

APLIKASI KEAMANAN BERBASIS J2EE

TUGAS AKHIR

MATA KULIAH EC 7010

KEAMANAN SISTEM LANJUT

Oleh :

NAMA : SITI MARDLIYAH

NIM : 23203147



**PROGRAM MAGISTER TEKNIK ELEKTRO
BIDANG KHUSUS TEKNOLOGI INFORMASI- DIKMENJUR
INSTITUT TEKNOLOGI BANDUNG
2004**

DAFTAR ISI

DAFTAR ISI

DAFTAR GAMBAR

ABSTRAK

BAB I	PENDAHULUAN	
	1.1 Permasalahan	1
	1.2 Tujuan Penulisan	1
BAB II	KONSEP DASAR TEKNOLOGI J2EE	
	2.1 Java dan Platform J2EE	2
	2.2 <i>Enterprise</i> (perusahaan) saat ini	2
	2.3 J2EE	3
	2.4 Arsitektur Sistem	4
	2.4.1 Arsitektur <i>2-tier</i>	4
	2.4.2 Arsitektur <i>3-tier</i>	5
	2.4.3 Arsitektur <i>n-tier</i>	6
	2.4.4 Arsitektur Perusahaan	7
	2.5 Teknologi J2EE	8
	2.5.1 Teknologi komponen	8
	2.5.2 Teknologi <i>Service</i>	8
	2.5.3 Teknologi Komunikasi	8
BAB III	MODEL KEAMANAN	
	3.1 Arsitektur Keamanan J2EE	9
	3.1.1 Managemen kode melalui JVM dan <i>class file verifier</i> , <i>class loader</i> dan <i>security manager</i>	9
	3.1.2 <i>Platform Roles</i>	11
	3.1.3 <i>Security Role</i> dan <i>Deployment Descriptor</i>	11
	3.1.4 <i>Programmatic Security</i> (Keamanan <i>programmatic</i>)	11
	3.2 Kriptografi	12
BAB IV	KEAMANAN APLIKASI BERBASIS J2EE	
	4.1 Ancaman Keamanan dan Mekanismenya	13
	4.2 Autentikasi	13
	4.2.1 <i>Domain</i> Proteksi	14
	4.2.2 Mekanisme Autentikasi	16
	4.2.2.1 Autentikasi Web Tier	16
	4.2.2.1.1 Konfigurasi Autentikasi	17
	4.2.2.1.2 Autentikasi Hybrid	18
	4.2.2.1.3 Perubahan Identitas Autentikasi	18
	4.2.2.2 Autentikasi EJB Tier	18
	4.2.2.3 Seleksi Identitas Client	20
	4.2.2.4 Informasi Perusahaan dengan Sistem Autentikasi <i>Tier</i>	21
	4.2.3 Pola Panggilan Autentikasi	22
	4.2.4 Pembuka Lingkungan Autentikasi sebagai Referensi	22

4.3	Autorisasi	23
4.3.1	<i>Declarative</i> Autorisasi	23
4.3.2	<i>Programmatic</i> Autorisasi	24
4.3.3	<i>Declarative versus programmatic</i> Autorisasi	25
4.3.4	Isolasi	25
4.3.5	Pengaruh Seleksi Identitas	25
4.3.6	Enkapsulasi untuk Akses Kontrol	25
	4.3.6.1 Pembagian Indentitas Pengakses	26
	4.3.6.2 Identitas Pengakses Pribadi	26
4.3.7	Pengontrol Akses Untuk <i>Resource</i> J2EE	26
	4.3.7.1 Pengontrol Akses ke <i>Web Resource</i>	26
	4.3.7.2 Pengontrol Akses ke <i>Enterprise Beans</i>	27
	4.3.7.3 <i>Resource</i> Yang Tidak Diproteksi (<i>unprotected</i>)	27
4.4	Pesan yang Diproteksi	28
4.4.1	Mekanisme <i>Integrity</i>	28
4.4.2	Mekanisme <i>Confidentiality</i>	29
4.4.3	Identitas Komponen yang Sensitif	29
4.4.4	Jaminan <i>Confidentiality</i> pada <i>Web Resource</i>	29
4.5	<i>Auditing</i>	30
BAB V	KESIMPULAN	31
DAFTAR ISI		

DAFTAR GAMBAR

Gambar 2.1	Arsitektur <i>2-tier</i>	4
Gambar 2.2	Arsitektur <i>3-tier</i>	5
Gambar 2.3	Arsitektur <i>N-tier</i>	6
Gambar 2.4	Arsitektur Perusahaan	7
Gambar 4.1	<i>Domain</i> Proteksi	14
Gambar 4.2	Skenario Autentikasi	15
Gambar 4.3	Konfigurasi aplikasi tipe J2EE	19
Gambar 4.4	Arsitektur Protokol CSv2	20

APLIKASI KEAMANAN BERBASIS J2EE

ABSTRAK

Pada lingkungan perusahaan *computing*, kegagalan, kerjasama atau kurangnya *availability* pada *resource computing* dapat membahayakan kelangsungan hidup perusahaan. Organisasi harus mendapatkan langkah-langkah tertentu untuk mengidentifikasi ancaman keamanan, pada saat perusahaan mengidentifikasi ancaman keamanan maka diharapkan langkah-langkah yang diambil tersebut dapat mengurangi ancaman keamanan.

Model programming aplikasi *Java to Enterprise Edition* (J2EE) mengisolasi *developer* dari mekanisme-spesifik dalam implementasi detail aplikasi keamanan. *Platform* J2EE menyediakan isolasi dengan cara meningkatkan *portability* dalam aplikasinya, sedang perizinan dideploy pada berbagai macam lingkungan keamanan. Produk J2EE dan aplikasi J2EE tidak diperbolehkan untuk mengganti infrastruktur keamanan perusahaan yang telah ada. Perusahaan tetap berupaya untuk mendapatkan nilai yang signifikan pada saat melakukan integrasi infrastruktur tersebut. Model pemrograman untuk aplikasi J2EE diusahakan sebagai layanan keamanan yang mempengaruhi kebutuhan layanan baru atau mekanismenya.

Pembahasan makalah ini dimulai dengan konsep dan teknologi J2EE, model keamanan, beberapa konsep keamanan dan mekanismenya. Makalah ini menjelaskan sedikit tentang konsep dan teknologi J2EE, model keamanan, fokusnya adalah tentang keamanan dan karakteristik pada aplikasi.

BAB I

PENDAHULUAN

Dengan kemajuan internet, beberapa bisnis kenyataannya telah memasuki pasar baru yang terbuka diseluruh dunia sehingga sangat mempengaruhi perekonomian dunia. Melalui internet dan tumbuhnya *e-commerce*, aset-aset informasi organisasi menjadi lebih bernilai. Pergeseran informasi ekonomi ini memperkuat beberapa pebisnis berpikir untuk melakukan bisnis secara praktis. Dalam persaingan global ini, maka perlu mengadopsi teknologi baru yang menjadi faktor kunci untuk menguatkan perusahaan dalam mengeksploitasi asset informasi.

1.1 Permasalahan

J2EE menggunakan model aplikasi *multitier* terdistribusi untuk aplikasi perusahaan. Platform J2EE menetapkan kontrak *deklarative* antar *develop* dan komponen aplikasi *assemble* serta konfigurasi aplikasi dalam lingkungan operasional. Pada konteks aplikasi keamanan, aplikasi *provider* diperlukan untuk persyaratan keamanan dalam aplikasinya. Mekanisme *declarative security* digunakan dalam suatu aplikasi dan dinyatakan dalam *syntax declarative* pada dokumen yang disebut *deployment descriptor*. Aplikasi *deployer* selanjutnya dikerjakan menggunakan *tool container-khusus* untuk memetakan persyaratan aplikasi yang ada dalam *deployment descriptor* ke mekanisme keamanan yang diimplementasikan oleh server J2EE dan *Web Container*.

Programmatic security dihubungkan pada tujuan keamanan yang dibuat oleh *security-aware* aplikasi. *Programmatic security* berguna pada saat *declarative security*-nya tidak cukup untuk menyatakan model aplikasi keamanan. Pada saat aplikasi membentuk otorisasi (*authorization*) pada saat parameter melakukan *call* pada *enterprise bean* atau *web container*, maka aplikasi lain membatasi akses berdasarkan informasi *user* yang disimpan dalam database. Pada aplikasi J2EE dan *Java web service* dibuat dalam komponen yang dapat dideploy pada *container* yang berbeda. Komponen ini digunakan untuk membangun beberapa *tier* aplikasi *enterprise*. Adapun tujuan arsitektur keamanan J2EE untuk mencapai keamanan *end-to-end* dalam setiap keamanan *tier*. Produk J2EE dan aplikasi J2EE tidak memperbolehkan untuk mengganti infrastruktur keamanan perusahaan yang telah ada. Perusahaan berupaya untuk mendapatkan nilai yang signifikan pada saat melakukan integrasi infrastruktur yang ada. Model pemrograman untuk aplikasi J2EE diusahakan sebagai layanan keamanan yang mempengaruhi kebutuhan layanan baru atau mekanismenya.

1.2 Tujuan Penulisan

Tujuan penulisan makalah ini adalah untuk memberikan tinjauan umum mengenai berbagai aspek keamanan dan mekanismenya serta implementasi J2EE. Adapun makalah ini meliputi :

- Konsep dasar teknologi J2EE,
- Model Keamanan J2EE,
- Aplikasi Keamanan berbasis J2EE

BAB II KONSEP DASAR TEKNOLOGI J2EE

2.1 Java dan Platform J2EE

Java adalah bahasa pemrograman berorientasi obyek yang dibuat oleh Sun Microsystems pada tahun 1991. Java didesain untuk menjadi bahasa yang mudah, kecil dan *portable* terhadap berbagai platform. Pada saat program dikompilasi program akan diterjemahkan ke kode mesin atau instruksi prosesor yang spesifik pada prosesor. Lingkungan pengembang java ada dua bagian, yaitu *java compiler* dan *java interpreter*. Evolusi java merupakan pengembangan *applet* untuk dijalankan di *browser*, model pemrogramannya saat ini telah mampu mendukung aplikasi *enterprise* (perusahaan) dengan baik. Java hanya dalam waktu lima tahun telah menarik untuk level yang sangat tinggi dalam teknik dan bisnis komunikasi.

Dari permulaannya, java telah mencetuskan model pemrograman dan teknologi baru yang berbeda *domain-ranging* dari peralatan, aplikasi telepon untuk perusahaan. Pada waktu yang sama java telah bertindak sebagai katalis dalam pembuatan teknologi domain yang hasilnya kuat dan aman. *Java's enterprise computing platform* yang dikenal sebagai *Java 2 Platform, Enterprise Edition (J2EE)* merupakan salah satu domain. J2EE menggunakan model aplikasi *multitier* terdistribusi untuk aplikasi perusahaan. Aplikasi logik dibagi atas komponen-komponen sesuai fungsinya dan komponen-komponen aplikasi lainnya yang membuat J2EE terinstal dikomputer yang berbeda bergantung pada *tier* di lingkungan J2EE. *Two multitier J2EE* aplikasi terbagi atas:

- Komponen *client-tier* berjalan di komputer klient
- Komponen *web-tier* berjalan di J2EE server
- Komponen *Business-tier* berjalan di J2EE server
- *Enterprise Information System (EIS)-tier software* berjalan di EIS server

2.2 Enterprise (perusahaan) saat ini

Adanya pergeseran bisnis praktis mempercepat level pengembangan aplikasi. Penyimpanan dan alokasi waktu untuk pengembangan aplikasi telah disembunyikan, saat permintaan yang kompleks bertambah. Walaupun perhatian tersebut memberikan informasi kecil bagi pengembang, revolusi ini mendorong perubahan teknologi dan ekonomi yang membentangi dengan cepat. Berikut ini telah dibuat beberapa hal baru yang menarik untuk pengembang aplikasi perusahaan diantaranya:

- *Responsiveness*
Tanpa batas waktu yang selalu dipentingkan, perubahan informasi akan cepat mendukung ekonomi untuk merespon secara langsung dengan segera dan informasi yang kritis itu ditetapkan sehingga persaingan semakin tinggi.
- *Programming Productivity*
Pengadopsian langsung teknologi tidak cukup kecuali jika digunakan dengan tepat berdasarkan pada sesuatu yang penting dan integrasi yang sesuai dengan teknologi yang relevan. Selanjutnya mampu untuk mengembangkan dan mendeploy aplikasi secara efektif dan cepat. Pencapaian ini dapat dilengkapi dengan bermacam-macam teknologi dan standar yang telah dikembangkan, sehingga membutuhkan beberapa pengembangan keahlian untuk mendapatkan dan mencari masalahnya sendiri. Lebih jauh perubahan standar merupakan sesuatu yang menarik untuk memperoleh teknologi yang lebih efisien.

- *Reliability and Availability*
Pada saat ekonomi internet *downtime* dapat menyebabkan sesuatu yang fatal untuk suksesnya bisnis. Mampu menyediakan operasi berbasis web untuk menjalankannya, dan saat pencarian dijalankan sesuatu yang kritis itu berhasil. Jika tidak cukup seseorang harus mampu menjamin *reliability* transaksi bisnisnya sehingga mereka akan diproses dengan lengkap dan akurat (teliti).
- **Keamanan**
Internet tidak hanya menambah angka secara eksponensial pada *user* tetapi juga nilai informasi perusahaan, selanjutnya keamanan informasi menjadi hal yang utama untuk dikembangkan. Teknologi menjadi lebih dikembangkan menyebabkan aplikasi perusahaan lebih rumit dan lebih kompleks, sehingga untuk mengimplementasikan model keamanan yang efektif menjadi bertambah sulit.
- *Scalability*
Kemampuan aplikasi tumbuh sesuai permintaan baru keduanya yaitu operasi dan *user*. *User* merupakan sesuatu yang penting ketika aplikasi berbasis *user* menjadi potensial untuk jutaan *user* individu melalui internet. Skala efektif tidak hanya membutuhkan kemampuan untuk menghendel pertambahan angka yang besar bagi *client* tetapi juga efektif untuk menggunakan sistem *resource*.
- *Integrasi*
Informasi telah dikembangkan sebagai kunci asset bisnis, informasi yang ada sebagai data lama dan sistem informasinya masih kuno. Perintah untuk memaksimumkan kegunaan informasi, aplikasinya butuh kemampuan untuk integrasi dengan sistem informasi yang ada, tidak perlu memudahkan *task* (tugas) teknologi pada saat ini yang sering dikembangkan pada beberapa sistem legal. Kemampuan untuk menggabungkan teknologi lama dan baru merupakan kunci suksesnya pengembangan perusahaan.

2.3 J2EE

Platform J2EE secara esensial merupakan sebuah distribusi aplikasi di lingkungan server, *platform* untuk lingkungan java yang disediakan adalah :

- Infrastruktur *runtime* untuk aplikasi *hosting*
- Sekumpulan *java extension API's* untuk membangun aplikasi.

Aplikasi yang dapat dikembangkan dengan sebuah program yang mendukung *web page*, komponen untuk implementasi transaksi database yang kompleks atau *java applet*, semua didistribusikan melalui jaringan.

J2EE merupakan standar referensi Sun's untuk pengembangan perusahaan, pertama disampaikan pada desember 1999, untuk versi 1.3 elemen-elemen intinya sebagai berikut:

- **Bahasa Java**
Java merupakan bahasa obyek oriented berasal dari C++ dengan fitur pengkodean yang sederhana seperti manajemen memory melalui *gerbage colection*, *no pointer* dan lain-lain. Java dirancang untuk "write once, run anywhere", tujuan akan tercapai menggunakan *bytecode portable*.
- **JVM/JRE**
JRE (*Java Runtime Environment*) berisi *java virtual machine*, seperti saat *compiler* dan *class-class* dasar. *Class-class* java dikompel dalam *platform* tidak bergantung *bytecode* yang dieksekusi dalam JVM.

- JSPs dan Servlet
Java Server Pages (JSP) analogi dari teknologi ASP, menyediakan kemampuan untuk membangun komposisi *web pages* dinamis pada HTML dengan menempelkan komponen dinamis, misalnya referensi *beans*.
Servlet dijelaskan sebagai *applet* yang dijalankan sebagai *web server* dan secara tradisional *Comman Gateway Interface* (CGI) digunakan untuk membangun aplikasi web yang dinamis.
- EJB
Enterprise Java Beans (EJB) digunakan untuk membangun aplikasi terdistribusi yang menyediakan *framework* komunikasi dan eksekusi untuk komponen terdistribusi. Layanan kritis yang diberikan oleh *container* EJB antara lain: manajemen transaksi, keamanan, manajemen *resource* dan *persistence*.
- JDBC
Java Database Connectivity (JDBC) merupakan sekumpulan API yang dihubungkan dengan database relasional, memanipulasi isinya, dan memproses *output* (keluaran) dengan pernyataan SQL. Banyak *vendor* database yang telah dikembangkan berbasis JDBC API.

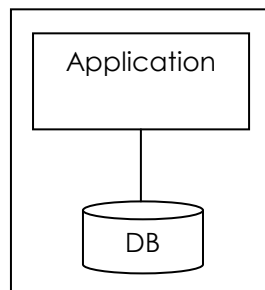
2.4 Arsitektur Sistem

Pengembangan aplikasi perusahaan sesuai dengan konsep arsitektur *n-tier*. Sistem *client/server* didasarkan pada arsitektur *2-tier*, ini merupakan penjelasan antar data logik dan presentasi/bisnis. Pada umumnya data mendukung aplikasi *entry* yang ada pada *client machine* saat *server* database mendeploy beberapa organisasi.

Arsitektur sistem J2EE antara lain: arsitektur *2-tier*, arsitektur *3-tier*, arsitektur *n-tier* dan arsitektur perusahaan. Berikut ini penjelasan singkat masing-masing.

2.4.1 Arsitektur 2-tier

Pada sebuah aplikasi tradisional *2-tier*, proses pengisian diberikan ke PC *client* pada saat server yang sederhana bertindak sebagai pengontrol trafik antara aplikasi dan data. Sebagai hasilnya, tidak hanya aplikasi itu mendapatkan kinerja yang membatasi *resource* pada PC, aplikasi dinyatakan untuk membuat beberapa *request* untuk data sebelum menghadirkan sesuatu ke *user*. Beberapa database *request* ini dapat menguatkan jaringan. Arsitektur *2-tier* dapat dilihat pada gambar 2.1.



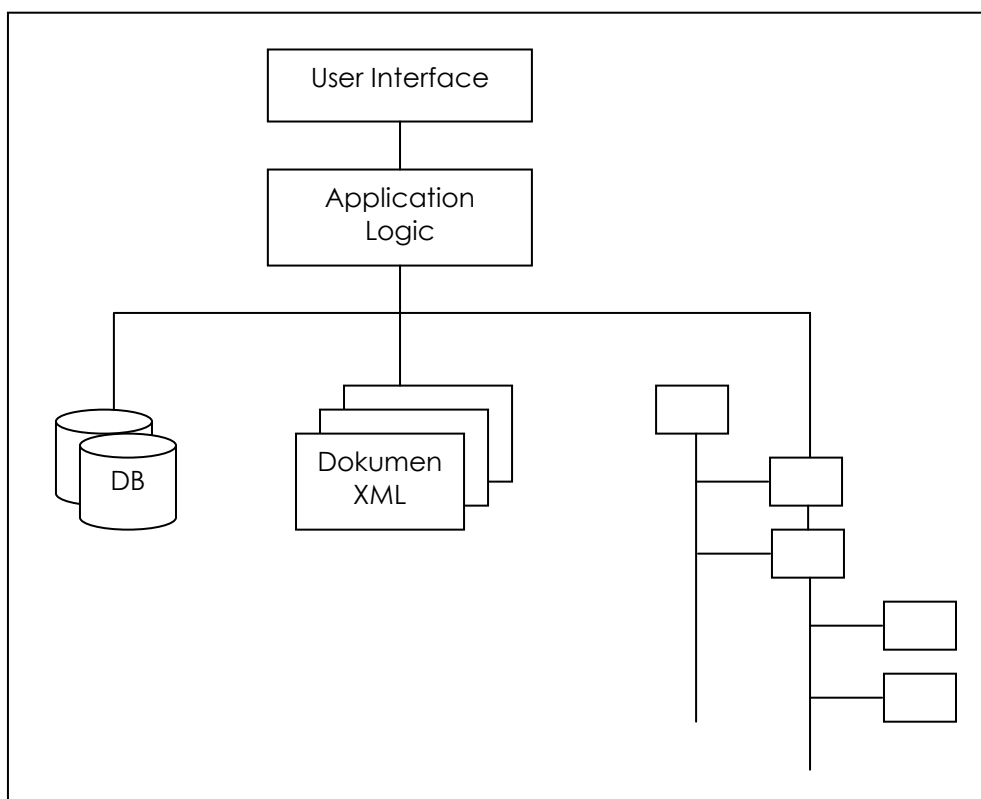
Gambar 2.1 Arsitektur 2-tier

Masalah lain dalam pendekatan *2-tier* adalah pemeliharaan. Perubahan paling kecil sesuai *rollout* lengkap untuk memasukkan data *user*. Jika memungkinkan prosesnya

otomatis, seseorang masih menampilkan dalam satu instalasi *client*. Beberapa *user* yang tidak siap *rollout* penuh sehingga mengabaikan perubahan pada group lainnya yang menuntut pembentukan perubahan yang cepat.

2.4.2 Arsitektur 3-tier

Komunikasi *software* dikembangkan dengan ide arsitektur 3-tier. Aplikasi dibagi dalam tiga *logical* layer yang terpisah, setiap layer merupakan set penentu yang baik pada suatu interface. *Tier* pertama dihubungkan ke layer presentasi dan khusus berisi *graphical user interface*. *Tier* tengah atau layer bisnis berisi logik aplikasi atau bisnis dan *tier* ketiga atau layer data berisi data yang dibutuhkan untuk aplikasi. Arsitektur 3-tier ini dapat dilihat pada gambar 2.2.



Gambar 2.2 Arsitektur 3-Tier

Tier tengah (aplikasi logik) berdasarkan kode *user calls* (melewati layer presentasi) untuk mendapatkan kembali data yang diinginkan. Layer presentasi menerima data dan formatnya untuk ditampilkan. Terpisahnya aplikasi logik dari tambahan besar *user interface* secara fleksibel untuk merancang aplikasi. Beberapa *user interface* dapat membangun dan mendeploy tanpa perubahan aplikasi logik, pemberian aplikasi logik menghadirkan kejelasan dalam penentuan *interface* untuk layer presentasi.

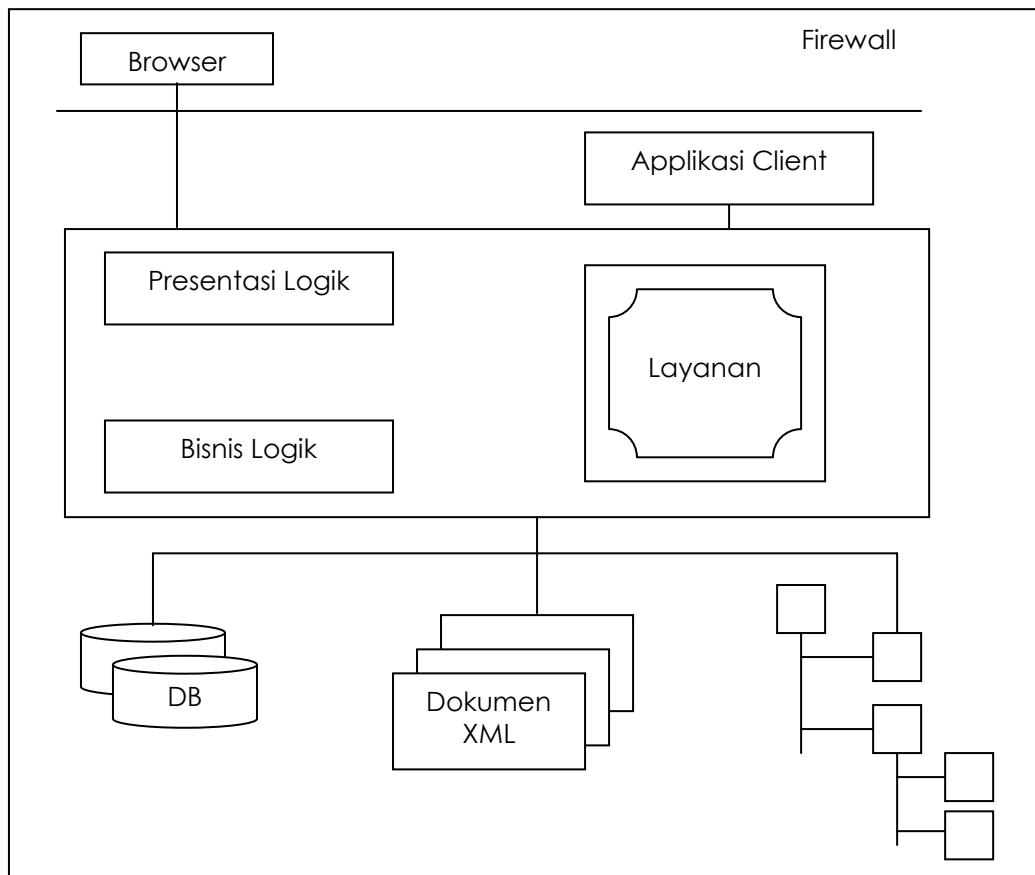
Tier ketiga berisi data yang dibutuhkan untuk aplikasi. Data ini dapat berisi beberapa *source* informasi, termasuk database perusahaan seperti oracle atau sybase, sekumpulan dokumen XML (data yang telah disimpan dalam dokumen sesuai dengan spesifikasi XML), atau layanan direktori yang lengkap seperti server LDAP. Tambahan

untuk mekanisme tradisional penyimpanan relational database, adanya beberapa *source* yang berbeda pada data perusahaan yang aplikasinya dapat diakses.

2.4.3 Arsitektur *n-tier*

Pada kenyataannya sistem *n-tier* dapat mensupport sejumlah konfigurasi yang berbeda. Arsitektur *n-tier* untuk aplikasi logik ditentukan oleh fungsi fisik. Arsitektur *n-tier* selanjutnya adalah sebagai berikut:

- *user interface* yang menghandel interaksi *user* dengan aplikasi, *user interface* dapat menjalankan *web browser* melewati *firewall*, aplikasi desktop padat atau peralatan *wireless*.
- Presentasi logik menetapkan bahwa tampilan *user interface* dan *request user* itu dihandel, bergantung apakah *user interface* itu mendukung kebutuhan untuk menjelaskan versi berbeda pada presentasi logik untuk menghandel *client* yang sesuai.
- Bisnis logik merupakan model aturan aplikasi bisnis yang sering melalui interaksi dengan data aplikasi.
- Layanan infrastruktur menyediakan tambahan fungsi yang dibutuhkan oleh bagian aplikasi seperti *messaging*, pendukung transaksi dan lain-lain.
- Layer data dimana sisa data perusahaan.



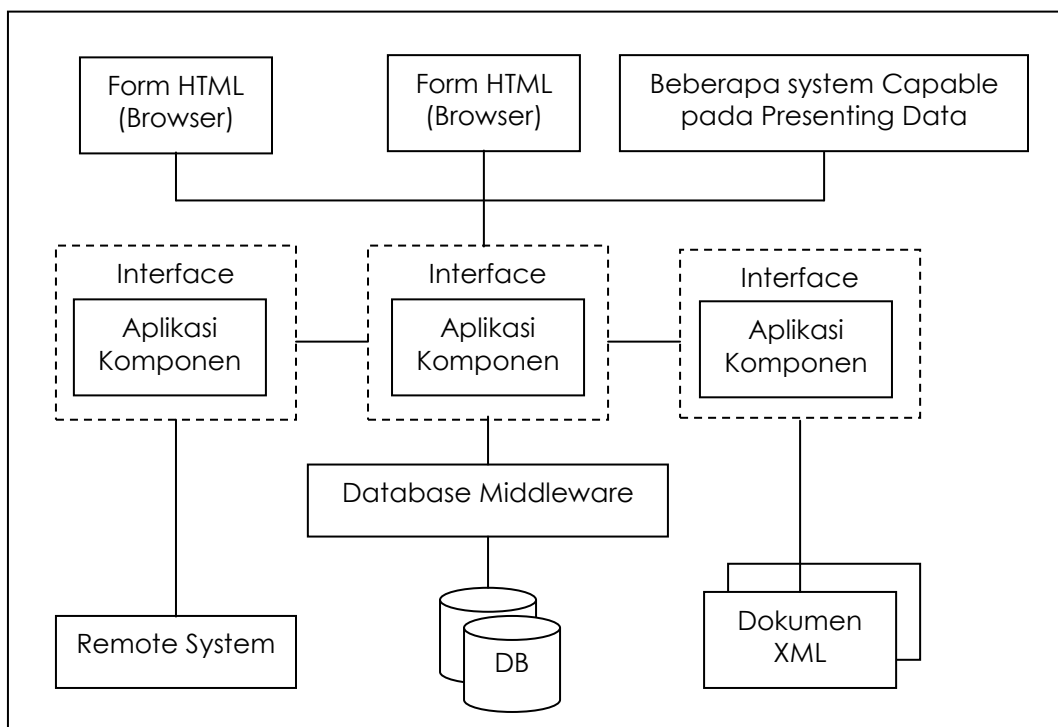
Gambar 2.3 Arsitektur *n-tier*

Aplikasi berdasarkan arsitektur pada gambar 2.3 dikerjakan dengan pola *model-view-controller* (MVC). Pada akhirnya data (model) terpisah dari informasi yang dihadirkan. Diantara aplikasi/bisnis logik (pengontrol) yang mengontrol aliran informasi. Selanjutnya aplikasi dirancang berdasarkan pada tiga komponen fungsional (model, *view* dan *controller*) berinteraksi dengan lainnya.

2.4.4 Arsitektur perusahaan

Arsitektur perusahaan berdasarkan *n-tier*, sehingga dibutuhkan perubahan persepsi. Untuk menghasilkan sistem *n-tier* pada sistem perusahaan, dibutuhkan perluasan secara sederhana pada *tier* tengah dengan cara mengizinkan beberapa objek aplikasi dari pada satu aplikasi. Objek aplikasi ini harus memiliki *interface* yang mengizinkan untuk bekerja sama dengan aplikasi lainnya.

Setiap kondisi obyek melewati *interface* yang akan menerima parameter tertentu dan mendapatkan hasil dalam set spesifikasi. Aplikasi obyek berkomunikasi dengan setiap pengguna *interface* lainnya. Adapun arsitektur perusahaan ini dapat dilihat pada gambar 2.4.



Gambar 2.4 Arsitektur Perusahaan

Dengan arsitektur perusahaan ini maka perusahaan dapat mempunyai beberapa aplikasi menggunakan sekumpulan set komponen organisasi. Ini mengembangkan standarisasi pada bisnis praktis untuk membuat satu set fungsi bisnis dengan masukan akses organisasi. Jika perubahan aturan bisnis hanya merubah pembuatan obyek bisnis maka perlu *interface* dan *subsequence* untuk beberapa obyek yang mengakses *interface*.

2.5 Teknologi J2EE

Semua arsitektur pada *platform* J2EE menyampaikan sekumpulan teknologi yang menyediakan suatu mekanisme untuk membangun lebih besar pada distribusi aplikasi perusahaan. Teknologi pada platform J2EE diantaranya adalah:

2.5.1 Teknologi komponen,

Teknologi komponen digunakan untuk menghendel beberapa bagian penting aplikasi, bisnis logik. Ada tiga tipe komponen yaitu: JSP, Servlet dan Enterprise JavaBeans

2.5.2 Teknologi Service,

Teknologi ini menyediakan komponen aplikasi untuk mendukung fungsi layanan yang lebih efisien. Beberapa layanan J2EE untuk aplikasi komponen dikelola oleh containernya.

2.5.3 Teknologi komunikasi,

Teknologi komunikasi yang paling jelas adalah aplikasi programmer yang memberikan suatu mekanisme komunikasi antar komponen yang berbeda aplikasi, apakah lokal atau jauh. Pengelompokan teknologi komunikasi ini berdasar teknologi yang memberikan bermacam-macam komponen dan layanan dalam aplikasi J2EE untuk komunikasi dengan lainnya. Adapun teknologi layanan yang sering digunakan antara lain:

- **Protokol-protokol Internet**

Client akan sangat sering membrowse sesuatu dimanapun berada diujung dunia. *Client* akan *request* dan *server response* dengan menggunakan komunikasi melalui tiga protokol utama yaitu: *Hypertext Transfer Protocol* (HTTP), *Transmission Control Protocol / Internet Protocol* (TCP/IP) dan *Secure Socket Layer* (SSL)

- **Protokol Objek yang jauh**

Aplikasi dimana komponen sering didistribusikan secara tepat ke beberapa *tier* dan server, beberapa mekanisme penggunaan komponen jauh diharapkan lebih baik seperti *client* tidak melakukan dalam komponen yang tidak lokal.

- **EXtensible Markup Language (XML)**

XML adalah bahasa *markup* berbasis teks yang menjadi bahasa standar pertukaran data di web. Tidak seperti HTML, XML tag mengidentifikasi data selain menentukan bagaimana menampilkannya. XML menjadi landasan dari penggunaan *web service*.

BAB III MODEL KEAMANAN

Enterprise development (perusahaan pengembang) membutuhkan keamanan sebagai tujuan yang disesuaikan dengan *development* dan *deployment* kode. *Development platform* dan *runtime environment* harus menyediakan keperluan untuk autentikasi dan otorisasi, data *integrity*, *audit trail*, *non-repudiasi* dan *privacy* data.

Responsibilitas untuk *task-task* ini disebarakan menurut beberapa elemen *platform* dalam tipe arsitektur *n-tier*.

3.1 Arsitektur Keamanan J2EE

Arsitektur keamanan J2EE ditetapkan sebagai *part* pada dokumentasi spesifikasi platform. Ini merupakan detail tugas manajemen keamanan dan spesifikasi tujuan arsitektur keamanan, tetapi bukan *security policy* yang khusus atau detail implementasi (seperti menggunakan teknologi keamanan khusus sesuai tujuan yang dimaksud).

3.1.1 Manajemen kode melalui JVM dan *class file verifier*, *class loader* dan *security manager*.

Dasar keamanan Java dilakukan dengan konsep "sandbox" untuk membatasi kemampuan pada kode yang tidak dipercaya karena merugikan sistem pada saat dijalankan. Secara historis, potongan kode yang tidak dipercaya seperti applet tidak akan diizinkan untuk mengakses *local disk*, hubungan jaringan terbuka dan lain-lain. Dengan pernyataan tersebut maka mendukung sertifikat untuk melewati *plug-in java*, asli dan *author* menugaskan applet untuk menyesuaikan definisinya, hal ini mungkin sebaiknya izin tiap individu applet berbasis pada *security policy*. Artinya bahwa applet tidak lama dalam menentukan *sandbox* yang gagal.

Sandbox diimplementasikan melewati JVM dan *class verifier*, tetapi juga melewati *class loader* dan *security manager/ACL manager*.

- **Keamanan JVM**

JVM memberikan *secure runtime environment* dalam mengelola memori, pemberian isolasi antara komponen-komponen yang dieksekusi pada *namespace* yang berbeda, pengecekan susunan *array* dan sebagainya. Metoda dinamis dialokasikan ke bermacam-macam *area* memori (*method area*, *GC heap*, *thread stack*) artinya bahwa tidak memungkinkan pelaku untuk menentukan apakah memori *area* dapat mencoba menyisipkan instruksi kegagalan. Pengecekan *array* mencegah akses memori yang tidak direferensikan.

- **Arsitektur Class Loader**

Ada dua type pada *class loader* yaitu *primordial class loader* yang merupakan bagian dari JVM dan *class loader object* yang digunakan untuk menyimpan *class-class* non-essensial. Hanya ada satu *primordial class loader* yang digunakan sebagai usaha JVM untuk menyimpan *class-class* dasar, yang kadang-kadang menggunakan *OS-dependent* asli. Ada beberapa contoh *class loader object* dalam JVM, yang dapat mempercepat *running* dan digunakan untuk menyimpan obyek dari *source* seperti jaringan, *local disk* atau penyimpanan data. Pengontrolan dibuat pada *class loader object* karena penting untuk difungsikan pada *class loader*.

Class loader merespon untuk penempatan *class-class* yang di *request* oleh JVM untuk penyimpanan di lingkungan *runtime*. Bagian *responsibility* mencegah kode *unauthorised* (yang tidak diperbolehkan) dan *untrusted* (tidak dipercaya) dari penempatan kembali kode yang dipercaya dengan membuat *class-class* dasar. Sebagai contoh, *class loader* harus mencegah penempatan kembali pada *class security manager*. Dicoba menempatkan kembali pada *class* dasar dengan *class* kode kegagalan yang dihubungkan ke *class spoofing*.

Semua *class loader* dengan memperluas *primordial class loader* disimpan oleh *class loader* lainnya, yang disimpan *class loader parent*. Selanjutnya penyimpanan *class loader* dinyatakan dalam struktur *tree* dengan *primordial class loader* pada *root* dan *class-class* sebagai *leaf node*.

Jika *class loader* menyimpan sebuah *class*, semua *subsequence* untuk *request* dihubungkan ke *class-class* secara langsung melalui *class loader*. Sebagai contoh *class loader* 'A' menyimpan *class* 'Building' dan 'Building' membuat *call* (panggilan) ke *methods* dalam *class* yang dipanggil (*called*) 'Cubicle', JVM akan menggunakan *class loader* 'A' untuk menyimpan 'Cubicle' dan beberapa *class-class* lain dihubungkan ke 'Building'.

Class loader mencegah *class spoofing* dengan melewati *request* *back up tree* melalui *parent class loader* nya selama *class loader* menyimpan *class request* tercapai. Pada *class security manager*, *primordial class loader* merespon secara terus menerus *class* yang gagal tersebut sehingga tidak akan disimpan. *Class-class* itu hanya sekali disimpan dan *class-class* dasar hanya disimpan dalam *local disk* pada sistem *classpath*.

Class loader juga mencegah keamanan dalam mengelola *namespace*. *Class* yang disimpan oleh *class loader* khusus hanya dapat dihubungkan dengan *class-class* lain dalam *namespace* yang sama yaitu disimpan oleh *class loader* yang sama atau *class* orang tua (*parents*).

▪ **Security Manager dan Access Controller**

Security Manager (manager keamanan) merespon untuk pengujian dan pengimplementasian *security policy* yang dispesifikasi oleh file-file *policy*. Pada java *security policy* didelegasikan ke *access controller*. Ketika *class* membuat *request* untuk izinnya, *class* diterima oleh manager keamanan melalui *request* ke *access controller*.

Metoda pengaturan izin dihubungkan dengan kode group, izin dalam java dikelompokkan dalam proteksi domain yang dihubungkan dengan *source code*. Dengan kata lain group pada saat izin dihubungkan dengan group-group pada *class-class*, *class-class* tersebut dikelompokkan dalam group aslinya.

Signed java code dalam pemberian izin berdasarkan pada sistem *policy* yang digunakan dalam kode-kode proteksi domain. Bergantung pada perizinan yang dihubungkan dengan *source code*, applet boleh akses penuh ke sistem *resource* atau boleh menerima *subset* yang kecil. Aplikasi java gagal jika tidak dihubungkan ke manager keamanan dan selanjutnya akses penuh ke sistem *resource*.

3.1.2 Platform Roles

Spesifikasi *platform* J2EE dinyatakan dalam organisasi atau *platform roles* yang dapat digunakan untuk memutuskan *responsibilities* dalam J2EE *development* dan *deployment cycle*. Tugas-tugas itu dinyatakan dalam spesifikasi *platform* yang merupakan produk *provider*, aplikasi komponen *provider*, aplikasi *assembler*, *deployer* dan sistem administrator. Tugas-tugas ini tidak absolute untuk merespon bermacam-macam tugas yang ditetapkan berbeda untuk meningkatkan perusahaan *developmet* dan metodologi *deployment*.

Pada tugas-tugas ini, paling jelas adalah implikasi keamanan. Produk *provider* dan aplikasi komponen *provider roles* merespon pengembangan *source code*, *assembler* merespon untuk menyatakan metode perizinan dan tugas-tugas keamanan (*security roles*), *deployer* memeriksa view keamanan pada aplikasi *deployed* dan mengisi tugas yang prinsip, dan sistem administator merupakan prinsip administrasi dan menyatakan bahwa lingkungan keamanan lokal menggunakan *platform* J2EE.

3.1.3 Security Role dan Deployment Descriptor

Deployment descriptor merupakan file XML yang mengirim ke setiap EJB dan disampaikan secara *declarative* pada aspek fungsi EJB dan bentuknya dihubungkan dengan *beans* lainnya.

Satu elemen dalam *descriptor* merupakan elemen `<security-role-ref>`. Tipe elemen ini digunakan oleh *bean developer* yang menetapkan semua *security roles* digunakan dalam kode EJB. Nama-nama *security role* dihubungkan dengan *links*, yang selanjutnya disebut *elsewhere* dalam *descriptor*.

Elemen `<security-role>` digunakan untuk memanggil tugas yang dinyatakan dalam elemen `<security-role-ref>`. Sebagai contoh:

```
.
.
.
<security-role-ref>
  <role-name>root</role-name>
  <role-link>super-user</role-link>
</security-role-ref>
.
.
.
<security-role>
<description> Ini merupakan security-role untuk role "root", yang
ditetapkan diatas </description>
<role-name>super-user</role-name>
</security-role>
```

gambaran *field* dalam elemen *descriptor* `<security-role>` merupakan suatu pilihan.

3.1.4 Programmatic Security (Keamanan programmatic)

Tugas para anggota dapat ditentukan secara *programmatic* dalam lingkungan J2EE menggunakan metoda `isUserInRole` dan `getUserPrincipal` pada obyek `HttpServletRequest` untuk web.

Sebagai bagian dalam kontrak *bean-container*, *container* memberikan obyek `EJBContext`. Penghubung metoda `EJBContext` adalah `isCallerInRole` dan

`getCallerPrincipal`. `getCallerPrincipal` mendapatkan kembali suatu prinsip yang dihubungkan dengan konteks keamanan, pada saat prediksinya cukup, `isCallerInRole` merupakan metoda *Boolean* yang digunakan untuk menentukan apakah *caller* (pemanggil) masih memiliki *specification security roles* (tugas keamanan yang spesifik).

3.2 Kriptografi

Java Cryptography Extension (JCE) merupakan sekumpulan *package* yang memberikan dukungan untuk enkripsi, perubahan kunci dan algoritma *Medium Access Controll* (MAC). JCE merupakan pilihan *package* untuk J2SDK 1.3 tetapi telah diintegrasikan dalam versi 1.4. Saat ini J2SDK memasukkan beberapa fungsi kriptografi, sebagian JCE mengeluarkan beberapa fungsi yang sesuai untuk pemerintahan Amerika mengeksport penerima. JCE menggunakan konsep CSPs (*Cryptographic Service Provider*) untuk *plug in* berbeda dalam implementasi algoritma enkripsinya.

Pengaturan *package* enkripsi yang lain adalah *add-on* untuk J2SDK 1.3 yang akan diintegrasikan ke versi 1.4 adalah JSSE, *java secure socket Extension*, pemberian fungsi SSL dan TLS ke *java developer*.

BAB IV KEAMANAN APLIKASI BERBASIS J2EE

4.1 Ancaman Keamanan dan Mekanismenya

Ancaman untuk asset perusahaan yang sangat kritis dengan sedikit kategori umum antara lain:

- Menyingkap informasi yang rahasia
- Modifikasi dan merusak informasi
- Penyalahgunaan pada *resource* yang diproteksi
- Kompromi keadaan yang dapat dipertanggung jawabkan (*accountability*)
- Penyalahgunaan kompromi yang ada

Kondisi lingkungan mempengaruhi aplikasi perusahaan operator, menunjukkan kepadanya ancaman dalam bentuk yang berbeda. Sebagai contoh satu sistem tradisional, sebuah ancaman dalam menyingkap sesuatu ditunjukkan oleh perusahaan sendiri sehingga mudah diserang pada saat pencarian informasi di file-file. Pada lingkungan terdistribusi beberapa *server* dan *client*, ancamannya disingkap dari suatu hasil terbukanya jaringan yang dihasilkan.

Walaupun tidak semua ancaman butuh untuk dibatasi, ada beberapa keadaan dimana pembukaan dapat dikurangi untuk level yang dapat diterima melalui kegunaan mekanisme keamanan berikut ini: autentikasi, otorisasi, *signing*, enkripsi dan *auditing*. Bahasan selanjutnya menggambarkan tentang mekanisme keamanan dalam *platform* J2EE dan menunjukkan bagaimana mekanisme untuk aplikasi J2EE yang aman..

4.2 Autentikasi

Pada komponen distribusi penghitungan, autentikasi adalah suatu mekanisme yang *caller* (pemanggil) dan layanan *provider* membuktikan kemampuannya, salah satunya dengan bertindak untuk kepentingan *user* atau sistem khusus. Ketika terbukti dua arah, komponen menyerahkan autentikasi manual. Autentikasi yang sesuai merupakan identitas panggilan dan membuktikan bahwa partisipan merupakan contoh yang *authentic* untuk identitas. *Entity* berpartisipasi dalam panggilan tanpa menyesuaikan atau membuktikan kemampuan identitas (yang, tanpa nama) yang tidak diautentikasi.

Ketika program *client* dijalankan oleh *user* yang membuat *call* (panggilan), identitas pemanggil (*caller*) seperti *user* itu. Ketika pemanggil merupakan komponen aplikasi bertindak sebagai perantara dalam rantai panggilan umum dengan beberapa *user*, identitas boleh dihubungkan dengan *user*, dimana komponen itu bukan merupakan *user* individu. Sebagai pilihan, satu komponen aplikasi boleh memanggil lainnya dengan identitasnya sendiri dan tidak dihubungkan dengan pemanggilnya.

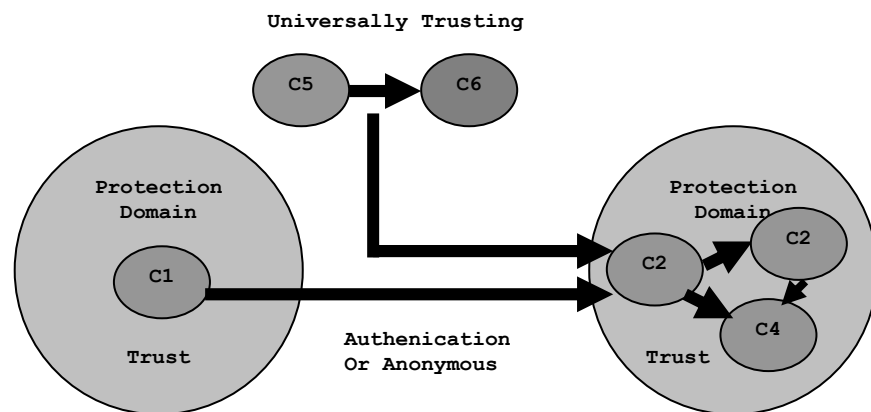
Autentikasi sering dicapai dalam dua phase. Pertama isi autentikasi yang disesuaikan oleh *performing* layanan tidak bergantung autentikasi yang butuh pengetahuan rahasia. Isi autentikasi dienkapsulasi identitasnya sehingga mampu untuk membuat *authenticator* dengan *entity* lain (dipanggil atau memanggil). Dasar autentikasi memerlukan pengontrol akses untuk isi autentikasi dan selanjutnya mampu untuk mengautentikasinya sehingga identitas tersebut dihubungkan. Diantara kemungkinan penjagaan dan mekanisme untuk mengontrol akses ke isi autentikasi antara lain :

- Pada saat *user* melakukan inisial autentikasi, proses *user* mulai mewarisi akses ke isi autentikasi,

- Ketika komponen diauthentikasi, akses ke isi autentikasi boleh ada sehingga dihubungkan dengan lainnya atau komponen yang dipercaya, seperti ke bagian aplikasi yang sama,
- Ketika komponen yang diharapkan menirukan pemanggil, pemanggil boleh menyerahkan isi autentikasi ke komponen yang dipanggil.

4.2.1 Domain Proteksi

Beberapa entitas boleh berkomunikasi tanpa memerlukan autentikasi. *Domain* proteksi diatur entitasnya untuk menerima atau mengetahui kepercayaan pada yang lain. Entitas seperti *domain* tidak butuh autentikasi ke lainnya.



Gambar 4.1 Domain Proteksi

Gambar 4.1 menggambarkan bahwa autentikasi hanya diperlukan untuk interaksi yang terbatas dengan *domain* proteksi. Ketika satu komponen interaksi dengan lainnya pada *domain* proteksi yang sama, pembatas tidak ditempatkan pada identitas yang dapat dihubungkan dengan pemanggilnya. Pemanggil boleh memperbanyak identitas pemanggil atau memilih identitas berdasarkan pengetahuan batasan otorisasi yang tidak mungkin oleh komponen yang dipanggil, karena pemanggil mampu untuk mengklaim identitas berdasarkan kepercayaan, maka tidak perlu autentikasi. Jika konsep *domain* proteksi dikerjakan maka untuk menghindari kebutuhan autentikasi, ada yang harus disesuaikan dengan batasan *domain* proteksi, sehingga kepercayaan itu tidak mencegah identitas yang tidak dibatasi.

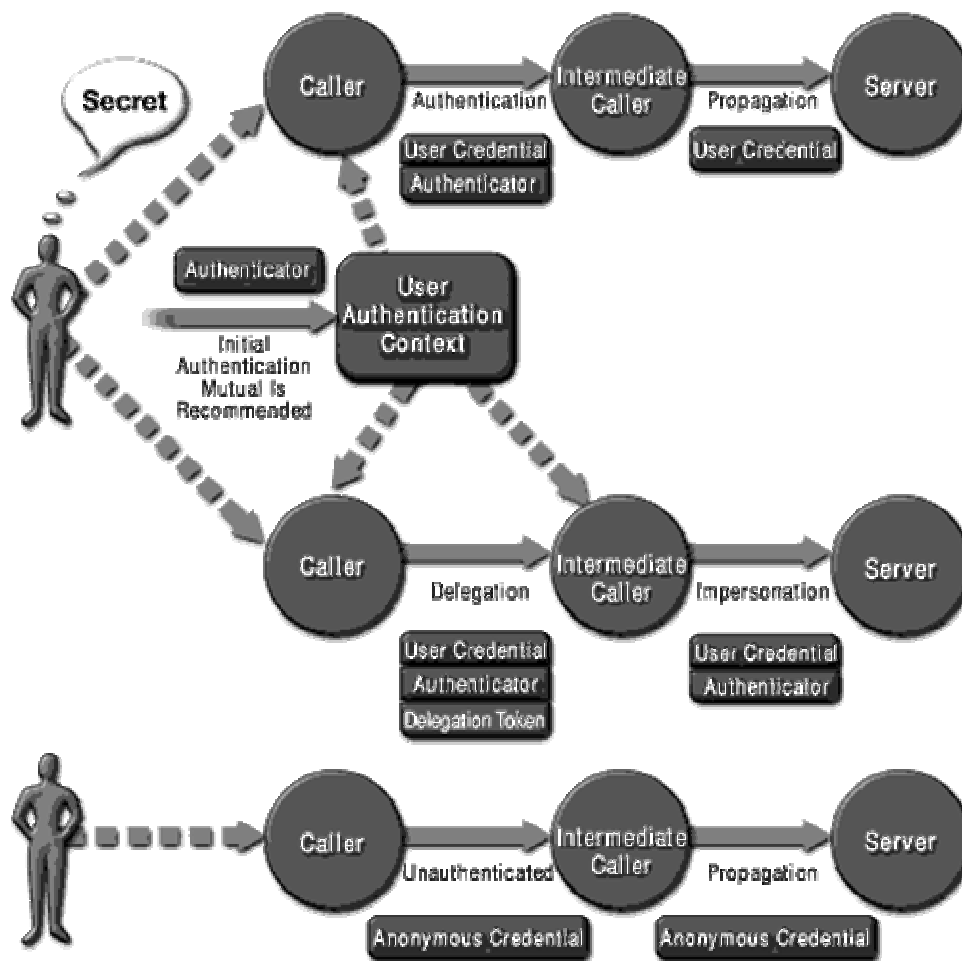
Pada arsitektur J2EE, *container* memberikan batasan autentikasi antara pemanggil luar dan komponen dalam *host*. Batasan pada *domain* proteksi tidak selalu sesuai dengan *container* itu. *Container-container* dijalankan diperbatasan dan implementasinya mendukung *domain* proteksi dalam rentang *container*. Walaupun *container* tidak membutuhkan komponen *host* dari *domain* proteksi yang berbeda, pengimplementasiannya boleh memilih untuk dikerjakan.

Untuk panggilan ke dalam, merupakan tanggung jawab *container* untuk membuat gambaran *authentic* pada identitas pemanggil yang ada ke komponen dalam *form* surat. Sertifikat X.509 dan layanan tiket Kerberos merupakan contoh surat yang digunakan dalam lingkungan *computing*. Analog untuk surat seperti passport atau lisensi pengendara yang digunakan untuk interaksi *person-to-person*.

Untuk panggilan ke luar, *container* menanggapi untuk menyesuaikan identitas pada komponen *calling*. Pada umumnya, ini dikerjakan pada *container* yang memberikan fungsi autentikasi dua arah ke penyelenggara disekitar domain proteksi pada pengembang aplikasi.

Tanpa bukti identitas komponen, interaksi *container* harus menentukan apakah ada yang cukup dipercaya antar-*container* untuk menerima gambaran *container* yang diberikan pada identitas komponen. Pada beberapa lingkungan, kepercayaan boleh diperkirakan secara sederhana, dan lainnya dinilai lebih jelas didasarkan pada autentikasi antar-*container* dan mungkin membandingkan identitas *container* dalam daftar identitas yang dipercaya. Jika sebuah bukti identitas tidak diberikan maka cukup kepercayaan antar-*container* sehingga hubungan tidak ada, *container* harus menolak atau bebas panggilan.

Gambar 4.2 mengilustrasikan konsep autentikasi dalam dua skenario, skenario autentikasi *user* dan skenario tidak autentikasi *user*. Autentikasi *user* meliputi komponen *calling* yang memakai hubungan autentikasi *user* untuk membuktikan identitasnya ke komponen selanjutnya. Ketika komponen *calling* membuat panggilan, komponen menyebarkan identitas pemanggil. Penyebaran identitas tidak terbukti, sehingga komponen hanya akan menerima jika tujuan tersebut dipercaya pemanggil, jika komponen terletak di *domain* proteksi yang sama.



Gambar 4.2 Skenario Autentikasi

Gambar penyebaran identitas itu juga berbeda dari delegasi dan peniruan berikutnya. Penyebaran layanan *provider* menunjang beban untuk menentukan apakah komponen harus menerima penyebaran identitas yang autentik. Delegasi, *user* memberikan komponen panggilan dengan akses ke isi autentikasi, sehingga memungkinkan komponen dipanggil untuk menirukan *user* pada panggilan berikutnya. Peniruan membutuhkan *user* yang dipercaya sebagai peniru untuk kepentingannya.

4.2.2 Mekanisme Autentikasi

Pada tipe aplikasi J2EE, *user* akan melakukan *container client* untuk interaksi dengan *resource* perusahaan dalam web atau EJB tiers. *Resource* yang ada untuk *user* boleh diproteksi atau tidak diproteksi. *Resource* yang diproteksi membedakan *presence authorization rules* yang membatasi akses ke beberapa subset surat identitas *non-anonymous*. Untuk akses ke *resource* yang diproteksi, *user* harus menghadirkan surat *non-anonymous* seperti identitas yang dapat sekali menilai *policy* otorisasi *resource*. Kurang percayanya hubungan antar *container client* dan *resource*, pembuktiannya harus disertai *authenticator* yang memperlihatkan *user right* untuk mengklaim identitasnya sendiri. Pada bahasan ini menggambarkan bermacam-macam mekanisme autentikasi yang didukung oleh platform J2EE dan bagaimana konfigurasinya.

4.2.2.1 Autentikasi Web Tier

Aplikasi komponen *provider* dapat menunjukkan bahwa kumpulan *web resource* (komponen web, dokumen HTML, file gambar, ringkasan arsip dan sebagainya) diproteksi secara spesifikasi oleh sebuah pembatas otorisasi. Ketika *user* tidak diautentikasi secara bertingkat untuk mengakses sebuah *web resource* yang diproteksi, *web container* akan tetap autentikasi *user* dengan *web container*. *Request* tidak akan diterima oleh *web container* sampai identitas *user* dijamin *web container* dan tergambar ke salah satu identitas sehingga boleh izin untuk akses *resource*. Autentikasi pemanggil dilakukan untuk akses pertama ke *resource* yang diproteksi yang dikenal sebagai autentikasi *lazy*.

Ketika *user* mencoba untuk mengakses ke *resource web-tier* yang diproteksi, *web container* giat melakukan mekanisme autentikasi untuk menentukan aplikasi yang dikembangkan. *Web container* J2EE harus mendukung tiga mekanisme autentikasi yaitu: HTTP autentikasi dasar, *form-based* autentikasi dan HTTPs autentikasi bersama. Sebagai tambahan dianjurkan untuk menggunakan *digest* autentikasi.

- Autentikasi dasar, pada prinsipnya autentikasi web server dengan menggunakan nama *user* dan password dari *web client*. *Digest* autentikasi adalah sebuah autentikasi *web client* ke web server dengan cara mengirim pesan *digest* bersama dengan pesan *request* HTTP. *Digest* dihitung menggunakan satu cara algoritma *hash* ke rangkaian pesan *request* HTTP dan password *client*. *Digest* dengan tipe yang lebih kecil dari *request* HTTP dan tidak berisi password. Jadi *digest* autentikasi sama persis dengan autentikasi dasar, kecuali password tidak dikirimkan dalam bentuk enkripsi, bagaimanapun mekanisme ini tidak lebih baik sehingga digunakan autentikasi dasar atau HTTPs *client* autentikasi.

- *Form-based* autentikasi melihat *customize developer* sebagai *interface* autentikasi *user* yang diwakili oleh *browser* HTTP. Seperti HTTP autentikasi dasar, *form-based* autentikasi relatif mudah diserang menggunakan mekanisme autentikasi, karena isi dialog *user* dikirim sebagai *plain text* dan *server* tujuan tidak diautentikasi.
- Dengan autentikasi bersama (*mutual authentication*), *client* dan *server* menggunakan sertifikat X.509 untuk menyesuaikan identitasnya. Autentikasi bersama terjadi di atas *channel* yang diproteksi oleh SSL. Mekanisme *hybrid* yang diutamakan salah satunya HTTP autentikasi dasar, *form-based* autentikasi, atau *digest* autentikasi melalui sebuah SSL yang diproteksi *channel*nya. SSL digunakan untuk memproteksi *authenticator* selama komunikasi jaringan dan untuk autentikasi *server* ke *client* seperti *authenticator* yang tidak berubah dengan entitas yang salah. HTTPs *client* autentikasi butuh menggunakan kunci sertifikat publik dan HTTPs (HTTPs melalui SSL). Ini merupakan pendekatan keamanan yang lebih baik dan ini membutuhkan sertifikat digital yang dapat digunakan untuk memeriksa *user* dengan pasti siapa yang telah diklaim.

4.2.2.1.1 Konfigurasi Autentikasi

Mekanisme autentikasi dikonfigurasi menggunakan elemen `login-config` pada komponen web untuk pengembangan selanjutnya. *Source code 1*, *code 2* dan *code 3* menggambarkan mekanisme autentikasi yang dibutuhkan dalam web *container*.

Source code 1 Konfigurasi autentikasi dasar HTTP

```
<web-app>
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>jpets</realm-name>
  </login-config>
</web-app>
```

Source code 2 Konfigurasi autentikasi *form-based*

```
<web-app>
  <login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
      <form-login-page>login.jsp</form-login-
page>
      <form-error-page>error.jsp</form-error-
page>
    </form-login-config>
  </login-config>
</web-app>
```

Source code 3 Konfigurasi autentikasi sertifikat *client*

```
<web-app>
  <login-config>
    <auth-method>CLIENT-CERT</auth-method>
  </login-config>
</web-app>
```

4.2.2.1.2 Autentikasi Hybrid

Pada kedua HTTP dasar dan *form-based* autentikasi, password tidak diproteksi untuk *confidentiality*. Ini mudah diserang dan dapat diatasi oleh protokol autentikasi yang dijalankan melalui SSL-proteksi yang memastikan bahwa semua isi pesan, termasuk *client authenticator*, diproteksi *confidentiality*. *Source code 4* menggambarkan bagaimana konfigurasi HTTP autentikasi dasar melalui SLL menggunakan elemen `transport-guarantee`. *Form-based* autentikasi melalui SLL dikonfigurasi dengan cara yang sama.

Source code 4 Mekanisme autentikasi SSL Hybrid

```
<web-app>
  <security-constraint>
    ...
    <user-data-constraint>
      <transport-
guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
</web-app>
```

4.2.2.1.3 Perubahan Identitas Autentikasi

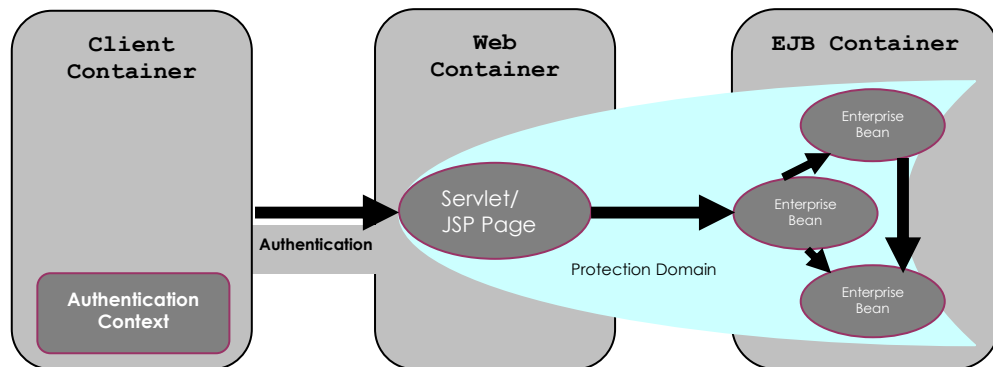
Kadang-kadang *user* membutuhkan perubahan identitas autentikasi. Ini dapat terjadi ketika sebuah rangkaian autentikasi *user* berkunjung ke web *resource* yang diproteksi dan ditolak karena tidak punya hak akses ke *resource*. Walaupun *user* dapat keluar dari *browser* dan memulai proses autentikasi, ini tidak selalu dapat diterima. Aplikasi ini memberikan *user* kesempatan untuk membuat bahasan autentikasi tidak berlaku dan autentikasi kembali sehingga lebih sesuai untuk identitas khusus.

Elemen `error-page` boleh digunakan pada pengembangan selanjutnya dalam aplikasi web untuk konfigurasi *resource* yang meliputi web *container*, pada saat proses *request* HTTP menghasilkan sebuah kode kesalahan HTTP dengan teliti. Fungsi ini dapat dilakukan dengan cara mengalihkan sebuah *request* yang tidak diberi hak ke *resource* tanpa web *container* dengan memberikan *user* kesempatan, sehingga bahasan autentikasi tidak berlaku. Sebagai contoh *resource error-handling* dapat menghasilkan *form user* yang berisi URI dan parameter *request* yang tidak diberi hak. *Form* akan memberi *user* pilihan sehingga bahasan autentikasi saat ini tidak berlaku. Pilihan dibuat tidak berlaku karena *form* berisi URI dan parameter *submit* dalam web *container*, dimana bahasan *resource* yang tidak berlaku diminta. *Resource* ini tidak berlaku dalam bahasan autentikasi saat dipanggil dengan `HttpSession.invalidate`. Kemudian *user* dialihkan lewat URI dan parameter yang ditempelkan ke *resource* asli yang tidak diautentikasi.

4.2.2.2 EJB Tier Autentikasi

Sebelumnya J2EE 1.3 dan EJB 2.0, *platform J2EE* tidak membutuhkan *container EJB* untuk implementasi khusus dalam mekanisme autentikasi. Pada beberapa lingkungan, jaringan teknologi *firewall* menghindari interaksi langsung (lewat

RMI) antara *container client* dan *enterprise beans*. Sebagai hasilnya, *container EJB* mengandalkan mekanisme autentikasi dan jaringan yang menerima web *container* yang menjamin identitas *user* untuk mengakses *enterprise beans* lewat komponen web yang diproteksi. Sebagai gambarannya terlihat pada gambar 4.3, seperti konfigurasi yang menggunakan web *container* yang dijalankan di lingkungan *domain* proteksi untuk komponen web dan *enterprise beans* yang dipanggil.



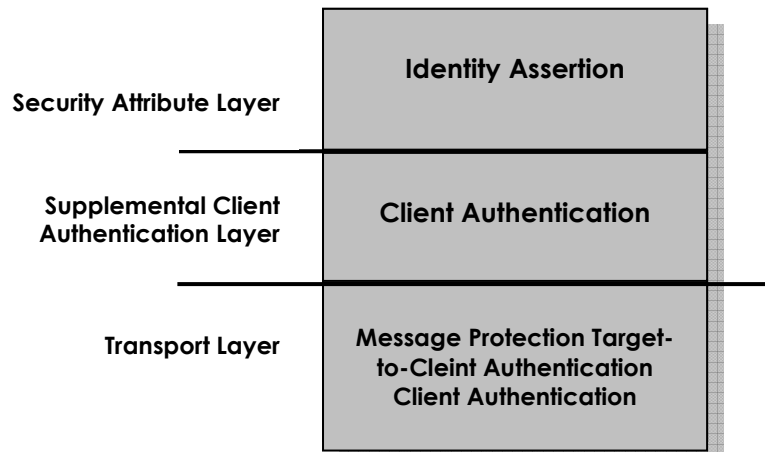
Gambar 4.3 Konfigurasi aplikasi tipe J2EE

Common Secure Interoperability (CSIv2)

Platform J2EE 1.3 membutuhkan *container EJB* dan *container client EJB* yang mendukung versi 2 pada protokol *Common Secure Interoperability (CSIv2)*. *CSIv2* merupakan standar *Object Management Group (OMG)* yang menentukan protokol kabel (*wire*) untuk membuat inovasi keamanan atas *RMI-IIOP*. *CSIv2* dirancang untuk digunakan pada lingkungan proteksi *integrity* atau *confidentiality* pesan dan *server* autentikasi ke *client* dilakukan pada layer transport, *SSL* atau *TLS*. *CSIv2* menentukan protokol *security attribute service (SAS)* yang digunakan melalui *transform* untuk melakukan autentikasi *client* dan penruan yang fungsinya tidak dapat dicapai menggunakan *transport* pokok. Mekanisme penruan untuk menonjolkan identitas, mungkin dibuat untuk penegasan identitas lain dari dirinya, berdasarkan pada kepercayaan yang diberikan selanjutnya. Penonjolan identitas dapat digunakan untuk *container J2EE* selanjutnya yang menyebarkan identitas pemanggil pada saat memanggil. *Container J2EE* melakukan mekanisme penonjolan identitas *CSIv2* untuk menyesuaikan identitas yang digunakan oleh komponen untuk memanggil komponen lainnya seperti pembuat aplikasi. Gambar 4.4 mengilustrasikan arsitektur *CSIv2*.

CSIv2 menentukan pemakaian bahasa untuk aplikasi *server* dengan menggunakan syarat keamanan komunikasi ke *client*. Aplikasi *server* menggunakan bahasa *Interoperable Object Reference (IORs)* sehingga syarat keamanan tersedia untuk memberitahu tindakan *client*-nya. Syarat keamanan merupakan tujuan yang disampaikan dalam mekanisme yang ditentukan oleh setiap layer *CSIv2*. Setiap menentukan kombinasi ini didukung dan dibutuhkan fungsi keamanan yang harus disesuaikan dengan *client* yang dituju. Ketika aplikasi *J2EE* dikembangkan dalam aplikasi *server*, pengembang harus

menentukan *policy* keamanan CSIV2 untuk komunikasi *client* dan dilakukan oleh aplikasi *server*. Aspek paling penting untuk *policy* adalah apakah tujuan itu butuh sebuah *integrity* dan/atau *confidentiality* untuk memproteksi transport, apakah tujuan membutuhkan autentikasi *client*, dan mekanisme atau mekanisme dibutuhkan untuk autentikasi *client*.



Gambar 4.4 Arsitektur Protokol CSIV2

4.2.2.3 Seleksi Identitas *Client*

Container pada J2EE *server-side* komponennya harus sesuai dengan identitas yang digunakan pada saat komponen memanggil komponen J2EE lainnya. Identitas disesuaikan dengan *container* yang bergantung pada seleksi identitas *policy* yang ditentukan oleh pembuat. Pembuat boleh menggabungkan satu dan dua seleksi identitas *policy* dengan komponen: `use-caller-identity` or `runas(role-name)`. *Policy* `use-caller-identity` menyebabkan *container* menggunakan identitas komponen pemanggil ke semua panggilan yang dibuat komponen. *Policy* `runas(role-name)` menyebabkan *container* menggunakan identitas statis yang diseleksi oleh pembuat berdasarkan prinsip identitas yang dipetakan dengan nama tugas keamanan.

Komponen *policy* untuk seleksi identitas ditentukan oleh J2EE web dan *resource* EJB. Aplikasi *developer* yang ingin mendapatkan komponen pemanggil bertanggung jawab terhadap aksi yang dilakukan komponen untuk kepentingannya sehingga harus menggabungkan *policy* dengan komponen `use-caller-identity`. Penggunaan identitas seleksi *policy* `runas(role-name)` untuk menghentikan rantai sehingga mampu digunakan untuk mendapatkan pemanggil dengan komponen khusus. *Source code 5* menggambarkan konfigurasi *policy* yang menyeleksi identitas *client* dalam sebuah EJB yang telah dikembangkan pembuat.

Source code 5 konfigurasi *policy* yang menyeleksi identitas EJB

```
<enterprise-beans>
```

```

    <entity>
      <security-identity>
        <use-caller-identity/>
      </security-identity>
      ...
    </entity>
    <session>
      <security-identity>
        <run-as>
          <role-name> guest </role-name>
        </run-as>
      </security-identity>
      ...
    </session>
    ...
  </enterprise-beans>

```

source code 6 menggambarkan konfigurasi *policy* yang menyeleksi identitas *client* dengan komponen web yang dikembangkan oleh pembuat. Kurangnya spesifikasi *run-as policy* diasumsikan dengan *use-caller-identity*.

Source code 6 konfigurasi *policy* yang menyeleksi identitas dengan aplikasi komponen web.

```

<web-app>
  <servlet>
    <run-as>
      <role-name> guest </role-name>
    </run-as>
    ...
  </servlet>
  ...
</web-app>

```

4.2.2.4 Informasi Perusahaan dengan Sistem Autentikasi *Tier*

Sistem informasi perusahaan digabungkan dengan komponen J2EE menggunakan mekanisme keamanan yang berbeda dan dioperasikan dalam *domain* proteksi yang berbeda dari akses *resourcenya*. Pada kejadian ini, *container calling* dikonfigurasi berdasar pengelolaan autentikasi ke *resource* komponen *calling*. Bentuk autentikasi ini disebut *container-managed resource manager sign on*. Arsitektur J2EE juga mengenal beberapa komponen yang butuh kemampuan untuk mengelola identitas pemanggil secara khusus dan menghasilkan *authenticator* yang cocok secara langsung. Untuk aplikasi ini, arsitektur J2EE memberikan berbagai komponen aplikasi yang digunakan sebagai mesin pencari yang dikenal dengan *application-managed resource manager sign on*. Aplikasi tersebut digunakan pada saat memanipulasi autentikasi secara detail sebagai aspek dasar fungsi komponen.

Elemen *resource-ref* pada komponen yang dikembangkan pembuat dengan mengumumkan *resource* yang digunakan oleh komponen. Nilai subelemen *res-auth* memberitahu apakah *sign on* ke *resource* yang dikelola oleh *container* atau aplikasi. Komponen yang dikelola *resource sign on* dapat menggunakan metoda `EJBContext.getCallerPrincipal` atau `HttpServletRequest.getUserPrincipal` untuk memperoleh identitas pemanggilnya. Sebuah komponen dapat memetakan identitas pemanggil ke identitas baru dan/atau autentikasi rahasia yang

dibutuhkan oleh sistem informasi perusahaan. Dengan *container-managed resource manager sign on*, *container* melakukan prinsip pemetaan untuk kepentingan komponen.

4.2.3 Pola Panggilan Autentikasi

Pada aplikasi *multitier* terdapat beberapa komponen untuk pola panggilan yang harus menghindari adanya alasan tertentu. Sebagai contoh, aplikasi panggilan yang memproteksi *resource* EJB dari *resource web* yang tidak diproteksi sehingga dapat dijalankan pada saat ada masalah. Karena paradigma *web tier autentikasi lazy* hanya memberi izin *user* sesuai autentikasinya pada saat *user* menerima akses ke *resource* yang diproteksi. Autentikasi *user* menerima kunjungan autentikasi yang diproteksi pada *resource* EJB dari *resource web* yang tidak diproteksi sehingga *user* tidak akan diberi kesempatan sesuai persyaratan autentikasi pada *resource* EJB. Autentikasi ini secara khusus digunakan pada saat *user* telah ditolak aksesnya oleh *container* EJB melalui halaman yang tidak diproteksi.

Pola panggilan lain harus dihindari adanya alasan keamanan. Sebagai contoh ketika aplikasi mengembangkan mekanisme autentikasi *hybrid*, *deployer* harus memastikan bahwa elemen `transport-guarantee` untuk setiap *web source* yang diproteksi diatur menggunakan `CONFIDENTIAL`. Selanjutnya, *client authenticator* tidak akan diproteksi secara penuh. Ketika *form-based* login digunakan melalui SSL, `transport-guarantee` pada halaman login seharusnya diatur dengan `CONFIDENTIAL`.

Self-Registrasi

Beberapa aplikasi *web-based* harus autentikasi *user* karena identitas tersebut tidak dapat mengetahui kemajuan aplikasi penggunaan pertamanya. Sebaliknya tipe autentikasi *user* di lingkungan komputer, pada saat *user* harus menunggu administrator mengatur *user*, maka aplikasi yang dibutuhkan perlu registrasi identitas autentikasi. *Self-Registrasi*, *user* butuh memberikan identitas dan passwordnya untuk memproteksi *account* dengan bentuk penambahan identifikasi, menyetujui beberapa perjanjian obligasi, dan/atau memberikan informasi *credit card* untuk pembayaran. Pada saat dialog registrasi lengkap, *user* boleh autentikasi seperlunya untuk akses disisi *resource* yang diproteksi.

Mekanisme registrasi diberikan oleh platform J2EE dengan platform-khusus. Aplikasi tersebut bergantung pada mekanisme untuk melakukan cara yang membolehkan aplikasi itu memakai fasilitas standar dan API untuk menambah platformnya.

4.2.4 Pembuka Lingkungan Autentikasi sebagai Referensi

Aplikasi komponen *provider* berdasarkan referensi dibuat untuk setiap komponen ke komponen J2EE lainnya dan untuk *resource* luar. Tambahan tugas dilokasi layanan, seperti memberitahu *deployer* disemua tempat untuk aplikasi autentikasi yang diperlukan. Sistem informasi perusahaan menggunakan elemen `ejb-ref`. Referensi sistem informasi perusahaan menggunakan elemen `resource-ref`. Kedua kejadian tersebut dibuat dalam *scope* komponen *calling*, dan dikumpulkan sebagai referensi *server* untuk menyingkap aplikasi antar-komponen /*resource* yang disebut *tree*.

Tool deployment Platform J2EE mewakili *enterprise beans* untuk referensi aplikasi *deployer* sehingga *deployer* itu mengetahui interaksi keamanan antar komponen *calling* dan *called*. *Deployer* harus menggunakan ilmu ini untuk menentukan mekanisme keamanan CSiv2 dimana keamanan itu sesuai untuk *enterprise beans* dengan

menggunakan semua cara. *Deployer* harus menggunakan ilmunya dalam berinteraksi antar-*container* yang terjadi sebagai hasil panggilan antar-komponen yang konfigurasinya sesuai dengan mekanisme keamanan antar *container* dan hubungan kepercayaan.

4.3 Autorisasi

Mekanisme otorisasi membatasi interaksi dengan *resource* yang dikumpulkan *user* atau sistem. Seperti mekanisme pemberian identitas pemanggil yang hanya diberikan kepada yang berhak untuk akses komponen. Mekanisme yang diberikan oleh platform J2EE dapat digunakan untuk mengontrol akses ke kodenya berdasarkan identitas pemilik, seperti lokasi dan penanda kode *calling*, serta identitas *user* pada kode panggilan.

Ada sebuah surat yang dibuat untuk memanggil komponen. Surat itu berisi informasi yang menggambarkan pemanggil melewati identitas atribut-nya. Pada kejadian pemanggil tanpa nama, digunakan surat khusus. Atribut yang unik merupakan identitas pemanggil dengan isi rahasia sesuai dengan yang diisuekan dalam surat. Bergantung pada tipe surat, ia dapat juga berisi atribut lain untuk menentukan pembagian otorisasi seperti anggota group dan membedakan kumpulan surat yang terhubung. Identitas dan atribut pembagi otorisasi surat digabungkan sebagai atribut keamanan pemanggil. Pada platform J2SE, atribut identitas kode digunakan oleh pemanggil untuk dimasukkan dalam atribut keamanan pemanggil. Akses untuk komponen yang dipanggil ditentukan oleh atribut keamanan pemanggil yang dibandingkan dengan kebutuhan untuk mengakses komponen yang dipanggil.

Pada arsitektur J2EE, *server container* sebagai loncatan otorisasi antara komponen *host* dan pemanggilnya. Loncatan otorisasi ada disisi loncatan autentikasi *container* sehingga otorisasi dinyatakan pada isi autentikasi yang berhasil. Untuk loncatan panggilan kedalam, *container* membandingkan atribut keamanan dari surat pemanggil dengan akses pengontrol tugas pada komponen tujuan. Jika tugas-tugas ditetapkan maka panggilan diperbolehkan, dan selanjutnya panggilan ditolak.

Ada dua dasar pendekatan untuk menentukan akses pengontrol tugas: kemampuan dan izin. Kemampuan utama apakah yang dapat dilakukan pemanggil? Siapakah yang memberi izin utama untuk mengerjakan sesuatu? Aplikasi J2EE model pemrogramannya diutamakan pada perizinan.

4.3.1 Declarative Autorisasi

Deployer menyesuaikan akses *container-enforches* untuk mengontrol tugas yang terhubung dengan aplikasi J2EE. *Deployer* menggunakan *tool* yang dikembangkan dalam pemetaan aplikasi model perizinan, yang didukung oleh tipe aplikasi *assembler*, *policy* dan mekanisme khusus di lingkungan operasional. Aplikasi model perizinan ditetapkan dalam *deployment descriptor* (gambaran pengembangan).

Gambaran pengembangan ditetapkan menggunakan logika khusus yang dikenal sebagai aturan keamanan dan dihubungkan dengan komponennya untuk menentukan kebutuhan khusus yang diizinkan untuk mengakses komponen. *Deployer* menugaskan logik khusus untuk pemanggil khusus yang disesuaikan dengan kemampuan *user* pada saat dijalankan dilingkungannya. Pemanggil menugaskan logik khusus berdasarkan nilai atribut keamanannya. Sebagai contoh, *deployer* boleh memetakan tugas keamanan untuk group keamanan di lingkungan operasional. Sebagai hasilnya beberapa pemanggil menggunakan atribut keamanan untuk menunjukkan anggota group yang ditugasi khusus untuk mewakili tugasnya. Contoh lain, *deployer* boleh memetakan tugas keamanan dalam

daftar yang berisi satu atau lebih identitas prinsip dalam lingkungan operasional. Pemanggil selanjutnya diauthentikasi oleh satu identitas yang ditugaskan khusus untuk diwakili dalam tugas tersebut.

Container EJB hanya diijinkan untuk akses metoda pemanggil yang telah mempunyai satu kekurangan khusus yang dihubungkan dengan suatu metoda. Tugas keamanan juga memproteksi kumpulan *web resource* menggunakan pola URL dan dihubungkan dengan metoda HTTP, seperti GET. *Web container* menjalankan persyaratan otorisasi yang sama untuk *container* EJB.

Kedua *tiers* digunakan untuk akses kontrol *policy* untuk menentukan waktu pengembangan, pada saat aplikasi dikembangkan. *Deployer* dapat memodifikasi *policy* yang diberikan oleh aplikasi *assembler*. *Deployer* menyaring persyaratan khusus yang dibutuhkan untuk mengakses komponen, dan menentukan hubungan diantara atribut keamanan yang diwakili oleh pemanggil dan *container* khusus. Pada beberapa *container*, pemetaan dari atribut keamanan khususnya di lingkungan aplikasi melakukan pemetaannya menggunakan satu komponen aplikasi berbeda dari aplikasi yang lain.

4.3.2 Programmatic Autorisasi

Container J2EE dibuat untuk akses pengontrol tujuan sebelum metoda pengiriman komponen panggilan. Logik atau *state* komponen pada kenyataannya tidak untuk akses tujuan. Meskipun demikian, sebuah komponen dapat menggunakan dua metoda antara lain : `EJBContext.isCallerInRole` (untuk pengguna kode *enterprise bean*) dan `HttpServletRequest.isUserInRole` (untuk pengguna *web komponen*), dilakukan dengan pengontrol akses *finer-grained*. Sebuah komponen menggunakan metoda ini untuk menentukan apakah pemanggil diperbolehkan untuk penyeleksian khusus komponen dasar yang dilakukan oleh parameter panggilan, kondisi internal komponen, atau faktor lain seperti waktu panggilan.

Aplikasi komponen *provider* pada komponen yang memanggil satu fungsi harus dinyatakan dengan pengaturan yang lengkap pada nilai `roleName` yang jelas untuk digunakan pada semua panggilan. Pengumuman ini mewakili gambaran pengembangan dengan elemen `security-role-ref`. Setiap elemen `security-role-ref` dihubungkan dengan nama khusus yang ditempelkan pada aplikasi sebagai `roleName` ke tugas keamanan. *Deployer* mencocokkan hubungan antara nama khusus yang ditempelkan pada aplikasi dan tugas keamanan yang ditentukan pada gambaran pengembangan. Hubungan diantara nama khusus dan tugas keamanan boleh berbeda untuk komponen dalam aplikasi yang sama.

Tambahan untuk pengujian spesifikasi khusus, aplikasi komponen dapat dibandingkan dengan identitas pemanggil, yang diperbolehkan menggunakan `EJBContext.getCallerPrincipal` atau `HttpServletRequest.getUserPrincipal`, yang membedakan identitas pemanggil yang ditempelkan pada kondisi komponen pada saat dibuat. Jika identitas pemanggil sama dengan pemanggil yang terkenal, komponen dapat mengizinkan pemanggil untuk melakukannya. Jika tidak, komponen dapat menghindari pemanggil dari interaksi lebih lanjut. Prinsip pemanggil menghasilkan *container* yang bergantung pada mekanisme autentikasi yang digunakan oleh pemanggil. Juga, *container* dari *vendor* yang berbeda menghasilkan prinsip yang berbeda untuk autentikasi *user* yang sama dengan mekanisme yang sama. Sejumlah variabel tetap dalam prinsip *form*, sebuah aplikasi *developer* memilih menggunakan beberapa identitas pemanggil yang berbeda, diwakili dengan *user* yang sama untuk dihubungkan dengan komponen

4.3.3 *Declarative versus Programmatic* Autorisasi

Ada *trade-off* antara akses eksternal untuk akses kontrol *policy* yang dikonfigurasi oleh *deployer* dan internal *policy* yang ditempelkan di aplikasi komponen *provider*. *Policy* eksternal lebih fleksibel sesudah aplikasi ditulis. *Policy* internal menyediakan fungsi yang lebih fleksibel saat aplikasi akan ditulis. Tambahan, *policy* eksternal itu jelas dan secara lengkap dipahami oleh *deployer*, saat *policy* internal dilupakan dalam aplikasi dan hanya boleh dimengerti secara lengkap oleh aplikasi *developer*. *Trade-off* ini harus dinyatakan sebagai pilihan model otorisasi untuk komponen dan metode khusus.

4.3.4 Isolasi

Pada saat merancang akses kontrol ada tugas untuk memproteksi *resource*, proteksi *resource* tersebut untuk menjamin bahwa *policy* otorisasi secara konsisten dilakukan menggunakan semua pola yang sesuai dengan *resource* yang boleh diakses. Sebagai contoh, ketika level metoda akses kontrol tugas digunakan oleh komponen, cara yang harus digunakan adalah metoda proteksi-rendah sehingga tidak dapat dijalankan untuk merusak *policy* yang dilakukan oleh metode proteksi yang lebih teliti. Seperti penjelasan paling signifikan ketika *state* komponen dibagi dengan metode proteksi yang berbeda atau pola URL. Aturan paling sederhana yang menyolok digunakan pada akses pengontrol tugas yang sama untuk semua pola pengaksesan dalam komponen dan bagian aplikasi yang perlu melakukan panduan kecuali kalau ada beberapa kebutuhan khusus untuk aplikasi arsitek lainnya.

4.3.5 Pengaruh Seleksi Identitas

Pada saat pengaturan aplikasi akses kontrol *policy*, aplikasi komponen *provider* berdasarkan keputusan *policy* dengan mengasumsikan identitas panggilan yang diseleksi oleh aplikasi pemanggil. Ketika panggilan dilewatkan melalui komponen selanjutnya, identitas pemanggil terdapat di dalam komponen yang dituju dan bergantung pada identitas untuk menyeleksi keputusan yang dibuat selanjutnya. Komponen yang dituju diasumsikan sebagai identitas pemanggil yang telah disebarikan selama rantai panggilan sehingga identitas pemanggil merupakan pemanggil yang diinisialisasi dalam rantai tersebut. Pada kejadian lain, komponen yang dipanggil harus diasumsikan untuk satu atau lebih pemanggil dalam pola panggilan yang akan dilakukan dengan identitas penyeleksi *policy* lain dari pada identitas yang disebarikan. Aplikasi *assembler* menanggapi komponen komunikasi sebagai identitas yang menyeleksi *policy* sebagai gambaran untuk dikembangkan. Kekurangan yang dimiliki oleh identitas digunakan untuk menyeleksi *policy* dari *assembler*, *deployer* yang harus mengasumsikan bahwa komponen akan memanggil komponen lain menggunakan identitas pemanggil.

4.3.6 Enkapsulasi untuk Akses Kontrol

Model komponen aplikasi merupakan loncatan otorisasi yang tidak diproteksi *resource*-nya, penggunaan komponen pengakses untuk implementasi rintangan otorisasi. Jika komponen pengakses menggunakan cara ini, akses kontrol salah satunya dapat menggunakan *container* eksternal, internal komponen atau keduanya.

Komponen pengakses boleh mengenkapsulasi pemetaan isi autentikasi yang sesuai untuk interaksi dengan *resource* eksternal. Ketika menggunakan prinsip pemetaan untuk autentikasi akan menguatkan akses ke *resource* sistem informasi perusahaan, mekanisme authorisasi dilakukan oleh komponen pengakses yang dapat mengontrol siapa yang berhak untuk akses pemetaan. Bergantung pada bentuk pemetaan, tugas autorisasi lebih kompleks atau kurang kompleks. Sebagai contoh jika semua akses *resource* dilakukan melalui satu konsep kekuasaan sistem informasi perusahaan sebagai tingkatan identitas, kemudian aplikasi J2EE dapat implementasi akses yang aman ke *resource* yang dibatasi siapa yang dapat menggunakan akses tersebut. Jika pemetaan isi autentikasi banyak ke banyak, kemudian konfigurasi autorisasi akses dibutuhkan untuk menetapkan apakah pemetaan dapat akses ke pemanggil dan apakah harus diasumsikan gagal jika pemanggil tidak menerima pemetaan yang dibutuhkan.

4.3.6.1 Pembagian Identitas Pengakses

Komponen pengakses diberikan untuk akses *resource* eksternal salah satunya adalah *container-managed sign on* atau *bean-managed sign on*. Perizinan dihubungkan dengan metoda-metoda pada komponen yang menjamin akses ke *resource* eksternal yang hanya dilakukan dengan prinsip J2EE yang telah diakses komponennya.

4.3.6.2 Identitas Pengakses Pribadi

Enterprise bean, seperti pembahasan *bean* selengkapnya, dapat menggunakan *bean-managed sign on* untuk *resource* eksternal. Bahasan *bean* dipercaya dalam proteksi *entity bean* untuk memetakan prinsip J2EE yang terhubung dengan prinsip *resource* eksternal, dan juga terhubung dengan *authenticator* jika perlu. Proteksi *entity bean* memegang semua pemetaan dan *bean* membatasi akses pemetaan khusus untuk prinsip yang khusus (menghasilkan `getCallerPrincipal`).

4.3.7 Pengontrol Akses ke *Resource* J2EE

Dengan tipe *client* yang menggunakan aplikasi J2EE *container* berinteraksi dengan *resource* perusahaan di web atau EJB *tier*. *Resource* ini boleh diproteksi atau tidak diproteksi. *Resource* yang diproteksi mempunyai tugas autorisasi untuk menentukan gambaran pengembangan selanjutnya untuk membatasi akses beberapa subset identitas *non-anonymous*. Untuk akses *resource* yang diproteksi, *user* harus menghadirkan surat namanya yang mungkin merupakan identitasnya untuk dinilai lawan *resource policy* autorisasi.

4.3.7.1 Pengontrol Akses ke Web *Resource*

Kontrol akses ke web *resource*, aplikasi komponen *provider* atau aplikasi *assembler* khusus untuk elemen `security-constraint` dengan subelemen `auth-constraint` di web yang dikembangkan oleh pembuat. *Source code 7* mengilustrasikan definisi *resource* yang diproteksi di web komponen yang dikembangkan pembuat.

Source code 7 konfigurasi autorisasi web *resource*

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>placeholder</web-resource-name>
```

```

        <url-pattern>/control/placeorder</url-pattern>
        <http-method>POST</http-method>
        <http-method>GET</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>customer</role-name>
    </auth-constraint>
</security-constraint>

```

4.3.7.2 Pengontrol Akses ke *Enterprise Beans*

Aplikasi komponen *provider* atau aplikasi *assembler* menentukan tugas keamanan untuk *enterprise bean* dapat menggunakan metoda khusus dalam *bean's remote*, rumah, lokal, dan interface rumah lokal menggunakan tugas keamanan sesuai yang diminta. Ini dikerjakan dengan bentuk elemen `method-permission`. Penugasan *user* digunakan untuk menentukan tugas jika sebuah *resource* diproteksi. Ketika tugas yang dibutuhkan di akses dalam *enterprise bean* maka tugasnya hanya untuk autentikasi *user*, *bean* diproteksi.

Source code 8 berisi dua *style* dalam metoda khusus. Pertama dihubungkan ke semua metoda interface (*remote*, *home*, *local*, dan *local home*) pada *enterprise bean*. Kedua dihubungkan dengan metoda khusus yang terjadi di *interface enterprise bean*. Jika ada beberapa metoda dengan nama berlebih yang sama, *style* ini dihubungkan ke semua metoda tersebut. Metoda khusus dapat dilakukan dengan kualitas selanjutnya untuk identitas metoda dengan nama berlebih sebagai parameter tandatangan (*signature*) atau dihubungkan dengan metoda khusus interface di *enterprise bean*.

Source code 8 konfigurasi otorisasi *enterprise bean*.

```

<method-permission>
    <role-name>admin</role-name>
    <method>
        <ejb-name>TheOrder</ejb-name>
        <method-name>*</method-name>
    </method>
</method-permission>

<method-permission>
    <role-name>customer</role-name>
    <method>
        <ejb-name>TheOrder</ejb-name>
        <method-name>getDetails</method-name>
    </method>
    <method>
        ...
    </method-permission>

```

4.3.7.3 *Resource* yang tidak Diproteksi

Beberapa gambaran isi aplikasi *web-tier* yang tidak diproteksi, ada beberapa pemanggil tanpa autentikasi. *Resource* yang tidak autentikasi karakteristik persyaratannya kurang dari pemanggil yang di autentikasi. Pada *web tier*, tidak menerima akses yang menyediakan aturan autentikasi sederhana.

Beberapa aplikasi juga menggambarkan *enterprise bean* yang tidak diproteksi. Sebagai contoh aplikasi sederhana yang boleh tanpa nama, tidak autentikasi *user* untuk akses EJB *resource* yang tetap. Pada EJB *tier*, aplikasi *assembler* menggunakan elemen `unchecked` pada elemen `method-permission` untuk menunjukkan bahwa metode ini dinyatakan oleh *container* khusus yang berhak, tidak bergantung pada identitas *caller*. *Source code 9* menggambarkan penggunaan elemen `unchecked`.

Source code 9 enterprise bean unchecked method-permission

```
<method-permission>
  <unchecked/>
  <method>
    <ejb-name>Catalogue</ejb-name>
    <method-name>browseSpecials</method-name>
  </method>
</method-permission>
```

4.4 Pesan yang Diproteksi

Pada sistem distribusi *computing*, sejumlah informasi yang signifikan dikirimkan melalui jaringan dalam bentuk pesan. Pesan boleh diterjemahkan dan dimodifikasi sesuai yang diharapkan dalam perubahan yang mempengaruhinya pada saat diterima. Pesan boleh ditangkap dan digunakan kembali sekali atau lebih untuk mendapatkan keuntungan lain. Pesan boleh dimonitor oleh *eavesdropper* untuk melakukan penangkapan informasi yang sebaliknya tidak ada. Seperti serangan dapat diminimumkan oleh pengguna rahasia dan mekanisme *confidentiality*.

4.4.1 Mekanisme Integrity

Integrity mechanisms (mekanisme *integrity*) menjamin bahwa komunikasi antar *entity* tidak akan merusak bagian lain, ada hal khusus yang dapat menahan dan memodifikasi komunikasi mereka. Mekanisme *integrity* dapat juga digunakan untuk menjamin bahwa pesan hanya dapat digunakan sekali.

Keutuhan pesan dijamin oleh pemberi pesan *signature* di dalam pesan. Pesan *signature* sudah diperhitungkan oleh pengguna algoritma *hash one-way* untuk merubah isi pesan menjadi tipe yang lebih kecil, pesan *digest* tetap-panjang yang ditandai (secara kriptografi di *encipher*, tipenya menggunakan mekanisme kunci publik). Pesan *signature* menjamin bahwa modifikasi pesan oleh salah satu pemanggil dideteksi oleh penerima.

Pada arsitektur J2EE, sebuah layanan *container* sebagai batas autentikasi antar pemanggil dan komponen *host*. Informasi boleh dialirkan dalam dua arah panggilan (itu berarti, sebuah panggilan mempunyai masukan, keluaran atau parameter masukan dan keluaran). *Deployer* merespon konfigurasi *container* untuk melindungi interaksi antar komponen. *Deployer* mengkonfigurasi *container* yang berbelit-belit dalam suatu panggilan untuk implementasi mekanisme *integrity* salah satunya karena panggilan melewati jaringan terbuka atau tidak diproteksi karena panggilan akan membuat komponen yang tidak dipercaya yang lain.

Satu cara melindungi keutuhan pesan tanpa membatasi ruang yang tidak perlu di lingkungan operasional adalah menangkap aplikasi khusus untuk mengetahui identitas

pesan yang harus diproteksi dengan utuh. Ruang untuk menangkap informasi ini dalam gambaran aplikasi yang dikembangkan.

4.4.2 Mekanisme *Confidentiality*

Confidentiality mekanisme (mekanisme *Confidentiality*) menjamin komunikasi pribadi diantara *entity*. *Privacy* dicapai oleh pengenkripsi isi pesan. *Symmetric* (atau bagian penyimpanan rahasia) mekanisme enkripsi umumnya butuh perhitungan untuk mengurangi *resource* dari mekanisme *asymmetric* (atau kunci publik). Pada umumnya mekanisme *asymmetric* digunakan untuk mengamankan perubahan kunci enkripsi *symmetric* yang digunakan untuk enkripsi trafik pesan.

Deployer merespon konfigurasi *container* yang memakai mekanisme *confidentiality* yang menjamin sensitifitas informasi yang tidak memperlihatkan tiga set. Meskipun ditingkatkan kinerjanya di bagian mekanisme penyimpanan rahasia, pesan enkripsi kerugiannya sangat signifikan. Ini dapat mempengaruhi kinerja mekanisme *confidentiality* yang dipakai saat tidak dibutuhkan. Aplikasi *assembler* harus mensupply *deployer* dengan informasi komponen yang harus diproteksi untuk *confidential*. *Deployer* mengkonfigurasi hubungan *container* untuk melakukan mekanisme *confidentiality* saat interaksi dengan bagian yang ada melalui jaringan terbuka dan jaringan yang tidak diproteksi. Tambahan pemakaian mekanisme *confidentiality* disesuaikan, *deployer* harus mengkonfigurasi *container* untuk menolak *request* panggilan atau *respon* dalam isi pesan yang harus diproteksi, keutuhan pesan boleh diperiksa pada sisi yang mempengaruhi penyelenggara *confidentiality*.

Platform J2EE membutuhkan *container* untuk mendukung keutuhan layer transport dan mekanisme *confidentiality* yang berbasis SSL/TLS sehingga sifat keamanan dipakai dalam komunikasi yang tetap untuk mempengaruhi pembentukan hubungan.

4.4.3 Identitas Komponen yang Sensitif

Rekomendasi untuk aplikasi *assembler* dalam identitas komponen dengan metoda panggilan, parameternya menghasilkan nilai yang harus diproteksi untuk keutuhan atau *confidentiality*. Gambaran pengembang ini digunakan untuk menyampaikan informasi. Untuk *enterprise beans* ini dikerjakan dengan subelemen `transport-guarantee` pada subelemen `user-data-constraint`. Kejadian dimana interaksi komponen-komponen dengan *resource* eksternal dikenal membawa informasi yang sensitif, informasi sensitif ini dinyatakan di subelemen `description` yang dihubungkan dengan `resource-ref`.

4.4.4 Jaminan *Confidentiality* pada Web Resource

Konfigurasi jaminan pada web *transport* ini penting dimengerti pemilik metoda HTTP untuk mempengaruhi sebuah hubungan dari satu *web source* ke lainnya. Ketika *resource* terhubung dengan *resource* lainnya, hubungan alami menentukan bagaimana konteks proteksi pada *resource* saat ini mempengaruhi proteksi dalam membuat *request* ke hubungan *resource*.

Ketika hubungan itu pasti (URL dimulai dengan `https://` atau `http://`), *http client container* akan mengabaikan isi *resource* saat ini dan akses itu dihubungkan berdasar *resource* alami pada URL yang tetap. Jika URL pada hubungan dimulai dengan `https://`, transport yang diproteksi akan disesuaikan dengan *server* sebelum *request* dikirim. Jika URL hubungannya dimulai dengan `http://`, *request* akan dicoba melalui transport yang

aman. Ketika hubungan relatif, *http client container* akan memproteksi akses yang terhubung ke *resource*, berdasarkan *resource* tersebut terjadi hubungan dalam proteksi.

Aplikasi *developer* harus menyatakan hubungan yang dimiliki secara hati-hati, ketika *request* dihubungkan harus membawa data *confidentiality* kembali ke *server*. Ada sedikit pilihan yang ada untuk memastikan keamanan dalam suatu kejadian. Sebagai contoh aplikasi *developer* menggunakan hubungan yang tetap aman untuk memastikan proteksi *transport* pada *request* yang membawa data *confidentiality*.

Ketika aplikasi mesin dan *user* hubungannya relatif, pilihan lain *deployer* ke konfigurasi aplikasi sehingga ada interaksi *confidentiality* dari satu *resource* ke lainnya, keduanya dikembangkan dengan sebuah jaminan *transport confidentiality*. Pendekatan ini menjamin bahwa *http client container* tidak mengirim *request* yang tidak diproteksi ke *resource* yang diproteksi.

Point yang dihubungkan dengan metoda `POST` lebih disukai dari pada metode `GET` untuk mengirim *request* data yang *confidentiality*, karena data dikirim melalui `GET` memakai potongan lokasi *browser* dan keduanya *log* disisi *client* dan *server*.

4.5 Auditing

Auditing praktis menggambarkan *record* keamanan yang dihubungkan dengan *event* yang dihendel sejumlah *user* atau sistem untuk akses. Nilai *auditing* tidak hanya menentukan apakah mekanisme keamanan itu membatasi akses ke sistem. Ketika keamanan diputuskan, biasanya lebih penting mengetahui siapa telah izin akses kemudian siapa yang telah merusak akses. Mengetahui siapa yang telah interaksi dengan sistem yang diizinkan untuk menentukan jumlah pelanggaran keamanan. Lebih dari, penggunaan *auditing* yang menilai efektifnya keamanan sistem, ada yang harus jelas mengerti apa yang diaudit dan yang tidak diaudit.

Deployer merespon konfigurasi mekanisme keamanan yang dipakai oleh *container* perusahaan. Setiap mekanisme dikonfigurasi melalui pembatas *container* dengan mencoba melakukan interaksi antar bagian. Sehingga memungkinkan *deployer* atau sistem administrasi untuk mereview pembatas keamanan sesuai *platform* yang dihubungkan dengan kelakuan audit dengan setiap pembatas sehingga *container* itu akan melakukan satu audit berikut ini:

- Semua evaluasi dimana pembatas terpenuhi,
- Semua evaluasi dimana yang tidak terpenuhi,
- Semua evaluasi tidak bergantung keluaran,
- Bukan evaluasi.

Kebijakan audit semua perubahan (hasil dari perkembangan atau urutan administrasi) dalam konfigurasi audit atau pembatas akan dilakukan oleh suatu platform. *Record* audit harus diproteksi sehingga penyerang tidak dapat lepas dari tindakannya menghilangkan *record* tambahan atau merubah isinya.

Model pemrograman J2EE mengganti beban *auditing* dari *developer* dan *integrator* yang merespon pengembang aplikasi dan manajemen. *Container* J2EE yang menyediakan fungsi pengauditan dengan fasilitas evaluasi pada *container* yang digunakan untuk *policy* keamanan.

BAB V KESIMPULAN

Tujuan utama *platform* J2EE mengurangi aplikasi *developer* (pengembang) dari detail mekanisme keamanan dan fasilitas pengembangan yang aman untuk aplikasi di lingkungan yang bermacam-macam. Platform J2EE mengamalkan tujuan tersebut untuk menentukan bagian yang jelas pada saat merespon siapa yang mengembangkan komponen aplikasi, siapa yang mengembangkan komponen *assembler* dalam aplikasi, dan siapa yang mengkonfigurasi aplikasi ini di lingkungan khusus. Izin komponen *provider* dan aplikasi *assembler* yang khusus pada bagian aplikasi membutuhkan keamanan, penjelasan pengembang diberikan disisi luar kode untuk pengembang komunikasi yang dibutuhkan *deployer*. Mereka mungkin juga menggunakan *tool container* khusus, yang diberikan *deployer* dengan cara yang lebih mudah ke mesin pencari dengan batas keamanan yang direkomendasikan oleh pengembang.

Aplikasi bagian *provider* mengidentifikasi semua keamanan bergantung pada yang ditempelkan dalam komponen berikut ini:

- Nama-nama pada semua nama tugas digunakan oleh bagian panggilan ke `isCallerInRole` atau `isUserInRole`.
- Referensi untuk semua *resource* eksternal diakses oleh bagian
- Referensi untuk semua panggilan antar komponen yang dibuat oleh komponen.

Aplikasi komponen *provider* memberikan model metoda perizinan, sejauh informasi itu identitas sensitifnya mengenai hal pribadi dalam informasi yang diubah di panggilan khusus.

Aplikasi *assembler* mengkombinasikan satu atau lebih komponen dalam aplikasi *package* dan kemudian menguraikan pandangan eksternal keamanan yang disediakan oleh komponen individu yang menghasilkan keamanan yang tetap untuk aplikasi keseluruhan. Obyektif aplikasi *assembler* menyediakan informasi sehingga dapat dilakukan tindakan pada *deployer*.

Deployer merespon hasil keamanan dalam aplikasi yang disediakan oleh aplikasi *assembler* dan pengguna dalam aplikasi yang aman di lingkungan operasional khusus. *Deployer* menggunakan *tool* pengembang *platform* khusus untuk memetakan tujuan yang disediakan oleh *assembler* untuk *policy* dan mekanisme yang khusus di lingkungan operasional. Mekanisme keamanan dikonfigurasi oleh *deployer* menggunakan implementasi *container* yang membutuhkan komponen *host* dalam *container*.

Mekanisme keamanan J2EE mengkombinasikan *container hosting*, ditambah deklarasi khusus untuk persyaratan aplikasi keamanan, dengan tersedianya mekanisme aplikasi yang ditempelkan ini memberikan model *powerful* untuk keamanan, *interoperable*, distribusi *component computing*.

DAFTAR PUSTAKA

- 1 Allamaraju Subrahanyam, Avedal Karl, et al “Profesional Java Server Programming J2EE Edition”, 2000, Work Press.
- 2 The Java™ 2 Platform, Enterprise Edition, Specification, v1.3. Copyright 2000, Sun Microsystems, Inc.
<http://java.sun.com/j2ee/>
- 3 Enterprise JavaBeans™ 2.0 Specification. Copyright 2001, Sun Microsystems, Inc.
<http://java.sun.com/products/ejb/docs.html>
- 4 The Java™ Servlet 2.3 Specification. Copyright 2001, Sun Microsystems, Inc.
<http://jcp.org/aboutJava/communityprocess/first/jsr053/>
- 5 Document formal/01-12-30 (CORBA 2.6 - chapter 26 - Secure Interoperability). The Object Management Group. Copyright 1997-2002.
< 01-12-30 doc?formal cgi-bin www.omg.org http:>
- 6 *The J2EE™ Tutorial*, S. Bodoff, D. Green, K. Haase, E. Jendrock, M. Pawlan, B. Stearns. Copyright 2002, Addison-Wesley.
<http://java.sun.com/j2ee/tutorial/>
- 7 INTERNET-DRAFT, The SSL Protocol Version 3.0. A. Freier, P. Karlton, and P. Kocher, IETF Transport Layer Security Working Group, 1996.
<http://www.netscape.com/eng/ssl3/draft302.txt>
- 8 RFC-2617, HTTP Authentication: Basic and Digest Access Authentication. J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. Copyright 1999, The Internet Society.
<http://www.ietf.org/rfc/rfc2617.txt>
- 9 RFC-2818, HTTP Over TLS. E. Rescorla. Copyright 2000, The Internet Society.
<http://www.ietf.org/rfc/rfc2818.txt>
- 10 *Applied Cryptography*. B. Schneier. Copyright 1996, John Wiley & Sons, Inc.