Panduan *Secure* SDLC

Oleh : Restia Moegiono {CEH|CHFI|ECSA|QRMO}

Riwayat Perubahan

Versi	Tanggal	Personel	Perubahan
0.0	25 Maret 2023	Restia Moegiono	Versi awal Panduan *Secure* SDLC

Daftar Isi

Pendahuluan

A. Latar Belakang

Secure Software Development Life Cycle (*Secure* SDLC) berupaya menjadikan keamanan sebagai tanggung jawab semua pihak, memungkinkan pengembangan perangkat lunak yang aman sejak awal. Sederhananya, *secure* SDLC penting karena keamanan dan integritas perangkat lunak itu penting. Ini mengurangi risiko kerentanan keamanan pada produk perangkat lunak dalam produksi, serta meminimalkan dampaknya jika ditemukan. *Secure* SDLC menempatkan keamanan di depan dan di tengah, yang jauh lebih penting dengan repositori kode sumber yang tersedia untuk umum, beban kerja *cloud*, kontainerisasi, dan rantai manajemen pemasok. *Secure* SDLC menyediakan kerangka kerja standar untuk menentukan tanggung jawab, meningkatkan visibilitas dan meningkatkan kualitas perencanaan dan pelacakan serta mengurangi risiko.

B. Tujuan

1. Untuk memastikan pengembangan perangkat lunak dilakukan secara aman.
2. Menjadi panduan cepat bagi pengembang perangkat lunak dalam menerapkan *Secure* SDLC.

C. Manfaat

1. Pengurangan biaya
   Berkat identifikasi awal masalah keamanan yang memungkinkan diimplementasikan kontrol keamanan secara paralel, maka tidak perlu lagi melakukan *patch* setelah fase produksi.
2. Mengutamakan keamanan
   Secure SDLC membangun budaya yang berfokus pada keamanan, menciptakan lingkungan kerja yang mengutamakan keamanan, dan semua pihak memperhatikannya. Sehingga perbaikan terjadi di seluruh bagian organisasi.
3. Strategi pengembangan
   Menentukan kriteria keamanan sejak awal meningkatkan strategi teknologi, membuat semua anggota tim pengembangan mengetahui kriteria keamanan produk, dan memastikan keamanan pengembangan di sepanjang siklus hidup.
4. Keamanan yang lebih baik
   Setelah proses *secure* SDLC dilakukan, maka postur keamanan meningkat di seluruh bagian organisasi. Organisasi yang sadar akan keamanan dapat mengurangi risiko serangan siber secara signifikan.

D. Kriteria *Secure* SDLC

1. Persyaratan Keamanan

Tahap ini bertujuan untuk:

a. Mengidentifikasi dan menentukan persyaratan keamanan yang relevan untuk perangkat lunak yang dikembangkan.
b. Mengidentifikasi persiapan keamanan untuk pengembangan perangkat lunak.

c. Memperoleh kebutuhan perlindungan menggunakan klasifikasi data, penggunaan dan penyalahgunaan *case modelling*, dan manajemen risiko.

1.1 Sumber dari Persyaratan Keamanan

Tahap ini bertujuan untuk:

a. Mendefinisikan dan menangani tujuan atau sasaran keamanan organisasi secara eksplisit.
b. Mengidentifikasi persyaratan yang dapat diterapkan pada konteks bisnis dan fungsionalitas perangkat lunak yang melayani konteks tersebut.

1.1.1 Identifikasi Persyaratan *Core Security*

a. Persyaratan Kerahasiaan (*Confidentiality*)

Persyaratan kerahasiaan bertujuan untuk memberikan pelindungan terhadap pengungkapan data atau informasi yang tidak sah yang bersifat rahasia atau sensitif. Mekanisme perlindungan kerahasiaan adalah sebagai berikut:

1) Kriptografi

Kriptografi adalah mekanisme pelindungan yang bertujuan untuk mencegah pengungkapan informasi yang dianggap rahasia. Jenis mekanisme tersebut antara lain:

a) Mekanisme terbuka (*overt*)

Contohnya enkripsi dan *hash*. Tujuan dari penulisan rahasia terbuka adalah untuk membuat informasi tersebut tidak dapat dipahami atau tidak dapat dipahami secara manusiawi bahkan jika diungkapkan sekalipun.

b) Mekanisme rahasia (*covert*)

Contohnya steganografi dan *digital watermarking*. Tujuan dari penulisan rahasia terselubung adalah untuk menyembunyikan informasi di dalam suatu media atau bentuk yang lain.

2) Penyamaran (*masking*)

Penyamaran adalah mekanisme pelindungan yang lebih lemah dari mekanisme pelindungan kriptografi, dimana informasi asli diberi tanda bintang atau X. Mekanisme penyamaran digunakan untuk melindungi dari serangan selancar bahu (*shoulder surfing attack*), yaitu serangan dimana seseorang melihat dari balik bahu orang lain dan mengamati informasi sensitif. Penyamaran nomor kartu kredit, kecuali 4 (empat) digit terakhir saat dicetak pada kuitansi atau ditampilkan di layar, adalah contoh penyamaran yang memberikan pelindungan kerahasiaan.

Persyaratan kerahasiaan harus ditentukan sepanjang siklus hidup informasi, mulai dari data asal hingga data dihentikan. Penting untuk secara eksplisit menyatakan persyaratan kerahasiaan untuk data non-publik:

1) Dalam transmisi *(In Transit)*

Ketika data ditransmisikan melalui jaringan yang tidak terlindungi, yaitu, data bergerak.

2) Dalam Pemrosesan (*In Process*)
   Ketika data disimpan dalam memori komputer atau media untuk diproses.
3) Dalam Penyimpanan (*In Storage*)
   Ketika data tidak aktif, dalam sistem transaksional maupun sistem non-transaksional, termasuk arsip, yaitu data tidak aktif.

Catatan:
Persyaratan kerahasiaan juga mungkin terikat waktu (*time bound*).

b. Persyaratan Integritas (*Integrity*)

Persyaratan integritas bertujuan untuk mengatasi 2 (dua) aspek utama keamanan perangkat lunak, yaitu jaminan keandalan dan pelindungan atau pencegahan terhadap modifikasi yang tidak sah. Integritas mengacu tidak hanya pada pelindungan modifikasi perangkat lunak (integritas sistem) tetapi juga data yang ditangani oleh perangkat lunak (integritas data).

Keandalan pada dasarnya bertujuan untuk memastikan bahwa perangkat lunak berfungsi seperti yang dirancang dan diharapkan. Ini juga dimaksudkan untuk memberikan kontrol keamanan yang akan memastikan bahwa keakuratan perangkat lunak dan data tetap terjaga. Pelindungan integritas juga mempertimbangkan kelengkapan dan konsistensi perangkat lunak atau data yang ditangani perangkat lunak.

c. Persyaratan Ketersediaan (*Availability*)

Persyaratan ketersediaan bertujuan untuk memastikan tidak ada gangguan terhadap operasional bisnis. Persyaratan ketersediaan adalah persyaratan perangkat lunak yang memastikan pelindungan terhadap upaya penghancuran perangkat lunak dan/atau data, sehingga membantu pencegahan *Denial of Service* (DoS) bagi pengguna.

d. Persyaratan Autentikasi

Persyaratan autentikasi bertujuan untuk memverifikasi dan memastikan keabsahan dan validitas identitas yang mengajukan klaim entitas untuk verifikasi. Kredensial autentikasi dapat diberikan oleh berbagai faktor atau kombinasi faktor yang mencakup pengetahuan, kepemilikan, atau karakteristik.

Autentikasi dua faktor (*two factor authentication/*2FA) adalah autentikasi yang menggunakan 2 (dua) faktor untuk memvalidasi klaim dan/atau kredensial entitas. Sedangkan autentikasi multi-faktor (*multi factor authentication/*MFA) adalah autentikasi yang menggunakan lebih dari 2 (dua) faktor.

Bentuk autentikasi yang paling umum adalah sebagai berikut:

1) Autentikasi Anonim
   Autentikasi anonim adalah sarana akses ke area publik perangkat lunak tanpa meminta kredensial seperti *username* dan *password*.
2) Autentikasi Dasar

Autentikasi dasar adalah salah satu spesifikasi HTTP 1.0, yang dicirikan oleh *browser* yang meminta pengguna untuk memberikan kredensial dalam bentuk teks jelas (*cleartext*) atau kode.

3) Autentikasi Intisari (*Digest*)
   Autentikasi *digest* adalah mekanisme tantangan/respons (*challenge/response*), yang mengirimkan intigasi (*digest*) pesan (berupa nilai *hash*) dari kredensial asli.
4) Autentikasi Terintegrasi
   Autentikasi terintegrasi umumnya dikenal sebagai autentikasi New *Technology LAN Manager* (NTLM) atau autentikasi tantangan/respons New *Technology* (NT), seperti autentikasi intisari (*digest*).
5) Autentikasi Berbasis Sertifikat Digital Milik Klien
   Autentikasi berbasis sertifikat digital milik klien berfungsi dengan memvalidasi identitas pemegang sertifikat digital. Sertifikat digital ini dikeluarkan untuk organisasi atau pengguna oleh *Certification Authority* (CA) yang menjamin keabsahan pemegangnya.
6) Autentikasi Berbasis Formulir
   Autentikasi berbasis formulir mengharuskan pengguna untuk memberikan *username* dan *password* untuk tujuan autentikasi, dan kredensial tersebut akan divalidasi terhadap direktori penyimpanan, yang dapat berupa direktori aktif, *database*, atau *file* konfigurasi.
7) Autentikasi Berbasis Token
   Autentikasi berbasis token biasanya digunakan bersama dengan autentikasi berbasis formulir dimana *username* dan *password* disediakan untuk verifikasi. Setelah verifikasi, token akan dikeluarkan untuk pengguna yang memberikan kredensial. Token kemudian digunakan untuk memberikan akses ke sumber daya yang diminta.
8) Autentikasi Berbasis *Smart Card*
   Autentikasi berbasis *smart card* dilakukan berdasarkan validasi kepemilikan (*what you have*). *Smart card* berisi *microchip* tertanam yang dapat diprogram dan digunakan untuk menyimpan kredensial autentikasi pemilik *smart card*.
9) Autentikasi Biometrik
   Autentikasi biometrik menggunakan karakteristik biologis (*what you are*) untuk memberikan kredensial identitas. Fitur biologis seperti pola pembuluh darah retina, fitur wajah, dan sidik jari digunakan untuk tujuan verifikasi identitas.

e. Persyaratan Otorisasi

Persyaratan otorisasi bertujuan untuk mengonfirmasi bahwa entitas yang diautentikasi memiliki hak dan hak istimewa (*privilege*) yang diperlukan untuk mengakses dan melakukan tindakan pada sumber daya yang diminta.

Pada aktivitas mengidentifikasi subjek dan objek, subjek adalah entitas yang meminta akses, sedangkan objek adalah item yang akan ditindaklanjuti oleh subjek. Subjek dapat berupa pengguna manusia atau proses perangkat lunak. Tindakan pada objek

perlu ditangkap secara eksplisit, umumnya adalah operasi data *Create, Read, Update,* atau *Delete* (CRUD).

Model kontrol akses untuk diterapkan terutama dari jenis berikut:

1) *Discretionary Access Control* (DAC)
   Model kontrol akses ini membatasi akses ke objek berdasarkan identitas subjek dan/atau grup tempat objek tersebut berada. DAC diimplementasikan baik dengan menggunakan identitas atau peran. DAC sering diamati diimplementasikan dengan menggunakan *access control list* (ACL), hubungan antara individu (subjek) dan sumber daya (objek) bersifat langsung dan pemetaan individu ke sumber daya oleh pemiliknya.
2) *Non-Discretionary Access Control* (NDAC)
   Model kontrol akses ini ditandai dengan perangkat lunak yang menerapkan kebijakan keamanan. Itu tidak bergantung pada kepatuhan subjek dengan kebijakan keamanan. Aspek *nondiscretionary* adalah bahwa itu tidak dapat dihindari dapat dikenakan pada semua subjek.
3) *Mandatory Access Control* (MAC)
   Pada model kontrol akses ini, akses ke objek dibatasi untuk subjek berdasarkan sensitivitas informasi yang terkandung dalam objek. Sensitivitas diwakili oleh label. Hanya subjek yang memiliki hak istimewa dan otorisasi formal yang sesuai yang diberi akses ke objek.
4) *Role Based Access Control* (RBAC)
   Pada model kontrol akses ini, individu (subjek) memiliki akses ke sumber daya (objek) berdasarkan peran yang ditugaskan kepadanya. Izin untuk mengoperasikan objek seperti *Create, Read, Update,* atau *Delete* juga ditentukan dan ditentukan berdasarkan tanggung jawab dan wewenang dalam fungsi pekerjaan.
5) *Resource Based Access Control*
   Pada model kontrol akses ini, akses diberikan berdasarkan sumber daya. Model kontrol akses berbasis sumber daya berguna dalam arsitektur yang didistribusikan dan multi-tier, termasuk arsitektur berorientasi layanan.

f. Persyaratan Akuntabilitas

Persyaratan akuntabilitas bertujuan untuk membantu membangun catatan riwayat tindakan pengguna. Jejak audit dapat membantu mendeteksi saat pengguna yang tidak sah melakukan perubahan, atau pengguna yang berwenang membuat perubahan yang tidak sah, dimana keduanya merupakan kasus pelanggaran integritas.

1.1.2 Mengidentifikasi Persyaratan Umum

a. Persyaratan Manajemen Sesi

Setelah autentikasi berhasil, *session identifier* (ID) dikeluarkan untuk pengguna dan *session* ID tersebut digunakan untuk melacak perilaku pengguna dan mempertahankan

status terautentikasi untuk pengguna tersebut hingga sesi tersebut ditinggalkan atau status berubah dari terautentikasi menjadi tidak terautentikasi.

b. Persyaratan Manajemen *Error* dan *Exception*

Error dan *exception* merupakan sumber potensial pengungkapan informasi.

c. Persyaratan Manajemen Parameter Konfigurasi

Parameter dan kode konfigurasi perangkat lunak yang menyusun perangkat lunak membutuhkan perlindungan terhadap peretas. Parameter dan kode tersebut biasanya perlu diinisialisasi sebelum perangkat lunak dapat dijalankan.

1.1.3 Mengidentifikasi Persyaratan Operasional

Tahap ini bertujuan untuk:

- Mengidentifikasi persyaratan seperti jumlah koneksi *database* untuk akses bersamaan;
- Mengidentifikasi adanya saling ketergantungan dengan aplikasi lain dalam ekosistem komputasi;
- Mengidentifikasi sumber daya komputasi dan bersama yang diperlukan;
- Mengidentifikasi kemampuan perangkat lunak yang diperlukan saat perangkat lunak melayani bisnis dengan fungsi yang dimaksudkan.

a. Persyaratan Lingkungan Penerapan

Persyaratan terkait tentang lingkungan dimana perangkat lunak akan digunakan harus diidentifikasi dan dianalisis.

b. Persyaratan Pengarsipan

Persyaratan pengarsipan harus secara eksplisit diidentifikasi karena arsip yang disimpan baik merupakan sarana untuk kelangsungan bisnis atau sebagai kebutuhan untuk memenuhi persyaratan peraturan atau kebijakan organisasi.

c. Persyaratan Anti-Pembajakan (*Anti-Piracy*)

Code obfuscation, code signing, anti-tampering, lisensi, dan mekanisme perlindungan IP harus dimasukkan sebagai bagian dari dokumentasi persyaratan, terutama jika dalam bisnis pembuatan dan penjualan perangkat lunak komersial.

1.1.4 Mengidentifikasi Persyaratan Lain

a. Persyaratan Urutan dan Waktu

Cacat desain pada urutan dan waktu dalam perangkat lunak dapat menyebabkan apa yang umumnya dikenal sebagai *race condition* atau serangan *Time of Check/Time of Use* (TOC/TOU). Faktanya, *race condition* adalah salah satu kelemahan paling umum yang diamati dalam desain perangkat lunak.

b. Persyaratan Internasional

Persyaratan internasional dapat terdiri dari 2 (dua) jenis, yaitu hukum dan teknologi. Persyaratan hukum bertujuan agar tidak melanggar peraturan apa pun, sedangkan persyaratan teknologi bertujuan untuk menentukan karakter pengkodean dan mengarahkan tampilan.

c. Persyaratan Pengadaan

Identifikasi dan penentuan persyaratan keamanan perangkat lunak tidak kalah pentingnya ketika diambil keputusan untuk membeli perangkat lunak daripada membangunnya sendiri.

1.2 Klasifikasi Data

Tahap ini bertujuan untuk memastikan data atau informasi sebagai aset digital yang paling berharga akan terlindungi.

1.2.1 Tipe Data

Tahap ini bertujuan untuk mengklasifikasikan data ke dalam tipe-tipe sebagai berikut:

a. Data Terstruktur

Data terstruktur mengacu pada data yang diorganisasikan ke dalam struktur yang dapat diidentifikasi. Contohnya data terstruktur dalam *database* (disimpan dalam bentuk kolom dan baris).

b. Data Tidak Terstruktur

Data tidak terstruktur merujuk pada data yang tidak memiliki struktur yang dapat diidentifikasi. Contohnya data tidak terstruktur termasuk gambar, video, *email*, dokumen, dan teks.

1.2.2 Memberi Label pada Data

Pelabelan data adalah upaya untuk memberikan label sesuai klasifikasi data pada data, berdasarkan dampak potensial terhadap aspek kerahasiaan, integritas, dan ketersediaan (C-I-A), terhadap ancaman pengungkapan, perubahan, atau penghancuran.

1.2.3 Kepemilikan dan Peran pada Data

Keputusan untuk mengklasifikasikan data, siapa yang memiliki hak akses, apa tingkat akses yang dimiliki, dan keputusan lainnya adalah keputusan yang harus dibuat oleh pemilik data.

1.2.4 *Data Lifecycle Management* (DLM)

Pendekatan berbasis kebijakan yang melibatkan prosedur dan praktik untuk melindungi data di sepanjang siklus hidup informasi, dimulai sejak data dibuat hingga data dibuang atau dihapus.

1.2.5 Persyaratan Privasi

Klasifikasi data dapat membantu dalam mengidentifikasi data yang perlu menerapkan persyaratan perlindungan privasi. Mengkategorikan data ke dalam berbagai tingkatan privasi, berdasarkan dampak pada pengungkapan, pengubahan, dan/atau penghancuran, dapat memberikan wawasan untuk memastikan bahwa tingkat kontrol privasi yang sesuai telah tersedia.

Pedoman praktik terbaik untuk privasi data yang perlu disertakan dalam analisis, desain, dan arsitektur persyaratan perangkat lunak dapat diatasi jika mematuhi aturan berikut:

a. Jika data tidak dibutuhkan, jangan mengumpulkannya;

b. Jika data perlu dikumpulkan hanya untuk diproses, maka kumpulkan setelah memberi tahu pengguna bahwa organisasi mengumpulkan informasi pengguna dan pengguna telah menyetujuinya, tetapi jangan menyimpannya.
c. Jika data perlu dikumpulkan untuk pemrosesan dan penyimpanan, maka kumpulkan dengan persetujuan pengguna, dan simpan hanya untuk periode penyimpanan eksplisit yang sesuai dengan kebijakan organisasi dan/atau persyaratan peraturan.
d. Jika terdapat kebutuhan untuk mengumpulkan dan menyimpan data, maka jangan mengarsipkannya jika data tersebut telah habis masa pakainya dan tidak ada persyaratan retensi.

Persyaratan dan kontrol privasi adalah sebagai berikut:

a. Anonimisasi Data

Dengan menghapus pengidentifikasi pribadi dari data secara permanen dan menyeluruh, anonimitas dapat terjamin. Anonimisasi adalah proses menghapus informasi pribadi dari data. Data anonim tidak dapat ditautkan ke satu akun individual mana pun.

Teknik anonimisasi berguna untuk memastikan privasi data, antara lain: penggantian (substitusi), supresi (penghilangan), generalisasi (identifikasi khusus informasi diganti menggunakan bentuk yang lebih umum), dan pengacauan (pengacakan, yang melibatkan perubahan acak pada data).

b. Pembuangan

Semua perangkat lunak rentan hingga perangkat lunak dan data yang diproses, ditransmisikan, dan disimpan, dibuang dengan cara yang aman. Hal ini sangat penting jika datanya sensitif atau terdapat data pribadi di dalamnya.

Sebagian besar peraturan privasi mensyaratkan penerapan kebijakan dan prosedur untuk mengatur pembuangan akhir informasi pribadi dan sanitasi perangkat keras dan media penyimpanan elektronik sebelum dikondisikan untuk dapat digunakan kembali.

c. Model Keamanan

Abstraksi formal dari kebijakan keamanan yang terdiri dari sekumpulan persyaratan keamanan yang perlu menjadi bagian dari perangkat lunak, sehingga perangkat lunak tahan terhadap serangan, dapat mentolerir serangan yang tidak dapat dilawan, dan dapat pulih dengan cepat dari keadaan yang tidak diinginkan jika dikompromikan.

d. Pseudonimisasi

Pseudonimisasi adalah manajemen data dan prosedur de-identifikasi dimana bidang informasi yang dapat diidentifikasi secara pribadi dalam data rekaman diganti dengan 1 (satu) atau lebih pengidentifikasi buatan atau samaran. Sebuah samaran tunggal untuk setiap bidang yang diganti atau kumpulan bidang yang diganti menyebabkan data rekaman kurang dapat diidentifikasi namun tetap cocok untuk analisis data dan pemrosesan data.

1.3 Pemodelan *Use Case* dan *Misuse Case*

Tahap ini bertujuan untuk mengidentifikasi kemungkinan perilaku buruk dan merekomendasikan persyaratan keamanan yang relevan berdasarkan fungsionalitas untuk perangkat lunak yang dikembangkan.

1.3.1 Menganalisis Skenario *Use Case*

Use case perangkat lunak menggambarkan perilaku yang diinginkan oleh pemilik perangkat lunak, sehingga akan mengidentifikasi perilaku perangkat lunak. Pemodelan dan pembuatan diagram *use case* sangat berguna untuk menentukan persyaratan ini. Pemodelan ini efektif dalam mengurangi persyaratan bisnis yang ambigu dan tidak lengkap dengan secara eksplisit menentukan dengan tepat kapan dan dalam kondisi apa perilaku tertentu terjadi.

Pemodelan *use case* meliputi pengidentifikasian aktor, perilaku perangkat lunak yang dimaksudkan (*use case*), dan urutan serta hubungan antara aktor dan *use case*. Aktor dapat berupa individu, peran, atau non-manusia.

Pada diagram *use* case: aktor direpresentasikan dengan *stick figure* dan skenario *use case* dengan bentuk elips, sedangkan panah mewakili interaksi atau hubungan antara *use case* dan aktor.

1.3.2 Menganalisis Skenario *Misuse Case*

Misuse case memodelkan skenario negatif untuk mengidentifikasi kelemahan yang menjadi ancaman. Skenario negatif adalah perilaku perangkat lunak yang tidak diinginkan oleh pemilik perangkat lunak dalam konteks *use case*. *Misuse case* memberikan wawasan tentang ancaman yang dapat terjadi terhadap perangkat lunak.

Dalam pemodelan *misuse case*, akan dibuat pemodelan untuk pelaku yang melakukan kesalahan, dan skenario atau perilaku yang tidak diinginkan. *Misuse case* dapat bersifat disengaja atau tidak disengaja. *Misuse case* dapat dibuat melalui *brainstorming* skenario negatif seperti penyerang.

1.3.3 Membuat Model Serangan

Untuk membuat model serangan dengan pertimbangan eksplisit dari serangan yang diketahui, diperlukan serangkaian persyaratan dan daftar ancaman, kemudian dapat ditelusuri daftar serangan yang diketahui satu per satu dan memikirkan apakah terdapat serangan yang sama berlaku untuk perangkat lunak. Untuk membuat model serangan, lakukan langkah berikut:

a. Pilih pola serangan yang relevan dengan perangkat lunak. Buatlah *misuse case* di sekitar pola serangan tersebut.
b. Sertakan siapa saja yang dapat memperoleh akses ke perangkat lunak karena sumber ancaman harus mencakup semua potensi sumber bahaya terhadap perangkat lunak.

1.3.4 Memilih Kontrol Mitigasi

Untuk mengusulkan kontrol mitigasi berdasarkan serangan yang diidentifikasi dari langkah sebelumnya, soroti intinya dengan mengusulkan kontrol keamanan. Hal ini membantu dalam memenuhi persyaratan keamanan.

1.4 Manajemen Risiko

Tahap ini bertujuan untuk:

a. Mengaktifkan pengamanan perangkat lunak yang menyimpan, memproses, atau mengirimkan informasi organisasi;
b. Memungkinkan pihak manajemen membuat keputusan manajemen risiko yang terinformasi dengan baik untuk membenarkan pengeluaran yang merupakan bagian dari anggaran perangkat lunak;
c. Membantu pihak manajemen dalam mengotorisasi (atau mengakreditasi) perangkat lunak berdasarkan dokumentasi pendukung yang dihasilkan dari kinerja manajemen risiko.

1.4.1 Penilaian Risiko

Tahap ini bertujuan untuk:

a. Menentukan sejauh mana potensi ancaman dan risiko yang terkait dengan perangkat lunak di seluruh *secure* SDLC-nya.
b. Untuk mengidentifikasi kontrol yang tepat untuk mengurangi atau menghilangkan risiko selama proses mitigasi risiko.

Metodologi penilaian risiko mencakup 9 (sembilan) langkah utama, dimana langkah 2, 3, 4, dan 6 dapat dilakukan secara paralel setelah langkah 1 selesai. Langkah-langkahnya seperti di bawah ini:

a. Langkah 1 – Karakterisasi Sistem

Karakterisasi sistem bertujuan untuk menentukan ruang lingkup penilaian risiko. Pada langkah ini, batas-batas perangkat lunak diidentifikasi, bersama dengan sumber daya dan informasi yang membentuk sistem. Karakterisasi perangkat lunak menetapkan ruang lingkup upaya penilaian risiko, menggambarkan batasan otorisasi operasional dan memberikan informasi yang penting untuk menentukan risiko, misalnya, perangkat keras, perangkat lunak, konektivitas sistem, dan divisi yang bertanggung jawab atau personel pendukung.

b. Langkah 2 – Identifikasi Ancaman

Identifikasi ancaman bertujuan untuk mengidentifikasi sumber ancaman potensial dan menyusun pernyataan ancaman yang mencantumkan sumber ancaman potensial yang berlaku untuk perangkat lunak yang sedang dievaluasi. Sumber ancaman didefinisikan sebagai apa saja keadaan atau peristiwa yang berpotensi menyebabkan kerusakan pada perangkat lunak. Sumber ancaman secara umum dapat berupa alam, manusia, atau lingkungan.

c. Langkah 3 – Identifikasi Kerentanan

Analisis ancaman terhadap perangkat lunak harus mencakup analisis kerentanan yang terkait dengan lingkungan perangkat lunak. Identifikasi kerentanan bertujuan untuk mengembangkan daftar kerentanan perangkat lunak (cacat atau kelemahan) yang dapat dimanfaatkan oleh sumber ancaman potensial.

d. Langkah 4 – Analisis Kontrol

Analisis kontrol bertujuan untuk:

1) Menganalisis kontrol yang telah diterapkan atau direncanakan akan diterapkan untuk meminimalkan atau menghilangkan kemungkinan (atau probabilitas) dari sumber ancaman yang mengeksploitasi kerentanan perangkat lunak.
2) Memperoleh tingkat kemungkinan yang menunjukkan kemungkinan terjadinya kerentanan potensial yang dapat dieksekusi pada konstruksi lingkungan ancaman terkait (langkah 5), maka penerapan dari kontrol saat ini atau kontrol yang direncanakan harus dipertimbangkan.

e. Langkah 5 – Penentuan Kemungkinan

Untuk memperoleh tingkat kemungkinan yang menunjukkan kemungkinan terjadinya kerentanan potensial yang dapat dieksploitasi pada konstruksi lingkungan ancaman terkait, faktor-faktor yang mengatur berikut ini harus dipertimbangkan:

1) Motivasi dan kapabilitas sumber ancaman;
2) Sifat kerentanan;
3) Keberadaan dan efektivitas kontrol saat ini.

f. Langkah 6 – Analisis Dampak

Analisis dampak bertujuan untuk menentukan dampak buruk yang dihasilkan dari keberhasilan eksploitasi sumber ancaman terhadap kerentanan. Sebelum memulai analisis dampak, perlu untuk mendapatkan informasi yang diperlukan berikut ini:

1) Misi sistem (misalnya, proses yang dilakukan oleh perangkat lunak);
2) Kekritisan sistem dan data (misalnya, nilai atau kepentingan sistem bagi organisasi);
3) Sensitivitas sistem dan data.

g. Langkah 7 – Penentuan Risiko

Penentuan risiko bertujuan untuk menilai tingkat risiko untuk perangkat lunak. Penentuan risiko untuk pasangan ancaman/kerentanan dapat dinyatakan sebagai fungsi dari:

1) Kemungkinan sumber ancaman tertentu mencoba menggunakan kerentanan tertentu;
2) Besarnya dampak jika sumber ancaman berhasil mengeksploitasi kerentanan;
3) Kecukupan kontrol keamanan yang direncanakan atau yang diterapkan untuk mengurangi atau menghilangkan risiko.

h. Langkah 8 – Rekomendasi Kontrol

Kontrol yang dapat mengurangi atau menghilangkan risiko yang teridentifikasi sesuai dengan operasi organisasi, harus disediakan. Tujuan dari penerapan kontrol yang direkomendasikan tersebut adalah untuk mengurangi tingkat risiko pada perangkat lunak dan datanya ke tingkat risiko yang dapat diterima. Faktor-faktor berikut harus dipertimbangkan dalam merekomendasikan kontrol dan solusi alternatif untuk meminimalkan atau menghilangkan risiko yang teridentifikasi:

1) Efektivitas opsi yang direkomendasikan (misalnya, kompatibilitas sistem);
2) Legislasi dan regulasi;
3) Kebijakan organisasi;
4) Dampak operasional;

5) Keamanan dan keandalan.

i. Langkah 9 – Dokumentasi Hasil

Setelah penilaian risiko selesai (sumber ancaman dan kerentanan diidentifikasi, risiko dinilai, dan kontrol yang direkomendasikan diterapkan), maka hasilnya harus didokumentasikan dalam laporan resmi atau ringkas.

Laporan penilaian risiko adalah laporan yang membantu pihak manajemen dan pemilik risiko dalam membuat keputusan tentang kebijakan, prosedur, anggaran, dan operasi sistem dan manajemen perubahan. Laporan penilaian risiko harus disajikan pendekatan sistematis dan analitis untuk menilai risiko sehingga pihak manajemen dapat memahami risiko dan mengalokasikan sumber daya untuk mengurangi dan memperbaiki kerugian potensial.

1.4.2 Mitigasi Risiko

Tahap ini bertujuan untuk memprioritaskan, mengevaluasi, dan menerapkan kontrol pengurangan risiko yang sesuai yang direkomendasikan dari proses penilaian risiko.

a. Opsi Mitigasi Risiko

Organisasi harus mempertimbangkan untuk memilih dari beberapa opsi pada mitigasi risiko. Mungkin tidak praktis untuk mengatasi semua risiko yang teridentifikasi, jadi prioritas harus diberikan pada pasangan ancaman dan kerentanan yang berpotensi menyebabkan dampak atau kerugian yang signifikan. Pilihan mitigasi risiko adalah:

1) Menerima risiko, yaitu menerima potensi risiko dengan terus mengoperasikan perangkat lunak atau menerapkan kontrol untuk menurunkan risiko ke tingkat yang dapat diterima;
2) Menghindari risiko, yaitu menghindari risiko dengan menghilangkan penyebab dan/atau konsekuensi risiko. Misalnya, melewatkan fungsi tertentu dari perangkat lunak atau mematikan perangkat lunak saat risiko teridentifikasi;
3) Mitigasi risiko, yaitu membatasi risiko dengan menerapkan kontrol yang meminimalkan dampak merugikan dari ancaman yang menjalankan kerentanan. Misalnya penggunaan pendukung, pencegahan, kontrol yang bersifat detektif;
4) Mentransfer risiko, yaitu mentransfer risiko dengan menggunakan opsi lain untuk mengkompensasi kerugian, seperti membeli asuransi
5) Perencanaan risiko, yaitu mengelola risiko dengan mengembangkan rencana mitigasi risiko yang memprioritaskan, mengimplementasikan, dan memelihara kontrol;
6) Penelitian dan pengakuan, yaitu menurunkan risiko kerugian dengan mengakui adanya kerentanan dan meneliti kontrol untuk memperbaiki kerentanan.

b. Strategi Mitigasi Risiko

Untuk memberikan panduan tentang tindakan untuk mengurangi risiko dari ancaman manusia yang disengaja:

1) Ketika terdapat kerentanan, maka harus menerapkan teknik penjaminan untuk mengurangi tingkat kemungkinan pada kerentanan yang diekploitasi;

2) Ketika kerentanan dapat diekploitasi, maka terapkan perlindungan berlapis, desain arsitektural, dan kontrol administratif untuk meminimalkan risiko atau mencegah insiden;
3) Ketika biaya untuk mengatasi penyerang kurang dari keuntungan potensial, maka terapkan pelindungan untuk mengurangi motivasi penyerang dengan meningkatkan biaya untuk mengatasi penyerang. Misalnya, penggunaan kontrol perangkat lunak untuk membatasi apa yang dapat diakses dan dilakukan oleh pengguna perangkat lunak yang secara signifikan dapat mengurangi faktor-faktor yang menguntungkan penyerang;
4) Ketika kerugian terlalu besar, maka terapkan prinsip-prinsip desain, desain arsitektural, dan pelindungan teknis dan non-teknis untuk membatasi tingkat serangan, sehingga dapat mengurangi potensi kerugian.

c. Pendekatan pada Implementasi Kontrol

Pendekatan pada implementasi kontrol bertujuan untuk melakukan tindakan kontrol dengan langkah-langkah sebagai berikut:

1) Langkah 1 – Prioritaskan Tindakan
   Memprioritaskan tindakan implementasi berdasarkan tingkat risiko.
2) Langkah 2 – Mengevaluasi Opsi Kontrol yang Direkomendasikan
   Langkah ini bertujuan untuk memilih opsi kontrol yang paling tepat dalam meminimalkan risiko. Menganalisis kelayakan (misalnya, kompatibilitas, penerimaan pengguna) dan efektivitas (misalnya, tingkat pelindungan, dan tingkat mitigasi risiko) dari opsi kontrol yang direkomendasikan.
3) Langkah 3 – Lakukan Analisis Biaya dan Manfaat
   Langkah ini bertujuan untuk membantu pihak manajemen dalam pengambilan keputusan dan mengidentifikasi kontrol yang hemat biaya, maka harus dilakukan analisis biaya dan manfaat.
4) Langkah 4 – Pemilihan Kontrol
   Langkah ini bertujuan untuk menentukan kontrol yang paling hemat biaya untuk mengurangi risiko terhadap misi organisasi.
5) Langkah 5 – Menetapkan Tanggung Jawab
   Langkah ini bertujuan untuk mengidentifikasi dan menugaskan orang yang tepat (staf internal atau staf kontraktor eksternal) yang memiliki keahlian dan keterampilan yang sesuai untuk menerapkan kontrol yang dipilih.
6) Langkah 6 – Membuat Rencana Implementasi Pengamanan
   Langkah ini bertujuan untuk membuat rencana implementasi pengamanan (atau rencana aksi).
7) Langkah 7 – Menerapkan Kontrol Terpilih
   Langkah ini bertujuan untuk menerapkan kontrol yang dapat menurunkan tingkat risiko tetapi tidak menghilangkan risiko (risiko residual).

d. Kategori Kontrol

Pengkategorian kontrol bertujuan untuk mempertimbangkan kontrol keamanan teknis, manajemen, dan operasional, atau kombinasi dari kontrol tersebut, serta untuk memaksimalkan efektivitas kontrol untuk perangkat lunak dan organisasi.

1) Kontrol keamanan teknis untuk mitigasi risiko dapat dikonfigurasi untuk melindungi dari jenis ancaman tertentu. Kontrol ini dapat berkisar dari tindakan sederhana hingga kompleks dan biasanya melibatkan arsitektur perangkat lunak, ilmu teknik, dan sistem keamanan yang merupakan campuran perangkat keras, perangkat lunak, dan *firmware*.
2) Kontrol keamanan manajemen, bersama dengan kontrol teknis dan operasional, diterapkan untuk mengelola dan mengurangi risiko kerugian dan untuk melindungi misi organisasi.
3) Kontrol manajemen fokus pada penetapan kebijakan, pedoman, dan standar perlindungan informasi, yang dilakukan melalui prosedur operasional untuk memenuhi tujuan dan misi organisasi.

e. Analisis Biaya dan Manfaat

Analisis biaya dan manfaat bertujuan untuk mengalokasikan sumber daya dan menerapkan kontrol yang hemat biaya pada organisasi, setelah mengidentifikasi semua kemungkinan kontrol dan mengevaluasi kelayakan dan keefektifannya. Analisis biaya dan manfaat harus dilakukan untuk setiap kontrol yang diusulkan untuk menentukan kontrol mana yang diperlukan dan sesuai untuk keadaan organisasi.

f. Risiko Residual

Analisis risiko residual bertujuan untuk mengetahui sejauh mana pengurangan risiko yang dihasilkan oleh kontrol baru atau kontrol yang ditingkatkan dalam hal pengurangan tingkat kemungkinan atau tingkat dampak dari sumber ancaman.

1.4.3 Evaluasi dan Penilaian Risiko

Tahap ini bertujuan untuk menekankan praktik yang baik dan perlunya evaluasi dan penilaian risiko yang berkelanjutan. Proses evaluasi dan penilaian harus dilakukan secara berkala, sehingga perlu ada jadwal khusus. Namun demikian, juga harus cukup fleksibel untuk memungkinkan perubahan jika diperlukan, seperti perubahan besar pada perangkat lunak dan lingkungan pemrosesan karena perubahan yang dihasilkan dari kebijakan dan teknologi baru.

2. Desain keamanan

Tahap ini bertujuan untuk:

a. Merancang arsitektur yang aman dengan mempertimbangkan elemen keamanan;
b. Mengamankan arsitektur perangkat lunak dengan memahami dan menganalisis ancaman terhadap perangkat lunak sebelum dibangun.

2.1 Pertimbangan Desain *Core Security*

Tahap ini bertujuan untuk merancang perangkat lunak untuk mengatasi elemen *core security* dari aspek kerahasiaan, integritas, ketersediaan, autentikasi, otorisasi, dan audit.

2.1.1 Desain Kerahasiaan

Desain kerahasiaan bertujuan untuk memastikan bahwa pelindungan terhadap pengungkapan informasi dapat dicapai, yaitu menggunakan teknik kriptografi dan penyamaran (*masking*). Penyamaran (*masking*) berguna untuk memberi pelindungan dari pengungkapan informasi saat data ditampilkan di layar atau dicetak. Sedangkan Kriptografi digunakan untuk memberikan jaminan kerahasiaan pada saat data ditransmisikan atau disimpan di tempar penyimpanan data transaksional atau diarsipkan secara *offline*. Faktor yang mempengaruhi pelindungan kriptografi antara lain:

a. Algoritma enkripsi, yaitu proses pengubahan karakter informasi dari teks yang dapat dibaca (*plaintext*) menjadi teks yang tidak dapat dibaca (*ciphertext*).
b. Ukuran kunci (panjang kunci), yaitu adalah panjang kunci yang digunakan dalam algoritma.
c. Manajemen kunci, yaitu pengelolaan kunci yang meliputi tahap pembangkitan, pertukaran, penyimpanan, rotasi, pengarsipan, dan pemusnahan.

Algoritma enkripsi pada kriptografi terutama terdiri dari 2 (dua) jenis:

a. Algoritma simetris, yaitu algoritma enkripsi yang menggunakan 1 (satu) kunci untuk operasi enkripsi dan dekripsi yang digunakan bersama antara pengirim dan penerima.
b. Algoritma asimetris, yaitu algoritma enkripsi yang menggunakan 2 (dua) kunci, yaitu kunci yang dirahasiakan disebut sebagai kunci *private*, dan kunci yang diungkapkan kepada siapa saja yang perlu melakukan komunikasi aman dan ditampilkan secara publik disebut sebagai kunci *public*. Algoritma asimetris digunakan pada:
    1) Sertifikat digital, yaitu dokumen elektronik yang menentukan format untuk kunci *public* dan digunakan oleh siapa saja untuk memverifikasi keaslian sertifikat digital itu sendiri karena di dalamnya terdapat sertifikat digital dari *Certificate Authority* (CA).
    2) Tanda tangan digital, yaitu skema matematis yang digunakan untuk memberikan verifikasi identitas dan memastikan bahwa data atau pesan tidak dirusak karena tanda tangan digital yang digunakan untuk menandatangani pesan tidak dapat dengan mudah ditiru kecuali jika dikompromikan. Tanda tangan digital juga menyediakan bukti untuk nir-penyangkalan.

2.1.2 Desain Integritas

Desain integritas bertujuan untuk memastikan bahwa tidak ada modifikasi perangkat lunak atau data yang tidak sah dengan menggunakan salah satu dari teknik berikut atau kombinasi dari teknik-teknik berikut:

a. *Hashing* (Fungsi *hash*), yaitu fungsi yang digunakan untuk memadatkan *input* dengan panjang variabel menjadi *output* berukuran tetap yang tidak dapat diubah yang dikenal sebagai intisari (*digest*) pesan atau nilai *hash*.

b. Integritas referensi, yaitu jaminan integritas data, terutama dalam *relational database management system* (RDBMS) yang memastikan bahwa data tidak dibiarkan dalam keadaan tanpa relasi. Menggunakan kunci *primary* dan kunci *foreign* terkait dalam *database* untuk memastikan integritas data.
c. Penguncian sumber daya, yaitu ketika 2 (dua) operasi bersamaan tidak diizinkan pada objek yang sama (misalnya pada *record* di *database*), karena salah satu operasi mengunci *record* itu agar tidak mengizinkan perubahan apa pun hingga selesai operasinya.

2.1.3 Desain Ketersediaan

Desain ketersediaan bertujuan untuk mendapatkan pelindungan terhadap upaya penghancuran dan *Denial of Service* (DoS) dengan pembangunan perangkat lunak yang tepat. Untuk mendukung ketersediaan, pengkodean perangkat lunak juga harus mempertimbangkan persyaratan konfigurasi seperti penggabungan koneksi, penggunaan kursor, dan konstruksi perulangan. Teknik yang digunakan untuk merancang perangkat lunak untuk mendukung aspek ketersediaan antara lain:

a. Replikasi, yaitu penyediaan infrastruktur jika terjadi satu titik kegagalan (*single point of failure*) pada kondisi tidak adanya kemampuan redundansi. Namun jika kegagalan terjadi, maka akan terdapat gangguan operasional yang merugikan pengguna.
b. *Failover*, yaitu penyediaan infrastruktur dengan peralihan otomatis dari perangkat lunak transaksional yang aktif/*server*/sistem/komponen perangkat keras/jaringan ke sistem yang disiagakan (redundan). Namun, peralihan bersifat manual.
c. Skalabilitas, yaitu penyediaan infrastruktur pada peningkatan beban pekerjaan sistem tanpa penurunan fungsi atau kinerjanya.

2.1.4 Desain Autentikasi

Desain autentikasi bertujuan untuk menentukan jenis autentikasi yang diperlukan seperti yang ditentukan dalam dokumentasi persyaratan, seperti menggunakan autentikasi multi-faktor (MFA) dan *single sign-on* (SSO). Direkomendasikan menggunakan autentikasi multi-faktor atau autentikasi dua faktor (2FA) untuk mengautentikasi prinsipal (pengguna atau sumber daya) yang memberikan keamanan yang lebih tinggi. Jika ada kebutuhan untuk menerapkan SSO, dimana identitas prinsipal yang dinyatakan diverifikasi sekali dan kredensial yang diverifikasi diteruskan ke sistem atau aplikasi lain, maka perlu menggunakan token. Sangat penting untuk mempertimbangkan desain perangkat lunak baik dampak kinerja maupun keamanannya.

2.1.5 Desain Otorisasi

Desain otorisasi bertujuan untuk memberikan perhatian pada dampak kinerja, dan prinsip pemisahan tugas dan hak istimewa terkecil (*least privilege*). Jenis otorisasi yang akan diterapkan sesuai dengan persyaratan harus ditentukan, seperti peran atau otorisasi berbasis sumber daya. Jika peran digunakan untuk otorisasi, maka harus dipastikan bahwa tidak ada peran yang bertentangan karena belum diterapkannya prinsip pemisahan tugas. Misalnya, pengguna tidak dapat berperan sebagai *teller* dan juga sebagai auditor untuk

transaksi keuangan. Merancang untuk otorisasi dapat dilakukan dengan menggunakan manajemen hak, yaitu tentang kontrol akses granular (spesifik dan presisi untuk setiap pengguna).

Mekanisme kontrol akses yang sesuai untuk objek yang bersifat umum antara lain:

a. Direktori

Salah satu cara sederhana untuk melindungi objek adalah dengan menggunakan mekanisme yang berfungsi seperti direktori *file*. Hal ini lebih mudah daripada melindungi *file* (kumpulan objek) dari pengguna sistem komputasi (kumpulan subjek) dimana setiap *file* memiliki pemilik unik yang memiliki hak akses (hak yang menyatakan siapa yang memiliki akses apa) dan hak untuk mencabut akses ke siapa pun kapan saja. Lebih mudah untuk mengelola hak akses pengguna pada direktori *file* dimana tercantum semua *file* yang dapat diakses oleh pengguna tersebut.

b. *Access Control List* (ACL)

Ada 1 (satu) daftar untuk setiap objek, dan daftar tersebut menunjukkan semua subjek yang memiliki akses ke objek dan apa saja hak aksesnya. Pendekatan ini berbeda dari daftar direktori karena terdapat satu daftar kontrol akses per objek, sementara direktori dibuat untuk setiap subjek.

c. Matriks Kontrol Akses

Matriks kontrol akses merupakan tabel dimana setiap baris mewakili subjek, setiap kolom mewakili objek, dan setiap entri adalah himpunan hak akses untuk subjek ke objek itu.

d. Kapabilitas

Kapabilitas dalam hal ini merupakan token yang tidak dapat dipalsukan, yang memberikan hak tertentu atas suatu objek. Secara teori, subjek dapat membuat objek baru dan dapat menentukan operasi yang diperbolehkan pada objek tersebut. Misalnya, pengguna dapat membuat objek, seperti file, segmen data, atau subproses, dan dapat menentukan jenis operasi yang dapat diterima, seperti *read*, *write*, dan *execute*. Tetapi pengguna juga dapat membuat objek yang benar-benar baru, seperti struktur data baru, dan dapat menentukan jenis akses yang sebelumnya tidak diketahui oleh perangkat lunak.

e. Kontrol Akses Berorientasi Prosedur

Merupakan keberadaan prosedur yang mengontrol akses ke objek (misalnya, dengan melakukan autentikasi penggunanya sendiri untuk memperkuat perlindungan dasar yang disediakan oleh sistem operasi dasar). Prosedur membentuk kapsul di sekitar objek dan hanya mengizinkan akses tertentu yang ditentukan.

2.1.6 Desain Akuntabilitas

Desain akuntabilitas bertujuan untuk mengaudit perangkat lunak terutama jika terjadi pelanggaran, terutama untuk tujuan forensik digital. Data *log* harus mencakup aspek *who, what, where* dan *when* pada operasi di perangkat lunak. Sebagai bagian dari *who,* penting untuk tidak melupakan aktor non-manusia seperti proses *batch*, layanan, atau *daemon*.

2.2 Pertimbangan Desain Tambahan

Tahap ini bertujuan untuk merancang perangkat lunak yang membahas pertimbangan desain lain yang diperlukan saat membangun perangkat lunak yang aman.

2.2.1 Bahasa Pemrograman

Bahasa pemrograman yang akan digunakan harus ditentukan untuk mengimplementasikan desain, yang membawa risiko atau manfaat keamanan yang melekat. Ada dua (2) jenis utama bahasa pemrograman:

a. Kode Tidak Terkelola

Misalnya, C/C++. Pada bahasa pemrograman ini, eksekusi kode tidak dikelola oleh lingkungan eksekusi *runtime* mana pun tetapi langsung dieksekusi oleh sistem operasi.

b. Kode terkelola

Misalnya Java dan .NET, C#, VB.Net. Pada bahasa pemrograman ini, eksekusi kode tidak dilakukan oleh sistem operasi secara langsung, melainkan dikelola oleh lingkungan *runtime*. Layanan keamanan dan non-keamanan seperti manajemen memori, *exception handling*, *bound checking*, *garbage collection*, dan pemeriksaan keamanan yang diberikan oleh lingkungan *runtime*, dan pemeriksaan keamanan yang dilakukan sebelum kode dijalankan.

2.2.2 Jenis, Format, Jangkauan, dan Panjang Data

Pertimbangan desain untuk memastikan integritas berkaitan dengan tipe data, format, rentang, dan panjang:

a. Tipe Data Primitif atau *Built-In*

Seperti *character*, *integer*, *floating-point number*, dan *boolean*.

b. Tipe Data yang Ditentukan oleh Pemrogram

Namun hal ini tidak direkomendasikan dari sudut pandang keamanan karena berpotensi meningkatkan kemungkinan serangan.

c. Nilai dan Operasi yang Diizinkan

Nilai dan operasi yang diizinkan pada kumpulan nilai, yang ditentukan oleh tipe data.

d. Ketidakcocokan dan kesalahan konversi

Ketidakcocokan dan kesalahan konversi yang dapat merusak status keamanan perangkat lunak, antara lain:

1) Konversi melebar (ekspansi), yaitu tipe data dikonversi dari rentang ukuran yang lebih kecil ke rentang ukuran yang lebih besar. Potensi kerugian pada data adalah hilangnya presisi.
2) Konversi menyempit (pemotongan), yaitu tipe data dikonversi dari rentang ukuran yang lebih besar ke rentang ukuran yang lebih kecil. Potensi kerugian pada data adalah kehilangan data jika nilai yang disimpan dalam tipe data baru lebih besar dari kisaran yang diizinkan.

2.2.3 Keamanan *Database*

Keamanan *database* bertujuan untuk memastikan keandalan, ketahanan, dan pemulihan perangkat lunak yang bergantung pada data yang disimpan dalam *database*. Dampak desain keamanan *database* adalah:

a. Serangan inferensi, dimana penyerang memperoleh informasi sensitif tentang *database* dari potongan informasi yang mungkin tersembunyi dan sepele (diperoleh secara sah oleh penyerang) menggunakan teknik *data mining* tanpa mengakses *database* secara langsung.
b. Serangan agregasi, dimana informasi pada tingkat klasifikasi keamanan yang berbeda, yang pada dasarnya tidak sensitif, akhirnya menjadi informasi sensitif ketika informasi disatukan secara keseluruhan.

Pertimbangan desain perlindungan terkait aset *database* adalah:

a. *Polyinstantiation*

Polyinstantiation menyediakan beberapa *instance* (atau versi) dari informasi *database*, sehingga apa yang dilihat oleh pengguna bergantung pada izin keamanan (*security clearance*) atau atribut tingkat klasifikasi dari pengguna yang mengakses. Ini adalah pendekatan keamanan *database* untuk menangani masalah inferensi dengan menyembunyikan informasi menggunakan label klasifikasi, dan agregasi dengan memberi label agregasi data yang berbeda secara terpisah.

b. Enkripsi *database*

Enkripsi *database* menyediakan enkripsi *data-at-rest* sebagai mekanisme pencegahan dan kontrol yang dapat memberikan pelindungan yang kuat terhadap pengungkapan dan pengubahan data. Selain enkripsi *database*, juga diperlukan autentikasi yang tepat dan mekanisme perlindungan kontrol akses untuk mengamankan kunci yang digunakan pada enkripsi *database*.

c. Normalisasi

Normalisasi yaitu teknik formal yang digunakan untuk mengatur data sehingga redundansi dan inkonsistensi dapat dihilangkan.

d. *Trigger* dan *view*

Trigger akan dipicu untuk dijalankan secara implisit pada *database* ketika peristiwa *trigger* terjadi. *Trigger* juga sangat berguna untuk mengotomatisasi dan meningkatkan mekanisme pelindungan keamanan. Sedangkan *view* adalah *output* dari *query* dan serupa dengan tabel virtual atau *query* tersimpan. *View* dibangun secara dinamis yang menyebabkan data yang disajikan dapat dibuat khusus untuk pengguna berdasarkan hak dan keistimewaannya.

2.2.4 Desain Antarmuka

Pertimbangan desain antarmuka yang dapat diterapkan saat membangun perangkat lunak, antara lain:

a. Antarmuka Pengguna (*User Interface*/UI)

UI ini harus mendukung model keamanan dan bertindak sebagai program perantara. Desain UI harus menjamin pelindungan dari pengungkapan informasi. Penyamaran

informasi sensitif, seperti *password* atau nomor kartu kredit dengan menampilkan tanda bintang di layar, adalah contoh UI yang aman, yang menjamin aspek kerahasiaan. Tampilan *database* juga dapat dikatakan sebagai contoh UI yang harus dibatasi.

b. Antarmuka Pemrograman Aplikasi (*Application Programming Interface/*API)

API digunakan untuk berkomunikasi dari satu komponen perangkat lunak dengan yang lain atau agar perangkat lunak berinteraksi dengan sistem operasi yang mendasarinya. API biasanya tersedia di *library*, dan mungkin termasuk spesifikasi untuk rutinitas, struktur data, kelas objek, dan variabel.

c. Antarmuka Manajemen Keamanan (*Security Management Interface/*SMI)

SMI digunakan untuk mengonfigurasi dan mengelola keamanan perangkat lunak itu sendiri, dan merupakan antarmuka administratif dengan hak istimewa tertinggi. SMI digunakan untuk tugas penyediaan pengguna seperti menambahkan pengguna, menghapus pengguna, mengaktifkan atau menonaktifkan akun pengguna, serta memberikan hak dan hak istimewa untuk peran, mengubah pengaturan keamanan, mengonfigurasi pengaturan *log* audit, dan jejak audit (*audit trail*), *exception log*, dan lainnya.

d. Antarmuka *Out-of-Band*

Antarmuka *out-of-band* memungkinkan administrator untuk terhubung ke komputer yang dalam keadaan tidak aktif atau mati.

e. Antarmuka *Log*

Antarmuka *log* untuk masuk atau keluar di lingkungan yang berbeda (pengembangan, pengujian, produksi), yang merupakan komponen penting dalam audit dan saat merancang perangkat lunak untuk audit.

2.2.5 Interkonektivitas

Desain interkonektivitas bertujuan untuk secara eksplisit merancang kompatibilitas perangkat lunak hulu dan hilir yang penting dalam hal pendelegasian kepercayaan, *single sign-on* (SSO), autentikasi berbasis token, dan pembagian kunci kriptografis antar aplikasi. Kompatibilitas perangkat lunak hulu dan hilir harus secara eksplisit dirancang.

Di sebagian besar aplikasi *mobile*, pelindungan data pada klien diserahkan kepada aplikasi itu sendiri, sehingga aplikasi harus dirancang untuk menghindari penyimpanan informasi sensitif apa pun di lingkungan *sandbox* aplikasi itu sendiri. *Publisher* dan API pihak ketiga yang menyediakan layanan kriptografi (enkripsi dan dekripsi) mungkin harus dipertimbangkan untuk memastikan aspek kerahasiaan.

Saat data disimpan di suatu lokasi di jaringan, *network-attached storage* (NAS) juga harus dilindungi. Harus dirancang agar NAS hanya dapat diakses oleh koneksi yang telah diautentikasi dan diotorisasi, dibatasi oleh rentang IP tertentu, dan jika perlu telah mengakomodir pelindungan terhadap ancaman IP *spoofing*.

2.3 Pemodelan Ancaman

Tahap ini bertujuan untuk mengidentifikasi, mengukur, dan mengatasi risiko keamanan yang terkait dengan perangkat lunak.

2.3.1 Langkah 1 - Dekomposisi Perangkat Lunak

Dekomposisi perangkat lunak bertujuan untuk mendapatkan pemahaman tentang perangkat lunak dan bagaimana perangkat lunak berinteraksi dengan entitas eksternal. Hal ini melibatkan pembuatan *use case* untuk memahami bagaimana aplikasi digunakan, mengidentifikasi titik masuk untuk melihat di mana penyerang potensial dapat berinteraksi dengan aplikasi, mengidentifikasi aset (item/area penyerang akan tertarik untuk menyerang) dan mengidentifikasi tingkat kepercayaan yang mewakili hak akses yang akan diberikan oleh perangkat lunak kepada entitas eksternal. Hal-hal yang perlu dipertimbangkan saat mendekomposisi perangkat lunak meliputi:

a. Ketergantungan eksternal;
b. Titik masuk (vektor serangan);
c. Aset;
d. Menentukan *attack surface* dengan menganalisis *input* (*browser input*, *cookie*, *file* properti, proses eksternal, umpan data, respons layanan, *flat file*, parameter *command line*, variabel lingkungan), *data flow*, dan transaksi;
e. Tingkat kepercayaan;
f. Analisis aliran data;
g. Analisis transaksi (area yang dicakup adalah validasi data/masukan data dari semua sumber yang tidak dipercaya, autentikasi, manajemen sesi, otorisasi, kriptografi pada *data at rest* dan *data in transit*, *error handling*/kebocoran informasi, *logging/auditing*)
h. Diagram *data flow*.

2.3.2 Langkah 2 - Menentukan dan Mengurutkan Ancaman

Tahap ini bertujuan untuk mengidentifikasi ancaman menggunakan metodologi kategorisasi ancaman. Contoh kategorisasi ancaman seperti *Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege* (STRIDE) dapat digunakan untuk mendefinisikan kategori ancaman sehubungan dengan tujuan penyerang, seperti audit dan *logging*, autentikasi, otorisasi, manajemen konfigurasi, pelindungan *data in storage* dan *data in transit*, validasi data, dan manajemen *exception*.

Untuk menentukan peringkat ancaman dapat menggunakan model peringkat ancaman-risiko *Damage, Reproducibility, Exploitability, Affected Users, and Discoverability* (DREAD). Metodologi ini akan menentukan peringkat risiko ancaman dengan menghitung rata-rata nilai numerik yang ditetapkan untuk kategori peringkat risiko.

2.3.3 Langkah 3 - Menentukan Penanggulangan dan Mitigasi

Tahap ini bertujuan untuk mengurangi kerentanan dengan penerapan penanggulangan sebagai pelindungan terhadap ancaman.

Penanggulangan tersebut dapat diidentifikasi menggunakan daftar pemetaan ancaman-penanggulangan. Setelah peringkat risiko ditetapkan untuk ancaman, maka ancaman dapat diurutkan dari risiko tertinggi ke risiko terendah, dan memprioritaskan upaya mitigasi, seperti menanggapi ancaman tersebut dengan menerapkan tindakan pencegahan yang teridentifikasi.

3. Pengembangan Keamanan

Tahap ini bertujuan untuk:

a. Menerapkan aspek teknologi dan proses pada penulisan kode yang aman.
b. Memastikan kontrol keamanan yang ditentukan dalam persyaratan keamanan telah diterapkan di dalam kode.
c. Memastikan ancaman yang teridentifikasi dari pemodelan ancaman telah dihindari selama pengkodean.

3.1 Kerentanan dan Kontrol Umum pada Perangkat Lunak

Tahap ini bertujuan untuk:

a. Untuk mengidentifikasi kerentanan paling umum yang dihasilkan dari pengkodean yang tidak aman.
b. Untuk menerapkan kontrol keamanan di dalam kode untuk melawan dan menggagalkan tindakan dari agen ancaman.

3.1.1 *Database* Kerentanan

Tahap ini bertujuan untuk menemukan *database* kerentanan atau repositori dari kerentanan yang diketahui yang telah ditemukan pada perangkat lunak yang diimplementasikan (misalnya, cacat/*flaw* dan *bug*). Kerentanan dan risiko keamanan perangkat lunak yang paling umum harus dicantumkan dalam *checklist secure* SDLC untuk memudahkan pengembang selama melakukan pengembangan. Namun, organisasi dapat mengacu pada daftar kerentanan perangkat lunak terkini yang ditemukan di seluruh industri pengembangan perangkat lunak, seperti OWASP *Top* 10 *Project* atau *Common Weakness Enumeration* (CWE/25) yang merupakan proyek yang dikelola oleh MITRE dan SANS Institute.

3.1.2 Praktek Pengkodean Defensif

Tahap ini bertujuan untuk mengenali, mengevaluasi, dan mengurangi permukaan serangan (*attack surface*) dari kode perangkat lunak. Hal ini terjadi karena permukaan serangan berpotensi meningkat setiap kali 1 (satu) baris kode ditulis. Beberapa contoh pengurangan permukaan serangan yang terkait dengan kode adalah:

a. Mengurangi jumlah kode dan layanan yang dijalankan secara *default*;
b. Mengurangi volume kode yang dapat diakses oleh pengguna yang tidak berhak;
c. Membatasi kerusakan ketika kode dieksploitasi.

Praktik dan teknik pengkodean defensif yang paling umum harus dicantumkan dalam *checklist secure* SDLC untuk memudahkan pengembang selama melakukan pengembangan.

3.2 Proses Perangkat Lunak yang Aman

Tahap ini bertujuan untuk:

Menjamin keamanan perangkat lunak dengan melakukan proses-proses tertentu sebagai tambahan penulisan kode keamanan.

3.2.1 Versi pada Kode Sumber

Tahap ini bertujuan untuk memastikan bahwa tim pengembangan bekerja dengan versi kode yang benar. Versi pada kode sumber memberikan kemampuan untuk:

a. Memutar kembali ke versi sebelumnya jika diperlukan;
b. Melacak kepemilikan dan perubahan kode;
c. Mendukung pembaruan versi.

3.2.2 Analisis Kode

Tahap ini bertujuan untuk melakukan proses otomatis pemeriksaan kode untuk kualitas dan kelemahan yang dapat dieksploitasi. Analisis kode dapat dilakukan dengan 2 (dua cara):

a. Analisis Kode Statis

Analisis kode statis melibatkan pemeriksaan kode tanpa menjalankan kode atau program perangkat lunak. Analisis ini biasanya dilakukan oleh pihak ketiga.

b. Analisis Kode Dinamis

Analisis kode dinamis adalah pemeriksaan kode saat dijalankan sebagai program. Analisis ini biasanya dilakukan oleh pengembang.

3.2.3 Reviu Kode

Tahap ini bertujuan untuk melakukan evaluasi kode sumber secara sistematis dan manual dengan tujuan menemukan masalah sintaks dan kelemahan kode yang dapat memengaruhi kinerja dan keamanan perangkat lunak. Reviu kode biasanya dilakukan oleh pengembang. Masalah semantik seperti logika bisnis dan cacat desain biasanya tidak terdeteksi dalam reviu kode, tetapi reviu kode dapat digunakan untuk memvalidasi model ancaman yang dihasilkan pada tahap desain.

3.2.4 Pengujian Pengembang

Pengujian yang dilakukan oleh pengembang ini menggunakan alat dan teknik pengujian yang sistematis untuk membuat perangkat lunak yang diuji dapat dipelihara dengan cacat sesedikit mungkin. Pengujian pengembang memerlukan pendekatan terstruktur dan penguasaan beberapa kompetensi inti, dimana memahami penggerak pengujian, teknik pengujian mendasar, dan pengujian unit adalah hal yang paling penting. Jenis pengujian umum yang dapat ditulis pengembang untuk aplikasi adalah:

a. Pengujian Unit (*Unit Test*)

Pengujian unit bertujuan untuk melakukan eksekusi bagian kode atau program kecil yang diuji secara terisolasi dari perangkat lunak lengkap. Diuji secara terisolasi berarti tidak memanggil implementasi kode yang tidak diuji, misalnya *database*, layanan *web*, atau dependensi kode lainnya. Pengembang akan melakukan pengujian ini selama pengembangan fitur.

b. Pengujian Integrasi

Pengujian integrasi bertujuan untuk melakukan eksekusi gabungan dari kode. Dalam jenis pengujian ini, akan dilakukan pemanggilan ke *database*, layanan *web*, atau dependensi kode lainnya. Pengembang akan melakukan pengujian ini selama pengembangan fitur.

c. Pengujian Regresi

Pengujian regresi bertujuan untuk melakukan pengulangan kasus uji yang dijalankan sebelumnya untuk tujuan menemukan cacat pada perangkat lunak yang sebelumnya telah melewati rangkaian pengujian yang sama. Pengujian ini biasanya digunakan sebelum mengimplementasikan kode ke lingkungan baru atau sebagai bagian dari proses pembuatan. Pengujian ini bisa menggunakan alat untuk otomatisasi atau manusia secara langsung.

d. Pengujian Perangkat Lunak

Pengujian perangkat lunak bertujuan untuk melakukan eksekusi perangkat lunak dalam konfigurasi akhirnya, termasuk integrasi dengan perangkat lunak dan sistem lainnya. Pengujian ini akan menguji keamanan, kinerja, kehilangan sumber daya, permasalahan waktu, dan masalah lain yang tidak dapat diuji pada tingkat integrasi yang lebih rendah. Seperti pengujian regresi, pengujian ini bisa menggunakan alat untuk otomatisasi atau manusia secara langsung.

3.3 Mengamankan Lingkungan Pengembangan

Tahap ini bertujuan untuk:

a. Melindungi kode sumber agar tidak diakses oleh orang yang tidak terkait dengan proyek selama fase pengembangan;
b. Memastikan integritas kode sumber yang dikembangkan.

3.3.1 Mengamankan Akses Fisik ke Perangkat Lunak yang Membangun Kode

3.3.2 Menggunakan *Access Control List* (ACL)

Access Control List (ACL) dapat mencegah akses pengguna yang tidak sah.

3.3.3 Menggunakan Perangkat Lunak Kontrol Versi

Penggunaan perangkat lunak kontrol versi bertujuan untuk memastikan bahwa kode yang dibangun adalah versi yang benar.

3.3.4 *Build Automation*

Build automation adalah proses pembuatan *script* atau mengotomatiskan tugas-tugas yang terlibat dalam proses *build,* namun dengan tetap membutuhkan aktivitas manual oleh tim pengembang. Beberapa aktivitas pada tahap otomatisasi *build* ini termasuk mengompilasi kode sumber menjadi kode mesin, dependensi *package*, penerapan, dan penginstalan. Saat *script* digunakan untuk membangun proses otomatisasi *build*, penting untuk memastikan bahwa kontrol dan pemeriksaan keamanan tidak dilewatkan.

3.3.5 Penandatanganan Kode (*Code Signing*)

Penandatanganan kode bertujuan untuk memastikan integritas kode sumber yang sedang dikembangkan.

4. Pengujian Keamanan

Tahap ini bertujuan untuk:

a. Memvalidasi dan memverifikasi fungsionalitas dan keamanan perangkat lunak menggunakan pengujian jaminan kualitas (*quality assurance*);
b. Memastikan kode yang dikembangkan dapat berjalan sesuai harapan.

4.1 Validasi Permukaan Serangan (*Attack Surface*)

Tahap ini bertujuan untuk memverifikasi keberadaan dan keefektifan kontrol keamanan yang dirancang dan diimplementasikan dalam perangkat lunak.

4.1.1 Pengujian Pasca Pengembangan

Untuk memastikan kebenaran perangkat lunak yang dikembangkan, maka langkah ini akan mengeksekusi analisis kode, khususnya *dynamic code analysis*, yaitu pemeriksaan kode pada saat program dijalankan (*run as a program*).

4.1.2 Pengujian Keamanan Menggunakan Metode Pengujian Keamanan

Metode yang digunakan untuk melakukan pengujian keamanan antara lain:

a. Pengujian *White Box*

Pengujian *white box* bertujuan untuk menguji struktur perangkat lunak berdasarkan pengetahuan tentang bagaimana perangkat lunak dirancang dan diimplementasikan. Pengujian ini secara luas dikenal sebagai *full knowledge assessment* karena penguji memiliki pengetahuan lengkap tentang perangkat lunak. Pengujian ini dapat digunakan untuk menguji *use case* (perilaku yang diinginkan) serta *misuse case* (perilaku yang tidak diinginkan) dari perangkat lunak dan dapat dilakukan kapan saja setelah pengembangan kode, meskipun disarankan untuk melakukan pengujian ini saat melakukan pengujian unit. Pengujian ini akan menganalisis keamanan secara metodologis dan struktural mengenai *data/information flow*, *control flow*, antarmuka, *trust boundary* (titik masuk dan keluar), konfigurasi, *error handling* dan lainnya.

b. Pengujian *Black Box*

Pengujian *black box* bertujuan untuk menguji perilaku perangkat lunak, juga dikenal sebagai *zero knowledge assessment* karena penguji memiliki pengetahuan yang sangat terbatas tentang perangkat lunak yang diuji. Tim penguji tidak mengetahui sama sekali mengenai dokumen arsitektur atau desain, informasi atau *file* konfigurasi, *use case*, *misuse case,* dan kode sumber perangkat lunak.

Pengujian *black box* dilakukan dengan menggunakan alat yang berbeda. Metodologi umum dimana pengujian *black box* dapat diselesaikan dengan alat adalah *fuzzing*, pemindaian, dan pengujian penetrasi.

c. Pengujian Validasi Kriptografi

Pengujian validasi kriptografi bertujuan untuk memastikan aplikasi yang menerapkan mekanisme kriptografi menggunakan algoritma kriptografi yang tepat dan praktik terbaik untuk manajemen kunci.

4.1.3 Melakukan Pengujian Keamanan Perangkat Lunak untuk Penjaminan Kualitas

Tahap ini bertujuan untuk menerapkan berbagai jenis pengujian dan bagaimana pengujian tersebut dapat dilakukan untuk membuktikan keamanan kode yang dikembangkan dalam fase pengembangan *secure* SDLC.

Pada revisi perangkat lunak, pengujian regresi harus dilakukan, dan untuk semua versi, baru atau revisi, pengujian keamanan harus dilakukan untuk memvalidasi kekuatan dari kontrol keamanan. Menggunakan daftar ancaman yang dikategorikan sebagai *template* pengujian keamanan efektif dalam memastikan cakupan komprehensif dari berbagai ancaman terhadap perangkat lunak. Idealnya, daftar ancaman yang sama yang digunakan saat memodelkan ancaman perangkat lunak akan menjadi daftar ancaman yang juga digunakan untuk melakukan pengujian keamanan. Dengan cara ini, pengujian keamanan dapat digunakan untuk memvalidasi model ancaman.

Pengujian keamanan, antara lain:

a. Pengujian validasi *input*;
b. Pengujian untuk kontrol cacat injeksi (*injection flaw*);
c. Pengujian untuk kontrol serangan *scripting* (*scripting attack*);
d. Pengujian untuk kontrol nir-penyangkalan (*non-repudiation*);
e. Pengujian untuk kontrol *spoofing*;
f. Pengujian untuk kontrol *error* dan *exception handling* (*failure testing*);
g. Pengujian untuk kontrol eskalasi hak istimewa (*privilege escalation*);
h. Pengujian untuk pelindungan anti-pembalikan (*anti-reversing*);
i. Pengujian ketahanan (*stress testing*).

4.2 Manajemen Data Pengujian

Tahap ini bertujuan untuk mengidentifikasi *input* data pengujian dan data yang diharapkan menjadi *output* setelah operasi normal dari perangkat lunak.

4.2.1 Mengidentifikasi *Output* Data Pengujian untuk Memastikan Persyaratan Perangkat Lunak

Mengidentifikasi *output* dari data pengujian yang diharapkan bertujuan untuk mengonfirmasi apakah perangkat lunak telah memenuhi persyaratan. Kualitas data pengujian berhubungan langsung dengan kualitas pengujian itu sendiri, sehingga data pengujian perlu dikelola.

Data pada tahap produksi tidak boleh diimpor dan diproses di lingkungan pengujian. Dianjurkan untuk menggunakan data tiruan dengan membuatnya dari awal di lingkungan pengujian atau simulasi.

4.2.2 Melakukan Pengujian dengan Transaksi Sintetis

Tahap ini bertujuan untuk melakukan transaksi dengan data *dummy* yang tidak terkait dengan proses bisnis sebenarnya. Transaksi sintetis bisa pasif atau aktif. Transaksi sintetis pasif tidak disimpan dan tidak memiliki dampak sisa pada perangkat lunak itu sendiri. Namun, jika transaksi dari pelanggan *dummy* diproses dan disimpan dalam aplikasi perangkat lunak, maka transaksi ini akan menjadi transaksi sintetis yang aktif. Penggunaan transaksi sintetis aktif mengharuskan pengembang untuk memperhatikan pengaturan data dan lingkungan sedemikian rupa sehingga tidak berdampak pada lingkungan produksi.

4.2.3 Solusi pada Manajemen Data Pengujian

Solusi pada manajemen data pengujian dapat digunakan untuk membantu dalam pembuatan *subset* data keseluruhan dari data produksi dan untuk mengurangi beberapa

kekhawatiran yang menyertai pembuatan data pengujian yang berkualitas dan pengelolaannya di lingkungan pengujian. Solusi tersebut secara otomatis menemukan hubungan data dengan menganalisis dan menangkap atribut tabel, serta memastikan bahwa aturan ekstraksi mempertimbangkan ruang penyimpanan yang tersedia di lingkungan pengujian, sehingga proses ekstraksi tidak berakhir dengan mengekstraksi sebagian besar data yang tidak dapat diimpor ke lingkungan pengujian, karena keterbatasan ukuran.

4.2.4 Pelaporan dan Pelacakan Cacat

Tahap ini bertujuan untuk melaporkan cacat (*defect/flaw*) dan kemudian melacak *bug* pengkodean, cacat desain (*design flaw*), anomali perilaku (*logic flaw*), *error*, *fault*, dan kerentanan pada perangkat lunak. Pelaporan cacat harus komprehensif dan cukup rinci untuk memberikan tim pengembangan perangkat lunak informasi yang diperlukan untuk menentukan akar penyebab masalah sehingga dapat segera mengatasinya.

5. Penerapan Keamanan

Tahap ini bertujuan untuk:

a. Memastikan penyelesaian dari rencana operasi dan dokumentasi aplikasi;
b. Memastikan persetujuan pihak manajemen dan penerimaan risiko untuk penerapan;
c. Memastikan aplikasi memenuhi fungsinya dan aman;
d. Memastikan lingkungan dan konfigurasi untuk penerapan yang aman.

5.1 Pertimbangan Penerimaan Perangkat Lunak

Tahap ini bertujuan untuk:

a. Memastikan dokumentasi aplikasi dan rencana operasi sudah siap;
b. Mendapatkan persetujuan dan penerimaan risiko oleh pihak manajemen.

5.1.1 Kriteria Penyelesaian

Tahap ini bertujuan untuk memvalidasi dan memverifikasi semua persyaratan fungsional dan keamanan sudah lengkap seperti yang diharapkan. Kriteria penyelesaian untuk fungsionalitas dan keamanan perangkat lunak dengan *milestone* yang jelas harus ditentukan dengan baik sebelumnya. Beberapa contoh *milestone* terkait keamanan termasuk, namun tidak terbatas pada:

a. Pembuatan persyaratan keamanan selain persyaratan fungsional dalam tahap persyaratan;
b. Penyelesaian model ancaman selama tahap desain;
c. Reviu dan persetujuan terhadap arsitektur keamanan pada akhir tahap desain;
d. Reviu kode untuk kerentanan keamanan setelah tahap pengembangan;
e. Penyelesaian pengujian keamanan pada akhir tahap pengujian aplikasi;
f. Penyelesaian dokumentasi sebelum tahap penerapan dimulai.

5.1.2 Manajemen Perubahan

Tahap ini bertujuan untuk memastikan proses sudah tepat untuk menangani permintaan perubahan. Manajemen perubahan adalah bagian dari manajemen konfigurasi. Perubahan pada lingkungan komputasi dan desain ulang arsitektur keamanan berpotensi menimbulkan kerentanan keamanan baru, sehingga dapat meningkatkan risiko. Proses dukungan yang diperlukan untuk perangkat lunak yang akan digunakan/dirilis harus ditetapkan.

5.1.3 Persetujuan untuk Menerapkan atau Merilis

Tahap ini bertujuan untuk memastikan semua pihak otoritas yang diperlukan untuk memberikan persetujuan. Tanpa persetujuan, maka tidak boleh ada perubahan yang diizinkan pada lingkungan komputasi produksi. Sebelum instalasi perangkat lunak baru, analisis risiko perlu dilakukan, dan risiko sisa ditentukan. Hasil analisis risiko, beserta langkah-langkah yang diambil untuk mengatasinya (mitigasi atau diterima), serta risiko residual harus dikomunikasikan kepada pemilik risiko. Persetujuan atau penolakan untuk menerapkan/merilis perangkat lunak harus menyertakan rekomendasi dan dukungan dari tim keamanan. Pada akhirnya, pihak yang berwenanglah yang akan bertanggung jawab atas persetujuan perubahan.

5.1.4 Kebijakan Penerimaan dan Pengecualian Risiko

Tahap ini bertujuan untuk memastikan risiko residual dapat diterima dan/atau dilacak sebagai pengecualian jika tidak berada dalam ambang batas (*threshold*). Risiko yang tersisa setelah penerapan kontrol keamanan (risiko residual) perlu ditentukan terlebih dahulu. Pilihan terbaik untuk mengatasi risiko total adalah memitigasinya sehingga risiko residual berada di bawah ambang batas yang ditentukan, dalam hal ini risiko residual dapat diterima. Risiko harus dapat diterima oleh pemilik risiko, bukan oleh pejabat di unit kerja yang membawahi fungsi TI.

5.1.5 Dokumentasi Perangkat Lunak

Tahap ini bertujuan untuk memastikan semua dokumentasi yang diperlukan sudah tersedia. Mendokumentasikan perangkat lunak sangat penting untuk penggunaan perangkat lunak yang efektif, aman, dan berkelanjutan. Dokumentasi perangkat lunak harus meliputi; perancangan, penginstalan, pengaturan konfigurasi, penggunaan, dan pengaturan. Beberapa tujuan utama dokumentasi adalah untuk membuat proses penerapan perangkat lunak menjadi mudah dan dapat berulang, serta untuk memastikan bahwa operasi tidak terganggu, dan dampak terhadap perubahan perangkat lunak dapat dipahami.

5.2 Verifikasi dan Validasi (V&V)

Tahap ini bertujuan untuk memastikan fungsionalitas aplikasi dan aman untuk lingkungan produksi.

5.2.1 Reviu

Tahap ini bertujuan untuk melakukan reviu pada akhir setiap tahap untuk memastikan bahwa perangkat lunak berfungsi seperti yang diharapkan dan memenuhi spesifikasi bisnis yang diharapkan. Hal ini dapat dilakukan secara informal maupun formal.

5.2.2 Pengujian

Tahap ini bertujuan untuk menunjukkan bahwa perangkat lunak sudah memenuhi persyaratan dan menentukan penyimpangan yang diharapkan dengan menggunakan hasil pengujian yang sebenarnya. Berbagai jenis pengujian yang dilakukan sebagai bagian dari V&V adalah:

a. Pengujian Deteksi Kesalahan (*Error Detection Test*)

Merupakan pengujian pada tingkat unit dan komponen. Kesalahan dapat berupa cacat (masalah desain) atau *bug* (masalah kode).

b. Pengujian Penerimaan (*Acceptance Test*)

Pengujian digunakan untuk menunjukkan apakah perangkat lunak siap untuk digunakan atau tidak. Perangkat lunak yang dianggap siap seharusnya tidak hanya divalidasi untuk semua persyaratan fungsional tetapi juga divalidasi untuk memastikan memenuhi persyaratan jaminan keamanan (*security assurance*).

c. Pengujian Pihak Independen (Pihak Ketiga)

Merupakan pengujian fungsionalitas dan jaminan perangkat lunak dimana perangkat lunak direviu, diverifikasi, dan divalidasi oleh orang lain selain pengembang perangkat lunak. Pengujian ini biasanya juga disebut sebagai verifikasi dan validasi pihak independen.

5.3 Sertifikasi dan Akreditasi (C&A)

Tahap ini bertujuan untuk:

a. Mendapatkan verifikasi teknis dari fungsionalitas aplikasi;
b. Mendapatkan penerimaan keseluruhan untuk penerapan ke dalam lingkungan produksi.

5.3.1 Mendapatkan Sertifikasi

Tahap ini bertujuan untuk mencapai sertifikasi untuk verifikasi teknis dari tingkat fungsional dan jaminan perangkat lunak. Sertifikasi keamanan mempertimbangkan perangkat lunak di lingkungan operasional. Sekurang-kurangnya, sertifikasi mencakup evaluasi penjaminan dari hal-hal berikut:

a. Hak pengguna, hak istimewa, dan manajemen profil;
b. Sensitivitas data dan aplikasi serta pengendalian yang sesuai;
c. Konfigurasi perangkat lunak, fasilitas, dan lokasi;
d. Interkonektivitas dan ketergantungan;
e. Mode keamanan operasional.

5.3.2 Mendapatkan Akreditasi

Untuk mendapatkan akreditasi atas penerimaan formal pihak manajemen, diperlukan verifikasi dari pengguna yang ditargetkan terhadap perangkat lunak setelah memahami

tingkat risiko terhadap perangkat lunak tersebut di lingkungan komputasi. Akreditasi ini adalah keputusan resmi pihak manajemen untuk mengoperasikan perangkat lunak di mode keamanan operasional untuk jangka waktu tertentu dan merupakan penerimaan formal dari risiko yang teridentifikasi terkait dengan pengoperasian perangkat lunak.

5.4 Instalasi

Tahap ini bertujuan untuk mengamankan lingkungan dan konfigurasi produksi aplikasi.

5.4.1 Penguatan (*Hardening*)

Tahap ini bertujuan untuk mengunci perangkat lunak ke tingkat yang paling ketat sehingga dalam kondisi aman. Tingkat keamanan minimum (atau paling ketat) ini biasanya dipublikasikan sebagai dasar yang harus dipatuhi oleh semua perangkat lunak dalam lingkungan komputasi. *Baseline* ini biasanya disebut sebagai *baseline* minimum yang dibuat berdasarkan penggunaan sistem operasi.

Penting untuk memperkuat sistem operasi *host* dengan menggunakan *baseline*, *update*, dan *patch*. Juga sangat penting untuk memperkuat aplikasi dan perangkat lunak yang berjalan di atas sistem operasi ini. Penguatan perangkat lunak melibatkan pengaturan konfigurasi yang diperlukan dan benar dan merancang perangkat lunak agar aman secara *default*.

5.4.2 Konfigurasi Lingkungan

Tahap ini bertujuan untuk memastikan bahwa parameter yang diperlukan untuk menjalankan perangkat lunak telah dikonfigurasi dengan tepat menggunakan *checklist* pra-instalasi. Namun demikian, masalah dinamis mungkin tidak selalu dapat diidentifikasi secara statis, maka *checklist* pra-instalasi tidak dapat memberikan jaminan bahwa perangkat lunak akan berfungsi tanpa melanggar prinsip keamanan yang dirancang dan dibuatnya. Oleh karena itu, sangat penting untuk memastikan bahwa lingkungan pengembangan dan pengujian cocok dengan susunan konfigurasi lingkungan produksi dan pengujian simulasi secara identik meniru konfigurasi lingkungan produksi dimana perangkat lunak akan diterapkan pasca penerimaan.

5.4.3 Manajemen Rilis

Tahap ini bertujuan untuk memastikan rilis perangkat lunak dengan benar ke dalam lingkungan komputasi operasi setelah sumber daya perangkat keras dan perangkat lunak diperkuat dan lingkungan dikonfigurasikan untuk operasi yang aman.

Manajemen rilis adalah proses untuk memastikan bahwa semua perubahan yang dilakukan pada lingkungan komputasi telah direncanakan, didokumentasikan, diuji secara menyeluruh, dan disebarkan dengan hak istimewa paling sedikit (*least privilege*) tanpa berdampak negatif terhadap operasi bisnis, pelanggan, pengguna akhir, atau tim dukungan pengguna yang ada.

5.4.4 *Bootstrap* dan *Startup* yang Aman

Tahap ini bertujuan untuk menentukan bahwa proses pengaktifan perangkat lunak sama sekali tidak berdampak negatif terhadap kerahasiaan, integritas, atau ketersediaan perangkat lunak setelah penginstalan perangkat lunak. *Power-On Self-Test* (POST) adalah

langkah pertama dalam *Initial Program Load* (IPL) dan merupakan kejadian yang perlu dilindungi agar tidak dirusak sehingga *Trusted Computing Base* (TCB) dapat dipelihara.

6. Pemeliharaan Keamanan

Tahap ini bertujuan untuk:

a. Memantau dan menjamin bahwa perangkat lunak akan terus berfungsi dengan cara yang andal, tangguh, dan dapat dipulihkan;
b. Mengidentifikasi perangkat lunak dan kondisi dimana perangkat lunak perlu dibuang atau diganti.

6.1 Operasi, Pemantauan dan Pemeliharaan

Tahap ini bertujuan untuk:

a. Memberikan layanan kepada bisnis atau pengguna akhir untuk kebutuhan pengoperasian dan pemeliharaan perangkat lunak.
b. Memastikan aspek penjaminan pada pemrosesan perangkat lunak yang andal, tangguh, dan dapat dipulihkan.

6.1.1 Melaksanakan Pengamanan Operasi

Tahap ini bertujuan untuk memastikan keamanan operasi dengan menjaga tetap aman atau menjaga tingkat ketahanan perangkat lunak di atas tingkat risiko yang dapat diterima. Tahap ini adalah penjaminan bahwa perangkat lunak akan terus berfungsi seperti yang diharapkan dengan cara yang andal untuk bisnis tanpa mengorbankan kondisi keamanannya dengan memantau, mengelola, dan menerapkan kontrol yang diperlukan untuk melindungi aset informasi. Berbagai jenis kontrol keamanan operasi, antara lain:

a. Kontrol detektif (*detective control*), yaitu kontrol yang dapat digunakan untuk membangun bukti historis dari tindakan pengguna dan proses pada perangkat lunak.
b. Kontrol preventif (*preventive control*), yaitu kontrol yang mempersulit keberhasilan penyerang dengan mencegah serangan secara aktif atau proaktif.
c. Kontrol pencegahan (*deterrent control*), yaitu kontrol yang tidak mencegah serangan, hanya bersifat pasif.
d. Kontrol korektif (*corrective control*), yaitu kontrol yang bertujuan untuk dapat memberikan pemulihan sebagai penjaminan perangkat lunak.
e. Kontrol kompensasi (*compensating control*) adalah kontrol yang harus diterapkan ketika kontrol perangkat lunak yang ditentukan pada persyaratan keamanan tidak dapat dipenuhi karena kendala bisnis atau teknis yang terdokumentasi.

6.1.2 Pemantauan Berkelanjutan

Tahap ini bertujuan untuk melakukan pemantauan terhadap sistem, perangkat lunak, atau proses. Penting untuk terlebih dahulu menentukan persyaratan pemantauan sebelum menerapkan solusi pemantauan. Persyaratan pemantauan perlu didefinisikan dari aspek bisnis di awal siklus hidup pengembangan perangkat lunak.

Pada persyaratan pemantauan, metrik terkait yang mengukur kinerja aktual dan operasi harus diidentifikasi dan didokumentasikan. Pengujian keamanan berkelanjutan harus dilakukan pada interval yang direncanakan atau sesuai dengan perubahan berdasarkan kebutuhan atau persyaratan.

6.1.3 Audit untuk Pemantauan

Tahap ini bertujuan untuk memberikan reviu dan pemeriksaan independen terhadap catatan dan aktivitas perangkat lunak. Audit dilakukan oleh auditor yang tanggung jawabnya meliputi pemilihan peristiwa yang akan diaudit pada perangkat lunak, pengaturan *audit flag* yang memungkinkan perekaman peristiwa tersebut dan menganalisis jejaknya dari peristiwa audit. Audit harus dilakukan secara berkala dan dapat memberikan wawasan tentang keberadaan dan keefektifan kontrol keamanan dan privasi.

6.2 Manajemen Insiden

Tahap ini bertujuan untuk mendeteksi dan memantau insiden pelanggaran keamanan.

6.2.1 Menentukan Peristiwa, Peringatan, dan Insiden

Tahap ini bertujuan untuk menentukan suatu insiden keamanan benar-benar terjadi atau tidak, sehingga harus ditentukan terlebih dahulu apa yang dimaksud dengan insiden. Kegagalan pada tahap ini dapat menyebabkan potensi kesalahan klasifikasi pada peristiwa dan peringatan sebagai insiden, dan hal ini dapat merugikan organisasi.

a. Peristiwa (*event*) adalah setiap tindakan yang diarahkan pada suatu objek yang mencoba mengubah keadaan objek tersebut;
b. Peringatan (*alert*) adalah peristiwa yang ditandai, yang perlu diteliti lebih lanjut untuk menentukan apakah kejadian dari peristiwa tersebut merupakan suatu insiden.
c. Peringatan dapat dikategorikan ke dalam insiden, dan kejadian buruk dapat dikategorikan ke dalam insiden keamanan jika melakukan pelanggaran atau mengancam pelanggaran pada kebijakan keamanan jaringan, sistem, atau aplikasi perangkat lunak.

6.2.2 Identifikasi Jenis Insiden

Tahap ini bertujuan untuk mengidentifikasi beberapa jenis insiden, yang meliputi namun tidak terbatas pada:

a. *Denial of Service* (DoS), yaitu serangan yang mencegah atau mengganggu pengguna yang sah untuk menggunakan jaringan, sistem, atau aplikasi perangkat lunak dengan cara menghabiskan sumber daya.
b. Kode berbahaya (*malicious code*), yaitu insiden yang berkaitan dengan entitas berbahaya berbasis kode seperti virus, *worm*, dan *trojan horse* yang berhasil menginfeksi *host*.
c. Akses tidak sah (*unauthorized access*), yaitu insiden terkait kontrol akses dimana penyerang memperoleh akses logik atau fisik ke jaringan, sistem atau aplikasi perangkat lunak, data, atau sumber daya TI lainnya, tanpa memiliki hak eksplisit untuk melakukannya.

d. Penggunaan yang tidak pantas (*inappropriate usage*), yaitu tindakan dimana penyerang melanggar penggunaan sumber daya perangkat lunak atau kebijakan yang dapat diterima (*acceptable use policy*) di organisasi.
e. Insiden yang kompleks, yaitu insiden yang mencakup 2 (dua) atau lebih jenis insiden.

6.2.3 Proses Tanggap Insiden

Tahap ini bertujuan untuk memungkinkan dan menjamin keamanan operasi organisasi dan tetap dalam bisnis. Fase utama dari proses tanggap insiden melibatkan persiapan, identifikasi, penahanan, perbaikan dan pemulihan, dan pembelajaran yang diperoleh. Penjelasan singkatnya adalah sebagai berikut:

a. Persiapan, yaitu menetapkan kemampuan tanggap insiden, mencegah insiden dengan memastikan bahwa sistem, jaringan, dan aplikasi dalam keadaan yang cukup aman.
b. Identifikasi, yaitu menganalisis dan memvalidasi setiap insiden, mengikuti proses yang telah ditentukan sebelumnya dan mendokumentasikan setiap langkah yang diambil, melakukan pemrioritasan insiden, serta melaporkan insiden yang ditemukan ke pihak yang relevan.
c. Penahanan, yaitu menjalankan strategi penahanan untuk meminimalkan pengaruh insiden terhadap sumber daya dan mencegah kerusakan.
d. Perbaikan, yaitu melakukan penghapusan ancaman dan perbaikan kontrol keamanan.
e. Pemulihan, yaitu mengembalikan sistem yang terdampak ke operasi normal.
f. Pembelajaran yang diperoleh (*lesson learned*), yaitu melakukan perubahan berdasarkan pembelajaran yang diperoleh dari insiden dan melakukan peningkatan.

6.3 Manajemen Permasalahan

Tahap ini bertujuan untuk menentukan dan menghilangkan akar penyebab masalah (insiden yang tidak diketahui) dan untuk meningkatkan layanan yang diberikan perangkat lunak kepada operasional bisnis agar tidak terulang kembali.

6.3.1 Pemberitahuan Insiden

Tahap ini bertujuan untuk mengidentifikasi dan memberi tahu insiden yang tidak diketahui atau terjadi permasalahan pada perangkat lunak.

6.3.2 Analisis Akar Penyebab

Tahap ini bertujuan untuk menentukan penyebab masalah dengan menerapkan langkah-langkah analisis akar penyebab (*root cause analysis /*RCA). Analisis akar penyebab dilakukan untuk menentukan mengapa masalah tersebut terjadi secara berulang kali dan sistematis, hingga tidak ada lagi penyebab atau masalah yang harus dijawab.

6.3.3 Penentuan Solusi

Tahap ini bertujuan untuk menentukan solusi, baik solusi sementara atau solusi permanen yang akan diterapkan.

6.3.4 Permintaan Perubahan

Tahap ini bertujuan untuk menyertakan solusi (untuk mendukung pengguna), kesalahan yang diketahui (*known error*), memperbarui informasi masalah, informasi manajemen, dan permintaan perubahan.

6.3.5 Menerapkan Solusi

Tahap ini bertujuan untuk mengimplementasikan solusi yang teridentifikasi setelah memulai permintaan perubahan.

6.3.6 Memantau dan Melaporkan

Tahap ini bertujuan untuk memantau solusi untuk masalah dengan menyiapkan pelaporan.

6.4 Manajemen Perubahan

Tahap ini bertujuan untuk:

a. Menentukan pemulihan dan penyelesaian masalah setelah akar penyebab masalah teridentifikasi;
b. Melacak kerentanan dan memantau penyelesaian masalah untuk memastikan bahwa solusinya sudah efektif dan masalah tidak terjadi lagi.

6.4.1 Manajemen *Patch* dan Kerentanan

Tahap ini bertujuan untuk memperbarui atau memperbaiki perangkat lunak yang ada dengan *patch*, yaitu potongan kode yang digunakan agar perangkat lunak tidak rentan terhadap *bug*. *Patching* adalah proses penerapan pembaruan atau perbaikan ini. *Patch* dapat digunakan untuk mengatasi masalah keamanan pada perangkat lunak atau sekadar menyediakan fungsionalitas tambahan dan merupakan bagian dari penguatan (*hardening*). Beberapa langkah penting yang perlu diambil sebagai bagian dari proses *patch* meliputi:

a. Memberi tahu pengguna perangkat lunak atau sistem tentang *patch*;
b. Menguji *patch* dalam lingkungan yang disimulasikan sehingga tidak ada masalah dengan kompatibilitas sebelumnya atau ketergantungan (hulu atau hilir).
c. Mendokumentasikan perubahan beserta rencana *rollback*. Perkiraan waktu untuk menyelesaikan penginstalan *patch*, kriteria untuk menentukan keberhasilan *patch*, dan rencana *rollback* harus dimasukkan sebagai bagian dari dokumentasi.
d. Mengidentifikasi masa pemeliharaan atau kapan waktu pemasangan *patch* harus dilakukan. Waktu terbaik untuk memasang *patch* adalah saat ada sedikit gangguan pada operasi normal bisnis, tetapi dengan sebagian besar perangkat lunak beroperasi dalam kondisi normal. Mengidentifikasi waktu terbaik untuk menerapkan *patch* merupakan tantangan saat ini.
e. Memasang *patch.*
f. Menguji *patch* setelah pemasangan di lingkungan produksi juga diperlukan. Kadang-kadang *reboot* atau *restart* perangkat lunak dimana *patch* diinstal diperlukan untuk membaca atau memuat pengaturan konfigurasi dan perbaikan yang lebih baru untuk diterapkan. Validasi kompatibilitas sebelumnya dan dependensi juga perlu dilakukan.

g. Memvalidasi bahwa *patch* tidak menurunkan status keamanan dan mengoperasionalkan sistem dan perangkat lunak sesuai dengan *baseline* keamanan minimum.
h. Pemantauan sistem yang di-*patch* sehingga tidak ada efek samping yang tidak diharapkan saat pemasangan *patch*.
i. Melakukan analisis *post-mortem* jika *patch* harus dibatalkan dan menggunakan pelajaran yang didapat untuk mencegah masalah di masa mendatang. Jika *patch* berhasil, maka *baseline* keamanan minimum perlu diperbarui.
j. Melakukan *virtual patching* untuk meng-*cover* sistem *legacy*. *Patch* virtual mengacu pada pembentukan lapisan penegakan kebijakan keamanan langsung yang mencegah eksploitasi kerentanan yang diketahui, tanpa mengubah kode sumber aplikasi, perubahan biner, atau memulai ulang aplikasi. *Patch* virtual dapat digunakan untuk menambahkan perlindungan sementara, memberikan waktu bagi tim pengembang untuk menerapkan *patch* fisik sesuai dengan jadwal pembaruan. *Patch* virtual juga dapat digunakan secara permanen untuk sistem lawas yang mungkin tidak dapat di-*patch*.

6.4.2 Pencadangan, Pemulihan, dan Pengarsipan

Tahap ini bertujuan untuk memastikan operasi dan kelangsungan bisnis tidak terganggu. Kelangsungan bisnis tanpa gangguan merupakan faktor penting dalam pengoperasian perangkat lunak yang aman. Tidak hanya data yang harus tersedia tetapi juga sistem itu sendiri.

Selain pencadangan terjadwal secara rutin, saat *patch* dan pembaruan perangkat lunak dibuat, disarankan untuk melakukan pencadangan penuh atas perangkat lunak yang sedang diubah. Penting juga untuk memverifikasi integritas dan pemulihan cadangan (terutama cadangan data).

Selain itu, ketika perangkat lunak telah terinfeksi oleh *malware*, maka satu-satunya pilihan yang tersisa adalah untuk memastikan integritas yang berkelanjutan dengan memformat dan menginstal ulang perangkat lunak sepenuhnya disertai dengan memulihkan data dari cadangan yang aman, tepercaya, dan terverifikasi.

Arsip dapat berguna dalam dukungan pengguna, terutama untuk pelanggan sebelumnya. Integritas pada arsip dapat dicapai dengan menggunakan *hashing* dan manajemen kunci yang tepat agar data yang dilindungi secara kriptografis pada arsip dapat digunakan saat melakukan pemulihan.

6.5 Pembuangan

Tahap ini bertujuan untuk mengidentifikasi perangkat lunak yang tidak beroperasi untuk dibuang guna mengurangi risiko residual.

6.5.1 Kebijakan Akhir Masa Pakai (*End-of-Life*)

Pada tahap ini dilakukan aktivitas manajemen risiko pada komponen sistem yang akan dibuang atau diganti untuk memastikan bahwa perangkat keras dan perangkat lunak dibuang dengan benar.

Untuk mengelola risiko selama fase pembuangan, penting bagi kita untuk memiliki kebijakan Akhir Masa Pakai (*End-of-Life*/EOL) yang harus ditetapkan dan dipatuhi. Kebijakan EOL adalah persyaratan pertama dalam pembuangan perangkat lunak yang aman serta data dan dokumen terkaitnya.

6.5.2 Kriteria Pembuangan (*Sunset Criteria*)

Tahap ini bertujuan untuk membuang atau mengganti produk seperti perangkat lunak atau perangkat keras yang menjalankan perangkat lunak dengan mengikuti kriteria pembuangan (*sunset criteria*) sebagai pedoman.

6.5.3 Proses Pembuangan (*Sunset Process*)

Tahap ini bertujuan untuk membuang teknologi terkait perangkat lunak yang dianggap tidak aman, tetapi tidak memiliki cara untuk mengurangi risiko ke tingkat yang dapat diterima, sehingga perangkat lunak harus dihentikan sesegera mungkin.

Sesuai dengan kebijakan EOL organisasi, proses EOL yang sesuai harus ditetapkan. Proses EOL adalah serangkaian tonggak dan aktivitas teknis dan bisnis, yang ketika selesai, membuat perangkat keras atau perangkat lunak menjadi usang dan tidak lagi diproduksi, dijual, diperbaiki, diperbaiki, dipelihara, atau didukung.

6.5.4 Pembuangan Informasi dan Sanitasi Media

Tahap ini bertujuan untuk memastikan penjaminan perangkat lunak dipertahankan dengan mengikuti langkah-langkah pembuangan perangkat lunak, dan juga untuk memastikan bahwa media yang menyimpan informasi juga disanitasi atau dimusnahkan sebagaimana mestinya. Sanitasi adalah proses menghilangkan informasi dari media seperti pemulihan dan pengungkapan data tidak mungkin dilakukan. Proses ini juga mencakup penghapusan label rahasia, tanda, dan *log* aktivitas yang terkait dengan informasi tersebut.

Checklist Secure SDLC

A. *Checklist* Persyaratan Keamanan

No.	Persyaratan Keamanan	Detail Persyaratan Keamanan	Keterangan	*Checklist*
1.1	**Sumber dari Persyaratan Keamanan**			
1.1.1	**Identifikasi Persyaratan *Core Security***			
	a. Persyaratan kerahasiaan (*confidentiality*)	1) Apakah diterapkan kriptografi, dengan: a) Mekanisme terang-terangan (*overt*). Misalnya: enkripsi, hash? b) Mekanisme terselubung (*covert*). Misalnya: steganografi, *digital watermarking*? 2) Apakah diterapkan penyamaran (*masking*)? 3) Apakah diterapkan kerahasiaan pada siklus hidup informasi: a) *Data in transit*? b) *Data in process*? c) *Data in storage*?		
	b. Persyaratan integritas (*integrity*)	Apakah terdapat diterapkan pelindungan untuk memastikan: 1) Integritas sistem? 2) Integritas data?		
	c. Persyaratan ketersediaan (*availability*)	Apakah diterapkan pelindungan dari: 1) Upaya penghancuran perangkat lunak/data? 2) Serangan *Denial of Service* (DoS)?		
	d. Persyaratan autentikasi	Apakah diterapkan autentikasi untuk memastikan keabsahan dan memvalidasi identitas pada: 1) Pengguna? 2) Administrator? 3) *Super user* (jika ada)?		
	e. Persyaratan otorisasi	Apakah diterapkan kontrol akses untuk memastikan otoritas entitas yang diautentikasi pada sumber daya?		
	f. Persyaratan akuntabilitas	Apakah diterapkan fungsi untuk pemeliharaan catatan riwayat tindakan pengguna (*audit trail*)?		
1.1.2	**Mengidentifikasi Persyaratan Umum**			

No.	Persyaratan Keamanan	Detail Persyaratan Keamanan	Keterangan	*Checklist*
	a. Persyaratan manajemen sesi	Apakah diterapkan *session identifier* untuk melacak perilaku pengguna dan mempertahankan status terautentikasi?		
	b. Persyaratan manajemen *error* dan *exception*	Apakah diterapkan pelindungan terhadap informasi *error* dan *exception*?		
	c. Persyaratan Manajemen Parameter Konfigurasi	Apakah diterapkan pelindungan terhadap informasi parameter dan kode konfigurasi perangkat lunak yang menyusun perangkat lunak?		
1.1.3	**Mengidentifikasi Persyaratan Operasional**			
	a. Persyaratan Lingkungan Penerapan	Apakah lingkungan dimana perangkat lunak akan digunakan telah diidentifikasi dan dianalisis?		
	b. Persyaratan Pengarsipan	Apakah terdapat kebutuhan pengarsipan untuk mendukung kelangsungan bisnis atau untuk memenuhi persyaratan peraturan?		
	c. Persyaratan Anti Pembajakan (*Anti-Piracy*)	Apakah diterapkan pelindungan anti-pembajakan pada perangkat lunak yang akan dikomersialkan?		
1.1.3	**Mengidentifikasi Persyaratan Lain**			
	a. Persyaratan Urutan dan Waktu	Apakah terdapat pelindungan terhadap kondisi *race condition* atau serangan *Time of Check/Time of Use* (TOC/TOU)?		
	b. Persyaratan Internasional	Apakah terdapat persyaratan hukum dan teknologi secara internasional yang harus dipatuhi?		
	c. Persyaratan Pengadaan	Apakah terdapat persyaratan keamanan perangkat lunak sudah dipenuhi pada perangkat lunak yang akan dibeli?		
1.2	**Klasifikasi Data**			
1.2.1	**Tipe Data**			
	a. Data Terstruktur	Apakah terdapat data terstruktur (*database*) pada perangkat lunak?		
	b. Data Tidak Terstruktur	Apakah terdapat data tidak terstruktur (gambar, video, *email*, dokumen, dan teks) pada perangkat lunak?		
1.2.2	Memberi Label pada Data	Apakah terdapat pelabelan pada data sesuai aspek C-I-A?		
1.2.3	Kepemilikan dan Peran pada Data	1) Apakah pemilik data sudah terdefinisi? 2) Apakah pemilik data sudah menjalankan perannya dalam mengelola data?		
1.2.4	*Data Lifecycle Management* (DLM)	Apakah terdapat prosedur dan praktik *Data Lifecycle Management* (DLM)?		

No.	Persyaratan Keamanan	Detail Persyaratan Keamanan	Keterangan	*Checklist*
1.2.5	Persyaratan Privasi	1) Apakah terdapat pedoman yang mengatur implementasi privasi pada data? 2) Apakah kontrol privasi sudah diterapkan sesuai persyaratan?		
1.3	**Pemodelan *Use Case* dan *Misuse Case***			
1.3.1	Menganalisis Skenario *Use Case*	1) Apakah sudah dibuat diagram *use case*? 2) Apakah sudah analisis terhadap skenario *use case*?		
1.3.2	Menganalisis Skenario *Misuse Case*	1) Apakah sudah dibuat diagram *misuse case*? 2) Apakah sudah analisis terhadap skenario *misuse case*?		
1.3.3	Membuat Model Serangan	1) Apakah sudah diidentifikasi serangan yang relevan dengan perangkat lunak? 2) Apakah sudah dianalisis siapa saja yang dapat menjadi sumber ancaman?		
1.3.4	Memilih Kontrol Mitigasi	Apakah sudah dianalisis kontrol keamanan untuk memitigasi serangan pada perangkat lunak?		
1.4	**Manajemen Risiko**			
1.4.1	Penilaian Risiko	Apakah sudah dilakukan aktivitas penilaian risiko berikut: 1) Karakterisasi sistem? 2) Identifikasi ancaman? 3) Identifikasi kerentanan? 4) Analisis kontrol? 5) Penentuan kemungkinan? 6) Analisis dampak? 7) Penentuan risiko? 8) Rekomendasi kontrol? 9) Dokumentasi hasil?		
1.4.2	Mitigasi Risiko	Apakah sudah dilakukan aktivitas berikut: 1) Penentuan opsi mitigasi risiko? 2) Penentuan strategi mitigasi risiko? 3) Menentukan pendekatan pada implementasi kontrol? 4) Mengkategorikan kontrol? 5) Analisis biaya dan manfaat? 6) Menentukan risiko residual?		
1.4.3	Evaluasi dan Penilaian Risiko	Apakah proses evaluasi dan penilaian sudah dilakukan secara berkala?		

B. *Checklist* Desain Keamanan

No.	Persyaratan Keamanan	Detail Persyaratan Keamanan	Keterangan	*Checklist*
2.1	**Pertimbangan Desain *Core Security***			
2.1.1	Desain Kerahasiaan	Apakah sudah terdapat desain aspek kerahasiaan menggunakan: 1) Teknik kriptografi, dengan mempertimbangkan: a) Algoritma enkripsi? b) Ukuran kunci? c) Manajemen kunci? 2) Penyamaran (*masking*)?		
2.1.2	Desain Integritas	Apakah sudah terdapat desain aspek integritas menggunakan: 1) *Hashing* (fungsi *hash*)? 2) Integritas referensi? 3) Penguncian sumber daya?		
2.1.3	Desain Ketersediaan	Apakah sudah terdapat desain aspek ketersediaan menggunakan: 1) Pengkodean perangkat lunak? 2) Replikasi? 3) *Failover*? 4) Skalabilitas?		
2.1.4	Desain Autentikasi	Apakah sudah terdapat desain untuk autentikasi menggunakan: 1) Autentikasi 2FA atau MFA? 2) *Single Sign-On* (SSO)?		
2.1.5	Desain Otorisasi	Apakah sudah terdapat desain untuk otorisasi menggunakan mekanisme kontrol akses: 1) Direktori? 2) *Access Control List* (ACL)? 3) Matriks kontrol akses? 4) Kapabilitas menggunakan token yang tidak dapat dipalsukan? 5) Kontrol akses berorientasi prosedur?		
2.1.6	Desain Akuntabilitas	Apakah sudah terdapat desain untuk mendukung *audit trail*?		
2.2	**Pertimbangan Desain Tambahan**			
2.2.1	Bahasa Pemrograman	Apakah bahasa pemrograman yang akan digunakan sudah ditentukan?		
2.2.2.	Jenis, Format, Jangkauan, dan Panjang Data	Apakah sudah ditentukan: 1) Tipe data primitif atau *built-in*?		

No.	Persyaratan Keamanan	Detail Persyaratan Keamanan	Keterangan	*Checklist*
		2) Tipe data yang ditentukan oleh pemrogram? 3) Nilai dan operasi yang diizinkan? 4) Ketidakcocokan dan kesalahan konversi?		
2.2.3	Keamanan *Database*	Apakah sudah terdapat desain keamanan *database* dengan mempertimbangkan: 1) *Polyinstantiation*? 2) Enkripsi *database*? 3) Normalisasi? 4) *Trigger* dan *view*?		
2.2.4	Desain Antarmuka	Apakah sudah terdapat desain antarmuka pada: 1) Antarmuka pengguna (*user interface/*UI)? 2) Antarmuka pemrograman aplikasi (*application programming interface/*API)? 3) Antarmuka manajemen keamanan (*security management interface/*SMI)? 4) Antarmuka *out-of-band*? 5) Antarmuka *log*?		
2.2.5	Interkonektivitas	Apakah sudah terdapat desain interkonektivitas pada perangkat lunak hulu dan hilir?		
2.3	**Pemodelan Ancaman**			
2.3.1	Dekomposisi Perangkat Lunak	Apakah sudah dilakukan dekomposisi perangkat lunak dengan mempertimbangkan: 1) Ketergantungan eksternal? 2) Titik masuk (vektor serangan)? 3) Aset? 4) *Attack surface*? 5) Tingkat kepercayaan? 6) Analisis aliran data? 7) Analisis transaksi ? 8) Diagram *data flow*?		
2.3.2	Menentukan dan Mengurutkan Ancaman	1) Apakah sudah mengidentifikasi ancaman dengan metode pengkategorian ancaman (misalnya STRIDE)? 2) Apakah sudah mengurutkan ancaman menggunakan model peringkat ancaman-risiko (misalnya DREAD)?		

No.	Persyaratan Keamanan	Detail Persyaratan Keamanan	Keterangan	*Checklist*
2.3.3	Menentukan Penanggulangan dan Mitigasi	1) Apakah sudah mengurutkan ancaman berdasarkan peringkat risiko? 2) Apakah sudah mengidentifikasi penanggulangan berdasarkan pemetaan ancaman-penanggulangan?		

C. *Checklist* Pengembangan Keamanan

No.	Persyaratan Keamanan	Detail Persyaratan Keamanan	Keterangan	*Checklist*
3.1	**Kerentanan dan Kontrol Umum pada Perangkat Lunak**			
3.1.1	*Database* Kerentanan	1) Apakah organisasi mengacu ke daftar kerentanan perangkat lunak (OWASP Top 10 Project atau CWE/25)? 2) Apakah terdapat *database* kerentanan yang ditemukan pada perangkat lunak yang diimplementasikan?		
3.2.3	Praktek Pengkodean Defensif	Apakah teknik pengkodean defensif untuk mengurangi *attack surface* sudah dipraktekan?		
3.2	**Proses Perangkat Lunak yang Aman**			
3.2.1	Versi pada Kode Sumber	Apakah versi pada kode sumber sudah diterapkan?		
3.2.2	Analisis Kode	Apakah analisis kode diterapkan menggunakan: 1) Analisis kode statis? 2) Analisis kode dinamis?		
3.2.3	Reviu Kode	Apakah dilakukan evaluasi kode sumber secara sistematis dan manual?		
3.2.4	Pengujian Pengembang	Apakah pengembang sudah melakukan pengujian menggunakan alat dan teknik pengujian yang sistematis yang meliputi: 1) Pengujian unit? 2) Pengujian integrasi? 3) Pengujian regresi? 4) Pengujian perangkat lunak?		
3.3	**Mengamankan Lingkungan Pengembangan**			
3.3.1	Mengamankan Akses Fisik ke Perangkat Lunak yang Membangun Kode	Apakah akses fisik ke perangkat lunak yang membangun kode sudah diamankan?		
3.3.2	Menggunakan *Access Control List* (ACL)	Apakah digunakan *Access Control List* (ACL) untuk mencegah akses pengguna yang tidak sah?		

No.	Persyaratan Keamanan	Detail Persyaratan Keamanan	Keterangan	*Checklist*
3.3.3	Menggunakan Perangkat Lunak Kontrol Versi	Apakah perangkat lunak kontrol versi sudah digunakan?		
3.3.4	*Build Automation*	Apakah *build automation* sudah digunakan?		
3.3.5	Penandatanganan Kode (*Code Signing*)	Apakah *code signing* sudah digunakan?		

D. *Checklist* Pengujian Keamanan

No.	Persyaratan Keamanan	Detail Persyaratan Keamanan	Keterangan	*Checklist*
4.1	**Validasi Permukaan Serangan (*Attack Surface*)**			
4.1.1	Pengujian Pasca Pengembangan	Apakah sudah dilakukan *dynamic code analysis*, yaitu pemeriksaan kode pada saat program dijalankan?		
4.1.2	Pengujian Keamanan Menggunakan Metode Pengujian Keamanan	Apakah dilakukan pengujian keamanan menggunakan: 1) Pengujian *white box*? 2) Pengujian *black box*? 3) Pengujian validasi kriptografi?		
4.1.3	Melakukan Pengujian Keamanan Perangkat Lunak untuk Jaminan Kualitas	Apakah dilakukan pengujian keamanan perangkat lunak untuk menjamin kualitas yang meliputi: 1) Pengujian validasi *input*; 2) Pengujian untuk kontrol cacat injeksi (*injection flaw*); 3) Pengujian untuk kontrol serangan *scripting* (*scripting attack*); 4) Pengujian untuk kontrol nir-penyangkalan (*non-repudiation*); 5) Pengujian untuk kontrol *spoofing*; 6) Pengujian untuk kontrol *error* dan *exception handling* (*failure testing*); 7) Pengujian untuk kontrol eskalasi hak istimewa (*privilege escalation*); 8) Pengujian untuk pelindungan anti-pembalikan (*anti-reversing*); 9) Pengujian ketahanan (*stress testing*).		
4.2	**Manajemen Data Pengujian**			
4.2.1	Mengidentifikasi *Output* Data Pengujian untuk	1) Apakah *output* dari data pengujian sudah sesuai dengan harapan atau telah memenuhi persyaratan?		

No.	Persyaratan Keamanan	Detail Persyaratan Keamanan	Keterangan	*Checklist*
	Memastikan Persyaratan Perangkat Lunak	2) Apakah menggunakan data tiruan di lingkungan pengujian atau simulasi?		
4.2.1	Melakukan Pengujian dengan Transaksi Sintetis	Apakah transaksi pada pengujian dilakukan menggunakan data *dummy* yang tidak terkait dengan proses bisnis sebenarnya?		
4.2.3	Solusi pada Manajemen Data Pengujian	Apakah digunakan solusi (alat atau layanan) pada manajemen data pengujian?		
4.2.4	Pelaporan dan Pelacakan Cacat	Apakah terdapat mekanisme untuk melaporkan cacat (*defect/flaw*) dan kemudian melacak *bug* pengkodean, cacat desain (*design flaw*), anomali perilaku (*logic flaw*), *error*, *fault*, dan kerentanan pada perangkat lunak?		

E. *Checklist* Penerapan Keamanan

No.	Persyaratan Keamanan	Detail Persyaratan Keamanan	Keterangan	*Checklist*
5.1	**Pertimbangan Penerimaan Perangkat Lunak**			
5.1.1	Kriteria Penyelesaian	Apakah terdapat kriteria penyelesaian untuk fungsionalitas dan keamanan perangkat lunak dengan *milestone* yang jelas?		
5.1.2	Manajemen Perubahan	Apakah terdapat mekanisme untuk menangani permintaan perubahan pada penerapan perangkat lunak?		
5.1.3	Persetujuan untuk Menerapkan atau Merilis	Apakah terdapat mekanisme persetujuan/penolakan untuk menerapkan/merilis perangkat lunak?		
5.1.4	Kebijakan Penerimaan dan Pengecualian Risiko	Apakah terdapat kebijakan untuk penerimaan dan pengecualian risiko?		
5.1.5	Dokumentasi Perangkat Lunak	Apakah terdapat dokumentasi perangkat lunak yang memadai, yang meliputi: 1) Perancangan? 2) Penginstalan? 3) Pengaturan konfigurasi? 4) Penggunaan? 5) Pengaturan?		
5.2	**Verifikasi dan Validasi (V&V)**			
5.2.1	Reviu	Apakah dilakukan reviu pada akhir setiap tahap untuk memastikan bahwa perangkat lunak berfungsi seperti yang diharapkan dan memenuhi spesifikasi bisnis yang diharapkan?		

No.	Persyaratan Keamanan	Detail Persyaratan Keamanan	Keterangan	*Checklist*
5.2.2	Pengujian	Apakah sudah dilakukan pengujian untuk memastikan persyaratan sudah dipenuhi dan menentukan penyimpangan yang diharapkan dengan: 1) Pengujian deteksi kesalahan? 2) Pengujian penerimaan? 3) Pengujian pihak independen?		
5.3	**Sertifikasi dan Akreditasi (C&A)**			
5.3.1	Mendapatkan Sertifikasi	Apakah dilakukan sertifikasi perangkat lunak yang mencakup evaluasi penjaminan: 1) Hak pengguna, hak istimewa, dan manajemen profil? 2) Sensitivitas data dan aplikasi serta pengendalian yang sesuai? 3) Konfigurasi perangkat lunak, fasilitas, dan lokasi? 4) Interkonektivitas dan ketergantungan? 5) Mode keamanan operasional?		
5.3.2	Mendapatkan Akreditasi	Apakah terdapat mendapatkan akreditasi atas penerimaan formal pihak manajemen pada penggunaan perangkat lunak?		
5.4	**Instalasi**			
5.4.1	Penguatan (*Hardening*)	1) Apakah dilakukan penguatan (*hardening*) sistem operasi *host* dengan menggunakan *baseline*, *update*, dan *patch*? 2) Apakah dilakukan penguatan (*hardening*) perangkat lunak melibatkan pengaturan konfigurasi dan perancangan perangkat lunak agar aman secara *default*?		
5.4.2	Konfigurasi Lingkungan	1) Apakah sudah dipastikan bahwa lingkungan pengembangan dan pengujian cocok dengan lingkungan produksi? 2) Apakah terdapat *checklist* pra-instalasi untuk menentukan parameter yang diperlukan dalam menjalankan perangkat lunak dengan konfigurasi yang tepat?		
5.4.3	Manajemen Rilis	Apakah terdapat mekanisme untuk memastikan rilis perangkat lunak dengan benar ke dalam lingkungan komputasi operasi?		
5.4.4	*Bootstrap* dan *Startup* yang Aman	Apakah *Power-On Self-Test* (POST) sudah dijalankan setelah instalasi perangkat lunak?		

F. *Checklist* Pemeliharaan Keamanan

No.	Persyaratan Keamanan	Detail Persyaratan Keamanan	Keterangan	*Checklist*
6.1	**Operasi, Pemantauan dan Pemeliharaan**			
6.1.1	Melaksanakan Pengamanan Operasi	Apakah sudah diterapkan kontrol keamanan operasi yang meliputi: 1) Kontrol detektif? 2) Kontrol preventif? 3) Kontrol pencegahan? 4) Kontrol korektif? 5) Kontrol kompensasi?		
6.1.2	Pemantauan Berkelanjutan	1) Apakah terdapat pemantauan berkelanjutan terhadap sistem, perangkat lunak, atau proses? 2) Apakah terdapat metrik terkait yang mengukur kinerja aktual dan operasi pemantauan?		
6.1.3	Audit untuk Pemantauan	Apakah audit terhadap catatan dan aktivitas perangkat lunak sudah dilakukan secara berkala?		
6.2	**Manajemen Insiden**			
6.2.1	Menentukan Peristiwa, Peringatan, dan Insiden	Apakah terdapat kebijakan atau prosedur yang memuat definisi tentang peristiwa, peringatan, dan insiden?		
6.2.2	Identifikasi Jenis Insiden	Apakah terdapat kebijakan atau prosedur yang memuat tentang jenis insiden yang ditangani di organisasi?		
6.2.3	Proses Tanggap Insiden	Apakah terdapat kebijakan atau prosedur yang memuat proses tanggap insiden?		
6.3	**Manajemen Permasalahan**			
6.3.1	Pemberitahuan Insiden	Apakah terdapat prosedur yang mengatur tentang pemberitahuan insiden?		
6.3.2	Analisis Akar Penyebab	Apakah terdapat prosedur yang mengatur tentang analisis akar penyebab dari permasalahan?		
6.3.3	Penentuan Solusi	Apakah terdapat prosedur yang mengatur tentang penentuan solusi dari permasalahan?		
6.3.4	Permintaan Perubahan	Apakah terdapat prosedur yang mengatur tentang permintaan perubahan terkait permasalahan?		
6.3.5	Menerapkan Solusi	Apakah terdapat prosedur yang mengatur tentang penerapan solusi dari permasalahan?		

No.	Persyaratan Keamanan	Detail Persyaratan Keamanan	Keterangan	*Checklist*
6.3.6	Memantau dan Melaporkan	Apakah terdapat prosedur yang mengatur tentang pelaporan permasalahan yang bertujuan memantau penerapan solusi.		
6.4	**Manajemen Perubahan**			
6.4.1	Manajemen *Patch* dan Kerentanan	Apakah terdapat prosedur yang mengatur tentang manajemen *patch* dan kerentanan?		
6.4.2	Pencadangan, Pemulihan, dan Pengarsipan	Apakah terdapat prosedur yang mengatur tentang pencadangan, pemulihan, dan pengarsipan?		
6.5	**Pembuangan**			
6.5.1	Kebijakan Akhir Masa Pakai (*End-of-Life*)	Apakah terdapat kebijakan yang mengatur tentang *End-Of-Life* (EOL) perangkat lunak?		
6.5.2	Kriteria Pembuangan (*Sunset Criteria*)	Apakah terdapat prosedur yang mengatur tentang kriteria pembuangan perangkat lunak (*sunsiet criteria*)?		
6.5.3	Proses Pembuangan (*Sunset Process*)	Apakah terdapat prosedur yang mengatur tentang proses pembuangan perangkat lunak (*sunset process*)?		
6.5.4	Pembuangan Informasi dan Sanitasi Media	Apakah terdapat prosedur yang mengatur tentang pembuangan informasi dan sanitasi media?		

Referensi

- What is Secure SDLC? (https://www.checkpoint.com/cyber-hub/cloud-security/what-is-secure-sdlc/)
- Guidelines for Secure Software Development Life Cycle (SSDLC). Ministry of Communication and Multimedia Malaysia and CyberSecurity Malaysia.