EGY ÚJSZERŰ HOZZÁFÉRÉS SZABÁLYOZÁSI MÓDSZER A LINUX OPERÁCIÓS RENDSZER KERNELÉBEN

A NOVEL ACCESS CONTROL METHOD IN THE KERNEL OF THE LINUX OPERATING SYSTEM

*Vincze Dávid**

ABSTRACT

This paper presents a novel access control method, which can help in situations where long-running processes take care of many independent and untrusted data or program code in a sequential execution order. After a short overview of the basics of access control methods, the concepts of the new method are introduced. This new method tends to be a both fast and secure solution. Also the implementation of this method in the Linux kernel is briefly explained. Finally an application example making use of this new method is presented, which can be useful in real world scenarios. In this example the Apache web server was modified to make use of the introduced new method.Opposed to other solutions which either have security or performance weaknesses, this new method preserves the required level of security while causing only a very low performance overhead.

1. BEVEZETÉS

A több felhasználós számítógép rendszerekben elsődleges szempont az adatok, illetve az alaprendszer védelme, mind a felhasználók, mind pedig a külső hatások ellen. A legtöbb biztonságot fokozó megoldás, a biztonsági szintek növekedésével jelentősen csökkentheti a rendszer használhatóságát, illetve teljesítményét. Az üzemeltetők számára hamar világossá válik, hogy a teljesítmény és a biztonság együttes növelése egymást kizáró tényezők. A teljesítmény általában fontosabb szerepet játszik a döntésekben, mivel gazdasági szempontból egyértelműen ez az előnyösebb: gyorsabb végrehajtás, több előfizető egy kiszolgálón, kevesebb erőforrás elegendő ugyanannyi ügyfélhez, stb. Ilyen módon a biztonság sokszor háttérbe szorul, holott néha a biztonsági hiányosságok okozta anyagi kár nagyobb, mint amennyit a megelőzésre lehetett volna fordítani, nem beszélve az elszenvedett presztízs, reputáció veszteségről.

Jelen cikk egy olyan újszerű megközelítést mutat be a hozzáférés szabályozás olyan lehetőségére, ahol a biztonság fokozása mellett a teljesítmény veszteség minimális, illetve a használhatóság sem csökken, sőt bizonyos esetekben bővül a használhatóság. Az első fejezetben egy rövid áttekintés olvasható a hozzáférés szabályozás lehetőségeiről egy modern, Linux kernel alapú rendszeren, majd bemutatásra kerül az ajánlott új módszer és végül az új eljárás egy lehetséges gyakorlati alkalmazási példája.

2. HOZZÁFÉRÉS SZABÁLYOZÁS

Jogkör ellenőrzés szempontjából a hagyományos UNIX-ok – a Linux-ot is ide sorolva – felhasználói módban két jogosultsági szintet különböztetnek meg: egyik a rendszergazdai szint (0-s azonosító), a másik a normál felhasználói szint (nem 0-s azonosító). Az alapvető hozzáférés ellenőrzése egyszerű: a rendszergazdai szinten futó folyamatoknál (processzeknél) az ellenőrzés elmarad, viszont a normál felhasználói szinten a felhasználói és csoportazonosítók kerülnek összevetésre az elérni kívánt erőforrás által megkövetelttel. Az összevetés három különböző módon ellenőrzi a hozzáférést: először az erőforrás tulajdonosának azonosítójával veti össze a kezdeményező felhasználó azonosítóját, ugyanez a lépés megtörténik a csoport azonosítók összevetésével, majd utolsó lépésben pedig, ha egyiknél sem volt egyezés, akkor a csoport és tulajdonoson kívül az egyéb („mások”) felhasználók számára is definiálhatóak külön hozzáférési jogosultságok. Utóbbinál fontos megjegyezni, hogy ebben a csoportban nem tesz különbséget a felhasználók között, mindenki egyenrangúan tartozik a „mások”-ba.

Ezen felül egyes erőforrás típushoz (pl. fileok, IPC erőforrások) úgynevezett ACL-ek (Access Control List) [6], azaz hozzáférés szabályozási listák rendelhetőek. Ezek a listák a hagyományos tulajdonos/csoport/mások hármason felül további jogosultságokat határozhatnak

**egyetemi tanársegéd, Miskolci Egyetem, Ált. Inf. Tsz.*

meg konkrét felhasználók vagy csoportok számára, ezzel kiküszöbölvén az említett tradicionális modell hiányosságait.

A POSIX (Portable Operating System Interface) kompatibilis több felhasználós operációs rendszereken a felhasználók elkülönítése érdekében minden felhasználóhoz rendelve van legalább egy felhasználói azonosító (uid – userid), valamint egy alapértelmezett csoport (gid – groupid), amibe beletartozik. Ezek alapján történik az alapvető azonosítás és jogkör ellenőrzés. A POSIX.1-ben [5] szabványosított *setuid()* / *seteuid()* / *setreuid()* / *setresuid()* rendszerhívások lehetővé teszik az aktuális processz felhasználói azonosítójának megváltoztatását. Ugyanígy léteznek rendszerhívások a processz csoport azonosítójának megváltoztatására is: *setgid()* / *setegid()* / *setregid()* / *setresgid()*.

Direkt módon abban az esetben szokás használni, amikor egy rendszergazda által indított és így rendszergazdai jogosultságokkal futó programnak csak néhány dolog elvégzéséhez van szüksége a privilegizált jogokra. Miután ezeket a feladatokat elvégezte a *setuid()* rendszerhívás meghívásával átvált egy normál felhasználói azonosítóra, így megszabadul a rendszergazdai jogosultságoktól és folytatja tovább a futását. Fontos, hogy miután megszabadult a processz a rendszergazdai jogosultságoktól, már nem lesz joga újból sikeresen végrehajtani a *setuid()* rendszerhívást, így már nem lesz képes újból átállni a processz más felhasználói azonosítóra.

A POSIX.1e [6] szabvány definiálja a capabilities rendszert, miszerint a fentinél (rendszergazda vs. nem rendszergazda) kifinomultabb jogkör kiosztás és ellenőrzés is használható. A POSIX.1e szabvány nem készült el teljes egészében, csak egy tervezet készült róla, és később azt is visszavonták, de ettől függetlenül a capabilities rendszer több operációs rendszeren is részben megvalósított. A capabilities rendszer több privilégiumot, képességet határoz meg, amelyek bármely processzre ráruházhatóak, bármely processztől elvehetőek, felhasználótól függetlenül. A Linux alatti megvalósításban [7] használható képességek közül néhány:

- CAP_SETPCAP: Egy másik processz képességinek beállításához való képesség. Természetesen csak az aktuális processz képességeinek részhalmazából lehet továbbadni, illetve elvenni. Felhasználói azonosító ellenőrzés a két processz között nem történik, így a használata kockázatos lehet.Éppen ezért újabb kernelekben már nincs használatban.
- CAP_SETUID: Tetszés szerinti változtatását enged meg az aktuális processz csoport azonosítóján (*setuid()*hívások engedélyezése).
- CAP_SETGID: Tetszés szerinti változtatását engedi meg az aktuális processz csoport azonosítóján és mellékcsoport azonosító listáján. A *setgid()* rendszerhívások megengedése.

Alaphelyzetben a rendszergazda az összes lehetséges képességgel rendelkezik, a normál felhasználók pedig nem rendelkeznek egyetlen képességgel sem. Abban az esetben, amikor rendszergazdai jogosultsággal indul egy program, akkor menet közben átállhat egy másik normál felhasználóra (lásd korábban *setuid()*), viszont a képességek közül megtarthat kijelölt capability-ket normál felhasználóként is. Előfordulhat, hogy az adott képességre egész élete során szüksége van a folyamatnak, viszont amennyiben csak pl. inicializációhoz (pl. naplófile megnyitása vagy 1024 alatti TCP/UDP porthoz kötés), akkor a feladat elvégzése után a képesség eldobható.

A bemutatott megoldások különböző megközelítéseket, módszereket alkalmaznak, de mindegyik megoldásnál az a közös pont, hogy az alapján dönti el a hozzáférés jogosultságát, hogy a kérvényező folyamatnak éppen aktuálisan mi a felhasználói, illetve csoport azonosítója.

A hozzáférés szabályozás lehetőségeinek rövid áttekintése után bemutatásra kerül a következő fejezetben egy újszerű módszer, ami egy lehetséges megoldást ad a futásközbeni felhasználói azonosító váltogatás biztonságossá tételére.

3. A JAVASOLT ÚJ MÓDSZER

A módszer lényege a futásidőben való felhasználói azonosító váltogatás ugyanazon a processzen. Az előző fejezetben megismert módon rendszergazdai felhasználói azonosítóról bármikor át lehet váltani bármelyik normál felhasználói azonosítóra. Viszont innentől kezdve már nincs lehetőségea processznek újból változtatni a felhasználói azonosítóján. Ellenben ez jól kihasználható olyan alkalmazásoknál, amelyek nem megbízható adatokkal dolgoznak, netán nem megbízható külső programkódrészeket futtatnak. Megoldás lehet a CAP_SETUID beállítása az adott processzre, hogy továbbra is legyen jogosultsága a *setuid()* rendszerhívás családot sikeresen végrehajtani. Ez biztonsági szempontból hátrányos lehet, hiszen így maga a nem megbízható kódrészlet is kérvényezheti az felhasználói azonosító átállítást, holott csak adott felhasználói azonosítóval volna szabad futnia. Tehát ez a megoldás nem kielégítő. Célszerű volna egy olyan új operációs rendszer (kernel) funkciót implementálni, ami adott feltétel teljesülése esetén visszaadhat meghatározott jogosultságokat a futó processz számára (pl. felruházza CAP_SETUID képességgel).

Az alapvető problémát igazából annak az egyértelmű eldöntése jelenti, hogy éppen hol jár a végrehajtás egy adott processzen belül. Nem lehet tudni, hogy épp

megbízható, vagy épp egy kívülről (hálózatról, más felhasználótól) származó, nem megbízható kódrész kérvényezi-e a jogosultságok visszakérését.

Ezért keresni kell egy olyan módszert, ami különbséget tud tenni a két állapot között, hogy megbízható kód fut-e éppen vagy sem. Megfelelő megoldásnak lehet az alapján különbséget tenni, hogy éppen milyen memóriacímen jár a végrehajtás a kérvényező processzben. Tehát meg kell vizsgálni minden egyes jogosultág visszakövetéskor, hogy honnan lett kérvényezve az adott processzen belül.

A vázolt problémákat nem lehet kizárólag felhasználói térben futó programokkal megoldani.Egy kernel térben futó eljárásra is szükség van a megvalósításhoz, mivel a kernel fogja meghozni a döntést, hogy az adott processz megkaphatja-e a többlet jogosultságot, illetve csak a kernel képes feljogosítani egy processzt új képességekkel. A továbbiakban egy olyan új rendszerhívás kerül bemutatásra, ami a felhasználói térbe való visszatérési cím alapján dönt és adott jogosultsági szintre emeli a hívó processzt.

A gyakorlatban ez úgy valósult meg, hogy az új rendszerhívás (*getmyretaddr()*) meghívásakor – mivel a kernelbe lépés előtt eltárolódik a veremben (stack) az a cím, ahová majd a hívás végeztekor vissza kell térni a kernelből – a veremben visszafelé haladva megkeresi a visszatérési címet. Ha ez a cím egy korábban megbízhatónak minősített címről jön, akkor többlet jogokhoz juthat a processz, ha ugyanez a hívás máshonnan hívódik meg, nem lesznek kiváltságai a processznek.

Egy rendszerhívás meghívásakor a processz kontextus egy része is lementődik a verembe. Miután a rendszerhívás véget ért, a vezérlés egy *ret* instrukcióval tér vissza a rendszerhívás utáni instrukcióra a hívó processzen belülre. Azt pedig, hogy hová térjen vissza, a veremből olvassa ki. Ez utóbbi címet kell megkeresni, ezt használhatjuk hívó címként, (valójában ez az éppen utána következő instrukció címe). A hívó cím birtokában úgy fog megtörténnia jogosultság ellenőrzés, hogy a *getmyretaddr()* rendszerhívás első meghívásakor inicializálódik a processz kontextushoz újonnan hozzáadott *retaddr* nevű változó, aminek az alaphelyzetbeli értéke 0. Az inicializálás csak akkor lesz sikeres, ha a processz rendelkezik a CAP_SETUID és a CAP_SETGID képességekkel. A későbbiekben a *getmyretaddr()* hívásakor ha a *retaddr* értéke nem 0, és az eltárolt *retaddr* és az aktuális hívócím megegyeznek, akkor a processz felruházódik a CAP_SETUID és CAP_SETGID képességekkel. Ez után már a felhasználói térben futó program dolga, hogy átállítsa a felhasználói és csoport azonosítókat és megszabaduljon a két képességétől, hogy a nem megbízható kód résznek ne legyen csak minimális jogköre.

Mindenképpen említést kell tenni a módszer okozta, újonnan nyílt biztonsági kockázatokról és a lehetséges védekezési módokról.

Az egyik ilyen a futó kód felülírásának a lehetősége. A processz kontextusban tárolt *retaddr* értékét közvetlenül nem lehet lekérdezni a kerneltől, ebből adódóan azt egy felhasználói térben futó program nem tudhatja meg ilyen módon. Azonban a saját programkódját tudja olvasni a processz, így a *getmyretaddr()* hívás címe visszafejthető. Így ha a megfelelő memóriacímre saját programkódot illeszt be egy lehetséges támadó, akkor a CAP_SETUID és CAP_SETGID képességek birtokába juthat, mivel ki tudja kerülni a képességek eldobását. Alaphelyzetben az a memória rész, ahol a kód fut, nem írható, de ezt át tudja állítani az *mprotect()* rendszerhívással a támadó, hogy írhatóvá váljon. Ennek kiküszöbölésére beillesztettem egy ellenőrzést az *mprotect()* rendszerhívás kódjába: ha olyan memórialapra kér a processz írási jogot, amibe beleesik a *retaddr* értéke, akkor a rendszerhívás nem fog végrehajtódni, és hibával tér vissza.

Egy másik kijátszási módszer lehet, ha már egy processz kontextusában inicializálva van a *retaddr* változó és ez a processz végrehajt egy *exec()* rendszerhívást, ami új kóddal írja felül a processz futtatott kódját. Ilyenkor a *retaddr* inicializálva marad, így ha a megfelelő címről hívódik meg a *getmyretaddr()* rendszerhívás, akkor jogtalanul hozzájuthat a két képességhez. A védekezési módszer egyszerű: az *exec()* rendszerhívásnak ki kell nulláznia a *retaddr* értékét.

A harmadik eljárás a veremben lévő adatok meghamisítása lehet. Erre azért van lehetősége a feltételezett támadónak, mert a rendszerhívás a glibc *syscall()* csomagoló függvényével lett meghívva. Ha közvetlenül hívja meg a támadó a rendszerhívást, akkor a kernel által *retaddr*-nak vett címet meg tudja hamisítani. Megoldás, hogy az eljárást használó processzből is közvetlenül kell meghívni a *getmyretaddr()* hívást.

4. ALKALMAZÁSI PÉLDA

Napjainkban az Internet második legfontosabb szolgáltatása (az e-mail után) a web. A különböző webalkalmazások felépítése változatos lehet, különböző programnyelveken, nem ritkán egyedi keretrendszereket felhasználva íródnak, számos különböző technológia alkalmazásával. Külső közös kapcsolódási pontjuk a web szerver, amely az egész webalkalmazást egy szabványosított interfészen keresztül elérhetővé teszi a hálózaton. Alapvetően két fajta kiszolgáló (itt erőforrás értelemben) figyelhető meg. Egyik ahol egy dedikált alkalmazás fut, ami egy megbízható felhasználóhoz, ügyfélhez tartozik. A másik típus, pedig, ahol sok egymástól különböző, egymástól idegen, nem megbízható felhasználó számos, egymáshoz semmilyen

módon nem kapcsolódó webalkalmazása fut. Ez utóbbival foglalkozunk a továbbiakban.

Több felhasználós rendszereken a legtöbb esetben ezeket az alkalmazásokat kiszolgáló web szerverek a gyors feldolgozás érdekében minden felhasználó programjait, szkriptjeit ugyanazzal a közös felhasználói azonosítóval és jogkörrel futtatják. A futtatott szkriptek így egyenrangúak, függetlenül attól, hogy melyik felhasználóhoz tartoznak, mivel mind a web szerver közös jogosultságával fut. Ez sok esetben kényelmetlen lehet és rosszindulatú felhasználók ezt könnyedén kihasználhatják [2].

Az egyik legelterjedtebb web szerver az Apache HTTPD [1], így kézenfekvő választás ezen tesztelni a bemutatott új eljárást. Tradicionálisan az Apache webkiszolgálók a *prefork* modell (MPM) szerint működnek. Ebben a modellben egyszerre több, meghatározott számú gyermek processzt indít el és felügyel a web szerver fő processze. Egy-egy gyermek processz élete során sok kérést ki tud szolgálni, tipikusan több százat, de akár ezreket is, így nem szükséges minden kéréshez új processzt létrehozni, inicializálni, ami jelentős erőforrás megtakarítást jelent.

A fentebb leírt problémák kiküszöbölésére számos megoldás látott napvilágot az Apache web szerverhez [3], [4], azonban jelentős teljesítmény csökkenéssel vagy más jellegű biztonsági kockázattal jár az alkalmazásuk. Az előbbiekben bemutatott új módszert alkalmazva az alap probléma olyan módon hidalható át, hogy egyszerre gyors és biztonságos környezetben futhatnak a webalkalmazások.

A *prefork* MPM-be került be az előbbiekben bemutatott új módszer implementációja. Az alapötlet röviden a következő: az adott gyermek processz alaphelyzetben rendelkezik a *CAP_SETUID* és *CAP_SETGID* képességekkel, a kérés feldolgozásának kezdetén meghívja a web szerver a *getmyretaddr()* rendszerhívást, így az első hívásnál az inicializálás és a felhasználó váltás sikeres lesz. A váltás után eldobja a két képességet és kiszolgálja a kérést immáron a megfelelően beállított azonosítókkal. Egy következő kérés feldolgozásakor már nem rendelkezik a két képességgel a *getmyretaddr()* meghívásakor, de ha azt ugyanarról a helyről hívták meg, mint a legelső esetben, akkor újra birtokolni fogja őket, de az azonosító váltások (*setuid()* és *setgid()* rendszerhívások) után ismét megszabadul tőlük és elkezdi feldolgozni a következő kérést.

Ezekkel a módosításokkal az Apache képes lett egy gyors és az eredetinél biztonságosabb módszert alkalmazni, annak érdekében, hogy a hozzá intézett kéréseket szükség eseten más és más felhasználói, illetve csoport azonosítóval szolgálja ki, megkönnyítve a webalkalmazás fejlesztők és üzemeltetők munkáját.

5. ÖSSZEFOGLALÁS

A cikkben bemutatásra került egy újszerű hozzáférés szabályozási módszer. A módszer lényege, hogy a jogosultság eldöntéséhez azt vizsgálja, hogy honnan lett kérelmezve a jogosultság emelés a kérelmező processzen belülről. Az új eljárás a Linux operációs rendszer kernelében lett implementálva. Konkrét alkalmazás példaként pedig a jelenleg legelterjedtebb web szerver (Apache HTTPD) lett felkészítve a módszer használatára. Így az Apache HTTPD alkalmassá válik arra, hogy a különböző felhasználók programjait, szkriptjeit a saját hozzájuk tartozó felhasználói azonosítójával. Így az adott felhasználó jogosultságaival tudja futtatni ezeket a web szerver, más hasonló módszereknél jelentősen alacsonyabb erőforrás igénnyel.

6. KÖSZÖNETNYILVÁNÍTÁS

A bemutatott kutató munka a TÁMOP-4.2.1.B-10/2/KONV-2010-0001 jelű projekt részeként az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg

7. IRODALOM

[1.] NetCraft Ltd. Web Server Survey, April2012: http://news.netcraft.com/archives//2012/04/04/april-2012-web-server-survey.html

[2.] S. GARFINKEL: Web Security, Privacy & Commerce, Second Edition, O'Reilly Media Inc., Jan. 2002, ISBN: 978-0596000455

[3.] S. H. GUNDERSON: The Apache 2 ITK MPM http://mpm-itk.sesse.net (Hozzáférve: 2012. Apr.)

[4.] D. HARA, Y. NAKAYAMA: Secure and High-performance Web Server System for Shared Hosting Service, Proceedings of the 12th International Conf. on Parallel and Distributed Systems (ICPADS'06), Minneapolis, USA, 2006

[5.] IEEE Standard for Information Technology 1003.1-2004: Portable Operating System Interface (POSIX)

[6.] IEEE Standard for Information Technology P1003.1E Draft: Portable Operating System Interface (POSIX)

[7.] B. TOBOTRAS: Linux Capabilities FAQ, 1999 http://ftp.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.4/capfaq-0.2.txt (Hozzáférve: 2012. Apr.)