Johanyák, Zs. Cs.: Biztonsági kockázatok elemzése a szoftverfejlesztésben, A GAMF Közleményei, Kecskeméti Főiskola, Gépipari és Automatizálási Műszaki Főiskolai Kar, Kecskemét, 2002, ISSN 0230-6182, pp. 45-51. http://johanyak.hu

Biztonsági kockázatok elemzése a szoftverfejlesztésben

Johanyák Zsolt Csaba

Bonyolult szoftverrendszerek esetén gyakran felbukkanó probléma, hogy bár rendeltetésszerűen működnek, de a piacra kerülést követően biztonsági szempontból kritikus hiányosságokra derül fény. A jelen cikk célja egy olyan, a minőségügy területén elterjedt módszer alkalmazási lehetőségének bemutatása, amely a széleskörűen használt tesztelési eljárások mellett, azokat kiegészítve törekszik a feltételezett biztonsági rések okainak feltárására.

1. Az elemzés szükségessége

Egy szoftverrendszernél könnyen előfordulhat, hogy kielégíti a megrendelő által megfogalmazott követelményeket, de nem biztonságos. Például egy Web böngésző tökéletesen letölti a felhasználó által megkívánt lapokat, lefuttatja a rajtuk elhelyezett szkripteket, de egy előre nem látott és ki nem védett rosszindulatú utasítássor eredményeként hozzáférhetővé teszi a gépen tárolt információkat. Gyakran előfordul, hogy a fejlesztők a rendszer bonyolultsága következtében a tényleges alkalmazás előtt nem képesek beazonosítani az összes biztonsági kockázatot. Amikor ez jelentős költséggel járhat, akkor a megszokott ellenőrzési technikák mellett olyan, a minőségügy hagyományos területein jól bevált módszerek alkalmazására van szükség, mint a hibafaelemzés (Fault Tree Analysis).

2. Az elemzés célja

Az FTA a katasztrofális eseményekre koncentrálva lehetővé teszi azon környezeti feltételek beazonosítását, amelyek mellett egy egyébként helyes rendszerállapot (működési mód) biztonsági szempontból kritikussá válhat. A módszert eredetileg az 1960-as években dolgozták ki az Egyesült Államokban a Minuteman rakétarendszer biztonsági elemzésére [1], később széleskörű alkalmazást nyert a gépipar és az elektronika területén a különböző rendszerek megbízhatóságának értékelése során.

Az elemzés célja egy szoftver terv vagy egy implementáció biztonsági szempontból történő értékelése és a kockázatok megszüntetése.

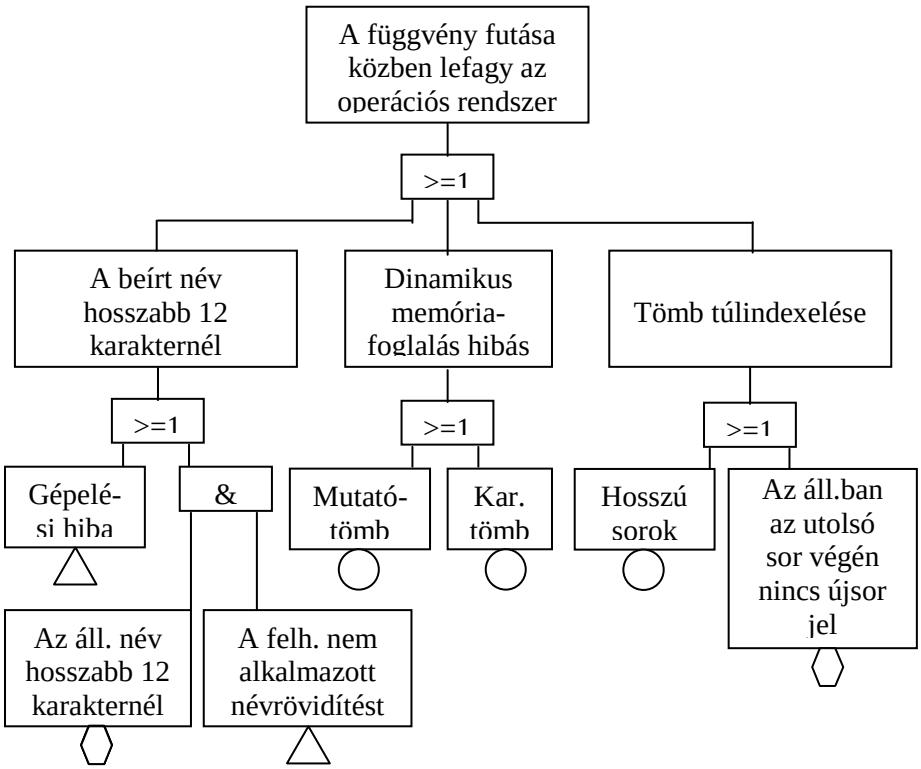
Eredményeképpen módosulhat a terv a megkívánt biztonsági szint megvalósítása érdekében. A hibafaelemzés lehetővé teszi:

- a fő-eseményhez vezető összes hiba és hibakombináció, valamint ezek okainak azonosítását,
- a különösen kritikus események és esemény-láncolatok kimutatását,
- olyan tesztesetek összeállítását, amelyekkel beazonosíthatók a leginkább veszélyt okozó modulok,
- a hibaterjedési mechanizmusok tiszta és áttekinthető dokumentálását.

3. Az elemzés módszere

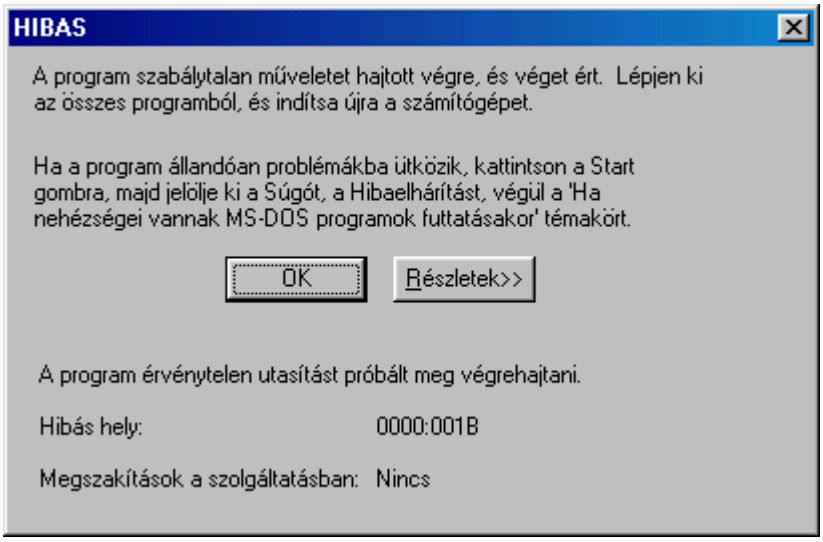
A szoftverrendszerek tervezése egy előre láncoló (adatvezérelt) következtetési folyamatként írható le, ahol a fejlesztők a kiinduló adatokból, elvárásokból és korábban – nem feltétlenül azonos körülmények számára – létrehozott komponensekből fokozatosan finomítva építkeznek.

Az alábbiakban ismertetésre kerülő módszer ezzel szemben hátrafele láncoló



1. ábra. Hibafa

(célvezérelt) technikát alkalmaz. Az elemző a tervezőtől teljesen eltérő szemszögből tekint a rendszerre. Egy feltételezett rendszerhibából (főesemény) indul ki, és fokozatosan felderíti azokat az alkotóelem és részrendszer meghibásodási lehetőségeit, melyek az adott esemény bekövetkezéséhez vezetnek. Az áttekinthető munkát fastruktúra-szerű grafikus megjelenítés segíti (1. ábra).



2. ábra. Hibaüzenet

A hibafaelemzés független az alkalmazott programozási nyelvtől és technikától, és a legtöbb esetben nem függetleníthető a hardvertől és operációs rendszertől. Például egy karakteres felületen dolgozó Pascal nyelvű program, ami egy bizonyos fejlesztőrendszer könyvtárában szereplő képernyőtörlési eljárást használta a Pentium II és Celeron processzorok megjelenéséig kiválóan működött, de az új hardveren nullával történő osztási hibával leállt. A vizsgálat során nem hagyható figyelmen kívül az emberi tényező, a felhasználó figyelmetlensége vagy hozzá nem értése.

A szoftver életciklusának minden szakaszában alkalmazhatjuk az FTA-t, kezdve az előzetes terv elkészültétől egészen a karbantartási műveletekig, de elsősorban a kódolási szakasz lezárásaként ajánlatos végrehajtani.

3.1. Előzetes kockázatelemzés

A szoftver hibafaelemzését megelőzi a teljes rendszer kockázatelemzése, aminek során beazonosítják azokat a nemkívánatos eseményeket, amelyek komoly következménnyel járhatnak. Itt fontos, hogy ne vesszünk el a részletekben, az

elemzést végző csapat egyértelmű határvonallal kell megjelölje, hogy melyek azok a témakörök, amelyek biztonsági szempontból kritikusak. Egy komplex rendszernél nem határozható meg előre az összes veszély, a módszer eredményessége az elemzést végző csapat ismereteinek és ötleteinek a függvénye.

A módszer ismertetése során felmerülő fogalmak megvilágítására vegyünk egy egyszerű programot, ami lemezkezelést és dinamikus memóriafoglalást tartalmaz. Az előzetes kockázatelemzés során a vizsgálatot végrehajtó csapat a „Betolt() függvény futása közben lefagy az operációs rendszer” leírású veszélyre hívta fel a figyelmet. A programot lefuttatva egy húsz betűből álló állománynevet megadva a 2. ábrán szereplő hibaüzenet jelenik meg Windows 98 alatt. A megnevezett függvény és az értelmezéshez szükséges típusdeklaráció az alábbiakban szerepel.

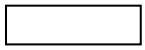
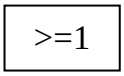
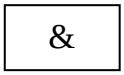
```
typedef char *String;
String *Betolt(int &sorszam)
{   char s[13],z[81];
    printf("Az állomány neve:");
    gets(s);
    FILE *fp=fopen(s,"rt");
    sorszam=0;
    int c;
    while((c=fgetc(fp))!=EOF)
    if(c=='\n') sorszam++;
    String *T=new String[sorszam];
    int i=0;
    while(fgets(z,81,fp)!=NULL)
    {   T[i]=new char[81];
        strcpy(T[i],z);
        printf("%d%s\n",i,z);
        i++;
    }
    getch();
    fclose(fp);
    return T;
}
```

3.2. Az ok-okozati kapcsolatok feltárása

Az elemzés kiinduló pontja a kockázatelemzés során megfogalmazott veszélyek listája. Ezeket egyenként feldolgozva, az általuk meghatározott főesemény bekövetkezését feltételezve több hibafa kifejtése történik párhuzamosan vagy

egymás után. A gyökértől (főesemény) kiindulva meghatározzák a bekövetkezést előidéző eseményeket vagy hiányosságokat, majd ezeket egyenként kifejtve, rekurzív technikát alkalmazva haladnak az okok tejes részletességű megismerése felé. Bonyolultabb helyzetekben egy halszálka (Ishikawa) diagram segítségével tehető rendszerezetté és áttekinthetővé a munka.

1. táblázat

Név és jel	Megjegyzés
	A kapuk bemeneteinek és kimeneteinek leírása.
	VAGY kapcsolat. A kapu több bemenettel is rendelkezhet.
	ÉS kapcsolat. A kapu több bemenettel is rendelkezhet.
	Elsődleges hiba.
	Másodlagos hiba.
	Kezelési hiba.

Egy esemény bekövetkezésének több feltétele is lehet. Ha bármelyik feltétel egyedül is előidézheti az eseményt, akkor ezeket egy logikai „vagy” kapun keresztül kapcsolják az eseményhez. Amennyiben csak az összes feltétel egyidejű teljesülése esetén következik be a hiba, akkor a kapcsolódás logikai „és” kapun keresztül történik. Ez a kiterjesztés egészen addig tart, amíg minden levél esetén számítható bekövetkezési valószínűséghez jutnak vagy az esemény további kifejtése már nem lehetséges. A kiterjesztés akkor is leáll, ha egy esemény bekövetkezésének előfeltétele egy hardver hiba, ami független a szoftvertől. A grafikus ábrázolás során az 1. táblázatban szereplő jeleket alkalmazzák.

Példánkban a főeseményt egyaránt előidézheti az, hogy a felhasználó 12-nél hosszabb állománynevet ad meg, azaz túlindexeli az s tömböt, valamelyik

memóriafoglalás sikertelen vagy a sorszámlálás-beolvasás hibás koncepciójú megoldása az elképzeltnél hosszabb állománysorokkal párosul. Mivel a fentiekben szereplő programrészlet egy „kitenyésztett” példája a hibás szoftverfejlesztésnek, ezért az olvasó számára bőven marad lehetőség a jelen esetben terjedelmi korlátok által behatárolt hibafa részletes kifejtésére.

A fa levelei által képviselt, tovább már nem bontható meghibásodások három osztályba sorolhatók:

- elsődleges hiba,
- másodlagos hiba,
- kezelési hiba.

Az *elsődleges hiba* egy olyan meghibásodás, mely az előírt működési körülmények között áll elő. Ennek oka a szoftvermodul (komponens) koncepciójában vagy kódolásában rejlik.

A *másodlagos hiba* egy olyan meghibásodás, ami nem megengedett külső behatások következtében áll elő. Ezek lehetnek hardver és szoftver környezeti feltételek, alkalmazási körülmények, vagy más futó folyamatok hatásai.

A *kezelési hibát* a nem megfelelő használat okozza.

3.2. Megoldások keresése

Az elemzés meghatározta az esemény bekövetkezésének feltételeit. A kockázat csökkentésének vagy megszüntetésének egyik módja olyan ellenőrzések beépítése a kódba, amelyek időben felismerik a feltételek meglétét, és lehetővé teszik a közbeavatkozást. Ez történhet a hagyományos programfejlesztésben elterjedten alkalmazott feltételvizsgálatokkal és visszacsatolásokkal vagy az objektum orientált technikában előszeretettel alkalmazott kivételkezelés segítségével.

Összefoglaló

Egy szoftver fejlesztése során a befektetett munka igen kis hányadát teszi ki a biztonságkritikus kódok keresése, a kapcsolódó költségek is elhanyagolhatók a tesztelési és verifikálási kiadások mellett. Az így születő kód sokkal robusztusabb, mint a módszer alkalmazása előtt volt.

IRODALOM

[1] NASA-GB-1741-13-96: NASA Guidebook for Safety Critical Software - Analysis and Development

[2] Helmer, G. – Wong, J. – Slagell, M. – Honavar, V. – Miller, L. – Lutz, R.: A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System, 1st Symposium on Requirements Engineering for Information Security, Indianapolis USA, 2001.
pp. 63 - 74.

[3] Leveson, N. G.: Safeware: System Safety and Computers, Addison-Wesley, New York, 1995.

Safety analysis in software development

Zsolt Csaba Johanyák

Summary

One problem that often appears by the development of complex software systems is that however they meet the functional requirements, when using them appear deficiencies related to safety. This paper aims to present one of the methods widely used in the field of quality assurance, which completes the common testing methods by exploring the causes of the supposed security flaws.

Sicherheitsrisikoanalyse in der Softwareentwicklung

Zsolt Csaba Johanyák

Zusammenfassung

Bei der Entwicklung von komplexen Softwaresystemen taucht häufig das Problem auf, daß obwohl sie den funktionalen Erwartungen entsprechen, trotzdem erscheinen Sicherheitslücken während der Benutzung. Dieser Beitrag präsentiert den Einsatz einer Methode, die auf dem Gebiet der Qualitätssicherung ausgebreitet Verwendung findet und die herkömmlichen Testverfahren durch die Forschung den Ursachen den angenommenen Sicherheitslücken ergänzt.

A Kötet szerzői:

Johanyák Zsolt Csaba
főiskolai adjunktus
Kecskeméti Főiskola Gépipari és Automatizálási Műszaki Főiskolai Kar
Kalmár Sándor Informatikai Intézet
Informatika Tanszék
E-mail: johanyak.csaba@kefo.hu

A lektor neve: Bodor Zoltán
Beosztása: Informatikus
Munkahelye: Bács-Kiskun Megyei Önkormányzat Pedagógiai Intézete