Deferred Authorization translator for the Hurd

Dr. Arne Babenhauserheide

<2021-12-29 Mi>

I realized a project to have your computer ask for authorization when processes access a file. It is built as the translator *checkperms* and the simple permission granting program *queryauth*.

This project is built for the Hurd, where it can be done without too much fuss — and without kernel hacking or dbus-communication.[1]

The translator can delegate permission-granting to the program via two FIFO files. The goal is to create a simple replacement for the use-case of polkit of granting privilege to a process to access some resource after user-interaction with a permission-granting daemon.

This is the simplest structure I could devise for the use-case: The whole system is implemented in about 150 lines of C for the translator (building on hello-mt) and 30 lines of bash for the permission granting program.

This code is supported by a small nlnet grant to provide one component for sound in the Hurd: practical fine-grained access-control.

The original plan in that project was to add sound itself. I retargeted it last year to access control to avoid running into conflicts with the currently running rump-kernel work.

Contents

[1] The Hurd is a collection of servers that run on the Mach microkernel to implement file systems, network protocols, file access control, and other features that are implemented by the Unix kernel or similar kernels (such as Linux). In short: Like Linux, but easier to hack on, with technical advantages that allow avoiding quite a few crutches. And actually older. It allows doing cool things like this deferred authorization translator.

Code

The translator and the related tools are available in the checkperm-deferred-authorization branch in the hurd repository.

The code for the program is provided in this article

Usage Example

We restrict a the node /hello to require explicit permission for every PID that does not have the group `user`. This notably does include processes started by root.

How it looks

First shell as root:

```
settrans -cga /hello $(realpath ~/Dev/hurd/trans/checkperms) --groupname=user
su - user --shell /bin/bash -c 'cat /hello'
# ⇒ HELLOWORLD # user has the group user
cat /hello # root does not have the group user, so
           # this blocks until positive reply in the other shell
```

Second shell (run the program):

```
Process 732 tries to access file /hello but is not in the required group user.
USER       PID %CPU %MEM    SZ   RSS TT STAT   START     TIME COMMAND
root       732  0.0  0.1  148M 3.55M p2 Sso  Mon 1AM  0:01.10 -bash
Grant permission and add group "user" for 5 minutes? [y/N]> y
```

First shell as root:

```
# ⇒ HELLOWORLD
# only blocks once despite getting two reads from cat,
# because for the second read cat already has the group `user`.
```

Trying it yourself

Setup the development environment with the code at ~/Dev similar to https://www.draketo.de/software/hurd-development-environment

Compile and setup the translator:

```
cd ~/Dev/hurd && \
patch -p1 < checkperms.patch && \
autoreconf -i  && \
./configure --without-parted && \
```

```
make && \
touch trans/checkperms.c && \
CFLAGS="$CFLAGS -g" make && \
echo HELLOWORLD > /hello && \
settrans -cga /hello $(realpath ~/Dev/hurd/trans/checkperms) --groupname=user
```

Create the FIFOs:

```
USER=root
GROUP=user
mkdir -p /run/$USER/request-permission
mkdir -p /run/$USER/grant-permission
mkfifo /run/$USER/request-permission/$GROUP
mkfifo /run/$USER/grant-permission/$GROUP
```

Setup the permission-granting program in a separate shell:

```
USER=root
GROUP=user
while true; do
  PID="$(cat /run/$USER/request-permission/$GROUP)"
  echo Process $PID tries to access file /hello but is not in the required group $GRO
  ps-hurd -p $PID -aeux
  if [[ "$(read -e -p 'Grant permission and add group "'$GROUP'" for 5 minutes? [y/N]
    addauth -p $PID -g $GROUP
    echo 0 > /run/$USER/grant-permission/$GROUP
    (sleep 300 && rmauth -p $PID -g $GROUP 2>/dev/null) &
  else
    echo 1 > /run/$USER/grant-permission/$GROUP
  fi
done
```

Access the translator as user without the required group and with the group:

```
su - user --shell /bin/bash -c cat /hello'
cat /hello &
```

queryauth

To simplify usage there are two helper tools:

- queryauth-setup FILE GROUP [PROGRAM]
- queryauth GROUP

queryauth-setup sets the checkperms translator on FILE for the current user, guarded by GROUP, using the authorization query PROGRAM (queryauth if not given)

queryauth waits for requests to add the GROUP and queries the user when needed.

Concept

The translator

The translator is started with a GROUP as argument. When the file is accessed, the translator checks whether the process has the given group. If it does, it returns data read from the underlying file.

If the process lacks the required group, the translator retrieves its USER and PID and writes the PID into a FIFO located at

`/run/USER/request-permission/GROUP`

Then it reads from

`/run/USER/grant-permission/GROUP`

It blocks until it gets a reply. If it reads a 0 (=success), it reads from the file and returns the data.

The permission granting program

The permission granting program reads the PID from

`/run/USER/request-permission/GROUP`

retrieves information about the PID and asks the user whether to allow the program.

If the USER answers no, the RET value is non-zero.

If the USER answers yes, the RET value is zero (0) and the program adds the GROUP to the process at PID (using addauth).

It also starts a daemon that will remove the group again after 5 minutes (modelled after the temporary permissions to run privileged without password granted by sudo).

The program then writes the RET value into

`/run/USER/grant-permission/GROUP`

What if the translator crashes?

If the translator crashes, the permissions return to those of the underlying node. For every user except root this usually means that the process does not have access to the file.

The failure-mode should therefore be safe.

Current limitations

read-only

The current implementation only provides read-access, writing is prevented. This is not an intrinsic limitation, only an implementation artefact.

delegate

The underlying file is currently read by the translator and the data returned to the reading process. To reduce delays, it could directly delegate to the underlying file. With the long term goal to provide multiplexing of access, for example for audio, reading via the translator could be preferable, though.

writing via system shell

Writing to and reading from the FIFOs is currently done with `system()`. It would be nicer to move to an implementation that does not rely on the system-shell.

potential race-condition

Accesses from two different translators can currently race for the reply. To fix this, the translator should write the PID and a random LABEL into the request. The program should repeat that label for replies to ensure that the reply and request can be matched. If receiving a non-matching reply, it MUST be written into the grant again after a random delay to enable a matching translator to retrieve the grant.
REQUEST: PID LABEL
GRANT: RET LABEL (RET=0 is success)
LABEL=$RANDOM

multiple permission-granting programs

The system assumes having a single permission granting program per user. For a setup with multiple unconnected sessions per user (like several TTYs) the permission granting program needs to coordinate between these.

This can be as easy as adding a timeout to the question to the user and writing what you read back into the request if you time out.

Possibilities

The most important use-case for this translator is to make it easier to start programs with reduced permissions and only add these when required.

To setup deferred permissions for a single file, you can create a group just for that file. Then each file can have its own permission granting program. Having dedicated groups decouples authentication and authorization while staying in the conventional *nix permissions scheme.

You can also set this translator on a file that gets accessed first when a process accesses a set of related files that all have the same group. Since the authorization-program here adds the group for 5 minutes, the other files can afterwards be accessed, too.

Since the translator simply defers to a program, that program could do any action to get authorization, including `curl`. Administrators for a local network could therefore set up terminals for unprivileged users that request permissions from a local server when accessing a file. That way permissions can easily be coordinated over multiple machines. (naturally this does not restrict root who can always use settrans -g to get raw access to the file)

Personal note

There's a magic in being asked on the second shell whether cat on the first shell should be allowed to access a file for 5 minutes — and all that in 150 lines of C and 30 lines of shell.

The Hurd is pretty cool!

(though I have to admit that getting the translator to actually work took ages)