

TUGAS KEAMANAN SISTEM INFORMASI (EC-5010)

MAKALAH

**“MEKANISME DAN IMPLEMENTASI
KEAMANAN PADA *SESSION INITIATION
PROTOCOL (SIP)*”**

Nama : Martinus Indra S.
NIM : 13200039
Dosen : Budi Rahardjo



DEPARTEMEN TEKNIK ELEKTRO
INSTITUT TEKNOLOGI BANDUNG

2004

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Sekarang ini komunikasi multimedia sudah menjadi hal yang tidak dapat dipisahkan lagi dalam aplikasi internet. Aplikasi ini meliputi *Voice over Internet Protocol* (VoIP), konferensi multimedia, *Instant Messaging*, dan sebagainya. Oleh karena itu sangat dibutuhkan adanya suatu manajemen dalam pertukaran data yang melibatkan sekumpulan pengguna ini. Fungsi manajemen ini dapat dilakukan oleh *Session Initiation Protocol* (SIP).

Masalah keamanan merupakan salah satu aspek yang sangat penting pada sebuah sistem informasi. Demikian juga dengan masalah keamanan pada SIP. Meskipun demikian SIP bukanlah protokol yang mudah dijamin keamanannya. Operasinya yang melibatkan banyak pengguna, elemen *intermediate*, dan protokol lainnya menyebabkan faktor keamanan jauh dari sederhana.

1.2 Tujuan

Tujuan penulisan makalah ini adalah membahas mekanisme dan implementasi keamanan pada SIP. Dengan adanya mekanisme keamanan ini, diharapkan masalah keamanan pada aplikasi yang menggunakan basis SIP dapat diatasi.

1.3 Metode Penulisan

Makalah ini ditulis berdasarkan beberapa referensi yang diperoleh penulis dari internet. Implementasi dilakukan dengan meng-*compile source-code* dan konfigurasi.

1.4 Sistematika Penulisan

Makalah ini terdiri dari 5 bab, yaitu sebagai berikut :

BAB 1 : PENDAHULUAN

Bab ini terdiri dari latar belakang penulisan makalah, tujuan penulisan makalah, metode penulisan yang digunakan, dan sistematika penulisan makalah.

BAB 2 : SESSION INITIATION PROTOCOL (SIP)

Karena SIP tidak dipelajari pada kuliah, maka akan dibahas terlebih dahulu mengenai konsep SIP secara singkat, meliputi pengertian SIP, komunikasi dengan SIP, beberapa aplikasi yang menggunakan SIP, serta kelebihan yang ditawarkan oleh SIP.

BAB 3 : MEKANISME KEAMANAN PADA SIP

Bab ini membahas tentang model serangan pada SIP, mekanisme keamanan pada SIP, dan beberapa keterbatasan dari mekanisme keamanan tersebut.

BAB 4 : IMPLEMENTASI KEAMANAN PADA SIP

Bab ini membahas tentang implementasi dari mekanisme keamanan yang dibahas pada bab 3, meliputi autentikasi dan enkripsi.

BAB 5 : KESIMPULAN

Bab ini berisi kesimpulan dari bab-bab sebelumnya.

BAB 2

SESSION INITIATION PROTOCOL (SIP)

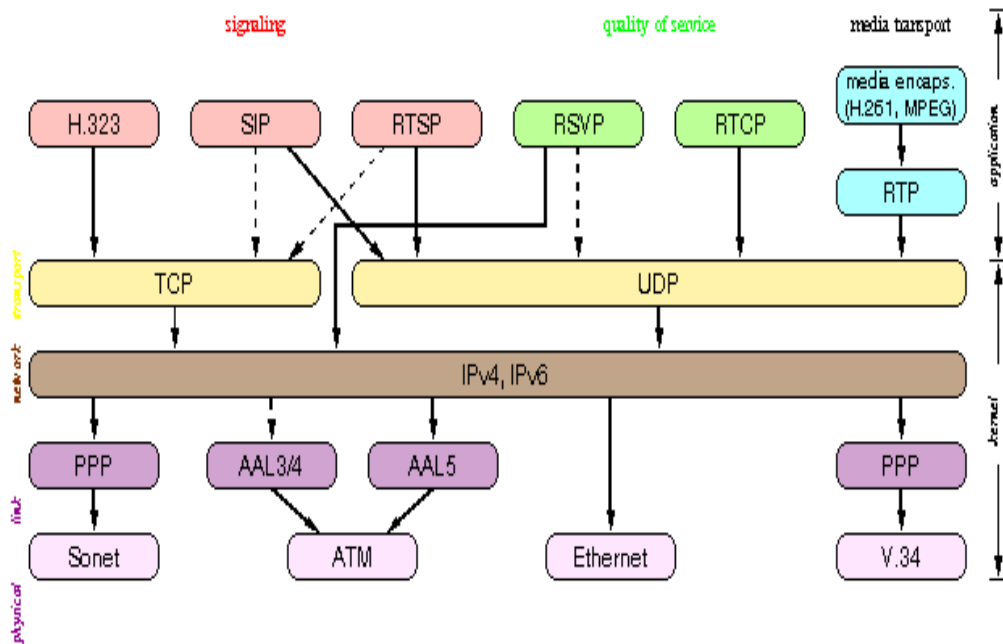
2.1 Pengertian SIP

SIP adalah suatu *signalling* protokol pada layer aplikasi yang berfungsi untuk membangun, memodifikasi, dan mengakhiri suatu sesi multimedia yang melibatkan satu atau beberapa pengguna. Sesi multimedia adalah pertukaran data antar pengguna yang meliputi suara, video, atau text.

SIP tidak menyediakan layanan secara langsung, tetapi menyediakan fondasi yang dapat digunakan oleh protokol aplikasi lainnya untuk memberikan layanan yang lebih lengkap bagi pengguna, misalnya dengan RTP (*Real Time Transport Protocol*) untuk transfer data secara *real-time*, dengan SDP (*Session Description Protocol*) untuk mendeskripsikan sesi multimedia, dengan MEGACO (*Media Gateway Control Protocol*) untuk komunikasi dengan PSTN (Public Switch Telephone Network). Meskipun demikian, fungsi dan operasi dasar SIP tidak tergantung pada protokol tersebut. SIP juga tidak tergantung pada protokol layer transport yang digunakan. (Posisi SIP pada layer TCP/IP dapat dilihat pada gb. 1.1)

Pembangunan suatu komunikasi multimedia dengan SIP dilakukan melalui beberapa tahap :

- User location : menentukan lokasi pengguna yang akan berkomunikasi.
- User availability : menentukan tingkat keinginan pihak yang dipanggil untuk terlibat dalam komunikasi.
- User capability : menentukan media maupun parameter yang berhubungan dengan media yang akan digunakan untuk komunikasi.
- Session setup : “ringing”, pembentukan hubungan antara pihak pemanggil dan pihak yang dipanggil.
- Session management : meliputi transfer, modifikasi, dan pemutusan sesi.



gambar 1

Multimedia Protocol Stack

(sumber : <http://www.cs.columbia.edu/~hgs/internet/>)

2.2 Komunikasi dengan SIP

Komunikasi pada SIP dilakukan dengan mengirimkan *message* yang berbasis HTTP. Setiap pengguna mempunyai alamat yang dinyatakan dengan SIP-URI (*Uniform Resource Identification*).

Contoh SIP URI : *sip: martin@bandung.com*

Selain itu, alamat juga dapat dituliskan dalam tel-URL yang kemudian dikonversikan menjadi SIP-URI dengan parameter 'user' diisi 'phone'.

Contoh : tel: +62-22-2534119

ekivalen dengan

sip: +62-22-2534119@bandung.com ; user=phone

Hubungan yang dibangun oleh SIP pada proses *signalling* bersifat *client-serve*. Dengan demikian ada 2 jenis *message*, yaitu *request* dan *response*.

<i>SIP Request Messages</i>	<i>Descriptions</i>
INVITE	Indicates that the user or service is being invited to participate in a session.
ACK	Confirms that the client has received a final response to an INVITE request.
BYE	Indicate the user wishes to terminate the call.
CANCEL	Cancels a pending request but does not affect a completed request.
REGISTER	Register the address listed in the To header field with a SIP server.
OPTIONS	Queries the capability of the servers.
INFO	Allows for the carrying of the session related control information that is generated during a session.

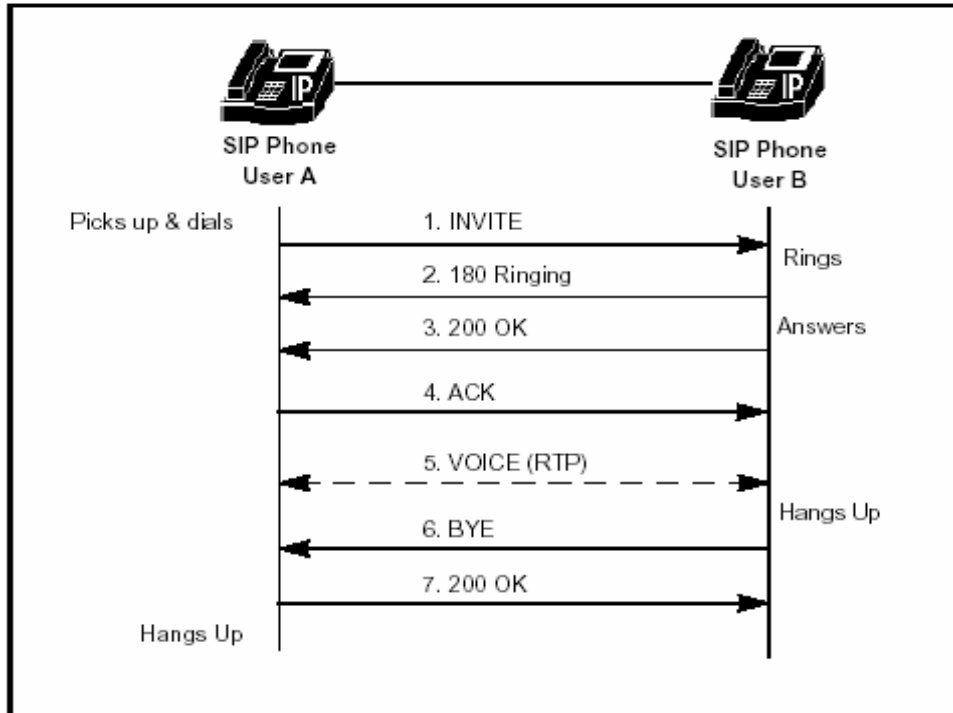
tabel 1
SIP Request Message

<i>SIP Response Message Types</i>	<i>Description</i>
1xx	Information Responses For example: 180 Ringing
2xx	Successful Responses For example: 200 OK
3xx	Redirection Responses For example: 302 Moved Temporarily
4xx	Request Failures Responses For example: 403 Forbidden
5xx	Server Failure Responses For example: 504 Gateway Time-out
6xx	Global Failure Responses For example: 600 Busy Everywhere

tabel 2
SIP Respond Message

2.2.1 Operasi dasar pada SIP

Contoh : User A menggunakan aplikasi SIP pada PC (*softphone*) untuk memanggil User B (juga menggunakan *softphone*) melalui internet.



gambar 2
Basic Call Flow Diagram

Step	Description
1	INVITE: User A initiates a call to User B.
2	180 Ringing: User B sends a ringing signal back to User A.
3	200 OK: User B picks up.
4	ACK: User A acknowledges that it received the 200 message.
5	VOICE: A two-way voice channel is established over Real-time Transport Protocol (RTP) and a conversation takes place between User A and B.
6	BYE: User B hangs up.
7	200 OK: The call is torn down and User A hangs up.

tabel 3
Call Flow Details

2.2.2 Operasi dengan proxy

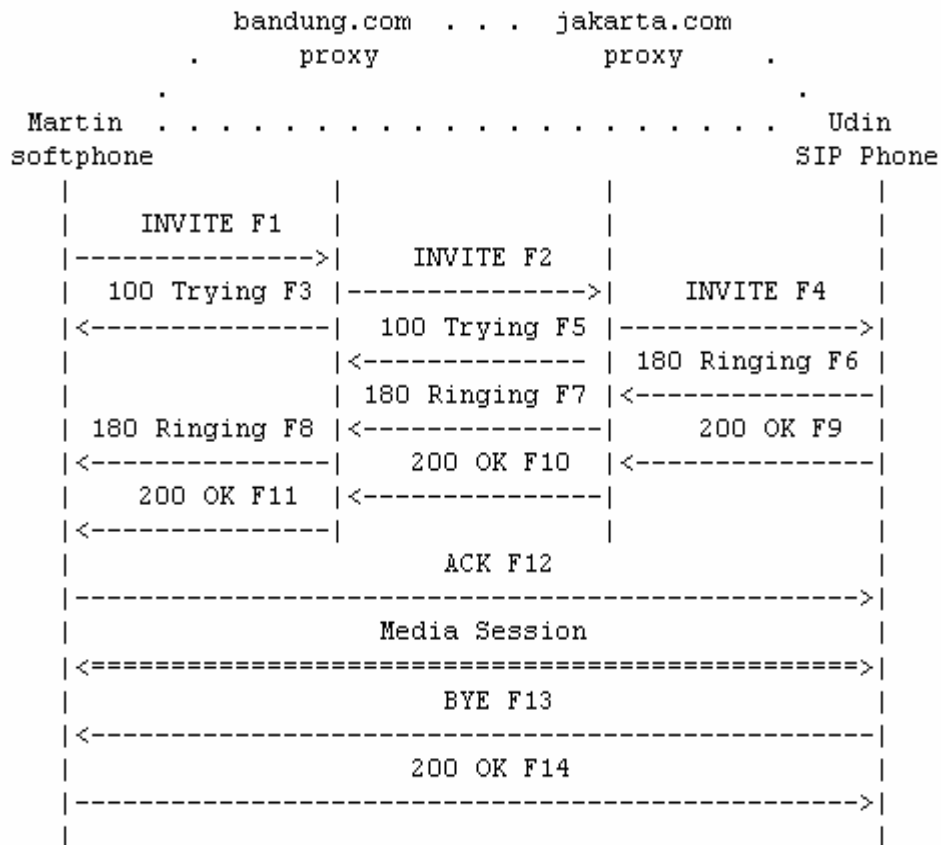
Jika pengguna mempunyai domain yang berbeda, maka pengiriman *message* dapat melibatkan proxy pada masing-masing domain. Proxy berfungsi membuat *request* atas nama *client*. Proxy juga dapat melakukan autentifikasi terhadap *message* yang diterimanya sebelum diteruskan.

Contoh : Martin menggunakan *softphone* (dengan alamat *sip: martin@bandung.com*, di mana *bandung.com* adalah domain dari SIP service provider tempat Martin berada) untuk melakukan panggilan kepada Udin (dengan alamat *sip: udin@jakartai.com*, di mana *jakarta.com* adalah domain dari SIP provider tempat Udin berada).

INVITE adalah *request* yang dikirimkan oleh pihak pemanggil (Martin) kepada pihak yang dipanggil (Udin) untuk membuka komunikasi. INVITE terdiri dari beberapa *header-fields* yang memberikan informasi mengenai *message* yang dikirimkan.

Proxy-server (*bandung.com*) menerima INVITE *request*, kemudian mengirimkan *response* (*Trying/100*) kepada *softphone* Martin yang mengindikasikan bahwa INVITE *request* telah diterima. Proxy *bandung.com* mencari alamat proxy tujuan (*jakarta.com*) pada *database*. Jika ditemukan, INVITE *request* di-*routing*-kan kepada hop/proxy selanjutnya. Sebelum meneruskan *request*, suatu proxy-server menambahkan header tambahan pada **Via** yang berisi alamat proxy tersebut. Demikian juga dengan proxy-server *jakarta.com*. Setelah ditambahkan header tambahan pada **Via**, proxy ini meneruskan INVITE *request* kepada *softphone* Udin.

Pada contoh ini, Udin menjawab panggilan dari Martin. *SIP phone* Udin mengirimkan *response* berupa OK (kode 200). Pertukaran *message* ini disertai *negotiation capabilities* untuk menentukan media maupun parameternya sehingga komunikasi dapat berjalan dengan baik.



gambar 3
Proxy-mode call flow

Pada contoh di atas, INVITE dapat dituliskan sebagai berikut :

```

INVITE sip:udin@jakarta.com SIP/2.0
Via: SIP/2.0/UDP bandung.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: Udin <sip:udin@jakarta.com>
From: Martin <sip:martin@bandung.com>;tag=1928301774
Call-ID: a84b4c76e66710@bandung.com
CSeq: 314159 INVITE
Contact: <sip:martin@bandung.com>
Content-Type: application/sdp
Content-Length: 142
  
```

(Martin's SDP not shown)

Keterangan :

- Baris pertama merupakan *start-line* (*request-line/status-line*) yang menunjukkan awal dari sebuah *message*. Pada *request*, *start-line* disebut ***request-line***, yang terdiri dari `<metode> <request-URI> <SIP-version>`
Pada contoh di atas, metode dari *request* adalah INVITE, URI ditujukan ke *sip:udin@jakarta.com*, dan SIP yang digunakan adalah versi 2.0
- **Via** berisi alamat proxy di mana *client* akan menerima *response*. Pada contoh di atas alamat proxy-nya adalah *bandung.com*. Via juga dapat berisi parameter *branch* yang menunjukkan identitas suatu *transaction* pada proxy.
- **Max-Forwards** menunjukkan jumlah *hop* maksimum yang dapat dilalui oleh *message* sebelum mencapai tujuan. Angka pada max-forwards di *decrement* setiap bertemu *hop*.
- **To** berisi alamat dari tujuan. Alamat ini dapat dituliskan dalam bentuk *display name* (*Udin*), atau dalam bentuk SIP URI (*sip:udin@jakarta.com*).
- **From** berisi alamat pengirim yang dapat dituliskan dalam bentuk *display name* (*Martin*), atau dalam bentuk SIP URI (*sip:martin@bandung.com*).
Header ini juga mempunyai parameter *tag* yang menunjukkan identitas *softphone client* yang digunakan.
- **Call-ID** berisi kombinasi karakter yang menunjukkan identitas dari suatu *call* yang dimulai pada waktu tertentu (di-*generate* berdasarkan *random string* dan *ip-address/hostname* dari *softphone*).
Kombinasi dari *Call-ID*, *To-tag*, dan *From-tag* menunjukkan suatu identitas dari hubungan *peer-to-peer* yang disebut *dialog*.
- **CSeq** (*Command Sequence*) berisi angka integer dan nama metode. Angka integer pada CSeq di-*increment* setiap ada *request* dalam suatu dialog
- **Content-Type** menunjukkan deskripsi dari *message-body*.
- **Content-Length** menunjukkan panjang dari *message-body* (*byte*).
- **SDP-Message** dibawa oleh SIP message sebagai *attachment*.

Response dari *softphone* Udin berupa OK, dapat dituliskan sebagai berikut:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP jakarta.com
    ;branch=z9hG4bKnashds8;received=192.0.2.3
Via: SIP/2.0/UDP bogor.com
    ;branch=z9hG4bK77ef4c2312983.1;received=192.0.2.2
Via: SIP/2.0/UDP bandung.com
    ;branch=z9hG4bK776asdhs ;received=192.0.2.1
To: Udin <sip:udin@jakarta.com>;tag=a6c85cf
From: Martin <sip:martin@bandung.com>;tag=1928301774
Call-ID: a84b4c76e66710@bandung.com
CSeq: 314159 INVITE
Contact: <sip:udin@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

(Udin's SDP not shown)
```

Keterangan :

- Baris pertama (*start-line*) pada *response message* disebut *status-line*. *Status-line* ini berisi <*SIP-version* = SIP/2.0> <*status-code*=200> <*status-phrase*=OK>.
- **Via, To, From, Call-ID, dan Cseq** di-copy dari INVITE *request*. Pada *Via header-field* sudah ditambahkan header tambahan yang berisi alamat tiap proxy yang dilewati. *Softphone* Udin menambahkan parameter *tag* pada header **To**.
- **Contact** berisi alamat URI dari Udin yang berfungsi untuk mempermudah pertukaran *message* di waktu yang akan datang.

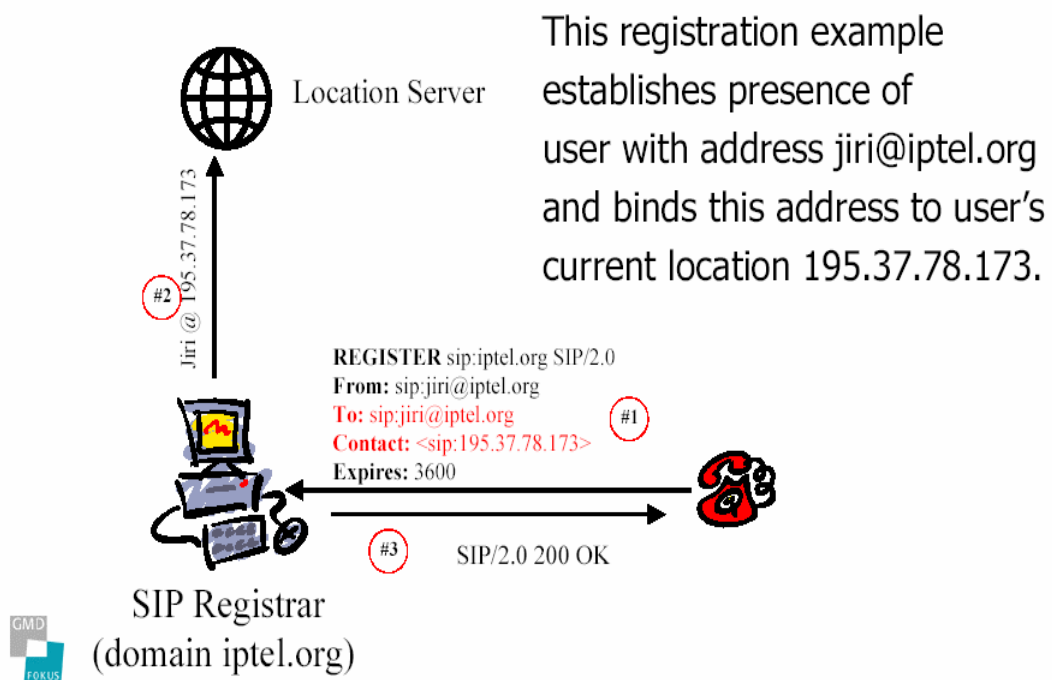
2.2.3 Registrasi pada SIP

Salah satu keistimewaan SIP adalah adanya mobilitas yang tinggi bagi pengguna. Untuk mengetahui keberadaan seseorang pada saat tertentu, suatu SIP proxy-server harus mendapatkan informasi di mana seseorang berada. Proses ini disebut registrasi. Seorang pengguna mula-mula harus melakukan proses registrasi agar proxy terdekat mengetahui keberadaannya, sehingga proxy tersebut dapat menerima atau mengirimkan pesan kepada pengguna.

Proses registrasi dilakukan oleh pengguna dengan cara mengirimkan *request* yang disebut REGISTER kepada proxy-server. Proxy-server ini disebut *registrar*. Sebuah *registrar* bertugas menerima dan menyimpan data yang berisi alamat pengguna. *Registrar* dan proxy-server biasanya merupakan satu kesatuan.

Contoh proses registrasi pada SIP :

SIP Registration



gambar 4
Proses Registrasi pada SIP
(sumber : <http://www.iptel.org/siptutorial>)

Suatu REGISTER *request* mempunyai *header-field* sebagai berikut :

- **Request URI** berisi nama domain dari *registrar*. Header ini tidak boleh mengandung karakter '@' dan nama pengguna. Pada contoh di atas *sip:iptel.org*.

- **To** berisi alamat sip dari seorang pengguna yang akan disimpan pada *registrar*. Pada contoh ini *sip: jiri@iptel.org*
- **From** berisi alamat seseorang yang melakukan proses registrasi. Biasanya sama dengan alamat pada **To**, kecuali registrasi dilakukan oleh pihak ketiga.
- **Contact** biasanya berisi SIP-URI yang menunjukkan suatu SIP *endpoint*. Pada contoh ini *sip:195.37.78.173*

2.3 Aplikasi dan Kelebihan SIP

2.3.1 Aplikasi :

- *Voice over Internet Protocol (VoIP)*
- Konferensi multimedia
- *Text-messaging*
- *Event-notification -> voicemail notification, callback notification*
- *Unified Messaging -> voicemail2email*

2.3.2 Kelebihan

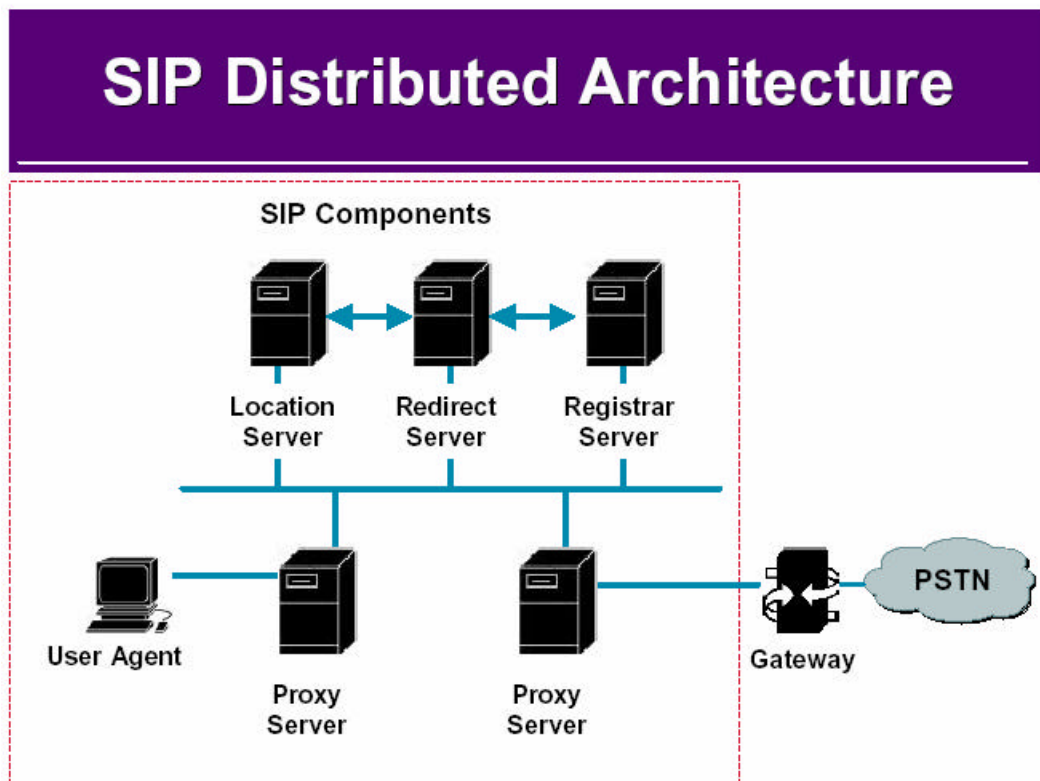
➤ *General-purpose*

SIP dapat diintegrasikan dengan protokol standar IETF lainnya untuk membuat suatu aplikasi yang berbasis SIP.

➤ *Arsitektur yang terdistribusi dan scalable*

- **Proxy-server**
Menerima *request* dari *user-agent-client*, melakukan autentikasi, memprosesnya, dan mengirimkan *request* tersebut kepada *hop* selanjutnya atas nama *client* tersebut.
- **Redirect-server**
Menerima *request* dari *client*, membandingkan alamat tujuan yang ingin dicapai, setelah ditemukan, alamat tersebut dikembalikan kepada *client*.

- Registrar-server
Menerima REGISTER *request* dari *client*.
- Location-server
Menyimpan data yang diperoleh dari registrar-server. Location-server digunakan oleh proxy/redirect server untuk mendapatkan informasi mengenai alamat tujuan yang ingin dicapai.



gambar 5
Arsitektur Sistem berbasis SIP
(sumber : <http://www.vovida.org>¹)

Dengan adanya fungsi yang terdistribusi, proses pengembangan pada salah satu komponen tidak akan mengganggu komponen lainnya. (*scalable*)

1. vovida.org adalah sebuah web yang memfasilitasi beberapa *open-source-project*, yang bertujuan mengembangkan sistem komunikasi yang bersifat *open-source* .Beberapa *source-code* project yang berbasis SIP dapat di-*download* dari web ini.

➤ Sederhana

Pengiriman *message* berbasis HTTP (*text-based*), bukan *binary-based*. Hal ini menyebabkan SIP mudah diimplementasikan.

➤ *Mobility*

- Seorang pengguna dapat menerima *message/call* yang ditujukan kepadanya, meskipun berpindah dari satu lokasi ke lokasi lainnya. Proxy-server akan meneruskan *call* ke lokasi pengguna pada saat ini.
- *Device* yang digunakan dapat berupa PC, baik di rumah maupun di kantor, *wireless phone*, *IP-phone*, ataupun telepon biasa.

➤ Layanan dapat dibuat dengan *Call Processing Language* (CPL) dan *Common Gateway Interface* (CGI), antara lain :

- *call waiting*, *call forwarding*, *call blocking* (basic feature)
- *call-forking* (melakukan *call* kepada beberapa *endpoint*)
- *Instant-messaging*
- *Find-me / follow-me*

BAB 3

MEKANISME KEAMANAN PADA SIP

3.1 Model Serangan pada SIP

3.1.1 Pembajakan pada proses registrasi

Proses registrasi pada intinya bertujuan agar seorang pengguna SIP dikenali oleh proxy-servernya. Caranya adalah dengan mengirimkan REGISTER *request*. (proses registrasi dapat dilihat pada subbab 2.2.3)

Sebuah proxy (*registrar*) melihat header **From** dari REGISTER untuk menentukan apakah *client* tersebut berhak memodifikasi isi dari header **Contact**. Registrasi ini diperbolehkan untuk dilakukan oleh pihak ketiga, atas nama pihak pertama, dengan header **From** berisi alamat pihak ketiga tersebut. Dengan demikian header **From** dapat dimodifikasi oleh pengguna lainnya.

Hal inilah yang dapat memancing seseorang untuk melakukan niat tidak baik. Seorang pengguna yang jahat dapat saja berpura-pura sebagai pihak ketiga, kemudian mengubah header **Contact** dengan alamatnya. Akibatnya, semua *message* yang ditujukan kepada pihak pertama tadi dialihkan kepadanya.

3.1.2 Menyamar sebagai sebuah server

Domain tujuan dari sebuah *request* dapat dilihat pada header **Request-URI**. Sebelum sampai ke tujuan, biasanya sebuah *message* melalui server dari domain tersebut. Jika seseorang berhasil menyamar sebagai server pada suatu domain, maka seluruh *request* yang sampai dapat disalahgunakan.

Misalnya registrasi yang ditujukan kepada *asli.com* berhasil ditangkap oleh *palsu.com*. Kemudian dibalas dengan mengirimkan *response* 301 (*Moved Permanently*). *Response* ini seolah-olah datang dari *asli.com*, padahal datangnya dari *palsu.com*. Akibatnya proses registrasi seterusnya dikirimkan ke *palsu.com*.

3. 1. 3 Kerusakan pada isi *Message*

Misalnya seorang pengguna mengirimkan isi *message* dengan kunci enkripsi yang diketahui oleh suatu proxy (*hop-by-hop-encryption*). Meskipun proxy tersebut dapat dipercaya, tetapi seorang administrator yang nakal pada domain tersebut dapat saja mendekripsi *crypted-text* tersebut, bahkan memodifikasi kunci enkripsinya. Hal ini bisa dikategorikan sebagai serangan *man-in-the-middle* yang mengubah karakteristik keamanan yang diinginkan oleh pengguna.

Jenis serangan ini tidak hanya mengancam kunci enkripsi, tetapi juga beberapa *header-field* seperti **Subject**, bahkan isi dari *message*, misalnya MIME (*Multipurpose Internet Mail Extensions*) dan SDP (*Session Description Protocol*), sehingga pembicaraan bisa disadap.

3. 1. 4 Menghentikan Sesi

Setelah suatu dialog dibangun, *request* dapat dikirimkan berdasarkan urutan tertentu untuk memodifikasi *state* dari dialog/sesi tersebut.

Misalnya seseorang berhasil menangkap beberapa *message* yang dikirimkan mula-mula oleh kedua *participant* untuk membangun komunikasi. Dengan demikian ia bisa mengetahui parameter-parameter yang sensitif terhadap suatu sesi seperti **To tag**, **From tag**, **Call-ID**, **Cseq**, dan sebagainya. Kemudian ia mengirimkan *request* BYE seolah-olah dari salah satu *participant*. Akibatnya suatu sesi berakhir secara prematur.

3. 1. 5 *Denial of Service Attack*

Jenis serangan ini bertujuan menghabiskan *resource* dari suatu elemen dalam jaringan, biasanya dengan mengirimkan paket dalam jumlah besar ke suatu target (*network flooding*)

Seseorang dapat saja mengirimkan *request* gadungan dengan alamat pengirim diisi target yang akan diserang. Kemudian *request* ini dikirimkan kepada banyak elemen SIP. Akibatnya sang target dibanjiri oleh *response* dari banyak elemen SIP.

3.2 Mekanisme Keamanan

3.2.1 HTTP Digest Authentication

SIP adalah protokol yang berbasis internet dan bersifat *web-centric*. Penulisan sintaks dan pengiriman *message* diadopsi dari HTTP. Bahkan desain protokol dan model layanan yang diberikan oleh SIP lebih menyerupai HTTP dibandingkan protokol *signalling* lainnya. (Misalnya H323 yang protokol untuk *signalling*-nya diadopsi dari ISDN Q.SIG)

Oleh karena itu, proses autentikasi pada HTTP dapat juga digunakan pada SIP. Pada SIP, metode *Basic authentication* sudah ditinggalkan karena sudah tidak relevan lagi. Metode autentikasi yang digunakan adalah *Digest Authentication*. Pada *digest authentication*, *user name* dan *password* dienkripsi dulu sebelum dikirimkan, sedangkan pada *basic authentication* tidak dienkripsi.

Ketika pihak yang dituju/proxy menerima *request* dari *client*, proxy tersebut dapat melakukan autentikasi sebelum *request* diproses lebih lanjut. Jika *request* tidak mengandung autentikasi pada header **Authorization**, pihak yang dituju/proxy tersebut dapat memberikan *challenge-response* berupa 401 (Unauthorized) atau 407 (Proxy Authentication Required). *Client* yang bersangkutan harus (jika memang mampu) mengirimkan kembali *request* dengan autentikasi.

Contoh : A mengirimkan *request* kepada B dan gagal karena tidak ada autentikasi, B memberikan *response* 401 (Unauthorized) :

```
SIP/2.0 401 Unauthorized
Authenticate: Digest realm="GW service",
              domain="wcom.com",
              nonce="wf84f1ceczx41ae6cbe5aea9c8e88d359",
              opaque="42", stale="FALSE", algorithm="MD5"
```

A mencoba kembali mengirimkan *request* dengan autentikasi :

```
INVITE sip:UserB@ss1.wcom.com SIP/2.0
Authorization:Digest username="UserA",
              realm="GW service",
              nonce="wf84f1ceczx41ae6cbe5aea9c8e88d359",
              opaque="42", uri="sip:UserB@ss1.wcom.com",
              response="42ce3cef44b22f50c6a6071bc8"
```

Keterangan :

- **realm** adalah string yang ditampilkan kepada pengguna sehingga tahu mana *username* dan *password* yang harus digunakan.
- **domain** berisi nama domain tempat tujuan berada.
- **nonce** berisi string yang digenerate secara unik setiap kali *response* 401 dikirimkan, biasanya *base64*
- **opaque** berisi string yang harus dikembalikan oleh *client* dalam keadaan yang sama, biasanya *base64*
- **stale** bernilai “**TRUE**” jika *request* yang diterima sudah mengandung *username* dan *password* yang benar. Bernilai “**FALSE**” jika sebaliknya.
- **algorithm** menunjukkan algoritma yang digunakan pada proses autentikasi. *Default* menggunakan MD5.
- **response** terdiri dari 32 digit hex yang digunakan untuk membuktikan keabsahan *password*.

$KD(H(A1), nonce-value : H(A2))$

for MD5: $H(H(A1) : nonce-value : H(A2))$

where $A1 = username : realm : passwd$

$A2 = method : uri$

Metode keamanan dengan HTTP Authentication ini dapat digunakan untuk mencegah pembajakan pada proses registrasi, penyamaran sebagai sebuah server palsu, dan *denial-of-service-attack*.

3. 2. 2 Enkripsi *end-to-end*

Enkripsi *end-to-end* maksudnya tidak melibatkan interaksi dengan proxy dalam menentukan kunci yang dipakai pada enkripsi. Dengan demikian proxy tidak mengetahui kunci yang digunakan untuk enkripsi.

Enkripsi *end-to-end* mengandalkan kunci yang disepakati oleh dua pihak yang terlibat dalam dialog. Biasanya *message* yang dikirimkan dienkripsi dengan kunci publik dari penerima, sehingga *message* yang telah dienkripsi tersebut hanya bisa didekripsi oleh si penerima dengan kunci *private*-nya.

Contoh : isi *message* pada pengiriman *request* INVITE (pada subbab 2.2.2) dapat dienkripsi menjadi sebagai berikut :

```
INVITE sip:udin@jakarta.com SIP/2.0
Via: SIP/2.0/UDP bandung.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: Udin <sip:udin@jakarta.com>
From: Martin <sip:martin@bandung.com>;tag=1928301774
Call-ID: a84b4c76e66710@bandung.com
CSeq: 314159 INVITE
Contact: <sip:martin@bandung.com>
```

```
*****
* Content-Type: application/sdp *
* *
* v=0 *
* o=martin 53655765 2353687637 IN IP4 bandung.com *
* s=- *
* t=0 0 *
* c=IN IP4 bandung.com *
* m=audio 3456 RTP/AVP 0 1 3 99 *
* a=rtpmap:0 PCMU/8000 *
*****
```

Keterangan : tanda '*' menunjukkan bagian yang dienkripsi

Tujuan utama dari enkripsi *end-to-end* ini adalah untuk mencegah *intermediate element* (misal : proxy) mengubah isi dari *message* yang dikirimkan sehingga keutuhan (*integrity*) dan kerahasiaan (*confidentiality*) isi *message* terjaga.

3. 2. 3 Enkripsi *hop-by-hop*

Tidak seluruh bagian *message* dapat dienkripsi secara *end-to-end*. Header-header seperti **Request-URI**, **To**, dan **Via** harus dapat dibaca oleh proxy agar dapat di-*routing*-kan secara benar. Enkripsi secara *hop-by-hop* bertujuan mengenkripsi seluruh bagian *message* sehingga tidak sembarang orang bisa mengetahui isi paket. Enkripsi *hop-by-hop* juga mengenkripsi paket yang telah dienkripsi secara *end-to-end*. Tetapi proxy tetap dapat mengetahui tujuan dan ke mana paket tersebut harus di-*routing*-kan.

Enkripsi *hop-by-hop* dilakukan dengan melakukan **enkripsi pada layer transport**. Salah satu alternatif yang cukup populer adalah dengan menggunakan protokol TLS (*Transport Layer Security*).

Protokol TLS ini beroperasi berdasarkan protokol SSL (*Secure Socket Layer*). TLS terdiri dari 2 layer, yaitu :

➤ *TLS Record Protocol*

- Bagian bawah layer TLS yang berada di atas protokol transport yang *reliable* (misal. TCP).
- Menggunakan kriptografi yang simetris (misal. DES). Kunci simetris ini *di-generate* secara unik setiap kali dibangun koneksi.
- Koneksi yang dibangun bersifat *reliable*.

➤ *TLS Handshake Protocol*

- Berada di atas layer *TLS Record Protocol*.
- Menyediakan layanan kepada kedua pihak yang akan berkomunikasi untuk melakukan proses autentikasi satu sama lain, dan menentukan algoritma dari kunci yang akan dipakai untuk enkripsi.
- Dapat menggunakan kriptografi yang asimetris (misal. RSA)
- Negosiasi yang dilakukan bersifat *reliable*. Jika ada pihak yang mencoba memodifikasi akan terdeteksi.

Adanya enkripsi *hop-by-hop* dengan TLS ini menjamin keamanan komunikasi antara dua *hop*. Contoh (pada subbab 2.2.2) : Komunikasi antara Martin dan proxy-nya (*bandung.com*), proxy *bandung.com* dan proxy *jakarta.com*, serta proxy *jakarta.com* dan Udin. Dengan demikian keamanan komunikasi antara Martin dan Udin terjamin.

Penggunaan TLS pada SIP ini diimplementasikan dengan menggunakan sintaks '**SIPS**' menggantikan '**SIP**' pada URI. Sintaks ini disebut skema **SIPS URI**. (*SIP Secure*). Contoh : *sips: martin@bandung.com*

Ketika seorang pengguna menggunakan sintaks 'sips' pada **Request-URI**, artinya setiap *hop* bertanggungjawab untuk meneruskan paket tersebut melalui TLS hingga sampai di tujuan.

3.3 Keterbatasan

3.3.1 HTTP Digest Authentication

Kelemahan autentikasi menggunakan HTTP *Digest* dalam SIP adalah tidak adanya mekanisme untuk menjaga keutuhan *message (integrity)*. Proses autentikasi pada SIP hanya dimaksudkan untuk meminta keaslian pihak yang mengirimkan atau menerima suatu *request*.

3.3.2 Enkripsi *end-to-end*

Ketika enkripsi *end-to-end* digunakan, kunci enkripsi yang akan digunakan pada *response* seharusnya disertakan pada *request*. Namun sering terjadi *request* yang dikirimkan tidak menyertakan kunci enkripsi yang diinginkan, sehingga *response* yang dikirimkan dienkripsi dengan kunci yang tidak sesuai. Akibatnya terjadi kebingungan ketika *response* ini diterima.

3.3.3 Enkripsi *hop-by-hop*

Penggunaan SIPS-URI tidak menjamin tiap *hop* akan menggunakan TLS sampai kepada tujuan. Hal ini disebabkan tidak semua *hop* yang ditempuh saat *me-routing*-kan paket *support* transport dengan TLS.

Penggunaan enkripsi *hop-by-hop* dengan protokol TLS hanya bisa diimplementasikan pada protokol transport yang bersifat *connection-oriented* (misal. TCP). TLS tidak bisa digunakan pada UDP.

Salah satu solusi yang penulis tawarkan adalah dengan menggunakan IPSec. IPSec adalah protokol yang bekerja pada layer *network*. Alasannya karena baik TCP maupun UDP dapat bekerja di atas IP. Pada tulisan ini tidak akan dibahas mengenai IPSec karena tidak terintegrasi dengan aplikasi SIP. IPSec biasanya diimplementasikan pada level sistem operasi pada *host*

BAB 4

IMPLEMENTASI KEAMANAN PADA SIP

Salah satu komunitas yang mengembangkan *software* berbasis SIP adalah VOCAL¹ (*Vovida Open Communication Application Library*). Ada beberapa versi VOCAL, penulis menggunakan VOCAL-1.5.0 untuk implementasi SIP

Pada bab 3 telah dibahas mengenai masalah keamanan pada SIP dan mekanisme untuk menanganinya. Pada bagian ini, penulis akan membahas implementasi dari mekanisme keamanan tersebut yang terdapat pada VOCAL-1.5.0.

4.1 Autentikasi

Tujuan dari autentikasi ini adalah untuk mengetahui keaslian identitas pihak yang mengirimkan/ menerima suatu *request*. Pada implementasinya ada 2 jenis autentikasi, yaitu:

- Autentikasi yang berbasis *password*.
Suatu *client* diminta mengirimkan *password* sebelum melakukan operasi. Ada 2 jenis *password* yaitu *read-only username/password* dan *read- write username/password*.
- *Access control* yang berbasis *IP address*.
Metode ini kurang aman karena adanya ancaman *IP address spoofing*. Meskipun demikian pengaturannya lebih mudah dilakukan.

Source-code untuk autentikasi berbasis *password* pada VOCAL-1.5.0 disimpan di direktori `provisioning/pslib/PSSecret.cxx` (dapat dilihat pada lampiran A).

¹ VOCAL adalah salah satu *project* yang dikembangkan oleh vovida.org. *Project* ini bersifat *open-source* dan bertujuan mengembangkan aplikasi dan layanan VoIP di masyarakat. VOCAL juga menyediakan *source code* (C++) berbasis SIP yang dapat di-*download* secara bebas.

Username dan *password* untuk autentikasi ini disimpan dalam direktori `/usr/local/vocal/etc/vocal.passwd`

Format : `user:encrypted_password:value` (1 = *readwrite*, lainnya *readonly*)

Contoh isi direktori `/usr/local/vocal/etc/vocal.passwd` :

```
vocal:CJHjOT8TwnSiQ:1
reader:1K7ck5bhv5bYY:0
```

4.2 Enkripsi

Tujuan dari enkripsi ini adalah untuk mencegah *sniffer*. Implementasi enkripsi ini memerlukan protokol TLS (*Transport Layer Secure*) .

Source-code untuk TLS pada VOCAL-1.5.0 disimpan di direktori `util/transport/TlsConnection.cxx` (dapat dilihat pada lampiran B).

Adapun kelemahan dari kode ini adalah rentan terhadap serangan *man-in-the-middle*, karena *client* tidak melakukan pemeriksaan terhadap sertifikat yang diterimanya. Mungkin ini perlu diperbaiki pada implementasi di masa yang akan datang.

Penulis telah mencoba meng-*compile* program ini pada *Mandrake Linux* 9.2 (gcc versi 3.31). Program ini membutuhkan *development libraries* '*openssl*'. Langkah selanjutnya adalah membuat sertifikat digital (*cert.pem*) yang disimpan pada direktori `/usr/local/vocal/etc`. Sertifikat digital ini digunakan untuk keamanan dalam menentukan kunci publik yang akan digunakan untuk enkripsi.

Untuk membuat sertifikat digital, masuk ke direktori `/usr/local/vocal/etc` , kemudian ketik perintah di bawah ini sebagai *root* :

- `/usr/share/ssl/misc/CA.pl -newca`
- `openssl req -new -nodes -keyout key.pem -outform PEM -out newreq.pem`
- `/usr/share/ssl/misc/CA.pl -sign`

Sertifikat digital (*cert.pem*) yang dihasilkan dapat dilihat pada halaman selanjutnya.

Keterangan : sertifikat digital ini hanya contoh (*self-signed*)

Certificate:

Data:

Version: 3(0x2)

Serial Number: 1 (0x1)

Signature Algorithm: md5WithRSAEncryption

Issuer: C=ID, ST=Jawa Barat, L=Bandung, O=DSP-ITB, CN=Martin

Validity

Not Before: Jun 2 16:54:53 2004 GMT

Not After : Jun 2 16:54:53 2005 GMT

Subject: C=ID, ST=Jawa Barat, L=Bandung, O=DSP-ITB, CN=Martin

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:c4:ed:05:8d:dd:38:a3:50:fe:34:a8:2a:21:b3:
7a:f3:24:10:e6:37:13:32:3a:bb:66:03:a5:87:38:
6a:f9:8f:21:88:a6:06:0b:7a:33:db:c2:a3:e2:d8:
3c:2b:c7:ba:27:e2:11:d7:22:dd:17:a8:b9:8a:36:
ae:b4:27:ff:8f:3e:50:d7:cb:8e:dd:74:f7:9a:6a;
96:9e:22:25:bb:8c:33:eb:e7:e2:43:0d:12:ff:00:
49:b2:91:8c:36:19:f8:fd:0b:8e:af:24:8e:e6:e9:
c1:35:d8:d9:2f:aa:a9:0b:17:f4:79:7a:9f:37:8a:
d9:5e:77:aa:bf:e0:7a:7a:ad

Exponent: 65537 (0x10001)

X509v3 extensions:

X509v3 Basic Constraint:

CA:FALSE

Netscape Comment:

OpenSSL Generated Certificate

X509v3 Subject Key Identifier:

40:02:6F:D9:31:E4:63:50:5F:52:8C:11:1E:B7:22:2D:74:60:85:98

X509v3 Authority Key Identifier:

keyid:96:0A:20:7F:9C:71:FB:F7:7D:93:B7:B9:3F:52:4D:A2:41:C2:FF:BC

DirName:/C=ID/ST=Jawa Barat/L=Bandung/O=DSP-ITB/CN=Martin

serial:00

Signature Algorithm: md5WithRSAEncryption

14:37:92:d0:89:e8:8a:ef:8f:d9:9b:22:48:de:36:ee:b7:39:
45:df:e8:ef:74:26:4e:4d:09:68:43:95:da:f7:5b:79:e5:89:
96:1f:7a:d2:c2:9e:1d:d9:fb:13:27:d9:25:f1:80:4f:ee:3e:
5f:b2:6c:f0:98:24:75:ff:45:bd:cb:3a:31:18:90:63:0c:18:
ea:fa:32:1d:01:1c:8a:05:a5:22:38:6d:db:82:9e:41:df:56:
cd:27:15:d0:b9:2a:cb:d9:f9:99:8c:52:cb:b9:fd:b3:6c:70:
b7:bd:fc:9e:34:7a:f8:f5:97:24:47:15:d3:1f:d9:11:e3:16:
6d:07

BAB 5

KESIMPULAN

- Model serangan yang sering terjadi pada SIP pada prinsipnya sama dengan model serangan pada sistem informasi pada umumnya, yaitu meliputi :
 - *Interruption* : *Denial of Service Attack*
 - *Interception* : penyadapan sesi/dialog , *password*
 - *Modification* : pembajakan pada proses registrasi, modifikasi header dan isi paket.
 - *Fabrication* : penghentian suatu dialog secara prematur oleh pihak ketiga, server gadungan.

- Solusi dari model serangan tersebut adalah mekanisme keamanan yang meliputi proteksi kerahasiaan dan keutuhan *message (confidentiality and integrity)* serta penyediaan layanan autentikasi bagi *participant (authentication)*.

- Implementasi dari mekanisme keamanan tersebut meliputi penggunaan *password* untuk autentikasi dan *access-control*, penggunaan enkripsi baik *end-to-end* (dengan *public key cryptosystem*) maupun *hop-by-hop* (dengan protokol *TLS*) untuk menjaga kerahasiaan dan keutuhan data, dan penggunaan sertifikat digital untuk keamanan dalam menentukan kunci publik yang akan digunakan untuk enkripsi

REFERENSI

- Request for Comments: 3261 , *SIP: Session Initiation Protocol* . The Internet Society . 2002
- Request for Comments: 2543 , *SIP: Session Initiation Protocol* . The Internet Society . 1999
- Request for Comments: 2246 , *The TLS Protocol Version 1.0* . The Internet Society . 1999
- Request for Comments: 2069 , *HTTP Authentication: Basic and Digest Access Authentication* . The Internet Society . 1999
- Henning Schulzrinne , *The Session Initiation Protocol (SIP)* . Dept. of Computer Science , Columbia University , New York . 2001
- Dorgham Sisalem and Jiri Kuthan , *Understanding SIP* .
(<http://www.iptel.org/siptutorial>)
- <http://www.vovida.org> , *Voice over IP Protocols : Session Initiation Protocol* . 2001
- <http://www.vovida.org/vocal> , *VOCAL Technology Overview* . 2002
- <http://www.vovida.org/vocal> , *Securing Provisioning for VOCAL* . 2002
- <http://www.vovida.org/vocal> , [vocal-1.5.0.tar.gz](http://www.vovida.org/vocal/vocal-1.5.0.tar.gz)

LAMPIRAN

A. PSSecret.cxx

```
/* =====  
* The Vovida Software License, Version 1.0  
*  
* Copyright (c) 2000 Vovida Networks, Inc. All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions  
* are met:  
*  
* 1. Redistributions of source code must retain the above copyright  
* notice, this list of conditions and the following disclaimer.  
*  
* 2. Redistributions in binary form must reproduce the above copyright  
* notice, this list of conditions and the following disclaimer in  
* the documentation and/or other materials provided with the  
* distribution.  
*  
* 3. The names "VOCAL", "Vovida Open Communication Application Library",  
* and "Vovida Open Communication Application Library (VOCAL)" must  
* not be used to endorse or promote products derived from this  
* software without prior written permission. For written  
* permission, please contact vocal@vovida.org.  
*  
* 4. Products derived from this software may not be called "VOCAL", nor  
* may "VOCAL" appear in their name, without prior written  
* permission of Vovida Networks, Inc.  
*  
* THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED  
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES  
* OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND  
* NON-INFRINGEMENT ARE DISCLAIMED. IN NO EVENT SHALL VOVIDA  
* NETWORKS, INC. OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT DAMAGES  
* IN EXCESS OF $1,000, NOR FOR ANY INDIRECT, INCIDENTAL, SPECIAL,  
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,  
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR  
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY  
* OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE  
* USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH  
* DAMAGE.  
* =====  
* This software consists of voluntary contributions made by Vovida  
* Networks, Inc. and many individuals on behalf of Vovida Networks,  
* Inc. For more information on Vovida Networks, Inc., please see  
* <http://www.vovida.org/>.  
*/  
  
static const char* const PSSecret_cxx_Version =  
    "$Id: PSSecret.cxx,v 1.2.2.1 2003/03/13 17:46:12 bko Exp $";  
  
#include "PSSecret.hxx"  
#include <fstream>  
#include "cpLog.h"  
#include "vocalconfig.h"  
#include "Lock.hxx"  
#include "VloException.hxx"  
#include "FileStat.hxx"  
  
using Vocal::IO::FileStat;  
  
PSSecret* PSSecret::instance_ = 0;  
  
PSSecret::PSSecret()  
    :myReadLastModified(0),  
    myWriteLastModified(0)  
{  
}
```

```

void PSSecret::create(const char* readFile, const char* writeFile)
{
    if(!instance_)
    {
        instance_ = new PSSecret;
    }

    instance_->myReadFile = readFile;
    instance_->myWriteFile = writeFile;
    instance_->update(true);

    assert(instance_);
}

Data PSSecret::readEntry(const char* filename, const char* user) throw (VException& )
{
    Data tmp;

    // need to read the data
    std::ifstream openFile(filename);

    if(!openFile)
    {
        // do something
        throw VIOException("failed to read file", __FILE__, __LINE__, 0);
    }

    char buf[256];
    while(openFile && !openFile.eof())
    {
        openFile.getline(buf, 256);
        buf[255] = '\0';

        if(strncmp(buf, user, 6) == 0)
        {
            tmp = buf;
        }
    }

    openFile.close();
    if(tmp == "")
    {
        cpLog(LOG_ERR, "could not find an entry for %s in %s", user, filename);
    }
    else
    {
        cpLog(LOG_DEBUG, "found user %s in file %s", user, filename);
    }
    return tmp;
}

void PSSecret::update(bool force)
{
    Vocal::Threads::Lock x(myMutex);

    FileStat readStat(myReadFile);

    if(force || (readStat.lastModified() > myReadLastModified))
    {
        cpLog(LOG_DEBUG_STACK, "trying to read file: %s", myReadFile.c_str());

        // do this if the file has changed
        try
        {
            myReadSecret = readEntry(myReadFile.c_str(), "reader:");
            myReadLastModified = readStat.lastModified();
        }
        catch (VException& e)
        {
            cpLog(LOG_ERR, "failed to read reader password from %s (cannot get data from pserver)",
                myReadFile.c_str());
            myReadSecret = "";
        }
    }
    else
    {

```

```

        cpLog(LOG_DEBUG_STACK, "skipping rereading %s because up to date",
            myReadFile.c_str());
    }

    FileStat writeStat(myWriteFile);
    if(force || (writeStat.lastModified() > myWriteLastModified))
    {
        cpLog(LOG_DEBUG_STACK, "trying to read file: %s", myWriteFile.c_str());
        try
        {
            myWriteSecret = readEntry(myWriteFile.c_str(), "vocal:");
            myWriteLastModified = writeStat.lastModified();
        }
        catch (VException& e)
        {
            cpLog(LOG_DEBUG, "failed to read writer password from %s (cannot synchronize redundant pservers)",
                myReadFile.c_str());
            myWriteSecret = "";
        }
    }
    else
    {
        cpLog(LOG_DEBUG_STACK, "skipping rereading %s because up to date",
            myWriteFile.c_str());
    }
}

Data PSSecret::readSecret()
{
    if(!instance_)
    {
        cpLog(LOG_ALERT, "you should not create PSSecret from scratch! -- you must instantiate first");
        assert(0);
        // exit(-1);
        PSSecret::create(VOCAL_INSTALL_PATH "/etc/vocal.secret", VOCAL_INSTALL_PATH "/etc/vocal.secret"); //
        xxx we should read from the vocal.conf file
    }
    instance_>update(false);
    LocalScopeAllocator lo;
    cpLog(LOG_DEBUG_STACK, "read secret: %s", instance_>myReadSecret.getData(lo));
    if(instance_>myReadSecret == "")
    {
        cpLog(LOG_ERR, "read secret is empty!");
    }
    return instance_>myReadSecret;
}

Data PSSecret::writeSecret()
{
    if(!instance_)
    {
        cpLog(LOG_ALERT, "you should not create PSSecret from scratch! -- you must instantiate first");
        assert(0);
        // exit(-1);
        PSSecret::create(VOCAL_INSTALL_PATH "/etc/vocal.secret", VOCAL_INSTALL_PATH "/etc/vocal.secret"); //
        xxx we should read from the vocal.conf file
    }
    instance_>update(false);
    LocalScopeAllocator lo;
    cpLog(LOG_DEBUG_STACK, "write secret: %s", instance_>myWriteSecret.getData(lo));
    if(instance_>myWriteSecret == "")
    {
        cpLog(LOG_DEBUG, "write secret is empty!");
    }
    return instance_>myWriteSecret;
}

/* Local Variables: */
/* c-file-style: "stroustrup" */
/* indent-tabs-mode: nil */
/* c-file-offsets: ((access-label . -) (inclass . ++)) */
/* c-basic-offset: 4 */
/* End: */

```

B. TlsConnection.cxx

```
/* =====  
* The Vovida Software License, Version 1.0  
*  
* Copyright (c) 2000 Vovida Networks, Inc. All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions  
* are met:  
*  
* 1. Redistributions of source code must retain the above copyright  
* notice, this list of conditions and the following disclaimer.  
*  
* 2. Redistributions in binary form must reproduce the above copyright  
* notice, this list of conditions and the following disclaimer in  
* the documentation and/or other materials provided with the  
* distribution.  
*  
* 3. The names "VOCAL", "Vovida Open Communication Application Library",  
* and "Vovida Open Communication Application Library (VOCAL)" must  
* not be used to endorse or promote products derived from this  
* software without prior written permission. For written  
* permission, please contact vocal@vovida.org.  
*  
* 4. Products derived from this software may not be called "VOCAL", nor  
* may "VOCAL" appear in their name, without prior written  
* permission of Vovida Networks, Inc.  
*  
* THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED  
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES  
* OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND  
* NON-INFRINGEMENT ARE DISCLAIMED. IN NO EVENT SHALL VOVIDA  
* NETWORKS, INC. OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT DAMAGES  
* IN EXCESS OF $1,000, NOR FOR ANY INDIRECT, INCIDENTAL, SPECIAL,  
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,  
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR  
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY  
* OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE  
* USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH  
* DAMAGE.  
*  
* =====  
*  
* This software consists of voluntary contributions made by Vovida  
* Networks, Inc. and many individuals on behalf of Vovida Networks,  
* Inc. For more information on Vovida Networks, Inc., please see  
* <http://www.vovida.org/>.  
*  
*/  
  
static const char* const TlsConnection_cxx_version =  
    "$Id: TlsConnection.cxx,v 1.3 2002/09/28 00:03:34 sprajpat Exp $";  
  
#include "TlsConnection.hxx"  
#ifdef VOCAL_HAS_OPENSSL  
#include <openssl/err.h>  
#endif  
#include "cpLog.h"  
#include "Mutex.hxx"  
#include "Condition.hxx"  
  
bool TlsConnection::initThreads = false;  
  
extern "C"  
{  
    void my_locking_function(int mode, int n, const char *file, int line);  
};  
  
void my_locking_function(int mode, int n, const char *file, int line)  
{
```

```

#ifdef VOCAL_HAS_OPENSSL
    static Vocal::Threads::Mutex myMutex[2000];

    if(mode & CRYPTO_LOCK)
    {
        myMutex[n].lock();
    }
    else
    {
        myMutex[n].unlock();
    }
#endif
}

//id_function()

static void setCallbacks()
{
#ifdef VOCAL_HAS_OPENSSL
    CRYPTO_set_locking_callback(my_locking_function);
#endif
}

TlsConnection::TlsConnection()
: Connection(),
  ctx(0),
  ssl(0),
  meth(0),
  myCertificate(),
  myKey()
{
    if(!initThreads)
    {
        // set the callbacks
        setCallbacks();
        initThreads = true;
    }
}

TlsConnection::TlsConnection(Connection& other)
: Connection(other),
  ctx(0),
  ssl(0),
  meth(0),
  myCertificate(),
  myKey()
{
    if(!initThreads)
    {
        // set the callbacks
        setCallbacks();
        initThreads = true;
    }
}

bool
TlsConnection::isTls() const
{
    return ssl != 0;
}

int
TlsConnection::iclose()
{
    if(ssl)
    {
#ifdef VOCAL_HAS_OPENSSL
        cpLog(LOG_DEBUG_STACK, "closing SSL");
        SSL_shutdown(ssl);
#endif
    }
}

```

```

        SSL_free (ssl);
        SSL_CTX_free (ctx);
    #endif
    }
    cpLog(LOG_DEBUG_STACK, "closing regular connection");

    return Connection::iclose();
}

```

```

const TlsConnection&
TlsConnection::operator=(const TlsConnection& x)
{
    if(&x != this)
    {
        this->Connection::operator=(x);
        ctx = x.ctx;
        ssl = x.ssl;
        meth = x.meth;
        myCertificate = x.myCertificate;
        myKey = x.myKey;
    }
    return *this;
}

```

```

ssize_t
TlsConnection::iread(char* buf, size_t count)
{
    if(ssl)
    {
    #ifdef VOCAL_HAS_OPENSSL
        while(1)
        {
            ssize_t t = SSL_read(ssl, buf, count);
            if(t == -1)
            {
                ERR_print_errors_fp(stderr);
                int myerr = SSL_get_error(ssl, t);
                if(myerr == SSL_ERROR_WANT_READ)
                {
                    // select here to wait for the right bits but keep
                    // from looping relentlessly. this only will work
                    // on UNIX.
                    int fd = SSL_get_fd(ssl);

                    if(fd >= 0)
                    {
                        fd_set rfds;
                        struct timeval tv;
                        int retval;

                        FD_ZERO(&rfds);
                        FD_SET(fd, &rfds);
                        tv.tv_sec = 5;
                        tv.tv_usec = 0;

                        retval = select(fd + 1, &rfds, NULL, NULL, &tv);

                        if(retval <= 0)
                        {
                            {
                                errno = EIO;
                                return -1;
                            }
                        }
                    }
                    else
                    {
                        return -1;
                    }
                }
            }
            else
            {
                return t;
            }
        }
    #endif
    }
    return -1;
}

```

```

        }
        else
        {
            return t;
        }
    }
}
#endif
}
return Connection::iread(buf, count);
}

ssize_t TlsConnection::iwrite(char* buf, size_t count)
{
    if(ssl)
    {
#ifdef VOCAL_HAS_OPENSSL
        ssize_t t = SSL_write(ssl, buf, count);
        if(t == -1)
        {
            ERR_print_errors_fp(stderr);
            int myerr = SSL_get_error(ssl, t);
            if(myerr == SSL_ERROR_WANT_READ || myerr == SSL_ERROR_WANT_WRITE)
            {
                return 0;
            }
        }
        return t;
#endif
    }
    return Connection::iwrite(buf, count);
}

int
TlsConnection::initTlsClient()
{
#ifdef VOCAL_HAS_OPENSSL
    SSL_load_error_strings();
    meth = TLSv1_client_method();
    SSL_load_error_strings();
    ctx = SSL_CTX_new (meth);
    if(ctx == 0)
    {
        return -1;
    }

    ssl = SSL_new (ctx);
    if(ssl == 0)
    {
        return -1;
    }
    int err;

    err = SSL_set_fd (ssl, _connId);

    if(!err)
    {
        cpLog(LOG_DEBUG, "failed to connect socket to TLS!");
        return -1;
    }

    while(1)
    {
        err = SSL_connect (ssl);

        if(err <= 0)
        {
            // check for ERROR_WANT_READ / ERROR_WANT_WRITE
            int myerr = SSL_get_error(ssl, err);

            if(myerr == SSL_ERROR_WANT_READ || myerr == SSL_ERROR_WANT_WRITE)
            {
                cpLog(LOG_DEBUG, "try again!\n");
                usleep(1000);
            }
        }
    }
}

```

```

    }
    else
    {
        // issue
        cpLog(LOG_DEBUG, "TLS error: %d ( %s)", err, ERR_error_string(SSL_get_error(ssl, err),0));
        break;
    }
}
else
{
    break;
}
}

return err;
#else
return -1;
#endif
}

```

```

int
TlsConnection::initTlsServer(const char* certificate,
                             const char* key)
{
#ifdef VOCAL_HAS_OPENSSL
    // check for TLSness

    char test[100];

    int bytes = ::recv(_connId, test, 32, MSG_PEEK);

    test[bytes-1] = '\0';

    if(strcmp(test, ".") == 0)
    {
        cpLog(LOG_DEBUG_STACK, "TLS connection!\n");
    }

    char buf2[256];
    char* bptr = buf2;

    for(int i = 0; i < bytes ; ++i)
    {
        sprintf(bptr, "%2.2x", test[i]);
        bptr += 2;
    }
    cpLog(LOG_DEBUG_STACK, "%s %d", buf2, bytes);

    myCertificate = certificate;
    myKey = key;

    cpLog(LOG_DEBUG_STACK, "initializing TLS server connection");

    SSLey_add_ssl_algorithms();
    meth = TLSv1_server_method();
    SSL_load_error_strings();
    ctx = SSL_CTX_new (meth);
    if(ctx == 0)
    {
        cpLog(LOG_ERR, "no ctx");
        return -1;
    }

    // at this point, you need your certs.
    LocalScopeAllocator lo1;
    LocalScopeAllocator lo2;

    if (SSL_CTX_use_certificate_file(ctx,
                                     myCertificate.getData(lo1),
                                     SSL_FILETYPE_PEM) <= 0)
    {
        ERR_print_errors_fp(stderr);
        cpLog(LOG_ERR, "failed to set certificate file %s",
              myCertificate.getData(lo1));
        return -1;
    }

```

```

}

if (SSL_CTX_use_PrivateKey_file(ctx,
                                myKey.getData(lo2),
                                SSL_FILETYPE_PEM) <= 0)
{
    ERR_print_errors_fp(stderr);
    cpLog(LOG_ERR, "failed to set key file %s",
          myKey.getData(lo1));
    return -1;
}

int err;
assert(ctx != 0);
ssl = SSL_new (ctx);
if(ssl == 0)
{
    cpLog(LOG_ERR, "failed to create new ssl");
    return -1;
}
assert(_connId >= 0);
SSL_set_fd (ssl, _connId);

while(1)
{
    err = SSL_accept (ssl);
    if(err <= 0)
    {
        // check for ERROR_WANT_READ / ERROR_WANT_WRITE
        int myerr = SSL_get_error(ssl, err);

        if(myerr == SSL_ERROR_WANT_READ || myerr == SSL_ERROR_WANT_WRITE)
        {
            cpLog(LOG_DEBUG, "try again\n");
            vusleep(1000);
        }
        else
        {
            // issue
            cpLog(LOG_DEBUG, "TLS error: %d ( %s )", err, ERR_error_string(SSL_get_error(ssl, err),0));
            break;
        }
    }
    else
    {
        break;
    }
}
return err;
#else
return -1;
#endif
}

```

```

SSL*
TlsConnection::getSsl()
{
    return ssl;
}

```

```

bool
TlsConnection::hasTls()
{
#ifdef VOCAL_HAS_OPENSSL
    return true;
#else
    return false;
#endif
}

```

```
Data
TlsConnection::getErrMsg(int e)
{
    char tmp[1024];

#ifdef VOCAL_HAS_OPENSSL
    ERR_error_string_n(SSL_get_error(ssl, e), tmp, 1024);
#endif
    return Data(tmp);
}

/* Local Variables: */
/* c-file-style: "stroustrup" */
/* indent-tabs-mode: nil */
/* c-file-offsets: ((access-label . -) (inclass . ++)) */
/* c-basic-offset: 4 */
/* End: */
```