Automatizált biztonsági tesztelés tapasztalatai Trusted Computing területen

KŐSZEGI GÁBOR, TÓTH GERGELY, HORNÁK ZOLTÁN

BME Méréstechnika és Információs Rendszerek Tanszék, SEARCH Laboratórium
{gabor.koszegi, gergely.toth, zoltan.hornak}@mit.bme.hu

Lektorált

Kulcsszavak: OpenTC, Trusted Computing, automatikus biztonsági tesztelés, Flinder

Ez a cikk egy esettanulmány: a SEARCH Laboratórium által fejlesztett automatikus biztonsági tesztelő keretrendszert, a Flindert és a segítségével az EU FP6 OpenTC projektben elvégzett teszteléses hibakeresési feladat eredményeit, valamint annak elvégzése során szerzett tapasztalatokat összegzi. A feladat méreteit jól mutatja az elvégzett több mint 130 ezer teszteset, melyek négy gépen körülbelül két hét folyamatos futtatást vettek igénybe; melynek eredményeként a tesztelés alanyát jelentő 250 ezer soros TSS implementációban számos – közöttük súlyos, kihasználható – biztonsági szempontból veszélyes hibát fedeztünk fel.

1. Bevezetés

A manapság használatos szoftverekkel kapcsolatos biztonsági lyukak jó részét – az egész rendszerhez mérve – egészen apró hibák okozzák, amelyek bárhol előfordulhatnak, ezért kiszűrésük nehéz feladat. Szerencsére a legtöbb gondot okozó, tipikus esetekben (például buffer overflow, integer overflow, printf format string bug) ez a feladat nem reménytelen. A leggyakroribb hibák felderítésében ugyanis alkalmazhatóak automatikus, vagy fél-automatikus módszerek, amelyek a hibák túlnyomó többségét képesek hatékonyan felismerni.

Az ok amiért régebben kevés figyelmet szenteltek az ilyen hibák kiküszöbölésének az az, hogy a szoftverekben maradó programozói hibáknak, csak egy része válik biztonsági szempontból kritikussá és még ezeknek is csak kis hányada az, ami valós veszélyt hordoz magában, azaz a rendszer ellen történő támadás során kihasználható. A programhibák ezen kis hányada tette lehetővé azonban a legtöbb „kártevő” megjelenését: a mai vírusok és a férgek mind ennek köszönhetik létüket. A feltört gépekből szervezett „botneteknek” nevezett hálózatok felelősek a social enineeringre alapuló spam és phishing támadásokért.

A teszteléses hibafelderítés népszerűsége azért is növekedhetett, mert egy komplexebb rendszer teljes egészének formális verifikációja gyakorlatilag kivitelezhetetlen feladat mind időigényessége mind nagyon magas költsége miatt.

A projekt során használt Flinder keretrendszer a dinamikus teszteléses hibafelderítés módszertanát alkalmazza: ezért gyors és hatékony detektálási eszközt jelent a leggyakoribb biztonsági hibák feltárásában.

Jelen cikk egy ilyen automatizált biztonsági teszteléses feladat eredményeit és tapasztalatait foglalja össze: az EU 6. keretprogramjában indított Open Trusted Computing (OpenTC) projekt során a SEARCH Laboratórium tesztelte az Infineon által elkészített Linux alapú TSS implementációt.

2. Trusted Computing

A Trusted Computing leginkább bizalmi számítástechnikának fordítható – a felhasználó számítógépébe vetett bizalomról szól; a számítógép olyan módon történő működtetéséről, hogy a gép tulajdonosa megbizonyosodhasson rendszere integritásáról vagy adatai biztonságáról. Hasonlóan szükséges, hogy egy szoftvergyártó is megbizonyosodhasson arról, hogy a programjait nem módosítják, vagy használják az adott gépen illetéktelenül.

Az újdonság az architektúrában, hogy célhardverrel támogatott (TPM, Trusted Platform Module), amely chipek már néhány éve megtalálhatók a piacon és beépítésük néhány laptop típusba és asztali PC alaplapra már megtörtént. (2006-ban világszerte 60 millió TPM chipet adtak el, 2007-re pedig az IDC szerint a prognózis 120 millió, míg 2010-re 260 millió.)

A chip feladata a biztonsági alapszolgáltatások biztosítása, mint például:

– valódi véletlenszám-generálás,
– aszimmetrikus kulcsok generálása,
– rendszer integritás ellenőrzés,
– kulcsok biztonságos tárolása,
– nyilvános kulcsú tanúsítványok tárolása,
– kriptográfiai algoritmusok (RSA, SHA-1, AES stb.)
– biztonságos interfész,
– bontás ellenálló tokozás.

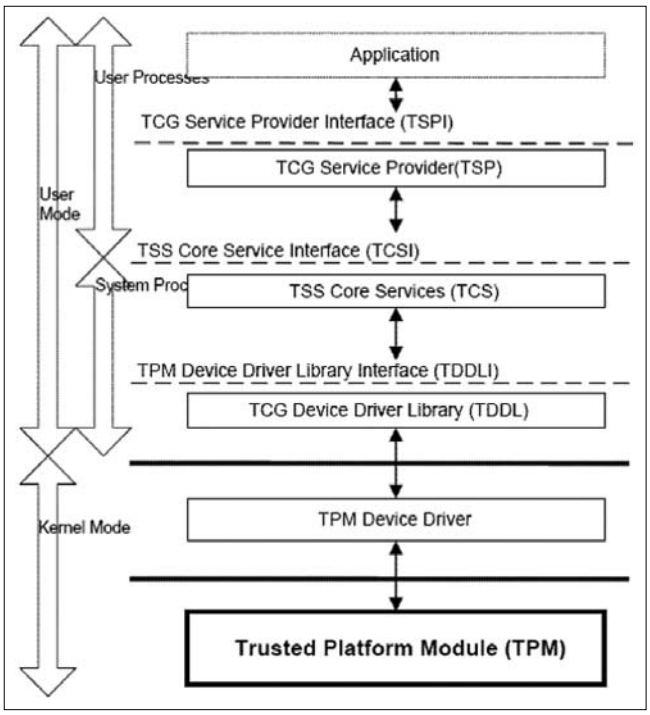
Ezekre a műveletekre épülő rendszer-szoftverek feladata pedig a chip szolgáltatásainak megosztása a párhuzamosan futó folyamatok között, valamint szoftveresen megvalósított többlet-szolgáltatások nyújtása.

3. OpenTC

Az Open Trusted Computing projekt célja a TCG által specifikált platform nyílt forráskódú megvalósítása. A projekt 2005 végén indult és 2008-ra nyilvános Linux disztribúciókba integrált Trusted Computing megoldásokat

fog kidolgozni. Az OpenTC projekt a teljes Trusted Computing architektúrán dolgozik, mind alacsony szintű eszköz-meghajtó programokat, mind felhasználói programokat is fog készíteni.

Elsőként azonban a projekt az alap Trusted Computing funkciókat készítette el.



1. ábra Trusted Software Stack

Az *1. ábra* a TCG platform szoftverének rétegzett felépítését mutatja be, ezen jól látható, hogy a különböző rétegek eltérő jogosultságokkal futnak.

A SEARCH Laboratórium által elvégzett tesztelés legfőbb célpontját képező Core Services (TCS) réteg az összekötő kapocs a felhasználói módban futó programok és az eszköz-meghajtó programok között, ennek megfelelően ennek a rétegnek rendszergazdai jogokkal kell futnia. Emellett, funkcionalitását tekintve ez az egész architektúra legösszetettebb modulja és tulajdonképpen egy hálózati szolgáltatást valósít meg. E tényezőket figyelembe véve, megállapítható, hogy ennek a rétegnek a legszélesebb a támadhatósági felülete, így a programozói hibáktól való mentesítése különösen kritikus a rendszer biztonsága szempontjából.

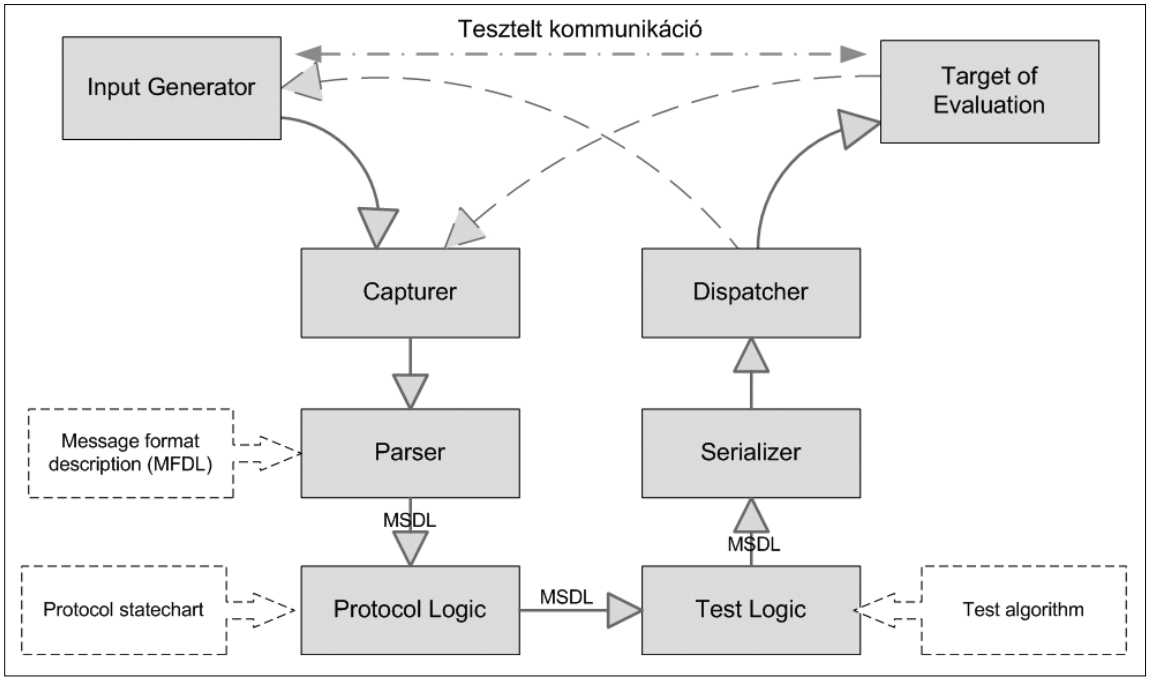
A fenti indokok miatt került sor az Infineon által elkészített implementáció szisztematikus, automatizált tesztelésére.

4. Flinder

A Flinder a SEARCH Laboratóriumban fejlesztett automatikus biztonsági tesztelő keretrendszer, célja a vizsgált rendszerben található biztonsági szempontból kritikus, tipikus programozói hibák (buffer overflow, integer overflow, printf format string bug) megtalálása, mellyel lehetővé teszi a hibák kijavítását, ami által növelhető a tesztelt megoldások minősége és biztonsági szintje.

A feladat elvégzéséhez a tesztelés célpontjának működését dinamikusan, annak futtatásával vizsgálja. A dinamikus tesztelés témakörén belül képes white-box és black-box tesztelésre is, az első módszer a forráskód módosításával történő, függvény szintű hibainjektálást tesz lehetővé, míg a black-box módszer a program bináris kódját használja csak – a belső működések figyelembe vétele nélkül – a szoftvert egészében vizsgálja, hogy az a különböző manipulált bemenetek hatására produkál-e valamilyen nem várt működést (kilép, lefagy stb.).

Hálózati protokollok és programok tesztelésére egyaránt alkalmas paraméterezhető általános célú programmodulokat tartalmaz, ezekből a kívánalmaknak megfelelően építhető fel egy tesztcsomag a konkrét feladat-



2. ábra A Flinder architektúrális felépítése

hoz. A könnyebb alkalmazhatóság érdekében beépítetten támogat sokféle kriptográfiai, tömörítési és kódolási eljárást. Az input/output adatok kezelése könnyen kiegészíthető extra funkciókkal, Python nyelvű szkriptek segítségével.

A tesztelés általános eljárása a következőképpen épül fel a Flinder rendszerben *(2. ábra)*:

- A tesztelés alapjaként szükség van egy legális bemenetre, vagy egy programra, ami ilyeneket képes előállítani (Input Generator).
- Ezután a Capturer által fogadott/elkapott bemenő adatok feldolgozása következhet.
- A Parser modul egy leíró fájl (Message Format Descriptor) alapján dolgozza fel a bemenetére érkező adatokat. Ez a leíró fájl tartalmazza az input adatok formátumának, struktúráinak részletes leírását.
- Miután a Parser átalakította a bementet egy a keretrendszer által értelmezhető általános belső adatszerkezetté (Message Structure Description Language), a Protokol Logic az üzenet tartalma alapján lépteti a vizsgált protokoll működését leíró állapotgépet (Protocol statechart).
- Ezután a Test Logic különböző változtatásokat végezhet az üzenet adatain, (például egész értékek átírása, bufferek hosszának, tartalmának változtatása) annak érdekében, hogy a módosított értékek a tesztelt programban a futás során előidézze a tipikus hibák szimptómáit.
- Ezt követően a Serializer elkészíti a belső adatreprezentációs szerkezet alapján a tesztüzenetet, mely tartalmazza a Test Logic által eszközölt módosítást is.
- Az így előálló üzenetet a Dispatcher modul küldi el a vizsgált programnak (ToE, Target of Evaluation), aztán figyeli annak viselkedését: sikeresen lefut-e, hibaüzenetet küld, lefagy (az operációs rendszer jelez, hogy hiba történt), majd ezek alapján értékeli a teszteset kimenetelét.

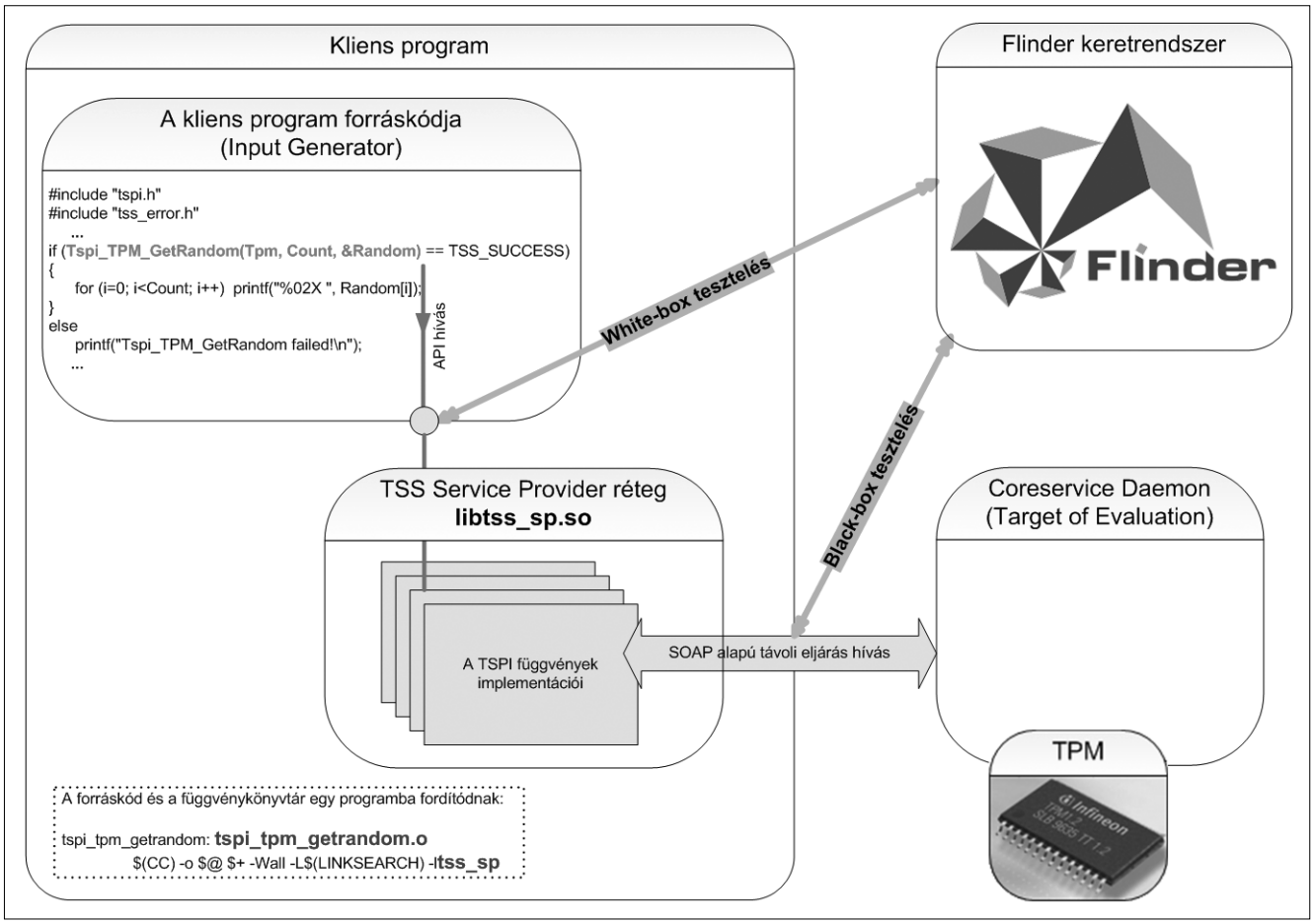
White-box tesztelés

A tesztelés módszere megegyezik mind white-box, mind black-box esetben, különbség a Capturer és Dispatcher modul működésében van.

Forráskód alapú tesztelés esetén a modulok egy részét a tesztelendő programhoz kell fordítani. Ezen modulok célja, hogy a white-box tesztelés során módosítani kívánt belső adatstruktúrát (például egy függvény paramétereit, egy objektum példányt stb.) közvetlenül a Flinder által kezelt MSDL formára konvertálják, majd a módosításokat tartalmazó, Flindertől érkező MSDL alapján az adatstruktúrát módosítsák.

Természetesen az egész tesztrendszert nem szükséges hozzáfordítani a tesztelt programhoz, hiszen az előbb említett modulok a folyamatok közti kommunikációval (IPC) kapcsolódnak a Flinder keretrendszerhez.

3. ábra
A black-box és a white-box tesztelés elhelyezkedése a tesztkörnyezetben



5. Tesztelés végrehajtása

Az OpenTC Infineon TSS implementációban a hibakeresést két különböző szinten valósítottuk meg: első megközelítésben a TCS interfészének black-box típusú tesztelése történt meg, amely a valóságban tulajdonképpen egy távoli-eljáráshívást megvalósító SOAP (Simple Object Access Protocol) alapú protokoll.

Második megközelítésben, minthogy a rendszer funkcióit TSPI szinten egy programozói függvénykönyvtár implementálja, kézenfekvő volt a forráskód alapú tesztelés végrehajtása is, amit a platform tesztprogramjainak módosításával vittünk véghez. Így a Core Services réteg felett elhelyezkedő TSP réteg vizsgálata is lehetővé vált, azáltal, hogy a hibáknak a rendszerbe történő injektálása e réteg interfészén keresztül történt.

A két megközelítés tesztelési környezetben való elhelyezkedését szemlélteti a *3. ábra*.

6. A tesztelés eredményei és tanulságok

Összesen 135 237 teszteset végrehajtására került sor. E hatalmas mennyiségben azonban mindössze 403 bizonyult olyannak, ami a szolgáltatásban hibát okozott.

Az előbb látott két adat között három nagyságrendnyi különbség van; a tesztesetek kevesebb mint 0,3 százalékában volt hiba. Ez egy nagyon fontos eredmény, hiszen ebből világosan megállapítható, hogy kézi teszteléssel lehetetlen lett volna ezeknek a hibáknak a megtalálása – a feladat szó szerint egyenértékű egy tű keresésével a szénakazalban.

Azonban a tesztelt 65 függvényből és 36 SOAP üzenetből 3 függvényben (4,6%) és 4 üzenet feldolgozásában (11%) találtunk hibát, ami jól mutatja azt is, hogy még egy ilyen biztonság-kritikus rendszer fejlesztése közben sem zárhatók ki teljes bizonyossággal a típushibák.

Ezen eredmények is igazolják az automatizált biztonsági tesztelési módszerek fontosságát: a szoftvergyártó más módon nem küszöbölhette volna ki ezeket a hibákat a végső termékéből, melyek bármelyike alkalmas lehetett volna különböző támadások kivitelezésére az egyszerű szolgáltatás-megtagadásos támadásoktól (denial of service, DoS) kezdve egy tetszőleges kód rendszergazdai jogosultságokkal való futtatásáig.

Egy nyilvános EU FP6 kutatás-fejlesztési projekt jó alkalom a biztonsági tesztelés eredményeinek bemutatására, melyek ipari megrendelések esetén szigorú titoktartási nyilatkozatok hatálya alá esnek. A tesztelés során az alábbi három fő tanulság szűrhető le:

- Még a mai biztonság-kritikus alkalmazásokban is követnek el tipikus programozói hibákat, bár ezeket 15 éve ismeri a szakma.
- A most látott TSS implementációhoz hasonló nagy bonyolultságú szoftverek manuális módszerekkel történő hibakeresése a gyakorlatban reménytelen feladat.
- Azonban az automatizált módszerek (biztonsági tesztelők) hasznos eszközök a tipikus hibák elleni védekezésben. Segítségükkel szisztematikusan tesztelhető a célrendszer funkcionalitása, kiküszöbölhetők a tipikus hibák és ezáltal nagyban növelhető a rendszerek biztonsági szintje és minősége.

Irodalom

Flinder whitepaper & test metodology
http://www.flinder.hu/library/index.html
Trusted Computing Group
http://www.trustedcomputinggroup.org
OpenTC
http://www.opentc.net
Buffer overflow
Aleph1, Phrack Magazine (Vol.7, Issue 49, File 14)
http://www.phrack.org./archives/49/p49-14
Heap overflow
Matt Conover, w00w00 Security Team
http://www.w00w00.org/files/articles/heaptut.txt
Integer bugs
Phrack Magazine
(Vol. 0x0b, Issue 0x3c, Phile #0x0a)
http://www.phrack.org./archives/60/p60-0x0a.txt
Exploiting format string vulnerabilities
scut, team teso
http://julianor.tripod.com/teso-fs1-1.pdf