# ANALISIS KEAMANAN SISTEM PERANGKAT LUNAK

**Ervin Kusuma Dewi[1], Azhari SN[2]**
*[1]Pascasarjana Ilmu Komputer, Universitas Gadjah Mada*
*Program Studi Monodisiplin S2/S3 Ilmu Komputer UGM, Gedung SIC Lt.3 FMIPA UGM Sekip Utara Bulak Sumur-Yogyakarta 55281*
*Telp/Fax : (0274) 5551333*
*[2]Ilmu Komputer, Universitas Gadjah*
*Program Studi Monodisiplin S2/S3 Ilmu Komputer UGM, Gedung SIC Lt.3 FMIPA UGM Sekip Utara Bulak Sumur-Yogyakarta 55281*
*Telp/Fax : (0274) 5551333*
*E-mail: dew1_07@yahoo.com, arisn.softcomp@gmail.com*

**ABSTRAK**
*Kesalahan kerentanan dan keamanan perangkat lunak dapat dikurangi jika secure software development process (SSDP) diikuti, seperti memenuhi aspek keamanan pada tahap membangun perangkat lunak. Keamanan perangkat lunak sangat penting dalam perangkat lunak, sebagian besar keamanan perangkat lunak tidak ditangani sejak software development life cycle (SDLC). Sistem perangkat lunak yang aman akan memberikan tingkat kepercayaan yang tinggi dari user, user akan merasa nyaman dan aman ketika berhubungan dengan sistem yang sudah kita bangun. Pada tulisan kali ini, kita akan membahas tentang keamanan pada fase-fase membangun perangkat lunak. Selain itu tulisan ini dapat membantu pengembang perangkat lunak untuk meningkatkan keamanan dan mengurangi vulnerability (kerentanan) pada perangkat lunak, sehingga kedepan keamanan harus terintegrasi ke dalam software development life cycle (SDLC). Keamanan harus dibangun "built-in" ke dalam produk yang yang kita kembangkan.*
.
*Kata kunci: keamanan, sdlc, vulnerability*

## 1. PENDAHULUAN

Dalam membangung rancangan perangkat lunak yang menjadi salah satu syarat penting adalah hasil perancangan yang juga memperhatikan keamanan perangkat lunak yang di rancang, sehingga tidak akan membuat *user* (pengguna) menjadi cemas. Menurut Curtis Coleman, MSIA , CISSP, CISM direktur *Global IT Governance* dari perusahaan *Seagate Technology* mengatakan bahwa kerawanan yang paling berbahaya pada perusahaan besar yang memiliki jaringan luas adalah pada aplikasinya. Secara khusus perangkat lunak yang dirancang dan dikembangkan tanpa memikirkan keamanan dengan matang oleh pengembang (Howard & LeBlanc, 2002; Ramachandran, 2002; McGraw, 2002).

*Vulnerability* (Kerentanan) dihasilkan oleh cacat *(defect)* yang tanpa sengaja ada di dalam perangkat lunak selama didesain dan pengembangan. Oleh karena itu, untuk mengurangi *vulnerability,* kecacatan harus dikurangi karena perangkat lunak telah menjadi penggerak utama dalam beberapa sektor bisnis, pendidikan dan lainya. Begitu banyak sektor yang bergantung pada perangkat lunak sehingga jika ditemukan cacat pada suatu perangkat lunak yang menyebabkan *vulnerability* ketika dirilis, akan rentan terhadap serangan dan dianggap tidak aman, maka di butukan jaminan kemanan terhadap perangkat yang betul-betul aman.

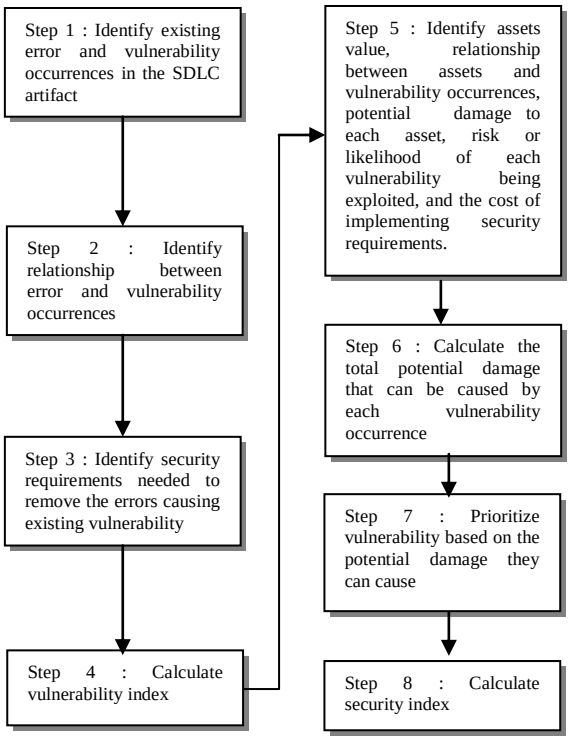
Kemanan perangkat lunak merupakan ide perakayasaan perangkat lunak sehingga perangkat lunak tersebut tetap berfungsi dengan benar di bawah serangan jahat yang merusak. Sistem keamanan pada perangkat lunak merupakan tren masa depan yang dapat dikatakan sebagai era baru setelah era anti virus yang membantu keamanan dari luar perangkat lunak. Sehingga tujuan pada jurnal kali ini akan di fokuskan pada penelitian keamanan suatu perangkat lunak sehingga dapat menjadi referensi untuk meningkatkan keamanan perangkat lunak.

Akibat adanya cacat / kerusakan pada sistem keamanan perangkat lunak akan menimbulkan :

a. Merusak nama perusahaan.
b. Perusahaan akan mengalami kerugian cukup banyak.
c. Kerugian *cost* (waktu dan biaya) untuk perbaikan.
d. Berkurangnya kepercayaan konsumen terhadap perusahaan.

Dalam tulisan *"Quantifying Security in Secure Software Development Phases"* (Khan & Zulkernine, 2009), mengusulkan metodologi yang memungkinkan para pengembang untuk menilai status keamanan SLDC .

Dalam metodologi yang diusulkan, pertama-tama mengidentifikasi dan mengukur aspek keamanan yang terkait seperti : *error* (kesalahan), jumlah kerentanan, resiko, hubungan antara kesalahan dan kerentanan *(vulnerability)*, persyaratan keamanan yang dibutuhkan dan lain-lain.Selain itu, mengitung indeks kemanan berdasarkan faktor resiko dan kemungkinan kerusakan.




Gambar 1. Metode Keamanan

Gambaran metodologi yang diusulkan oleh (Khan & Zulkernine, 2009) dalam gambar 1 adalah :

- Pertama, mengidentifikasi semua kelemahan dan kesalahan yang menyebabkan kerentanan dalam SLDC.
- Kedua, mengidentifikasi hubungan antara *error* (kesalahan) dan ketentanan.
- Ketiga, mengidentifikasi persyaratan keamanan yang diperlukan untuk menghapus kesalahan.
- Keempat, menghitung indeks kerentanan berdasarkan jumlah kerentanan yang telah di hapus dan dan total kerentanan yang ditemukan di awal
- Kelima, Untuk keputusan yang lebih baik kerentanan yang lebih tinggi menjadi prioritas untuk di hapus karena memperhitungkan potensi kerusakan yang dapat disebabkan .
- Keenam, Menghitung potensi total kerusakan yang mungkin disebabkan oleh setiap kemunculan kerentanan dalam kasus terburuk.
- Ketujuh, memprioritaskan kerentanan berdasarkan potensi yang dapat menyebabkan kerusakan .
- Kedelapan, menghitung indeks kemanan SLDC dengan menggunakan total kerusakan yang dapat menyebabkan penghapusan setelah dan sesudah kerentanan.

## 2. KERENTANAN PADA APLIKASI

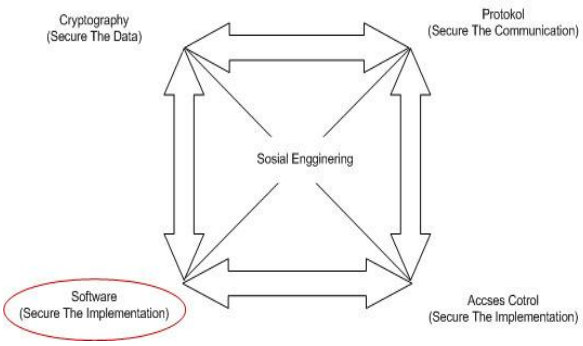
Kebanyakan *Vulenerability* dihasilkan oleh cacat *(defect)* pada perangkat lunak yang tanpa disengaja sehingga menyebabkan kerawanan. Di bawah ini merupakan beberapa serangan yang diakibatkan oleh kerentanan pada aplikasi (Lebanidze, 2005).

- *Unvalidated Input*
- *Broken Access Control*
- *Authentication and Session Management*
- *Cross Site Scripting* (XSS)
- *Buffer Overflow*
- *Injection Flow*
- *Improper Error Handling*
- *Insecure Storage*
- *Denial of Service*
- *Insecure Configuration Managemen*

### 2.1 Keamanan Sistem Perangkat Lunak

Institusi/organisasi yang sudah menerapkan praktek keamanan sistem perangkat lunak bisa memperoleh manfaat tambahan berupa berkurangnya *cost* (waktu dan biaya). Seiring dengan pengurangan lubang pada perangkat lunak, keamanan sistem perangkat lunak harus terintegrasi ke dalam siklus hidup pengembangan software (SLDC) secara penuh. Kemanan harus *"built in "* dalam produk yang sedang dikembangkan, dan tidak hanya diberikan keamanan di luar perangkat lunak seperti menginstal antivirus.

Menurut (Istiyanto, 2011), *Goal security* manajemen terdapat pada 5 pondasi yaitu ; Protokol, Cryptografi, *Access Control*, *Software*, Serta Sosial *Engineering.*




Gambar 2. *Goal Security Manajemen*

Pada gambar 2, terdapat beberapa manajemen keamanan salah satunya keamanan software. Keamanan software dilakukan melalui pengukuran pada ukuran-ukuran keamanan software yang bisa

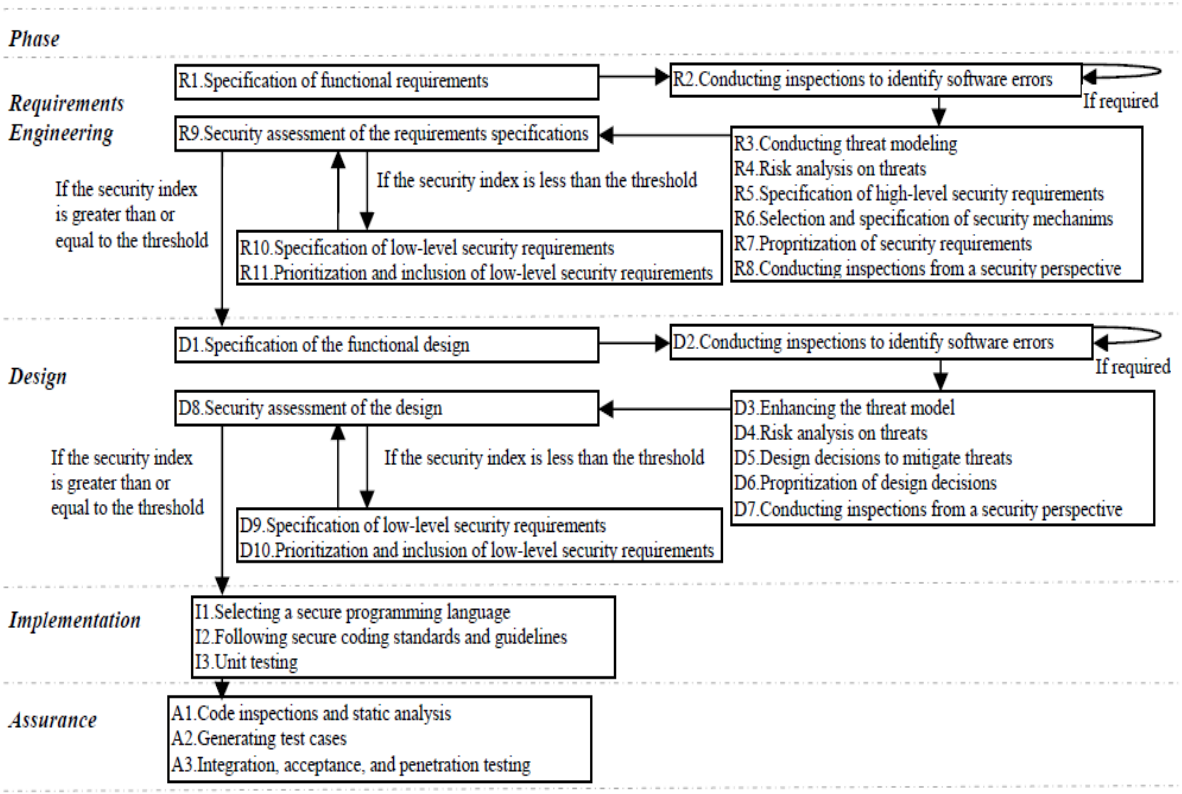
menimbulkan *vulnerability* pada setiap tahapan SDLC *(Software Development Life Circle)* sehingga koreksi awal bisa dilakukan. Titik pengukuran ini meliputi analisa arsitektural, pemodelan ancaman, verifikasi desain, review desain, *review* kode, analisa kode, test unit dan *test system*. Dengan praktek-praktek di atas diharapkan bisa menjamin kwalitas perangkat lunak yang dirilis akan aman dari serangan.

Terkadang kita sulit membedakan sistem keamanan pada level non aplikasi dan aplikasi. Tidak ada definisi yang jelas mengenai pola keamanan karena banyak penulis mendifinisikan dan merujuk berbagai sumber dari konteks keamanan. Ramachandran (2002) pola keamanan sebagai elemen dasar. Romanosky (2002) menyelidiki pola keamanan menggunakan format tertentu untuk memeriksa pola desain perangkat lunak (Gamma dkk, 1995). Serta beberapa penulis mendifinisikan pola keamanan untuk Aplikasi Web (Weiss, 2003; Kienzle & Penatua, 2002), keamanan pola untuk sistem agen (Mouratidis dkk, 2003), Keamanan Kroptografi perangkat lunak (Braga dkk, 1998), untuk java code (Mahmoud, 2000), serta metadata, otentikasi dan otorisasi (Fernandez, 2000; Lee Brown dkk, 1999).

## 3. TAHAP KEAMANAN PERANGKAT LUNAK

Keamanan perangkat lunak merupakan hal yang nyata kesalahan dalam banyak pengembangan sikulus perangkat lunak (SDLC) seperti yang sudah di jelaskan sebelumnya yaitu persyaratan spesifikasi, desain, *source code* bagian dari perangkat lunak yang menyebabkan vulnerability (khan & zulernine, 2009). Keamanan perangkat lunak menjadikan salah satu ancaman keamanan untuk perangkat lunak, seperti :

1. Kesalahan persyaratan spesifikasi *(Requirement specification)* : salah atau terdapat kekurangan persayaratan dalam *requirement specification* pada saat tahap perancangan perangkat lunak .
2. Kesalahan Desain : Kesalahan pada *logical decision*, yang merupakan representasi dari keputusan desain sehingga menyebabkan kesalahan pada tahap membuat desain.
3. Kesalahan *Source Code* : Kesalahan yang di sebabkan karena desain akan bedampak pada saat coding sehingga dapat menyebabkan kesalahan pada saat tahap implementasi.




Gambar 3. *Aktifitas SSDP(Secure Software Development Prosess )*

Dibawah ini merupakan penjelasan dari aktifitas pada SSDP.

Tabel 1. Penjelasan Aktifitas SSDP

| Activities | Input Artifact | Output Artifact |
|---|---|---|
| R1 | None | Functional requirements. |
| R2 | Functional requirements. | Functional requirements. |
| R3 | Information about assets and attackers. | Threat model. |
| R4 | Threat model. | Threat model. |
| R5 | Threat model. | High-level security Requirements |
| R6 | High-level security requirements. | Security mechanisms to be included. Functional requirements with specifications of security mechanisms. |
| R7 | High-level security requirements. | High-level security requirements. |
| R8 | Functional requirements with specification of security mechanisms. Checklist of existing security errors. | Security errors. |
| R9 | Functional requirements with specification of security mechanisms. | Security index. |
| R10 | Security errors. | Low-level security requirements. |
| R11 | Functional requirements with specification of security mechanisms. Low-level security requirements. | Final requirements. |
| D1 | Final requirements. | Functional design. |
| D2 | Functional design. | Functional design. |
| D3 | Functional design. Threat model. | Enhanced threat model. |
| D4 | Enhanced threat model. | Enhanced threat model. |
| D5 | Enhanced threat model. | Secure design decisions. |
| D6 | Secure design decisions. | Secure design decisions. |
| D7 | Functional design with secure design decisions. | Security errors. |
| D8 | Functional design with secure design decisions. | Security index. |
| D9 | Security errors. | Low-level security requirements. |
| D10 | Functional design with secure design decisions. Low-level security requirements. | Final design |
| I1 | None | None |
| I2 | None | Code |
| I3 | Code | Code |
| A1 | Code. Checklists for inspections and static code analysis. | Code |
| A2 | Final requirements. Final design. Lowlevel security requirements. | Test cases. |
| A3 | Code. Test cases. | Test cases. |

Tabel di atas merupakan *Secure Software Development Mode (SSDPM)* pada saat perancangan perangkat lunak yang merupakan ide strategi dari secure software Development proses (SSDP) untuk mempercepat dari salah satu tahap pengembangan selanjutnya, SSDPM banyak mengusulkan alternative yang strategis (khan & zulernine, 2009).

### 3.1 Keamanan Pada Fase Perencanaan

Fase perencanaan merupakan proses yang pertama yang dilakukan investigasi awal untuk mengumpulkan permasalahan-permasalahan yang ada di dalam sistem terutama yang berkaitan dengan keamanan system.

Membangun perangkat lunak yang aman adalah tanggung jawab semua stakeholder pada saat *software development lifecycle* (SLDC). Tujuan keamanan siklus hidup perangakat lunak yang professional *(Secure Software Life Cycle Professional (SSLP)* yaitu memastikan bahwa perangkat lunak yang dibangun tidak rentan terhadap gangguan keamanan (Mano Paul, 2008).

Perangkat lunak tidak 100% aman. Namun, perangkat lunak dapat dirancang, dikembangkan dengan *secure mindset*. Faktor kebutuhan kontrol keamanan diperlukan untuk meminimalkan dampak eksploitasi. Berikut ini merupakan misi dari SSLP dalam membangun pertahanan perangkat lunak dari *hack-resilient* (Mano Paul, 2008):

1. Lindungi kepercayaan pelanggan anda.
2. Mengetahui bisnis anda dan terjamin.
3. Memahami teknologi perangkat lunak.
4. Patuh terhadap peraturan dan Privasi.
5. Mengetahui Prinsip dasar keamanan software.
6. Pastikan melindungi informasi yang sensitive.
7. Desain software dengan fitur yang aman.
8. Mengembangkan *software* dengan fitur yang aman.
9. *Deploy* (Menyebarkan) software dengan fitur yang aman.
10. Mempelajari *(educate yourself)* cara membangun software yang aman

Selain hal di atas, pada tahap perancangan diperlukan validasi terhadap kebijakan,standart dan lainnya yang berkaitang dengan keamanan perangkat lunak dan dikonfirmasikan dengan stakeholdernya *(User)* hingga mereka menyetujui. Di bawah ini merupakan prinsip mengembangkan perangkat lunak secara aman (Curtis, 2005).

1 *Economic of Mechanism* : Melakukan desain sederhana dan sekecil mungkin
2 *Fail-Safe defaults* : Hak akses diberikan dengan jelas.
3 *Complete mediation* : Setiap akses harus diperiksa otoritasnya.
4 *Open design :* Desain sebaiknya tidak di rahasiakan namun tetap di jaga.
*5* *Separation of privilege :* Bila memungkinkan, mekanisme proteksi memerlukan paling tidak dua kunci untuk membukanya, hal ini akan lebih aman daripada hanya mempunyai satu kunci.
6 *Least Privilage* : Setiap program dan setiap pemakai dari system sebaiknya mengoperasikan dengan menggunakan minimal hak akases yang dibutuhkan untuk menyelesaikan pekerjaanya.
7 *Least Common Mechanism* : Meminimalkan jumlah mekanisme umum untuk pemakai.

8 *Psychological acceptability* : Human interface didesain untuk kemudahan penggunaan sehingga para pemakai secara rutin dan otomatis dapat mengaplikasikan mekanisme keamanan secara benar.

### 3.2 Keamanan Pada Fase Desain

Desain merupakan representasi pengambilan keputusan untuk memenuhi persyaratan perancangan perangkat lunak. Pada fase desain perangkat lunak yang perlu diperhatikan adalah :

a) Mengidentifikasi dan Mengevaluasi Ancaman

Menggunakan thread modeling untuk secara sistematis mengidentifikasi ancaman jauh lebih baik. Pemodelan biasanya dilakukan dengan memberi peringkat terhadap ancaman yang mungkin terjadi berdasarkan resiko serangan. Frekwensi kejadian, yang dikaitkan dengan kerugian atau kerusakan potensial yang bisa terjadi. Hal ini bisa memudakan kita dalam menangani ancaman dengan urutan yang sesui.

b) Membuat Desain yang aman

Sebaiknya menggunakan prinsip-prinsip desain yang sudah dicoba dan teruji dan focus pada area kritis dimana butuh pendekatan yang benar dan sering terjadi kesalahan yang meliputi : Validasi input, autentikasi, otorisasi , manajemen konfigurasi, proteksi data sensitive, kriptografi, manipulasi parameter, dan logging. Harus ada pula member perhatian yang serius pada isu-isu deployment termasuk topologi. Infrastruktur jaringan , kebijakan keamanan, dan prosedur.

c) Memeriksa Arsitektur dan Desain

Desain aplikasi harus diperiksa dalam hubunganya dengan lingkungan deployment dan kebijakan keaman yang terkait. Perlu membuat batasan yang ditentukan berdasarkan keamanan pada lapisan infrastruktur termasuk jaringan, firewall, remote application servers dan menganalisa pendekatan yang di ambil dalam tiap area.

Selian di atas (Bandar & dkk, 2010) mengusulkan sebuah set baru berupa metrik yang mampu menilai tanda-tanda keamanan aliran infromasi berorientasi obyek . Metrik tersebut memungkinkan pengembang perangkat lunak untuk membandingkan tingkat keamanan berbagai alternatife desain.

### 3.3 Keamanan Pada Fase Implementasi

Pastikan desain diterjemahkan ke dalam kode selama pelaksanaan. Berikut ini adalah kegiatan yang perlu dilakukan selama pelaksanaan untuk mencapai kode yang (Khan & Zulkernine, 2009) :

a) High-level bahasa pemrograman memberikan tingkat aman yang mendekati kesempurnaan. Namun, terkadang kita perlu menggunakan bahasa pemrograman yang low level untuk mendapatkan akses langsung ke perangkat keras. Bahasa pemrograman yang High Level harus dipilih namun harus tetap melihat persyaratan pemrograman pengembangan perangkat lunak.
b) Standar keamanan *coding* dan mengikuti petunjuk harus diikuti untuk menghindari kesalahan *source code.*
c) Unit pengujian (testing) harus dilakukan dengan memikirkan masalah keamanan. Kesalahan keamanan dilaporkan dalam perangkat lunak yang digunakan sebagai referensi.

### 3.4 Fase *Assurance*(Jaminan)

Selama fase jaminan, perangkat lunak tersebut harus akan diperiksa secara berkala untuk memenuhi persyaratan. Kegiatan yang dilakukan selama fase ini adalah sebagai berikut (Khan & Zulkernine, 2009) :

1. Kode harus diperiksa (Jika ada kesalahan berdasarkan daftar dari kesalahan yang dilaporkan sebelumnya ) untuk mengidentifikasi perangkat lunak dan kesalahan keamanan. Selain itu, alat analisis statis harus digunakan untuk menemukan kesalahan. Kesalahan yang ditemukan harus segera diselesaikan.
2. Uji kasus harus dikembangkan berdasarkan kebutuhan fungsional dan keamanan.
3. Integrasi dan pengujian perlu dilakukan berdasarkan pengujian syarat keamanan. Penetration harus dilakukan berdasarkan kerentanan yang diketahui dan potensi serangan yang akan di temui oleh perngkat lunak. Kesalahan keamanan yang ditemukan harus dihilangkan.

## 4. DISKUSI

Dari analisis di atas tentunya akan membuat kita semakin memahami pentingnya membangun suatu perangkat lunak yang didalamnya sudah terbangun keamanan.

a) Ada cara mengukur keamanan perangkat lunak menurut (Khan & Zulkernine, 2008) yang bisa kita gunakan. Namun, sebelumnya kita harus memiliki daftar dari semua kerentanan yang ada pada perangkat lunak. Setelah kita memiliki daftar kerentanan barulah jumlah kerentanan dalam persyaratan spesifikasi, dokumen desain, dan kode dapat di tulis $VR_T$ , $VD_T$dan $VC_T$ Selain itu, jumlah total kerentanan selama persyaratan spesifikasi , dokumen desain, dan kode ditulis sebagai berikut $VR_R$, $VD_R$ dan $VC_R$

setelah mendapatkan jumlah total, kita dapat menghitung indeks dengan menggunakan persamaan sebagai berikut :

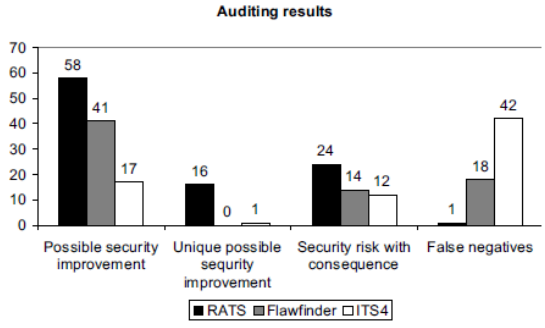
$$VI_{REG} = \frac{VR_R}{VR_T} * 100 \quad (1)$$

$$VI_{DES} = \frac{VD_R}{VT_T} * 100 \quad (2)$$

$$VI_{COD} = \frac{VC_R}{VC_T} * 100 \quad (3)$$

b) (Dai & Bai, 2011) menggunakan pendekatan dengan menggabungkan *Unit Modeling Language* untuk menjebatani keamanan perangkat lunak dengan aplikasi keamanan perangkat lunak dengan cara mengkoordinasikan sumber daya untuk mengembangkan perangkat lunak yang dengan keamanan sistematis dan efesien.

c) (Carlsson & Baca, 2005) melakukan revisi kode dengan menggabungkan pemeriksaan manual dan analisis *static tools* untuk melihat rata-rata baris kode untuk satu kerentanan yang dieksploitasi. Tujuanya adalah meningkatkan keamanan kode program. Dari hasil audit ditemukan 823 kerentanan keamanan dari 4000 baris kode. Pada gambar di bawah ini ada 59 atau sekitar 7,2 % dari peringatan total.




Gambar 3. *Auditing results* setelah manual *examination*

Metode di atas merupakan salah satu cara yang bisa kita pakai untuk meningkatkan keamanan perangkat lunak yang kita bangun yaitu dengan menggunakan *tools* untuk mengaudit barisan kode program yang sudah dibuat, sehingga dengan begitu dapat diketahui kerentanan dan jenis serangan yang akan menyerang perangkat lunak yang kita bangun.

## 5. KESIMPULAN

Penyebab utama dari *vulnerability* adalah kerena adanya lubang pada perangkat lunak. Kemanan perangkat lunak harus sistematis berada pada *software development life cycle* (SDLC). Setiap fase memiliki tahap keamanan yang harus diperhatikan pada saat membangun perangkat lunak agar nantinya dapat menjadikan produk yang handal dan aman.

Kemanan dalam perangkat lunak dapat diukur dengan melakukan perhitungan dengan menggunakan persamaan yang sudah dijelaskan sebelumnya atau dengan cara menggunakan tools untuk mengaudit barisan kode. Dengan begitu kita dapat membangun keamanan perangkat lunak dapat membantu membangun keamanan secara *"built in"*.

Perangkat lunak yang memiliki keamanan secara *"built in"* akan lebih mendapatkan kepercayaan dari user. Sekarang sudah saatnya membangun perangkat lunak yang memiliki keamanan secara *"built in"*.

Selain memberikan kepercayaan yang tinggi, fase-fase keamanan software secara *"built in"* akan memberikan *cost* (waktu dan biaya) garansi yang lebih minim, serta memberikan *assurance* yang lebih kepada user dibandingkan dengan perangkat lunak yang tidak memiliki keamanan secara *"built in"*.

## PUSTAKA

Alshammari, B., fridge, C., & Corney, d. (2010). Security Metrics for Object-Oriented Design.

Carlsson, B., & Baca, D. (2005). Software Security Analysis - Execution Phase Auidit. *IEEE Computer Society* .

Chandra, S., Khan, R. A., & Agrawal, A. (2009). Security Estimation Framework: Design Phase Perspective. *IEEE Computer Society* .

Curtis, C. (2005). case Study : An Evolution of putting security into SDLC.

Dai, L., & Bai, Y. (2011). An Organization-Driven Approach for Enterprise. *IEEE Computer Society* .

Eugene Lebanidze. (2005). Securing Enterprise Web Applications at the Source : An Application Security Perspective .

Istiyanto, J. E. (Pemain). (2011). *Goal Security Management.* Kelas, Yogyakarta, Yogyakarta, Sleman.

Khan, M. U., & Zukernine, M. (2009). Actifity and Artifact views of a secure software development proses. *IEEE Computer Society* , 2.

Khan, M. U., & Zulkernine, M. (2008). Quantifying Secrity in Secure Software Development Phasess. *IEEE Computer Society* , 1.

Mano Paul. (2008). The ten Best Practices for secure software development. *Information System Security Certification COnsortiu,Inc* .

Peterson, G. (2004). Software Security. *CHI Publlishing Ltd* .