BUDAPESTI MŰSZAKI- ÉS GAZDASÁGTUDOMÁNYI EGYETEM
KÖZLEKEDÉSMÉRNÖKI KAR
KÖZLEKEDÉSAUTOMATIKAI TANSZÉK

COTS szoftverek alkalmazása biztonsági környezetben

Agócs Ferenc UPJ0C6

Budapest, 2011.

TARTALOMJEGYZÉK

1. Bevezetés

A biztonságkritikus rendszerekben alkalmazott COTS (Commerical off-the-shelf) szoftverből adódó kockázat csökkentésére számos technológiát alkalmazhatunk (előzetes alkalmassági tanúsítás, COTS funkciókorlátozás, HW és/vagy SW architektúra megfelelő kialakítása és a verifikációs technikák széles skálájának alkalmazása). Ezen technikák kerülnek röviden bemutatásra az alábbiakban, azonban ezek önálló alkalmazása nem elégíti ki a DO-178B, EN61508 stb. biztonsági szabványok által támasztott követelményeket. A biztonsági rendszereket fejlesztő mérnököknek tehát körültekintően kell megvizsgálni a COTS termékek funkcionalitását és származását, majd az eredményeknek megfelelően alkalmazni számos az alábbiakban vizsgált technikát a biztonsági szabványoknak való megfelelés érdekében.

2. Előzetes terméktanúsítvány

Egy COTS szoftver biztonsági rendszerbe integrálásának folyamatát, a rendszer későbbi tanúsítását nagymértékben megkönnyítheti, ha az adott COTS rendszerelem rendelkezik előzetes terméktanúsítvánnyal. A dokumentum keretén belül előzetes terméktanúsítványnak nevezzük azt a dokumentumot, mely igazolja, hogy az adott COTS rendszerelem korábban már alkalmazásra került valamilyen biztonságkritikus rendszer megvalósításában. A tanúsítványnak tartalmaznia kell a komponens régebbi rendszerben betöltött szerepét, funkcióit. Természetesen az új rendszerbe történő integrálás esetén is szükséges újra elvégezni az adott COTS modul és a teljes rendszer értékelését, de az új tanúsításban már támaszkodhatunk az előző rendszerek tapasztalataira, az onnan származó működési tulajdonságokra, paraméterekre.

3. COTS funkciókorlátozás

E technika alkalmazásakor a választott COTS komponenst nem a teljes funkcionalitásában használjuk, ezzel csökkentve a komponensben rejlő esetleges hibák előfordulását a biztonságkritikus rendszereinkben. Ez a technológia azonban többlet verifikációs lépéseket jelent a fejlesztés során, ugyanis egyrészt igazolni kell az ellátott funkció „jóságát”, másrészt igazolni kell, hogy a funkciókorlátozást megfelelően hajtottuk végre, tehát a nem kívánt funkciók valóban nem lesznek hatással a rendszerre.

4. Hardver architektúra

Manapság, a szoftver és hardver rendszerek bonyolultsága folyamatosan növekszik. Ebből következően egyik rendszer fejlesztése sem képzelhető el önálló folyamatként. A fejlesztés során fontos a szoftver és a hardver fejlesztések összehangolása. Erről akkor sem szabad megfeledkezni, amikor csak HW vagy SW módosítás, fejlesztés a feladat, mivel a rendszer egy adott funkciójával szemben támasztott biztonsági követelmény kielégítéséhez a két rendszerelem megfelelő kombinációja szükséges. Az alábbiakban néhány napjainkban alkalmazott hardver technológia kerül bemutatásra. Az, hogy melyik technológiát érdemes alkalmazni egy adott esetben, az a COTS felhasználásának helyétől, módjától függ.

4.1. Rendszer felosztás

A technika lényege, hogy a rendszer azon részét, mely a biztonságkritikus funkciókat valósítja meg külön választjuk azon részeitől, melyek nem biztonság releváns funkciókat valósítanak meg és biztosítjuk ezen részegységek visszahatás mentességét a biztonsági rendszermag felé.

4.2. Memória felosztás

A technika alkalmazásakor úgy osztjuk fel a memória területet, hogy külön választjuk a program kódot tartalmazó memória részt az adatokat tároló memória résztől úgy, hogy a memória címek egy csoportja nem lehet káros hatással egy másik cím csoportra.

4.3. Időfelosztás

Ezen technika lényege, hogy az egymással versengő programrészeket úgy kell időben eltolni, hogy azok működése a másik programrész működését ne befolyásolhassa.

4.4. Watch-Dog

Watch-dog-ok olyan elemek, melyeket szabályos időközönként meg kell szólítania a futó szoftvernek. Amennyiben a szoftver az adott időablakban nem küld jelet a watch-dog felé, abban az esetben az újraindítja vagy leállítja a figyelt folyamatot. A Watch-dog-ok lehetnek dedikált hardver elemek, vagy különálló szoftver egységek.

4.5. Redundancia alkalmazása

E technika esetében a rendszerben többletet alkalmazunk annak érdekében, hogy egy elem meghibásodása esetén a rendszer továbbra is el tudja látni feladatát. Maga a fizikai megvalósítás többféleképpen is lehetséges (elem, fokozat, rendszer redundancia, aktív/passzív stb.)

4.6. Kommunikáció

A rendszer biztonságkritikus részeinek kommunikációját úgy kell kialakítani, hogy azok egyértelműen leválaszthatóak, elhatárolhatóak legyenek a nem biztonsági funkciókat ellátó rendszerelemek adatforgalmától. Tehát biztosítható legyen a nem biztonsági funkciókat ellátó modulok visszahatás mentessége a biztonságkritikus funkciókat ellátó rendszerelemekre.

4.7. Második felügyelő rendszer alkalmazása

Ezen a technikán azt értjük, hogy a fő rendszer mellé telepítünk egy második megfigyelő rendszert, mely a fő rendszer kimeneteit figyeli. Abban az esetben, ha a fő rendszer a biztonságostól eltérő állapotot vezérelne ki, a megfigyelő közbeszól, és biztonságos akadályozó állapotba vezeti az irányító rendszert ezzel megelőzve a veszélyes állapot kialakulását. Egy második felügyelő rendszer alkalmazása tipikus rendszertechnikai kialakítás Fail-Safe biztonsági stratégia esetén.

5. Szoftver architektúra

5.1. Fault Detection and Accomodation

A hibakeresés a rendszer meghibásodott állapotainak ellenőrzési folyamata. A hibakeresés értékeket ellenőriz és a hibák időbeliségét vizsgálja, amelyek fizikai (pl. hőmérsékleti és feszültség-eszközök), logikai (pl. hibafelfedő kódok), funkcionális (pl. állítások) vagy külső (pl. megvalósíthatósági ellenőrzés) jellegűek lehetnek. A hiba-elkülönítési technikák azonosítani tudják azon biztonságos állapotokat, amelyekben a rendszer megfelelően üzemel. A diagnosztikai programrészeken keresztül a szoftver ellenőrzi önmagát, valamint a futtató hardvert a nem megfelelőségek szempontjából. Ezek a programok periodikusan vagy folyamatosan futhatnak háttérfolyamatként. Diagnosztikai programok működésük során két- vagy többszörös számításokat, paritás ellenőrzéseket valamint ciklikus redundancia-ellenőrzést alkalmazhatnak. A kritikus funkciók redundanciával tervezettek, a redundáns komponensek szavazással döntik el az egyes komponensek megfelelőségét.

5.2. Retry Fault Recovery

A helyreállítás újbóli megkísérléssel gyakran alkalmazott megoldás egymással kommunikációs kapcsolatban álló rendszerek esetén, de nem megszokott technika gyors, valós-idejű rendszerekben. E technika alkalmazása esetén a rendszer figyelemmel kíséri saját működését és hiba esetén újraindítja önmagát egy megelőző biztonságos állapotában és folytatja működését. Valós-idejű rendszerekben való alkalmazás esetén, további biztosítékok szükségesek, hogy a helyreállítás még azelőtt befejeződhessen, hogy a hiba hatása kívülről rendszerszinten megjelenhessen.

5.3. n-Version Programming

N-verziós programozás esetén független csoportok állítanak elő n darab szoftvert. Általában három verziót készítenek, azonban, biztonsági állapottal rendelkező rendszerek esetében két változat is alkalmazható a biztonsági állapot felé való eltolás esetén. A szoftvertermék mind az *n*-verziója része a szoftverrendszernek.

A közös módusú hibák elleni védekezés érdekében gyakran különböző programozási nyelveket és algoritmusokat alkalmaznak a független szoftverek esetén. Azonban ezzel nem zárhatóak ki a szoftver specifikációban elkövetett tervezési hibából adódó közös módosú hibák! Különböző szavazási stratégiák alkalmazhatóak a verziók között, hogy a legnagyobb gondossággal válasszuk ki a megfelelő kimenetet. Jelentős a vita abban a tekintetben, hogy az összes lehetőség *n*-verziós megvalósításához megtehetjük-e azt a feltételezést, hogy a függetlenség vezethet statisztikailag független hibákhoz. A bizonyítékok azt mutatják, hogy ez az előfeltételezés hibás lehet.

5.4. Recovery Block Programming

A recovery block programming egy olyan technika, amelynél a függetlenül írt szoftver modulok saját maguk helyességét ellenőrzik. Ebben az esetben a COTS komponenseket külön modulba kell szervezni a szoftverben és bármilyen a modulban felmerülő hiba esetén ki kell lépni még azelőtt, mielőtt a hibás működésnek nem várt következménye lenne. Ha a modul hibát észlel egy másik modult indíthatunk, mely úgy avatkozik be, hogy semmissé tegye a COTS modul hibájából adódó hatásokat és biztosítsa a rendszer további hibamentes futását.

5.5. Model Following

A modellkövetés az a technológia, amikor a COTS komponenst egy egyszerűsített modelljével helyettesítjük a rendszerben és ezen modell alkalmazásával bizonyítjuk a valós COTS rendszerelem megfelelő működését. Az, hogy pontosan milyen technikát alkalmazunk a COTS komponens modellezésére nagymértékben függ a komponens által ellátandó funkció bonyolultságától. Így ezen technika alkalmazásakor használhatjuk akár a legegyszerűbb table look-up modelleket, vagy közel teljes rendszermodelleket is.

5.6. Object-Oriented Architectures

A múltban a valósidejű biztonságkritikus rendszerek programozási nyelv választásánál kerülték az objektum orientált nyelvek alkalmazását. Ennek oka azok dinamikus tulajdonságai, a poliformizus alkalmazása esetén azok követhetőségének hiánya. Viszont manapság bizonyos megkötések betartása esetén egyre bővebb körben alkalmazzák ezen nyelveket is az alacsonyabb szintű kritikus funkciókat ellátó szoftverek esetében. OO nyelvek használata esetén ilyen megkötések például:

- ne alkalmazzunk többszörös öröklődést,
- kerüljük az egymásba ágyazást,
- ne engedjük meg az objektumok dinamikus létrehozását,
- ne használjunk dinamikus változókat.

6. Verifikációs technológiák

A COTS szoftverek esetében rendszerint nem rendelkezünk ismeretekkel a szoftverek belső szerkezetéről, működéséről, logikájáról. Így ezekben az esetekben az előírt teszteléseket csak úgy tudjuk végrehajtani, ha különböző értékeket adunk a szoftver bemenetére és figyeljük az ezekre adott kimenetei egyeznek-e a várt kimeneti értékeknek. Az ilyen elven végzett teszteket nevezzük fekete doboz tesztelésnek. A fejezet további részeiben a fekete doboz tesztelésnek különböző típusai kerülnek bemutatásra.

6.1. Stress Testing

A stressz teszt egy klasszikus fekete doboz tesztelési forma. Ez egy kifejezetten jó technika a magasabb szintű szoftver komponensek tesztelésére. A módszer alapvető célja, hogy meghatározza az alkotóelem meghibásodási módjait. A stressz teszt folyamán kivételesen nagy

terhelést helyeznek a tesztelendő komponensre és azt vizsgálják, hogy ezen terhelés alatt mennyire könnyen látja el a feladatát és ebből következtetnek arra, hogy üzemi körülmények között mennyire állja majd meg a helyét a modul. Továbbá a teszt alatt vizsgálják a komponens várható meghibásodási idejét és módjait is.
Ezen technika alkalmazása a COTS komponensek esetében kifejezetten ajánlott, mivel a teszt kapcsán jobban fény derül a szoftver valós működésére és meghibásodási módjaira, amennyiben ezek előkerülnek a teszt során. Ezekből az adatokból kiindulva viszont már pontosabban becsülhető a COTS komponens kockázati tényezője a rendszer működőképességére nézve, így amennyiben szükséges még időben módosítható a rendszer architektúra a megfelelő biztonsági szint, vagy rendelkezésre állás elérésének érdekében.

6.2. Process Scenario Testing

A PST lényege, hogy a szoftvert olyan környezetben teszteljék, mely körülmények között várhatóan teljesítenie kell feladatát. Ez magába foglalja a környezeti határértékeken való tesztelést is. Így ezzel a technikával széles skáláját fedhetjük le a működési körülményeknek, akár olyanokat is, melyek előállítása a valós környezetben túlságosan veszélyes, vagy közel lehetetlen lenne.

6.3. Equivalence-Class Testing

A nagy biztonságkritikus rendszerek nagyszámú bemenettel rendelkezhetnek. Az ilyen rendszereknél összes bemenet tesztelése nehezen megvalósítható, túlságosan időigényes megoldást jelentene. Ezért, a bemeneteket csoportokra oszthatjuk. Az egyes csoportokba azok a bemenetek kerülnek, melyekről tudjuk, vagy feltételezzük, hogy a tesztelt program azonos módon dolgozza fel , reprezentálja őket. A tesztek végrehajtásához, a bemeneti csoportok kialakításához különböző irányelveket követhetünk, de mindig szem előtt kell tartani, hogy ezen teszt esetében nem tudjuk az összes bemeneti variációt tesztelni. Ebből adódóan feltáratlan hiba maradhat a szoftverben, ha csak ezt a módszert alkalmazzuk!

6.4. Boundary-Value Testing

A határérték teszt esetében megvizsgáljuk, hogy bemeneti és kimeneti oldalon milyen értékek fordulhatnak elő. Ebből kiindulva olyan bemeneti adatokat választunk teszteléshez, mely megegyezik a bemeneti érték határértékeivel, illetve az felett és alatt helyezkednek el (3 érték/határérték) és ezen értékekre vizsgáljuk a rendszer kimenetét. A határérték teszt egy másik, kiterjesztett formája, amikor megvizsgálják a kimenetek értékeinek határérzékeit is és olyan

bemeneteket adnak a rendszernek, melyekkel a elérhetik a kimenetek határértékeit és várhatóan nem lépik túl azokat.

6.5. Random-Input Testing – Véletlen hibainjektálás teszt

A véletlen bemeneti értékeken alapuló tesztelést statisztikai tesztelésnek is szokás nevezni. A szoftver megbízhatóságának számszerűsítésére, a szoftverterméket és komponenseit mind szisztematikus mind véletlen tesztelés alá kell vetni. A szisztematikus perspektíva egy teljes áttekintést nyújt az ellenőrző tesztelésre, míg a véletlen perspektíva specifikus tesztesetekre alkalmazható. A véletlen megközelítéshez, a rendszerviselkedés egy előre látható eloszlása határozza meg a tesztek végrehajtását. A nem megszokott állapotokat vagy ritkán használt rendszerrészeket nem tartalmaz ez az előre látható eloszlás. Emiatt különösen a biztonságkritikus rendszerek esetében, gondos figyelmet kell fordítani azon, amelyek a rendszertől ritka, de lényeges körülmények között várnak beavatkozást. Más megközelítés alapján a véletlenszerű tesztesetek csak a bemeneti tartományra vonatkozóan alkalmazhatóak.

Amióta a tesztesetek száma igen nagy lehet, néha automatizált tesztelő eszközök kerülnek használatra a tesztbemeneti adatok biztosítására. Mivel a specifikáció nem szükséges, ez az eljárás gyengén dokumentált rendszerek esetében alkalmazható, amelyek jellemzően nem idegenek néhány COTS terméktől.

Ne feledjük, hogy jelentős az ellentmondás a megfelelő véletlenszerű vizsgálatok és a mennyiségi szoftver-megbízhatóság között. A szoftver megbízhatóság nem könnyen számszerűsíthető, így a vita dúl.

6.6. Performance Testing – Teljesítmény teszt

A teljesítmény teszttel a COTS szoftver számos valósidejű tulajdonsága vizsgálható, mint például a válaszidők, memória felhasználás, adatáramlás sebessége stb. A teljesítmény egy kritikus követelmény a valós idejű beágyazott rendszerekkel szemben. Nem megfelelő időzítések esetében a rendszer képtelen lehet ellátni feladatát és hibaállapotba kerülhet a rendszer. Ezért is fontos a teljesítmény tesztek elvégzése annak érdekében, hogy a COTS szoftverek alkalmazása mellett meghatározhatóak legyenek azok a terhelés szintek, ahol a rendszer még képes ellátni feladatát. Azonban külön figyelmet kell fordítani arra is, hogy COTS szoftverek esetében nehéz biztosítani a teljesítmény tesztek reprodukálhatóságát és ezáltal nehéz a szoftverek helyességét bizonyítani a rendszer szempontjából legrosszabb (worst-case values) esetekben.

6.7. Formal Methods

A DO-178B/ED-12B a formális módszereket úgy definiálja, mint „leíró jellegű jelölések illetve analitikai módszerek felhasználása a műszaki rendszerek matematikai modelljeinek leírására, a rendszer fejlesztésére."

A formális módszerek fejlesztési eszközt biztosítanak a rendszer leírására annak specifikációs, tervezési és implementációs fázisaiban. Az ilyen leírás eredményének szigorú jelöléseit lehet matematikai elemzés alá vetni, a különböző osztályú ellentmondások vagy helytelenségek felfedésére. A gyakorlatban a formális módszerek alkalmazása a rendszer vagy alrendszer komplexitása által korlátozott. A legtöbb kritikus rendszerrel kapcsolatban egy formális matematikai megközelítés további bizonyosságot adhat annak tulajdonságairól.

A formális modellek általánosságban csak a rendszerviselkedés néhány aspektusát írják le. A fejlesztés verifikációs oldalán nehéz annak biztosítása, hogy a futó rendszer aktuális viselkedése megegyezzen a formális modellben leírttal. Az egyezőség olyan paraméterek függvénye, mint a programozási nyelv, a fordító, az operációs rendszer és a hardver. Nem lehet azt feltételezni, hogy a program megfelelően működik egyszerűen azért, mert a magas-szintű forráskód és a specifikáció között ez már bizonyított.

6.8. Reverse Engineering

Az alkalmazott COTS szoftverekkel kapcsolatos hiányzó adatok megszerzésének egy módszere lehet a reverse engineering alkalmazása. Ez egy igen nehéz és időigényes feladat, melynek erőforrás igénye megközelítheti egy új szoftver fejlesztésénél fellépőket. Ráadásul a reverse engineering esetén semmi sem garantálja számunkra, hogy a végzett munka hibamentes volt, így a kapott adatokból valóban az eredeti COTS rendszer rajzolódott ki. Ettől függetlenül amennyiben jól végezték el a feladatot a módszer eredményeként kapott sw architektúra, forráskód stb. megfelelő bizonyíték lehet arra, hogy az ismert szabványok szerint tanúsítható legyen a szoftver az adott felhasználásra. Sőt a reverse engineering eredményeként kiderülhet, hogy a COTS szoftver fejlesztői esetlegesen egy adott ismert szoftver szabvány alapján fejlesztették a programjukat, így további bizonyítékok kerülhetnek elő melyek könnyebben tanúsíthatóvá és használhatóvá teszik az adott COTS szoftvert.