X. FIATAL MŰSZAKIAK TUDOMÁNYOS ÜLÉSSZAKA

Kolozsvár, 2005. március 18-19.

BIZTONSÁGI SZEMPONTBÓL VESZÉLYES PROGRAMOZÓI HIBÁK AUTOMATIKUS FELDERÍTÉSE

Árendás Csaba, Bachraty Gergely

This paper describes an automated testing technique for detecting some special programming bugs that might be the reason of security flaws (buffer overflow, heap overflow, printf bug, integer overflow, signed ness bug etc.). The presented technique uses XML format description of the valid input data, and generates test vectors with the help of this format description and a selection of valid inputs. The novelty in the solution is that the generation of test vectors takes into account the target program's previous responses. If the program survives the previous input values (like buffer size) are increased; otherwise are decreased. After a few steps of successive approximation the procedure determines which size of the input buffer can cause a crash, which usually is a clear sing of unhandled overflows. We will show that applying such interactive test methods several security related bugs, that used to be the reason of security flaws can be detected with black-box tests, where the source code is not known.

Jelen publikáció egy automatikus tesztelési módszert ismertet, amellyel felismerhetőek olyan különleges programozói hibák (buffer overflow, heap overflow, printf bug, integer overflow, signed ness bug), amelyek biztonsági szempontból veszélyesek, mert sebezhetőséget okozhatnak. A bemutatott módszer XML formátum leírást használ a helyes input leírására. A rendszer a formátumleírás és a rendelkezésre álló helyes input gyűjtemény felhasználásával generálja le a következő teszt vektort. Ennek a megoldásnak az újdonsága, hogy a teszt vektor generálásánál figyelembe vesszük a tesztelt program előző válaszait. Ha a program „túléli” az előző bemeneteket (buffer méreteket), akkor azok mérete növelve, máskülönben csökkentve lesz. Több lépéses szukcesszív approximációs közelítés után a tesztelő rendszer pontosan meghatározza mekkora bemeneti buffer méret okozza a program elszállását, ami általában egyértelmű jele egy nem kezelt túlcsoordulásnak. Felismerhető, hogy ilyen tesztelési eljárás alkalmazásával számos olyan biztonsági hiba, ami sebezhetőséget eredményezhet, felderíthető. Mindezt fekete doboz szemléletű teszteléssel, amikor is a forráskód nem ismert.

1. Bevezetés

A programfejlesztés mai technikája mellett a legtöbb szoftverben rendkívül sok olyan programozási hiba marad, amelyek jogosulatlan visszaélésekre adhatnak módot, ezzel alapjaiban aláássák a rendszer védelmét és sokszor a teljes rendszert is kiszolgáltatottá teszik a támadásoknak. A probléma fontosságát és a veszély nagyságát csak növeli, hogy adott esetben a támadó számára egyetlen hiba megtalálása és kihasználása elegendő a védelmi eszközök megkerüléséhez és a számítógép feletti

teljes irányítás átvételéhez. Mivel ezek a hibák rendkívül komoly veszélyt jelentenek a biztonságra, az ellenük való védekezés alapvető fontosságú. Sajnos a közhiedelemben és a gyakorlatban az él, hogy ezen hibák ellen védekezni szinte reménytelen – mondván, hogy az összes hibát megtalálni és kivédeni nem lehet - holott ugyanúgy mint minden más veszélyforrás esetén, a biztonsági szempontból kritikus hibákon belül az egyes típushibák ellen is lehet és ezáltal célszerű is specifikus védelmet alkalmazni.

2. Buffer overflow

Az egyik legismertebb [23]., ebbe a családba tartozó hiba a buffer overflow [1]., [2]., [22]., ami hagyományos programozási nyelvekben (C, C++, Pascal) egy gyakran elkövetett hiba, amikor egy fix hosszúságúra lefoglalt tömb, illetve buffer határait a program nem ellenőrzi és így bizonyos helyzetekben (tipikusan valamilyen túlzottan hosszú bementeti érték hatására) a tömb számára lefoglalt memóriartartományon kívül is felülír értékeket ([1].ábra.).

```
void vulnerable_function(char* array){
    int a=1, b=2, c=3;
    char buffer[12];
    strcpy(buffer, array);
}
```

1. ábra Buffer overflow hibát tartalmazó függvény

Sajnos a gyakorlati életben egy támadó meglepően egyszerű módszerekkel úgy manipulálhatja ezt a nem kívánt helyzetet, hogy az általa készített tetszőleges programkód lefuttatását is el tudja érni, aminek eredménye képpen képes átvenni a teljes irányítást a számítógép felett ([2].ábra).

2. ábra Buffer overflow hiba szemléltetése és kihasználása

3. Védekezési lehetőségek

Amint a [3].ábrán látható, a védekezési lehetőségeket az alábbi felosztásban csoportosítottuk:

Prevenció: Olyan eszközöket, megoldásokat alkalmazunk, aminek következtében a hiba fel sem léphet. Ebbe a kategóriába sorolhatóak legtipikusabban a programnyelvi eszközök [5]., [6]., [7]., [8] és a tesztelő programok [14]., [21].

Detekció: A kész, adott esetben eladott szoftverben való hibakeresés. Ez lehet mind forráskód [14]., [21]. mind futtatható állomány [12]., [15]. szintjén.

Kihhasználás megakadályozása: Ez esetben nem kíséreljük meg megkeresni a hibát, viszont a hiba támadásra való kihasználását megnehezítjük vagy ellehetetlenítjük [13]., [16]., [18]., [19]., [20]., [15].

	Prevention	Detection	Prevent. Exploitation
Development	Development, CC, 3D3, testing	testing	
Compiler	JAVA, .NET	Visual Studio 6.0, 7.x	gcc, Visual Studio 6, 7.x
Application		ITS, SNORT	SecureStack
Operating system		Libsafe	PAX, BSD, XPSP2 Libsafe, NSA, OW RSBAC Adamantix

3. ábra Védekezési lehetőségek csoportosítása

4. Interaktív Tesztelő Rendszer

Az interaktív tesztelő rendszert az előbbi felsorolás szerint a detekció (felismerés) kategóriába sorolhatjuk. Szemben a hasonló célt szolgáló megoldásokkal [3]., [10]., [11]., [17]., [12]., a tesztelő rendszer XML formátum leírást használ a helyes input leírására. A formátum leírásra egy legjobb példa egy képnéző program tesztelése esetén a bemeneti fájlok, azaz a képek strukturális leírása. BMP képformátum esetén az első két karakter rögzített (BM), majd következnek a fejléc további mezői, majd pedig a kép maga pixelenként. A formátum leírás ezt a struktúrát építi fel előre definiált formális elemekből. Ennek köszönhetően sokkal célzottabban lehet olyan teszt vektorokat generálni, amelyek ilyen hibák létezését kutatják. Emellett a formátum leírás nagy mértékben csökkenti a feleslegesen legenerálandó tesztvektorok számát, hiszen egy meglévő struktúrához lehet igazodni, annak a helyességét vizsgálva. A rendszer a formátumleírás és a rendelkezésre álló helyes input gyűjtemény felhasználásával generálja le a következő teszt vektort. Ennek a megoldásnak az újdonsága, hogy a teszt vektor generálásnál figyelembe vesszük a tesztelt program előző válaszait. Ha a program „túléli” az előző bemeneteket (buffer méreteket), akkor azok mérete növelve, máskülönben csökkentve lesz. Több lépéses szukcesszív approximációs közelítés után a tesztelő rendszer pontosan meghatározza mekkora bemeneti buffer méret okozza a program elszállását, ami általában egyértelmű jele egy nem kezelt túlsordulásnak. Természetesen nem csak egy buffer lehet a programban, általános több bufferes esetben is hasonlóan járunk el. Mindaddig növeljük egyszerre minden egyes buffer méretét, amíg hibát nem érzékelünk (valahol, valamelyik, vagy akár több buffer is hibát okozott). Ez esetben visszalépünk az előbbi állapothoz és csak szelektíven növeljük bizonyos bufferek méretét. Ezen lépések ismétlésével pontosan meghatározható, hogy mely bufferek sebezhetőek, és pontosan mekkora input esetén. A formátum leírással párhuzamosan kezelve, meg is nevezhető az adott mező, vagy mezők, amik a hibát okozták.

Irodalomjegyzék

- [1]. Michael Howard and David LeBlanc: Writing secure code, Microsoft Press, Redmond, Washington, 2003, pages 51-68, 127-170, 615-626
- [2]. Aleph One: Smashing the Stack for Fun and Profit. In Phrack Magazine, 1996
- [3]. Andre Zuquete: StackFences: a run-time approach for detecting stack overflows. In 1st International Conference on E-business and Telecommunication Networks (ICETE 2004), 2004
- [4]. Security Enhanced Linux, National Security Agency, www.nsa.gov/selinux
- [5]. Secure Programing, www.splint.org, <http://lclint.cs.virginia.edu/usenix01.pdf>
- [6]. Brandon Bray: Compiler Security Checks In Depth. In Microsoft Developer Network, 2002
- [7]. FreeBSD, http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/index.html
- [8]. SPLINT - Secure programing Lint, www.splint.org
- [9]. Blexim: Basic Integer Overflows. In Phrack Magazine, 2002
- [10]. David Larochelle, David Evans: Statically Detecting Likely Buffer Overflow Vulnerabilities. In USENIX Annual Technical Conference, 2001
- [11]. Timothy Tsai and Navjot Singh: Libsafe 2.0: Detection of Format String Vulnerability Exploits. In libsafe-2.0 White Paper, 2001
- [12]. BFBTester - Brute Force Binary Tester, <http://bfbtester.sourceforge.net>
- [13]. Adamantix (trusted Debian), www.adamantix.org
- [14]. Flaw finder - source code security scanner, www.dwheeler.com/flawfinder/
- [15]. Snort, Intruder Detection System, <http://www.snort.org>
- [16]. Openwall - www.openwall.com/linux
- [17]. Olatunji Ruwase and Monica S. Lam: A Practical Dynamic Buffer Overflow Detector, in Proceedings of the 11th Annual Network and Distributed System Security Symposium, 2004
- [18]. GR security kernel patch - www.grsecurity.net
- [19]. RSBAC – Rule Set Based Access Control, <http://www.rsbac.org>
- [20]. PAX kernel patch, <http://pageexec.virtualave.net>
- [21]. RATS, www.securesoftware.com
- [22]. Pierre-Alain Fayolle, Vincent Glaume: A Buffer Overflow Study – Attacks & Defense, Networks and Distributed Systems, www.securiteam.com, 2002
- [23]. Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie and Jonathan Walpole: Buffer overflows: Attacks and Defenses for the Vulnerability of the Decade. DARPA Information Survivability Conference and Exposition, 2000

Árendás Csaba, Bachraty Gergely

Budapesti Műszaki és Gazdaságtudományi Egyetem, Villamosmérnöki és Inf. Kar, hallgató
arendas@sch.bme.hu, (+36) 70 242 7069, gregorius@freemail.hu, (+36) 205470105