Korte uitleg: Wat betekent "root worden"?

Hierover is veel onbegrip zoals blijkt uit verschillende draden op de forums (ook de Engelse). Zo praat men over "root worden of sudo gebruiken" alsof dat alternatieven zijn. Je gebruikt (eventueel) `sudo` om `root` te worden. Zoals wel vaker gebruiken mensen "root worden" terwijl ze geen idee hebben wat het is.

Zoals we van *Korte uitleg: De Kernel* en *Korte uitleg: Processen* weten hebben alle prcocessen een bijbehorende gebruikersidentificatie/userid (`uid`) en groepsindentificatie/groepsid (`guid`). We praten dan over de "eigenaar" (owner) van een proces. Als het `uid 0` is mag die eigenaar veel meer van de Kernel dan alle andere `uid`s. Alle `uid`s zijn geconfigureerd in `/etc/passwd` en hebben daar ook een naam, de `username`. De `username` van de user met `uid 0` is `root`.

De keuze van die naam is achteraf misschien niet zo handig, maar wijzigen levert waarschijnlijk meer ellende op dan zo houden. Het is tenslotte al meer dan 40 jaar zo.

Doe eens:

```
henk@boven:~> grep -e root -e $(whoami) /etc/passwd
root:x:0:0:root:/root:/bin/bash
henk:x:500:500:Henk van Velden:/home/henk:/bin/bash
henk@boven:~>
```

Dan zie je twee regels uit de `/etc/passwd` file. Een regel bevat `:` gescheiden velden. Het eerste veld is de `username`. Het derde veld is de `uid`. Het vierde veld is de primaire `guid`.

Dus een proces heeft als eigenaar `uid 0` of niet en het draait dus "als root" of niet. Een kind proces erft zijn eigenaar. Dat is dus een redelijk gesloten situatie. Maar aangezien proces `PID 1` als eigenaar `root` heeft en dus al zijn kinderen en kleinkinderen, hoe kan het dan dat er processen zijn die niet `root` als eigenaar hebben? De Kernel kan op verzoek een kindproces starten met een andere eigenaar. Alleen een `root` proces kan dat vragen. Nu weten we dus hoe een `root` proces als `login` een kindproces als `bash` kan opstarten voor een gewone gebruiker. Maar omgekeerd?

We moeten dus het probleem oplossen hoe een gewone gebruiker een proces kan laten starten met als eigenaar `root`. Eigenlijk willen we een algmenere oplossing: hoe kan een gebruiker een proces laten opstarten als een andere gebruiker. De oplossing zit in één van de bitjes uit de groep van toegangsbitjes (permission bits), die aan een bestand zijn verbonden. Dat zijn de bitjes die aangeven of een bestand mag worden gelezen, beschreven, of uitgevoerd door de eigenaar, de groep of iedereen (zie *Korte uitleg: Wie mag wat met welk bestand*). Dat is het `suid` (set `uid`) bit. Dit bit geeft aan dat de Kernel het programma in een process moet opstarten met als eigenaar degene die eigenaar is van het bestand. Een voorbeeld:

```
henk@boven:~> ls -l $(which su)
-rwsr-xr-x 1 root root 44256 23 jan 14:04 /usr/bin/su
henk@boven:~>
```

We zien dat op de plaats van het eigenaars x-bit een `s` staat. Dit betekent dat het x-bit is gezet en dat het bestand `suid` is. Dat wil zeggen dat voor iedereen die het bestand mag uitvoeren (en dat is iedereen want de x-bits voor groep en anderen zijn gezet), het proces de eigenaar van het bestand, dat is `root`, als eigenaar krijgt. Plotseling kan iedereen een `root` proces van het programma su starten!

Is dat niet gevaarlijk? Ja, dat is hartstikke gevaarlijk!

Daarom:

- mag het `/usr/bin/su` bestand niet beschrijfbaar zijn voor anderen dan de eigenaar, anders kan het gewijzigd worden;
- mag de directory waarin het bestand staat (in dit geval `/usr/bin`) niet beschrijfbaar zijn behalve voor eigenaar `root` (en die moet ook de eigenaar zijn), anders kan je het programma `su` door en ander programma met dezelfde naam vervangen;
- moet het programma `su` door-en-door betrouwbaar zijn om te voorkomen dat er lekken inzitten en gebruikers "zomaar" processen als `root` kunnen opstarten.

Dat laaste moet gewaarborgd worden door

1. Het ontwerp. Je moet bijvoorbeeld het password opgegeven van de gebruiker die je met `su` wilt worden (veelal wil je `root` worden en dat is dus het `root` password). Weet je dat password niet, dan stopt `su` en kom je niet verder.
2. De codering moet natuurlijk foutloos zijn. Nu bestaat `su` al meer dan 40 jaar, dus dat zit hopelijk wel goed. Bovendien kun je het in een FOSS omgeving zelf controleren.

Zijn er meer van dat soort programma's? O ja, kijk maar:

```
henk@boven:/usr/bin> ls -l|grep '^-rws'
-rwsr-xr-x 1 root trusted    52264 27 nov 09:58 at
-rwsr-xr-x 1 root shadow     86136 16 jul  2012 chage
-rwsr-xr-x 1 root shadow     82296 16 jul  2012 chfn
-rwsr-xr-x 1 root shadow     81848 16 jul  2012 chsh
-rwsr-xr-x 1 root trusted    55888  4 feb 14:27 crontab
-rwsr-xr-x 1 root audio      27400 16 jul  2012 eject
-rwsr-xr-x 1 root shadow     19256 16 jul  2012 expiry
-rwsr-xr-x 1 root trusted    31568 15 jul  2012 fusermount
-rwsr-xr-x 1 root shadow     85928 16 jul  2012 gpasswd
-rwsr-xr-x 1 root root       40112  5 nov  2012 mount
-rwsr-xr-x 1 root root       19352 16 jul  2012 newgrp
-rwsr-xr-x 1 root shadow     81808 16 jul  2012 passwd
-rwsr-xr-x 1 root root       40000 16 jul  2012 ping
-rwsr-xr-x 1 root root       44592 16 jul  2012 ping6
-rwsr-xr-x 1 root root       23376 16 jul  2012 pkexec
```

```
-rwsr-xr-x 1 root root       44256 23 jan 14:04 su
-rwsr-xr-x 1 root root      121240  1 mrt 19:11 sudo
-rwsr-xr-x 1 root root       19176  5 nov  2012 umount
henk@boven:/usr/bin
```

En er zijn er nog wel meer in andere directories. En je kunt ze uiteraard zelf maken voor je eigen `uid`.

Je ziet wel eens mensen op fora die de toegang tot sommige programma's, die voor `root` gebruik zijn, makkelijker hebben gemaakt door het `suid` bit te zetten. Is dat een goed idee?

We zien in bovenstaand lijstje ook `sudo`. Waarom `sudo` en waarom roept iedereen alsmaar: dan doe ik

```
sudo ....
```

Oorspronkelijk waren er twee methodes om "root te worden": inloggen op een terminal (toen bestond alleen nog maar de CLI, die uiteraard nog niet zo heette), of `su`. En dat is een alles of niets situatie. Je weet het `root` password of niet en dus kun je het systeem volkomen kapot maken of helemaal niet. Er was behoefte om sommige taken (klein beetje kapot maken?) aan sommige gebruikers toe te staan. Je kunt daarvoor natuurlijk een programma schrijven dat van alles controleert (mag deze gebruiker iets en zo ja, mag hij dan mounten en zo ja, mag hij dan andere dingen dan de cd-rom mounten, en zo ja mag hij dat dan vanaf een sessie aan de andere kant van de wereld en zo ja, ...) en dat programma `suid root` maken. Een meer algemeen programma met uitgebreide configuratie mogelijkheden is `sudo`.

Het is gebruikelijk is om `sudo` zo te installeren dat niets mag. Uiteraard de veilige stand. Een systeembeheerder kan dan indien nodig bepaalde zaken gaan toestaan aan bepaalde gebruikers. Heel gebruikelijk is dan dat een gebruiker zich extra moet identificeren door zijn eigen password op te geven (het `root` password weet hij dus niet), zodat een passant bij een per ongeluk openstaande sessie, niet zomaar zijn gang kan gaan. Het is ook mogelijk om te configureren dat geen password nodig is.

Sommige linux distributies hebben een geheel eigen kijk op het werken met `root`. daar is `sudo` zo geconfigureerd dat het aan alle gebruikers toestaat om alles te doen, mits het `root` password wordt opgegeven. Waarschijnlijk om het leven van personen die dit soort benadering gewend zijn niet te moeilijk te maken doet de geïnstalleerde `sudo` op openSUSE hetzelfde. De `sudoers` configuratie bevat:

```
## in the default (unconfigured) configuration, sudo asks for the root password.
## this allows use of an ordinary user account for administration of a freshly
## installed system. when configuring sudo, delete the two
## following lines:
defaults targetpw   # ask for the password of the target user i.e. root
all     all=(all) all   # warning! only use this together with 'defaults targetpw'!
```

maar dit is een redelijk onzinnige configuratie, omdat die hetzelfde is als het gebruik van `su`. en dat hadden we al 40 jaar.

De aanbeveling: "when configuring sudo, delete the two following lines:" ziet niemand natuurlijk behalve als je serieus gaat configureren.

Samenvattend:

- voor sommige zaken (zoals het opblazen van je systeem) moet een proces van eigenaar `root` zijn, anders staat de Kernel dat niet toe;
- je kunt `root` processen starten door in te loggen als `root`;
- je kunt een `root` proces starten door een `suid root` programma te starten;
- er zijn `suid root` programma's die je toestaan om willekeurige programma's (inclusief een `shell`) als root te starten, vanaf de CLI beveel ik `su` daarvoor aan.

Voor de praktijk zie: *Korte uitleg: "root worden" in de praktijk*.