# AZ APACHE WEBSZERVER BIZTONSÁGI ÉS EGYÉB KIEGÉSZÍTÉSEI

***Vincze Dávid, vincze.david@uni-miskolc.hu***

*Miskolci Egyetem Számítóközpont*

## BEVEZETÉS

Napjainkban egyre szélesebb körben terjednek el a webalkalmazások. Többfelhasználós rendszereken a legtöbb esetben ezeket az alkalmazásokat kiszolgáló webszerverek a gyors feldolgozás érdekében minden felhasználó szkriptjét ugyanazzal a közös felhasználói azonosítóval és jogkörrel futtatják. A futtatott szkriptek így egyenrangúak, függetlenül attól, hogy melyik felhasználóhoz tartoznak, mivel mind a webszerver jogosultságával fut. Ez sok esetben kényelmetlen lehet és rosszindulatú felhasználók ezt könnyedén kihasználhatják, pl. elérhetik egymás szkriptjeinek forráskódjait, adatbázis hozzáférés jelszavait. Könnyen előfordulhat olyan helyzet is, hogy más felhasználóhoz tartozó filet le lehet törölni, vagy felül lehet írni, egyes esetekben le tudják állítani a webszervert, stb.

A gyakorlatban hamar jelentkeznek a közös felhasználó miatt kialakult problémák, leggyakrabban: file feltöltéseknél külön kell a rendszergazdának vagy a felhasználónak a jogosultságokat beállítani az adott jegyzékre; a feltöltött fileokkal műveleteket végezni nehézkes, mivel a webszerver felhasználói azonosítójával jött létre; ezeket a fileokat nem lehet törölni pl. FTP elérésen keresztül; a webszerver felhasználójának quotájához számít bele a file mérete és nem ahhoz a felhasználóhoz, aki a filet feltöltötte; nem szedhető szét felhasználókra az erőforrások igénybevételének korlátozása, illetve naplózása; PHP szkriptekben a mail() függvénnyel küldött levelek küldőjének beállítása, és ehhez hasonló apróságnak tűnő, de komoly problémák. Mivel manapság a webkiszolgálók túlnyomó többsége ezt a felépítést követi, így sokakat érint: internet szolgáltatókat, vállalatok, szervezetek saját gépeit, oktatási intézményeket, stb.

Ezen problémák kiküszöbölésére mutat be lehetséges megoldást az előadás, egyszerre gyors és biztonságos környezetet biztosítva a webalkalmazások számára futásidejű felhasználói azonosító váltogatás alkalmazásával.

Manapság az Apache HTTPD az egyik legismertebb webszerver, és nagy részük Linux operációs rendszer alatt fut. Az Apache webszerver és a Linux kernel nyílt forráskódúak, szabadon módosíthatóak, így lehetőség nyílik ezek forráskódjainak megváltoztatására. Az említett megoldás Linux operációs rendszer alá lett kifejlesztve C és Assembly (x86) programnyelven.

Az előadásban dióhéjban bemutatom a webszerverek, kiváltképp az Apache HTTPD működését, részletezem a futásidőben történő felhasználói azonosító váltás lehetőségeit, a Linux kernel kibővítését, módosítását, az Apache webszerverben szükséges módosításokat és mindezek beállításait, használatát. Bemutatásra kerülnek még a fejlesztés közben létrejött melléktermékek, amik önállóan is megállják a helyüket: a kérések által elhasznált processzor idő naplózása, az aktuális kérések megjelenítése a processzek neveiben, illetve a kiszolgálás szabályozása a rendszer terhelésének függvényében. Végül a különböző módszerek, kiegészítések teljesítményéről, teljesítményének összehasonlításáról és a lehetséges további fejlesztésekről is információt kap az olvasó.

Az elkészült kiegészítések egy része stabilan használható a jelenlegi állapotukban is. Több, a szerző által üzemeltetett nagy forgalmú szerveren hiba nélkül üzemelnek a dolgozatban bemutatott kiegészítésekkel ellátott Apache webszerverek.

# 1. WEBSZERVEREK MŰKÖDÉSÉNEK ÁTTEKINTÉSE

A legelső webszerverek kizárólag közönséges fileokat voltak képesek kiszolgálni. Ez azt jelenti, hogy a webszerver a megfelelő helyen elhelyezett fileokat (pl. dokumentumok, képek, videók) a kéréstől függetlenül mindig teljesen ugyanúgy adja vissza a klienseknek: egy az egyben a file tartalmát. Ezt a kiszolgálási módot nevezik statikusnak.

Manapság már elvárt, hogy az oldalak tartalma bizonyos információkat figyelembe véve jöjjön létre, vagy egy-egy kérés megadott eseményeket tudjon kiváltani. Ezeknek az igényeknek a kielégítésére egy mai átlagos webszerver nem csak szimpla fileokat tud kiszolgálni, hanem képes meghívni a szerveren elhelyezett programokat, értelmezni és végrehajtani szkripteket és azok futási eredményeit közölni a kérést küldő klienssel. Ezt a módot nevezik dinamikusnak. Egy napjainkban átlagosnak mondható weblap egyaránt tartalmaz statikus és dinamikus elemeket is. A dinamikus elemek generálják magát az oldal tartalmát, nem ritkán relációs adatbázisokból futásidőben kinyert információk alapján. A statikus elemek pedig maradnak a képek, stíluslapok, kliens oldali szkriptek, videók, stb.

A netcraft.com legfrissebb felmérése [1] szerint az Interneten a webkiszolgálók több, mint a felén az Apache webszerver különböző verziói futnak, azok közül is a legtöbb Linux operációs rendszer alatt működik. Tradicionálisan az Apache webkiszolgálók a *prefork* modell szerint működnek (a 2.0 előtti verziók kizárólag ezt a modellt támogatták). Ebben a modellben egyszerre több, meghatározott számú gyermek processzt indít el és felügyel a webszerver 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. Hasonló ehhez a *worker* modell működése, azzal a különbséggel, hogy nem processzek, hanem szálak végzik a kiszolgálást. Éppen a szálak miatt nincs elterjedve olyan környezetben, ahol sok idegen felhasználó szkriptjeit szolgálja a webszerver. A *perchild* elnevezésű modell szerint virtualhost-onként jön létre egy-egy gyermek processz, amik más-más felhasználói és csoport azonosítóval futnak, és a gyermek processzeknek vannak változó számú kiszolgáló szálaik. Egyik hátránya, hogy így minimálisan annyi gyermek processznek mindig futnia kell, mint ahány virtualhost konfigurálva van a webszerveren, akkor is ha azokhoz nem érkezik soha sem kérés, ez több ezer virtualhostnál nem feltétlen előnyös. Legnagyobb hátránya, hogy fejlesztése nagyon kezdeti fázisban van, jelenlegi állapotában működésképtelen, és úgy tűnik a fejlesztése végleg abbamaradt a 2.2-es verziójú Apache HTTPD-ben. Egy másik megvalósítása ennek a módszernek a *metuxmpm* [2] volt, de ennek a fejlesztése szintén abbamaradt. Az *mpm_peruser* a *metuxmpm* továbbfejlesztése, ami szintén ezt az elvet követi, de a szálak helyett mindenütt processzeket használ, ami több virtualhost esetén jelentős erőforrás pazarlást eredményez. Egy külső fejlesztésű modul, az *mpm_itk* [3] a *prefork* modellen alapulva képes különböző felhasználói és csoport azonosítóval futtatni a gyermek processzeket, de gyakorlatilag a *prefork* előnye ennél a megoldásnál is elvész, mivel egy processz csak egy hálózati kapcsolatot tud kiszolgálni. Az *mpm_itk*-ról még lesz szó bővebben a későbbiekben.

Dinamikus tartalom kiszolgálására az első elterjedt módszer a *Common Gateway Interface (CGI)* [4] mechanizmus volt. Működése egyszerű, a webszerver nem egy filet olvas be és küldi el a tartalmát a kérés kezdeményezőjéhez, hanem a lekért file egy program, amit a

webszerver elindít és a kérés módjától függően (GET/POST) a neki szánt információt átadja argumentumként, vagy a szabványos bemenetére írja. A program lefutását követően, amit a szabványos kimenetére ír, azt küldi el a webszerver a kérő kliensnek. Ilyenkor külön processz indul el a kiszolgáló gyermek processzből, ami a megadott bináris programot képes futtatni, a futás után pedig megszűnik a processz. Szkripteket is futtathat a szkript interpreterének automatikus betöltésével. Nem használja ki a *prefork* modell előnyét, hiszen minden egyes kérés feldolgozásához teljesen új processzt hoz létre a kért program futtatásához, a processz létrehozás és inicializálás pedig még jobban megnöveli az erőforrásigényt. Egyre inkább kezd háttérbe szorulni ez a megoldás a PHP és a más szkriptnyelvek (pl. Python, Ruby, Perl) terjedésének köszönhetően, amiknek léteznek a webszerverbe ágyazható interpretereik.

A *suexec* megoldás alap elgondolása a CGI mechanizmust egészíti ki egy olyan tulajdonsággal, hogy a webszerver által futtatott programot nem a webszerver, hanem a program tulajdonosának felhasználói és csoport azonosítójával futnak. A megoldás lényege, hogy nem közvetlen a webszerver indítja el az oldalt generáló programot, hanem egy ún. wrapperen keresztül hívja meg a programot. A wrapper egy bináris program és suid bittel rendelkezik a filerendszeren, a wrapper tulajdonosa a rendszergazda, így annak a jogait fogja felvenni futtatáskor. Tehát a wrapper rendelkezik a megfelelő jogokkal a felhasználói és a csoport azonosító átváltásához, a feladata mindösszesen annyi, hogy megváltoztassa a hatályban lévő azonosítókat és csak ezután indítsa el az eredetileg végrehajtandó programot. A wrapper közbeiktatása miatt valamelyest lassabb a kiszolgálás, mint egy normál CGI végrehajtása. A *suphp* a *suexec* átalakítása kifejezetten PHP-s környezetre. Nem enged más interpretert használni, kizárólag csak a PHP értelmezője használható.

A *mod_php* a PHP interpreterét beágyazza az Apache webszerverbe, így pl. a *prefork* modellnél, minden gyermek processzben elérhető lesz. Ez minimálisan, de megnöveli minden gyermek processz memóriaigényét, de cserébe a használatával megspórolható rengeteg processz létrehozás, és az interpreter betöltése, inicializálása minden egyes kérésnél. Jelentősen nagyobb teljesítményt nyújt, mint a CGI-s megoldások, ezért használata széles körben elterjedt. Hátránya ugyanaz, mint a normál CGI-knek, hogy egyazon felhasználói és csoport azonosítóval történik a végrehajtása mindegyik szkriptnek. Valamelyest segít ezen a problémán a PHP *safe_mode* opciójának a bekapcsolása, ami számos korlátozást vezet be a szkriptekre nézve. Hasznos kiegészítés, egyszerű szkriptekhez elégséges, de közel sem nyújt tökéletes megoldást, és sok esetben kényelmetlenségeket okozhat a PHP programozóknak, illetve sok előre elkészített PHP-s webalkalmazás képtelen megfelelően működni *safe_mode* alatt. Hasonló webszerverbe ágyazott interpreterek elérhetőek más szkriptnyelvekhez is, úgy mint *mod_pyhton*, *mod_ruby*, *mod_perl*.

A leírtak ismeretében világossá válik, hogy vagy a teljesítmény, vagy a biztonság mellett kell dönteni a webszerverek üzemeltetőinek, és mivel a teljesítmény általában fontosabb szerepet játszik a döntésekben (egyértelműen érezhető az anyagi oldalon: gyorsabb végrehajtás, több előfizető egy szerveren, kevesebb erőforrás elegendő ugyanannyi ügyfélhez), így a biztonság háttérbe szorul (pedig néha a biztonsági hiányosságok okozta kár nagyobb lehet, mint amennyit a megelőzésre kellett volna fordítani, nem beszélve az elszenvedett presztízs veszteségről).

Az előadás a *mod_php* és a *suexec* alapelgondolásainak az ötvözésével, a *prefork* MPM továbbfejlesztésével és különböző módosításokkal a Linux kernelben, mutat be lehetséges megoldást, mely alkalmazásával mind a teljesítmény, mind a biztonság egyaránt megmarad.

# 2. A FELHASZNÁLÓI AZONOSÍTÓ MEGVÁLTOZTATÁSA

## *2.1. A SETUID()/SETGID() RENDSZERHÍVÁS CSALÁDOK*

A POSIX (Portable Operating System Interface) kompatibilis többfelhasználós operációs rendszereken a felhasználók elkülönítése érdekében minden felhasználóhoz rendelve van legalább egy felhasználói azonosító (uid – user id) és egy alapértelmezett csoport (gid – group id), 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. Az utóbb leírt módon működik az Apache webszerver is, a rendszergazdai jogok ahhoz szükségesek, hogy 1024 alatti TCP portot - 80-as és/vagy 443-as portot (HTTP/HTTPS port) - tudjon lefoglalni és a naplófileokat meg tudja nyitni. Miután ezekkel végzett, normál felhasználóként folytatja tovább a futását.

## *2.2. A CAPABILITIES RENDSZER*

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). A hozzáférés ellenőrzése egyszerű, a rendszergazdai szinten futó processzeknél az ellenőrzés elmarad, a normál felhasználói szinten futóknál pedig a felhasználói és csoport azonosítók kerülnek összevetésre az elérni kívánt erőforrás által megkövetelttel.

A POSIX.1e [6] szabvány 25. fejezete írja le a capabilities rendszert, miszerint a fentinél 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_NET_BIND_SERVICE: 1024 alatti TCP/UDP portokra való bindelést engedélyezi.
- CAP_SETPCAP: Egy másik processz képességinek beállításához való képesség. Természetesen csak az aktuális processz képességeit lehet továbbadni, illetve elvenni. Felhasználói azonosító ellenőrzés a két processz között nem történik.
- CAP_SETUID: Tetszés szerinti változtatását engedi meg az aktuális processz csoport azonosítóján. A setuid() rendszerhívások megengedése.

- CAP_SETGID: Tetszés szerinti változtatását engedi meg az aktuális processz csoport azonosítóján és mellékcsoport azonosító listáján. A setgid() rendszerhívások megengedése.

Természetesen alaphelyzetben a rendszergazda az összes képességgel rendelkezik, a normál felhasználók pedig nem rendelkeznek egyetlen képességgel sem.

## *2.3. A LINUX KERNELBEN SZÜKSÉGES MÓDOSÍTÁSOK*

A vázolt problémákat nem lehet kizárólag felhasználói térben futó programokkal megoldani, egy kernel térben futó eljárásra is szükség van a megvalósításhoz. Ebben alfejezetben egy olyan rendszerhívás kerül bemutatásra, ami a felhasználói térbe való visszatérési cím alapján bizonyos kiváltságokhoz juttatja a hívó processzt. Az itt tárgyalt eljárást használja a következő, 3. fejezetben bemutatott megoldás, a menet közbeni azonosítóváltások megvalósításához.

Az alapvető problémát igazából annak az egyértelmű eldöntése jelenti, hogy éppen hol jár a végrehajtás a kérés kiszolgálásában. Nem lehet tudni, hogy magának a webszervernek a kódja, vagy pl. a *mod_php* által értelmezett kód fut éppen. Mivel egy webszerver processz az élete során mindig olyan tulajdonságokkal rendelkezik, amik alapján nem lehet különbséget tenni, hogy éppen rendes vagy egy felhasználó által írt kódot futtat. Ezért keresni kell egy olyan módszert, ami különbséget tud tenni a két állapot között. Megfelelő megoldásnak láttam az alapján különbséget tenni, hogy éppen milyen memóriacímen jár a végrehajtás a processzben. A gyakorlatban ezt úgy kiviteleztem, hogy egy bizonyos rendszerhívás meghívásakor – mivel a kernelbe lépés előtt eltárolódik a stack-en az a cím, ahova majd a hívás végeztekor vissza kell térni a kernelből – a stack-en visszafelé haladva megkeresi a visszatérési címet a hívás, és amennyiben ez a cím a webszerver rendes kódjába mutat, akkor többlet jogokhoz juthat a processz, ha ugyanez a hívás máshonnan hívódik meg, nem lesznek kiváltságai a processznek.

Első lépésben a visszatérési cím megkeresését implementáltam, egy *getmyretaddr*() nevű rendszerhívással a kernelben. Bemenő paramétere nincs a hívásnak, mindösszesen annyi a feladata, hogy kiírja a stack tartalmát 24 elemnyi (24 x 32 bit) mélységben. Az első két elem kiíratása a kernel üzenetei közé a következő kóddal tehető meg:

```
asm volatile (  "movl (%%esp),%%eax     \n\t"
                "movl %%eax,%0          \n\t"
                : "=m" (tempvar));
printk("0x00: 0x%x\n",tempvar);
asm volatile (  "movl 0x04(%%esp),%%eax \n\t"
                "movl %%eax,%0          \n\t"
                : "=m" (tempvar));
 printk("0x04: 0x%x\n",tempvar);
```

A bemutatott megfigyelési módszer befolyásolja az eredményt az ideiglenes változó használata miatt, ami +4 byte-ot használ a stack-en. A vezérlés egy *ret* utasítással tér vissza a rendszerhívás utáni instrukcióra a webszerver programjában. Azt pedig, hogy hová térjen vissza, a stackről olvassa ki. Ez utóbbi címet kell megkeresni, ezt használhatjuk hívó címként, bár igazából ez az éppen utána lévő cím. A hívó cím birtokában úgy fog kinézni a jogosultság

ellenőrzés, hogy a *getmyretaddr*() rendszerhívás első meghívásakor fog inicializálódni a processz kontextusban egy újonnan hozzáadott, *retaddr* nevű változó, aminek az alaphelyzetbeli értéke 0. Az inicializálás 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, illetve a *retaddr* és a hívócím megegyeznek, akkor a processz felruházódik a CAP_SETUID és CAP_SETGID képességekkel. Ezután a felhasználói térben futó program dolga, hogy átállítsa a felhasználói és csoport azonosítókat, és azután megszabaduljon a két képességétől.

Mindenképpen említést kell tenni az újonnan nyílt biztonsági kockázatokról és a lehetséges védekezési módokról.

Az egyik 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 kódját tudja olvasni a processz, így a *getmyretaddr*() hívás címe visszafejthető. Ilyen módon ha a megfelelő memóriacímre saját programkódot illeszt be a 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 féle eljárás a stack 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, akár a vdso-n keresztül, akkor a kernel által *retaddr-nak* vett címet meg tudja hamisítani. Megoldás, hogy a webszerverből is közvetlenül kell meghívni a *getmyretaddr()* hívást a vdso-n keresztül és nem a glibc *syscall*() függvényével. Illetve a kernelben azt is ellenőrizni kell, hogy a vdso-n keresztül jöjjön a hívás, ami egyszerűen kivitelezhető, hiszen ez a cím a stack 0x40 mélységében megtalálható, a memória térképből pedig megállapítható, hogy éppen hol helyezkedik el a vdso, és ennek a lapnak a 0x410 offszet címén van a visszatérési cím.

# 3. AZ APACHE WEBSZERVER MÓDOSÍTÁSAI

## *3.1. MPM_GETMYRETADDR*

Az alcímben szereplő nevű új MPM a *prefork* MPM továbbfejlesztése a 2.3. fejezetben bemutatott *getmyretaddr*() rendszerhívás használatára. Az alapötlet a következő: a kérés feldolgozásának kezdetén meghívja a webszerver a *getmyretaddr()* rendszerhívást, az adott gyermek processz alaphelyzetben rendelkezik a *CAP_SETUID* és *CAP_SETGID* képességekkel, 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. A továbbiakban az elképzelés implementációjának részletei olvashatóak.

A tényleges kiszolgálást a webszerver gyermek processzei végzik. Minden egyes kérés feldolgozása közben meghívódik a *ap_process_request_internal()* függvény, tehát ez egy megfelelő hely, ahová *getmyretaddr()* hívást el lehet helyezni. Még mielőtt a rendszerhívásra kerülne a vezérlés, meg kell állapítani, hogy milyen felhasználói és csoport azonosítóra kell átállni: ha a konfigurációban meg van adva a *ServeAsUIDGID* direktíva (lásd későbbiekben), akkor azt használja, ha nincs akkor a kiszolgált file tulajdonosának, csoportjának az azonosítóját fogja beállítani (a *mod_userdir* használatánál nagyon hasznos ez a lehetőség). Ha sikertelen a file információinak a megállapítása, akkor 404-es HTTP hibakódot ad vissza a webszerver a kliensnek. Ha a kiszolgálandó file a rendszergazda tulajdona lenne, akkor a kiszolgálást megtagadja a webszerver egy 403-as HTTP hibakóddal.

Teljesítmény fokozás miatt, csak akkor történik azonosító váltás, ha az valóban szükséges, tehát akkor ha a felhasználó, amire váltani kell nem egyezik meg azzal amivel jelenleg a webszerver fut. Ha szükséges a váltás, akkor meghívódik a *getmyretaddr()*. Közvetlen ez után következik a már ismertetett rendszerhívásokkal a felhasználói és csoport azonosítók megváltoztatása, azután a képességek érvénytelenítése. Amennyiben a képességeket valamilyen oknál fogva nem sikerül semmissé tenni, a webszerver belső hibával, 500-as hibakódot közöl a klienssel és a kiszolgálás nem történik meg. Természetesen az esemény naplózásra kerül a webszerver hibanaplójába.

Ezzekel a módosításokkal elkészült a lényegi része a kiegészítésnek. A forráskód lefordítása és megfelelő konfiguráció után egy próba futtatással teszteltem a programkódot. Egy egyszerű HTTP kérést intéztem a webszerverhez. A kernel üzenetei között látható, hogy a keresett memória cím: `userstack5: 0x80a3b39`. A GDB debuggerrel a webszerver egyik gyermek processzére (ebből a szempontból megegyező) akaszkodva ellenőriztem, hogy valóban ez-e a megfelelő visszatérési cím:

```
...
0x080a3b2d <ap_process_request_internal+1668>: movl $0x145,(%esp)
0x080a3b34 <ap_process_request_internal+1675>: call 0x80632b8 <syscall@plt>
0x080a3b39 <ap_process_request_internal+1680>: mov  %edi,0x8(%esp)
0x080a3b3d <ap_process_request_internal+1684>: mov  %edi,0x4(%esp)
0x080a3b41 <ap_process_request_internal+1688>: mov  %edi,(%esp)
0x080a3b44 <ap_process_request_internal+1691>: call 0x8063d08 <setresgid@plt>
...
```

A 0x145 (325) számú (rendszerhívás csomagoló - *syscall()*) függvényéből tér vissza, tehát a jó címre mutat.

Próbaképpen az Apache webszerverbe ágyazott PHP interpreterben, a *libphp5.so*-ban két helyen elhelyeztem egy-egy *getmyretaddr()* rendszerhívást. Az egyiket a *posix_uname()* függvényben, ami a PHP kiegészítések (extension) között található, a másikat a *phpinfo()* információs oldalt generáló függvényben, ami pedig az Apache-PHP összekötő modulban van. A két esetben a két cím a következő volt:

```
userstack5: 0xb7a71566
userstack5: 0xb7c2f1ec
```

Ez a webszerver processzeinek memóriatérképein (ebből a szempontból egyformák a gyermek processzek) már jól látszódik, hogy a *mod_php5*-ből lettek meghívva:

```
...
b7878000-b7ce3000 r-xp 00000000 fe:04 87839431
                  /mnt/work/nws/apache-2.0.63/modules/libphp5.so
b7ce3000-b7d01000 rw-p 0046b000 fe:04 87839431
                  /mnt/work/nws/apache-2.0.63/modules/libphp5.so
...
```

A debuggerrel közelebbről megvizsgálva a két memóriacímet, látható, hogy itt is valóban jók a kapott értékek:

```
...
0xb7a71554 <zif_posix_uname+52>:  js     0xb7a71644 <zif_posix_uname+292>
0xb7a7155a <zif_posix_uname+58>:  movl   $0x145,(%esp)
0xb7a71561 <zif_posix_uname+65>:  call   0xb7de7700 <syscall>
0xb7a71566 <zif_posix_uname+70>:  mov    %ebx,(%esp)
...
...
0xb7c2f1db <zm_info_apache+889>:  call   0xb7b20d03 <php_info_print_table_row>
0xb7c2f1e0 <zm_info_apache+894>:  movl   $0x145,(%esp)
0xb7c2f1e7 <zm_info_apache+901>:  call   0xb7de7700 <syscall>
0xb7c2f1ec <zm_info_apache+906>:  lea    -0x10(%ebp),%eax
0xb7c2f1ef <zm_info_apache+909>:  mov    %eax,0x4(%esp)
0xb7c2f1f3 <zm_info_apache+913>:  movl   $0xb,(%esp)
0xb7c2f1fa <zm_info_apache+920>:  call   0x8085ea0 <ap_mpm_query>
...
```

Az első esetben tényleg a *posix_uname()*, a második esetben tényleg az *phpinfo()* függvényből hívódott meg a *getmyretaddr()* rendszerhívás. Tehát ez a módszer jól használható annak eldöntésére, hogy éppen jogosult-e a webszerver a *CAP_SETUID* és *CAP_SETGID* képesség megszerzésére.

A kiszolgáláshoz beállítandó felhasználó és csoport azonosítót az Apache konfigurációjában lehet megadni, egy újonnan létrehozott direktívával, aminek a neve *ServeAsUIDGID*, paraméterként két számot vár, amik rendre a felhasználói és csoport azonosítók.

Ezekkel a módosításokkal az Apache képes lett egy gyors és biztonságosnak mondható módszert alkalmazni, annak érdekében, hogy a hozzá intézett kéréseket szükség eseten más és más felhasználói, illetve csoport azonosítóval szolgálja ki.

## *3.2. ÖSSZEHASONLÍTÁS AZ MPM_ITK-VAL*

Az *mpm_itk* nevű multi processing module szintén a *prefork* MPM-on alapszik. A lényegi változtatás itt is a felhasználói és csoport azonosító váltogatására irányul. A *prefork* modellt követve, előre létrehoz adott számú gyermek processzt, de nem áll át semleges, nem rendszergazdai jogokkal rendelkező felhasználóra, hanem megmarad addig rendszergazdai jogosultságokkal (csak részben, mert, amiket tud képességet, azokat elveszi magától, de a file eléréseknél 0-s felhasználói és csoport azonosítóval történnek) processz, amíg be nem érkezik hozzá egy kérés, és a kért virtualhost-ban megadott felhasználói és csoport azonosítót fogja felvenni. Ezután már nem képes visszaváltani rendszergazda felhasználóra vagy átváltani

másik felhasználóra. Tehát, amint megszakad a kapcsolat, a processz is megszűnik. Egyazon kapcsolaton belül viszont ki tud több kérést is szolgálni, amennyiben azok azonos hosztnév alá esnek, és engedélyezve van a webszerver konfigurációjában a keep-alive funkció. A keep-alive funkció bekapcsolása nélkül gyakorlatilag minden kérés egy processzt használ el, ami lényeges teljesítmény csökkenést eredményez a hagyományos *prefork* MPM-hez képest: a CGI mechanizmus teljesítményéhez közelít felülről. Keep-alive használatával valamelyest javul a helyzet. Azonban a processzek újrahasznosítására nem képes, mivel a kapcsolatok bontásakor a processz megszűnik, csak egy kapcsolatot tud kiszolgálni egy processz. Az *mpm_getmyretaddr* viszont képes a processzek újrahasznosítására, egy processz ki tud szolgálni több különböző kapcsolaton keresztül érkezett kéréseket is.

# 4. AZ APACHE EGYÉB KIBŐVÍTÉSEI

A bemutatott megoldáshoz szorosan nem kapcsolódó, de teszteléshez és statisztikákhoz jól hasznosítható, önállóan is használható bővítések találhatóak ebben a fejezetben. Az egyik a kérések által elfogyasztott processzoridőt naplózza, egy másik bővítés pedig a webszerver gyermek processzeinek a nevét változtatja meg minden alkalommal, amikor kérés történik, és a processz névbe beleírja a kérés módját és konkrét URL-jét. A harmadik kiegészítés pedig a rendszer terhelésének függvényében tudja szüneteltetni a kiszolgálást, ezzel elkerülhetővé téve az adott szerver számítógép további túlterhelését.

## *4.1. PROCESSZORIDŐ NAPLÓZÁS*

Hasznos lehet hibakeresésnél és statisztika készítésnél is egyaránt ha rendelkezésre áll, hogy mennyi időt vett igénybe egy-egy kérés kiszolgálása. Az Apache 2.0 beépítve támogatja a kérésenként eltelt idő naplózását. Sajnos ez nem mindig hasznos információ, mivel ez az érték magában foglalja azt az időintervallumot is, amit a gyermek processz nem számításokkal, hanem várakozásokkal töltött el. Nagyon gyakori eset, amikor egy lassú internetkapcsolattal rendelkező felhasználó tölt fel nagyobb adatállományokat a webszerveren keresztül, ilyenkor a kérés kiszolgálási ideje közel sem egyezik meg az igénybe vett processzoridővel. Nagyobb terhelés esetén az is előfordulhat, hogy az operációs rendszer ütemezője érezhetően kevésszer futtatja az adott processzt, ilyenkor is nagy eltérések lesznek a két mért időintervallum között.

Az Apache forráskódjában első lépésben a kérés információit tároló adatszerkezetet (a *request* struktúrát) kell kibővíteni, hogy minden kéréshez tartozhassanak erőforrás használatra vonatkozó mért adatok. Már a kérés beérkezésekor le kell kérdezni a gyermek processz erőforráshasználtságát, mivel egy gyermek processz sok különböző kérést kiszolgál, tehát nem nulláról indul az erőforráshasználata minden egyes kérésnél, illetve maga a webszerver is elhasznál némi processzoridőt, ezért szükség van az erőforrás adatokra a kérés indulásának pillanatában. A lekérdezés kiszolgálása után a naplófileban egy „u:” és egy „s:” előtaggal ellátott decimális szám jelenik meg, ahol az „u” után az felhasználói térben, az „s” után a kernel térben eltöltött processzor idő található. Végezetül meg kell adni azt is, hogy milyen mintára illeszkedjen ez a függvény a naplózási formátumok közül. Ebben az esetben a „%k” mintára fog illeszkedni. A módosítások elvégzése után igénybe vehető a processzoridő naplózás az Apache konfigurációs filejában például az egyik előre definiált *LogFormat* direktíva paraméterének a „%k”-val való bővítésével:

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %k" combined
```

## *4.2. A WEBSZERVER PROCESSZEINEK NEVÉNEK VÁLTOZTATÁSA*

Szintén hibakereséshez és valós idejű megfigyeléshez hasznos ha a gyermek processz nevében láthatóvá válik az éppen kiszolgált kérés metódusa és URI-ja. Például a hosszan futó kérések, vagy egy PHP szkriptben elkövetett programozási hibánál (pl. végtelen ciklus) így azonnal kiszűrhető, hogy melyik PHP szkripttel van probléma. Linux operációs rendszeren egy processznek van többek között egy rövid, 16 karakter hosszúságú parancs neve és egy változó hosszúságú 0. argumentuma. A továbbiakban az utóbbit nevezzük processz névnek. Alaphelyzetben a *ps* parancs a processz nevet jeleníti meg, mivel ez a nagyobb információ tartalmú. A processz név egyszerűen változtatható, hiszen a 0. argumentum felülírható. A következőekben bemutatott programrészletek a *setproctitle* függvény könyvtárat használják, ami Dmitry V. Levin keze munkája [8]. Ennek a függvény könyvtárnak használatával egyetlen függvénnyel elvégezhető a processz név megváltoztatása, az ellenőrzéseket és konverziókat is elvégzi.

Első lépésben a gyermek processz létrejöttekor állítódik át a processz név, itt még nincs kérés tehát egy fix névre állítódik: „*virgin*”, jelezvén azt, hogy ez a gyermek processz még nem szolgált ki kérést. A kérés kiszolgálása közben megjelenik a névben a lekérdezés metódusa, a host kanonikus neve, illetve a lekért file. A kiszolgálás végeztével a processz név visszaáll egy fix névre, de nem arra amire a gyermek processz létrejöttekor („*virgin*”), hogy meg lehessen különböztetni a már kéréseket végrehajtott és a még frissen készült gyermek processzeket, a név „*ready*” lesz.

## *4.3. RENDSZERTERHELÉS FÜGGŐ KISZOLGÁLÁS*

Nagyobb forgalmú webszerver számítógépeken gyakran előfordulhat, hogy összességében sok kérés érkezik, vagy csak az átlagosnál több látogató kér le olyan oldalakat, aminek az előállítása nagy erőforrás igényű szkriptekkel történik. Ilyenkor gyorsan megemelkedhet a rendszerterhelés, így a kérések feldolgozását saját maguk lassítják le, mivel a kérések feldolgozása párhuzamosan történik. Nagy eséllyel okoznak torlódást, illetve akkora terhelést, hogy a szolgáltatás elérhetetlenné váljon. Ilyen esetekben hasznos lehet egy olyan funkciója a webszervernek, hogy a kérés kiszolgálását csak abban az esetben indítja el, ha a rendszer terhelés egy bizonyos határérték alatt van. Egy ilyen megoldásnak a működését és implementálását mutatom be ebben az alfejezetben.

Maga a lényegi része rövid a kiegészítésnek, de a konfigurációs direktívák kezelése és ellenőrzése jó néhány programsort kitesz. A konfigurációban megadhatóak a rendszer terhelésre vonatkozó határértékek 1, 5 és 15 perces átlagra vonatkoztatva. Amint eléri a kiszolgáló gép a megadott terhelési értéket, a webszerver 503-as HTTP hibakódot ad vissza a kliens számára. Az 503-as hibakód jelentése: „Service Unavailable”, tehát a szolgáltatás átmenetileg nem elérhető. Az ide vonatkozó szabvány [9] ilyen esetekre is javasolja ezt a hibakódot, továbbá meghatároz egy opcionális HTTP fejlécet, amiben megadható egy javasolt időtartam, aminek a letelte után a kliens megpróbálkozhat a kérés ismétlésével. Ennek az időtartamnak a meghatározását kétféleképpen láttam célszerűnek, egyik, hogy egyszerűen egy fix számot adhat meg a rendszergazda a konfigurációban, a másik, hogy egy időintervallumot lehet megadni, ami között véletlenszerűen választ egy időtartamot a webszerver és azt küldi vissza a kliensnek. Ezzel a terhelési csúcsokat lehet valamelyest elsimítani, mivel a fix

időtartamnál, a kliensek egyszerre fognak újrapróbálkozni, így újból felszökhet a rendszer terhelése.

# 5. KONFIGURÁCIÓ

Az alábbiakban található néhány példa konfiguráción keresztül kerül bemutatásra, hogy hogyan lehet a webszerverben beállítani a futtató processz felhasználói és csoport azonosítóját, illetve az előző fejezetben taglalt kiegészítések paramétereit.

A *getmyretaddr()* rendszerhívást használó megoldás konfigurációs direktívája a már említett *ServeAsUIDGID*. Globálisan és virtualhost-onként is megadható. Két számot vár paraméterként, az első a kérés kiszolgálásakor használandó felhasználói, a második pedig a csoport azonosító. Az azonosítóknak $2^{32}$ alatt kell lenniük. Ha nincs megadva a direktíva, akkor 0 értékkel inicializálódik, ami azt jelenti, hogy a kiszolgált file tulajdonosának és csoportjának azonosítóit állítja majd be a webszerver. Ez a működési mód közvetlen is megadható ha a direktíva paraméterei 0 értékűek, hasznos lehet akkor ha globálisan be van állítva egy fix azonosító páros, de egy adott virtualhost-ban pl. *mod_userdir*-el van kiszolgálva tartalom.

A 4.1 fejezetben bemutatott kiegészítés használatához az Apache konfigurációs filejában a *LogFormat* direktívát kell módosítani, konkrétan egy „%k”-val kell kiegészíteni, hogy az elfogyasztott processzor idő is naplózásra kerüljön.

A processznév változtató kiegészítésnek nincsenek konfigurációs lehetőségei, ha a patch applikálva lett, akkor a processz név változtatás mindenképpen megtörténik.

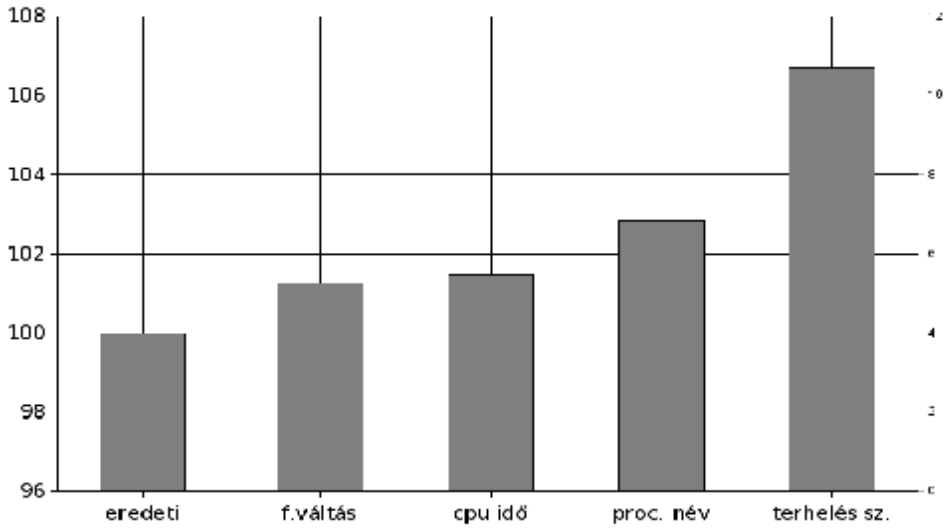
Végül a 4.3 fejezetben ismertetett kiegészítés direktíváinak használatához következik néhány felhasználási utasítás. A *LoadAvgMax* direktívának három lebegő pontos értéket kell megadni, rendre az 1, 5 és 15 perces átlagos rendszerterhelési határértékeket, ahol a webszervernek fel kell függeszteni a kiszolgálást. Ha valamelyik érték 0, akkor az azt jelenti, hogy azt a terhelés értéket nem kell figyelembe venni az ellenőrzésnél. A *LoadAvgRetryAfter* direktíva határozza meg, hogy a határ elérésekor milyen idő értékkel küldje vissza a *Retry-After* header-t a webszerver, az időt másodpercekben lehet megadni. Ha az érték 0, akkor nem kerül *Retry-After* header a válaszba. Harmadik dírektívaként a *LoadAvgRetryAfterRandom* vehető igénybe, ez két egész számot vár paraméternek. Ezen két szám között fog a webszerver egy pszeudo véletlen számot előállítani és ezt küldi vissza a kliensnek várakozási értéknek a *Retry-After* headerben. A *LoadAvgRetryAfter* elsőbbséget élvez a *LoadAvgRetryAfterRandom* direktívával szemben.

# 6. TELJESÍTMÉNYVIZSGÁLAT

Az Apache webszerverhez és a Linux kernelhez készült különböző kiegészítések természetesen befolyással vannak a teljesítményre, mivel a webszerver több kódot futtat így a teljesítményt valamelyest csökkenti a megoldás alkalmazása. Ennek a csökkenésnek a mértékét az Apache webszerverrel együtt szállított *ab* (Apache benchmark) nevű segédprogrammal könnyedén meg lehet mérni. Az *ab* nem tesz mást, mint megadott számú kérést küld a webszervernek és leméri a kérések teljesítésének időtartamát.

Az *ab* által mért értékek közül elegendő a kérések kiszolgálása által igénybe vett időt felhasználni. Célszerű egy-egy futtatást többször is elvégezni a mérési hibák kiküszöbölése érdekében. A méréseket különböző párhuzamossági szintekkel, többszöri futtatással végeztem

el. A 1. ábrán látható a különböző kiegészítések okozta teljesítmény változások összehasonlítása. Ezen eredmények alapján lehet mérlegelni, előre tervezni az üzemeltetőknek az igényelt plusz teljesítménnyel kapcsolatban. Általában a biztonságot fokozó módosítások teljesítmény romlással és/vagy funkcionalitásbeli korlátozásokkal járnak. Az előbbi állítás igaz ebben az esetben is, viszont az a helyzet áll fenn, hogy a funkcionalitás kibővül, éppen a biztonságnak köszönhetően, illetve a teljesítmény romlás minimális.




*1. ábra: a teszteredmények grafikonnal szemléltetve*

# ÖSSZEFOGLALÁS

A bemutatott módszerek alkalmassá teszik az Apache HTTPD webszervert arra, hogy a felhasználók szkriptjeit értelmező és végrehajtó beágyazott interpreter (pl. *mod_php*, *mod_python*, *mod_perl*, *mod_ruby*) mindenkinek a saját azonosítójával, így az adott felhasználó jogosultságaival tudja futtatni a webszerver. Megszüntetve ezzel a közös felhasználói azonosítóval futtatott szkriptek közös jogosultságokkal való rendelkezésből fakadó problémákat. Ezen kívül több új lehetőség merül fel abból adódóan, hogy minden felhasználónak a saját azonosítójával futnak a szkriptjei, így például: részletesen korlátozhatóvá és mérhetővé válik az egyes felhasználók által a szkriptekből kezdeményezett hálózati forgalom; az adatbázis kezelőkhöz való hozzáférés OS szintű azonosítással, jelszó nélkül; a PHP-ben található *mail()* függvény által küldött e-mail-ek *Return-path* fejlécének automatikus és helyes beállítása; a webszerveren keresztül feltöltött fileok kezelésének egyszerűsödése; és más egyebek.

Néhány felhasználós rendszeren még viszonylag könnyen ellenőrizhetőek a felhasználók által a webszerveren elhelyezett szkriptek, de egy olyan rendszeren, amihez több százan férnek hozzá, gyakorlatilag lehetetlen. Üzemi környezetben pedig nem ritka a több ezres, vagy akár tízezres nagyságrendű felhasználó számmal rendelkező rendszer.

Továbbfejlesztési lehetőség pl. a *getmyretaddr()*-féle megoldásban a felhasználó és csoport váltást integrálni a *getmyretaddr()* rendszerhívásba teljesítmény javítási célból, illetve az említett biztonsági kockázatok megszüntetése és további tesztelése. Továbbá más webszerverekhez, illetve más operációs rendszerekre való portolás, mint pl. BSD és más UNIX rendszerek. A más operációs rendszerekre való átültetéshez a futásidőben történő

felhasználó váltás más lehetőségeit kell megvizsgálni, mivel nem mindenhol támogatott a capabilites rendszer, illetve a zárt forráskód miatt, nem lehet egyszerűen új rendszerhívást implementálni, viszont a különálló kiegészítések könnyebben átvihetőek más rendszerekre.

# HIVATKOZÁSOK


***[1] April 2008 Web Server Survey***

http://news.netcraft.com/archives/2008/04/14/april_2008_web_server_survey.html

***[2] Request for Comments #3875***

http://www.ietf.org/rfc/rfc3875

***[3] Muxmpm/metuxmpm for Apache 2.0.48***

http://www.sannes.org/metuxmpm/

***[4] The Apache 2 ITK MPM***

http://mpm-itk.sesse.net

***[5] Portable Operating System Interface, IEEE Standard 1003.1, 2004 Edition***

http://www.unix.org/version3/ieee_std.html

***[6] Portable Operating System Interface, IEEE Draft P1003.1e***

http://wt.xpilot.org/publications/posix.1e/download/Posix_1003.1e-990310.pdf.bz2

***[7] Linux kernel capabilities FAQ***

http://ftp.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.4/capfaq-0.2.txt

***[8] Dmitry V. Levin setproctitle függvénykönyvtára***

http://www.sisyphus.ru/srpm/setproctitle

***[9] Request for Comments #2616***

http://www.ietf.org/rfc/rfc2616