Mérési útmutató az „Access Control (ACC)” című méréshez

2021. április 11.

A mérést kidolgozta:
Lestyán Szilvia
Futóné Papp Dorottya

BME, CrySyS Adat- és Rendszerbiztonság Laboratórium

Tartalomjegyzék

1. Bevezetés

A mérés célja különböző rendszereken megvalósított hozzáférésvédelmi megoldások vizsgálata. Ennek kapcsán a hallgatók megismerkednek a hagyományos UNIX hozzáférés-védelmi mechanizmusaival: `suid` bit, `sgid` bit. A modern hozzáférés-védelmi mechanizmusok közül a mérésen szerepelnek a hozzáférés-védelmi listák és a SELinux alapjai.

2. Hagyományos UNIX hozzáférés-védelem

Többfelhasználós rendszerekben alapvető fontosságú, hogy az egyes felhasználók saját adatait a lehetőség szerint megvédjük a szándékos vagy szándékolatlan rongálástól, nem is beszélve a magántitok védelméről és más kérdésekről. E célok biztosítására a UNIX egy tulajdonosi kategóriák és tevékenységek szerinti engedélyezési rendszert használ, s ennek alapján dönt az állományokhoz való hozzáférés engedélyezéséről, illetve megtagadásáról.

Egy UNIX felhasználót a felhasználói azonosítója (uid) azonosít a rendszerben, ezenfelül minden felhasználó valamilyen csoportba is tartozik, így van egy csoportazonosítója (gid) is. Amikor valamilyen módon hozzá szeretnénk férni egy fájlhoz vagy mappahoz, a rendszer aszerint sorol be minket, hogy milyen tulajdonosi viszonyban vagyunk az adott fájllal: lehetünk tulajdonosa (azonos uid), lehetünk csoporttagok (azonos gid), vagy "egyéb". A tulajdonost a `chown`, a csoportot pedig a `chgrp` paranccsal konfigurálhatjuk.

A fájllal kapcsolatos tevékenységek szempontjából három fő csoport van: az állomány olvasása, írása, illetve végrehajtása (mappa esetében keresése). Egy-egy fájlművelet elvégzése előtt a UNIX ellenőrzi, hogy melyik tulajdonosi kategóriába esünk, s utána azt, hogy ebben a kategóriában engedélyezett-e vagy sem a végrehajtani kívánt művelet.

A hozzáférési jogosultságok az `ls -la` lista alapján szemléltethetőek a legkönnyebben:

```
total 364
-rw-r--r-- 1 demo guest 18 Aug 23 20:42 file1
drwxrwxrwx 3 demo guest 1024 Aug 23 20:59 newdir
-rw-rw-rw- 1 demo guest 18 Aug 23 20:42 newfile
lrwxrwxrwx 1 demo guest 10 Mar  9 20:32 sample -> text
-rw-rw-rw- 2 demo guest 18 Aug 23 20:59 text
```

A lista baloldali oszlopa tartalmazza a hozzáférési jogosultságokat. Az oszlop legszélső mezője a fájltípust kódolja:

- közönséges fájl (`-`)
- mappa (`d`)
- speciális pipe (`p`)

- szimbolikus link (`l`)
- karakteres készülékmeghajtó (device driver, `c`)
- blokkos készülékmeghajtó (`b`)

A fájltípus után következő kilenc karakter kódolja a jogosultsági biteket háromszor hármas bontásban: olvasás (`r`), írás (`w`) és végrehajtás (`x`). Ha egy művelet engedélyezett, a neki megfelelő betű látszik a listán, ha nem, a `-` karakter jelzi a tiltást. Előfordulhat, hogy más karakterek vannak ebben a kilenc karakterben, néhányat a továbbiakban ismertetünk. Az első hármas csoport a tulajdonos, a második a csoport, végül a harmadik a többiek jogosultságait mutatja. Konkrét példát véve, a fenti listán szereplő `file1` egy közönséges fájl, amit a tulajdonosa írhat és olvashat, de nem hajthat végre (`r--`), ám a csoporttagok és a többiek csak olvashatják.

A jogosultsági biteknek nem csak karaktereket, hanem számokat is megfeleltethetünk. A jogosultsági bitek egymás mellett megfelelnek egy kettes számrendszerbeli számnak, amit egy oktális számmal kódolhatunk. A kettes számrendszerbeli reprezentáció leírásánál az `r` bit áll a legnagyobb helyiértékű helyen, míg az `x` bit a legkisebb helyiértékű. Adott helyiértéken akkor szerepel 1-es, ha az adott bit be van billetve engedélyezésbe. Ha nincs bebillentve, akkor az adott helyiértéken 0 szerepel. Például, az `rw-` jogosultság megfelel a kettes számrendszerbeli 110-nak, amit oktálisan 6-ként kódolunk.

Attól függően, hogy közönséges fájlról vagy mappáról van szó, az olvasás-írás-végrehajtás fogalma változik. Közönséges fájloknál az olvasás a fájl megnyitását és tartalmának kiírását jelenti. Az írás a tartalom módosítása, a végrehajtás pedig a fájl futtatása végrehajtható állományként. Mappáknál az olvasási jogosultság a mappa tartalmának listázására vonatkozik. Az írási jogosultsággal rendelkezők módosíthatják a mappa tartalmát: fájlokat hozhatnak létre, törölhetnek, átnevezhetnek.

Ezzel szemben egy mappa olvasása azt jelenti, hogy tudhatunk a létezéséről, listázni tudjuk magát a mappát és a benne található fájlok neveit (pl. `ls` paranccsal). Egy mappa akkor írható, ha bejegyzéseket tudunk létrehozni, módosítani, vagy törölni benne, illetve törölhetjük/létrehozhatjuk magát a mappát (kapcsolódó parancsok: `mkdir`, `rmdir`, `mv`, `cp`, `rm`, `ln`). Mappák esetén a végrehajtási jogosultság megfelel a belépés (traverse) engedélynek, enélkül nem érhetjük el a mappa tartalmát, még akkor sem, ha azt listázni tudjuk. Például tegyük fel, hogy van egy `prog` nevű végrehajtható állományunk, ami egy olyan mappában szerepel, amelyikre engedélyezett az olvasás, de nem engedélyezett az írás és a végrehajtás (`r--`). Ekkor magát a mappát látjuk ugyan a kilistázáskor, és azt is megtudhatjuk, hogy tartalmazza a `prog` nevű állományt, de az állomány tartalmát és metaadatait (méret, jogosultságok, utolsó hozzáférés ideje, stb) nem érhetjük el. Ha ugyanez a mappa nem olvasható, de kereshető (`--x`), akkor nem fogjuk tudni kilistázni

az benne található `prog` végrehajtható állományt, de ha kiadjuk a végrehajtási parancsot, akkor az állomány végre fog hajtódni, hiszen be tudtunk lépni a mappába, hogy elérjük.
További olvasmányok és feladatok a következő linken találhatóak:
`https://www.szabilinux.hu/ufi/3_4.html`

3. A `suid` és `sgid` bitek, valamint a sticky bit

A UNIX típusú rendszerekben a programok azokkal a felhasználói jogokkal futnak, amivel az őket elindító felhasználó rendelkezik. Bizonyos esetekben azonban ez nem szerencsés. Talán a legegyszerűbb példa a felhasználói azonosító megváltoztatására az accounthoz tartozó jelszó megváltoztatása. Egy különleges privilégiumokkal nem rendelkező felhasználó általában nem írhatja közvetlenül a rendszer jelszófájlját, hiszen akkor bármikor korlátlan jogokhoz juthatna. Mégis, szükség van arra, hogy a saját jelszavát megváltoztassa, amihez viszont írnia kell a jelszófájlt. Ezt az ellentmondást oldják fel a suid és az sgid bitekkel.

A suid (Set User Identification) a felhasználói azonosító megváltoztatása, ezáltal a privilégiumok megváltoztatása.A suid bitet a jogosultságok között a tulajdonosra vonatkozó végrehajtási bit helyén láthatjuk, `s` jelzéssel:

```
-rwsr-xr-x 1 root root 28896 Sep 7 13:40 /usr/bin/passwd
```

Az `s` a set-uid (set user-id) jog engedélyezését jelzi, ha a tulajdonosi kategóriánál szerepel. Ha e jog engedélyezett, akkor akárki is futtatja a szóbanforgó programot, a futtatás idejére olyan jogokkal fog rendelkezni, mint a program tulajdonosa. A példában szereplő `passwd` program esetében ez a rendszergazdát jelenti.

A UNIX újabb verziói lehetőséget biztosítanak arra is, hogy egy felhasználó több csoportba is tartozhasson (másodlagos csoportok). Konfigurálásukhoz rendszergazdai jogosultságra van szükség. A másodlagos csoportok a csoportmunkák támogatásánál előnyösek. Így lehetőség nyílik arra, hogy valaki elsődlegesen a saját felhasználójának csoportjába tartozzék, de minden olyan fájlhoz hozzáférjen a másodlagos csoporton keresztül, amely fájlok olyan projekthez tartoznak, amelyben ő résztvesz. Ezt úgy érjük el, hogy a projekt fájlainak csoport tulajdonosa az a csoport, amit a csapattagok másodlagos csoportként használnak.

A `sgid` bit (Set Group Identification) a csoportazonosító megváltoztatására szolgál, beállítása esetén a program annak a csoportnak a jogaival fog futni, amelyik csoportnak a fájl a birtokában van. A sgid illetve a set-gid (set group-id) jogot a tulajdonos csoport jogosultsági bitjei között láthatjuk a végrehajtás helyén `s` karakterrel jelölve. A `sgid` bitet mappák esetén is be lehet kapcsolni. Ennek eredményeként, ha ebben a könyvtárban bárki létrehoz egy fájlt (ehhez a többi jognak rendben kell lennie), akkor a fájl

csoporttulajdonosa nem az a csoport lesz, amelyikbe a felhasználó tartozik, hanem az, akinek a könyvtár a birtokában van.

A `sticky bit` bekapcsolása fájlok esetén azt jelzi az operációs rendszernek, hogy a fájlt tartsa a memóriában a végrehajtás után is. Ennek a tulajdonságnak akkor van értelme, ha azt szeretnénk, hogy egy program minél gyorsabban induljon el, ne kelljen várni a betöltődésére. A sticky bitet is be lehet kapcsolni mappák esetén is. Az ilyen bittel ellátott mappákba bárki írhat fájlokat (feltéve, hogy erre jogosultsága van), de mindenki csak a sajátját törölheti. Ezt a lehetőséget azért tervezték, hogy az olyan, mindenki által írható könyvtárakban, mint például a `/tmp`, a felhasználók ne tudják a másik felhasználó által írt fájlokat módosítani, letörölni.

4. Hozzáférés-védelmi listák

A hagyományos UNIX hozzáférés-védelem egyik hiányossága, hogy a hozzáférést csak korlátozott részletességgel konfigurálhatjuk be. A hagyományos keretek között, ha egy fájlhoz másik felhasználónak is engedélyt akarunk adni, akkor hozzá kell adnunk őt a fájlt birtokló csoporthoz. Ekkor azonban minden olyan fájlhoz hozzáférést kapott a kérdéses felhasználó, amit ez a csoport birtokol, pedig nem ezt szerettük volna elérni. Az Access Control List (ACL) erre a problémára nyújt megoldást úgy, hogy a hagyományos hozzáférési listát kiegészíthetjük más felhasználókra és csoportokra vonatkozó engedélyekkel, valamint új maszkkal. Az ACL használatával a fent említett probléma kiküszöbölhető azzal, hogy a fájlhoz tartozó ACL-be felvesszük a kérdéses felhasználót olvasási jogosultsággal. Ekkor a rendszer a felhasználót nem a másoknak adott jogosultságok alapján bírálja el, hanem azok alapján a jogosultságok alapján, amit az ACL rá vonatkozóan ír.

Amennyiben létezik ACL egy objektumra, a listázáskor a hagyományos jogosultságok mellett egy `+`-t fogunk látni. A mappákra és fájlra definiált ACL-eket a `getfacl` paranccsal nézhetjük meg:

```
$ getfacl teamfolder
# file: teamfolder
# owner: user
# group: team
user::rwx
user:angela:rw-
group::rwx
other::r-x
default:user:rw-
default:group:rw-
default:other:r--
```

A kimenetben láthatjuk az objektum nevét, amire az ACL vonatkozik, valamint az objektum tulajdonos felhasználóját és csoportját. Az ACL tartal-

mazza, hogy a tulajdonos felhasználónak, a csoportnak és másoknak milyen jogosultságai vannak.

Amennyiben egy ACL csak a tulajdonos felhasználóra, csoportra és másokra tartalmaz bejegyzéseket, *minimális* ACL-nek nevezzük. Amennyiben `default` és/vagy néven nevezett felhasználókra/csoportokra vonatkozóan is tartalmaz bejegyzéseket, *kiterjesztett* ACL-nek nevezzük. A `default` bejegyzések rögzítik, hogy az újonnan létrehozott fájlok milyen jogosultságokat örököljenek a fájlrendszer-hierarchiában felettük álló objektumtól.

Az ACL-ek segítségével a fájlok eléréséhez maszkot is definiálhatunk. Hagyományos esetben a *maszk* a fájlok és mappák létrehozásakor kap szerepet, a default értéket az `umask` paranccsal nézhetjük meg. Ez egy oktális szám, amelyet a default engedélyek oktális értékéből von le a rendszer. Létrehozáskor a fájl default engedélyei a 666 értéknek felelnek meg (mappánál 777), ebből vonódik le a default maszk érték (általában 022). Végeredményként fájlok esetén általában 644-et kapunk, vagyis `rw-r--r--` engedélyek lesznek a fájlon. Ez a mechanizmus általában jó hozzáférés védelmet eredményet, de minden felhasználó csak egy maszk értékkel rendelkezik, vagyis minden általa létrehozott fájl ugyanazokkal a jogosultságokkal fog létrejönni. Ezek a jogosultságok azonban nem minden esetben vannak összhangban az adott mappában érvényes fájl jogosultsági irányelvvel, elképzelhető, hogy olyan jogosultságokat is tartalmaznak, amit az irányelv tilt. Ilyen esetekben az ACL segítségével megadhatunk egy lokálisan értelmezett maszk értéket, ami leírja, hogy mik a maximálisan engedélyezhető jogosultságok. Ha például az ACL maszk szerint maximum `rw-` jogosultság engedélyezett a fájlokon, akkor hiába látjuk egy végrehajtható állománynál az `r-x` jogosultságot, a maszk miatt nem fogjuk tudni futtatni.

5. AppArmor

Az AppArmor egy Linux kernel biztonsági modul, amely lehetővé teszi a rendszeradminisztrátoroknak, hogy a programok hatáskörét limitálják adott programra szabott profilokkal. Engedélyezhető például a hálózati hozzáférés, raw socket access, adott elérési útvonal alatt található fileok írása, olvasása és végrehajtása. Az AppArmor *mandatory access controlt* (MAC) implementál. Ez azt jelenti, hogy képes korlátozni azt is, hogy az erőforrás tulajdonosa hogyan férhet hozzá az erőforráshoz. Ezt a hagyományos hozzáférés védelmi mechanizmusok nem tudták megtenni, mivel azok a *discretionary access control* (DAC) implementációi. DAC esetén sikeres hitelesítés után a tulajdonos döntheti el az adott erőforráshoz tartozó hozzáférési jogosultságokat, pl. bármikor átadhatja az erőforrást másnak, vagy információt oszthat meg az erőforrásról.

A profilok létrehozása mellett egy tanulási folyamatot is lehetove tesz az AppArmor, ahol a profilokat sértő folyamatok csak logolva vannak. Majd

ezt a logot felhasználva készíthetünk új profilokat.

Az AppArmor A SELinux egyik (legjobb) alternatívája. Ellentétben a SELinux-szal az AppArmort sokkal könnyebb használni és konfigurálni. A SELinux label-eket tesz az egyes fileokra, ezzel szemben az AppArmor elérési útvonalakkal dolgozik. Továbbá az AppArmor működéséhez a filerendszeren kevesebb módosítást kell végrehajtani, mint a SELinuxhoz, mivel a SELinux esetén a filerendszernek támogatnia kell a biztonsági cimkéket (label), ezért nem tud hozzáférési védelmet nyújtani NFS-en keresztül mountolt fileokhoz. Az AppArmor filerendszer független.

5.1. Monitorozás és management

Az **aa-status** paranccsal ellenőrizhetjük, hogy mely profilok vannak betöltve és azok milyen státuszban vannak.

5.1.1. Complain és Unconfined módok

Amikor egy profil complain módban van, akkor az AppArmor majdnem minden folyamatot engedélyez neki, viszont logolja azokat. Ezeket profilozához fel tudjuk használni a későbbiekben. Például nem tudjuk, hogy egy alkalmazásnak milyen hozzáférésekre van szüksége, akkor ezeket kikereshetjük a logokbol. Az unconfined mód mindent engedélyez az alkalmazásnak, viszont nem is logolja azt. Ez általában akkor történik meg, ha egy profilt az alkalmazás elindítása után töltöttük be. Fontos megjegyezni, hogy csak olyan alkalmazásokat sorol fel az aa-status parancs, amelyekre létre lett hozva profil, a többit nem.

5.1.2. Profilok létrehozása

Két fő eszközünk van profilok létrehozásához az **aa-genprof** és az **aa-logprof**. Az elsőt arra használjuk, hogy monitorozzunk egy alkalmazást, majd az alapján egy profilt hozzunk létre hozzá, azaz az AppArmor futtatás közben tanulja meg, hogy mire lehet szüksége a programnak, ezeket nem magától menti el, hanem a felhasználónak felsorolja a találtakat és annak kell végül engedélyeznie vagy tiltania egy egy működési folyamatot. Az aa-logprof-ot akkor használjuk, amikor már egy létező profilt kell frissítenünk, például, ha annak működése megváltozott (például verziófrissítés).

Az AppArmor első promptjára egy példa:

```
 [(S)can system log for AppArmor events] / (F)inish
Reading log entries from /var/log/audit/audit.log.
Updating AppArmor profiles in /etc/apparmor.d.

Profile:  /home/user/bin/pelda.sh
Execute:  /usr/bin/touch
```

```
Severity: 3

(I)nherit / (C)hild / (N)amed / (X) ix On / (D)eny / Abo(r)t / (F)inish
```

A prompt rámutat, hogy az adott alkalmazás, amit éppen profilozunk mihez szeretne hozzáférni. Az AppArmor arról is figyelmeztethet minket, hogy mennyire tartja súlyosnak a hozzáférést. Ezt egy 1-10-es skálán adja meg, ahol az 1 a legalacsonyabb veszélyforrást és 10 a legsúlyosabbat jelöli, de vigyázzunk, mert nem minden esetben ad a valóságnak megfelelő értéket!

A fenti opciók jelentései a következők:

- **Inherit:** "Ix"-szel jelölt szabályt hoz létre a profilon belül, amely azt jelzi, hogy a végrehajtott binary a szülő profilból örökli a jogosultságot.
- **Child:** "Cx"-szel jelölt szabály, amely egy rész-profil létrehozását teszi lehetővé a szülő profilon belül.
- **Deny:** Egy olyan szabályt hoz létre, amely "deny"-al kezdődik a profilban, ezáltal a szülő folyamatot eltíltja a forrás használatától.
- **Abort:** Kilép az AppArmor-ból mentés nélkül.
- **Finish:** Kilép, de mentéssel.

Egy elérési útvonalhoz a következő lehetőségeket adhatjuk meg:

- **Allow:** Engedélyezi az útvonalat a korábban igényelt jogosultságokkal.
- **Deny:** Megtagadja az útvonalat a korábban igényelt tulajdonságokkal.
- **Ignore:** Átugorja ezt a pontot, és a következő logprof futtatásakor újra megkérdezi.

További olvasmányok és feladatok a következő linkeken találhatóak:
`https://medium.com/information-and-technology/so-what-is-apparmor-64d7ae211ed`
`https://www.youtube.com/watch?v=Uq1d60TLebE`

6. Security Enhanced Linux - kiegeszito anyag

A Security Enhanced Linux (SELinux) egy biztonsági kiegészítés a Linux kernelhez, mely nagyobb felügyeletet biztosít felhasználók és rendszeradminisztrátorok számára egyaránt azáltal, hogy *mandatory access controlt* (MAC) implementál. Ez azt jelenti, hogy képes korlátozni azt is, hogy az erőforrás tulajdonosa hogyan férhet hozzá az erőforráshoz. Ezt a hagyományos hozzáférés védelmi mechanizmusok nem tudták megtenni, mivel azok a *discretionary access control* (DAC) implementációi. DAC esetén sikeres

hitelesítés után a tulajdonos döntheti el az adott erőforráshoz tartozó hozzáférési jogosultságokat, pl. bármikor átadhatja az erőforrást másnak, vagy információt oszthat meg az erőforrásról.

A SELinux három működési móddal rendelkezik:

- *Enforcing:* a SELinux megakadályoz minden tevékenységet, kivéve azokat, amelyek explicit engedélyezünk; a tevékenységeket naplózza
- *Permissive:* a SELinux engedélyezve van és aktívan naplóz, de nem lép közbe, ha tiltania kéne
- *Disabled:* a SELinux nincs engedélyezve a gépen, sem logolás, sem tiltás nem történik

A hozzáférés-védelem érvényre jutattásához minden fájl, socket, felhasználó, folyamat, stb. rendelkezik egy címkével, amit SELinux *kontextusnak* nevezünk. A kontextus formátuma `user:role:type:level(optional)`. A fájlok és mappák kontextusait a fájlrendszeren tárolja a számítógép, a folyamatok, portok, stb. kontextusát pedig a kernel menedzseli. A kontextusok menedzselését *labelingnek* vagy címkézésnek is hívjuk. A kontextusok megtekintéséhez a `-Z` kapcsolót használhatjuk a különböző listázó programoknál, pl. `ls -Z`.

A kontextus `type` része egyfajta csoportosítást biztosít a különböző fájlok és objektumok között, folyamat esetében *domainnek* nevezzük. A *type enforcement* mechanizmus felelős annak menedzseléséért, hogy mely kontextussal rendelkező objektumok között megengedett az interakció. Például, az Apache2 webszerver, melyek domainje `httpd_t`, használhatja a weboldalak fájljait, amelyek típusa `httpd_sys_content_t`, de nem érheti el a `/etc/shadow` fájlt, melynek típusa `shadow_t`. Természetesen előfordulhat, hogy egy folyamatnak más típussal rendelkező objektumhoz kell hozzáférnie. Ezt a kontextusok közötti átmenetet *domain transionnek* nevezzük és külön engedélyezni kell.

A kontextus `user` része adja meg a SELinux felhasználót, amely *nem egyezik meg a hagyományos Linux felhasználókkal.* A SELinux felhasználó az a személy, aki a hozzáférés védelmi szabályok szerint bizonyos szerepeket tölthet be. A szabályok minden Linux és SELinux felhasználó között 1:1 megfeleltetést biztosítanak, így a Linux felhasználók megöröklik a SELinux felhasználók korlátozásait.

Megjegyzés: Egyes Linux operációs rendszerek esetében a fejlesztők nem tartják karban a SELinux policy-ket, ilyen például a Debian és az Ubuntu is. Ha a való életben SELinuxot szeretnénk használni, érdemes egy olyan Linux disztribúciót használni, ahol a letölthető policy-ket a fejlesztő csapat aktívan karban tartja (pl. Red Hat, CentOS).

7. Feladatok

A mérés során egy webszervert fogunk felkonfigurálni, ami egy megosztott mappában tárolt weboldalt hosztol. A megosztott mappában egy képzeletbeli vállalat dolgozói tárolnak dokumentációkat és egy wiki oldalt. A vállalatnak szakmai gyakorlatos dolgozói is vannak, akik csak időlegesen kapnak hozzáférést a rendszer egy-egy erőforrásához.

7.1. Megosztott mappa konfigurálása

Kapcsolódjon a virtuális géphez, a bejelentkezéshez használja a `user`/`password` párost!

7.1.1. Felhasznalok kezelese

Vegyünk fel három új felhasználót *alice*, *joe* es *eric* usernevekkel és tetszőleges jelszavakkal. Eric és Alice kerüljenek be a *contentdevs*, joe pedig az *interns* csoportokba. (Azaz a masodlagos csoportjaik legyenek ezek.) Milyen fileokban történt változás, és milyen változás? Milyen felhasználói fiókok vannak a gépen és melyik csoportokhoz tartoznak a különböző felhasználók? Csak azokat a felhasználói fiókokat sorolja fel, amelyekhez parancssoron be lehet jelentkezni!

Melyik felhasználó tud rendszergazdai jogosultságot igénylő parancsokat futtatni?

7.1.2. Mennyi? Harminc...

Szeretnénk megtudni, hány felhasználó létezik a rendszerben (beleértve a szolgáltatási és rendszerfiókokat is). A későbbiekben minden reggel szeretnénk a felhasználók számát lekérni és feljegyezni, így a feladat megoldásához ne a *majd kézzel megszámolom* módszert alkalmazza! (Segítség: a feladat megoldásához nem szükséges programot vagy scriptet írni.)

Hozzon létre egy új fájlt *prog1.cpp* néven, majd másolja bele az alábbi sorokat!

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    setuid(0);
    system("id");
    return 0;
}
```

Használja a g++ fordítót az imént létrehozott forráskód lefordítására. Az így keletkező *a.out* állományt tegye futtathatóvá. Futtassa le rootként és egy tetszőleges nem-root felhasználóval is. Amennyiben nem root a tulajdonosa az állománynak, úgy mostantól legyen a root a tulajdonos. Állítsa be, hogy mindig a tulajdonos nevében fusson az alkalmazás, függetlenül attól, hogy ki indítja el. Futtassa le újból rootként és egy nem-root felhasználóként is. Mi a különbség a két-két futtatás között?

Milyen háttérszínnel jeleníti meg az *ls -la* parancs az állományt?

7.1.3. Megosztott mappa létrehozása

Váljon `root`-tá és hozza létre a `/data/docs` mappát! Ezt a mappát fogják használni a felhasználók.

A mappa tulajdonos felhasználója legyen `eric`, tulajdonos csoportja pedig a `contentdevs` csoport!

7.1.4. Mappa megosztása csoporton belül

Váltson át az `eric` felhasználói fiókra és hozzon létre egy üres szöveges fájlt a `/data/docs` mappában! Milyen jogosultságokkal jött létre a fájl? Ki(k) a fájl tulajdonosai?

Az `alice` nevű felhasználó szintén tagja a `contentdevs` csoportnak. Milyen jogosultsága van a létrejött fájlhoz? Válaszát indokolja és parancssorban ellenőrizze!

Tud-e új fájlt létrehozni `alice` a `/data/docs` mappában? Válaszát indokolja és ellenőrizze a parancssoron!

Adjon írási jogosultságot a tulajdonos csoportnak a `/data/docs` mappán!

Tudja-e `alice` módosítani az `eric` által látrehozott fájlt? Válaszát indokolja és ellenőrizze a parancssoron!

Hogy orvosolná a problémát? Több megoldást is vázoljon!

Billentse be a `sgid` jogosultsági bitet a `/data/docs` mappán és próbálkozzon újra! Mi változott? Tudná-e `eric` módosítani az `alice` által létrehozott fájlt, vagyis valóban megosztott-e a mappa a `contentdevs` csoport tagjai között?

7.1.5. Hozzáférés-védelmi lista konfigurálása

A megosztott tartalmakért felelős csoporthoz beosztanak egy szakmai gyakorlatos hallgató (`joe`). A hallgató feladata segíteni a tartalom-fejlesztésben és a wiki oldal elkészítésében. A rendszergazda azonban nem szeretné `joe`-t hozzáadni a `contentdevs` csoporthoz. Miért?

Ellenőrizze, hogy a rendszeren telepítve van-e az `acl` csomag!

Konfigurálja a `/data/docs` mappa hozzáférés védelmi listáját úgy, hogy annak mostani és jövőbeli tartalmához is hozzáférjen `joe`! Megoldását ellenőrizze! Hogyan néz ki a módosított ACL? Az ACL felvétele után milyen új

(eddig nem látott) karakter jelenik meg az *ls -la* kimenetében a */data/docs* könyvtárra vonatkozóan?

7.2. AppArmor

Ellenőrizze, hogy telepítve van az AppArmor, majd hogy valóban fut-e a service. Hány profil van betöltve, és hány van enforcing módban?
Hozza létre a /data/files mappát és az alábbi teszt.sh szkriptet,amit a /data-ban helyezzen el. A feladat célja, hogy bemutassa az AppArmor hogyan akadályozza meg a szkriptet, hogy az más mappákhoz is hozzáférjen.

```
#!/bin/bash
echo "AppArmor teszt"
touch files/myfile.txt
echo "file kész"
rm files/myfile.txt
echo "file törölve"
```

Adjon a szkriptnek futtatási jogot, és futtassa le. Indítsa el a profilozást a szkripten, majd nyisson egy új shellt (új terminalban) és újra futtassa a teszt.sh-t. Fontos, hogy a szkript monitorozás közben fusson! Miután lefutott a szkript, nyomjon egy *s* billentyűt az aa-genprof terminálban. Milyen jogosultságot kell adni a szkriptnek?

A következő prompt-ban milyen interface-hez adhatunk jogosultságot? Valamint ezután miről tájékoztat még minket az AppArmor?

Mentse el a változtatásokat és fejezze be a profilozást, végül pedig futtassa újra a szkriptet, hogy lássuk, hogy az AppArmor valóban nem avatkozik bele a helyes működésébe. Bizonyosodjon meg róla, hogy a teszt.sh szkript valóban enforce módban van.

Most pedig nyissa meg újra a szkriptet és írja át a file-ok elérési útvonalát, úgy, hogy az ne a files mappába, hanem egyenesen a /data-ba írjon. Futtassa a szkriptet, mi történt?

Indítsa el az aa-logprof parancsot a szkripten és engedélyezze az új elérési útvonalat a fileokhoz amíg a szkript újra futtathatová nem válik.

A következő feladathoz írja meg a következő bash scriptet a /data mappában:

```
#!/bin/bash
echo "Hello World" > /tmp/hello.txt
cat /tmp/hello.txt
rm /tmp/hello.txt
```

Készítsen egy AppArmor profilt a szkripthez! Miket engedélyezett? Nyissa meg a generált profilt, hogy néz ki?

A szkriptben sajnos van egy sérülékenység, mi az?