Software & Tools

Making MakeTeXPK safer for UNIX installations

Michał Jaegermann

Abstract

Leaving directories open for anybody in the world to write to, in order to ensure that `MakeTeXPK` will work, is often a security concern. This note describes how to avoid that on UNIX systems without losing functionality.

– – ∗ – –

If you run TEX on UNIX, or any other UNIX-like system, and you are using a very convenient and popular setup with automatic bitmap generation, then you'll undoubtedly notice that this requires directories with `'write'` permissions to everybody. In theory, this is not much different than having your `/tmp` directory open that way. In practice, though, especially when your programs are compiled with the `kpathsea` library and you have multiple unprotected font directories, this may cause substantial security and administrative headaches. An option to create automatically all bitmaps in one directory used only for that purpose, and to move them later by hand after careful examination to final locations, is not very practical if you serve multiple printers with different characteristics and is always unattractive to busy system administrators.

A first, but rather feeble line of defense, if your variant of UNIX supports it, is to set the 'sticky' bit on the directories in question like this: `chmod at` pk+. That way files can be removed or overwritten only by their owner(s). Unfortunately this does not prevent a "denial of service" attack. For example, a perpetrator may fill up directories with garbage such as empty files with expected names thus preventing later generation of required bitmaps. The above is just a prank, without lasting long-term consequences, but repeated often enough it may turn into a major nuisance. There are other, more insidious possible threats, which do occur in practice, especially when an attacker comes over a network using compromised legitimate accounts.

If you ever even thought of running `MakeTeXPK` "suid" (that means, in a mode which gives a program all privileges of its *owner*) you should drop the idea immediately, especially if `MakeTeXPK` is owned by `root`. `MakeTeXPK` is a *shell script* and for many reasons running shell scripts in "suid" mode is one of the biggest security holes you can think of. Modern UNIX kernels usually simply disable "suid" scripts, but even if they are permitted on your system, avoid using them. Problems with "world writable" directories pale in comparison.

Fortunately, there are safer approaches. As long as you have a compiler and an editor, `MakeTeXPK` and relatives (`MakeTeXTFM`, `MakeTEXMF`, . . .) may be executed indirectly by a small compiled "wrapper" program and this latter program can be safely made "suid".

The notes below describe the steps leading to such a modification. Due to assorted variations in TEX distributions and installations it is highly unlikely that they will work for you literally as given — unless you happen to run a very similar system. To make details easier to follow they will be shown for one particular distribution (`teTeX`, version 0.2, for Linux). This is only an example but it should give an idea how to proceed in other cases. Modifications should be rather straightforward. There are also possible variations in UNIX behaviour. See, for example, the Solaris note below.

It goes without saying that you need `root` access for all (okay, most) of these steps.

- Create new user account, say `tex`, on your system and include it in some innocuous group, e.g., `tex`.

The only purpose of this "user" will be to own common TEX files. You may already have some suitable group, such as `nogroup` or `nobody`, so instead you may include your new user there.

- Give the `tex` account `/bin/false` for a login shell and disable password by putting `*` in the corresponding field. Home directory is not terribly important. Nobody will ever be logging into this account.

The completed entry in the password file will look something like this:

```
tex:*:117:65535:Owner of TeX files:
                /usr/local/tex:/bin/false
```

- Do `touch /usr/spool/mail/tex` to create an empty file. This action closes a loophole in some mail delivery programs. You may find the file already in place, made by some system administration utility. Change owner and group of this file to those of root (`chown root:root /usr/spool/mail/tex`) and also remove all read and write permissions on it (`chmod 000 /usr/spool/mail/tex`).
- Make sure that `MakeTeXPK`, and similar scripts you want to execute the same way, are **not** in

your `$PATH`, or at least not earlier than the intended location of your "wrapper" program(s). Original scripts will not be called directly, however, the "wrapper" program will "inherit" their names thus presenting the same interface to users and other programs.

Any convenient location will do for scripts, but if your system conforms to the TeX Directory Structure, then a subdirectory of `$TEXMFROOT` will be a logical place. For the particular `teTeX` distribution, version 0.2, it is enough to delete links in a directory `/usr/local/bin`. Real scripts `MakeTeXPK`, `MakeTeXMF`, and `MakeTeXTFM` reside in `/usr/local/tex/scripts-0.2/bin/` and may be left there.

- Edit all scripts in question to supply absolute paths to all executables. Don't forget to perform this task in other scripts which may be called by our `MakeTEX...` script (`append_db` in the `teTeX` distribution). Change the mode used when creating new files to 444 and to 755 for directories.

A proper way to do this is to start a script with a series of shell variable definitions similar to

```
MF="/usr/local/bin/mf"
```

and replace all later occurrences of `mf` in the script by `$MF`. That way, if you later move your METAFONT executables to some other place, script editing will be limited to one place; similarly for other programs. Absolute locations are required since, for security reasons, we will limit `$PATH` only to `"/bin:/usr/bin"`. This means that in theory you may leave things like `test`, `echo` or `rm` alone. In the latter case, for "dangerous" commands like `rm -f`, it is still a good idea to replace them with definitions similar to `RM="/bin/rm -f"` to get better control of what you are really executing.

As a side effect this keeps user-aliased or redesigned versions of these commands from putting out unexpected and unwanted text or hanging because of a need for `tty` input as in the case of the common and usually desirable alias of `rm` to `rm -i`. The unplanned appearance of extraneous text on `stdout` is one of the most common reasons for the `MakeTeXPK` to fail on its first pass.

Depending on your level of mistrust, you may use a similar approach to `echo` and `test` as well, but in the sample files, I have chosen to be more relaxed. Moreover, they may be "built-ins" in your shell.

- Edit sample source given in the appendix to adjust it to your system and compile.
- Install results of the compilation somewhere in your `$PATH`. The directory `/usr/local/bin` is usually a good place. Go there and name your program `MakeTeXPK`. Change its ownership and group to that of user `tex` by typing `chmod tex:tex MakeTeXPK`. (Depending on your variant of UNIX you may have to use a dot instead of a colon to separate the user and the group name, or you may have to do that in two steps, using also another command, `chgrp`, to accomplish the above. Use the group to which you assigned your `tex` user. This is only an example). The program needs "execute" and "set uid" privileges (`chmod 4755 MakeTeXPK`). Also for your other `MakeTeX...` scripts, provide corresponding "call points" with their names via file links (`ln MakeTeXPK MakeTeXTFM`; `ln MakeTeXPK MakeTeXMF`).

Solaris note: Passing ownership privileges to a subprocess, as illustrated above, works for Linux and other assorted UNIX systems. Still, I am informed by Ulrik Vieth (`vieth@thphy.uni-duesseldorf.de`), that on Solaris systems this happens only when the wrapper is "suid" and owned by `root`. Therefore a

```
setuid(geteuid());
```

line from a sample source will not work as intended (either the call will fail or the subprocess will not be owned by the `tex` account). One should instead set explicitly `TEXGROUP` and `TEXUSER` of a type `uid_t` and include a replacement code like this

```
setgid(TEXGROUP);
setuid(TEXUSER);
```

in the given order — to achieve the same effect. This may apply as well to other UNIX variants. *Caveat emptor!*

- Change ownership of all your font files to `tex`. Actually you may make `tex` an owner of whole directory trees in TeX system files. Assuming that all you want to assign that way is in a tree rooted in `texmf`, you may accomplish that by doing `chown -R tex:tex texmf`. If your `chown` does not understand the `-R` (recursive) flag, then something similar to the following should do:

```
find texmf -print | xargs -n1 chown tex:tex
```

See also `chown` remarks in the previous item.

- Remove "write" permissions for anybody but owner on all directories in question. A command like the following one should accomplish that task (be careful, you do not want to change non-directories):

```
find texmf -type d -print | xargs -n1 chmod 755
```

You are done. Now, whenever MakeTeXPK is called directly from a command line or by some other program like dvips, your "wrapper" program will be executed instead. It will call, in turn, a "real" script but one already with the id of the owner of your font directories.

Concluding remarks

The presented solution is not entirely without problems. Due to "out of sync" ownership and permissions, kpathsea library functions may fail, depending on the exact moment this happened, when trying to write the missfont.log file in cases when font-making was not successful. This can likely be hard to resolve without modifying the library itself. If you do encounter this problem then a simple workaround would be to create an empty missfont.log, owned by you, and to give it write permissions for everybody (touch missfont.log; chmod a+w missfont .log). When you are done simply change permissions back to the original state.

Another possible trouble spot will occur when you have "private" fonts because you are conducting some font-making experiments, for example, and you would like to have bitmaps created in places owned by you and not by tex (otherwise you will not be able to remove the results of failed tests). In this case make yourself a private, executable copy of the MakeTeXPK script, edit it accordingly and make sure that it can be found *earlier* in your $PATH than the system program with the same name. You will not able to deposit anything in system directories, but this is most likely what you want anyway. Your script does not have to run "suid", so repeating all of the above is not necessary.

Last but not least, there is another possible approach to the whole problem. There are only a few commands (mv, mkdir, chmod) which have to modify TEX "system" directories. Instead of running the whole MakeTeX in "suid" mode you may write special versions (texmv, texmkdir, texchmod) of these, which would operate "suid" tex, and use them as replacements in the MakeTeXPK script whenever needed. Whether this is a better idea depends entirely on your situation and security requirements.

◇ Michał Jaegermann
10923 36 Avenue
Edmonton, Alberta,
Canada T6J 0B7
Email: michal@ellpspace.math.ualberta.ca,\\ michal@gortel.phys.ualberta.ca

Appendix – Sample wrapper program code

Sample C code for a wrapper program for teTeX, version 0.2, Linux distribution. Adapt with caution!

```
/*********************************************/
/*                                           */
/*  Executable wrapper for MakeTeX...        */
/*  programs.  Calls its namesake from       */
/*  TOOLS directory.  Provide links with     */
/*  different names to make it multipurpose  */
/*                                           */
/*  Michal Jaegermann, Feb 11 1995           */
/*                                           */
/*********************************************/

#include <unistd.h>
#include <string.h>
#include <stdlib.h>

#define VERSION_S "0.2"
/*
 * If you do not have an ANSI compiler you may
 * use an "explicit" single string in TOOLS
 * define; this is just a way to make future
 * modifications easier.
 */
#define TOOLS  "/usr/local/tex/scripts-" \
               VERSION_S "/bin/"
#define ASIZE  120

/*
 * This is a list of names under which we are
 * willing to execute. It must be NULL
 * terminated.
 */
const char *accepted[] = {
    "MakeTeXPK",
    "MakeTeXTFM",
    "MakeTeXMF",
    NULL
};

int
main(int argc, char **argv)
{
    char doer[ASIZE] = TOOLS;
    int idx = 0;
    /*
     * If your compiler is broken and the
     * construction below does not work then
     * "tail = strchr(doer, '\0');", or
     * equivalent, will serve as well.
     */
    char *tail = doer + (sizeof(TOOLS) - 1);
    char *start;

    /* find our base name */
    start = (start = strrchr(argv[0], '/')) ?
                     (start + 1) : argv[0];
    /* check if we are on the list */
    while (1) {
        if (NULL == accepted[idx])
            exit(1);      /* not on the list */
        if (0 == strcmp(accepted[idx], start))
            break;        /*  this is ours   */
        idx += 1;
    }
    /*
     * Set pretty bland, but hopefuly secure
     * environment; we intend to run this
     * program 'suid'.
     */
    setenv("PATH", "/bin:/usr/bin", 1);
    setenv("IFS", " ", 1);
    /*
     * You may want/need some other calls to
     * setenv().  For example, if your system
     * has an environment variable pointing to
     * shared libraries it should be set here.
     */

    /*
     * Attach our name at the end of a
     * directory string.  This assumes that
     * real scripts in TOOLS directory  will
     * be called by their own names (but
     * indirectly)
     */
    strcpy(tail, start);

    /*
     * Get the privileges of the owner of this
     * program, then execute the script and
     * return its results
     */
    setuid(geteuid());
    return execv(doer, argv);
}
```