II. TINJAUAN PUSTAKA

Saat ini, keamanan jaringan komputer masih sering tidak diperhatikan dan dipandang sebelah mata oleh banyak kalangan bisnis. Bahkan, orang yang aktif dalam dunia teknologi informasi itu sendiri terkadang tidak peduli dengan keamanan jaringan. Faktanya keamanan jaringan sangat penting, untuk mendukung berlangsungnya aktifitas kerja seseorang dari satu komputer ke komputer lain.

2.1 RPC (Remote Procedure Call)

Menurut Tommy Marki (Marki 2006:3-4), RPC adalah suatu protokol yang menyediakan suatu mekanisme komunikasi antar proses yang mengijinkan suatu program untuk berjalan pada suatu komputer tanpa terasa adanya eksekusi kode pada sistem yang jauh (*remote system*).

RPC mengasumsi keberadaan dari *low-level* protokol transportasi seperti TCP atau UDP untuk membawa pesan data dalam komunikasi suatu program. Protokol RPC dibangun di atas protokol *eXternal Data Representation* (XDR), yang merupakan standar dari representasi data dalam komunikasi *remote*. Protokol XDR mengubah parameter dan hasil dari tiap *servis* yang

disediakan. Protokol RPC mengijinkan pengguna (*users*) untuk bekerja dengan prosedur *remote* sebagaimana bekerja dengan prosedur lokal.

Prosedur panggilan *remote* (*remote procedure calls*) didefinisikan melalui rutin yang terkandung didalam protokol RPC. Tiap *message* dari panggilan akan disesuaikan dengan *message* balikan. Protokol RPC sendiri sebenarnya adalah suatu protokol untuk ”meneruskan pesan” yang mengimplementasikan protokol non-RPC lain seperti panggilan *remote batching* dan *broadcasting*. Protokol ini juga mendukung adanya prosedur *callback* dan *select subroutin*e pada sisi *server*.

2.1.2 Lokal dan Target

Menurut Tommy Marki (Marki 2006: 4), Lokal adalah komputer atau proses yang mengakses suatu *servis*/layanan atau *resources* dari proses atau komputer pada suatu jaringan. Target adalah komputer yang menyediakan *servis*/layanan dan *resources*, dan yang mengimplementasikan *servis* jaringan. Tiap *servis* pada *network* adalah susunan dari program *remote*, dan tiap program *remote* mengimplementasi prosedur *remote*. Semua prosedur berikut parameternya dan hasilnya didokumentasi secara spesifik pada protokol suatu program.

2.1.3 *Protokol Message RPC*

Protokol Message RPC didefinisikan dengan menggunakan deskripsi data *eXternal Data Representation* (XDR) yang meliputi struktur,

enumerasi dan union. Protokol *Message* ini membutuhkan faktor-faktor pendukung sebagai berikut (Jan Newmarch, 1995: 4 -5) :

1. Spesifikasi yang unik untuk tiap prosedur *call.*
2. Respon *message* yang sesuai untuk tiap *message* yang diminta.
3. Otentifikasi lokal untuk tiap layanan dan sebaliknya.

Protokol *Message* RPC memiliki dua (2) struktur yang berbeda, yaitu *call message* dan *reply message*. Tiap lokal yang akan melakukan RPC pada suatu target di jaringan akan menerima balasan berupa hasil dari eksekusi prosedur tersebut. Dengan menggunakan spesifikasi yang unik untuk tiap *prosedure remote*, maka RPC dapat mencocokkan *messag*e balasan untuk tiap *call message* yang diminta lokal (Marki, 2006).

2.1.4 *Call Message*

Tiap *call message* pada RPC mengandung nilai-nilai *unsigned integer* yang digunakan untuk mengidentifikasi prosedur *remote* yang diminta. Nilai-nilai ini adalah (Marki, 2006):

1. Nomor Program
2. Nomor Versi dari Program
3. Nomor Prosedur

2.1.5 *Reply Message*

Reply message yang dikirimkan oleh target jaringan secara bervariasi tergantung apakah *call messages* yang diminta lokal diterima atau ditolak. *Reply message* mengandung informasi yang digunakan untuk

membedakan kondisi-kondisi yang diminta sesuai dengan *call messages*. Informasi ini antara lain:

1. RPM mengeksekusi *call message* dengan sukses.
2. Implementasi *remote* tidak sesuai dengan protokol yang digunakan. Versi yang lebih rendah atau tinggi akan ditolak.
3. Program *remote* tidak tersedia pada sistem *remote.*
4. Program *remote* tidak mendukung versi yang diminta oleh lokal.
5. Nomor prosedur yang diminta tidak ada.

2.2 Fitur dalam RPC

Menurut Tommy Marki (Marki 2006: 6-7), RPC memiliki fitur-fitur sebagai berikut: *batching calls, broadcasting calls, callback procedures* dan *using the select subroutine.*

2.2.1 *Batching Calls*

Fitur *Batching calls* mengijinkan lokal untuk mengirim *message calls* ke target dalam jumlah besar secara *sequence* (berurutan). *Batching* mengnakan protokol *streaming byte* seperti TCP/IP sebagai mediumnya. Pada saat melakukan *batching*, lokal tidak menunggu target untuk memberikan *reply* terhadap tiap *messages* yang dikirim, begitu pula dengan target yang tidak pernah mengirimkan *messages reply*. Fitur inilah yang banyak digunakan lokal, karena arsitektur RPC didesain agar pada tiap *call message* yang dikirimkan oleh lokal harus ada proses menunggu balasan dari target. Oleh karena itu maka pihak lokal harus dapat mengatasi *error* yang kemungkinan terjadi

karena pihak lokal tidak akan menerima peringatan apabila terjadi *error* pada *messag*e yang dikirim.

2.2.2 *Broadcasting Calls*

Fitur *Broadcasting* mengijinkan lokal untuk mengirimkan paket data ke jaringan dan menunggu balasan dari jaringan. Fitur ini menggunakan protokol yang berbasiskan paket data seperti UDP/IP sebagai mediumnya. *Broadcast* RPC membutuhkan layanan *port mapper* RPC untuk mengimplementasikan fungsinya.

2.2.3 *Callback Procedures*

Fitur *Callback Procedures* mengijinkan target untuk bertindak sebagai lokal dan melakukan RPC *callback* ke proses yang dijalankan oleh lokal.

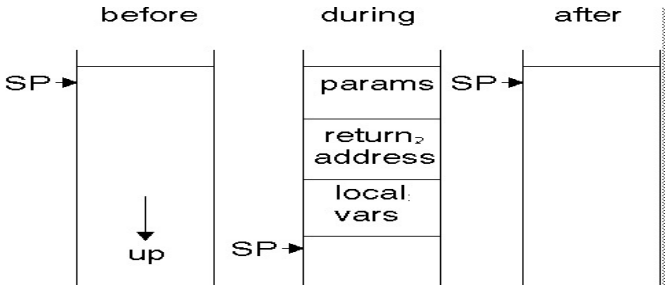
2.2.4 Menggunakan *select Subroutine*

Fitur ini akan memeriksa deskripsi dari suatu *file* dan *messages* dalam antrian untuk melihat apakah mereka siap untuk diterima atau dikirim, atau mereka dalam kondisi ditahan sementara. Prosedur ini mengijinkan target untuk menginterupsi suatu aktivitas, memeriksa datanya, dan kemudian melanjutkan proses aktivitas tersebut.

2.3 Model dan Cara Kerja RPC

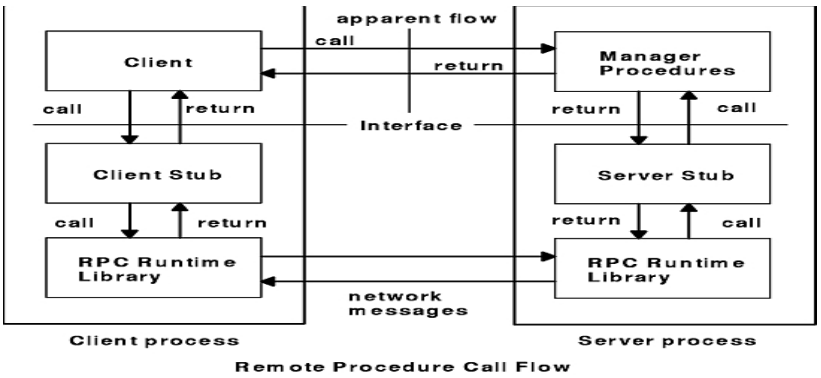
Prosedur *call* umumnya berkaitan dengan penggunaan *stack,* penyimpanan parameter yang diterima dalam *stack* tersebut dan pengalokasian ruang untuk variabel lokal. Namun selain itu ada yang disebut dengan *Prosedur Call*

Remote yang berarti eksekusi proses RPC dilakukan dari lokal dengan target sistem lain. Sistem prosedur *remote* ini memiliki cara kerja yang mirip, namun berbeda dengan prosedur *call* biasa (Marki, 2006).



Gambar 1. Prosedur Call Lokal

Tiap prosedur yang dipanggil dalam RPC, maka proses ini harus berkoneksi dengan *server remote* dengan mengirimkan semua parameter yang dibutuhkan, menunggu balasan dari *server* dan melakukan proses hingga selesai. Proses di atas disebut juga dengan stub pada sisi lokal. Sedangkan, stub pada sisi target adalah proses menunggu tiap *message* yang berisi permintaan mengenai prosedur tertentu. Target harus membaca tiap parameter yang diberikan, kemudian memberikan prosedur lokal yang sesuai dengan permintaan dan parameter. Kemudian setelah eksekusi, target harus mengirimkan hasil kepada pihak pemanggil proses.

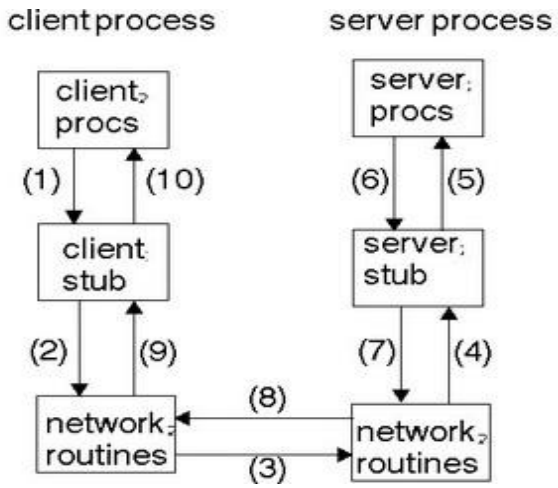


Gambar 2. *Remote Procedure Call Flow*

Diagram di atas memberikan gambaran mengenai *flow* dari eksekusi dalam proses RPC. Berikut ini adalah diagram yang akan menjelaskan secara rinci mengenai proses yang terjadi pada lokal dan target dalam eksekusi suatu prosedur RPC.

Berikut penjelasan dari diagram di atas (Marki, 2006: 17):

1. Lokal memanggil prosedur stub lokal. Prosedur *Stub* akan memberikan parameter dalam suatu paket yang akan dikirim ke jaringan. Proses ini disebut sebagai *marshalling*.
2. Fungsi *Network* pada O/S (Operating system - Sistem Operasi) akan dipanggil oleh *stub* untuk mengirim suatu *message*.
3. Kemudian *kernel* ini akan mengirim message ke sistem *remote*. Kondisi ini dapat berupa *connectionless* atau *connection-oriented.*
4. *Stub* pada sisi target akan melakukan proses *unmarshals* pada paket yang dikirim pada *network*.



Gambar 3. Proses RPC

5. *Stub* pada target kemudian mengeksekusi prosedur panggilan lokal.

6. Jika eksekusi prosedur ini telah selesai, maka eksekusi diberikan kembali ke *stub* pada target.
7. *Stub* target akan melakukan proses *marshals* lagi dan mengirimkan *message* nilai balikan (hasilnya) kembali ke jaringan.
8. *Message* ini akan dikirim kembali ke lokal.
9. *Stub* lokal akan membaca *message* ini dengan menggunakan fungsi pada jaringan.
10. Proses *unmarshalled* kemudian dilakukan pada *message* ini dan nilai balikan akan diambil untuk kemudian diproses pada proses lokal.

Proses diatas akan dilakukan berulang-ulang (rekursif) dalam pengeksekusian RPC dalam suatu *remote* sistem.

2.4 DCOM

DCOM (*Distributed Componen Object Model*) adalah kumpulan konsep *Microsoft* dan program *interface* dimana obyek program dari lokal dapat meminta pelayanan dari target atas program obyek pada komputer di dalam jaringan. DCOM ini berbasis COM, dimana menyediakan sekumpulan *interface* yang memungkinkan lokal dan target untuk melakukan komunikasi pada komputer yang sama. DCOM merupakan protokol yang berfungsi untuk mengaktifkan komponen pada perangkat lunak (*software*) agar dapat berkomunikasi langsung dengan jaringan (Anshary dan Juanita, 2005).

2.5 Metode Waterfall

Model *Waterfall* adalah suatu jenis model pengembangan sistem teknologi informasi yang dikenalkan pada tahun 1970 oleh Winston W. Royce. Sebelum model tersebut ada, sejumlah kegagalan pemakaian dalam implementasi sistem proyek perangkat lunak sering terjadi karena ketiadaan parameter yang sesuai dan pendekatan mengenai cara serta kendali mengenai metode tugas manajemen proyek perangkat lunak.

Tujuan model ini adalah untuk memperkenalkan bagaimana proses desain sistem sebagai kerangka untuk pengembangan sistem dalam upaya membantu secara teratur dan efisien melalui suatu rangkaian tahapan dengan analisa kelayakan sistem termasuk atas release sistem dan pemeliharaannya.

Dinamakan *Waterfall* karena model tersebut menggambarkan arah kemajuan sistem dari puncak ke bawah, seperti air yang terjun dari suatu ketinggian dengan berbagai panoramanya. Model *Waterfall* berfase tunggal pada waktu yang sama ke arah bawah dalam suatu efek *cascading*. Sekarang ini, model *Waterfall* dipertimbangkan sebagai suatu model klasik dan model jenis sistem konservatif. Namun bagaimanapun juga, model ini masih sangat dibutuhkan dan harus tetap ada untuk suatu pemahaman pokok pengembangan sistem dalam upaya merancang manajemen sistem perangkat lunak (Jogiyanto, 1991).

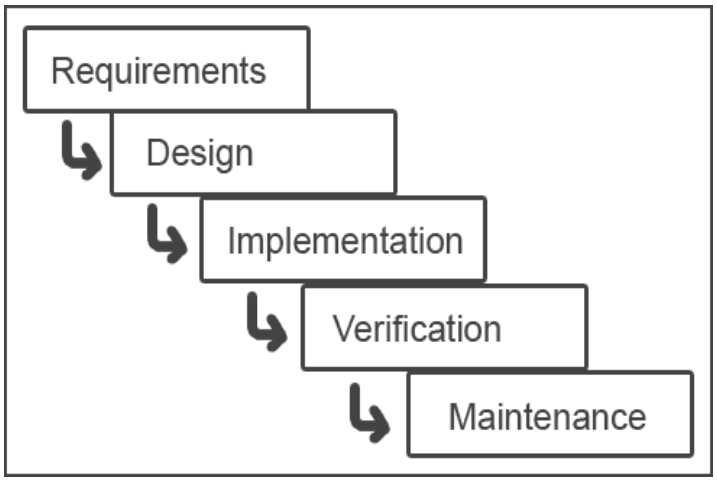
Dalam model *Waterfall*, desain sistem dipecahkan dalam sejumlah langkah linier dan sequential di mana evolusi sistem dilihat bagaikan air yang

mengalir semakin turun melalui fase-fase tertentu. Model *Waterfall* mempunyai tujuan berbeda dari masing-masing fase *(phase)* pengembangan. Metode pengembangan ini tidaklah membolehkan fase tertentu langsung menggantikan fase berikutnya sampai operasi fase yang terdahulu telah terpenuhi. Keluaran *(output)* dari fase masing-masing membentuk masukan *(input)* pada fase berikutnya. Oleh karena itu, masing-masing fase harus terpenuhi dahulu pada gilirannya untuk memelihara pertaliannya antara masukan dan keluaran (Bista, 2006).

2.5.1 Karakteristik Model Waterfall

Karakteristik dari model *Waterfall* ini meliputi beberapa bagian, yaitu (Iyuadi, 2007):

1. Aktivitas mengalir dari satu fase ke fase lainnya secara berurutan.
2. Setiap fase dikerjakan terlebih dahulu sampai selesai, dan kemudian mulai melaksanakan fase berikutnya.



Gambar 4. Tahapan Metode Waterfal

Tahapan penelitian pada model waterfall meliputi :

1. **Software Analisis Requirement**

   Pada tahap ini berupaya mengumpulkan informasi dasar yang diperlukan untuk pembuatan sistem.

2. **Design**

   Proses ini digunakan untuk mengubah kebutuhan-kebutuhan yang telah didokumentasikan pada tahap sebelumnya menjadi representasi data yang akan digunakan dalam pembuatan sistem sehingga menjadi suatu rancangan sistem yang lengkap. Design harus dapat mengimplementasikan kebutuhan yang telah disebutkan pada tahap sebelumnya. Proses ini juga harus didokumentasikan sebagai konfigurasi dari *software.*

3. **Implementasi dan *Coding***

   Setelah proses Design sistem selesai, hasil rancangan berupa desain harus diubah bentuknya menjadi bentuk yang dapat dimengerti oleh komputer. Pada tahap ini rancangan tersebut ditransformasi ke dalam bahasa pemrograman melalui proses *coding*. Tahap ini merupakan implementasi dari tahap rancangan sistem yang secara teknis akan dikerjakan oleh *programmer*.

4. **Verifikasi/Testing**

   Setelah tahap implementasi *Testing* harus dilakukan untuk menguji fungsionalitas program maupun fungsi dari sistem.

5. Maintenance

Setelah semua fungsi-fungsi *software* bebas dari *error*, dan hasilnya benar-benar sesuai dengan kebutuhan yang sudah didefinisikan sebelumnya, proses selanjutnya adalah *instalasi* sistem baru dan dioperasikan. Setelah memasuki tahapan operasional, pemeliharaan sangat diperlukan untuk menjamin reliabilitas sistem. Pemeliharaan suatu *software* sangat diperlukan, termasuk di dalamnya adalah pengembangan, karena *software* yang dibuat tidak selamanya statis. Ketika dijalankan, mungkin saja *software* yang dikembangkan masih ada kesalahan kecil yang tidak ditemukan sebelumnya, atau ada penambahan fitur-fitur yang belum ada pada *software*, sehingga dibutuhkan pengembangan lebih lanjut.

2.6 GNU C Compiler

GNU C Compiler yang lebih dikenal dengan GCC, merupakan salah satu *compiler* yang terkemuka di Linux. GNU C Compiler/GCC adalah salah satu produk andalan *Free Sofware Foundation* yang merupakan bagian dari proyek GNU (indrajit,2002).

Meskipun GCC bersifat *free* namun dapat diandalkan sebagai salah satu *compiler* yang mendukung standar C dan C++, sehingga mampu bersaing dengan produk–produk komersial lainya. Proses yang dilakukan GCC ada empat bagian sebagai berikut (Indrajit, 2002:3):

1. *Preprocessor*
2. *Compilation*
3. *Assembly*
4. *Linking*

Dari keempat proses di atas semuanya dapat dijalankan satu persatu sehingga pengguna lebih memahami setiap tahapan proses kompilasi yang dilakukan. Ini merupakan kelebihan menggunakan *compiler* dengan GCC dengan *compiler* komersial lainya yang langsung memberikan proses jadi tanpa memperkenalkan setiap tahapan prosesnya (Indrajit, 2002:3).

2.7 Pemrograman Bahasa C

Bahasa C adalah salah satu bahasa pemrograman yang populer di dunia dan memiliki kemampuan lebih dari bahasa pemrograman yang lain. Banyak sekali aplikasi–aplikasi yang ditulis dengan bahasa C, atau paling tidak inti utama programnya ditulis dalam bahasa C. Bahkan, *Sofware Development Kit* untuk Windows ditulis dalam bahasa C (Nanggroe, 2009).

Bahasa C merupakan bahasa pemrograman yang sifatnya *portable*, yaitu dengan sedikit atau tanpa perubahan suatu program yang ditulis dengan bahasa C pada suatu komputer dapat dijalankan pada komputer lain. Bahasa C merupakan *general-purpose language*, yaitu bahasa pemrograman yang dapat digunakan untuk tujuan apa saja.

Bahasa C diciptakan oleh Dennis Ritchie dan Brian Kernighan merupakan bahasa pemrograman yang sangat stabil dan handal. Sebenarnya bahasa C

merupakan pengembangan dari bahasa BCPL yang telah lebih dahulu ada. Sebagai bahasa yang digolongkan dalam *middle level language*, bahasa C mempunyai kemudahan dalam mengakses perangkat keras, juga kecepatan prosesnya yang mendekati *low level* language seperti bahasa Assembly, tetapi memberi kemudahan yang tidak ditawarkan bahasa Assembly. Selain itu, bahasa C jauh lebih mudah untuk dipelajari dibandingkan dengan bahasa *low level*, karena mendekati frase-frase dalam bahasa manusia, yaitu Bahasa Inggris (Nanggroe, 2009).

Bahasa C merupakan salah satu bahasa pemrograman yang dapat digunakan untuk membuat suatu aplikasi, dan yang akan digunakan pada pembangunan sistem ini.

Beberapa keuntungan dari bahasa C ini adalah:

1. Dapat bekerja pada lingkungan yang minim *resource.*
2. Dukungan pustaka atau *library* yang banyak, salah satunya adalah *ncurses* yang akan digunakan pada pengembangan sistem ini.
3. Kode Bahasa C sifatnya *Portable* (Hartono, 1993).

2.8 Distribusi GNU Linux

Distribusi GNU Linux atau yang lebih dikenal dengan istilah *distro* adalah paket sistem operasi GNU Linux yang terdiri dari kernel linux, program instalasi, beberapa aplikasi *open source* lain dan dokumentasi. Biasanya

distribusi GNU Linux dikemas dalam bentuk CD-ROM atau DVD. Selain itu distribusi GNU Linux juga dapat di*download* langsung dari *website* pengembangan *distro* tersebut.

Dilihat dari jenisnya distribusi Linux dapat dibedakan menjadi beberapa macam, yaitu *LiveCD, LiveUSB, CD/DVD instalansi, dan Floppy disk*. Pada umumnya distribusi Linux dikembangkan untuk tujuan yang khusus misalnya untuk *desktop*, *server* internet/intranet, *router*, dan *proxy*. Selain produk linux yang sudah ada para *develover* juga membuat *distro* untuk digunakan sendiri (Info Linux, 2007).

Berikut ini adalah contoh distribusi non komersial atau *free software* diantaranya:

1) *Ubuntu*
2) *Slackware*
3) *Debian Fedora Core*
4) *Suse*
5) *Madriva Linux*
6) *Gentoo*