile Token

Activity diagram mobile token yang digambarkan pada Gambar 3.3 menggambarkan aktivitas yang terjadi pada saat *user* membangkitkan nilai *response* dengan *mobile* token. Aktivitas yang dilakukan *user* dan sistem sebagai berikut:

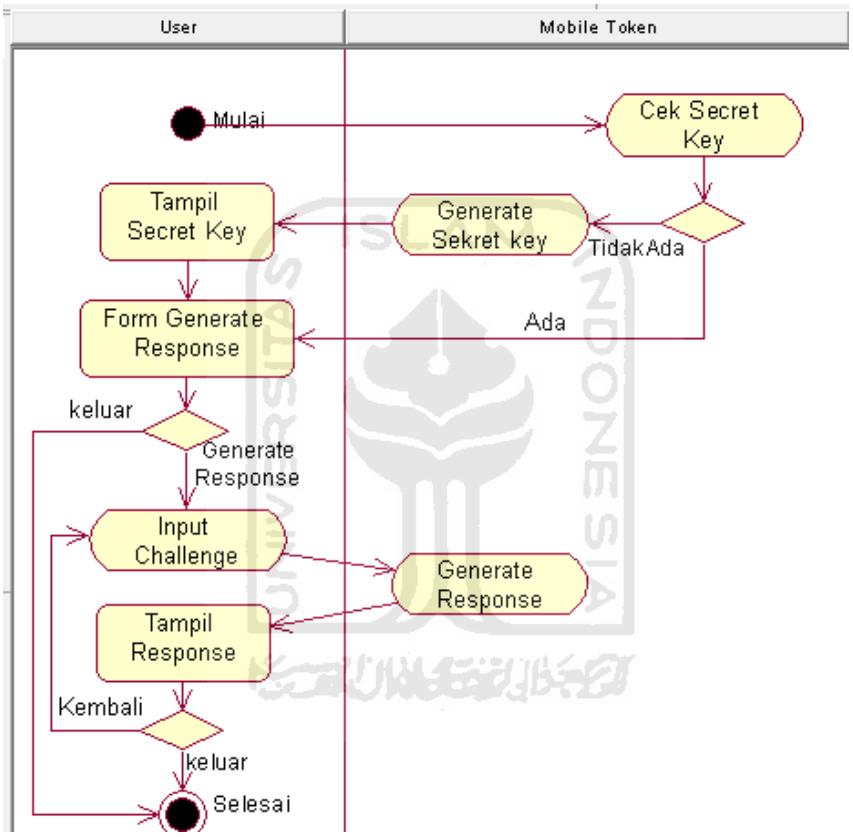
a. User

1. Tampil *secret key*: nilai inisialisasi user (*secret key*) yang di bangkitkan *mobile* token di tampilkan kepada *user*.
2. Form generate *response*: form yang digunakan *user* untuk mendapatkan nilai *response* dengan memasukkan nilai *challenge.*
3. Input *challenge*: untuk mendapatkan nilai *response* user harus memasukan nilai *challenge* pada form *generate*.
4. Tampil *response*: sistem menampilkan nilai *response* untuk nilai *challenge* yang diberikan *user.* Nilai *response* ini akan digunakan *user* untuk autentikasi dalam form autentikasi pada *activity diagram* autentikasi *challenge/response* Gambar 3.2.

b. Sistem

1. Cek *secret key*: pada saat *mobile* token dijalankan *secret key* dicek. Jika dalam *mobile* token belum terdapat *secret key*, *mobile* token akan membangkitkan nilai *secret key* berupa *string* acak. Jika dalam *mobile* token sudah terdapat *secret key*, *mobile* token akan menampilkan form untuk membangkitkan nilai *response*.
2. *Generate secret key*: Jika dalam *mobile* token belum terdapat *secret key mobile* token akan membangkitkan nilai *secret key* berupa *string* acak dan akan ditampilkan kepada user serta disimpan dalam *local storage mobile* token. Nilai *secret key* tersebut akan digunakan *user* untuk registrasi dan disimpan dalam *server* basisdata.

3. *Generate response*: setelah user memasukkan nilai *challenge* sistem akan membangkitkan nilai *response* untuk nilai *challenge* tersebut. Nilai *response* dibangkitkan dari informasi EPOH, *seckret key* dan *challenge.* Nilai *response* ini akan ditampilakn kepada *user* dan digunakan *user* untuk autentikasi dalam form autentikasi pada *activity diagram* autentikasi *challenge/response* Gambar 3.2.



Gambar 3.3 Activity Diagram Mobile Token

3.2.2 Algoritma

Teks algoritma adalah hasil pemikiran konseptual yang berisi langkah-langkah penyelesaian masalah yang mudah dimengerti dan dapat ditranslasikan ke dalam bahasa pemrograman (Munir, 2006:35,41). Dalam penelitian tugas akhir ini penulis menggunakan teks algoritma sebagai salah satu metode penyelesaian masalah. Algoritma yang dibuat sebagai berikut:

3.2.2.1 Algoritma Mekanisme Manajemen Session

Algoritma mekanisme manajemen *session* akan dibuat dengan bahasa pemrograman PHP. Adapun algoritmanya bisa dilihat pada Algoritma 3.1. algoritma ini digunakan dalam *activity diagram* manajemen *session* pada Gambar 3.1 untuk membangkitkan nilai *fingerprint*, autentikasi *fingerprint* untuk *request* dari *user* serta untuk cek waktu akativits terakhir (*inactivity*) *user*.

```
program sessionSecure
{program untuk manajemen session}

Deklarasi
salt, userAgent, servername, fingerprint,  referrer, url,
host, fprintRequest: string
last_activity , max, maxtime: integer

function regenerate_SID()
{fungsi untuk membangkitkan session id}
function Math.random()->integer
{membangkitkan nilai acak}
function expired(input max:integer)
{fungsi untuk cek inactivity user}
function MD5(input x : string)->string
{melakukan membangkitkan hash md5 dari x}
function parse_url(input referer : string)
{fungsi untuk mengubah string menjadi array}
function time()
{mengambil timestamp}
function session_destroy()
{hapus semua session user}

ALGORITMA
{membangkitkan fingerprint pada server}
read(salt)
userAgent<-$_SERVER[HTTP_USER_AGENT]
servername<-$_SERVER[SERVER_NAME]
fingerprint<-salt.userAgent.servername
temp<-fingerprint {simpan dalam variabel session, akan
                   digunakan untuk autentikasi informasi user
                   agent dan HTTP referer dari request}
return(fingerprint){kembalikan nilai finger print}

{autentikasi fingerprint}
If regenerate==true then {jika parameter regenerate id true}
regenerate_SID(){panggil fungsi untuk membagkitkan session
                 id baru}
endif
read(salt, fingerprint)
userAgent<-$_SERVER[HTTP_USER_AGENT]
referer<-$_SERVER[HTTP_REFERER]
url<-parse_url(referer) {ubah referer jadi array}
```

```
host<-url['host']  {ambil array host}
fprintRequest<-MD5(salt.userAgent.host) {membangkitkan
                                        fprintR}
if fprintRequest==fingerprint then {autentikasi
                  fprintRequest dengan fingerprint dari
                  variabel session}
   return true
else
   return false
endif

{cek waktu inactivity user}
read(max)       {baca waktu max inactivity}
maxtime<-max*60 {ubah waktu inactivity max jadi detik}
last_activity<-time() {simpan timestamp dalam variabel}
temp<-last_activity   {simpan dalam variabel session}
if time()-last_activity>maxtime then {cek inactivity}
   session_destroy(); {hapus session}
else{
   last_activity<-time(){simpan timestamp dalam variabel}
temp<-last_activity {update last_activiti dalam variabel
                    session}
endif
```

Algoritma 3.1 Algoritma Mekanisme Manajemen Session

3.2.2.2 Algoritma mekanisme Autentikasi Challenge/Response

Pada mekanisme autentikasi *challenge/response* akan dibuat dua mekanisme yaitu mekanisme pada sisi *server* dan mekanisme pada sisi *client* (*mobile* token). Untuk algoritmanya sebagai berikut:

c. Algoritma Autentikasi Challenge/Response

Pada sisi *server* dibuat menggunakan bahasa pemrograman PHP. Adapun algoritmanya bisa dilihat pada Algoritma 3.2. Mekanisme *challenge/response* ini akan diimplementasikan pada aplikasi berbasis web untuk autentikasi transaksi-transaksi yang penting. Algoritma ini digunakan untuk membangkitkan nilai *challenge*, membangkitkan dan autentikasi nilai *response* serta penanganan kesalaan autentikasi. Algoritma ini digunakan pada *activity diagram* autentikasi *challenge/response* pada Gambar 3.2.

```
program AutentikasiChallenge/Response
{Program untuk membangkitkan autentikasi challenge/response}

Deklarasi
T, Ttol, count, i : integer
```

```
n, hash, chall : String

function sha1(input x : string)->string
{membangkitkan hash sha1 dari x}
function MD5(input x : string)->string
{membangkitkan hash md5 dari x}
function strtolower(input x : string)->string
{membuat x menjadi lower case}
function rand()
{membangkitkan string random}
function substr(input a:string, input b,c:integer)->string
{mengambil a mulai dari b sebanyak c}
function gmdate()->integer
{menghitung waktu dalam format EPOH (jumlah detik sejak 1
januari 1970)}
function session_destroy()
{hapus semua session user}

ALGORITMA
read(T_tol, secretKey) {baca waktu toleransi dan nilai
                        inisislisasi user}
count<-0

{membangkitkan challenge}
hash<-sha1(uniqid(rand(), true))
n<-rand(1, 36) {membangkitkan angka acak dari 1 s/d 40-4}
chall<-substr(strtolower(hash, $n, 4)) {ambil 4 digit}
temp<-chall    {simpan chall dalam temp, variable session}
write(chall)   {berikan nilai challenge kepada user}

{membangkitkan response server dan autentikasi}
read(chall) {baca nilai chall dari temp}
T_tol<-T_tol*60  {T_tol dalam detik}
T<-gmdate() {hitung epoh dalam detik}
 {epoh jumlah waktu sejak 1 januari 1970 s/d waktu sekarang}
for i <- T-T_tol to i <= T+T_tol do
       {membangkitkan token berulang selam waktu toleransi
       sebelum dan setelah waktu sekarang}
    i<-i+1
    token<-substr(md5(substr(i,0,9)+secretKey+chall),0,6)
       {substr(i,0,9)= i/10,membangkitkan satu token per 10
detik}
       {substr(md5(…),0,6)=ambil 6 digit untuk token}
    read(res){baca nilai response dari user}
    if res==token then {autentikasi}
       count <- 0   {inisialisasi jml autentikasi gagal}
       return true  {jika berhasil nilai true}
    else
       count<-count+1 {hitung jml autentikasi gagal}
       temp<-count    {simpan count dalam variable session}
       return false   {autentikasi gagal nilai false}
    endif
endfor

{penenganan kesalahah autentikasi}
read (jml, count)
```

```
if count+1>jml then   {cek jml salah}
    session_destroy() {hapus session user}
endif
```

Algoritma 3.2 Algoritma Autentikasi Challenge/Response

d. Algoritma Aplikasi Token (Mobile Token)

Mobile token dibuat menggunakan bahasa pemrograman *java script*. Adapun algoritmanya bisa dilihat pada Algoritma 3.3. Algoritma tersebut digunakan untuk membangkitkan nilai *secret key* yang merupakan nilai inisialisasi untuk *user* yang digunakan untuk membedakan *user* yang satu dengan *user* yang lain. Selain itu juga digunakan untuk membangkitkan nilai *response* yang digunakan *user* untuk autentikasi.

```
program tokenClient
{program untuk membangkitkan sekret key dan response pada
client}

Deklarasi
Chars, randomstring, res: string
string_length, rnum, i, T: integer
Time : date

function Math.random()->integer
{membangkitkan nilai acak}
function getTime()->integer
{menghitung waktu dalam format EPOH dalam milisecond}
function Math.floor(input x:float)->integer
{membuat nilai integer atau membulatkan nilai kebawah}
function MD5(input x : string)->string
{membangkitkan hash md5 dari x}
function substr(input a:string, input b,c:integer)->string
{mengambil a mulai dari b sebanyak c}

ALGORITMA
{membangkitkan secret key}
chars <- "0123456789abcdefghijklmnopqrstuvwxyz"
string_length <- 8 {inisialisasi panjang string}
for i <- 0 to i < string_length do {perulangan sampai 8}
     i<-i+1
        {membangkitkan angka acak dari 0 s/d panjang chars}
     rnum <- Math.floor(Math.random() * chars.length)
        {ambil satu digit dari chars}
     randomstring <- chars.substring(rnum,rnum+1)
endfor
secretKey <- randomstring {simpan randomstring di local
                           storage sebagai variabel sekretKey}
write(randomstring){tampilkan secret key untuk user}

{membangkitkan response}
```

```
read(chall, secretKey)
T<-getTime() {EPOH dalam mili second}
  {epoh jumlah waktu sejak 1 januari 1970 s/d waktu sekarang}
T<-T/10000 {EPOH dalam second / 10 second}
T<-Math.floor(T) {bulatkan angka ke bawah}
res<-MD5(T+chall+secretKey)) {membangkitkan response}
return res.substr(hash, 0, 4) {ambil 4 carakter awal}
write(res) {tampilkan response}
```

Algoritma 3.3 Algoritma Mobile Token

3.3 Rencana Pengujian

Pengujian dilakukan untuk mengetahui apakah mekanisme keaman yang dibuat dapat bekerja sesuai dengan tujuan serta untuk mendapatkan mekanisme kemanan dengan tingkat kesalahan minimal. Dalam penelitian ini pengujian akan dilakukan dalam dua tahap, yaitu pengujian implementasi dan *penetration testing.*

3.3.1 Implementasi Pada Aplikasi Berbasis Web

Pada tahap ini mekanisme keamanan akan diimplementasikan pada suatu aplikasi berbasis web sederhana, dengan tujuan untuk mengetahui apakah mekanisme keamanan dapat diimplementasikan dan dapat bekerja pada aplikasi berbasis web.

3.3.2 Penetration Testing

Penetration testing merupakan pengujian dengan melakukan penetrasi atau serangan menggunakan beberapa teknik *hijacking*. pengujian ini dilakukan dengan tujuan untuk mengetahui tingkat keamanan dari sistem yang menggunakan mekanisme keamanan yang telah dibuat. Pengujian akan dilakukan terhadap empat buah sistem sebagai berikut:

1 Sistem yang hanya menggunakan autentikasi *user name* dan *password* tanpa mekanisme keamanan lain.
2 Sistem yang menggunakan autentikasi *user name* dan *password* serta URL *session.*

3 Sistem yang menggunakan autentikasi *user name* dan *password* serta *cookie session.*
4 Sistem yang menggunakan autentikasi *user name* dan *password* serta mekanisme keamana yang telah dibuat yaitu manajemen *session* dan autentikasi *challenge/response.*

Sistem pertama sampai ketiga akan digunakan sebagai pembanding. Sistem-sitem tersebut dipilih sebagai pembanding karena ketiga sistem tersebut memiliki mekanisme keamanan yang berbeda, sehingga diharapkan bisa menjadi tolak ukur untuk pengujian mekanisme keamanan yang dibuat.

Beberapa teknik *hijacking* yang digunakan pada pengujian tahap ini sebagai berikut:

1. Sniffing

   Sniffing akan dilakukan menggunakan wireshark, *sniffing* dipilih dalam pengujian ini karena teknik ini sangat efektif untuk melihat semua paket yang lewat pada lalu lintas jaringan. Termasuk *request* dan *response* HTTP yang didalamnya terdapat *session id.*

2. Man-in-the-Midle

   Man-in-the-Midle akan dilakukan dengan *add-on* firesheep dari Mozilla firefox. Firesheep dipilih karena bisa menampilkan daftar *user* yang melakukan autentikasi *user name* dan *password* dalam *slide bar* firesheep dan situs web yang bersangkutan bisa dibuka dari daftar tersebut.

3. Session Fixation

   Teknik ini dipilih karena teknik ini sangat berbahaya. Teknik ini bisa dilakukan dengan jarak jauh dan teknik ini merupakan salah satu teknik yang tidak bisa dicegah dengan penggunaan protokol HTTPS.

BAB IV

HASIL DAN PEMBAHASAN

4.1 Implementasi Sistem

Tahap implementasi sistem merupakan tahap di mana perancangan diimplementasikan ke dalam bahasa pemrograman sehinnga dapat digunakan pada kondisi sebenarnya. Implementasi mekanisme manajemen *session* dan autentikasi *challenge*/*response* adalah sebagai berikut.

4.1.1 Mekanisme Manajemen Session

Mekanisme manajemen *session* diimplementasikan dalam bahasa pemrograman PHP. Mekanisme tersebut dibuat dalam satu kelas dengan nama sessionSec yang di dalamnya dibuat enam fungsi sebagai berikut:

1. __construct()

   Fungsi ini berjalan ketikan pembuatan objek baru dari kelas sessionSec. Fungsi ini bertugas untuk inisialisasi dan memanggil fungsi axpired() untuk cek waktu maksismal *session* ketika *user* tidak melakukan aktivitas. Fungsi ini juga digunakan untuk menambahkan beberapa konfigurasi *runtime* PHP serta untuk memulai *session.*

2. sfpSet()

   Fungsi ini bertugas untuk inisialisasi nilai *salt* yang merupakan suatu *string* rahasia yang digunakan untuk memperkuat hasil enkripsi. Fungsi ini juga memanggil fungsi fprint() untuk membangkitkan nilai *fingerprint* dan akan menyimpannya dalam variabel *session* yang akan digunakan untuk autentikasi nilai *fingerprint* pada setiap *request.* Fungsi ini juga memanggil fungsi regenerate_SID() untuk membangkitkan *session id* baru pada setiap *request.*

3. fprint()

   Fungsi ini digunakan untuk membangkitkan nilai *fingerprint* pada *server* dengan MD5 dari *salt, user agent* dan *server name* yang akan digunakan untuk autentikasi setiap *request*. *Fingerprint server* dibangkitkan dengan fungsi berikut.

```
$fingerprint=MD5($salt.$_SERVER[‘HTTP_USER_AGENT’].$_SERVER[
‘SERVER_NAME’);
```

4. cekFprint()

   Fungsi ini bertugas untuk membaca informasi *user agent* dan HTTP *referer* dari setiap *request*, kemudian akan membangkitkan nilai *fingerprint* untuk *request* dan melakukan autentikasi dengan nilai *fingerprint* yang dihasilkan oleh fungsi fprint() yang telah disimpan dalam variabel *session*. *Fingerprint* untuk *request* dibangkitkan dengan fungsi berikut.

```
$url=parse_url($_SERVER['HTTP_REFERER']);
$host=$url['host']; //ambil host daari referer
$fingerprintRequest=MD5($salt.$_SERVER['HTTP_USER_AGENT'].$h
ost);
```

5. expired()

   Fungsi ini digunakan untuk menghapus *session id* dan mengarahkan *user* ke form login jika *user* tidak melakukan aktivitas dalam jangka waktu yang ditentukan. Script untuk cek waktu aktivitas *user* sebagai berikut.

```
$maxtime=$max*60; // waktu max inactivity dalam detik
if(empty($_SESSION['last_activity'])){
    $_SESSION['last_activity']=time();// catat timestamap
}
if(time()-$_SESSION['last_activity']>$maxtime){
    session_destroy(); //hapus session user
}
else{
    $_SESSION['last_activity']=time();//update last activity
}
```

6. regenerate_SID()

   Fungsi untuk membangkitkan *session id* baru untuk *user*.

Dalam aplikasi berbasis web, implementasi mekanisme manajemen *session* sebagai berikut:

a. Proses login

1. Sertakan file sessionSec.class.php.

```
include '../sessionSec.class.php';
```

2. Buat *object* dari kelas sessionSec setelah autentikasi *user name* dan *password* sukses. Selain itu panggil fungsi sfpSet() untuk konfigurasi *runtime* PHP dan membangkitkan nilai *fingerprint server*.

```
$obj=new sessionSec($max);
$obj->sfpSet($salt);
```

$max, bernilai *integer* dalam menit yang merupakan nilai batas maksimal waktu *inactivity user*. Nilai *default* adalah 0.

$salt, bernilai *string* yang digunakan untuk membangkitkan nilai *fingerprint*. Nila *default* adalah 'salt'.

b. Halaman yang membutuhkan session *user*.

Pada setiap halaman yang memebutuhkan *session*, mekanisme manajemen *session* akan digunakan untuk cek waktu *inactivity user*, membangkitkan *session id* baru dan autentikasi nilai *fingerprint* dari setiap *request*. Untuk implementasinya sebagai berikut:

1. Sertakan file sessionSec.class.php.

```
include '../sessionSec.class.php';
```

2. Buat *object* dari kelas sessionSec

```
$obj=new sessionSec($max);
//$max=max waktu inactivity user
```

3. Cek *fingerprint* untuk request.

Cek *fingerprint* pada *request* bisa dilakukan dengan memasukkan fungsi cekFprint() sebagai kondisi dalam statemen if pada PHP. Contohnya sebagai berikut:

```
if(!$obj->cekFprint()){ //autentikasi
   header('Location:login-form.php'); //redirect
}
```

Script diatas akan mengarahkan *user* ke halaman form-login.php. ketika autentikasi *fingerprint request* gagal. Untuk kondisi `!$obj->cekFprint()` bisa diganti dengan `$obj->cekFprint()`, `$obj->cekFprint()==true` atau `$obj->cekFprint()==false`. Jika ingin menggunakan fitur untuk membangkitkan *session id* baru bisa memberikan nilai *true* untuk parameter pada fungsi `cekFprint()` sehingga menjadi `cekFprint(true)`.

Dengan memanfaatkan mekanisme manajemen *session* tersebut setiap *request* akan memiliki satu *session id* yang berbeda. Selain itu ketika *user* tidak melakukan aktivitas selama batas waktu yang ditentukan ($max), *session id* akan dihapus. Jika cek *fingerprint* gagal, *user* akan diarahkan ke form login. Sehingga jika *session id* berhasil dicuri *attacker, session id* tidak bisa digunakan selama HTTP *header* dari *request* yang dilakukan *attacker* tidak memiliki informasi *user agent browser* dan HTTP *referer* yang sama dengan informasi yang terdapat pada HTTP *header* korban.

4.1.2 Mekanisme Autentikasi Challenge/Response

Mekanisme autentikasi *challenge*/*response* diimplementasikan dalam dua mekanisme yaitu pada *server* dan *client.* Pada *server* mekanisme *challenge*/ *response* diimplementasikan dalam aplikasi berbasis web menggunakan bahasa pemrograman PHP. Sedangkan pada disisi *client* diimplementasikan pada aplikasi *mobile* (*mobile* token). *Mobile* token *dibuat* menggunakan *phonegap* yang berfungsi untuk membangkitkan nilai *secret key* dan nilai *response.*

4.1.2.1 Mekanisme Challenge/Response

Mekanisme autentikasi *challenge*/*response* dibuat dalam dua kelas. Pertama kelas token yang diguankan untuk membangkitkan nilai *challenge* dan nilai *response server* serta melakukan autentikasi nilai *response* dari *user*. Kelas

kedua adalah kelas enkripDekrip yang digunakan untuk enkripsi nilai *secret key* sebelum disimpan dalam basisdata dan untuk dekripsi nilai *secret key* sebelum digunakan. Fungi-fungsi yang dibuat dalam mekanisme ini sebagai berikut:

1. __construct()

   Fungsi ini untuk memberikan inisialisasi waktu toleransi yang akan digunakan untuk membangkitkan nilai *response* pada *server*. Seain itu fungsi ini juga akan memanggil fungsi `session_start()` untuk memulai *session* jika *session* belum dijalankan.

2. genChal()

   Fungsi untuk membangkitkan nilai *challenge* untuk diberikan kepada *user*. Membangkitkan nilai *challenge* menggunakan fungsi SHA1 untuk lebih jelasnya sebagi berikut.

```
$hash = sha1(uniqid(rand(), true));
$n = rand(1, 36); //membangkitkan angka acak dari 1 s/d 40-4
$challenge = substr(strtolower($hash), $n, 4); //ambil 4
```

3. cekRes()

   Fungsi ini digunakan untuk membangkitkan nilai *response* pada *server* (`$token`) dan melakukan autentikasi dengan nilai *response* yang dikirimkan *user*. Untuk *script*-nya sebagai berikut.

```
$chal=trim($chal);//nilai challenge yang dibangkitkan server
$res=trim($res); //nilai response dari user
$secretkey=trim($secretkey); //nilai secret key user
$maxperiod = $this->Ttol*60; //toleransi waktu dalam detik
$time=gmdate("U"); //hitung nilai epoh dalam detik
for($i = $time - $maxperiod; $i <= $time + $maxperiod; $i++)
{
  //membangkitkan response di server
  $token= substr(md5(substr($i,0,9).$secretkey.$chal),0,6);
  if($res == $token)return true; //autentikasi
}
```

   `$maxperiod` merupakan waktu toleransi *server* untuk membangkitkan nilai *response* pada *server* (`$token`). Toleransi waktu ini digunakan untuk singkronisasi waktu antara *server* dan *client. Server* akan membangkitkan

nilai *response* pada *server* (`$token`) secara berulang pada selang waktu (`$maxperiod`) yang telah ditentukan. Jika dalam selang waktu tersebut terdapat nilai *response* yang sama dengan nilai *response* yang dikirimkan *user* autentikasi berhasil, sistem akan memberikan nilai *true* yang mengindikasikan autentikasi berhasil. Sebaliknya jika tidak ada nilai *response* yang sama dengan nilai *response* yang dikirimkan *user* sistem akan memberikan nilai *false* atau autentikasi gagal.

Selain toleransi waktu (`$maxperiod`), *server* juga diberi toleransi jangka waktu dalam membangkitkan *response* yaitu 10 detik sekali. Karena sangat sulil membuat waktu *server* dan *client* sama persis sampai pada tingkat detik. Selain itu agar nilai response server yang dibangkitkan tidak terlalu banyak sehingga bisa menghemat waktu autentikasi.

4. cekSlh()

   Fungsi ini digunakan untuk memberikan penanganan kesalahan autentikasi yang dilakukan *user*. Jika jumlah kesalahan sampai pada batas yang ditentukan fungsi ini akan menghapus *session user*.

5. resetCount()

   fungsi ini digunakan untuk mengembalikan jumlah kesalahan autentikasi menjadi 0 ketika autentikasi berhasil.

6. enkrip() dan dekrip()

   Fungsi ini berada pada kelas enkripDekrip yang digunakan untuk enkripsi dan dekripsi *secret key*. Fungsi ini dibuat dengan algoritma *rijndael* dengan memanfaatkan modul MCrypt PHP.

Implementasi sistem autentikasi *challenge/response* pada aplikasi berbasis web kelas enkripDekrip diimplementasikan pada proses login dan pada registrasi *user*. Sedangkan kelas token di implementasikan pada halaman atau transaksi-transaksi tertentu yang membutuhkan keamanan khusus. Untuk implementasinya sebagai berikut:

a. Pada proses login

1. Sertakan file token.class.php.

```
include '../token.class.php';
```

2. Buat objek dari kelas enkripDekrip setelah autentikasi *user name* dan *password* sukses. Selain itu panggil fungsi dekrip() untuk dekripsi *secret key user* dari basisdata.

```
$obj=new enkripDekrip('kunci');
  //buat objek baru dan inisialisasi kunci
$secret=$obj->dekrip('sekretkey'); //buat object baru
$_SESSION[secretkey] = $secret; //simpan secret key
   //dalam variabel session karena akan digunakan untuk
  // membangkitkan response pada server ($token)
```

b. Implementasi kelas enkripDekrip pada *register user*

Untuk melakukan enkripsi *secret key* sebelum disimpan ke basisdata bisa dilakukan dengan cara berikut.

```
 $obj=new enkripDekrip('kunci'); //buat object baru
 $secret=$obj->enkrip('sekretkey'); //ekripsi sekretkey
```

c. Implementasi kelas token untuk keamanan khusus

1. Sertakan file token.class.php.

```
include '../token.class.php';
```

2. Buat objek baru untuk kelas token.

```
$obj = new token($T_tol); //$T_tol= waktu toleransi dalam
                          //menit
```

3. Autentikasi *response.*

```
If (submit){ //jika user submit form autentikasi
//cek response yang dikirim user
$sukses = $obj->cekRes('challenge','response','secrekey');
  if ($sukses) { // jika sukses
      $obj->resetCount(); //reset jumlah salah ke 0
      //aksi yang ingin dilakukan
  else { //jika autentikasi gagal
      $obj->salah($max); //cek salah autentikasi $max=jml
                         // maksimal salah.
      //pesan kesalahan
  }
}
$obj->genChal() //membangkitkan challenge
//form token
```

4.1.2.2 Mobile Token

Pada sisi client dibuat *mobile* token yang digunakan untuk membangkitkan nilai *secret key*. *Secret key* merupakan nilai rahasia yang digunakan untuk inisialisasi *user* serta untuk membangkitkan nilai *response.* Dalam mekanisme ini terdapat dua fungsi utama yaitu genSK() dan genRes(). Fungsi genSK() digunakan untuk membangkitkan nilai *secret key* dengan mengambil secara acak delapan *digit string* dari angka 0 sapai 9 dan huruf kecil a sapai z. untuk *script*-nya sebagi berikut.

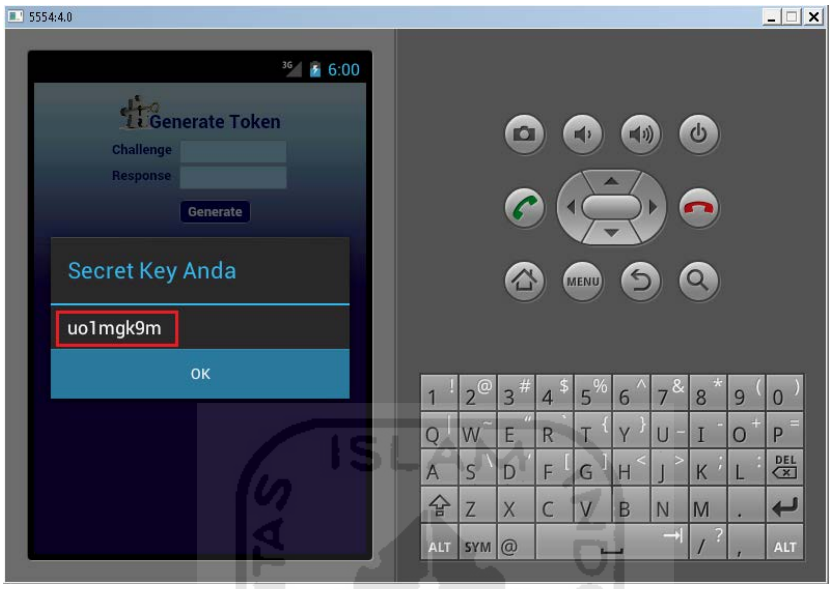
```
var char = "0123456789abcdefghiklmnopqrstuvwxyz";
var string_length = 8; //panjang string untuk secret key
var randomstring = '';
for (var i=0; i<string_length; i++) {//looping sampai 8 digit
     //membangkitkan angka acak
      var rnum = Math.floor(Math.random() * chars.length);
      //ambil satu digit karakter
     randomstring += chars.substring(rnum,rnum+1);
}
```

Fungsi untuk membangkitkan nilai *response* pada *client* hampir sama dengan fungsi untuk membangkitkan nilai *response* pada *server* (`$token`) perbedaanya adalah *response* pada *client* hanya dibangkitkan 10 detik sekali tanpa berulang. Untuk *script*-nya sebagai berikut.

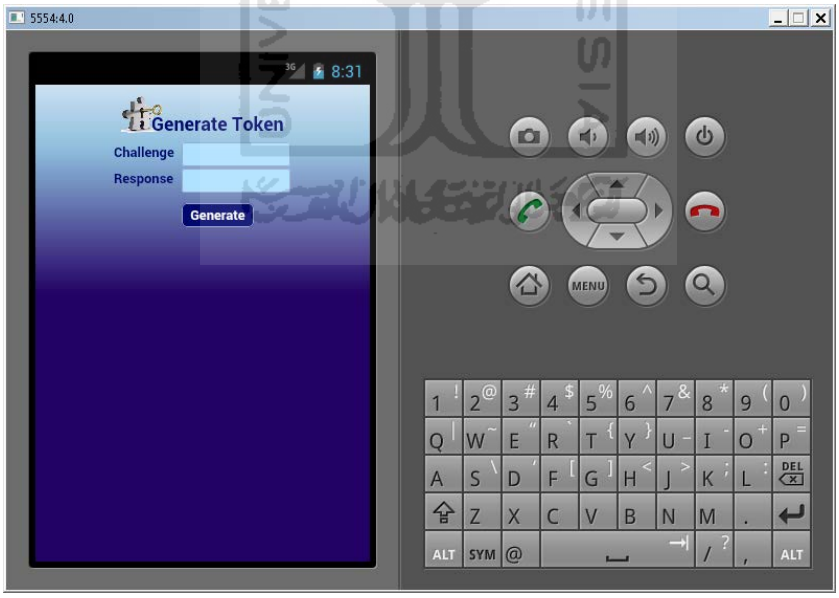
```
var time = new Date();
time = time.getTime() / 10000; //epoh dalam mili detik /1000
                               //dan membangkitkan per 10 detik /10
time = Math.floor(time); //bulatkan kebawah
var res = hex_md5(time + secret + chall);
//membangkitkan response
return res.substring(0,6); //ambil 6 digit karakter utk response
```

Implementasi *mobile* token bisa dilihat pada Gambar 4.1 yang menampilkan *secret key* untuk *user* pada saat pertamakali aplikasi dijalankan. Pada gambar tesebut *secret key* yang diberikan berupa delapan *digit* karakter. Untuk menggunakan aplikaasi token ini, nilai *secret key* harus disimpan pada *server* basisdata, karena nilai *secret key* ini digunakan untuk identifikasi *user* yang akan membedakan antara *user* yang satu dengan *user* yang lain. *Secret key* hanya ditampilkan sekali saat aplikasi pertama kali dijalankan. Setelah itu pada saat

dijalankan, aplikasi akan selalu menampilkan halaman untuk membangkitkan nilai *response* seperti pada Gambar 4.2.



Gambar 4.1 Prompt Secret Key User



Gambar 4.2 Halaman Untuk Membangkitkan Nilai Response

4.2 Pengujian Sistem

Pengujian dilakukan untuk mengetahui kinerja dari mekanisme keamanan yang telah dibuat. Pengujian ini dilakukan dalam dua tahap. Tahap pertama

mekanisme keamanan yang telah dibuat akan diimplementasikan pada aplikasi berbasis web. Pengujian pada tahap ini bertujuan untuk mengetahui apakah implementasi mekanisme keamanan dapat bekerja dengan baik pada aplikasi berbasis web. Pengujian tahap kedua adalah *penetration testing.* Pengujian ini dilakukan setelah mekanisme keamanan dapat berjalan dengan baik pada aplikasi berbasis web atau dengan kata lain telah melalui pengujian tahap pertama. Pengujian tahap kedua dilakukan untuk mengetahui tingkat keamanan dari sistem.

4.2.1 Implementasi Pada Aplikasi Berbasis Web

4.2.1.1 Mekanisme Manajemen Session

Pada pengujian tahap ini yang pertama dilakukan adalah identifikasi permasalahan yang mungkin terjadi pada sistem ketika diimplementasikan mekanisme manajemen *session*. Hasil identifikasi permasalahan tersebut akan menjadi materi pengujian sistem. Hasil identifikasi permasalahan mekanisme manajemen *session* ketika diimplementasikan pada aplikasi berbasis web sebagai berikut:

1. Apakah mekanisme manajemen *session* bisa diimplementasikan dan berjalan pada aplikasi berbasis web?
2. Apakah sistem bisa mempertahankan *session user* dari mulai *login* sampai *logout*?
3. Apakah fitur *back* dan *refresh* pada *browser* bisa digunakan?
4. Apakah *user* bisa membuka halaman dengan mengetikkan alamat di *address bar* pada *browser*?
5. Apakah fitur untuk membangkitkan *session id* baru berjalan?
6. Apakah fitur autentikasi *fingerprint* untuk *request* berjalan?
7. Apakah fitur *inactivity user* berjalan?

Hasil identifikasi masalah tersebut yang menjadi materi pengujian sistem pada tahap ini. Setelah beberapa kali melakukan pengujian berdasarkan identifikasi masalah tesebut hasil pengujian dapat dilihat pada Tabel 4.1.

Tabel 4.1 Hasil Pengujian Mekanisme Manajemen Session Tahap Satu

No	Masalah Yang Diujikan	Hasil
1	Apakah mekanisme manajemen *session* bisa diimplementasikan dan berjalan pada aplikasi berbasis web?	Berjalan
2	Apakah sistem bisa mempertahankan *session user* dari mulai *login* sampai *logout*?	Bisa
3	Apakah fitur *back* dan *refresh* pada *browser* bisa digunakan?	Bisa
4	Apakah *user* dapat membuka halaman dengan mengetikkan alamat alamat di *address bar* pada *browser?*	Tidak
5	Apakah fitur membangkitkan *session id* baru berjalan?	Berjalan
6	Apakah fitur autentikasi *fingerprint* untuk *request* berjalan?	Berjalan
7	Apakah fitur *inactivity user* berjalan?	Berjalan

Dari hasil pengujian pada Tabel 4.1, *User* tidak dapat membuka halaman dengan mengetikkan alamat alamat di *address bar* pada *browser* karena adanya autentikasi *fingerprint* yang menggunakan HTTP *referer*. Sehingga ketika *user* membuka halaman dengan mengetikakan alamat langsung di *address bar* pada *browser* dalam HTTP *header* dari *request* tersebut tidak mengandung HTTP *referer*.

Penerapan autentikasi menggunakan HTTP *referer* memang memiliki kekurangan yaitu *user* tidak bisa membuka halaman dengan mengetikkan alamat langsung di *address bar* pada *browser*. Tetapi autentikasi dengan HTTP *referer* sangat baik untuk melindungi web dari serangan *hijacking*. Karena dalam melakukan *hijacking* setelah mendapatkan *session id, attacker* biasanya langsung melakukan *request* tertentu dengan *session id* yang didapatkan dan *request* yang dilakukan *attacker* tidak akan memiliki HTTP *referer* sehingga *request* akan gagal.

4.2.1.2 Mekanisme Challenge/Response

Seperti pengujian pada mekanisme manajemen *session* di sini pengujian diawali dengan identifikasi masalah yang mungkin terjadi ketika implementasi mekanisme autentikasi *challenge/response* pada aplikasi berbasis web. Hasil identifikasi masalah adalah sebagai berikut:

1. Apakah mekanisme dapat diimplementasikan dan berjalan pada aplikasi berbasis web?
2. Apakah nilai *challenge* bisa dibangkitkan dan *challenge* hanya dapat dipakai satu kali?
3. Apakah autentikasi *response* dapat berjalan?
4. Apakah fitur penanganan kesalahan autentikasi dapat berjalan?

Implementasi sistem autentikasi *challenge/response* bisa dilihat pada Gambar 4.3. Gambar tersebut merupakan autentikasi ketika *user* akan menghapus data produk. Sebelum melakukan proses hapus data, sistem akan memberikan halaman autentikasi yang berisi nilai *challenge* dan sistem akan melanjutkan proses hapus data ketika autentikasi nilai *response* yang diberikan *user* sukses.

Gambar 4.3 Autentikasi Challenge/Response Pada Aplikasi Berbasis Web

Dalam mekanisme autentikasi *challenge/response* jika autentikasi gagal sistem akan menampilkan pesan kesalahan berwarna merah seperti pada Gambar 4.4. Dalam gambar tersebut, *user* telah melakukan satu kali kesalahan autentikasi

dengan batas maksimal tiga kali kesalahan. Jika *user* melakukan kesalahan sampai tiga kali, *user* akan diarahkan ke form login.

Gambar 4.4 Pesan Kesalahan Autentikasi Challenge/Response

Hasil pengujian autentikasi *challenge/response* secara keseluruhan bisa dilihat pada Tabel 4.2. Dari hasil pengujian tersebut bisa diambil kesimpulan bahwa mekanisme autentikasi *challenge/response* dapat di implementasikan dan berjalan pada aplikasi berbasis web.

Tabel 4.2 Hasil Pengujian Autentikasi Challenge/Response

No	**Masalah Yang Diujikan**	**Hasil**
1	Apakah mekanisme bisa diimplementasikan dan berjalan pada aplikasi berbasis web?	Bisa
2	Apakah nilai *challenge* bisa dibangkitkan dan *challenge* hanya dapat dipakai satu kali?	Bisa
3	Apakah autentikasi *response* berjalan?	Berjalan
4	Apakah fitur penanganan kesalahan autentikasi berjalan?	Berjalan

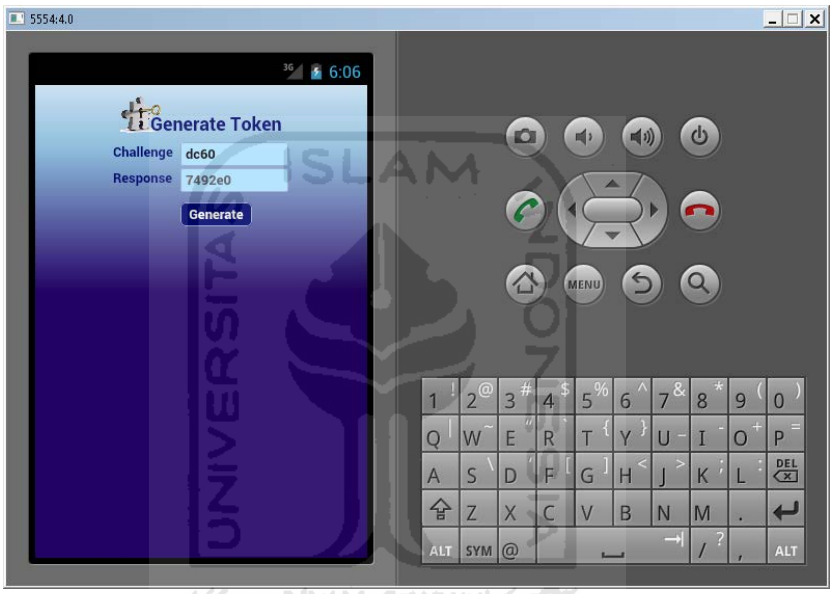
4.2.1.3 Aplikasi Mobile Token

Identifikasi masalah untuk melakukan pengujian token pada sisi *client* sebagai berikut:

1. Apakah aplikasi *mobile* token bisa membangkitkan nilai *secret key* untuk *user* ?

2. Apakah aplikasi *mobile* token bisa membangkitkan nilai *response* dan bisa digunakan untuk autentikasi?

Setelah melakukan beberapa kali pengujian hasil pengujian pada aplikasi *mobile* token adalah *mobile* token bisa membangkitkan nilai *secret key* seperti yang digambarkan pada Gambar 4.1. Aplikasi *mobile* token juga bisa membangkitkan nilai *response* dan nilai *response* tersebut bisa digunakan untuk autentikasi seperti yang digambarkan pada Gambar 4.5.



Gambar 4.5 Membangkitkan Nilai Response Pada Mobile Token

4.2.2 Penetration Testing

Pengujian tahap ini dengan melakukan eksploitasi atau serangan terhadap sistem. Pada pengujian tahap ini dilakukan menggunakan tiga teknik *hijacking* yaitu *sniffing*, *man-in-the-middle* dan *session fixation*. Pengujian dilakukan terhadap empat buah sistem sebagai berikut:

1 Sistem pertama: sistem yang hanya menggunakan autentikasi *user name* dan *password* tanpa mekanisme keamanan lain.

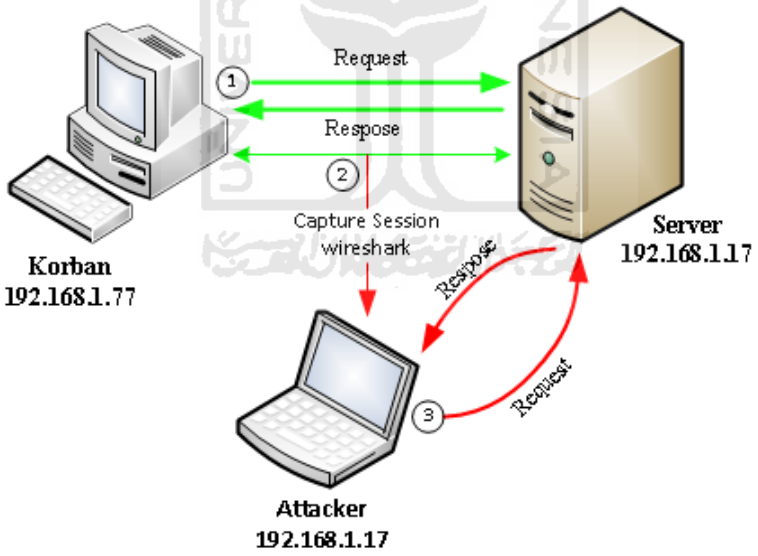
2 Sistem kedua: sistem yang menggunakan autentikasi *user name* dan *password* serta menggunakan URL *session* untuk mempertahankan *state user*.

3 Sistem ketiga: sistem yang mengunakan autentikasi *user name* dan *password* serta menggunakan *cookie session* untuk mempertahankan *state user*.

4 Sistem keempat: sistem yang menggunakan autentikasi *user name* dan *password* serta manajemen *session* dan autentikasi *challenge/response.*

4.2.2.1 Penetrasi Dengan Sniffing

Sniffing dilakukan menggunakan wireshark. Dengan sekenario korban mebuka web dari koputer *guest* (192.168.1.77) pada VirtualBox dan *sniffing* dilakukan dari komputer host (192.168.1.17) yang juga merupakan web *server*.



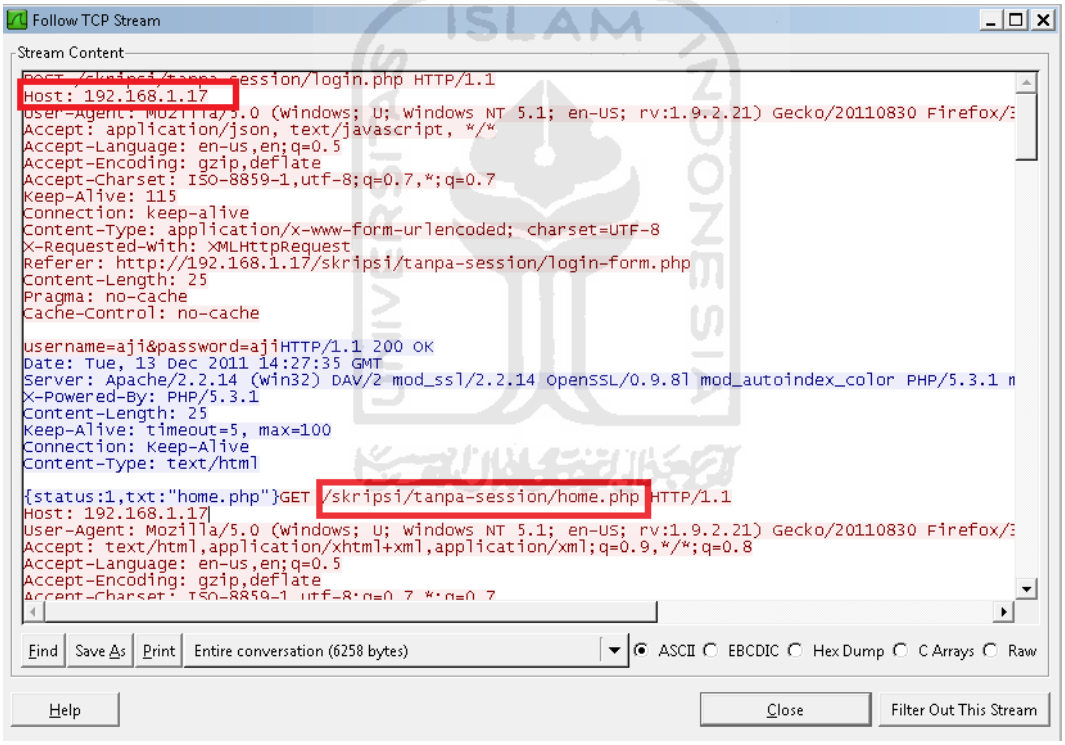
Gambar 4.6 Skenario Sniffing

Sniffing dengan wireshark tidak bisa dilakukan menggunakan satu komputer dengan web *browser* yang berbeda karena wireshark hanya bisa menangkap paket yang telah melalui *ethernet card.* Sehingga pengujian dilakukan dengan dua komputer satu komputer korban dan satu komputer sebagai web *server* sekaligus komputer untuk *siffing* seperti yang ditunjukkan pada Gambar

4.6. Dengan sekenario seperti pada Gambar 4.6 komunikasi yang dilakukan korban dengan web *server* akan meleweti *Ethernet Card* dari masing-masing komputer, sehingga paket data dalam komunikasi tersebut dapat ditangkap dengan wireshark. Hasil *sniffing* dari keempat sistem sebagai berikut:

1. Sistem Pertama (sistem tanpa session)

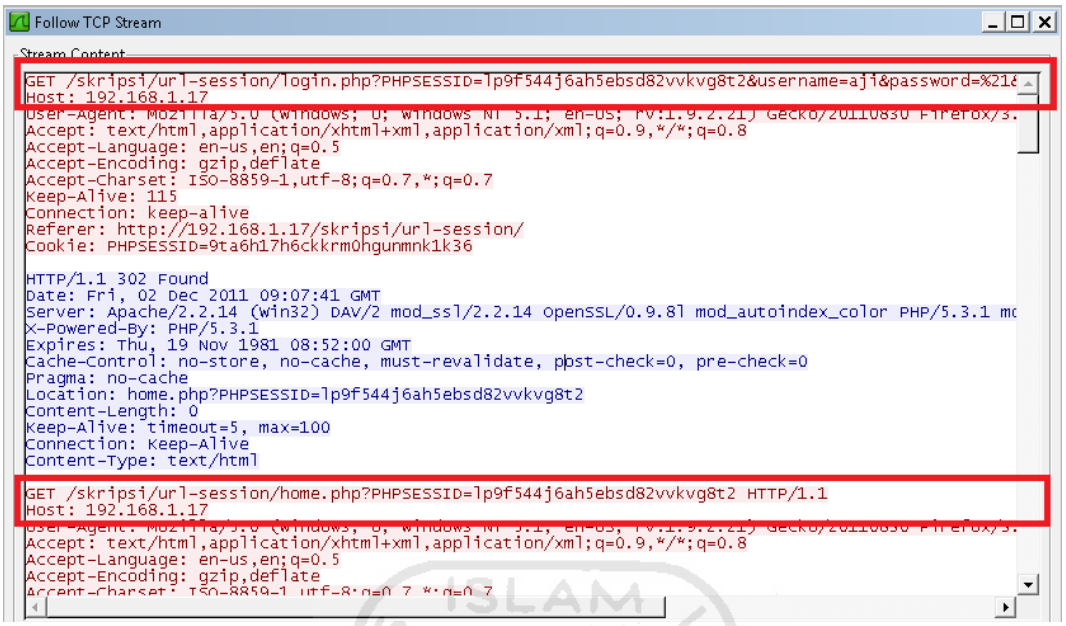
   Hijacking berhasil dengan hasil seperti pada Gambar 4.7 pada gambar tersebut didapatkan alamat *host* 192.168.1.17 dan URL /skripsi/tanpa-session/home.php. Karena sistem tidak mengunakan session, sistem lansung bisa diakses dengan alamat *host* dan URL tersebut.



Gambar 4.7 Hasil Sniffing Sistem Tanpa Session

2. Sistem Kedua (sistem dengan URL session)

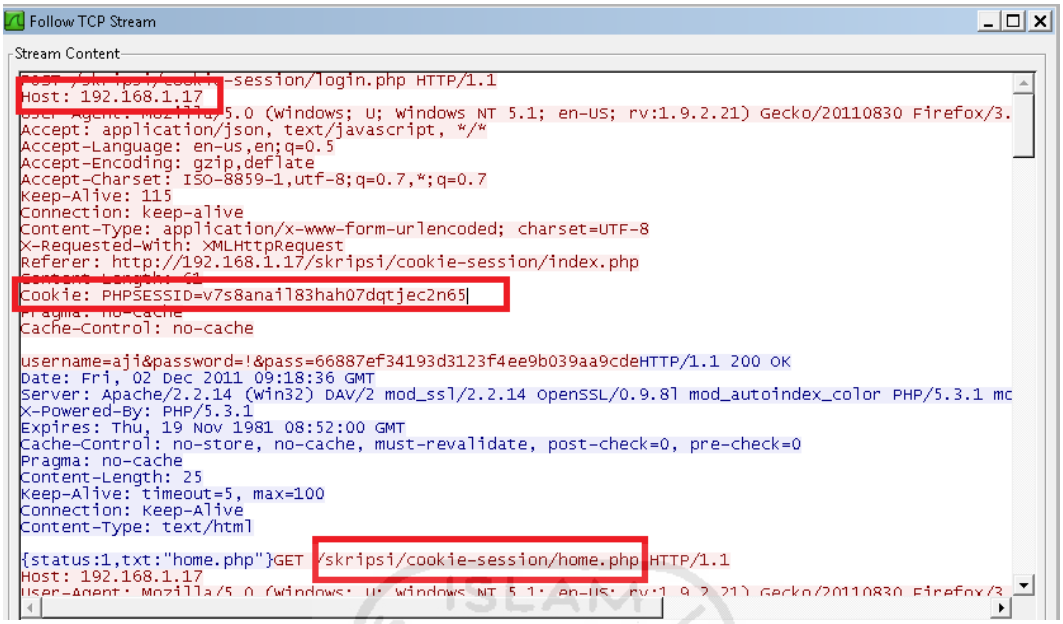
   Hijacking pada sistem yang menggunakan URL *session* juga berhasil karena didapatkan alamat *host* dan alamat URL yang mengandung *session id.* untuk lebih jelasnya bisa dilihat pada Gambar 4.8.



Gambar 4.8 Hasil Sniffing Sistem Dengan URL Session

3. Sistem Ketiga (sistem dengan cookie session)

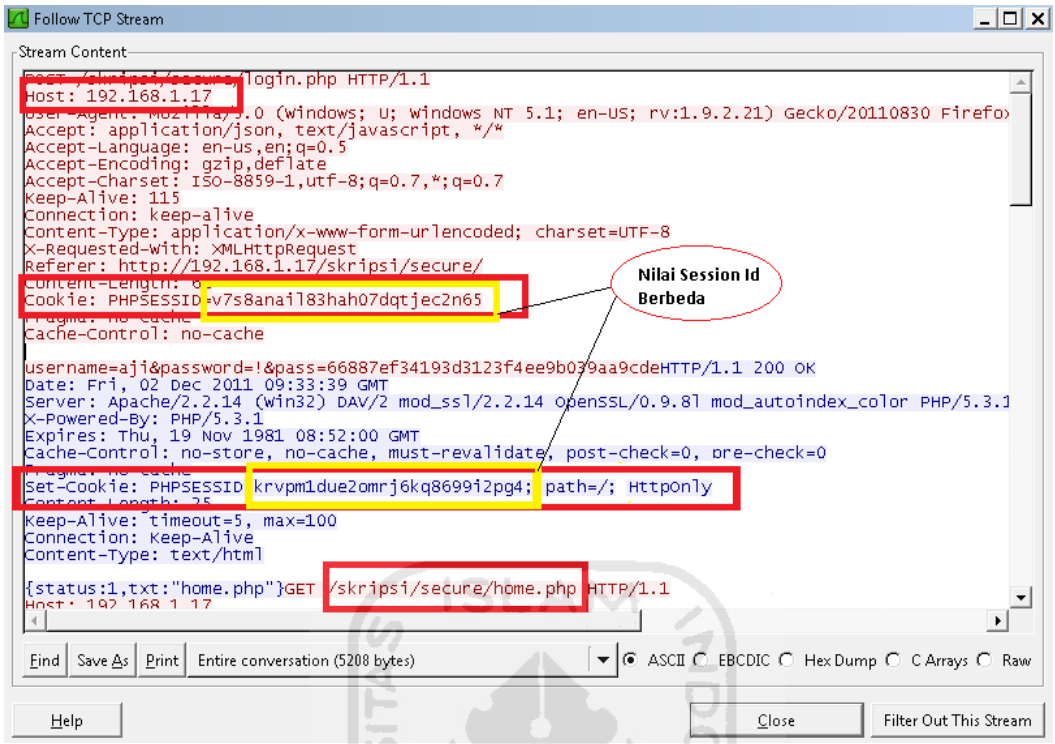
Hijacking pada sistem ketiga berhasil karena didapatkan alamat *host* dan alamat URL serta nilai *session id* yang terdapat di variabel “cookie” pada HTTP *header* dari *request.* Hasil *sniffing* bisa dilihat pada Gambar 4.8. Alamat URL tersebut bisa dibuka dengan menambahkan *cookie* pada web *browser*. Pada pengujian ini untuk menambahkan *cookie* pada web *browser* menggunakan *add-on* web developer Mozilla firefox.



Gambar 4.9 Hasil Sniffing Sistem Dengan Cookie Session

4. Sistem Keempat (sistem menggunakan manajemen session)

Hijacking pada sistem yang menggunakan manajemen *session* tidak berhasil. Hasil *sniffing* bisa dilihat pada Gambar 4.10. Walaupun dari hasil *sniffing* didapatkan alamat *host,* URL dan *session id.* informasi tersebut tersebut tidak bisa digunakan. Karena *request* dengan menggunakan URL dengan langsung menuliskannya di *address bar* pada *browser*, *request* tersebut tidak akan mengandung HTTP *referer* dan kemungkinan *user agent* yang dipakai juga berbeda, sehingga *request* tersebut tidak akan bisa melewati autentikasi *fingerprint*. Pada Gambar 4.10 juga membuktikan bahwa setiap *request* memiliki *session id* yang berbeda. Hasil *sniffing* Gambar 4.10 menunjukkan bahwa pesan *request* dan *response* memiliki *session id* yang berbeda. Pada HTTP *header* dari pesan *response*, *server* memberikan *session id* baru yang berbeda dengan *session id* pada pesan *request.*



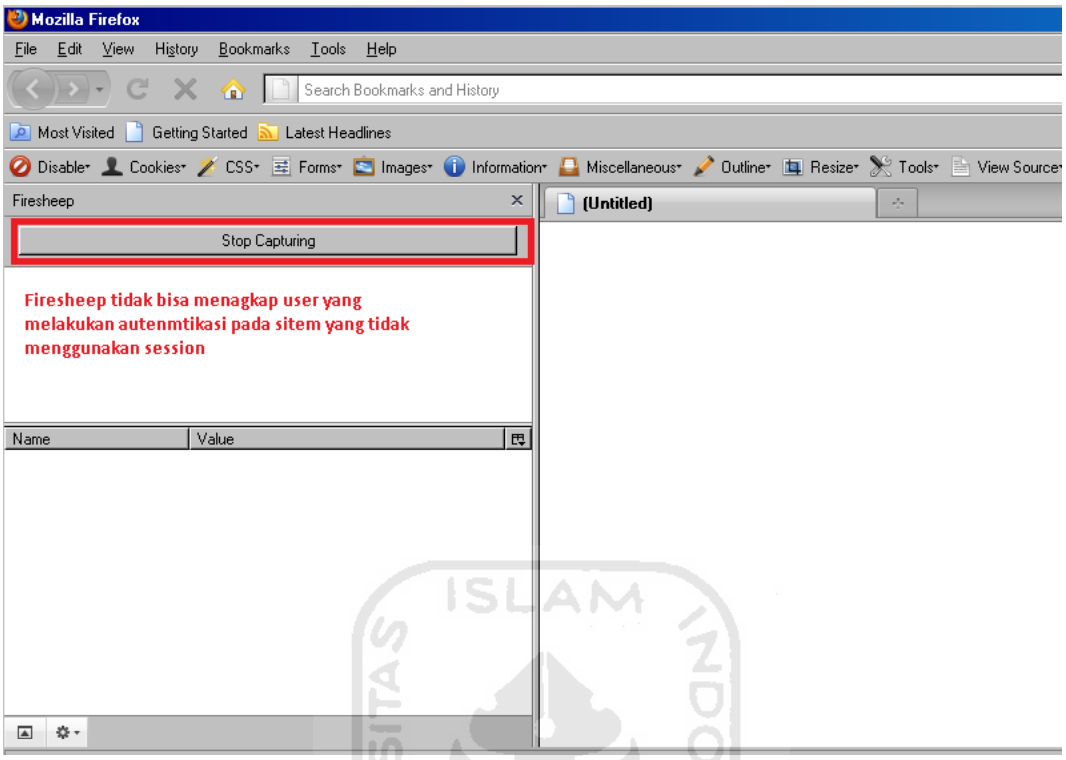
Gambar 4.10 Hasil Sniffing Sistem Dengan Manajemen Session

4.2.2.2 Penetrasi Dengan Man-in-the-Middle

Penetrasi selanjutnya adalah dengan teknik *man-in-the-middle* menggunakan *add-on* Firesheep Mozilla firefox. Hasil pengujiannya sebagai berikut:

1. Sistem Pertama (sistem tanpa session)

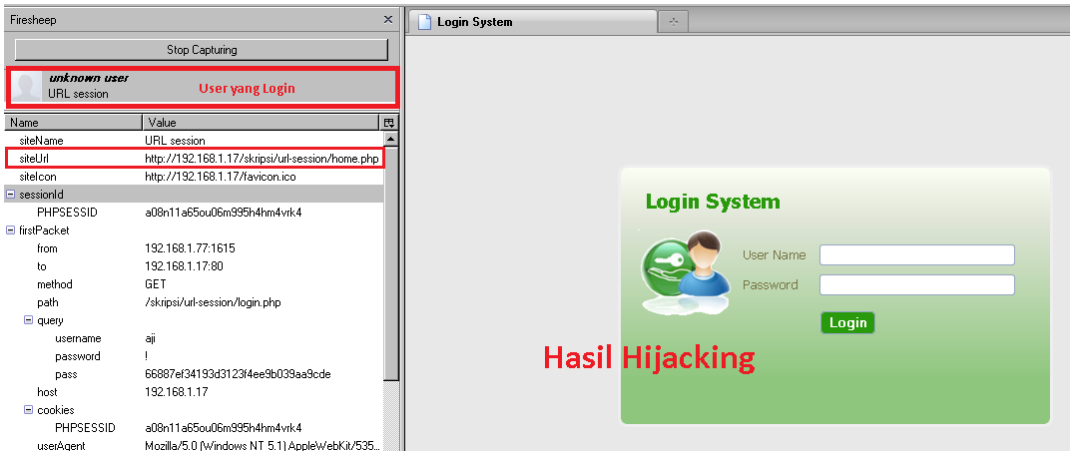
   Hijacking gagal karena sistem tidak menggunakan *session*, sehingga firesheep tidak bisa menampilkan *user* yang melakukan autentikasi *user name* dan *password* pada *slidebar* firesheep. Untuk pengujian bisa dilihat pada Gambar 4.11.



Gambar 4.11 Hasil Man-in-the-Middle Sistem Tanpa Session

2. Sistem Kedua (sistem dengan URL session)

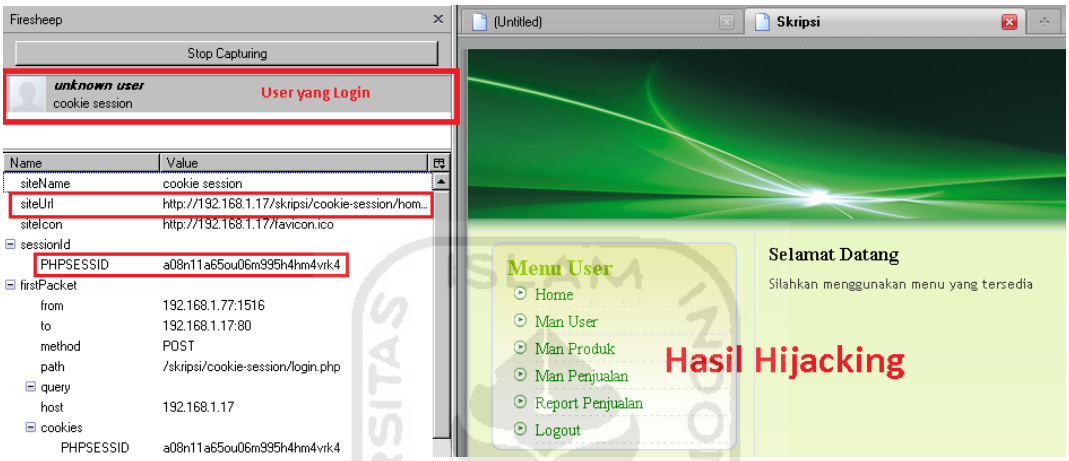
   Hijacking pada sistem kedua gagal karena firesheep tidak bisa menambahkan *session id* pada URL. Untuk hasilnya bisa dilihat pada Gambar 4.12.



Gambar 4.12 Hasil Man-in-the-Middle Sistem Dengan URL Session

3. Sistem Ketiga (sistem dengan cookie session)

   Hijacking pada sistem ketiga berhasil. Firesheep bisa menangkap *session id* yang disimpan dalam *cookie* dan menambahkan *session id tersebut* pada *cookie* Mozilla firefox. Untuk hasilnya bisa dilihat pada Gambar 4.13.



Gambar 4.13 Hasil Man-in-the-Middle Sistem Dengan cookie Session

4. Sistem Keempat (sistem menggunakan manajemen session)

   Hijacking man-in-the-middle dengan firesheep pada sistem keempat gagal. Firesheep tidak bisa melalui autentikasi *fingerprint* pada setiap *request.* Hasilnya bisa dilihat pada Gambar 4.14.

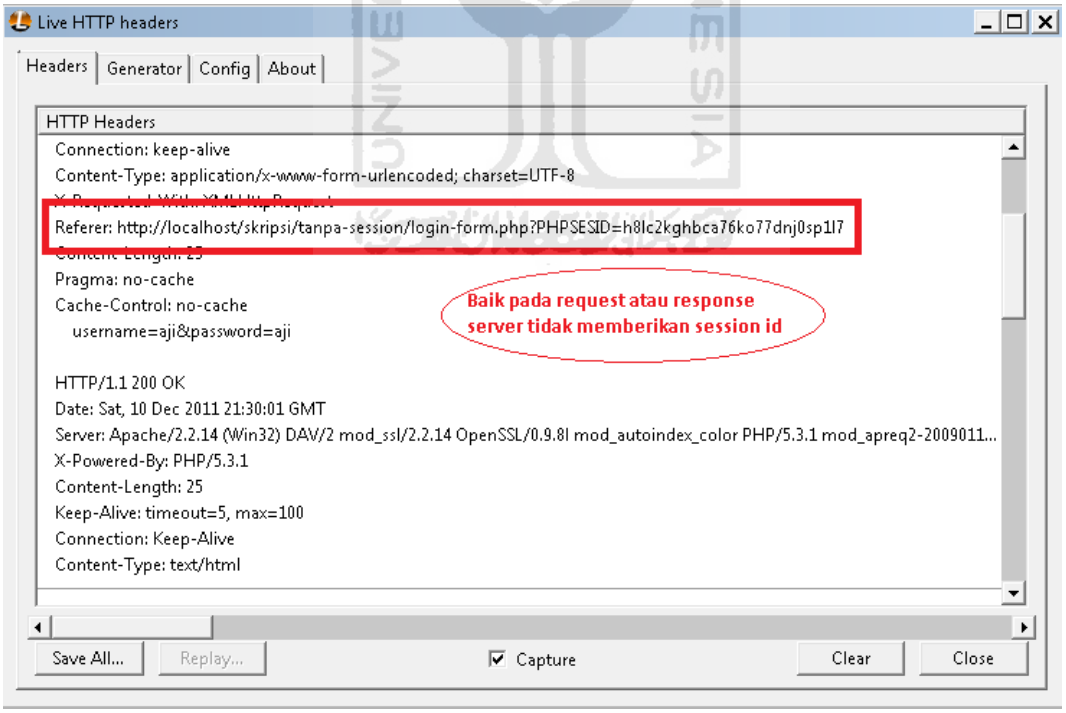
Gambar 4.14 Hasil Man-in-the-Middle Sistem Dengan manajemen Session

4.2.2.3 Penetrasi Dengan Session Fixation

Penetrasi dengan teknik *session fixation* dilakukan dengan memaksa korban menggunakan *session id* yang dipilih *attacker*. Serangan ini memanfaatkan teknik *social engineering* dengan mengirimkan URL form login yang mengandung *session id.* Sehingga korban akan login dengan *session id* yang telah diketahui *attacker.*

1. Sistem Pertama (sistem tanpa session)

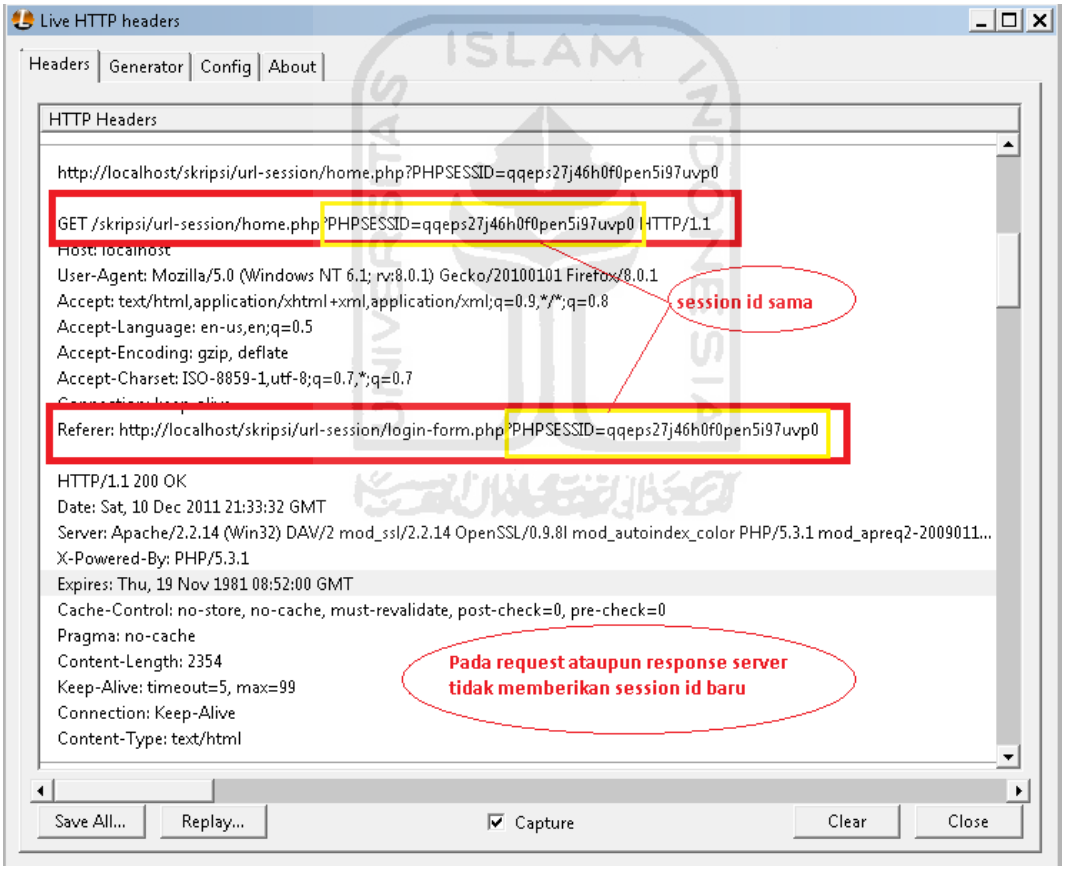
   Hijacking dengan teknik *session fixation* pada sistem pertama berhasil. Karena *attacker* bisa mengakses sistem asalkan mengetahui alamat URL nya. Pesan *request* dan *response* HTTP pada saat melakukan serangan *session fixation* bisa dilihat pada Gambar 4.15. Pada gambar tersebut, pada pesan *request* atau *response* HTTP tidak mengandung *session id*, sehingga alamat URL bisa diakses.



Gambar 4.15 Request/Response Sistem Tanpa Session

2. Sistem Kedua (sistem dengan URL session)

   Hijacking pada sistem kedua berhasil. Karena korban bisa dipaksa login dengan form login yang mengandung *session id* yang dipilih *attacker*. setelah korban login, *server* tidak membangkitkan *session id* baru, sehingga *session id* yang terdapat pada URL korban sama dengan *session id* yang dipilih *attacker*. Sehingga *attacker* bisa mengakses sistem dengan *session id* tersebut. Untuk lebih jelasnya bisa bisa dilihat pada Gambar 4.11 yang menggambarkan pesan *request* dan *response* HTTP pada saat melakukan serangan *session fixation.*

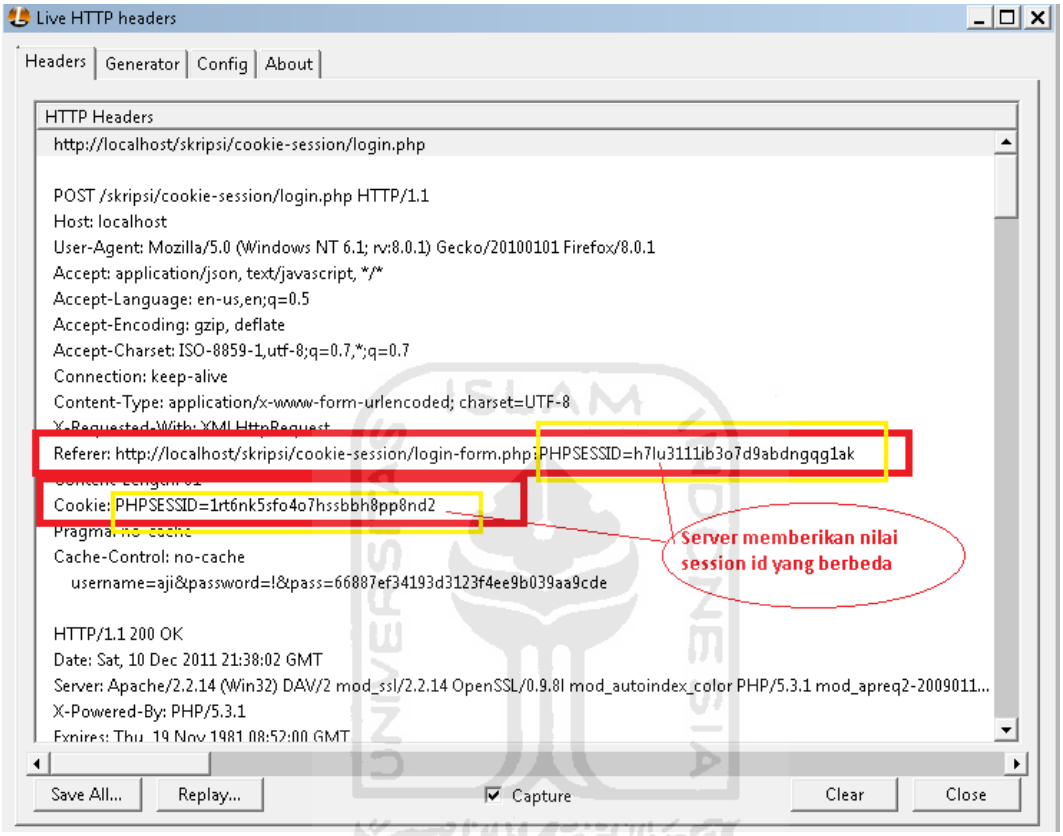


Gambar 4.16 Request/Response Sistem dengan URL Session

3. Sistem Ketiga (sistem dengan cookie session)

   Hijacking pada sistem ketiga gagal. *Server* menolak *session id* yang terdapat pada URL karena *server* membangkitkan *session id* baru dengan

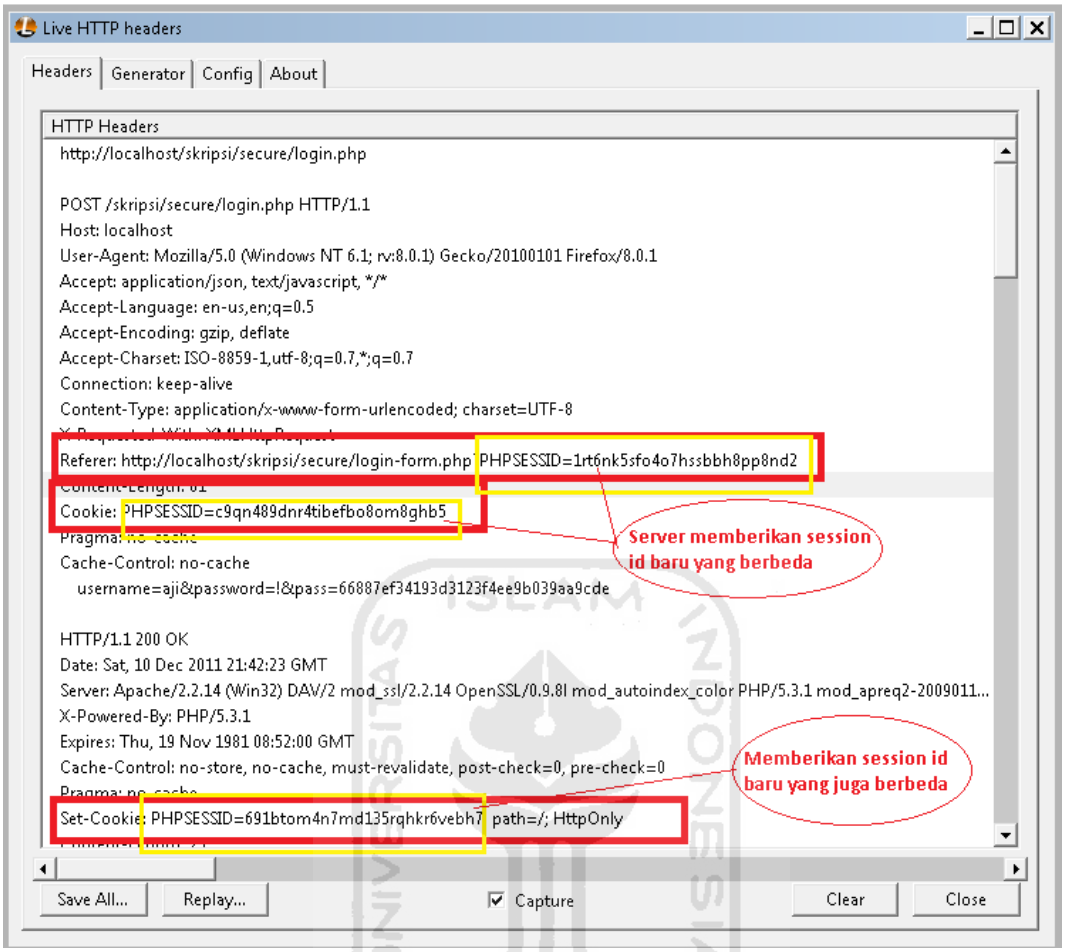
nilai yang berbeda, sehingga *session id* yang dipilih *attacker* tidak bisa digunakan. Pesan *request* dan *response* HTTP pada saat melakukan serangan *session fixation* bisa dilihat pada Gambar 4.17.



Gambar 4.17 Request/Response Sistem Dengan Cookie Session

4. Sistem Keempat (sistem menggunakan manajemen session)

   Hijacking pada sistem keempat gagal. *Server* menolak *session id* yang terdapat pada URL karena *server* membangkitkan *session id* dengan nilai yang berbeda, sehingga *session id* yang dipilih *attacker* tidak bisa digunakan. *Attacker* juga tidak bisa melewati autentikasi *fingerprint* pada tiap *request*. Pesan *request* dan *response* HTTP pada saat melakukan serangan *session fixation* bisa dilihat pada Gambar 4.18.



Gambar 4.18 Request Response Sistem Dengan Manajemen Session

Hasil pengujian tahap kedua atau *penetration testing* secara keseluruhan bisa dilihat pada Tabel 4.3.

Tabel 4.3 Hasil Pengujian Mekanisme Keamanan Tahap Dua

	TEKNIK HIJACKING					
	SNIFFING (Wireshark)		MAN-in-the-MIDDLE (Firesheep)		SESSION FIXATION	
SISTEM	Hasil	Akses	Hasil	Akses	Hasil	Akses
Pertama	Sukses	Full	gagal	-	Sukses	Full
Kedua	Sukses	Full	Gagal	-	Sukses	Full
Ketiga	Sukses	Full	Sukses	Full	Gagal	-
keempat	Gagal	-	Gagal	-	Gagal	-

Keterangan :

1. Sistem pertama : sistem tanpa menggunakan session.
2. Sistem kedua : sistem menggunakan URL *session.*
3. Sistem ketiga : sistem menggunakan *cookie session.*
4. Sistem keempat : sistem menggunakan manajemen *session* dan autentikasi token.

Dari hasil pengujian pada Tabel 4.3 bisa diambil kesimpulan bahwa mekanisme manajemen *session* yang dibuat dapat melindungi aplikasi berbasis web dari serangan *session hijacking* karena dari pengujian dengan tiga teknik *hijacking* semuanya gagal. Akan tetapi masih banyak teknik *hijacking* yang belum dicoba. Mungkin salah satunya bisa menembus mekanisme manajemen *session* yang telah dibuat karena dalam dunia pengamanan tidak ada keamanan yang sempurna. Sehingga diperlukan mekanisme keamanan tambahan berupa mekannisme autentikasi *challenge/response*. Dengan tujuan jika mekanisme manajemen *session* berhasil ditembus oleh *attacker* mekanisme autentikasi *challenge/response* dapat membatasi hak akses *attacker.*

4.3 Kelemahan Mekanisme Keamanan

a. Mekanisme manajemen session.

   Kelemahan mekanisme manajemen *session* terdapat pada algoritma *fingerprint* yang digunakan untuk autentikasi *request* dari *user*. Algoritma *fingerprint* tersebut masih bersifat rahasia. Sehingga ketika *attacker* berhasil mengetahui algoritma yang digunakan untuk membangkitkan nilai *fingerprint* tersebut, kemungkinan *attacker* akan berhasil melakukan *hijacking.*

b. Mekanisme autentikasi challenge/response.

   Kelemahan mekanisme autentikasi *challenge/response* terdapat pada token yang digunakan untuk membangkitkan nilai *response* (*mobile* token). Token (*mobile* token) dibuat berbasis *software* token sehingga akan lebih

mudah dieksploitasi untuk mendapatkan algoritma yang digunakan dalam membangkitkan nilai *response* dibandingkan dengan *hardware* token seperti yang biasa digunakan pada Internet *banking*.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari penelitian dan pembahasan yang telah dilakukan dalam pembuatan mekanisme keamanan untuk melindungi aplikasi berbasis web dari serangan *session hijacking* dapat diambil kesimpulan sebagai berikut:

1. Mekanisme manajemen *session* dan autentikasi autentikasi *challenge/response* dapat bekerja dan diimplementasikan pada aplikasi web berbasis PHP & MySQL.

2. Mekanisme manajemen *session* dapat mencegah *attacker* dalam melakukan serangan *session hijacking* dengan teknik *sniffing, man-in-the-middle* dengan firesheep dan *session fixation*.

3. Mekanisme autentikasi *challenge/response* dapat digunakan sebagai mekanisme keamanan tambahan yang membatasi hak akses *attacker* yang berhasil menerobos mekanisme manajemen *session*, sehingga dapat melindungi data pada aplikasi berbasis web dari akses ilegal *attacker*.

5.2 Saran

Berdasarkan pengujian dan pembahasan yang telah dilakukan maka penulis dapat memberikan saran untuk penelitian dan pengembangan mekanisme keamanan selanjutnya sebagai berikut:

1. Memperkuat nilai *fingerprint* yang digunakan untuk autentikasi *request* user. Misalnya dengan menambahkan MAC *addres*s, waktu (*timestamp*), *string* acak atau informasi unik lain yang bisa digunakan untuk memperkuat nilai *fingerprint.*

2. Menambah fitur IDS (Intrusion Detection System) pada mekanisme manajemen *session.* Jika terjadi serangan *hijacking* sistem tersebut dapat mengirimkan pesan kepada korban ketika ada yang mencoba atau telah membajak *account*-nya.

DAFTAR PUSTAKA

Alshanetsky, Ilia. 2005. php|architect's Guide to PHP Security. (On-line) Available at http://www.filesonic.com/file/115478441/php.architects.Guide.to.PHP.Security.Sep.2005.rar.

EC-Council. V7. Session Hijacking. (On-line) Available at http://dc387.4shared.com/download/x_84Uezb/CEHv7_Module_11_Session_Hijack.pdf?tsid=20110917-024900-15ef80f5.

Gourley, David & Brian Tolly. 2002. HTTP *The Definitive Guide*. (On-line) Available at http://www.mediafire.com/?kg247ihayh61op6.

Internet Engineering Task Force (IETF). 1999. RFC:2616 HTTP/1.1. (On-line) Available at http://www.ietf.org/rfc/rfc2616.txt.

Munir, Rinaldi. 2006. Algoritma dan Pemrograman. Banduung: Informatika.

Munir, Rinaldi. 2006. Kriptografi. Bandung: Informatika.

Oni, Marvello. 2010. Analisis Penggunaan Kriptografi dalam Online Banking. (On-line) Available at http://www.informatika.org/~rinaldi/Kriptografi/2010-2011/Makalah1/Makalah1-IF3058-Sem1-2010-2011-035.pdf.

OWASP (Open Web Aplication Security Project). 2009. Session Hijacking Attack. (On-line) Available at https://www.owasp.org/index.php/Session_hijacking_attack.

Rahardjo, Budi. 2005. Keamanan Sistem Informasi (On-line) Available at http://budi.insan.co.id/books/handbook.pdf.

Shiflett, Chris. 2005. Essential PHP Security. (On-line) Available at http://dump.udderweb.com/Books/unsorted/Essential_PHP_Security_2005.pdf.

Shiflett, Chris. 2003. HTTP Developer's Handbook. (On-line) Available at http://www.mediafire.com/?kg247ihayh61op6.

Shklar, Leon & Richard Rosen. 2003. Web Application Architecture Principles, protocols and practices. (On-line) Available at http://depositfiles.com/en/files/6sjdmdgx3.

S'to. 2009. Web Hacking Skenario & Demo. Jakarta: Jasakom.

Thomas, Stephen. 2001. HTTP Essentials Protocols for Secure, Scaleable Web Sites. (On-line) Available at http://www.mediafire.com/?xzbnwijjiu1.

LAMPIRAN

.