Universitatea Politehnica Bucuresti

Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

SECURITATEA PRODUSELOR SOFTWARE

Studenti: Hulia Suliman – 442A

Ion Alexandru Nicolae – 442A

Radu Constantin Ciobanica – 441A

Stanciu Pompiliu – 441A

Profesor Coordonator: Stefan Stancescu

Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Cuprins:

Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

❖ Pompiliu Stanciu:

- **Analiza Statica**
  - **Introducere ..... pagina 44**
    - **Metode Formale ..... pagina 45**
    - **Verificarea modelului ..... pagina 46**
  - **Testarea fiabilitatii ..... pagina 49**
    - **Profile operationale ..... pagina 50**
  - **Testarea securitatii .....pagina 52**
  - **Bibliografie .....pagina 54**

Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Hulia Suliman

Fiabilitate si Securitate

- **Obiective:**

  Scopul acestui capitol este de a introduce fiabiltatea si securitatea software. Dupa parcurgerea acestui capitol veti putea:

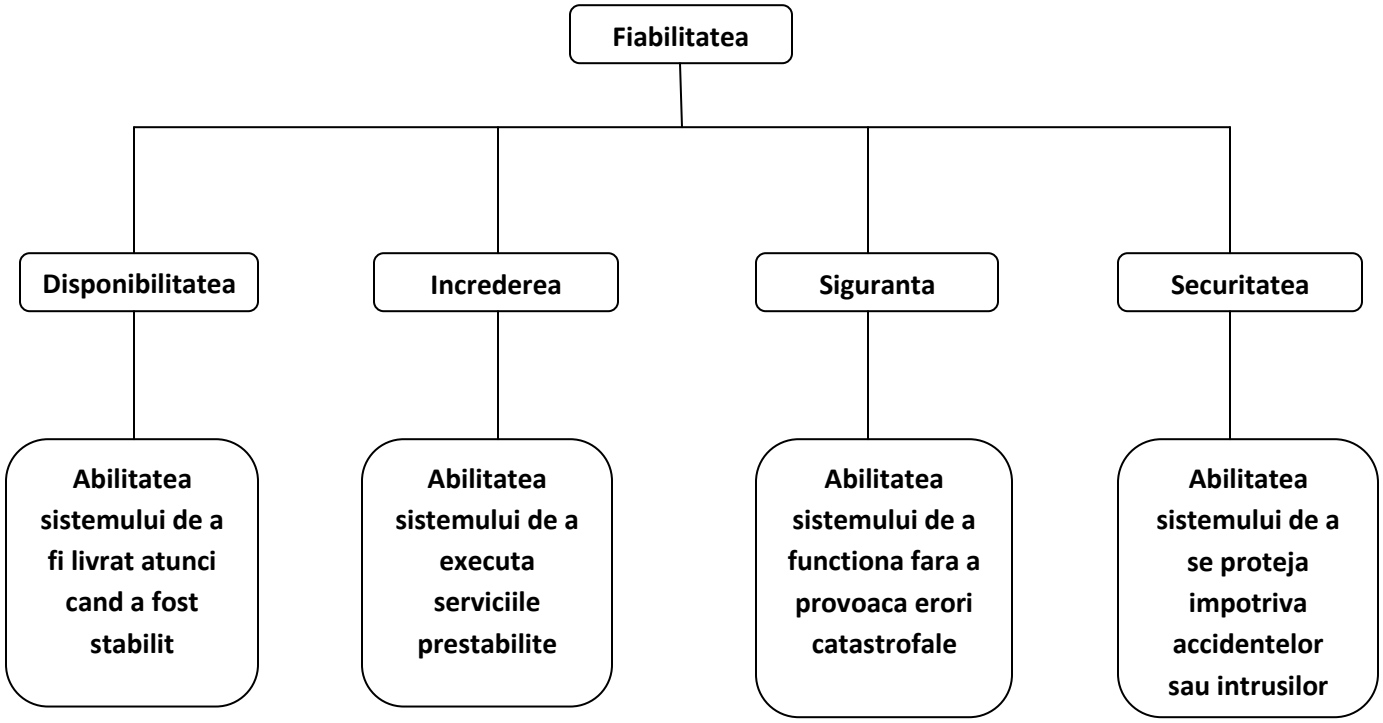
  - Intelege de ce fiabilitatea si securitatea sunt de obicei mai importante decat caracteristicile functionale ale unui sistem software;
  - Intelege cele patru proprietati ale fiabilitatii, si anume *disponibilitatea, increderea, sigurante si securitatea*;
  - Intelege ca pentru a obtine un produs software securizat, de incredere, trebuie sa evitai greselile in timpul dezvoltarii sistemului pentru a detecta si a corecta erorile atunci cand sistemul functioneaza si pentru a limita daunele cauate de esecuri operationale;
  - Deveni familiar cu terminologia de specialitate care este utilizata atunci cand se discuta despre fiabilitate si securitate.

Fiabilitatea sistemelor software, de obicei, este mai importanta decat detaliile functionale din urmatoarele motive:

1) *Esecurile sistemului afecteaza un numar mare de oameni*: Multe sisteme includ functionalitati care sunt foarte rar accesate. Daca aceste optiuni nu ar exista, atunci ar fi afectat un numar mic de utilizatori. Esecurile sistemului, care afecteaza disponibilitatea sistemlui, afecteaza toti utilizatorii sistemului. Esec poate insemna ca functionarea normala a sistemului este imposibila.
2) *Utilizatorii resping adesea sistemele care sunt nefiabilr sau nesigure*: Daca utilizatorii gasesc ca un sistem nu este de incredere sau nu este sigur, ei vor refuza sa-l foloseasca. In plus, utilizatorii vor refuza sa mai cumpere sau sa mai foloseasca alte produse ale aceleiasi companii care a produs sistemul nesigur, deoarece ei cred ca si aceste produse sunt de neincredere si nesigure.
3) *Costurile sistemelor eronate pot fi foarte mari*: De exemplu, pentru aplicatii cum ar fi un sistem de control al unui reactor sau sistem de navigatie al unei aeronave, costul esecului sistemului este mult mai mare decat costul sistemului care controleaza aplicatiile.
4) *Sistemele care nu sunt de incredere pot provoca pierderi de informatie*: Unele informatii sunt scumpe si greu de obtinut si mentinut. Costurile recuperarii datelori pierdute sau corupte, de obicei, sunt foarte ridicate.

Proprietatile Fiabilitatii

Fiabilitatea unui sitem software reprezinta proprietatea sistemului care reflecta autenticitatea ei. In aceste circumstante, increderea inseamna gradul de incredere al utilizatorilor ca sistemul va functiona asa cum se asteapta si ca nu va esua in conditii normale de lucru. Nu are sens folosirea cifrelor pt a exprima fiabilitatea unui sistem softaware. In general, se utilizeaza termeni precum *nu este fiabil, foarte fiabil* sau *ultra-fiabil* pentru a exprima gradul de incredere intr-un sistem.



Exista patru dimensiuni principale ale fiabilitatii, asa cum sunt prezentate in imaginea de mai sus:

1) *Disponibilitatea:* Reprezinta probabilitatea ca sistemul sa fie gata si sa functioneze, oferind servicii utilizatorilor la orice moment de timp;
2) *Increderea:* Reprezinta probabilitatea, ca dupa un anumit timp, sistemul sa ofere servicii corecte utilizatorilor, asa cum se asteapta ei;
3) *Siguranta:* Reprezinta probabilitatea ca sistemul sa cauzeze pagube, atat utilizatorilor, cat si mediului inconjurator;
4) *Securitatea:* Reprezinta probabilitatea ca sistemul sa reziste impotriva accidentelor sau a persoanelor neautorizate.

Aceste proprietati ale fiabilitatii sunt complexe si pot fi separate la randul lor in alte proprietati mai simple. De exemplu, *securitatea* include *integritate*a (sa te asiguri ca programele sistemului sau datele nu sunt afectate) si *confidentialitatea* (sa te asigur ca informatia nu poate fi accestata decat de persoanele care sunt autorizate). *Incredrea* include *corectitudinea* (sa te asiguri ca serviciile sistemul sunt cele precizate) si *timpul limita* (sa te asiguri ca sistemul este livrat cand este cerut). Bineinteles, ca nu toate aceste proprietati pot fi respectate de toate sistemele softwre.

De asemenea, pe langa proprietatile fiabiliatatii prezentate anterior, ne putem gandi la alte proprietati ale sistemelor:

1) *Posibilitatea de a fi reparat:* Erorile sistemelor sunt inevitabile, insa distrugerile si pierderile cauzate pot fi minimizate daca sistemul poate fi reparat in scurt timp. Posibilitatea ca un sistem software sa fie reparat este foarte usoara daca exista acces la codul sursa si daca exista perosane care pot face acest lucru. Produsele software ‘open source’ sunt de foarte mare ajutor in acest caz, insa reutilizarea produsului devine mai grea.
2) *Mentenabilitatea:* Pe masura ce sistemul este folosit, vor aparea noi cerinte si este foarte importanta mentinerea functionala a sistemului adaptandu-l la noile cerinte. Mentenabilitatea produselor software trebuie sa presupuna costuri ieftine si o probabilitate mica de defectare in urma schubarilor.
3) *Capacitatea de supravietuire:* Reprezinta abilitatea sistemului software de a furniza servicii in timp ce este ‘atacat’ sau cand anumite parti nu mai functioneaza.
4) *Toleranta la erori:* Aceasta proprietate tine de design-ul sistemului si reprezinta modul in care sistemul ocoleste si tolereaza eririle de natura umana. Cand o eroare produsa de un utilizator apare, sistemul ar trebui pe cat posibil sa o detecteze si sa o repare in mod automat sau sa ceara utilizatorului sa reia comzile gresite anterior.

Toate aceste proprietati ale fiabilitatii produselor software sunt relative. Un produs poate deveni de neincredere daca un ‘intrus’ modificat informatiile sale. Acest lucru compromite disponibilitatea produsului. Daca un sistem este ‘infectat cu un virus’, atunci nu mai poti avea incredere in el, deoarce nu mai este sigur si virusul poate determina schimbarea funcionaltatilor produsului.

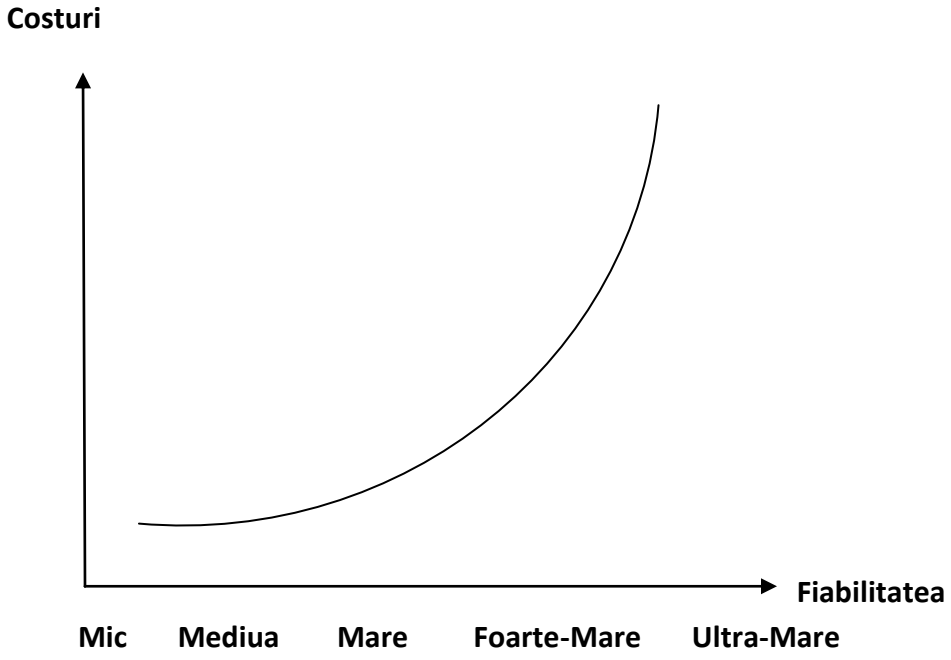
Pentru a dezvolta produse de incredre trebuie sa te asuguri ca:

1) Eviti introducerea accidentala a erorilor in sistem;

2) Tu ai facut design-ul proceselor de verificare si validare ale erorilor care afecteaza fiabilitatea sistemului;
3) Tu ai facut design-ul mecanismului de protectie care apara produsul impotriva tacurilir externe care compromit disponibilitatea sau securitatea sistemului;
4) Tu ai configurat mediul de lucru si sistemul de operare corect.

De asemenea sistemul trebuie sa includa un mecanism de recuperare care sa fie rapid pe cat posibil.

Urmatoarea figura arata relatia dintre costuri si fiabilitatea sistemului. Daca produsul nu este foarte fiabil, atunci si costurile sunt relativ scazute. Costurile pentru a imbunatati un sistem sunt ridicate, iar beneficiile nu sunt mari. De asemenea, mai exista si problema tesatrii produselor software pentru a arata fiabilitatea lor. Cu cat produdul sofwtare devine mai fiabil, cu atat erorile se diminueaza. Insa acest lucru necesita din ce in ce mai multe teste pentru a sti cate erori mai exista in sistem. Cum testarea este foarte scumpa, costurile urca dramatic, pe masura ce fiabilitatea produsului creste.



Disponibilitate si Incredere

Disponibilitatea si increderea sistemului sunt proprietati apropiate care pot fi exprimate ca probabilitati numerice. Disponibilitatea este probabilitatea ca un sistem sa fie utilizabil si sa satisfaca cerintele utilizatorului. Increderea este probabilitatea ca serviciile sistemului sa fie furnizate ca in specificatiile acestuia.

Increderea si disponibilitatea sunt asemanatoare dar uneori una este mai importanta decat alta. Daca utilizatorii se asteapta la functionarea continua a sistemului, atunci acesta trebuia sa aiba o disponibilitate ridicata. Daca apar pierderi in sistem care duc la cedarea acestuia, el isi poate reveni rapid, caz in care nu afecteaza foarte mult utilizatorii. In aceste sisteme, increderea poate fi relativ scazuta.

O centrala telefonica care ruteaza apelurile telefonice este un exemplu in care disponibilitatea este mai importanta decat increderea. Utilizatorii se asteapta la un ton telefonic cand ridica un receptor, deci sistemul are cerinte mari de disponibilitate. Daca o eroare in sistem apare, acesta are capacitatea de a se reface foarte rapid. Centralele telefonice pot reseta sistemul si sa refaca cererea de conexiune. Acest lucru poate fi facut foarte rapid, utlizatorii neobservand eroarea ce a intervenit. Mai mult, chiar daca un apel este intrerupt, consecintele nu sunt serioase. Asadar, disponibilitatea este mai importanta decat increderea intr-un astfel de sistem.

Disponibilitatea si increderea unui sistem pot fi definite mai precis dupa cum urmeaza:

1) *Increderea:* Probabilitatea unei operatiuni fara erori intr-un anumit timp, intr-un anumit mediu, pentru un scop specific
2) *Disponibilitatea:* Probabilitatea ca un sistem, la un moment de timp, sa fie operational si sa livreze serviciile cerute.

Una din problemele practice in dezvoltarea sistemelor de incredere este faptul ca notiunile noastre despre incredere si disponibilitate sunt mai vaste decat aceste definitii limitate. Definitia increderii este ca mediul in care sistemul este folosit si scopul pentru care este folosit trebuie luat in considerare. Daca masuram increderea sistemului intr-un mediu, nu putem afirma ca increderea este aceeasi daca sistemul este folosit intr-un mod diferit.

Spre exemplu, luam cazul unui calculator intr-un birou unde utilizatorii urmeaza instructiuni si il folosesc doar pentru un anumit scop, neincercand sa-l foloseasca pentru altceva. Daca

masuram increderea acestuia intr-o universitate, aceasta este diferita deoarece aici studentii vor explora limitele acestui sistem.

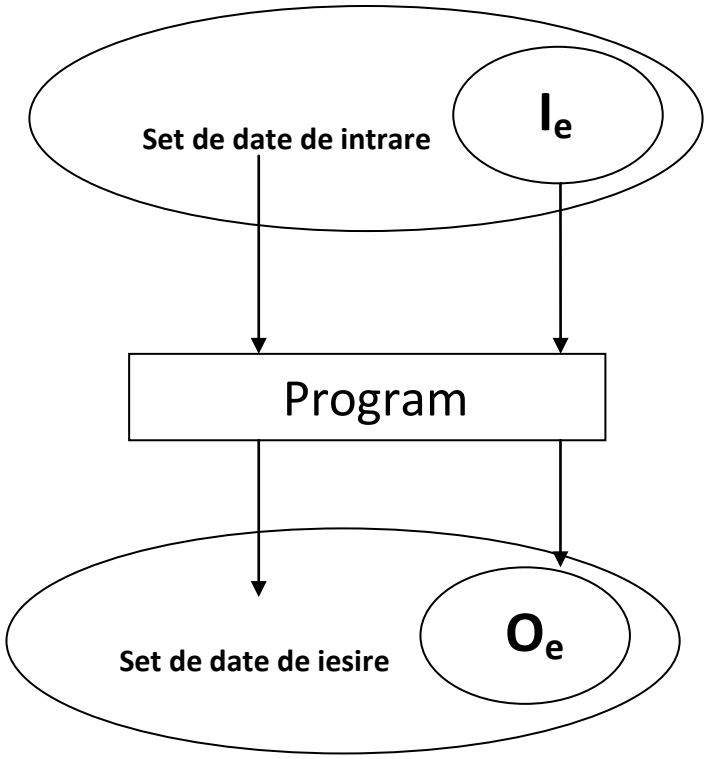
Aceste definitii standard nu iau in considerare severitatea erorilor si consecintele nedisponibilitatii. De exemplu, erorile care apar asupra datelor dintr-un calculator nu pot fi comparate cu blocarea unui calculator, fapt care se rezolva doar prin restartarea acestuia.

O definitie stricta a increderii se refera la implementarile din sistem in concordanta cu specificatiile acestuia. Din pacate, multe specificatii sunt incomplete si ramane ca un specialist software sa interpreteze cum ar trebui sa se comporte un sistem.

Disponibilitatea nu repinde doar de cate ori pica un sistem. De exemplu daca A pica o data pe an, iar B pica o data pe luna, insa lui A ii ia 3 zile sa se restarteze dupa o eroare, iar lui B ii ia 10 minute, atunci putem afirma ca B are o disponibilitatea mai ridicata.

Nici procentajul in care un sistem este disponibil nu este in totalitate relevant. Daca un sistem este picat intre 3 AM si 4 AM nu afecteaza atat de mult ca un sistem daca pica 10 minute in timpul orelor de lucru.

Figura de mai jos arata prelucrarea unor date de intrare si furnizarea unor date de iesire. De exemplu, la introducerea unei adrese web, la iesire va aparea pagina web.



Intrarile ilustrate in elipsa I_e cauzeaza erori in sistem si date de iesire eronate. Increderea programului depinde de numarul de date de intrare care fac parte din setul de date posibile care duc la erori in sistem. Daca intrarile din I_e sunt executate de parti folosite des din sistem,

erorile vor fi frecvente. Totusi, daca aceste sutn executate de un cod rar folosit din sistem, utilizatorii vor experimenta erori mai rar.

Pentru ca fiecare utilizator foloseste sistemul intr-un mod diferit, acestuia au perceptii diferite despre incredere. Erori care afecteaza increderea sistemului pentru un utilizator pot sa nu apara niciodata in modul de lucru diferit al altui utlizator.

Problemele unui sistem nu duc la erori in sistem si erorile unui nu duc la probleme in sistem deoarece:

1) Nu tot codul dintr-un program este executat. Codul care include o problema (esecul de a initializa o variabile) poate sa nu fie executat niciodata din cauza modului in care utilizat programul
2) Erorile sunt temporare. O variabila de stare poate avea o valoare incorecta cauzata de executia unui cod gresit. Totusi, inainte ca acesta sa fie accesat si sa cauzeze o problema in sistem, alte date de intrare ale sistemului ar fi putut fi procesate si sa reseteze starea la o valoare corecta.
3) Sistemul poate include detectie de probleme si mecanism de protectie. Acestea asigura ca un comportament eronat este descoperit si corectat inainte de a afecta serviciile sistemului.

Diferenta intre greseli, erori si esecuri identifica 3 metode care sunt folosite pentru a imbunatati increderea unui sistem:

1) *Evitarea greselilor:* Tehnici de dezvoltare sunt folosite pentru a minimiza posibilitatea erorilor de natura umana si/sau greselilor inainte ca acestea sa duc intr-un esec al sistemului.
2) *Detectia si indepartarea erorilor:* Folosirea tehnicilor de validare si verificare care cresc sansele ca greselile sa fie detectate si indepartate inainte de a fi folosite de sistem.
3) *Toleranta greselilor:* Tehnica ce asigura faptul ca problemele in sistem nu duc la erori in sistem si ca erorile din sistem nu duc la esecuri in sistem. Implementarea tehnicilor de auto-corectie si folosirea modulelor redundante sunt exemple pentru tehnicile de toleranta a greselilor.

Siguranta

Sistemele critice din punct de vedere al sigurantei sunt cele nu ar trebui sa afecteze negativ oamenii sau mediul inconjurator, chiar daca sistemul pica. Cateva exemple ar fi aparatura dintr-un avion, controlul procesarii unor produse chimice sau sistemele de control ale automobilelor.

Controlul hardware este mai simplu de implementat si analizat decat controlul software, insa in prezent se construiesc sisteme foarte complexe care nu pot fi controlate doar hardware. Asadar software-ul critic pentru siguranta se imparte in doua:

1) *Software critic primar:* Acesta este incorporat pe controlerul unui sistem. Functionarea eronata a acestui software poate cauza o functionare eronata a hardware-ului, care resulta intr-un accident sau afectarea mediului inconjurator.
2) *Software critic secundar:* Software care poate provoca un accident in mod indirect. De exemplu software-ul folosit pentru un sistem care produce un obiect. Un functionare gresita a acestui sistem poate fabrica gresit obiectul, ce poate duce la un accident.

Increderea si siguranta unui sistem sunt legate intre ele insa un sistem de incredere poate fi nesigur si invers. Putem enumera 4 motive pentru care sistemele software de incredere nu sunt neaparat sigure:

1) Nu putem fi niciodata siguri ca un sistem este perfect. Probleme ce nu au aparut niciodata pot aparea dupa multi ani de operare fara greseala
2) Specificatiile pot fi incomplete si nu descriu comportamentul sistemului in anumite situatii critice.
3) Nefunctionarea corecta a hardware-ului poate cauza sistemul sa se comporte intr-un mod neasteptat si sa ofere software-ului un mediu de lucru neprevazut. Cand comportamentele sunt aproape de a cauza picarea sistemului, se pot comporta anormal si sa genereze semnale gresite care pot fi prelucrate software.
4) Operatorii unui sistem pot introduce instructiuni care sunt corecte, dar in anumite situatii sa duca la o functionare gresita a sistemului. Un exemplu real ar putea fi cand rotile unui avion s-au ridicat cand acesta era la sol. S-a descoperit ca un tehnician apasase aceasta comanda din greseala. Aceasta este o comanda valida insa nu ar trebui executata decat daca avionul este in aer.

Pentru a asigura siguranta este necesar sa prevenim accidentele sau ca impactele acestora sa fie minime. Acest lucru poate fi realizat in 3 moduri complementare:

1) *Evitarea hazardului:* De exemplu un sistem de taiere care necesita ca operatorul sa foloseasca ambele maini pentru a apasa o comanda pentru a evita ca mainile acestuia sa fie in calea dispozitivului de taiere
2) *Detectia si indepartarea hazardului:* Sistemul este conceput pentru ca hazardul sa fie detectat si indepartat inainte de intamplarea unui accident. De exemplu un sistem de prelucrare chimica, cand detecteaza o crestere a presiunii, sa deschida o valva pentru a preveni explozia.
3) *Reducerea impactului pentru accidente:* Sistemul trebuia sa includa mecanisme de protectie in caz de accident. Spre exemplu un motor de avion sa contina un sistem automat de extinctoare. Daca apare un incendiu, acesta poate fi controlat inainte sa prezinte un pericol major.

Accidentele apar de multe ori cand mai multe lucruri merg gresit la un moment dat. De exemplu, stricarea unui sistem de racire poate duce la supraincalzire care cauzeaza partile hardware sa transmita semnale gresite. Totusi, este imposibil sa prezicem toate combinatiile de erori care pot aparea asadar accidentele sunt o parte inevitabila a folosirii sistemelor complexe.

Securitatea

Securitatea este proprietatea unui sistem de a se proteja de atacurile externe, care pot fi accidentale si intentionate. Aceste atacuri externe sunt posibile deoarece majoritatea calculatoarelor de uz general sunt conectate la Internet si, deci, accesibile din exterior. Exemple de atacuri pot fi instalarea unor virusi, folosirea neautorizata a serviciilor sistemului sau modificarea neautorizata a datelor din sistem.

Pentru anumite sisteme securitatea este cel mai important aspect al acestuia. Sistemele militare, sistemele pentru comert electronic si sistemele care implica procesarea si schimbarea datelor confidentiale trebuie proiectate astfel incat sa atinga un nivel inalt de securitate.

In orice sistem legat la retea, exista 3 tipuri de atacuri de securitate:

1) *Atacuri la confidentialitatea sistemului si datele acestuia:* Acestea pot divulga informatii unor persoane sau programe care nu sunt autorizate sa aiba accesc la acestea
2) *Atacuri la integritatea sistemului:* Acestea pot distruge informatii din sistem
3) *Atacuri la disponibilitatea sistemului:* Acestea pot restrictiona accesul la sistem utilizatorilor autorizati

In practica, majoritatea vulnerabilitatilor rezulta din greseli ale oameni decat din probleme de natura tehnica. Utilizatorii aleg parole usor de ghicit sau le scriu in locuri unde pot fi gasite. Administratorii de sistem fac setari gresite legate de aplicatiile care pot fi instalate sau nu se foloseste software pentru protectie (anti-virus, firewall etc.).

Cateva metode pentru a creste securitatea sistemului sunt urmatoarele:

1) *Evitarea vulnerabilitatii:* Controale cu scopul de a asigura ca atacurile nu au succes. De exemplu, sistemele militare sensibile nu sunt conectate la retele publice, deci accesul din exterior este imposibil. O alta varianta este folosirea datelor criptate, astfel incat nu pot fi citite de cine reuseste sa patrunda in sistem.
2) *Detectia atacurilor si neutralizarea:* Se refera la controale care monitorizeaza activitatea sistemului si il verifica pentru comportamente anormale. Daca acest lucru se intampla atunci sistemul se restarteaza automat, interzice accesul anumitor utilizatori etc.
3) *Limitarea expunerilor si recuperare:* Sisteme care se recupereaza dupa anumite probleme. Aici putem include si polite de asigurare care recupereaza costurile unui sistem pierdut dupa un atac.

Fara un nivel rezonabil de securitate, nu putem fi siguri de disponibilitatea, increderea si siguranta unui sistem. Daca un sistem este atacat atunci increderea si siguranta sunt compromise.

Avantajele dezvoltarii si folosirii unui software securizat:

- **Protejarea datelor** organizatiei si/sau ale clientilor;
- **Minimizarea costurilor** in ceea ce proveste remedierea vulnerabilitatilor deosebite in timpul duratei de viata a sftware-ului;
- **Evitarea sanctiunilor comerciale sau legale** cauzate de atacuri;
- **Evitarea pierderii de imagine si de incredere a clientilor** cauzate de atacuri;
- **Cresterea calitatii** produselor software si **diferentierea de competitor.**

Activitati de securitate din SDLC

Etapa	Activitate
Analiza	Definirea obiectivelor si a cerintelor de securitate
Design	Proiectarea solutiei de securitat Analiza 'Threat model'
Implementare	Tehnica de programare securizata Revizuirea codului
Testare/Acceptare	Evaluarea securitatii Penetration testing: 'Ethical Hacking'
Mentenanta	Scanari periodice de vulnerabilitati Penetration testing: 'Ethical Hacking'

Intr-o acceptiune unitara, **ciclul de viata al dezvoltarii unui software (din engleza: Software Development LifeCycle – SDLC)** se refera la pasii structurali si metodologia folosite pentru dezvoltarea unui produs de tip software.

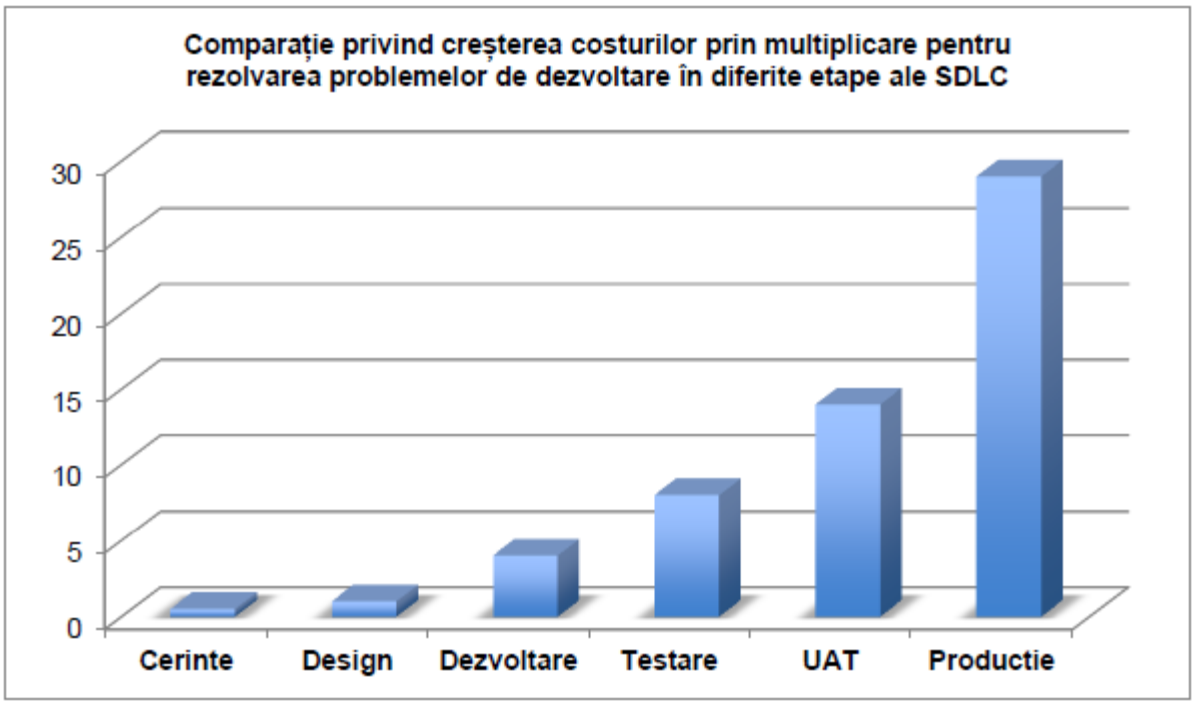
Datorita complexitatii tehnologice in crestere a sistemelor si proceselor implicate in SDLC, creste deopotriva si complexitatea vulnerabilitatilor disponibile la diverse niveluri ale unei aplicatii, care daca sunt exploatate, pot determina scaderea controalelor de securitate si la nivelul altor niveluri ale aplicatiei.

Securitatea in SDLC poate fi interpretata ca un factor de succes pentru business, mai degraba decat o bariera in cadrul realizarii proiectelor, in special pentru ca:

- activitatea de securitate este mult mai ieftina daca este planificata inca de la inceputul proiectului

- implementarea unor controale de securitate in sisteme la inceputul proiectului in loc sa fie implementate la sfarsit, poate creste performanta sistemelor per ansamblu

- se poate reduce necesitatea costisitoare de reevaluare in post-productie a unor controale si masuri necesare pentru scaderea riscurilor la un nivel acceptabil.



Managementul riscurilor este unul dintre aspectele cheie ale managementului securitatii si se inscrie in conceptele de baza de securitate cu care se poate lucra in SDLC.

Managementul riscurilor consta in evaluari preliminare asupra necesitatii controalelor de securitate, a identificarii, dezvoltarii, testarii, implementarii si verificarii controalelor de securitate, astfel incat niciun proces initiat in scopuri distructive se va incadra intr-un nivel acceptabil de risc.

In sistemul de management al securitatii se inscriu si aspectele legate de guvernanta de securitate: politici, norme, standarde si proceduri de securitate.

Concepte esentiale	- Confidentaialitate - Integritate - Disponibiliate
Concepte generale	- Autentificarea - Autorizarea - Audit si jurnalizare - Managementul sesiunilor - Managementul erorilor si exceptiilor
Concepte de design	- Separarea drepturilor - Securitatea pe niveluri - Fail secure - Open design - Mecanisme de least common - Legatrui intre componentele existente

Concluzie:

Securitatea informatiei are un rol foarte important in fiecare etapa din ciclul de dezvoltare a unui produs software. Lipsa implicarii specialistilor in securitate sau utilizarii principiilor de securitate in fiecare etapa de dezvoltare poate duce la aparitia de vulnerabilitati in cadrul produsului si chiar anularea eforturilor depuse in celelalte etape de dezvoltare.

Bibliografie:

- Software Engineering – Ninth Edition – Ian Sommerville
- http://www.clubafaceri.ro/30747/analiza-vulnerabilitati-si-evenimente-de-securitate-209332.html
- http://www.eyenet.ro/securitate-it.html
- http://ro.wikipedia.org/wiki/Fiabilitatea_securit%C4%83%C8%9Bii_informatice

Specificatiile de siguranta si securitate

- **Obiective:**

  Scopul acestui capitol este de a explica cum trebuie sa specificam cerintele de securitate si de siguranta functionala si nefunctionala. Dupa parcurgerea acestui capitol veti putea:

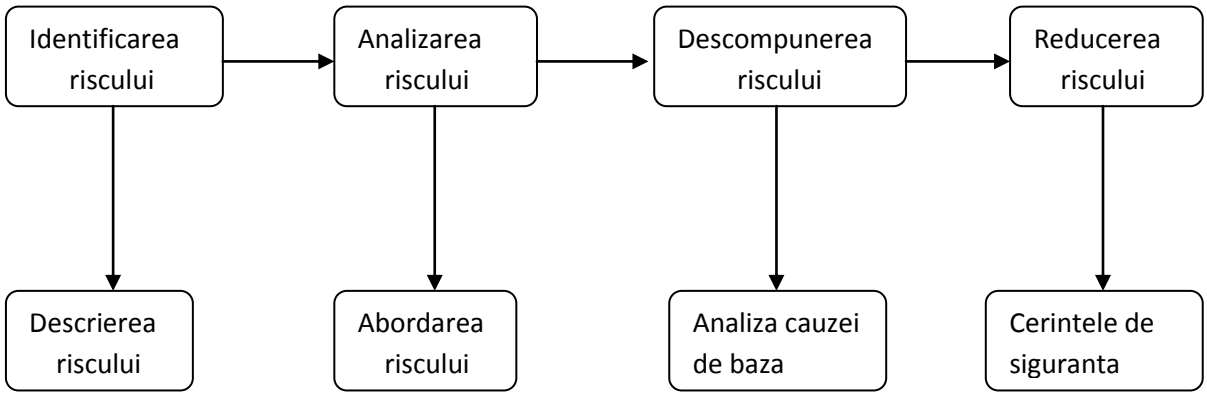
  - Intelege cum o abordare riscanta poate fi folosita pentru identificarea si analizarea cerintelor de siguranta, incredere si securitate
  - Cunoaste diferite tipuri de cerinte de securitate care pot fi necesare intr-un sistem complex
  - Fi constienti de avantajele si dezavantajele folosirii specificatiilor formale, matematice a unui sistem

Siguranta unui sistem nu depinde doar de o inginerie buna. Necesita, de asemenea, atentie la detalii cand cerintele sistemului sunt implicate si includerea unor cerinte software speciale care sunt asociate pentru a asigurata siguranta si securitatea unui sistem. Aceste cerinte de siguranta si securitate sunt de doua tipuri:

1) *Cerinte functionale:* Definesc facilitati de verificare si recuperare care ar trebui incluse in sistem si facilitati care asigura protectia impotriva problemelor din sistem si atacurilor din exterior
2) *Cerinte nefunctionale:* Definesc increderea si disponibilitatea necesara a sistemului

Specificarea cerintelor datorate riscului

Cerintele pentru siguranta si securitate pot fi gandite ca cerinte de protectie. Acestea specifica cum un sistem ar trebui sa se protejeze de greselile interne, sa opreasca problemele in sistem care pot dauna mediul, sa opreasca accidentele si atacurile mediului in care se afla sistemul asupra sistemului si sa faciliteze recuperarea in cazul unei probleme in sistem. Pentru a descoperi aceste cerinte de protectie trebuie sa intelegem riscurile sistemului si ale mediului asupra lui.



Specificarea riscurilor este o abordare des folosita de dezvoltatorii unor sisteme criticle din punct de vedere are securitatii si sigurantei. Se concentreaza asupra acelor evenimente care pot cauza cele mai mari daune si care sunt cel mai putin probabile sa se intample. Un asemenea proces de specificare a riscurilor, ilustrat in figura de mai sus, implica intelegerea riscurilor la care este expus sistemul, descoperirea cauzelor si generarea cerintelor care sa poata face fata acestor riscuri. Etapele acestui proces sunt:

1) *Identificarea riscului:* Potentialele riscuri asupra sistemului sunt identificate. Acestea sunt dependente de mediul in care sistemul urmeaza a fi folosit. Riscurile pot aparea din interactiuni intre sistem si conditii rare in mediul de operare
2) *Analiza riscului si clasificarea:* Fiecare risc este tratat separat. Acelea care sunt serioase si plauzibile sunt selectate pentru o analiza ulterioara. In aceasta etapa, riscurile pot fi eliminate deoarece este putin probabil ca acestea sa apara sau pentru ca nu pot fi detectate de catre software
3) *Descompunerea riscului:* Fiecare risc este analizat pentru a descoperi potentialele cauze de baza. Acestea pot fi vulnerabilitati inerente ale hardwareului si softwareului care rezulta din deciziile dezvoltatorilor.
4) *Reducerea riscului:* Sunt facute propuneri pentru moduri in care riscurile identificate pot fi reduse sau eliminate. Acestea contribuie la cerintele de siguranta ale sistemului care definesc metodele de aparare asupra riscului sau cum ar trebui riscul controlat.

Pentru sistemele largi, analiza riscului poate fi structurata in mai multe faze unde fiecare faza considera fiecare tip de risc:

1) *Analiza preliminara a riscului,* unde este identificat fiecare risc major din mediul de operare al sistemului. Acestea sunt independente de tehnologia folosita pentru

dezvoltarea sistemului. Scopul analizei preliminare este de a dezvolta un set initial de cerinte de securitate si siguranta pentru sistem.

2) *Analiza ciclului de viata a riscului,* care are loc in decursul dezvoltarii sistemlui si care se preocupa de riscurile care apar in cadrul deciziilor de design a sistemului. Diverse tehnologii si arhitecturi de sistem au propriile riscuri asociate. In aceasta etapa, trebuie extinse cerintele pentru protectia asupra acestor riscuri.
3) *Analiza riscului operational,* care se preocupa cu interfata sistemului catre utilizator si riscurile cauzate de greseli ale operatorului. Din nou, deciziile o data luate asupra designului interfetei catre utilizator, pot fi facute modificari ulterioare asupra cerintelor de protectie.

Aceste etape sunt necesare deoarece este imposibil sa luam toate deciziile legate de securitate si siguranta fara informatii complete despre implementarile sistemului. Cerinte de securitate pot necesita modificari deoarece intra in conflict cu optiuni de securitate furnizate de un sistem in stoc.

De exemplu, o cerinta de securitate ar fi ca utilizatorii sa se identifice intr-un sistem cu o fraza de acces in locul unei parole. Acestea sunt mai greu de ghicit pentru un atacator. Totusi, daca se ia decizia de a folosi un sistem existent care foloseste autentificare doar pe baza unei parole, atunci acest sistem de securitate nu poate fi implementat. Asadar, poate fi necesar sa includem alte functii in sistem care sa compenseze pentru riscul crescut de folosire a unei parola in locul unei fraze de acces.

Specificarea sigurantei

Sistemele critice din punct de vedere al sigurantei sunt sistemele in care problemele pot afecta mediul de lucru si pot cauza accidente oamenilor din acel mediu. Principala problema a specificarii sigurantei este de a identifica cerintele care vor minimiza probabilitatea ca acele probleme in sistem sa apara. Cerintele de securitate sunt in primul rand cerinte de protectie si nu au legatura cu operarea normala a sistemului. Totusi, nu are rost in a construi un sistem foarte sigur daca nu este eficient din punct de vedere al costurilor.

Identificarea situatiilor neprevazute (hazardului)

In sistemele critice, principalele riscuri apar din cauza situatiilor neprevazute care pot duce la accidente. Putem analiza problema hazardului considerand diferite tipuri, cum ar fi de natura fizica, electrica, biologica etc. Fiecare din aceste clase poate fi analizata ulterior pentru a

descoperi diferitele situatii neprevazute care pot aparea, la fel si posibile combinatii intre acestea care prezinta un potential pericol.

Un inginer experimentat, lucrand cu specialisti in domeniu identifica situatiile neprevazute din experienta anterioara si din analiza domeniul aplicativ al sistemului.

Evenimentele de natura software se refera la esecul de a livra un serviciu sau esecul monitorizarii si sistemelor de protectie. Monitorizarea si sistemele de protectie sunt incluse intr-un dispozitiv pentru a detecta conditiile de lucru, cum ar fi nivelul bateriei scazut ce poate duce la picarea sistemului.

Abordarea situatiilor neprevazute (hazardului)

Procesul de abordare a situatiilor neprevazute are ca obiectiv intelegerea probabilitatii ca o situatie neprevazuta sa apara si consecintele acestei intamplari sau a unei intamplari asociate.

Pentru fiecare astfel de situatie, rezultatul analizei si procesului de clasificare este exprimat sub forma unor riscuri, acesta luand in considerare posibilitatea unui accident si consecintele sale. Putem enumera trei categorii de risc care pot fi folosite in abordarea situatiilor neprevazute:

1) Riscurile intolerabile in sistemele critice sunt cele care pun in pericol viata. Sistemul trebuie sa fie conceput astfel incat situatiile neplacute sa nu apara sau, daca apar, sa ne asiguram ca sunt detectate inainte de a cauza un accident.
2) Riscurile rezonabil practice sunt cele care sunt mai putin serioase sau au o probabilitate mica de aparitie. Sistemul trebuie conceput astfel incat probabilitatea unui accident sa fie minimizata, luand in considerare si costurile.
3) Riscurile acceptabile sunt cele in care accidentele asociata produc pagube minore. Dezvoltatorii ar trebui sa ia in calcul aceste riscuri atat timp cand nu cresc costurile, timpul de raspuns sau alte atribute non-functionale.

Desi costurile financiare pentru acceptarea riscurilor si plata pentru accidentele rezultate pot fi mai mici decat costurile pentru prevenirea lor, opinia publica poate cere investitii in prevenirea accidentelor.

Spre exemplu, este mai ieftin pentru o companie sa curete deseurile in loc sa instaleze sisteme care sa previna aparitia acestora. Totusi publicul si presa nu tolereaza asemenea accidente, si remedierea dupa aparitia unei probleme nu este acceptabila. Asemenea evenimente pot duce la o reclasificare a riscurilor

Abordarea situatiilor neplacute implica estimarea probabilitatii si severitatea acestora. Acest lucru este destul de dificil deoarece este posibil ca inginerii implicati sa nu fi avut experienta cu

tipul de problema aparuta. Probabilitatile si severitatea sunt asociate folosind termeni ca: “probabil”, “putin probabil” si “rar”, respectiv “mare”, “medie” si “scazuta”.

Sistemul de monitorizare hardware ar trebui sa monitorizeze starea sistemului si sa avertizeze potentiale probleme. Aceste avertizari permit detectarea situatiilor neprevazute inainte de a cauza un accident. Ca exemplu putem da inchiderea sistemului, care e detectat monitorizand starea bateriei.

Software-ul de monitorizare este, bineinteles, legat de siguranta. Esecul in detectarea unui eveniment poate duce la un accident. Daca sistemul de monitorizare da gres dar hardware-ul functioneaza corect, atunci nu este o problema serioasa. Totusi, daca sistemul de monitorizare da gres iar problema hardware nu poate fi detectata, acest lucru poate avea consecinte grave.

Analiza situatiilor neprevazute (hazardului)

Analiza situatiilor neprevazute este procesul de descoperire a cauzelor de baza a acestora in sistemele critice din punct de vedere al sigurantei. Scopul unui inginer este de a gasi acele evenimente sau combinatii de evenimente care pot cauza probleme in sistem. Pentru acest lucru, putem folosi o abordare “varf-baza” sau “baza-varf”. Deductiv, tehnica “varf-baza”, care tinde sa fie mai usor de folosit, incepe cu evenimentul neprevazut si ajunge la posibilele probleme in sistem. Tehnica “baza-varf” pleaca de la problemele propuse in sistem si identifica eventualele situatii care pot aparea de la acestea.

O tehnica des folosita pentru abordarea situatiilor neprevazute o reprezinta realizarea unui “arbore al evenimentelor”. Aceasta tehnica este usor de inteleg fara cunostiinte de specialitate in domeniu. Pentru a realiza un astfel de arbore plecam de la evenimentele deja identificate. Pentru fiecare eveniment, mergem inapoi pentru a descoperi posibilele cauze. Evenimentul trebuie pus la baza arborelui si identificat starile sistemului care pot duce la acesta. Pentru fiecare din aceste stari, identificam stari ulterioare care duc la ele. Descompunem pana cand ajungem la cauza de baza a riscului. Evenimentele care pot aparea numai din combinatii ale acestor cauze au o probabililtate scazuta sa duca la un accident decat evenimentele cu o singura cauza de baza.

“Arborele evenimentelor” este folosit, de asemenea, pentru a identifica potentialele probleme hardware. Acesta poate oferi informatii despre cerinte software pentru a fi detectate, si eventual corectate.

Reducerea riscului

O data ce potentialele riscuri si cauzele lor de baza au fost identificate, putem enunta cerintele de siguranta care se ocupa de riscuri si asigura faptul ca nu apar incidente sau accidente. Exista 3 strategii posibile care se pot folosi:

1) *Evitarea situatiilor neprevazute:* Sistemul este conceput astfel incat sa nu apara situatii neprevazute
2) *Detectarea si indepartarea situatiilor neprevazute:* Sistemul este conceput astfel incat situatiile neplacute sunt detectate si neutralizate inainte sa duca la un accident
3) *Limitarea impactului:* Sistemul este conceput astfel incat consecintele unui accident sunt minimizate

In mod normal, dezvoltatorii sistemelor critice folosesc o combinatie a acestor abordari. Intr-un sistem critic din punct de vedere al sigurantei, situatiile neprevazute (hazardul) pot fi abordate prin minimizarea probabilitatii de aparitie si adaugarea unui sistem de protectie care ofera un sistem de rezerva (backup). Spre exemplu, intr-un sistem de procesare chimica dintr-o fabrica, sistemul va incerca sa detecteze si sa evite excesul de presiune din reactor. Totusi, poate fi si un sistem de protectie independent care monitorizeaza presiunea si deschide in valva de aerisire daca este detectata presiune ridicata.

Specificarea increderii

Increderea unui sistem depinde de increderea hardware, software si a operatorilor sistemului. Sistemul software trebuie sa ia acest lucru in considerare. La fel ca includerea cerintelor care compenseaza pentru picarea softwareului, trebuie sa exista cerinte de incredere care ajuta la detectarea si recuperarea dupa picarile hardware si erorile operatorilor.

Increderea este diferinta de siguranta si securitate in sensul ca este un atribut al sistemului masurabil. Asadar, este posibil sa specificam nivelul de incredere necesar, monitorizand operatiile sistemului in timp, si apoi sa verificam daca s-a atins nivelul de incredere cerut. Spre exemplu, o cerinta de incredere poate fi aceea ca problemele in sistem care necesita restartarea nu ar trebui sa apara mai mult de o data pe saptamana. De fiecare data cand apare o problema, poate fi analizata si putem verifica daca s-a atins nivelul cerut de incredere. Daca nu, putem modifica cerintele de incredere sau sa solicitam o schimbare a sistemului subliniind problemele sistemului. Putem decide, apoi, sa acceptam un nivel scazut al increderii deoarece

costurile de schimbare a sistemului sunt mari sau remedierea problemei ar putea avea efecte adverse, cum ar fi performanta scazuta sau timp de raspuns.

In schimb, siguranta si securitatea se refera la evitarea situatiilor nedorite decat la specificarea unui nivel dorit de siguranta si securitate. Chiar si o asemenea situatie intr-un ciclu de viata a unui sistem este inacceptabila si, daca apare, trebuie efectuate schimbari in sistem. Nu putem afirma “problemele sistemului ar trebui sa rezulte in mai putin de 10 accidente pe an”; de indata ce apare un incident, problema din sistem trebuie remediata.

Cerintele de incredere sunt, asadar, de doua feluri:

1) Cerinte nefunctionale, care definesc numarul problemelor acceptabile in timpul functionarii normale a sistemului , sau in timpul in care sistemului nu este utilizabil. Acestea reprezinta cerinte de incredere cantitative.
2) Cerinte functionale, care definesc sistemul si functiile software care evita, detecteaza sau tolereaza problemele in software si, deci, asigura ca aceste probleme nu duc la cedarea sistemului

Cerintele de incredere cantitative duc la cerinte functionale de sistem. Pentru a atinge un nivel cerut in incredere, cerintele functionale si de design a sistemului ar trebui sa specifice problemele ce trebuie sa fie detectate si actiunile care ar trebui sa fie luate pentru a asigura ca aceste probleme nu duc la cedarea sistemului.

Procesul de specificare a increderii poate fi bazat pe un proces de specificare a riscurilor:

1) *Identificarea riscului:* In aceasta parte, trebuie identificate tipurile de probleme in sistem care pot duce la pierderi economice. Spre exemplu, un sistem de comert electronic poate fi indisponibil astfel incat clientii nu pot efectua comenzi.
2) *Analiza riscului:* Acest lucru implica estimarea costurilor si consecintelor diferitelor tipuri de probleme software si selectarea problemelor cu consecinte grave pentru o analiza ulterioara.
3) *Descompunerea riscului:* In aceasta etapa, facem o analiza a cauzei radacina pentru problemele serioase si foarte probabile. Totusi, acest lucru poate fi imposibil deoarece cauzele de baza pot depinde de deciziile de proiectare a sistemului.
4) *Reducerea riscului:* In aceasta etapa, trebuie generate specificatii de incredere cantitative care stabilesc probabilitatile acceptabile a diferitelor tipuri de probleme. Acestea trebuie sa ia in calcul si costurile problemelor. Putem folosi diverse probabilitati pentru diferite servicii ale sistemului. Putem genera, de asemenea, cerinte functionale de incredere. Din nou, acest lucru poate fi efectuat dupa ce s-au lua deciziile in legatura cu proiectarea sistemului.

Calculul increderii

In termeni generali, increderea poate fi exprimata ca probabilitatea ca o problema in sistem sa apara cand un sistem este in folosinta intr-un mediu de operare specificat. Daca afirmam, spre exemplu, ca 1 din 1000 de tranzactii va esua, atunci putem specificarea probabilitatea de esec ca fiind 0.001.

Exista doua metrici importante folosite pentru a specifica increderea alatura de o metrica aditionala care este folosita pentru a specifica disponibilitatea sistemului. Alegerea metricii depinde de tipul de sistem care este specificat si cerintele domeniului de aplicatie. Metricile sunt:

1) *Probabilitatea de cedare la cerere(Probability of failure on demand-POFOD):* Daca folosim aceasta metrica, definim probabilitatea ca cererea unui serviciu de la sistem sa rezulte intr-o picare a sistemului. Deci daca POFOD=0.001 inseamna ca exista 1/1000 sanse ca sa apara o problema cand se efectueaza o cerere.
2) *Rata de aparitie a problemelor(Rate of occurence of failures-ROCOF):* Aceasta metrica seteaza numarul probabil de cedari ale sistemului care poate fi observat raportat la un interval de timp sau la un numar de executii in sistem. In exemplul de mai sus, ROCOF este 1/1000. Reciproc se foloseste timpul mediu al cedarii (mean time of failure-MTTF), care este uneori folosit ca metrica pentru incredere. Asadar un ROCOF de 2 cedari pe ora implica MTTF=30 minute.
3) *Disponibilitatea(Availability-AVAIL):* Disponibilitatea unui sistem reflecta capacitatea acestuia de a livra servicii cand este solicitat. AVAIL este probabilitatea ca un sistem sa fie operational cand se efectueaza o cererea asupra unui serviciu. Deci, daca AVAIL=0.9999, inseamna ca, in medie, un sistem va fi disponibil 99.99% din timpul de operare.

Pentru a aproxima increderea unui sistem, trebuie sa retin date despre operarea acestui. Aceste date pot include:

1) Numarul de picari ale sistemului fiind dat un numar de cereri asupra serviciilor sistemului. Acestea sunt folosite pentru a calcula POFOD.
2) Timpul sau numarul de tranzactii intre picarile sistemului plus timpul total trecut sau numarul total de tranzactii. Acestea sunt folosite pentru a calculca ROCOF sau MTTF.
3) Timpul de reparatie sau de restartare dupa ce o picare a sistemului a dus la pierderea serviciului. Acesta este folosit pentru masurarea disponibilitatii. Disponibilitatea nu depinde doar de timpul intre cedari ale sistemului, ci si de timpul trecut cat se aduce sistemul in stadiu functional.

Cerinte de incredere non-functionale

Cerintele de incredere non-functionale sunt specificatii cantitative despre increderea si disponibilitatea necesare unui sistem, calculate folosind una din metricile de mai sus. Increderea cantitativa si specificarea disponibilitatii sunt folosite de multi ani in sistemele critice din punct de vedere al sigurantei dar sunt rar folosite sistemele critice pentru afaceri. Totusi, din ce in ce mai multe companii cer servicu 24/7 pentru sistemele lor si este posibil ca aceste tehnici sa fie folosite mai des. Exista mai multe avantaje in folosirea specificatiilor cantitative de incredere:

1) Procesul de decidere a nivelului increderii necesar ajuta la clarificarea necesitatilor actionarilor, ajutandu-i sa inteleaga ca exista diverse tipuri de probleme in sistem, iar nivele mari de incredere presupun costuri ridicate.
2) Asigura o baza pentru a intelege cand sa oprim testarea unui sistem. Acest lucru se intampla cand se atinge nivelul cerut de incredere.
3) Este un mod de a aborda diverse strategii de proiectare cu intentia de a imbunatati increderea unui sistem.
4) Daca sistemul trebuie aprobat pentru a intra in serviciu, dovada faptului ca s-a atins in nivel de incredere cerut este important pentru certificarea sistemului.

Pentru a stabili un nivel cerut de incredere a sistemului, trebuie sa luam in considerare pierderile asociate care duc la picarea sistemului. Acestea nu sunt simple pierderi financiare, ci si de reputatie pentru o afacere. Spre exemplu daca un utilizator incearca sa acceseze un site de comert electronic si observa ca acesta nu este disponibil, se va orienta catre alt site in loc sa astepte remedierea problemei. Daca acest lucru se intampla mai mult de o data, exista probabilitatea pierderii respectivului client.

Exista un numar de pasi care pot fi urmati pentru a evita supraspecificarea increderii sistemului:

1) Specificarea cerintelor disponibilitatii si increderii pentru diverse tipuri de probleme. Poate fi o probabilitatea scazuta a erorilor serioase decat cele minore.
2) Specificarea cerintelor disponibilitatii si increderii pentru fiecare serviciu in parte. Problemele care afecteaza serviciile cele mai critice ar trebui specificate ca putin probabile decat cele doar cu efecte locale. Putem decide apoi limitarea specificarii increderii cantitative pentru cele mai critice servicii ale sistemului.
3) Deciderea daca chair avem nevoie de incredere mare intr-un sistem software sau daca disponibilitatea generala a sistemului poate fi atinsa prin alte moduri. Spre exemplu, putem folosi mecanisme de detectie a erorilor pentru a verifica iesirile sistemului si sa avem procese in locul acestora pentru corectia erorilor. Ca urmare, nu este necesar un nivel inalt de incredere intr-un sistem care genereaza iesirile corespunzatoare.

Pentru a exemplifica ultimul punct, consideram cerintele de incredere pentru un ATM bancar care ofera monetar si ofera alte servicii clientilor. Daca exista probleme hardware sau software, atunci acestea duc la intrari gresite in baza de date a conturilor clientilor. Acestea pot fi evitate prin specificarea unui nivel inalt al increderii hardware si software al ATM-ului. Pentru a specifica disponibilitatea unei retea de ATM-uri, trebuie identificare serviciile sistemului si sa specificam disponbilitatea ceruta pentru fiecare din acestea:

- servicul care cuprinde baza de date cu conturile clientilor
- servicii individuale oferite de ATM ca “scoaterea banilor”, “oferirea informatiilor despre cont”, etc.

Specificarea increderii functionale

Specificarea increderii functionale implica identificarea cerintelor care definesc constrangeri si facilitati care contribuie la increderea sistemului. Pentru sistemele in care increderea a fost cantitativ specificata, aceste cerinte functionale pot fi necesare pentru a asigura ca un nivel de incredere cerut a fost atins.

Exista 3 tipuri de cerinte functionale de incredere pentru un sistem:

1) *Cerinte de verificare:* Acestea identificare faptul ca se verifica intrarile din sistem pentru a asigura ca intrarile incorecte sunt detectate inainte de a fi procesate de sistem.
2) *Cerinte de recuperare:* Acestea sunt adaugate pentru a ajuta sistemul sa se recupereze dupa ce a aparut o problema. Tipic, aceste cerinte se ocupa cu mentinerea copiilor sistemului si datele acestuia si specificarea cum sa recupereze serviciile sistemului dupa o cedare.
3) *Cerinte de redundanta:* Acestea specifica facilitatile de redundanta a sistemului care asigura faptul ca cedarea unei singure componente nu duce la o pierdere completa a serviciului.

Nu exista regula simple pentru aflarea cerintelor functionale de incredere. In organizatiile care dezvolta sisteme critice, exista de obicei cunostiinte despre posibilele cerinte de incredere si cum acestea afecteaza increderea efectiva a unui sistem. Aceste organizatii sunt specializate in tipuri specifice de sisteme, asadar cerintele de incredere pot fi refolosite asupra unor sisteme asemanatoare.

Specificarea securitatii

Specificarea cerintelor de securitate dintr-un sistem este asemanatoare cu cerintele de siguranta. Totusi, securitatea este o problema mai delicata decat siguranta deoarece:

1) Cand consideram siguranta, trebuie sa presupunem ca mediul in care sistemul este instalat nu este ostil. Cand consideram securitatea, trebuie sa asumam ca atacurile asupra sistemului sunt intentionate si atacatorul are cunostiinte despre slabiciunile sistemului.
2) Cand apar probleme ale sistemului care presupun riscuri din punct de vedere al sigurantei, cautam erori care au cauzat problema. Cand atacuri intentionate cauzeaza probleme in sistem, gasirea cauzei de baza poate fi mai dificila intrucat atacatorul poate incerca sa anuleze cauza problemei.
3) Este de obicei acceptabil sa inchidem un sistem sau sa oprim serviciile pentru a evita o problema din punct de vedere al sigurantei. Totusi, atacurile asupra unui sistem pot fi intentionate sa-l inchida, fapt care ar insemna ca atacul a avut succes.
4) Evenimentele legate de siguranta nu sunt generate de un adversar inteligent. Un atacator poate proba sistemul de aparare intr-o serie de atacuri, modificand atacurile pe masura ce invata mai mult despre sistem si raspunsul acestuia.

Aceste diferente inseamna ca cerintele de securitate trebuie sa fie mai ridicate decat cele de siguranta. Exista mai multe tipuri de cerinte de securitate care acopera mai multe amenintari asupra sistemului. Firesmith (2003) a identificat 10 tipuri de cerinte de securitate care pot fi incluse in specificatiile unui sistem:

1) Cerinte de identificare care specifica daca un sistem trebuie sau nu sa-si identifice utilizatorii inainte de a interactiona cu acestia
2) Cerintele de autentificare specifica cum sunt identificati utilizatorii
3) Cerintele de autorizare specifica privilegiile si permisiunile de acces ale utilizatorilor identificati
4) Cerintele de imunitate specifica cum sistemul trebuia sa se protejeze de virusi, trojeni si alte amenintari similare.
5) Cerintele de integritate specifica cum ar trebui evitate coruperea datelor
6) Cerintele de detectie ale intrusilor specifica ce mecanisme ar trebui utilizate pentru a detecta atacuri asupra sistemului
7) Cerintele de recunoastere specifica faptul ca un membru al unei tranzactii nu poate nerecunoaste implicarea acestuia in acea tranzactie
8) Cerintele de intimitate specifica cum ar trebui retinute datele personale
9) Cerintele de verificare a securitatii specifica cum un sistem poate fi verificat si audiat

10) Cerintele de securitate a intretinerii sistemului specifica cum o aplicatie poate preveni schimbari autorizate de apararea mecanismelor securitate

Bineinteles, nu vom intalni toate aceste tipuri de cerinte de securitate in orice sistem. Acestea depind de tipul sistemului, situatia in care este folosit si utilizatorii presupusi.

Un punct important in abordarea riscului si procesului de management este polita de securitate a organizatiei. Aceasta se aplica tuturor sistemelor si ar trebui sa specifice ce este si ce nu este permis. Spre exemplu, un aspect al securitatii militare poate afirma “Cititorii pot examina doar documentele a caror clasificare este aceeasi sau sub nivelul de verificare a cititorului”. Acest lucru inseamna ca un cititor avand nivelul de verificare “secret”, poate accesa doar documente clasate ca “secrete”, “confidentiale” sau “dechise”, nu si documente de tip “top secret”

Polita de securitate contine conditiile care ar trebui mentinute intotdeauna de un sistem de securitate si, deci, ajuta la identificarea amenintarilor care pot aparea. Amenintarile reprezinta orice ar putea putea in pericol securitatea afacerii. In practica, politele de securitate sunt de obicei documente care definesc ce este si ce nu este permis. Totusi, pot exista generari automate de verificari care asigura ca polita este respectata.

Specificarea formala

Pentru mai mult de 30 de ani, multi cercetatori au sustinut folosirea metodelor formale de dezvoltare software. Metodele formale sunt abordari matematice a dezvoltarii software unde se defineste un model formal al softwareului. Putem analiza, apoi, acest model formal si a-l folosi ca o baza pentru specificarea formala ala sistemului. In principiu, este posibil sa pornim cu un model formal pentru software si sa demonstram ca un program dezvoltat este consistent cu acel model, astfel eliminand problemele software rezultate din erori de programare.

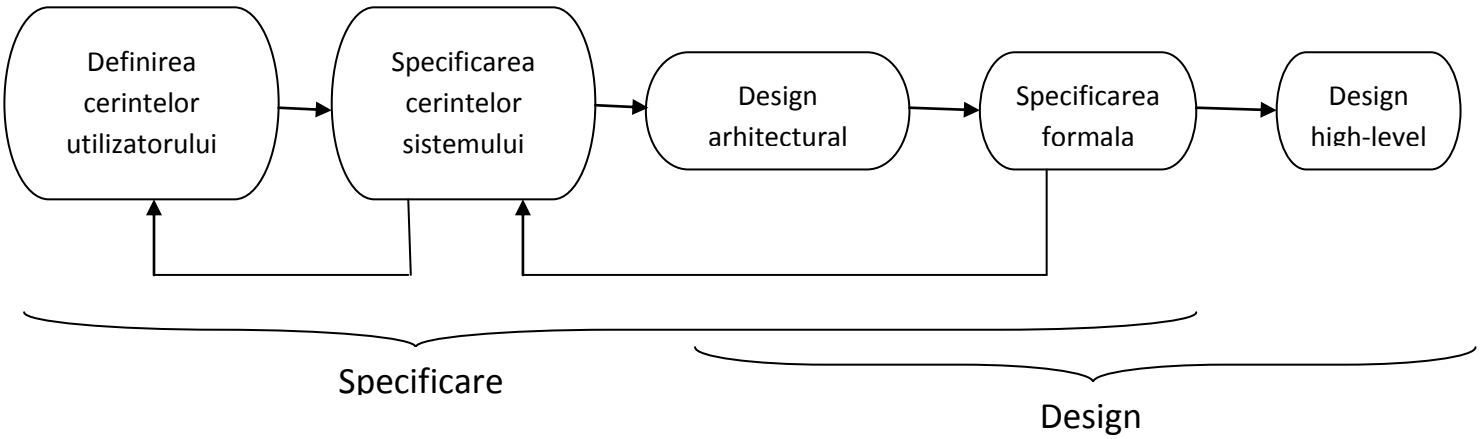
Punctul de start pentru toate procesele de dezvoltare formale este un model formal al sistemului, care serveste ca specificatiile sistemului. Pentru a crea acest model, trebuie translatate cerintele utilizatorului, exprimate in limbaj natural, diagrame si tabele, intr-un limbaj matematic cu anumite sintaxe. Specificarea formala este o descriere precisa a ceea ce trebuie sa faca un sistem. Folosirea manualului putem verifica daca comportamentul unui program este in concordanta cu specificatiile.

Specificatiile formale nu sunt esentiale doar pentru verificarea proiectarii si implementarii software. Ele reprezinta cel mai precis mod de specificare a sistemelor si de a reduce

neintelegerile. Mai mult, alcatuirea unei specificari formale ofera o analiza detaliata a cerintelor si este un mod eficient de a descoperi problemele din sistem.

Specificatiile formale sunt de obicei dezvoltate ca plan al unui proces software planificat, unde sistemul este complet specificat inaintea dezvoltarii. Cerintele de sistem si design sunt definite in detaliu si sunt atent analizate si verificate inaintea inceperii implementarii.

Figura de mai jos arata stagiile specificarii software si interfata cu designul software intr-un proces software planificat. Dezvoltarea specificatiilor formale fiind costisitoare, putem decide limitarea folosirii acestei abordari asupra acelor componente critice in operarea sistemului. Acestea sunt identificate in designul arhitectural al sistemului.



Avantajele dezvoltarii unei specificatii formale si folosirea acesteia intr-un proces formal de dezvoltare sunt:

1) Pe masura ce dezvoltam specificatiile formale in detaliu, dezvoltam si o intelegerea detaliata a cerintelor sistemului. Chiar daca nu folosim specificarea intr-un proces formal de dezvoltare, detectarea erorilor din cerinte este un procedeu bun pentru dezvoltarea unei specificatii formale. Problemele din cerinte care sunt descoperite devreme sunt mai usor de corectat decat daca sunt gasite intr-un stadiu avansat al procesului de dezvoltare
2) Specificatii fiind exprimate intr-un limbaj cu semantici definite, il putem analiza automat pentru a descoperi goluri si inconsistente
3) Costurile de testarea a programului pot fi reduse deoarece am verificat programul in concordanta cu specificatiile acestuia

In ciuda acestora avantaje, metodele formale au avut un impact limitat asupra dezvoltarii practice a softwareului, chiar si pentru sisteme critice. Dezavantajele dezvoltarii unei specificatii formale a sistemului sunt:

1) Expertii in domeniu nu pot intelege specificarea formala deci nu pot verifica daca aceasta reprezinta cu acuratete cerintele. Inginerii software, care inteleg specificarea formala, pot sa nu inteleaga domeniul de aplicare deci nu pot fi siguri ca specificarea formala este o reflexie corecta a cerintelor sistemului
2) Este foarte usor sa cuantificam costurile creearii unei specificari formale, dar mai dificil de a estima reducerile de cost care ar rezulta din folosirea acesteia. In consecinta, managerii nu doresc sa riste folosind aceasta abordare
3) Majoritatea inginerilor de sistem nu au fost instruiti sa foloseasca limbaje de specificare formala. Ca urmare, nu isi propun folosirea acesteia in procesul de dezvoltare.
4) Este dificila scalarea abordarii actuale a specificarii formale pentru sisteme foarte mari. Cand specificarea formala este folosita, se refera doar la partea de baza a softwareului decat la sisteme complete.

Bibliografie:

- Software Engineering – Ninth Edition – Ian Sommerville
- http://ro.wikipedia.org/wiki/Fiabilitatea_securit%C4%83%C8%9Bii_informatice
- http://www.ntech.ro/solutii-securitate-it
- http://www.securitatea-informatiilor.ro/solutii-de-securizare/asigurarea-securitatii-informatiilor-128.html
- http://www.elearningpapers.eu/ro/paper/securitate-informatic-i-educa-ie

Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Ciobanica Radu

Ingineria Securitatii

➢ Introducere:

Ingineria securității se ocupă de dezvoltarea și evoluția sistemelor care pot rezista atacurilor malware, destinate deteriorării sistemului sau a datelor acestuia.

Ingineria securității software-ului este o parte a domeniului mai general al securității computer-ului.Acest lucru a devenit o prioritate pentru întreprinderi și persoane fizice, datorită faptului că numărul infractorilor ce încearcă sa exploateze sistemele de rețea în scopuri ilegale este în creștere.

Inginerii de software ar trebui să fie conștienți de amenințările la adresa securității cu care se confruntă sistemele și de modalitățile de neutralizare ale atacurilor.

În prezent, proiectarea sistemelor ia în calcul gradul ridicat de imunitate la atacurile provenite din exterior, dar și modalități de back-up survenit după atacurile ce trec de bariera de securitate.Fără aceste măsuri de precauție, este aproape inevitabil ca atacatorii să nu compromită un sistem de rețea. De-a lungul timpului, a fost dovedită dexteritatea acestora în a fura date confidențiale din sistemele slab protejate sau a perturba serviciile oferite de aceste sisteme. Prin urmare, ingineria sistemelor de securitate este un aspect tot mai important al procesului de inginerie a sistemelor.

Majoritatea atacurilor externe se concentrează asupra infrastructurilor de sistem, deoarece componentele infrastructurii(de exemplu, broserele web) sunt bine cunoscute și disponibile pe scară largă.

Atacatorii pot sonda aceste sisteme pentru sectoare vulnerabile și pot, în același timp, partaja informații despre slăbiciunile descoperite. Cu cât mai mulți utilizatori folosesc același software, cu atât mai largă devine aplicabilitatea atacutilor.

Vulnerabilitățile infrastructurii pot permite atacatorilor accesul în sistem, ceea ce duce la expunerea datelor conținute de acesta. În practică, însă, există o distincție importantă între securitatea aplicațiilor și securitatea infrastructurii.Astfel, securitatea aplicațiilor este o problemă adresată inginerilor de software, care ar trebui să se asigure că sistemul este proiectat în așa fel încât să reziste atacurilor.

Securitatea infrastructurii este o problemă de management adresată managerilor de sistem care trebuie să configureze infrastructura pentru a rezista atacurilor. Managerii de sistem trebuie să optimizeze infrastructura pentru ca aceasta să permită utilizarea cât mai eficientă, indiferent de caracteristicile disponibile. Ei au, de asemenea, responsabilitatea de

a repara vulnerabilitățile de securitate ale infrastructurii care apar pe măsură ce software-ul este folosit.

Managementul sistemului de securitate nu este o sarcină unică, ci include o serie de activități, precum gestionarea utilizatorilor și a permisiunii, implementarea software-ului de sistem și întreținere, monitorizarea atacurilor, detectare și recuperare:

1. Managementul utilizatorilor și permisiunea includ adăugarea și eliminarea de utilizatori din sistem, asigurându-se că mecanisme adecvate de autentificare a utilizatorilor sunt în vigoare și de o permisiune a userilor corect corelată cu resursele de care au nevoie.
2. Implementarea software-ului de sistem și întreținerea includ instalarea și configurarea corespunzătoare a software-ului de sistem și a middleware-ului, astfel ca vulnerabilitățile de securitate să fie evitate. Aceasta implică, de asemenea, actualizarea acestui software în mod regulat cu versiuni sau patch-uri de securitate pentru problemele detectate.
3. Monitorizarea atacurilor, detecția și recuperarea includ activități ce monitorizează sistemul pentru accesul neautorizat la acesta, precum și detecția și punerea în aplicare a unor strategii de rezistență împotriva atacurilor și activității de backup, astfel încât funcționarea normală poate fi reluată după un atac extern.

Managementul Riscurilor

Managementul riscurilor este o problemă de afaceri, mai degrabă decât o problemă tehnică, astfel că inginerii software nu ar trebui să decidă ce controale ar trebui să fie incluse într-un sistem. Decizia aparține managementului superior și constă în acceptarea sau nu a costului de securitate sau expunerea care rezultă dintr-o lipsă de proceduri de securitate. Rolul inginerilor software în această arie este de a oferi orientare tehnică și raționament asupra problemelor de securitate. Aceștia sunt, prin urmare, participanți esențiali în procesul de management al riscului.

Asupra procesului de evaluare și gestionare al riscurilor are un rol esențial politica de securitate organizațională. Aceasta se aplică la toate sistemele și ar trebui să stabilească ce ar trebui și nu ar trebui fi permis. Politica de securitate prevede condițiile care ar trebui să fie întotdeauna menținute într-un sistem de securitate și, astfel, ajută la identificarea riscurilor și amenințărilor ce ar putea surveni.

Evaluarea riscurilor începe înainte ca decizia de a achiziționa sistemul să fi fost facută și ar trebui să continue pe parcursul procesului de dezvoltare a sistemului și după ce acesta a intrat în regim de funcționare.

Astfel, ideea de evaluare a riscurilor poate fi un proces pe etapele:

1. *Etapa preliminară*

În această etapă preliminară, deciziile privind cerințele detaliate ale sistemului, design-ul acestuia sau implementarea tehnologică nu au fost făcute.
Scopul acesui proces de evaluare este de a decide dacă un nivel adecvat de securitate poate fi realizat la un cost rezonabil. Dacă este cazul, se pot deriva cerințe specifice de securitate pentru sistem. Momentan nu sunt disponibile informațiile cu privire la vulnerabilitățile potențiale ale sistemului sau controalele care sunt incluse în componente reutilizate de sistem sau middleware.

2. *Evaluarea riscurilor din ciclu de viață*

Această etapă are loc în timpul dezvoltării ciclului de viață și este informată de proiectarea sistemului tehnic și de punerea în aplicare a deciziilor. Rezultatele evaluării pot conduce la modificări ale cerințelor de securitate și adăugarea de noi cerințe. Vulnerabilitățile cunoscute și cele potențiale sunt identificate, această detecție fiind utilizată pentru a altera luarea de decizii cu privire la funcționalitatea sistemului și modul în care aceasta urmează să fie implementat și testat.

3. *Evaluarea riscului operațional*

După ce un sistem a fost implementat și utilizat, evaluarea riscurilor ar trebui să ia în considerare modul în care sistemul este utilizat și să propună cerințe noi sau modificate. Ipoteze cu privire la cerința de funcționare efectuate pe vremea specificațiilor inițiale ale sistemului pot fi incorecte. Schimbări organizaționale pot însemna că sistemul este utilizat în diferite modalități față de cele prevăzute inițial. Prin urmare, evaluarea riscului operațional conduce la noi cerințe de securitate care trebuie să fie implementate pe măsură ce sistemul evoluează.

Pentru a efectua o evaluare a riscurilor este nevoie de identificarea posibilelor amenințări asupra unui sistem. O modalitate de a face acest lucru este de a dezvolta un set de *cazuri de abuz*(en: misuse case).[3]

Cazurile de abuz sunt o extensie a cazurilor de utilizare(en: *use case*) și sunt folosite pentru a modela securitatea încă din etapele inițiale ale dezvoltării sistemelor software.

Sindre și Opdahl definesc cazurile de abuz ca o lista sau secvență de pași care, dacă sunt operați corespunzător, pot dăuna sistemului și implicit, proprietarului.

Definiția lor pentru *misuser*(cel care folosește cazurile de abuz) face referire la o persoană care folosește sistemul cu intenții nefavorabile. Inițial, doar amenințările au fost modelate ca și cazuri de abuz. Conceptul de caz de utilizare au fost adaptate de la definiția de funcție de protecție a sistemului de riscurile identificate.

Riscurile pot fi caracterizate pe patru direcții:

- Riscuri de interceptare ce permit unui atacator accesul la un bun.
- Ruscuri de întrerupere care permit unui atacator să facă indisponibilă o parte a sistemului.
- Riscuri de modificare ce permit atacatorului să manipuleze un bun al sistemului.
- Riscuri de falsificare care permit unui atacator să insereze informații false într-un sistem.

Cazurile de abuz nu sunt utile doar în evaluarea preliminară a riscurilor, ci și în analiza de risc a ciclului de viață și în analiza riscului operațional. Acestea oferă o bază utilă pentru redarea unor atacuri ipotetice asupra sistemelor și evaluarea implicațiilor de securitate ale deciilor de proiectare care au fost făcute.

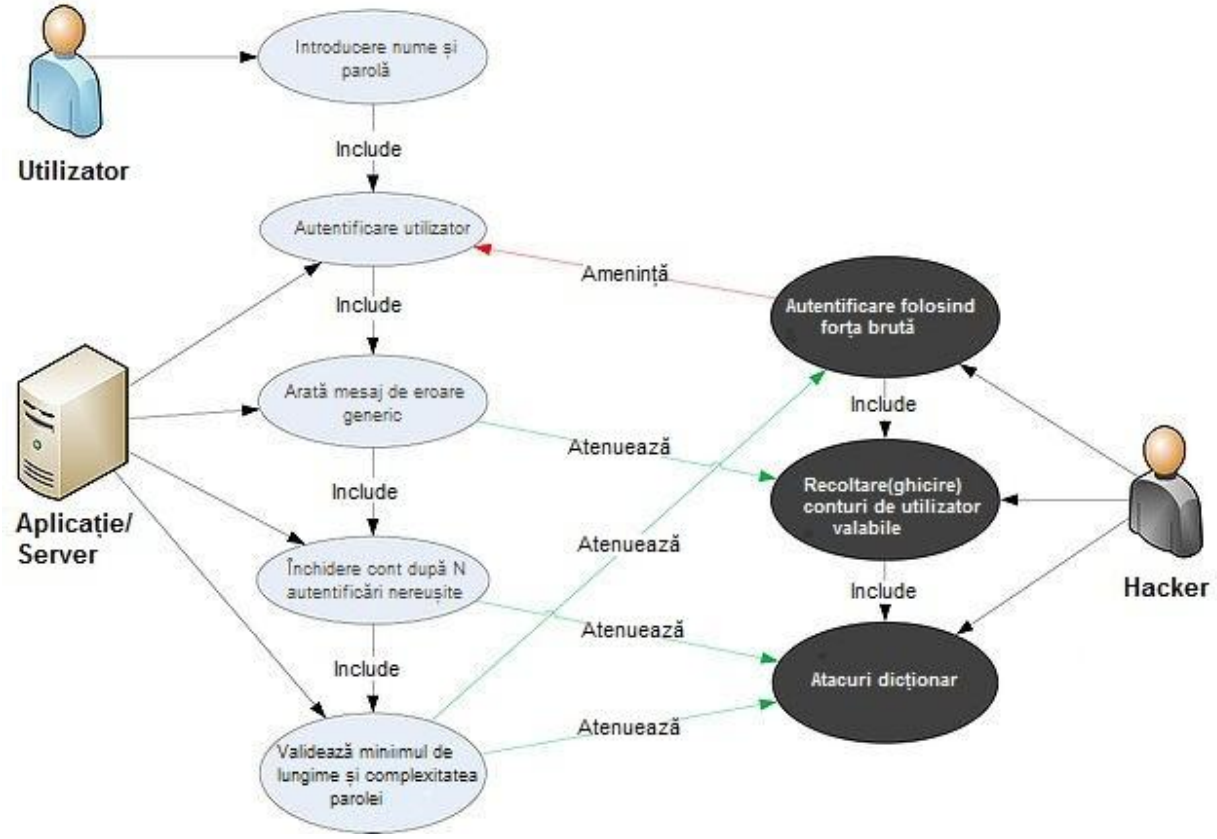


fig: Exemple de cazuri de utilizare(stânga) și cazuri de abuz(dreapta)[4]

Evaluarea riscurilor din ciclu de viață

Evaluarea riscurilor din ciclu de viață identifică detaliile de implementare și design care afectează securitatea. Aceasta este distincția importantă dintre evaluarea riscurilor și evaluarea preliminară a riscurilor. Evaluarea riscurilor din ciclul de viață afectează interpretarea cerințelor de securitate deja existente, generează noi cerințe și influențeaza proiectarea generală a sistemului.

În acest stagiu al evaluării riscurilor, informații detaliate asupra zonelor ce trebuie protejate și vulnerabilitățile sistemului ar trebui să fie cunoscute. Unele dintre aceste vulnerabilități vor fi inerente în alegerile optime de design.

Odată ce vulnerabilitățile au fost identificate, deciziile asupra pașilor necesari pentru a reduce riscurile asociate trebuie făcute. Acest lucru va implica de multe ori luarea deciziilor cu privire la cerințele de securitate suplimentare de sistem sau procesul operațional de folosire a sistemului. Exemple de astfel de cerințe:

1. Un program ce verifică parola va fi făcut și executat în fiecare zi.

Parolele utilizatorilor care apar în dicționarul sistemului vor fi identificate și utilizatorii cu parole slabe vor fi raportați la administratorii sistemului.

2. Accesul în sistem va fi permis numai calculatoarelor clienților care au fost aprobați și înregistrați de adminstratorii de sistem.

3. Toate calculatoarele clienților trebuie să aibă un singur browser web instalat care a fost aprobat de adminstratorii de sistem.

Evaluarea riscurilor de securitate

Această evaluare ar trebui să continue de-a lungul duratei de viață a sistemului pentru a identifica riscurile emergente și schimbările de sistem necesare pentru a putea face față la aceste riscuri. Noi riscuri pot apărea din cauza schimbărilor în cerințele de sistem, modificări în infrastructura sistemului sau schimbări în mediul în care sistemul este utilizat.

Procesul de evaluare a riscului operațional este similar celui de evaluare a riscurilor din ciclul de viață , dar cu adaos de informații cu privire la mediul în care sistemul este utilizat. Mediul este important deoarece caracteristicile sale pot duce la aparația unor noi riscuri pentru sistem. De exemplu, să presupunem că un sistem este utilizat într-un mediu în care utilizatorii sunt întrerupți frecvent. Un risc este faptul că întreruperea va însemna că utilizatorul trebuie să părăsească computerul nesupravegheat, acesta putând fi apoi utilizat de o persoană neautorizată pentru a obține informații din sistem. Acest lucru ar putea genera o cerință pentru execuția un screen saver protejat prin parolă, dacă sistemul a avut o perioadă de inactivitate.

Proiectarea securității

Proiectarea unui sistem sigur implică, în mod inevitabil, compromisuri. Cu siguranță este posibil să se elaboreze măsuri de securitate multiple într-un sistem care vor reduce șansele de succes ale unui atac. Cu toate acestea, măsurile de securitate necesită adesea o mulțime de culcul suplimentar și, astfel afectează performanța generală a unui sistem. De exemplu, se pot reduce șansele de a fi divulgate informații confidențiale prin criptarea informațiilor în cauză. Chiar și asa, impactul negativ asupra performanței este inevitabil, deoarece utilizatorii informațiilor criptate inițial vor trebui să aștepte ca acestea să fie decriptate, iar acest lucru va duce la încetinirea muncii lor.

Există o relație strânsă între fiabilitate și securitate. Utilizarea redundanței și a diversității, care este fundamentală pentru realizarea fiabilității, poate însemna că un sistem poate rezista și se poate recupera de la atacurile care vizează un design specific sau o caracteristică de implementare specifică.

Mecanismele care sprijină un nivel ridicat de disponibilitate pot ajuta sistemul să se recupereze de la așa-numitele respingeri ale atacurilor asupra serviciilor, unde scopul atacatorului este de a încetini sistemul sau de a-l opri permanent.

Există, de asemenea, tensiunile dintre securitate și ușurința în utilizare. Măsurile de securitate solicită, uneori, utilizatorului să-și amintească și să furnizeze informații suplimentare(de exemplu, mai multe parole). Cu toate acestea, uneori, utilizatorii uită aceste informații, astfel încât suplimentarea securității înseamnă ca ei nu pot folosi sistemul.

Designerii, prin urmare, trebuie să gasească un echilibru între securitate, performanță și ușurința în utilizare.

Pentru a proiecta un sistem sigur, pot fi luate în calcul următoarele aspecte:

1. Proiectarea arhitecturală – cum afectează securitatea unui sistem proiectarea arhitecturală?
2. Practică bună – ce este acceptat ca și practică bună la proiectarea sistemelor de siguranță?
3. Proiectarea pentru implementare – ce tip de suport ar trebui să fie proiectat în sisteme pentru a evita introducerea vulnerabilităților atunci când un sistem este gata de utilizare?

Proiectarea arhitecturală

În proiectarea unei arhitecturi de sistem care-și menține securitatea se pun două probleme fundamentale: protecția și distribuția.

Aceste probleme sunt potențial conflictuale. Pentru un sistem în care toate bunurile sunt stocate în același loc, protecția se va construi în jurul acelui loc. Construirea unui sistem de protecție unic are dezavantajul de compromitere a tuturor bunurilor, în același timp atunci când protecția acestuia eșuează. Adăugarea mai multor straturi de protecție afectează, de asemenea, gradul de utilizare a unui sistem, iar asta poate însemna că este mult mai dificil de îndeplinit gradul de utilizare de sistem și cerințele de performanță.

Pe de altă parte, dacă bunurile sunt distribuite, ele sunt mai scump de protejat, deoarece sistemele de protecție trebuie implementate pentru fiecare copie, fapt ce duce la renunțarea acestei protecții datorită costului.Astfel, șansele de penetrare ale protecției sunt mari. Cu toate acestea, în cazul în care se întâmplă acest lucru, nu se va suferi o pierdere totală.

Pentru sistemele de evidență a pacienților, de exemplu, este necesară utilizarea unei arhitecturi de baze de date centralizate. Pentru a asigura protecția, este recomandată utilizarea unei arhitecturi ierarhice, în care cele mai importante date sunt poziționate la baza acestei arhitecturi, iar în jurul lor sunt adăugate mai multe straturi de protecție.

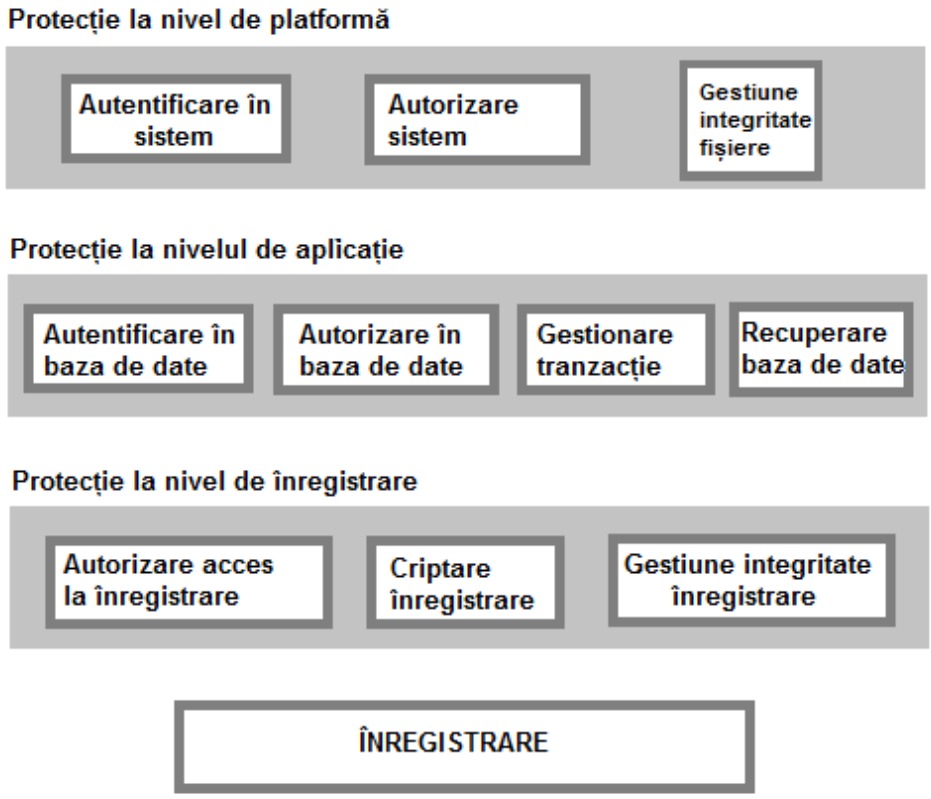


fig: Arhitectură pe mai multe straturi(ierarhică)[5]

Pentru a accesa și modifica evidența pacienților, un atacator trebuie să penetreze trei niveluri de sistem: protecția la nivel de platformă, protecția la nivel de aplicație și protecția la nivel de înregistrare.

Orientări de proiectare

Nu există reguli cu privire la modul de realizare a securității sistemului. Diferite tipuri de sisteme necesită măsuri tehnice diferite, pentru a atinge un nivel de securitate care este acceptabil pentru proprietarul sistemului. Atitudinile și cerințele diferitelor grupuri de utilizatori afecteaza profund ceea ce este și nu este acceptabil.

Cu toate acestea, există linii de orientare care au aplicabilitate largă la proiectarea sistemului de soluții de, care înglobează practici de proiectare bună pentru inginerie de sistem sigură.

Cele mai comune zece orientări de proiectare sunt:

1. Deciziile de securitate de bază privind o politică de securitate explicită
2. Evitarea unui singur punct de eșec
3. Metode controlate de eșec
4. Echilibrarea securității și a utilizării
5. Istoricul acțiunilor utilizatorilor
6. Folosirea redundanței și a diversității pentru reducerea riscului
7. Validarea tuturor intrărilor
8. Compartimentarea datelor importante
9. Proiectare pentru implementare
10. Proiectare pentru recuperare.

Proiectarea pentru implementare

Implementarea unui sistem implică configurarea software-ului pentru a funcționa într-un mediu operațional, instalarea sistemului pe calculatoarele din acel mediu și apoi configurarea sistemului instalat pentru acele calculatoare.

Configurarea poate fi un proces simplu, care implică stabilirea unor parametri integrați în software penru a reflecta preferințele utilizatorului.

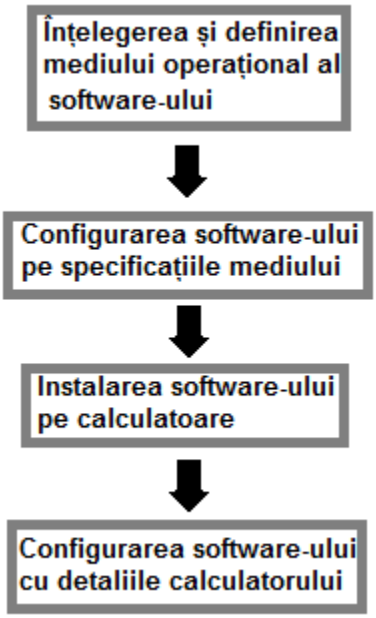


fig: procesul de configurare

Cu toate acestea, uneori configurația este complexă și necesită definirea specifică a modelelor de afaceri și regulile care afectează execuția software-ului. În acest stagiu al dezvoltării software se întâmplă ca vulnerabilitățile să fie introduse accidental.

Configurarea și implementarea sunt adesea văzute ca probleme de administrare a sistemului și astfel sunt considerate a nu face parte din procesele ingineriei software.

Desigur, bunele practici de management pot evita multe probleme de securitate ce apar din greșeli de configurare și implementare. Cu toate acestea, dezvoltatorii software au responsabilitatea de a "proiecta pentru implementare".

Metodele de a încorpora suportul de implementare într-un sistem pot fi:
- Includerea suportului pentru vizualizarea și analiza configurațiilor;
- Minimizarea privilegiilor implicite;
- Localizarea setărilor de configurare;
- Furnizarea de moduri ușoare pentru a remedia vulnerabilitățile de securitate

Supraviețuirea sistemului

Capacitatea de a supraviețui (sau reziliența) este o proprietate emergentă a unui sistem ca un întreg, mai degrabă decât o proprietate de componente individuale, care nu pot supraviețui singure. Reziliența sistemului reflectă abilitatea sa de a continua să livreze servicii de afaceri sau criticale pentru utilizatori legitimi în timp ce este atacat sau după ce o parte a sistemului a fost deteriorată. Prejudiciul poate fi cauzat de un atac sau o eroare de sistem.[6]

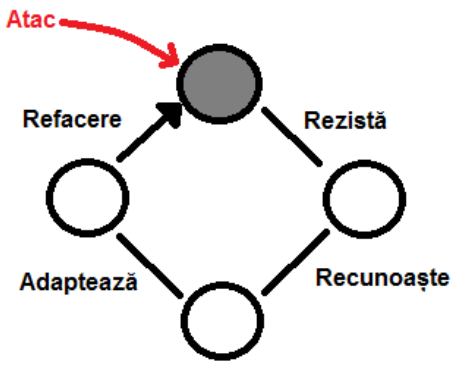


fig: ciclul de supraviețuire[6]

Rezistența este capacitatea de a respinge atacurile. Recunoașterea este capacitatea de a detecta atacurile când acestea apar și să evalueze gradul de daune și de compromitere. Recuperarea, o marcă a supraviețuirii, este capacitatea de a menține serviciile esențiale și elementele critice în timpul atacului, de a limita gradul de deteriorare și de a restabili serviciile complet în urma atacului.

Munca depusă la sporirea rezilienței sistemelor a fost determinată de faptul că viețile noastre economice și sociale sunt dependente de o infrastructură critică controlată de computer. Aceasta include infrastructura pentru livrarea utilităților (energie, apă, gaz, etc) și, la fel de importantă, a infrastructurii pentru furnizarea și gestionarea informațiilor (telefoane, internet, serviciu poștal, etc). Cu toate acestea, supraviețuirea nu este doar o simplă problemă de infrastructură. Orice organizație care se bazează pe sisteme de calculatoare legate in rețele ar trebui să se îngrijoreze de modul în care afacerea va fi afectată dacă sistemele lor nu supraviețuiesc unui atac sau defectării sistemului provocate de o catastrofă. De aceea, pentru sistemele critice de afaceri, analiza și proiectarea rezilienței ar trebui să facă parte din procesul de inginerie a securității.

Menținerea disponibilității serviciilor critice este esența supraviețuirii.
Lucruri de știut în această direcție:

- serviciile de sistem care sunt critice pentru afacere;
- calitatea minimă a serviciilor ce trebuie menținută;
- modul în care aceste servicii pot fi compromise;
- modul în care aceste servicii pot fi protejate;
- modul de recuperare rapidă când serviciile devin indisponibile.

Analiza rezilienței sistemelor este un proces ce se desfășoară în patru etape. Aceasta analizează cerințele și arhitectura curentă sau propusă a sistemului; identifică serviciile critice, scenariile de atac și punctele slabe ale sistemului și propune schimbări pentru a îmbunătăți supraviețuirea sistemului.

Activitățile cheie în aceste etape sunt:

- Înțelegerea sistemului
- Identificarea serviciilor critice
- Simularea atacurilor
- Analiza rezilienței.

Metoda analitică a rețelei de supraviețuire

Efortul principal al SEI(Software Engineering Institute) a fost dezvoltarea metoda Analizei Rețelei de Supraviețuire(en SNA) pentru evaluarea și îmbunătățirea capacității de supraviețuire ale arhitecturilor de rețea.[7]

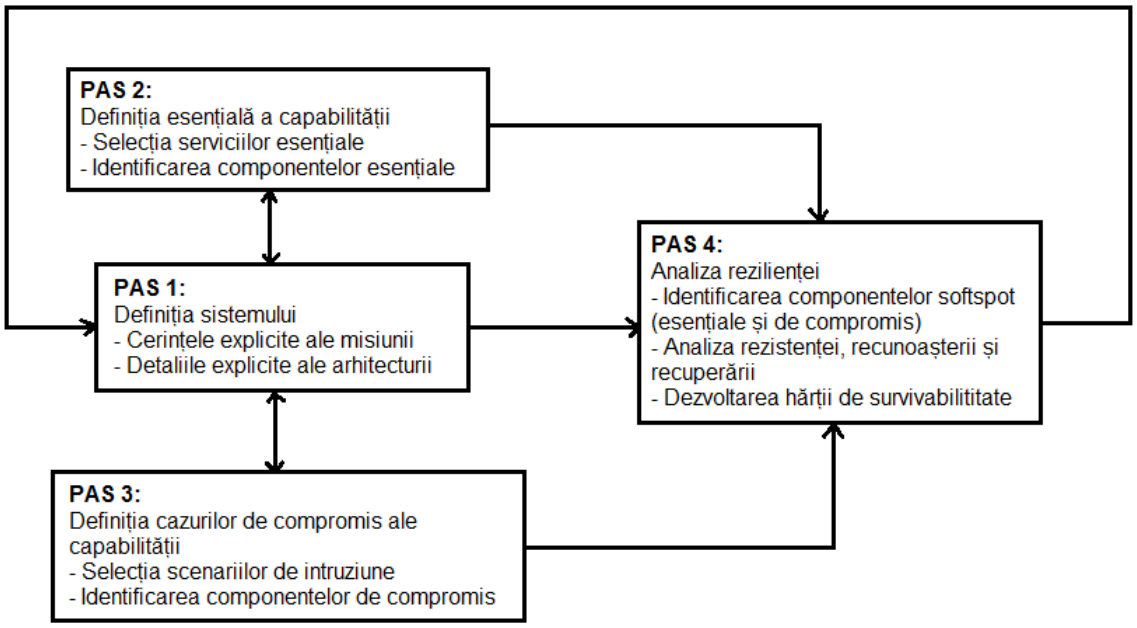


fig: Pașii unei metode de analiză a supraviețuirii rețelei[7]

Pas 1: Cerințele misiunii și a arhitecturii sistemului curent sunt formulate în mod explicit de către client și arhitecții sistemului. Cerințele sunt elaborate, de obicei, în cerințe specifice funcționale și non-funcționale pentru serviciile de sistem.

Pas 2: Serviciile esențiale(serviciile care trebuie menținute în timpul atacului) și elementele critice (a căror integritate, confidențialitate, disponibilitate și alte proprietăți trebuie menținute pe durata atacului) sunt identificate, pe baza obiectivelor misiunii și consecințele eșecului. Aceste servicii și elemente critice sunt caracterizate de scenariile de utilizare care sunt integrate în arhitectură pentru a identifica componentele esențiale corespunzătoare(componente care trebuie să fie disponibile pentru a furniza servicii de bază și să mențină elementele critice).

Pas 3: Scenariile de intruziune sunt selectate pe baza mediului în care este instalat sistemul și de evaluarea riscurilor și capabilitățile intrușilor. Aceste scenarii sunt, de asemenea, mapate în arhitectură pentru a identifica componentele de compromis corespunzătoare(componente care ar putea fi penetrate și afectate de intruziune).

Pas 4: Componentele *softspot* ale arhitecturii sunt identificate ca și componente care sunt în același timp esențiale și de compromis, pe baza rezultatelor pașilor 2 și 3. Acestor componente, împreună cu cerințele generale, le sunt apoi analizate trei proprietăți cheie ale rezilienței, și anume: rezistența, recunoașterea și recuperarea. Rezistența este capacitatea de a respinge atacurile. Recunoașterea este capacitatea de a detecta atacurile când acestea apar și evaluarea gradului de daune și de compromitere. Recuperarea, o marcă a supraviețuirii, este

capacitatea de a menține serviciile esențiale și elementele critice în timpul atacului, de a limita gradul de deteriorare și de a restabili complet serviciile în urma atacului.

Bibliografie

1. Software Engineering 9th ed - Pearson, 2011 – I. Sommerville
2. Security Risks: Management and Mitigation in the Software Life Cycle - 2004 – David P. Gilliam
    - http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/41564/1/04-0778.pdf
3. Towards Security Risk-oriented Misuse Cases - 2012 – Imam Soomro and Naved Ahmed
    - http://www.inf.unibz.it/sbp12/papers/P5-Soomro.pdf
4. Capturing Security Requirements through Misuse Cases - 2001 – Guttorm Sindre and Andreas L. Opdahl
    - http://www.nik.no/2001/21-sindre.pdf
5. Building Secure Software - 2001 - John Viega and Gary McGraw
    - http://collaboration.csc.ncsu.edu/CSC326/Website/lectures/bss-ch1.pdf
6. Survivability – A new security paradigm for protecting highly distributed mission-critical systems – 2000 – Howard F. Lipson, Ph.D. http://www.dependability.org/wg10.4/meeting38/05-Lipso.pdf
7. A Case Study in Requirements for Survivable Systems - 2009 - Robert J. Ellison, Richard C. Linger, Thomas Longstaff, Nancy R. Mead
8. http://www.sei.cmu.edu/library/assets/http___cmsauth.sei.cmu.edu_88_publications_articles_ellison-survivable-systems_ellison-survivable-systems.pdf

Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Pompiliu Stanciu

Analiza Statica

➢ Introducere:

Metodele de analiză statică reprezintă metode de verificare a sistemului care nu implică executarea unui program. Ele lucrează bazându-se pe o reprezentare sursă a software-ului – fie un model de specificație sau design pe sursa cod a programului.

Metodele de analiză statică pot fi folosite pentru a asigura că erorile vor fi detectate de specificația sau design-ul unui model de sistem înainte ca o versiune executabilă a sistemului să fie facută disponibilă.

Acestea au, de asemenea, avantajul că prezența erorilor nu perturbă verificarea sistemului. Când se testează un program, un defect poate masca sau ascunde un altul, astfel încât necesită ștergerea unui defect detectat și apoi repetarea procesului de testare.

Probabil cea mai comună metodă de analiză statică este analiza și inspecția în pereche/grup, unde specificația, design-ul sau programul este verificat de un grup de persoane. Aceștia examinează codul sau design-ul în detaliu, cautând posibile erori sau omisiuni. O altă metodă folosește uneltele de modelare a design-ului pentru a verifica anomalii ale UML, cum ar fi același numele folosit pentru diferite obiecte.

Cu toate acestea, pentru sistemele analitice, metode de analiză statică adiționale pot fi folosite pentru:

- Verificarea formală, unde se produc argumente matematice riguroase conform cărora un program se va conforma specificației sale.
- Verificarea modelului, unde se încearcă o teoremă pentru a verifica de inconsistențe descrierea formală a sistemului.
- Analiza automată a programului, unde sursa cod a unui program este verificată de anumite caracteristici despre care se știe că pot fi potențial eronate.

Aceste metode se află în strânsă legătură. Verificarea modelului se bazează pe un model formal al sistemului ce poate fi creat pornind de la specificație formală. Analizatorii statici pot folosi aserțiuni formale încorporate într-un program ca și comentarii pentru a verifica dacă codul asociat este compatibil cu aceste aserțiuni.

Pentru ca analiza statică să fie practică, trebuie să fie integrată continuu în workflow-ul echipei. Un astfel de exemplu este infrastructura Parasoft care este preconfigurată pentru integrare si testare continuă . Astfel:
- preia ultimele versiuni de cod de la sursă.

- verifică compatibilitatea codului cu regulile desemnate
- atenționează dezvoltatorul care a cauzat încălcarea regulilor
- distribuie detalii în legătură cu încălcarea regulilor dezvoltatorilor în cauză.

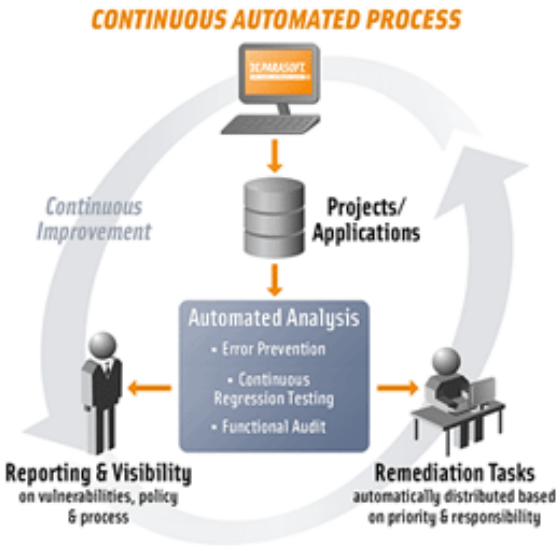


fig: Analiza statică Parasoft[2]

Metode formale

Metodele formale ale dezvoltării unui software se bazează pe un model formal al sistemului care servește drept specificație a sistemului. Aceste metode formale se preocupă, în principiu, cu o analiză matematică a specificației; cu transformarea specificației într-o reprezentare detaliată, echivalentă din punct de vedere semantic sau cu o verificare formală a unei reprezentări a sistemului este echivalentă semantic cu o altă reprezentare.
Metodele formale pot fi folosite in diferite stagii ale procesului V&V:

- O specificație formală a sistemului poate fi dezvoltată si analizată matematic împotriva inconsistențelor. Această metodă este folositoare în descoperirea erorilor și omisiunilor privind specificațiilor. Verificarea modelului este o metodă de analiză a specificației.
- Se poate verifica normal, folosind argumente matematice, dacă codul unui sistem software este compatibil cu specificația sa. Aceasta necesită o specificație formală. Este necesară în descoperirea erorilor privind programarea si design-ul.

Datorită unei discrepanțe semantice mari între specificația formală a unui sistem si un cod de program, este dificil de demonstrat că un program dezvoltat separat este compatibil cu specificația sa. Verificarea programului este astfel, bazată pe dezvoltarea transformațională.

Într-un proces de dezvoltare transformațională, o specificație formală este transformată printr-o serie de reprezentări ale codului programului. Uneltele software-ului susțin dezvoltarea transformărilor și ajută la verificarea compatibilității între reprezentări corespondete ale sistemului. Metoda B este probabil cea mai folosită metodă transformațională (Abrial, 2005; Wordsworth, 1996). A fost folosită pentru dezvoltarea sistemelor de control a trenurilor și a software-urilor în aeronautică și astronomie.

Susținătorii metodei formale declară că folosirea acestor metode duce la sisteme mult mai solide și sigure. Verificarea formală demonstrează că programul dezvoltat este compatibil cu specificația sa și ca erorile de implementare nu vor compromite siguranța sistemului. Dacă se dezvoltă un model formal al sistemelor concurente folosind o specificație scrisă într-un limbaj cum ar fi CSP (Schneider, 1999), se pot descoperi premise ce ar putea duce într-un punct mort în programul final și să fie capabil să le localizeze. Acest lucru este deosebit de dificil de realizat de unul singur.[1]

Specificația formală și corectura nu garantează ca software-ul va fi sigur pentru uzul practic. Motivul pentru aceasta fiind:

1. Specificația poate sa nu reflecte premisele reale ale utilizatorilor sistemului. Aceștia rareori înțeleg notațiile formale, astfel ei nu pot citi direct specificația formală pentru a găsi erori și omisiuni. Aceasta înseamnă că există o posibilitate semnificativă ca specificația formală să conțină erori și să nu fie o reprezentare precisă a cererilor sistemului.
2. Corectura poate conține erori. Corecturile programului sunt mari și complexe, astfel, fiind mari și complexe, conțin, de obicei, erori.
3. Corectura poate aduce ipoteze greșite despre felul în care sistemul este folosit. Dacă sistemul nu este folosit așa cum ar trebui, corectura poate fi invalidă.

În ciuda dezavantajelor, metodele formale și verificarea formală au un rol important în dezvoltarea sistemelor software analitice. Specificația formală este foarte eficientă în descoperirea acelor specificașii problemă e sunt, în cele mai multe cazuri, defectarea sistemului. Deși verificarea formală este încă impracticabilă pentru sistemele mari, poate fi folosită pentru verificarea importanței și siguranței componentelor analitice.

Verificarea modelului

Verificarea modelului implică crearea unui model al unui sistem și verificarea corectitudinii acelui model folosind unelte software specializare.

Procesul de verificare a modelului implică construirea unui model de sistem, de obicei ca o mașinărie finită extinsă. Modelele sunt exprimate în orice limbaj este folosit de sistemul de verificare al modelului – spre exemplu, modelul de verificare al SPIN folosește un limbaj numit Promela. Un set de proprietăți dorite pentru sistem sunt identificate și notate într-o notație formală, de obicei bazate pe o logică temporală. Un exemplu al unei asemenea proprietăți a

unui sistem poate fi aceea ca sistemul va atinge mereu starea de “transmitere” din starea de ”înregistrare”.

Verificatorul modelului explorează apoi toate direcțiile din interiorul modelului (toate stările de tranziție posibile), verificând dacă proprietatea se aplică fiecărei direcții. Dacă acest lucru se întâmplă, verificatorul modelului confirmă că modelul este corect privind proprietatea respectivă. Dacă nu se respectă într-o anumită direcție, verificatorul produce un contra exemplu ce ilustrează unde proprietatea nu este adevărată. Verificarea modelului este folositoare, în principiu, în validarea sistemelor concurente, ceea ce este dificil de testat datorită sensibilității lor la timp. Verificatorul poate explora intercalat, transmisiile concurente și să descopere potențiale probleme.

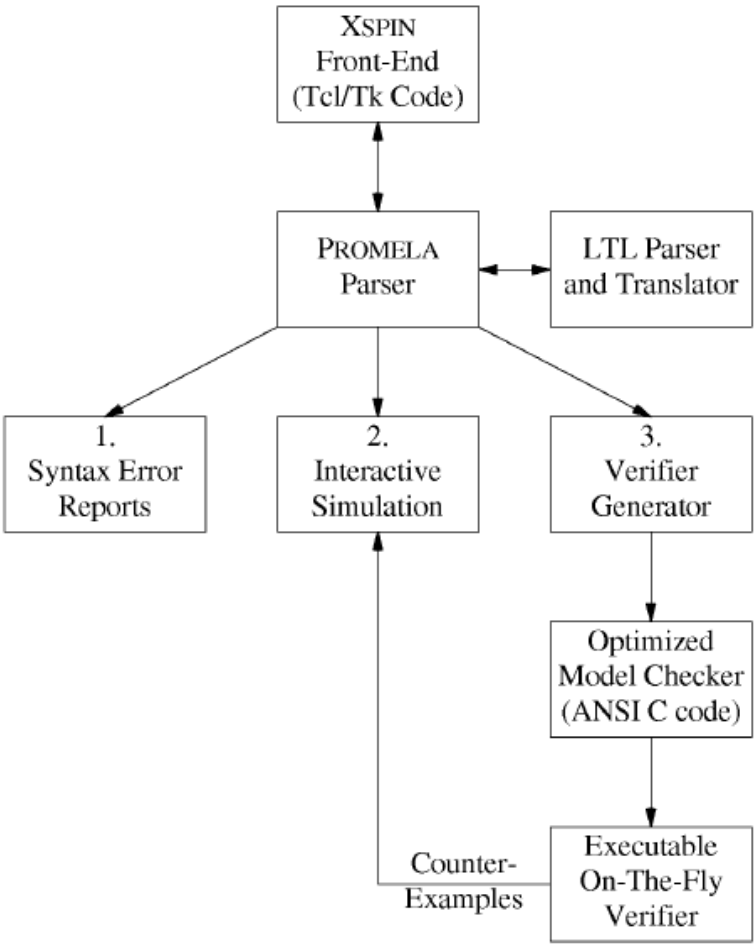


fig: Structura simulării și verificării SPIN[4]

O problemă cheie în verificarea modelului este crearea unui model de sistem. Dacă modelul trebuie creat manual (dintr-un document de cerințe sau design) va fi un proces costisitor, deoarece crearea unui model necesită foarte mult timp. Mai mult, există posibilitatea ca modelul creat să nu fie fidel cerințelor sau design-ului. Este, astfel, cel mai bine ca modelul să fie creat automat din codul sursă al programului. Sistemul Java Pathfinder (Visser, 2003) este un exemplu al unui sistem de verificare a modelului care lucrează direct de la o reprezentare a unui cod Java.

Verificarea modelului este, din punct de vedere al volumului de calcul, foarte costisitoare deoarece foloseste o abordare exhaustivă pentru a verificare toate direcțiile dintr-un model de sistem. Odată cu creșterea mărimii, crește și numărul de stări, rezultând o creștere a numărului de direcții ce trebuie verificate. Aceasta înseamnă, pentru sistemele mari, că verificarea modelului poate fi impracticabilă, datorită timpului cerut de computer pentru a demara aceste verificări.

În orice caz, în timp ce algoritmii pentru identificarea acelor părți ale stării ce nu au verificat o anumită proprietate se îmbunătățesc, va deveni din ce în ce mai practică folosirea rutinieră a verificării modelului în dezvoltarea sistemelor analitice. Nu este aplicabil sistemelor de organizare orientate spre date, dar poate fi folosit pentru verificarea sistemelor software încadrate ce sunt modelate ca mașinării de stare.

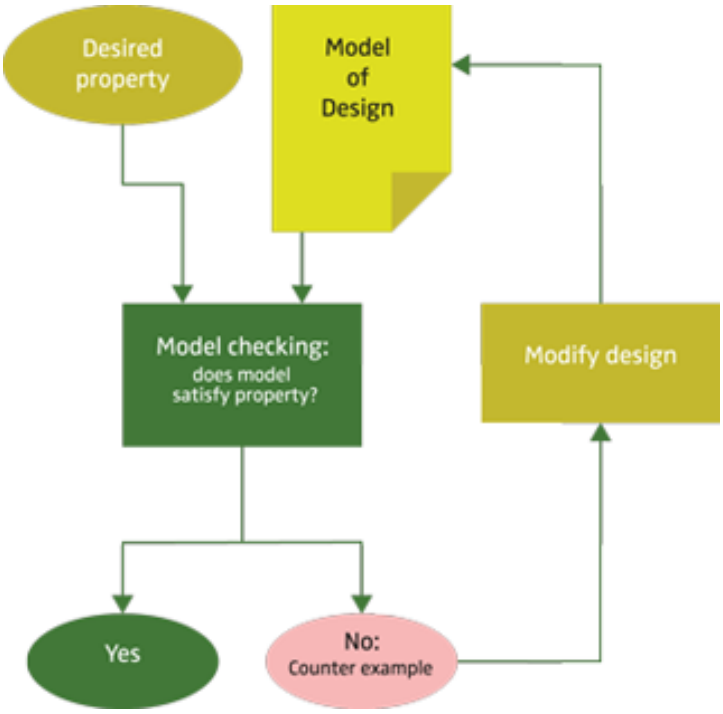


fig: analiza modelului pentru detecția timpurie a erorilor[3]

Descrierea metodei descrise de figura anterioară:

- Dezvoltarea unui model abstract comportamental al design-ului software
- Verificarea proprietăților folosind un instrument de verificare al modelului
- Repetarea abordării prin adaptarea modelului de proiectare și/sau prin adăugarea sau modificarea proprietăților.

Testarea fiabilității

Testarea fiabilității reprezintă un proces de testare ce își propune să măsoare fiabilitatea unui sistem. Există mai multi parametri ai fiabilității, cum ar fi POFOD, probabilitatea defectării la cerere și ROCOF, rata fenomenului de defectare. Acestea pot fi folosite pentru a specifica, din punct de vedere cantitativ, fiabilitatea sistemului. În procesul de testare a fiabilității sistemului se poate verifica dacă sistemul a atins nivelul cerut de fiabilitate.

În cazul unor unor caracteristici upgrade-ate, complexitatea software-ului va scădea, datorită sporirii funcționalității acestuia. Chiar și bug fix-urile pot fi un motiv pentru eșecul mai multor aplicații, dacă bug fix-urile respective induc alte defecte în sistem. Ca upgrade-uri de fiabilitate pot fi amintite reproiectarea și/sau reimplementarea unor module ce folosesc metode inginerești mai bune.

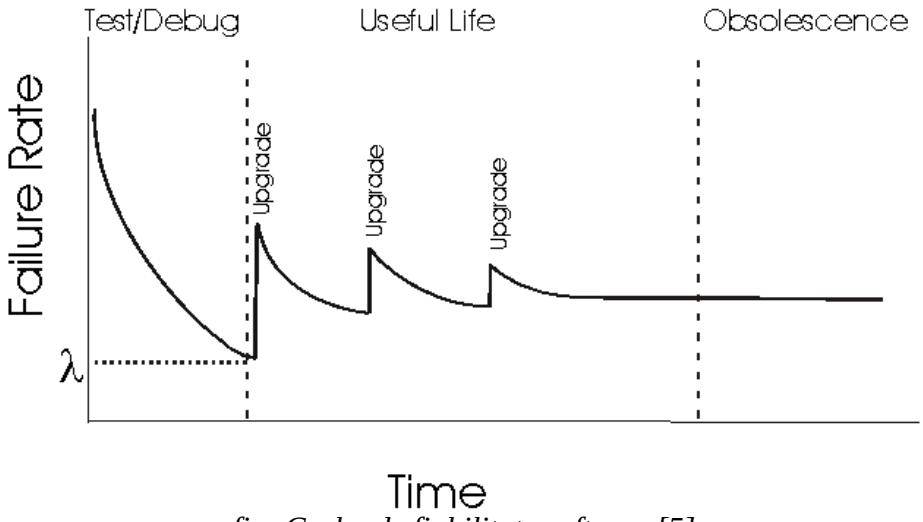


fig: Curba de fiabilitate software[5]

Procesul de măsurare a fiabilității include patru pași:

1. Se începe prin studierea sistemelor de același tip existente pentru a înțelege cum sunt acestea folosite în practică. Acest pas este important întrucât se încearcă măsurarea fiabilității așa cum este văzută de un utilizator al sistemului. Scopul este acela de a defini un profil operațional. Un profil operațional identifică clasele de intrări în sistem și probabilitatea ca aceste intrări să aibă o utilizare normală.

2. Următorul pas constă în construirea unui set de date care reflectă profilul operațional. Astfel, se creează date de testare cu aceeași distribuție a probabilității ca datele de testare pentru sistemul studiat. În mod normal, se va folosi un generator de date de testare care pot susține acest proces.
3. Se testează sistemul folosind aceste date și se numără tipurile de defectări ce se petrec. Momentele în care aceste defectări se petrec sunt, de asemenea, notate. Unitățile de timp alese ar trebui să fie potrivite pentru fiabilitatea parametrilor folosiți.
4. După ce se observă un număr semnificativ de defectări, se poate evalua fiabilitatea software-ului și să se decidă valoarea parametrilor potriviți de fiabilitate.

Această abordare în patru etape este numită uneori „testare statistică". Scopul testării statistice este să aprecieze fiabilitatea unui sistem, în contrast cu testarea defectelor, a cărui scop este să descopere deficiențele sistemului.

Această abordare a măsurării fiabilității, atractivă din punct de vedere conceptual, nu este ușor de aplicat practic.

Principalele dificultăți derivă din:

1. *Lipsa de precizie a profilului operațional*

Profilul operațional bazat pe experiența cu alte sisteme poate să nu fie o reflecție precisă a adevăratei utilizări a sistemului.

2. Costurile mari ale generării de date

Generarea de mari volume de date cerută de un profil operațional poate fi foarte costisitoare, cu excepția cazului în care întregul proces poate fi automatizat.

3. Lipsa de precizie a statisticilor atunci când se cere o fiabilitate de nivel înalt

Este necesară generarea unui număr semnificativ de defectări pentru a permite masurători exacte ale fiabilității. Când software-ul este deja sigur, se petrec un număr relativ mic de defectări și este dificil de generat defectări noi.

4. *Recunoașterea defectărilor*

Nu este întotdeauna evident dacă s-a produs sau nu o defectare a sistemului. Dacă există o specificație formală se pot observa deviații de la respectiva specificație, dar dacă specificația este făcută în limbaj natural se pot distinge ambiguități (ceea ce înseamnă că observatorii pot să nu ajungă la un consens cu privire la defectarea sistemului)

Profile operaționale

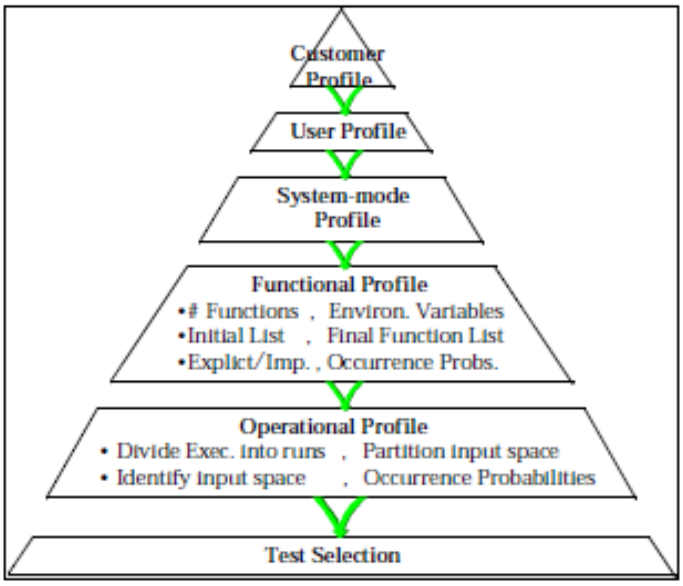
Profilul operațional al unui sistem software reflectă cum va fi acesta folosit în practică. Consistă dintr-o specificație de clase de intrări și din probabilitatea acestora de a se petrece. Când un nou sistem software înlocuiește un sistem automat deja existent, este destul de ușor de evaluat un model probabil al folosirii noului software. Acesta ar trebui să corespundă utilizării curente, cu o anumită permisiune făcută pentru noua funcționalitate care se presupune că este inclusă în noul software. Spre exemplu, un profil operațional poate fi

specificat pentru sistemul de telefonie deoarece companiile de telecomunicații cunosc modelul apelurilor pe care aceste sisteme trebuie să îl suporte.

Dezvoltarea unui sistem operațional precis este cu siguranța posibilă pentru anumite tipuri de sisteme, cum ar fi sistemele de telecomunicații, ce au o anumită structură de utilizare. Pentru alte tipuri de sisteme, totuși, există diverși utilizatori ce au moduri diferite de utilizare a sistemului. Diferiții utilizatori pot obține impresii diferite privind fiabilitatea deoarece folosesc sistemul în feluri variate.

Problema se diversifică în continuare întrucât, sistemele operaționale nu sunt statice, ci se schimbă în timpul utilizării. În timp ce utilizatorii cunosc mai multe detalii despre noul sistem și devin încrezători în folosirea acestuia, încep să îl utilizeze într-o manieră mai sofisticată. Datorită acestui lucru, este uneori imposibil de dezvoltat un profil operațional de încredere. În consecință, este dificil de găsit măsurători precise, deoarece acestea pot fi bazate pe presupuneri incorecte despre modul în care sistemul poate fi folosit.

Dezvoltarea unui profil operațional pentru un sistem include unu sau mai mulți din pașii următori:
1. Găsește profilul clientului
2. Stabilește profilul utilizatorului
3. Definește profilul de sistem
4. Determină profilul funcțional
5. Determină profilul operațional în sine



Testarea securității

Evaluarea unui sistem de securitate este din ce în ce mai importantă datorită faptului că tot mai multe sisteme analitice sunt dependente de internet și pot fi, astfel, accesate de oricine are acces la o conexiune la internet. În principiu există două motive pentru care testarea securității este atât de dificilă:

Cerințele securității, ca și anumite cerințe de siguranță, sunt cerințe de „nu ar trebui să", însemnând că ele specifică mai degrabă ce nu are trebui să se întâmple, în loc să ofere detalii de funcționalitate ale sistemului sau un comportament necesar. De obicei, nu se poate defini acest comportament nedorit ca simple constrângeri ce necesită a fi verificate de sistem.[1]

Dacă există resurse, se poate demonstra, cel puțin principial, că un sistem corespunde cerințelor sale de funcționare. Cu toate acestea, este imposibil de demonstrat că un sistem nu îndeplinește o anumită cerință. Indiferent de volumul de testare, vulnerabilitatea securității poate rămâne într-un sistem. Se pot, desigur, genera cerințe funcționale ce sunt desemnate să protejeze sistemul împotriva anumitori tipuri de atacuri. Chiar și în sisteme ce au fost în folosință timp de mai mulți ani, un atacator ingenios poate descoperi o nouă metodă de atac și poate penetra ceea ce s-a crezut a fi un sistem securizat.

Cei ce atacă un sistem sunt inteligenți și caută în mod activ vulnerabilități pe care le pot exploata. Aceștia sunt dispuși să experimenteze cu un sistem și să încerce anumite lucruri ce sunt înafara activității normale a utilizării sistemului. Spre exemplu, într-un câmp destinat introducerii numelui ei pot introduce 1000 de caractere conținând o combinație de litere, numere și semne de punctuație. În continuare, odată ce găsesc o vulnerabilitate, ei pot divulga aceasta informație crescând astfel numărul potențialilor atacatori.

Pentru a verifica securitatea unui sistem se poate folosi o combinație de testări, analiză bazată pe unelte și verificare formală:

- *Testarea bazată pe experiență*

  În acest caz, sistemul este analizat împotriva tipurilor de atac ce sunt cunoscute echipei de validare. Aceasta poate include cazuri de testare a dezvoltării sau examinarea codului sursă al sistemului. Spre exemplu, pentru a verifica faptul că un sistem nu este sensibil la cunoscutul atac SQL, se poate testa sistemul folosind intrări ce includ comenzi SQL. Acest tip de validare este de obicei folosit în combinație cu o validare bazată pe unelte, unde uneltele oferă informație ce poate fi de folos în testarea focalizată a sistemului. Liste de verificare conținând probleme de securitate cunoscute pot fi create pentru a ajuta procesului.

- *Tiger teams*

Aceasta este o formă de testare bazată pe experiență unde este posibil să se poată modifica experiența din exteriorul echipei de dezvoltare pentru a testa un sistem de aplicații. Se creează un tiger team căruia i se oferă obiectivul de a sparge securitatea sistemului. Aceștia simulează atacul asupra sistemului și își folosesc ingenuitatea pentru a descoperi noi moduri de a compromite securitatea sistemului. Membrii tiger team-ului ar trebui sa aibă experiență în prealabil privind testarea securității și găsirea vulnerabilităților securității într-un sistem.

- *Testarea bazată pe unelte*

Pentru această metodă diferite unelte de securitate, cum ar fi verificatoarele de parola, sunt folosite pentru a analiza sistemul. Verificatorii parolei detectează parolele nesigure, cum ar fi nume comune sau șiruri consecutive de litere. Această abordare este o extensie a validării bazate pe experiență, unde experiența vulnerabilităților securității este conținută de uneltele folosite. Analiza statistică este, desigur, un alt tip de testare bazată pe unelte.

- *Verificarea formală*

Un sistem poate fi verificat împotriva unei specificații de securitate formală. Cu toate acestea, la fel cum se întamplă și în alte situații, verificarea formală pentru securitate nu este folosită în mod obișnuit.

Testarea securității este, inevitabil, limitată de timpul și resursele disponibile echipei de testare. Aceasta înseamnă că, în mod normal, ar trebui adoptată o abordare bazată pe risc pentru testarea securității și concentrarea pe ceea ce se consideră că reprezintă cei mai importanți factori de risc pe care îi întâlnește sistemul. Dacă există o analiză a factorilor de risc la care este supus sistemul, aceasta poate fi folosită pentru a conduce procesul de testare. Pe lângă testarea sistemului împotriva cerințelor de securitate derivate de la aceste riscuri, echipa de testare ar trebui să spargă sistemul adoptând, mai degrabă, abordări alternative decât să amenințe calitatea sistemului.

În cazul în care se doresc anumite cerințe pentru un anumit nivel de securitate, se poate alege un product care a fost validat la acel nivel. În practică, totuși, aceste criterii au fost folosite în sistemele militare și nu au fost acceptate la scară largă pentru uzul comercial.

Bibliografie

1. . Software Engineering 9th edition – 2011 – I. Sommerville
http://tsime.uz.ac.zw/claroline/backends/download.php/U29mdHdhcmVfRW5naW5lZXJpbmdfOXRoX0VkaXRpb24ucGRm?cidReset=true&cidReq=CT210
2. Making Static Analysis a Continuous, Sustainable Process
http://www.parasoft.com/jsp/capabilities/static_analysis.jsp
3. Model checking for early error detection - ESI și partenerii: ASML, Chess, Océ, de la Eindhoven University of Technology, Radboud University Nijmegen, University of Twente
http://www.esi.nl/research/methods-and-tools/system-design/15.dot
4. The Model Checker SPIN – may 1997 – Gerard J. Holzmann
http://spinroot.com/spin/Doc/ieee97.pdf
5. Software Reliability – 1999 – Jiantao Pan
http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/
6. Operational Profiles
http://www.cs.colostate.edu/~cs530/rh/section9.pdf