

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	ii	
DAFTAR GAMBAR	iv	
ABSTRAK	v	
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> (Layanan)	8
2.5.3	Teknologi Komunikasi	9
BAB III	MODEL KEAMANAN	
3.1	Arsitektur Keamanan J2EE	10
3.1.1	<i>Code Management</i> melalui JVM dan <i>class file verifier</i> , <i>class loader</i> dan <i>security manager</i>	10
3.1.2	<i>Platform Roles</i>	12
3.1.3	<i>Security Role</i> dan <i>Deployment Descriptor</i>	12
3.1.4	<i>Programmatic Security</i> (Keamanan <i>programmatic</i>)	13
3.2	Kriptografi	13
BAB IV	KEAMANAN APLIKASI BERBASIS J2EE	
4.1	Ancaman Keamanan dan Mekanismenya	14
4.2	Authentikasi	14
4.2.1	Domain Proteksi	15
4.2.2	Mekanisme Authentikasi	17
4.2.1.1	Authentikasi <i>Web Tier</i>	17
4.2.1.1.1	Konfigurasi Authentikasi	18
4.2.1.1.2	Authentikasi <i>Hybrid</i>	19
4.2.1.1.3	Perubahan Identitas Authentikasi	19
4.2.1.2	Authentikasi <i>EJB Tier</i>	19
4.2.1.3	Seleksi Identitas <i>Client</i>	21
4.2.1.4	Informasi Perusahaan menggunakan Sistem Authentikasi <i>Tier</i>	22
4.2.2	Pola Panggilan Authentikasi	23

4.2.3	Pembuka Lingkungan Authentikasi sebagai Referensi	23
4.3	Autorisasi	24
4.3.1	<i>Declarative</i> Autorisasi	24
4.3.2	<i>Programmatic</i> Autorisasi	25
4.3.3	<i>Declarative versus programmatic</i> Autorisasi	26
4.3.4	Isolasi	26
4.3.5	Pengaruh Seleksi Identitas	26
4.3.6	Enkapsulasi untuk Akses Kontrol	26
4.3.6.1	Pembagian Indentitas Pengakses	27
4.3.6.2	Identitas Pengakses Pribadi	27
4.3.7	Akses Pengontrol Untuk <i>Resource</i> J2EE	27
4.3.7.1	Akses Pengontrol ke <i>Web Resource</i>	27
4.3.7.2	Akses Pengontrol ke <i>Enterprise Beans</i>	28
4.3.7.3	<i>Resource</i> Yang Tidak Diproteksi (<i>unprotected</i>)	28
4.4	Pesan yang Diproteksi	29
4.4.1	Mekanisme <i>Integrity</i>	29
4.4.2	Mekanisme <i>Confidentiality</i>	30
4.4.3	Identitas Komponen yang Sensitif.....	30
4.4.4	Jaminan <i>Confidentiality</i> pada <i>Web Resource</i>	30
4.5	<i>Auditing</i>	31
BAB V	KESIMPULAN	32

DAFTAR GAMBAR

Gambar 2.1	Arsitektur <i>2-tier</i>	5
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 2.5	J2EE <i>Server</i> dan <i>Container</i>	9
Gambar 4.1	<i>Domain</i> Proteksi	15
Gambar 4.2	Skenario Autentikasi	16
Gambar 4.3	Konfigurasi Aplikasi Tipe J2EE	20
Gambar 4.4	Arsitektur Protokol CSv2	21

APLIKASI KEAMANAN BERBASIS J2EE

ABSTRAK

Pada lingkungan *computing* perusahaan suatu kegagalan dan kurangnya *availability* pada *resource computing* dapat mengancam kelangsungan hidup perusahaan. Organisasi harus mengambil langkah-langkah tertentu untuk mengidentifikasi adanya ancaman keamanan, setelah ancaman keamanan teridentifikasi diharapkan langkah-langkah diambil untuk mengurangi ancaman keamanan tersebut.

Model pemrograman aplikasi *Java 2 Enterprise Edition* (J2EE) mengisolasi *developer* dari mekanisme-spesifik detail implementasi pada aplikasi keamanan. *Platform* J2EE menyediakan isolasi ini untuk meningkatkan *portability* aplikasinya, sedang penyediannya dideploy pada bermacam-macam lingkungan keamanan.

Produk J2EE dan aplikasi J2EE tidak dimaksudkan untuk mengganti infrastruktur keamanan perusahaan yang telah ada. Perusahaan tetap dapat memperoleh nilai yang signifikan jika melakukan integrasi infrastruktur tersebut. Model pemrograman aplikasi J2EE mencoba menggunakan pengaruh layanan keamanan yang ada dari pada memakai layanan baru atau mekanismenya.

Pembahasan makalah ini dimulai dengan sedikit penjelasan tentang konsep dan teknologi J2EE, model keamanan, dan fokus pembahasana adalah tentang konsep keamanan dan mekanisme aplikasi.

BAB I

PENDAHULUAN

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

1.1 Permasalahan

J2EE menggunakan model aplikasi *multitier* terdistribusi untuk aplikasi *enterprise* (perusahaan). Platform J2EE menetapkan kontrak *declarative* antara *develop* dan komponen aplikasi *assemble* serta konfigurasi aplikasi dalam lingkungan operasional. Pada konteks aplikasi keamanan, penyedia aplikasi memerlukan suatu persyaratan keamanan dalam aplikasinya. Mekanisme *declarative security* digunakan pada sebuah aplikasi yang dinyatakan dengan *syntax declarative* dalam suatu dokumen yang dikenal sebagai *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 ke *security decision* yang dibuat oleh aplikasi *security-aware*. *Programmatic security* berguna pada saat *declarative security*-nya sendiri tidak cukup jelas model aplikasi keamanan. Pada saat aplikasi membentuk otorisasi (*authorization*) berdasarkan waktu maka parameter-parameternya melakukan *call* pada *enterprise bean* atau *web container*, sementara aplikasi lainnya membatasi akses berdasarkan informasi yang disimpan *user* dalam *database*.

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 multi-tier*. Tujuan arsitektur keamanan J2EE adalah untuk mencapai keamanan *end-to-end* pada setiap *tier*. Produk J2EE dan aplikasi J2EE tidak dimaksudkan untuk mengganti infrastruktur keamanan perusahaan yang telah ada. Perusahaan tetap berusaha untuk mendapatkan nilai yang signifikan pada saat melakukan integrasi infrastruktur yang ada. Model pemrograman aplikasi J2EE mencoba menggunakan pengaruh layanan keamanan yang ada, dari pada membangun 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 merupakan 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 menerjemahkan 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* yang dijalankan di *browser*. Model pemrogramannya, pada saat ini telah mampu mendukung aplikasi *enterprise* (perusahaan) dengan baik. Hanya dalam waktu lima tahun Java telah menarik untuk kalangan teknik dan bisnis komunikasi.

Pada awalnya, java mencetuskan model pemrograman dan teknologi baru yang berbeda *domain-ranging* dari peralatan, aplikasi telepon, dan aplikasi perusahaan. Pada waktu yang bersamaan java 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 dengan fungsinya, dan komponen-komponen aplikasi lainnya yang membuat J2EE terinstal dikomputer yang berbeda bergantung pada *tier* di lingkungan J2EE. *Two multitiered J2EE applications* terbagi atas:

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

Meskipun aplikasi J2EE dapat terdiri atas tiga atau empat *tier*, aplikasi *multitier* J2EE secara umum dianggap sebagai *3-tier* karena mereka terdistribusi melalui tiga lokasi yang berbeda, yaitu komputer *client*, komputer *j2ee server* dan komputer *database* atau *legacy* di *backend*.

2.2 Enterprise (perusahaan) Saat ini

Pergeseran bisnis praktis mempercepat level pengembangan aplikasi. Penyimpanan dan alokasi waktu untuk pengembangan aplikasi telah disembunyikan, pada 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.

- *Responsiveness*
Tanpa adanya batas waktu yang selalu dipentingkan, perubahan informasi akan cepat mendukung ekonomi untuk merespon secara langsung dan informasi yang kritis itu telah ditetapkan untuk menghadapi persaingan yang semakin tinggi.
- *Programming Productivity*
Pengadopsian langsung teknologi tidak cukup kecuali jika teknologi itu digunakan dengan tepat berdasarkan suatu hal yang penting dan mengintegrasikannya sesuai dengan teknologi yang relevan. Selanjutnya dapat mengembangkan kemampuan dan

mendeploy aplikasinya secara efektif dan cepat. Pencapaian ini dapat dilengkapi dengan bermacam-macam teknologi dan standar yang telah dikembangkan, sehingga membutuhkan para ahli untuk mendapatkan dan mencari permasalahannya sendiri. Lebih jauh perubahan standar ini merupakan sesuatu yang menarik untuk mendapatkan teknologi yang lebih efisien.

- *Reliability* dan *Availability*

Saat ekonomi internet terjadi pada *downtime*, suatu hal yang fatal untuk suksesnya suatu bisnis dapat terjadi. Diperlukan suatu operasi berbasis web untuk menjalankannya, pada saat pencarian itu dilakukan 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 jumlah *user* secara eksponensial, tetapi juga nilai informasi perusahaan, selanjutnya keamanan informasi menjadi hal yang utama untuk dikembangkan. Teknologi yang berkembang menyebabkan aplikasi perusahaan menjadi 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 membutuhkan 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 menyediakan hal-hal berikut ini.

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

Aplikasi yang dapat dikembangkan dengan program diatas adalah yang mendukung *web page* atau komponen-komponen untuk 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 memenage memori melalui *gerbage colection*, *no pointer* dan lain-lain. Java dirancang untuk "*write once, run anywhere*", dan tujuan akan tercapai menggunakan *bytecode portable*.

- JVM/JRE
JRE (*Java Runtime Environment*) berisi *java virtual machine*, *compiler* dan beberapa *class-class* dasar. *Class-class* java dikompel dalam *platform-independent 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, untuk memanipulasi isinya, dan memproses *output* (keluaran) dengan pernyataan SQL. Banyak *vendor* database yang telah dikembangkan berbasis JDBC API.

2.4 Arsitektur Sistem

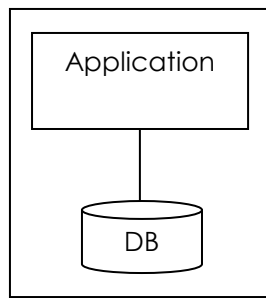
Pengembangan aplikasi perusahaan sesuai dengan konsep arsitektur *n-tier*. Sistem *client/server* didasarkan pada arsitektur *2-tier*, untuk itu perlu penjelasan terpisah diantara data dan presentasi/bisnis. Pada umumnya data mendukung seluruh aplikasi yang ada pada *client mechine* pada saat *server database* dideploy 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 *2-tier* tradisional, proses pengisian diberikan ke PC *client* pada saat *server* bertindak sebagai pengontrol trafik antara aplikasi dan data. Sebagai hasilnya, tidak hanya kinerja aplikasi yang berhak membatasi *resource* pada PC, tetapi jaringan trafik cenderung meningkat dengan baik. Pada saat seluruh aplikasi diproses pada sebuah PC, aplikasi itu terpaksa membuat beberapa permintaan untuk data sebelum memberikan sesuatu ke *user*. Beberapa permintaan *database* ini dapat memberatkan beban jaringan.

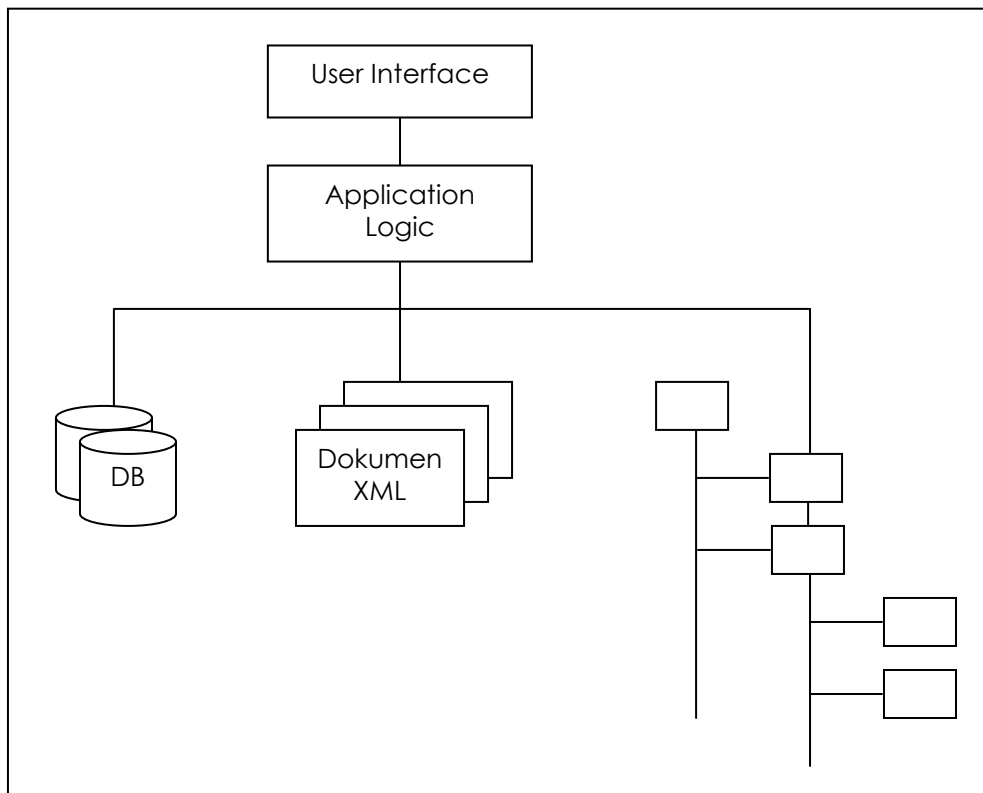
Masalah lain dalam pendekatan *2-tier* adalah pemeliharaan. Perubahan paling kecil untuk sebuah aplikasi itu meliputi sebuah bangunan yang lengkap untuk seluruh *user base*. Jika memungkinkan proses perubahan itu secara otomatis, maka seseorang masih dapat menampilkan hal baru untuk setiap satu instalasi *client*. Beberapa *user* tidak siap untuk membangun dan mengabaikan seluruh perubahan pada group lainnya yang menuntut pembentukan perubahan dengan cepat. Arsitektur *2-tier* dapat dilihat pada gambar 2.1.



Gambar 2.1 Arsitektur 2-Tier

2.4.2 Arsitektur 3-tier

Software community dikembangkan berdasarkan gagasan arsitektur 3-tier. Aplikasi ini dibagi dalam tiga *logical* layer yang terpisah, setiap layer diset dengan baik dalam 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) dengan berdasar kode *user calls* (melewati layer presentasi) untuk mendapatkan kembali data yang diinginkan. Layer presentasi kemudian

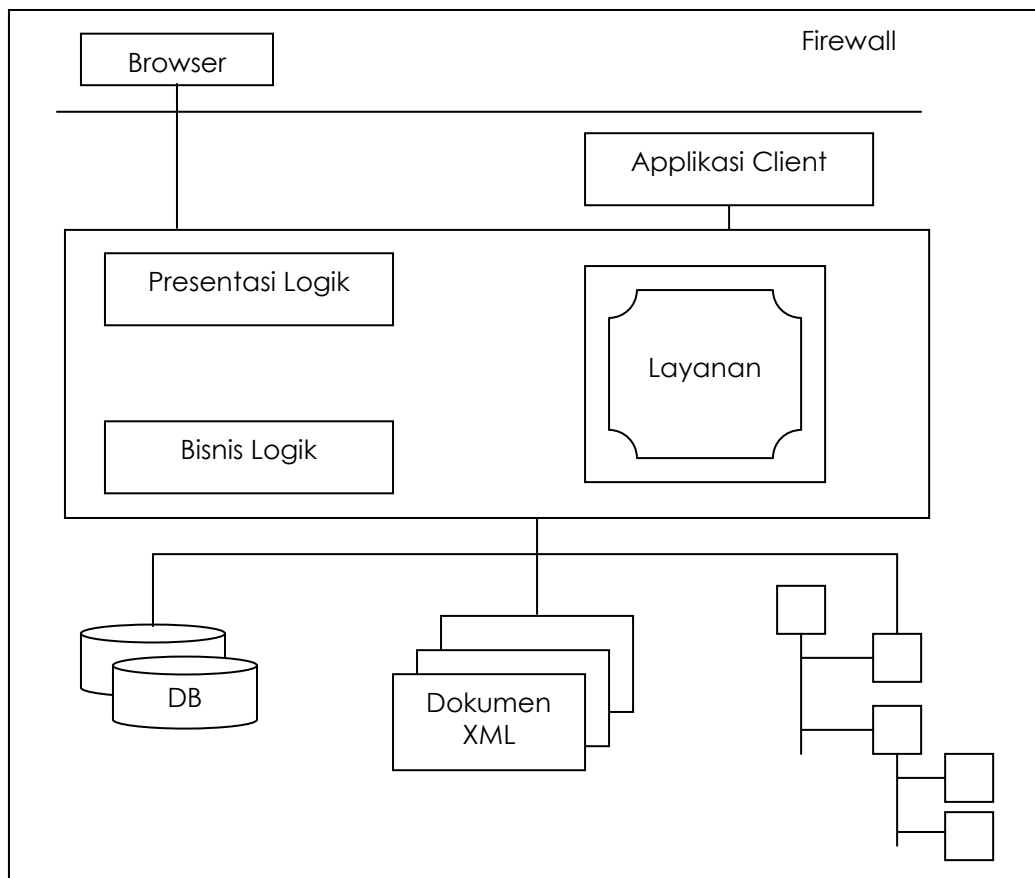
menerima data dan formatnya untuk ditampilkan. Ini memisahkan aplikasi logik dari *user interface* secara fleksibel besar pengaruhnya untuk suatu rancangan aplikasi. Beberapa *user interface* dapat membangun dan mendeploy tanpa merubah aplikasi logik, pemberian aplikasi logik ini akan menjelaskan dalam 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 itu sesuai dengan spesifikasi XML), atau layanan direktori yang lengkap seperti *server* LDAP. Tambahan untuk mekanisme penyimpanan *database relational* tradisional adalah 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 dibagi berdasarkan fungsinya bukan fisik. Berikut ini adalah arsitektur *n-tier*.

- *user interface* yang menghendel interaksi *user* dengan aplikasi, *user interface* dapat menjalankan *web browser* melewati *firewall*, aplikasi desktop yang sibuk atau peralatan *wireless*.



Gambar 2.3 Arsitektur *n-Tier*

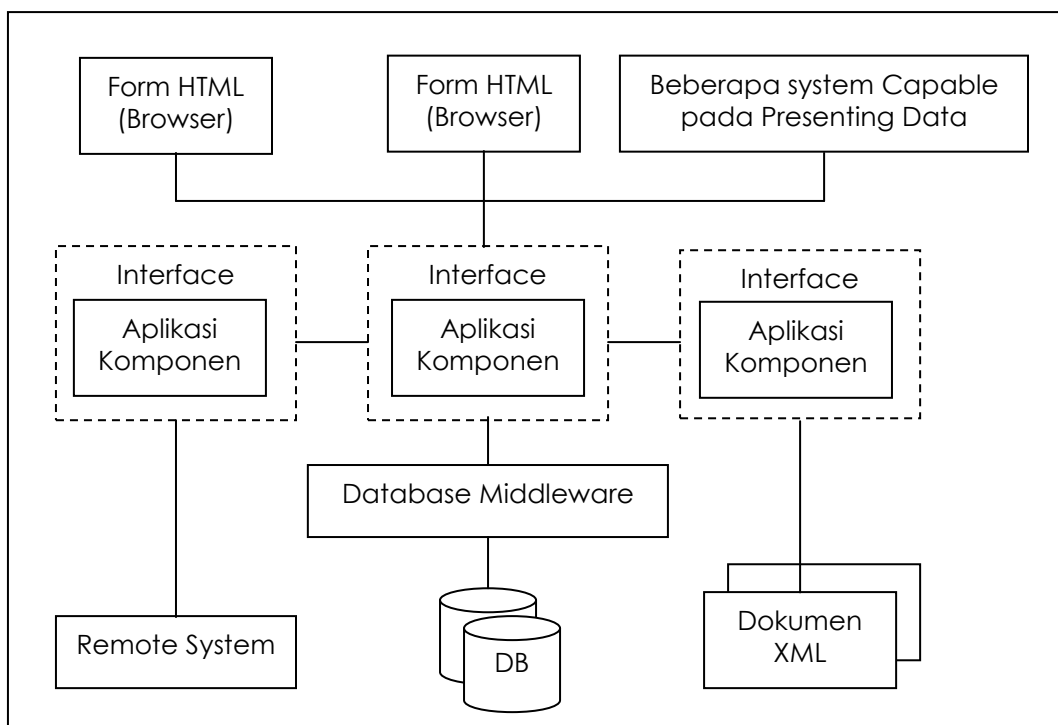
- Presentasi logik menyatakan bahwa tampilan *user interface* dan *request user* itu dihendel, bergantung apakah *user interface* itu mendukung kebutuhan untuk menjelaskan adanya versi berbeda pada presentasi logik untuk menghendel *client* yang cocok.
- Bisnis logik merupakan model aturan aplikasi bisnis yang sering berinteraksi dengan data aplikasi.
- Layanan infrastruktur menyediakan fungsi tambahan yang dibutuhkan oleh komponen aplikasi seperti *messaging*, pendukung transaksi dan lain-lain.
- Layer data dimana sisa data perusahaan.

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

2.4.4 Arsitektur perusahaan

Arsitektur perusahaan berdasarkan *n-tier*, sehingga dibutuhkan adanya perubahan persepsi. Untuk mendapatkan sistem *n-tier* pada sistem perusahaan, dibutuhkan suatu perluasan yang simple pada *tier* tengah dengan cara mengizinkan beberapa obyek aplikasi dari pada satu aplikasi. Obyek aplikasi ini harus memiliki *interface* yang mengizinkan untuk dikerjakan bersama dengan aplikasi lainnya.

Setiap kondisi obyek melewati *interface* yang akan menerima parameter tertentu untuk mendapatkan sekumpulan hasil yang spesifik. Obyek aplikasi berkomunikasi dengan lainnya meenggunakan *interface*. Adapun arsitektur perusahaan ini dapat dilihat pada gambar 2.4.



Gambar 2.4 Arsitektur Perusahaan

Dengan arsitektur perusahaan, seseorang dapat mempunyai beberapa aplikasi menggunakan sekumpulan komponen yang biasa melalui organisasi. Pengembangan standarisasi bisnis praktis pada dasarnya untuk pembuatan satu set fungsi bisnis yang mengakses seluruh organisasi. Jika sebuah aturan bisnis berubah, maka perubahan itu dibuat hanya untuk obyek bisnis.

2.5 Teknologi J2EE

Teknologi dalam arsitektur platform J2EE menyediakan suatu mekanisme untuk membangun suatu distribusi aplikasi perusahaan yang lebih besar. Teknologi pada platform J2EE antara lain.

2.5.1 Teknologi komponen,

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

- **JSP (*Java Server Page*)**

Teknologi JSP mengizinkan seseorang meletakkan kode servlet secara langsung ke dalam dokumen. Sebuah JSP ialah dokumen berbasis teks yang berisi dua tipe teks, yaitu static template data yang dapat ditampilkan dalam format teks, seperti HTML, WML dan XML, dan elemen JSP yang menentukan bagaimana halaman membentuk isi yang dinamis.

- **Servlet**

Servlet berisi definisi *class-class* servlet HTTP tertentu. Sebuah *class* servlet memperluas kemampuan server yang menyimpan aplikasi.

- **EJB (*Enterprise Java Beans*)**

Komponen EJB atau *enterprise beans* ialah kode dengan *field* dan *method* untuk implementasi modul *logic* bisnis. Ada tiga jenis *enterprise beans* yaitu *session beans*, *entity beans* dan *message driven beans*. *Enterprise beans* sering berinteraksi dengan *database*.

2.5.2 Teknologi Service (Layanan),

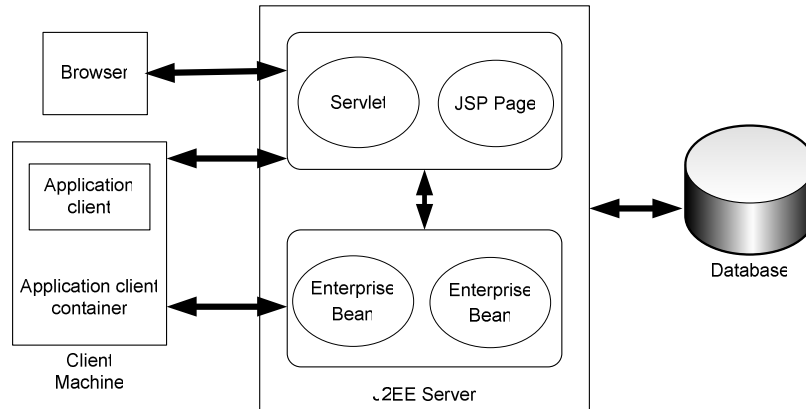
Teknologi ini menyediakan komponen aplikasi untuk mendukung fungsi layanan yang lebih efisien. Beberapa layanan J2EE untuk aplikasi komponen dikelola oleh *container*-nya.

- **Layanan Container**

Container merupakan *interface* di antara sebuah komponen dan platform *low level* spesifik yang mendukung komponen. Sebelum sebuah web, enterprise bean atau komponen *client* dapat dieksekusi, ia harus dirakit ke aplikasi J2EE dan disebarkan ke *container*-nya. Pengaturan *container* akan menghasilkan J2EE *security* model yang memungkinkan seseorang mengkonfigurasi sebuah komponen web atau enterprise bean sehingga *resource* sistem diakses hanya oleh *user* yang punya authorisasi.

- **Tipe Container**

Proses penyebaran (*deployment*) menginstal komponen-komponen aplikasi J2EE ke *container* J2EE.



Gambar 2.5 J2EE Server dan Container

- Server J2EE, bagian *runtime* dari produk J2EE. J2EE server menyediakan *container* web dan *Enterprise JavaBeans* (EJB).
- EJB *Container*, mengatur eksekusi dari enterprise bean untuk aplikasi J2EE. Enterprise bean dan container-nya berjalan di server J2EE.
- Web *Container*, mengatur eksekusi dari komponen JSP dan servlet untuk aplikasi J2EE.
- *Application Client Container*, mengatur eksekusi komponen *client*. Aplikasi *client* dan *container*-nya berjalan di *client*.
- *Applet Container*, mengatur eksekusi dari applet. Terdiri atas sebuah web *browser* dan *plug in java* yang berjalan di *client* secara bersama.

2.5.3 Teknologi komunikasi,

Teknologi komunikasi yang paling jelas adalah aplikasi programmer yang memberikan suatu mekanisme komunikasi antar bagian (*part*) yang berbeda aplikasi, apakah lokal atau *remote*. 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 sering *browser* sesuatu yang potensial dimanapun *client* itu berada. Komunikasi *client* dengan *server* melalui tiga protokol utama yaitu: *Hypertext Transfer Protocol* (HTTP), *Transmission Control Protocol / Internet Protocol* (TCP/IP) dan *Secure Socket Layer* (SSL)

- **Protokol Obyek Remote**

Aplikasi dimana komponen-komponen itu sering didistribusikan melalui beberapa *tier* dan *server*. Mekanisme menggunakan komponen dengan *remote* itu diharapkan, seperti *client* lebih suka tidak melakukan sesuatu dalam komponen lokalnya sendiri.

- **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

Target keamanan pada *Enterprise development* diharapkan sesuai dengan kode *development* dan *deployment*. Dalam *development platform* dan *runtime environment* telah tersedia fasilitas untuk keperluan autentikasi dan otorisasi, data *integrity*, *audit trail*, *non-repudiasi* dan *privacy data*.

Responsibilitas untuk *task-task* ini dikembangkan menurut elemen-elemen *platform*, khususnya arsitektur aplikasi *n-tier*.

3.1 Arsitektur Keamanan J2EE

Arsitektur keamanan J2EE dinyatakan sebagai bagian dalam dokumentasi *platform*. Berikut ini adalah detail dari tugas *security management* dan tujuan dari arsitektur keamanan.

3.1.1 *Code Management* melalui JVM dan *class file verifier*, *class loader* dan *security manager*.

Dasar keamanan Java menggunakan konsep "*sandbox*" untuk membatasi kemampuan dalam *untrusted code* (yang tidak dipercaya) karena membahayakan sistem pada saat dijalankan. Secara historis, jika kode yang datang bersifat *untrusted* tidak akan diberi izin untuk mengakses dari *local disk*, koneksi jaringan terbuka dan lain-lain. Sertifikat autentikasi mendukung proses perizinan dalam *security policy* melalui *java plug-in*, *origin* dan *author*.

Sandbox diimplementasikan melalui JVM dan *class verifier*-nya, selain itu juga melalui *class loader* dan *security manager/ACL manager*.

▪ Keamanan JVM

JVM memberikan *secure runtime environment* untuk pengelolaan memori, pemberian isolasi di antara komponen-komponen pengecekan dengan *namespace* yang berbeda, pengecekan susunan *array* dan sebagainya. Secara dinamis JVM digunakan untuk mengalokasikan *area* memori (*method area*, *GC heap*, *thread stack*), hal itu berarti bahwa tidak mungkin penyerang menentukan apakah *area* memori itu mencoba untuk menyisipkan instruksi yang membahayakan. Pengecekan *array* digunakan untuk mencegah akses memori yang tidak direferensikan.

Class file verifier menguji *class-class* berdasarkan struktur *class loading*. Pada saat *bytecode* dihasilkan oleh *Sun's compiler* secara relatif seharusnya bebas error. Hal ini memungkinkan seorang penyerang secara manual untuk membuat *bytecode* gagal.

Class file verifier menguji setiap *class* yang diload melalui cara: pengecekan dasar, dimana atribut-atribut fisik pada setiap file di cek (ukuran, panjang atribut dan angka unik) ke kelompoknya untuk menjamin bahwa *method* dan *field* yang dipilih memiliki atribut yang benar, pemisahan instruksi dari setiap *method*. Catatan, bahwa *class-class* dasar diset sebagai *class* yang dipercaya sedang *class-class* lain termasuk *class* yang diambil dari aplikasi diset ke *class* yang tidak dipercaya dan harus diverifikasi.

- **Arsitektur Class Loader**

Ada dua tipe *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 JVM untuk meload *class-class* dasar, dan pada umumnya menggunakan *OS-dependent*. *Class loader object* dalam JVM dapat mempercepat *running* dan juga dapat digunakan untuk meload obyek dari *source* seperti jaringan, *local disk* atau penyimpanan data. Pengontrol dibuat dalam *class loader object* karena penting untuk dimanfaatkan pada *class loader*.

Class loader mengatur pengalokasian *class-class* yang dibutuhkan oleh JVM untuk *loading* di dalam *runtime enviroment*. Salah satu tanggung jawab *class loader* adalah mencegah kode *unauthorised* atau *untrusted* menggantikan kode *trusted* dan untuk membuat *class-class* dasar. Sebagai contoh, *class loader* harus mencegah penggantian *security manager class*. Dan mencoba untuk mengganti *class* dasar yang kodenya membahayakan dengan *class spoofing*.

Loading class loader dinyatakan dalam struktur *tree* dimana *primordial class loader* sebagai *root* dan *class-class* lainnya sebagai *leaf node*. Jika *class loader* meload sebuah *class*, semua *subsequence* meminta untuk dihubungkan secara langsung ke *class-class* melalui *class loader*. Sebagai contoh *class loader* 'A' meload *class* 'Building' maka 'Building' membuat *call* (panggilan) ke *methods* dalam *class* yang dipanggil (*called*) 'Cubicle', JVM akan menggunakan *class loader* 'A' untuk meload 'Cubicle' dan *class-class* lainnya dihubungkan ke 'Building'.

Class loader mencegah terjadinya *class spoofing* dengan mengirimkan kembali *request* melalui *parent class loader* nya sampai *class loader* memenuhi *request* dari *class* tersebut. Pada *security manager class*, *primordial class loader* bertanggung jawab terhadap *class* membahayakan sehingga *class* tersebut tidak akan diload. *Class-class* tersebut hanya sekali diload sedangkan *class-class* dasar hanya diload dari *local disk* dengan sistem *classpath*.

Class loader juga mencegah keamanan dengan pengelolaan *namespace*. Sebuah *class* yang diload oleh *class loader* tertentu hanya dapat diakses oleh *class* dengan *namespace* yang sama (*class parents*).

- **Security Manager dan Access Controller**

Security Manager bertanggung jawab untuk memeriksa dan mengimplementasikan *security policy*, didetailkan pada *policy files*. Proses perizinan dalam suatu *class* itu diberikan oleh *security manager* melalui *access controller*.

Metoda perizinan yang terhubung dengan kode group dikelompokkan dalam proteksi domain berdasarkan *source code*-nya. Dengan kata lain group yang meminta izin itu terhubung dengan *class-class* dalam group aslinya.

Proses pemberian izin pada sistem *policy* berdasarkan pada kode-kode proteksi domain yang terlihat dalam *source code*. Aplikasi java secara *default* tidak mempunyai

hubungan dengan *security manager*, maka untuk akses selanjutnya dihubungkan ke sistem *resource*.

3.1.2 Platform Roles

Spesifikasi *platform J2EE* dijelaskan secara organisasi atau *platform roles* dapat digunakan untuk mendelegasikan tanggung jawab pada *J2EE development* dan *deployment cycle*.

Roles itu dinyatakan dalam *platform* khusus yaitu:

- penyedia produk dan penyedia komponen aplikasi, bertanggung jawab terhadap perkembangan *source code*,
- aplikasi *assembler*, bertanggung jawab untuk mendefinisikan metode perizinan dan *roles* keamanan,
- *deployer*, bertugas untuk memeriksa seluruh keamanan dalam suatu aplikasi dan prinsip pengisian *roles*,
- sistem administrator, pada prinsipnya bertugas mengadministrasi dan menjamin bahwa lingkungan keamanan lokal menggunakan *platform J2EE* yang benar.

Roles ini tidak absolute, dan tanggung jawab bermacam-macam *roles* ini dapat berubah sesuai dengan perusahaan pengembangnya dan metodologi *developmet*-nya.

3.1.3 Security Role dan Deployment Descriptor

Deployment descriptor merupakan file XML yang yang dikirim dengan setiap EJB dan penjelasan dari *deployment descriptor* dijelaskan pada setiap aspek fungsi EJB dan makeup. Dan setiap *deployment descriptor* dihubungkan dengan *beans* lainnya.

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

Elemen `<security-role>` digunakan untuk memanggil *roles* yang digambarkan pada elemen `<security-role-ref>`. Contoh *source code*:

```
.
.
.
<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>
```

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

3.1.4 *Programmatic Security (Keamanan programmatic)*

Roles membership dapat ditentukan secara *programmatic* dalam lingkungan J2EE menggunakan metoda `isUserInRole` dan `getUserPrincipal` sedangkan untuk web menggunakan `HttpServletRequest`.

Sebagai bagian dalam kontrak *bean-container*, *container* memberikan obyek `EJBContext`. Sebagai penghubung metoda `EJBContext` adalah `isCallerInRole` dan `getCallerPrincipal`. `getCallerPrincipal` berfungsi untuk mendapatkan sesuatu yang berhubungan dengan konteks keamanan. sedangkan `isCallerInRole` merupakan metoda *Boolean* yang digunakan untuk menentukan apakah *caller* (pemanggil) masih merupakan bagian *spesification security roles*.

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 *package* pilihan untuk J2SDK 1.3 tetapi telah diintegrasikan dalam versi 1.4. JCE menggunakan konsep CSPs (*Cryptographic Service Provider*) untuk *plug in* dalam mengimplementasikan algoritma enkripsi yang berbeda.

Java secure socket Extension (JSSE) merupakan *package* enkripsi untuk J2SDK 1.3 yang akan diintegrasikan ke versi 1.4. *Java developer* untuk kriptografinya menggunakan *secure socket layer (SSL)* dan *TLS*.

BAB IV KEAMANAN APLIKASI BERBASIS J2EE

4.1 Ancaman Keamanan dan Mekanismenya

Aset-aset perusahaan yang sangat rentan terhadap ancaman keamanan antara lain:

- Penyingkapan informasi rahasia.
- Modifikasi atau pengrusakan informasi.
- Penyalahgunaan *resource* yang diproteksi.
- Kompromi *accountability*.
- Penyalahgunaan kompromi yang ada.

Ancaman keamanan pada aplikasi jalannya perusahaan, memiliki bentuk yang berbeda bergantung pada kondisi lingkungan sekitarnya. Sebagai contoh, dalam lingkungan sistem tradisional, satu sistem ancaman keamanan berupa penyingkapan informasi rahasia dilakukan sendiri oleh sistem karena informasi tersebut disimpan dalam file-file, dengan kondisi tersebut maka informasi itu mudah terkena serangan dari luar. Pada lingkungan terdistribusi dengan beberapa server dan client, ancaman penyingkapan informasi terjadi karena terbukanya jaringan pada sistem tersebut.

Mekanisme keamanan yang digunakan pada platform J2EE antara lain: autentikasi, authorisasi, *signing*, enkripsi dan auditing. Berikut ini adalah penjelasan masing-masing mekanisme keamanan.

4.2 Autentikasi

Pada *computing* komponen terdistribusi, autentikasi adalah suatu mekanisme dimana *caller* dan penyedia layanan membuktikan ke satu sama lainnya bahwa mereka bertindak untuk kepentingan *user* atau sistem. Autentikasi itu terjadi dalam dua arah, yang dikenal sebagai autentikasi mutual. Autentikasi menset identitas pengguna dan membuktikan bahwa peserta itu telah sesuai dengan identitasnya. Sebuah entitas dimana peserta dalam sebuah panggilan tidak menyesuaikan dan membuktikan identitasnya (*anonymously*) dikenal *unauthenticated*.

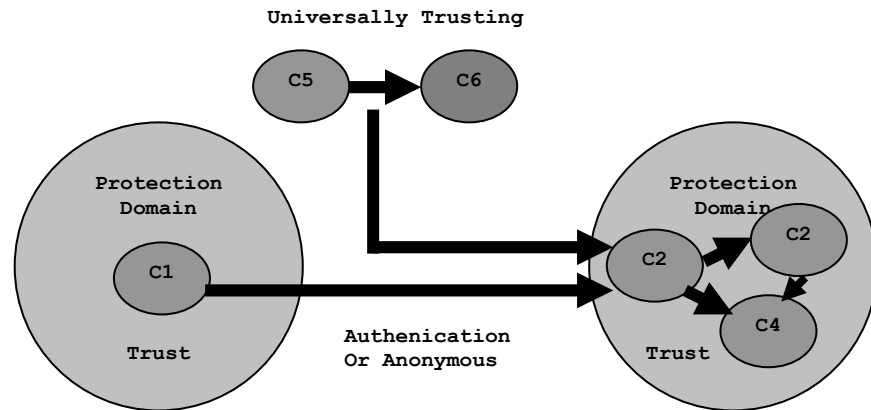
Pada saat program *client* dijalankan oleh *user* yang membuat *call* (panggilan), maka identitas *caller* ada pada user itu. Pada saat pemanggilan, komponen aplikasi itu bertindak sebagai perantara dalam rangkain panggilan dengan beberapa *user*, identitas komponen itu dihubungkan dengan *user* itu sendiri. Pilihan lain, satu komponen aplikasi boleh memanggil lainnya dengan identitasnya sendiri dan tidak dihubungkan dengan pemanggilnya.

Autentikasi biasanya dicapai dalam dua phase. Pertama konteks autentikasi ditentukan dengan kinerja layanan autentikasi yang membutuhkan informasi rahasia. Konteks autentikasi mengenkapsulasi identitasnya sehingga mampu membuat *authenticator*. Kemudian konteks autentikasi digunakan untuk autentikasi dengan entitas lainnya (dipanggil atau memanggil). Autentikasi pada dasarnya memerlukan akses pengontrol ke konteks autentikasi. Berikut ini kemungkinan *policy* dan mekanisme untuk akses pengontrol ke konteks autentikasi.

- Pada saat *user* melakukan inisial autentikasi, maka *user* mulai proses akses *inherit* ke konteks autentikasi,
- Pada saat komponen diautentikasi, akses ke konteks autentikasi dapat digunakan dengan komponen lainnya yang dipercaya, seperti ke bagian aplikasi yang sama,
- Pada saat komponen yang diharapkan menirukan pemanggil, pemanggil menyerahkan konteks autentikasi ke komponen yang dipanggil.

4.2.1 Domain Proteksi

Beberapa entitas dapat berkomunikasi tanpa memerlukan autentikasi. *Domain* proteksi diatur entitasnya untuk menerima atau mempercayai satu dengan yang lain. Entitas pada *domain* ini tidak membutuhkan autentikasi ke lainnya.



Gambar 4.1 Domain Proteksi

Gambar 4.1 mengilustrasikan bahwa autentikasi itu hanya diperlukan untuk interaksi melewati perbatasan pada *domain* proteksi. Ketika satu komponen berinteraksi dengan lainnya pada *domain* proteksi yang sama, maka tidak ada batasan yang ditempatkan pada identitas yang terhubung dengan pemanggilnya. Pemanggil boleh memperbanyak identitas pemanggil atau memilih identitas berdasarkan batasan otorisasi yang ditentukan oleh komponen yang dipanggil. Karena pemanggil dapat mengklaim identitas berdasarkan kepercayaan, maka autentikasi tidak diperlukan. Jika konsep *domain* proteksi dipakai untuk menghindari adanya autentikasi, maka harus ada penyesuaian batasan *domain* proteksi.

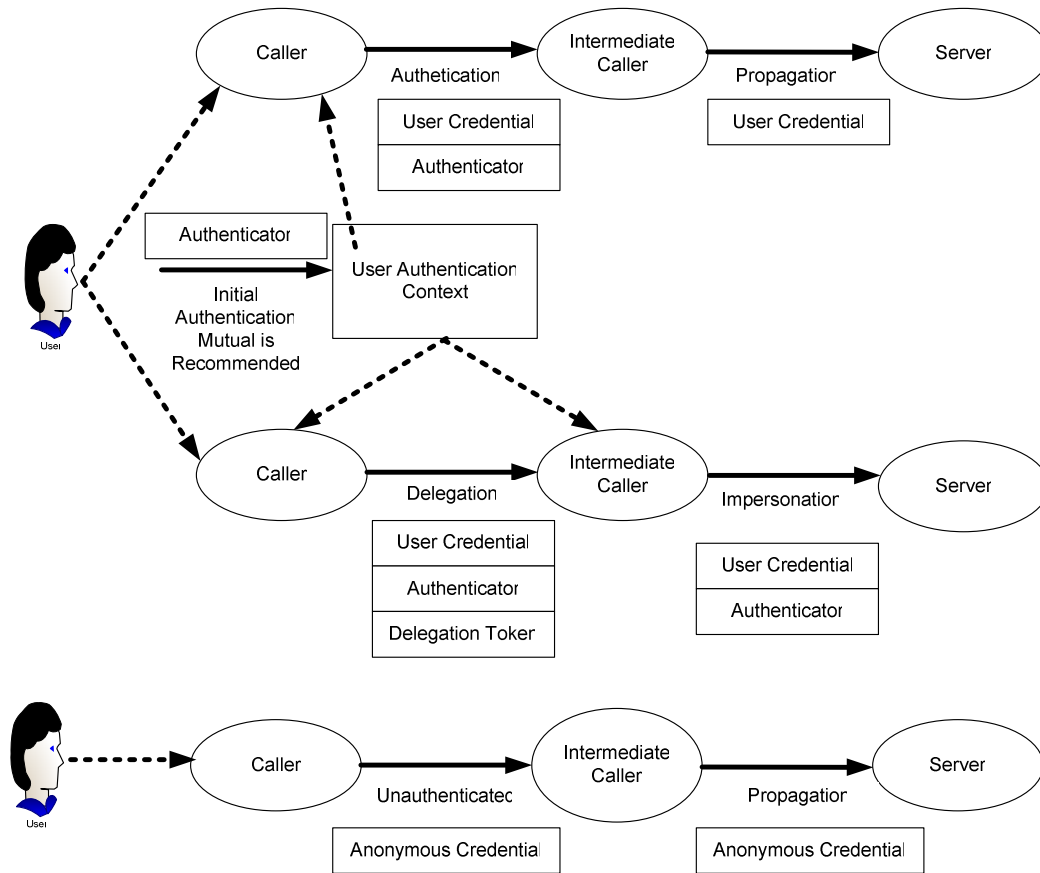
Pada arsitektur J2EE, *container* menyediakan batasan autentikasi antara pemanggil luar dan komponen dalam *host*. Batasan dalam *domain* proteksi tidak selalu disesuaikan dengan *container* itu. *Container-container* mempunyai batasan dan implementasinya mendukung *domain* proteksi dalam rentang *container*. Walaupun *container* tidak membutuhkan komponen *host* dari *domain* proteksi yang berbeda, implementasi dapat memilih untuk dikerjakan.

Untuk panggilan ke dalam, *container* bertanggung jawab untuk memberitahukan bahwa identitas caller itu adalah benar (*authentic*), *authentic* tersebut dalam bentuk *credential*. Contoh *credential* yang digunakan di lingkungan *computing* adalah sertifikat x.509 dan layanan tiket Kerberos. Contoh *surat* untuk interaksi personal adalah *passport* dan surat izin mengemudi (SIM).

Untuk panggilan ke luar, *container* menyesuaikan identitas pada komponen *calling*. Pada umumnya, ini merupakan pekerjaan *container* untuk memberikan autentikasi dua arah pada batasan domain proteksi di lingkungan pengembang aplikasi.

Tanpa bukti identitas komponen, interaksi *container* cukup ditentukan berdasarkan kepercayaan antar-*container* yang diberikan dalam identitas komponen. Pada beberapa lingkungan, kepercayaan dapat diasumsikan secara sederhana, autentikasi antar *container* itu membandingkan identitas *container* yang ada pada daftar identitas yang

dipercaya. Jika bukti identitas tidak diberikan maka cukup kepercayaan antar-*container* saja sampai hubungan tidak ada, sehingga *container* itu dapat menolak atau membebaskan panggilan.



Gambar 4.2 Skenario Autentikasi

Gambar 4.2 mengilustrasikan konsep autentikasi dalam dua skenario yaitu: autentikasi *user* dan skenario unauthenticated *user*.

Autentikasi User

Pada saat user meminta panggilan, yang dikerjakannya adalah mengisi autentikasi user untuk membuktikan identitasnya, selanjutnya identitas pemanggil disebar. Identitas yang disebar hanya akan diterima oleh target yang dimaksud oleh pemanggil, jika mereka masih ada pada domain yang sama.

Identitas yang disebar dari *delegation* kemudian dilanjutkan ke *impersonation* berikutnya. Identitas yang disebar dapat diambil sebagai *authentic* oleh penyedia layanan dalam *propagation*. Sedangkan pada *delegation*, *user* menyediakan komponen yang dipanggil untuk akses ke autentikasi *user* sehingga *user* dapat melanjutkan panggilan.

Unauthenticated User

Identitas unauthenticated *user* dalam bentuk surat tanpa nama (*anonymous*). Surat tanpa nama itu merupakan salah satu bentuk identitas yang tidak perlu dibuktikan dan penyebarannya bergantung pada kepercayaan saja

4.2.2 Mekanisme Autentikasi

Pada kebanyakan aplikasi J2EE, *user* menggunakan *container client* untuk berinteraksi dengan *resource* perusahaan dalam web atau EJB tiers. *Resource* yang tersedia untuk *user* dapat diproteksi atau tidak diproteksi. *Resource* yang diproteksi dibedakan *authorization rules* yang membatasi akses ke beberapa subset identitas *non-anonymous*. Untuk mengakses *resource* yang diproteksi, *user* harus menunjukkan *non-anonymous credential* seperti dengan adanya identitas yang dapat menilai *resource authorization policy*. Adanya hubungan kepercayaan antara *container client* dan *resource*, berdasarkan *credential* dan harus disertai dengan *authenticator* yang menyatakan bahwa *user* sesuai dengan identitas yang diakuinya. Bahasan selanjutnya menggambarkan bermacam-macam mekanisme autentikasi yang didukung oleh platform J2EE dan bagaimana mengkonfigurasikannya.

4.2.2.1 Autentikasi Web Tier

Penyedia komponen aplikasi dapat menunjukkan bahwa *web resource* (komponen web, dokumen HTML, file gambar, ringkasan arsip dan sebagainya) diproteksi oleh sebuah pembatas otorisasi. Ketika *user* tidak diautentikasi secara bertingkat untuk mengakses sebuah *web resource* yang diproteksi, maka *web container* akan memberitahu *user* untuk mengautentikasi dengan *web container*. Permintaan tidak akan diterima *web container* sampai identitas *user* dijamin *web container* dan diperlihatkan ke salah satu identitas sehingga diizinkan untuk akses *resource*. Autentikasi *lazy* merupakan autentikasi pemanggil yang digunakan untuk akses pertama ke *resource* yang diproteksi.

Ketika *user* mencoba untuk mengakses ke *resource web-tier* yang diproteksi, *web container* aktif melakukan mekanisme autentikasi untuk menentukan aplikasi selanjutnya. *Web container* J2EE mendukung tiga mekanisme autentikasi yaitu: HTTP *basic authentication*, *form-based* autentikasi dan HTTPs mutual *authentication*. Sebagai tambahan dianjurkan untuk menggunakan *digest* autentikasi.

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

- *Form-based* autentikasi mengizinkan *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 *mutual authentication*, *client* dan *server* menggunakan sertifikat X.509 untuk menyesuaikan identitasnya. *mutual authentication* terjadi pada *channel* yang diproteksi oleh SSL. Mekanisme *hybrid* yang diutamakan salah satunya adalah HTTP autentikasi dasar, *form-based* autentikasi, atau *digest* autentikasi melalui sebuah SSL yang diproteksi *channel*-nya. SSL digunakan untuk memproteksi *authenticator* selama jaringan berkomunikasi dan untuk autentikasi *server* ke *client*, *authenticator* tidak dapat merubah entitas yang salah. HTTPs *client* autentikasi membutuhkan sertifikat kunci publik dan HTTPs (HTTPs melalui SSL). Autentikasi ini merupakan pendekatan keamanan yang lebih baik dan sertifikat digital dibutuhkan untuk memeriksa siapa saja *user* yang telah diklaim.

4.2.2.1.1 Konfigurasi Autentikasi

Mekanisme autentikasi dikonfigurasi menggunakan elemen `login-config` pada *web component deployment descriptor*. *Source code 1*, *code 2* dan *code 3* mengilustrasikan 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 HTTP dasar dan *form-based* autentikasi, password tidak diproteksi untuk *confidentiality*. Sehingga mudah diserang dan hal tersebut dapat diatasi menggunakan autentikasi yang dijalankan melalui SSL-proteksi untuk memastikan semua isi pesan, termasuk *client authenticator*, proteksi *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>
```

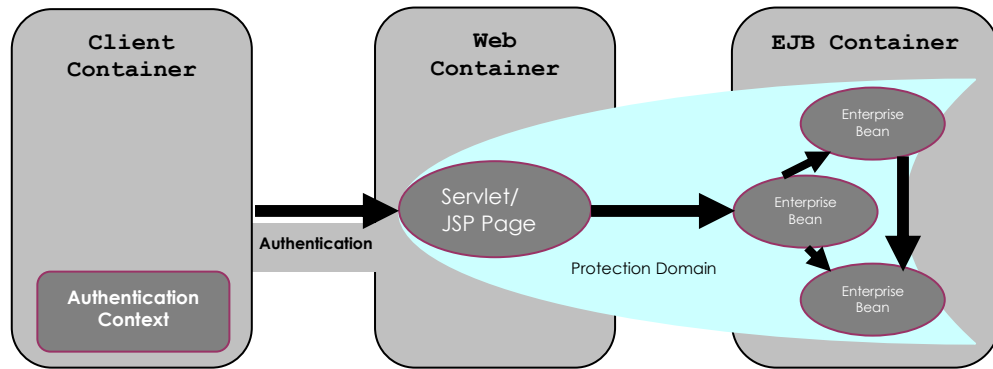
4.2.2.1.3 Perubahan Identitas Autentikasi

Perubahan identitas autentikasi pada *user* terjadi karena *user* mencoba untuk masuk ke web yang diproteksi dan ia ditolak karena tidak punya hak akses ke *resource* tersebut. Meskipun *user* dapat keluar dari *browser* dan memulai proses autentikasi kembali, tetapi kejadian ini tidak dapat selalu diterima. Aplikasi ini memberikan *user* kesempatan untuk membuat autentikasi tidak berlaku dan *user* melakukan autentikasi kembali menggunakan identitas khusus. Elemen `error-page` digunakan untuk pengembangan selanjutnya dalam aplikasi web untuk konfigurasi *resource* yang ada pada web *container*, pada saat proses permintaan HTTP menghasilkan sebuah kode kesalahan HTTP. Hal ini dilakukan dengan cara mengalihkan sebuah permintaan yang tidak diberi hak ke *resource* dan web *container* memberikan *user* kesempatan sampai autentikasi tidak berlaku. Sebagai contoh *resource error-handling* mengembalikan *form* untuk *user* mengisi URI dan parameter yang diminta tidak diberi hak. *Form* menyediakan pilihan kepada *user* untuk tidak memvalidasi autentikasi saat ini. Akibat pemilihan ini akan menyebabkan *form* berisi URI dan parameter yang akan di *submit* ke web *container*, dimana *resource* yang tidak berlaku tersebut diminta. *Resource* ini tidak memvalidasi autentikasi saat dipanggil dengan `HttpSession.invalidate`. Selanjutnya *user* dialihkan melalui URI dan parameter yang ditempatkan ke *resource* asli tidak diautentikasi.

4.2.2.2 Autentikasi EJB Tier

Sebelumnya J2EE 1.3 dan EJB 2.0, *platform J2EE* tidak membutuhkan *container EJB* untuk mengimplementasikan 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 untuk akses ke 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.



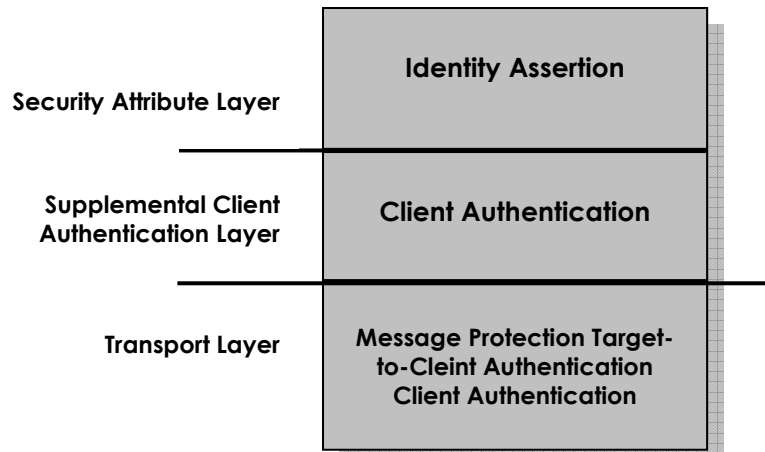
Gambar 4.3 Konfigurasi aplikasi tipe J2EE

Common Secure Interoperability (CSIv2)

Platform J2EE 1.3 membutuhkan *container* EJB dan *container client* EJB untuk mendukung versi 2 pada protokol *Common Secure Interoperability* (CSIv2). CSIv2 merupakan standar *Object Management Group* (OMG) yang mendefinisikan protokol kabel untuk membuat inovasi keamanan RMI-IIOP. CSIv2 dirancang untuk digunakan pada lingkungan proteksi *integrity* atau *confidentiality* pesan dan autentikasi *server* ke *client* dilakukan pada layer transport, mungkin pada SSL atau TLS. CSIv2 mendefinisikan protokol *security attribute service* (SAS) yang digunakan melalui *transform* untuk melakukan autentikasi *client* dan peniruan (*impersonation*) yang fungsinya tidak dapat dicapai dengan menggunakan *underlying transport*. Mekanisme peniruan untuk menonjolkan identitas, dapat dibuat untuk penegasan identitas lain dari dirinya, berdasarkan pada kepercayaan yang diberikan selanjutnya. Penonjolan identitas dapat digunakan untuk *container* J2EE yang selanjutnya 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 yang ada memberitahukan tindakan *client*-nya. Syarat keamanan merupakan tujuan yang disampaikan dalam mekanisme yang ditentukan oleh setiap layer CSIv2. setiap definisi kombinasi ini didukung dan dibutuhkan oleh fungsi keamanan yang harus sesuai dengan kebutuhan *client* yang dituju. Ketika aplikasi J2EE dikembangkan dalam aplikasi *server*, pengembang harus mendefinisikan *policy* keamanan CSIv2 untuk komunikasi *client*. Aspek paling

penting untuk *policy* adalah apakah tujuan itu membutuhkan sebuah *integrity* dan/atau *confidentiality* untuk memproteksi *transport*, apakah tujuan membutuhkan autentikasi *client*, dan mekanisme atau persyaratan mekanisme yang 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 tersebut memanggil komponen J2EE lainnya. Identitas disesuaikan dengan *container* yang bergantung pada seleksi identitas *policy* yang ditentukan oleh pembuat. Pembuat dapat menggabungkan salah satu seleksi identitas *policy* dengan komponen: `use-caller-identity` atau `runas(role-name)`. *Policy* `use-caller-identity` digunakan *container* sebagai identitas komponen pemanggil ke semua panggilan yang dilakukan oleh komponen. *Policy* `runas(role-name)` menyebabkan *container* yang digunakan adalah identitas statis yang diseleksi oleh pembuat berdasarkan prinsip identitas yang dipetakan dengan nama ketentuan keamanan.

Komponen *policy* untuk seleksi identitas ditentukan oleh J2EE web dan *resource* EJB. Aplikasi *developer* menggunakan komponen pemanggil untuk bertanggung jawab terhadap tindakan yang dilakukan komponen untuk kepentingannya sehingga harus menggabungkan *policy* dengan komponen `use-caller-identity`. Penggunaan identitas seleksi *policy* `runas(role-name)` untuk menghentikan rangkaian pemanggilan 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 Menggunakan 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 kondisi ini, *container calling* dapat dikonfigurasi untuk mengatur autentikasi ke *resource* pada komponen *calling*. Bentuk autentikasi ini disebut *container-managed resource manager sign on*. Arsitektur J2EE juga mengenal beberapa komponen yang beberapa diantaranya harus memiliki kemampuan untuk mengelola identitas pemanggil secara khusus dan secara langsung menghasilkan *authenticator* yang cocok. 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 dihindari karena adanya alasan tertentu. Sebagai contoh, aplikasi panggilan yang memproteksi *resource* EJB dari *resource web* yang tidak diproteksi dapat menyebabkan terjadinya masalah. Karena paradigma *web tier autentikasi lazy* hanya memberi izin *user* sesuai autentikasinya pada saat *user* menerima akses ke *resource* yang diproteksi. *User* yang tidak diautentikasi mencoba untuk mengakses 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 karena 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 mengautentikasi *user* karena identitas *user* tersebut tidak dapat diketahui dengan jelas pada penggunaan aplikasi pertamanya. Sebaliknya 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 selesai, *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 Pembukaan Lingkungan Autentikasi sebagai Referensi

Referensi yang diberikan penyedia komponen aplikasi dibuat untuk setiap komponen ke komponen J2EE lainnya dan untuk *resource* luar. *Roles* tambahan dilokasi layanan, seperti memberitahukan *deployer* disemua tempat 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 pengetahuannya 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 *credential* yang dibuat untuk pemanggilan komponen. *Credential* itu berisi informasi yang menggambarkan pemanggil melalui identitas atribut-nya. Pada kejadian pemanggil tanpa nama, digunakan *credential* khusus. Atribut yang unik merupakan identitas pemanggil dengan isi rahasia sesuai dengan yang ditampilkan dalam *credential*. Bergantung pada tipe *credential*, ia dapat juga berisi atribut lain untuk menentukan pembagian otorisasi seperti anggota group dan membedakan kumpulan *credential* yang terhubung. Identitas dan atribut pembagi otorisasi *credential* 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* bertindak sebagai batasan otorisasi antara komponen *host* dan pemanggilnya. Batasan otorisasi ada didalam batasan autentikasi *container* sehingga otorisasi dinyatakan pada konteks autentikasi yang berhasil. Untuk panggilan kedalam, *container* membandingkan atribut keamanan dari *credential* pemanggil dengan akses pengontrol *rules* pada komponen tujuan. Jika memenuhi ketentuan maka panggilan diperbolehkan, dan jika sebaliknya, maka panggilan ditolak.

Ada dua dasar pendekatan untuk menentukan akses pengontrol *rules*: 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 *rules* 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 ketentuan 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* berjalan dilingkungannya. Pemanggil menugaskan logik khusus berdasarkan nilai atribut keamanannya. Sebagai contoh, *deployer* boleh memetakan ketentuan keamanan untuk group keamanan di lingkungan operasional. Sebagai hasilnya setiap pemanggil menggunakan atribut keamanan untuk menunjukkan keanggotaan dalam suatu group yang

ditugasi khusus untuk mewakili aturannya. Contoh lain, *deployer* boleh memetakan ketentuan keamanan dalam daftar yang berisi satu atau lebih identitas dalam lingkungan operasional. Pemanggil selanjutnya diauthentikasi oleh satu identitas yang ditugaskan khusus untuk diwakili oleh aturan tersebut.

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

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

4.3.2 Programmatic Autorisasi

Container J2EE membuat keputusan kontrol akses sebelum mengirim metoda pemanggilan ke komponen. Logik atau *state* komponen bukan merupakan faktor utama untuk menentukan akses. 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.

Penyedia komponen aplikasi 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`. *Deployer* mencocokkan hubungan antara nama khusus yang ditempelkan pada aplikasi dan ketentuan keamanan yang ditentukan pada gambaran pengembangan. Hubungan diantara nama khusus dan ketentuan keamanan boleh berbeda untuk komponen dalam aplikasi yang sama.

Tambahan untuk pengujian khusus, aplikasi komponen dapat membandingkan identitas pemanggilnya menggunakan `EJBContext.getCallerPrincipal` atau `HttpServletRequest.getUserPrincipal`, untuk membedakan identitas pemanggil yang ditempelkan pada kondisi komponen pada saat dibuat. Jika identitas pemanggil sama dengan pemanggil yang dikenal, komponen dapat mengizinkan pemanggil untuk melakukannya. Jika tidak, komponen dapat mencegah pemanggil dari interaksi lebih lanjut. Pemanggil yang dihasilkan oleh *container* bergantung pada mekanisme autentikasi yang digunakan oleh pemanggil. *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, perwakilan *user* yang sama dihubungkan dengan komponen-komponen. Hal ini secara khusus dianjurkan untuk aplikasi yang fleksibel.

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 penyedia komponen aplikasi. *Policy* eksternal lebih fleksibel sesudah aplikasi ditulis, sebagai tambahannya *policy* eksternal itu transparan dan secara lengkap dipahami oleh *deployer*. *Policy* internal menyediakan fungsi yang lebih fleksibel saat aplikasi akan ditulis. *Trade-off* ini harus dinyatakan sebagai pilihan model otorisasi untuk komponen dan metode khusus.

4.3.4 Isolasi

Pada saat merancang akses kontrol ada *rules* 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 *rules* 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 *rules* yang sama untuk semua pola pengaksesan dalam komponen dan bagian aplikasi diperlukan untuk melakukan panduan kecuali jika ada beberapa kebutuhan khusus untuk aplikasi arsitek lainnya.

4.3.5 Pengaruh Seleksi Identitas

Pada saat pengaturan aplikasi akses kontrol *policy*, penyedia komponen aplikasi berdasarkan keputusan *policy* mengasumsikan tentang identitas panggilan diseleksi oleh aplikasi pemanggil. Ketika panggilan dilewatkan melalui komponen selanjutnya, identitas pemanggil terdapat di dalam komponen yang dituju dan bergantung pada identitasnya untuk menyeleksi keputusan yang dibuat selanjutnya. Komponen yang dituju diasumsikan sebagai identitas pemanggil yang telah disebarakan selama rangkaian panggilan sehingga identitas pemanggil merupakan pemanggil yang diinisialisasi dalam rangkaian 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 disebarakan. Aplikasi *assembler* menanggapi komponen komunikasi sebagai identitas yang menyeleksi *policy* sebagai gambaran untuk dikembangkan. Kekurangan yang dimiliki oleh identitas digunakan untuk menyeleksi *policy* dari perakitan, *deployer* harus mengasumsikan bahwa komponen akan memanggil komponen lain menggunakan identitas pemanggil.

4.3.6 Enkapsulasi untuk Akses Kontrol

Model komponen aplikasi bisa jadi merupakan batasan otorisasi yang tidak diproteksi *resource*-nya, penggunaan komponen pengakses untuk implementasi merupakan hambatan otorisasi. Jika komponen pengakses menggunakan cara ini, akses kontrol dapat menggunakan salah satu dari *container* eksternal atau internal komponen, atau dapat juga kedua-duanya.

Komponen pengakses boleh mengenkapsulasi pemetaan konteks 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, ketentuan otorisasi sedikit banyak menjadi lebih kompleks. Sebagai contoh jika semua akses *resource* dilakukan melalui satu konsep kekuasaan sistem informasi perusahaan sebagai tingkatan identitas, kemudian aplikasi J2EE dapat mengimplementasikan akses yang aman ke *resource*, untuk membatasi siapa yang dapat menggunakan akses tersebut. Jika pemetaan konteks autentikasi *many to many*, maka akses konfigurasi otorisasi 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, baik berupa *container-managed sign on* atau *bean-managed sign on*. Perizinan dihubungkan dengan metoda-metoda pada komponen untuk 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. *Session bean* mempercayakan proteksi *entity bean* untuk memetakan prinsip J2EE yang terhubung dengan prinsip *resource* eksternal, dan juga terhubung dengan *authenticator* jika diperlukan. Proteksi *entity bean* memegang semua pemetaan dan *bean* membatasi akses pemetaan khusus untuk prinsip yang khusus (menghasilkan `getCallerPrincipal`).

4.3.7 Akses Pengontrol ke *Resource* J2EE

Client biasanya menggunakan aplikasi J2EE *container* untuk berinteraksi dengan *resource* perusahaan di web atau EJB *tier*. *Resource* ini boleh diproteksi atau tidak diproteksi. *Resource* yang diproteksi mempunyai ketentuan otorisasi untuk menentukan gambaran pengembangan selanjutnya untuk membatasi akses beberapa subset identitas *non-anonymous*. Untuk akses *resource* yang diproteksi, *user* harus menghadirkan *credential* namanya yang mungkin merupakan identitasnya untuk dinilai terhadap *policy* otorisasi *resource*.

4.3.7.1 Akses Pengontrol ke Web *Resource*

Untuk akses kontrol ke web *resource*, penyedia komponen aplikasi atau perakit aplikasi 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 otorisasi web *resource*

```
<security-constraint>
  <web-resource-collection>
```

```

        <web-resource-name>placeorder</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 Akses Pengontrol ke *Enterprise Beans*

Penyedia komponen aplikasi atau perakit aplikasi yang menentukan ketentuan keamanan untuk *enterprise bean* dapat juga menentukan metoda dari *bean's remote*, rumah, lokal, dan interface rumah lokal menggunakan ketentuan keamanan sesuai yang diminta. Ini dikerjakan dengan bentuk elemen `method-permission`. Penugasan *user* digunakan untuk menentukan ketentuan jika sebuah *resource* diproteksi. Ketika ketentuan dibutuhkan untuk 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 yang sama, *style* ini dihubungkan ke semua metoda tersebut. Metoda khusus selanjutnya dapat dikualifikasi menggunakan identitas nama metoda sebagai parameter tandatangan (*signature*) atau dihubungkan ke metoda interface khusus pada *enterprise bean*.

Source code 8 konfigurasi autorisasi *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 fitur isi aplikasi *web-tier* yang tidak diproteksi, tersedia untuk semua pemanggil tanpa autentikasi. *Resource* yang tidak diproteksi memiliki karakteristik persyaratan berupa tidak adanya pemanggil yang diautentikasi. 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, *user* tidak autentikasi untuk mengakses beberapa *resource* EJB tertentu. Pada EJB *tier*, perakitan aplikasi menggunakan elemen `unchecked` pada elemen `method-permission` untuk menunjukkan bahwa metode ini dinyatakan oleh *container* khusus yang berhak, dan 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 *computing* terdistribusi, sejumlah informasi yang signifikan dikirimkan melalui jaringan dalam bentuk pesan. Pesan boleh diterjemahkan dan dimodifikasi sesuai yang diharapkan, perubahan tersebut pengaruhnya pada saat pesan diterima. Pesan boleh ditangkap dan digunakan kembali sekali atau lebih untuk keuntungan pihak lain. Pesan boleh dimonitor oleh *eavesdropper* untuk melakukan penangkapan informasi, dan sebaiknya tidak ada. Serangan tersebut dapat diminimumkan menggunakan mekanisme *confidentiality* dan *integrity*.

4.4.1 Mekanisme *Integrity*

Integrity mechanisms (mekanisme *integrity*) menjamin bahwa komunikasi antar entitas tidak akan merusak bagian lain, khususnya pihak yang dapat mengambil dan memodifikasi komunikasi mereka. Mekanisme *integrity* dapat juga digunakan untuk menjamin bahwa pesan hanya dapat digunakan sekali.

Keutuhan pesan dijamin oleh pemberi pesan dengan memberikan *signature* di dalam pesan. Pesan *signature* dihitung menggunakan algoritma *hash one-way* untuk merubah isi pesan menjadi tipe yang lebih kecil, pesan *digest* tetap-panjang yang selanjutnya diberi tanda (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 terlibat dalam suatu panggilan, salah satunya untuk implementasi mekanisme *integrity* karena panggilan melewati jaringan terbuka atau jaringan yang tidak diproteksi atau karena panggilan akan membuat komponen-komponen yang tidak dipercaya terhadap yang lain.

Satu cara melindungi keutuhan pesan tanpa batasan ruang di lingkungan operasional adalah menangkap aplikasi khusus untuk mengetahui identitas pesan mana yang harus diproteksi dengan utuh.

4.4.2 Mekanisme *Confidentiality*

Confidentiality mekanisme (mekanisme *Confidentiality*) menjamin komunikasi pribadi diantara entitas. *Privacy* dicapai dengan mengenkripsi isi pesan. *Symmetric* (atau *sharing* keamanan) merupakan mekanisme enkripsi yang 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 bertanggung jawab mengkonfigurasi *container* memakai mekanisme *confidentiality* yang menjamin sensitifitas informasi agar tertutup bagi pihak ketiga. Meskipun ditingkatkan kinerjanya di bagian mekanisme penyimpan 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 permintaan panggilan atau merespon isi pesan yang harus diproteksi, keutuhan pesan diperiksa pada sisi penyelenggara *confidentiality*.

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

4.4.3 Identitas Komponen yang Sensitif

Identitas perakit aplikasi komponen direkomendasikan dengan metoda panggilan, dimana parameter atau nilai yang dihasilkan 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`. Pada kejadian dimana interaksi komponen-komponen dengan *resource* eksternal dikenal untuk membawa informasi sensitif, informasi sensitif ini dinyatakan di subelemen `description` yang dihubungkan dengan `resource-ref`.

4.4.4 Jaminan *Confidentiality* pada Web Resource

Konfigurasi jaminan web *transport* ini penting untuk diketahui pemilik metoda HTTP karena berpengaruh terhadap sebuah hubungan dari satu *web source* ke lainnya. Ketika *resource* terhubung dengan *resource* lainnya, hubungan alami ini akan menentukan bagaimana konteks proteksi pada *resource* saat ini mempengaruhi proteksi dalam membuat permintaan ke *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 permintaan dikirim. Jika URL hubungannya dimulai dengan `http://`, permintaan 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 permintaan dihubungkan untuk membawa data *confidentiality* kembali ke *server*. Ada sedikit pilihan yang ada untuk memastikan keamanan dalam keadaan sedemikian. Sebagai contoh aplikasi *developer* menggunakan hubungan yang aman untuk memastikan permintaan proteksi *transport* saat membawa data *confidentiality*.

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

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 adalah praktik pengambilan *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, yang lebih penting adalah mengetahui siapa saja yang telah izin akses dan siapa saja yang telah merusak akses. Jumlah pelanggar keamanan dapat ditentukan dengan cara mengetahui siapa saja yang telah berinteraksi dengan sistem yang diizinkan. *Auditing* digunakan untuk menilai efektif tidaknya sistem keamanan yang ada, selain itu *auditing* harus dapat memisahkan mana sistem yang harus diaudit dan mana sistem yang tidak perlu untuk diaudit.

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

- Evaluasi semua pembatas yang terpenuhi,
- Evaluasi semua pembatas yang tidak terpenuhi,
- Evaluasi semua yang tidak bergantung pada keluaran,
- Tidak ada evaluasi.

Kehati-hatian juga perlu pada pengauditan semua perubahan (hasil dari perkembangan atau urutan administrasi) dalam konfigurasi audit atau pembatas akan dipaksakan 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 manajemen. *Container* J2EE menyediakan fungsi *auditing* dengan fasilitas evaluasi pada *container* yang akan digunakan untuk *policy* keamanan.

BAB V KESIMPULAN

Tujuan utama dari *platform* J2EE adalah untuk membebaskan *developer* aplikasi dari detail mekanisme keamanan sistem dan menyediakan pengembangan aplikasi yang aman pada berbagai lingkungan. Platform J2EE memenuhi tujuan di atas dengan cara mendefinisikan secara jelas pembagian tanggung jawab antara siapa yang mengembangkan komponen aplikasi, siapa yang merangkai komponen menjadi aplikasi, dan siapa yang mengkonfigurasi aplikasi untuk digunakan pada suatu lingkungan tertentu. Dengan mengizinkan pembuatan komponen dan perangkai aplikasi untuk menyediakan bagian-bagian dari aplikasi yang mempersyaratkan aspek keamanan, penentu *deployment* menyediakan sebuah program diluar kode bagian pengembang untuk mengkomunikasikan kebutuhannya ke pengembang. Mereka juga dapat menggunakan alat *container* khusus untuk memberikan pengembang cara yang lebih mudah untuk mengusahakan sistem keamanan yang direkomendasikan oleh *developer*.

Penyedia komponen mengidentifikasi semua kebutuhan keamanan yang termasuk dalam komponen, meliputi:

- Nama dari semua ketentuan yang digunakan oleh komponen yang dipanggil dalam `isCallerInRole` atau `isUserInRole`.
- Referensi kepada semua *resource* eksternal yang diakses oleh komponen.
- Referensi kepada semua pemanggilan *inter-componen* yang dibuat oleh komponen.

Penyedia komponen aplikasi dapat juga menyediakan model perizinan, bersamaan dengan informasi yang mengidentifikasi sensitifitas berdasarkan privasi informasi yang dipertukarkan dalam *cell-cell* tertentu.

Perangkai aplikasi menggabungkan satu atau lebih komponen kedalam sebuah paket aplikasi dan kemudian menguraikan keamanan yang disediakan oleh komponen untuk menghasilkan keamanan yang konsisten untuk aplikasi secara keseluruhan. Tujuan dari perangkai aplikasi adalah untuk menyediakan informasi ini sehingga dapat memberitahukan apa yang harus dilakukan oleh *deployer*.

Deployer bertanggung jawab terhadap segi keamanan dari aplikasi yang disediakan oleh perangkai aplikasi dan menggunakannya untuk mengamankan aplikasi pada lingkungan operasi tertentu. *Deployer* menggunakan *tool deployment platform-specific* untuk memetakan sisten keamanan yang disediakan oleh perangkai pada *policy* dan mekanisme yang secara khusus untuk lingkungan operasi tertentu. Mekanisme keamanan yang dikonfigurasi oleh *deployer* diimplementasikan oleh *container* atas nama komponen yang berada dalam *container*.

Mekanisme keamanan J2EE menggabungkan konsep-konsep *container hosting* ditambah dengan spesifikasi deklarasi dari persyaratan keamanan aplikasi, dengan ketersediaan mekanisme *application-embedded*. Hal ini menyediakan model keamanan, *interoperable*, atau penghitung komponen terdistribusi yang sangat baik.

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.