Insecure /tmp usage due to predictable paths #1137

Open **orlitzky** opened this issue on May 18 · 3 comments

orlitzky commented on May 18 Contributor

I just noticed commit `ed2fd9b` which led me to `git grep /tmp` in the codebase. There are several places where a fixed (or at least predictable) path under `/tmp` is used for temporary storage. This creates a general class of vulnerability that can usually be exploited by other users on the same machine.

For example, since `/tmp` is world-writable, anyone on the machine can create `/tmp/sing_log.1` through `/tmp/sing_log.65535`, making themselves the owner of those files. Later, singular will open and write to them as some other user, namely whoever is running singular. In an extreme case, if singular is running as root, and if the bad guy has created `/tmp/sing_log.N` as links pointing to important system files, then logging can erase (overwrite) those files.

I haven't tried to exploit them all, but in a few other cases, temporary files are used to store data or commands that are fed to other programs, which makes some level of code execution likely.

The `mkstemp()` function is a safe replacement within C, but would have to be exposed to the user somehow to avoid the same problem in singular code.

hannes14 commented on May 19 Member

should be fixed with `72df188`
for --log. For all other cases use --no-shell

orlitzky commented on May 21 Contributor Author

For the logs I think this is sufficient, but for the rest, the problem is much larger than just shell characters being injected. Here's a real example that takes advantage of the predictable files used by the sdb debugger.

1. First, if you're on linux and have this security option set, you'll have to turn it off for this particular example to work: `$ sudo sysctl fs.protected_regular=0`

2. As some other user (not your desktop user), create a bunch of files named `/tmp/sdX`, where `X` runs through enough integers to cover any PIDs that will be used in the near future.
3. As your normal user, run `singular --sdb`, hit an error or breakpoint, and then use `e` to enter edit mode.
4. Now sdb will have written some code to `/tmp/sdX` (for some `X`), but that file is controlled by the user who created it.
5. As the other user, edit the temporary file being used by sdb.
6. As your normal desktop user, close the editor.
7. The other user has now written a procedure that your normal desktop user will run.

There are more clever (and more automated) variants of this. The fundamental problem is that when using a predictable filename under `/tmp`, you may wind up using a file that someone else controls. Often, control of such a file can be exploited.

orlitzky commented on Jul 25 Contributor Author

I see you fixed the sdb issue in `5f28fbf`, thanks =)

I think that leaves only the more difficult problem of obtaining temporary file and directory names within singular code. For example, in `gfan.lib`,

```
string filename = "/tmp/gfanlib_secondaryFan_"+string(random(1,2147483647));
```

is attempting to create the file safely -- at least without a collision -- but will not fail if someone has pre-created all 2147483647 possible files and made them writable.

Similarly, when you need to run several commands in a row as in `tropical.lib`,

```
system("sh","cd /tmp; latex /tmp/newtonsubdivision"+string(rdnum)+".tex; dvips
/tmp/newtonsubdivision"+string(rdnum)+".dvi -o; command rm newtonsubdivision"+string(rdnum)+".log;
command rm newtonsubdivision"+string(rdnum)+".aux;  command rm
newtonsubdivision"+string(rdnum)+".ps?;  command rm newtonsubdivision"+string(rdnum)+".dvi; xdg-
open newtonsubdivision"+string(rdnum)+".ps &");
```

the shell will blindly proceed even if someone else has e.g. created a malicious dvi file beforehand, causing you to process his file after "latex" fails to write its output.

In the first case, you would like to obtain the temporary filename atomically, and fail if it cannot be done. In the second case, you'd like to run the latex pipeline all within one temporary directory, having obtained that temporary directory securely.

For those reasons I think it would be a good idea to wrap the C functions `mkstemp()` and `mkdtemp()`, making them available in singular code. That way all of the hand-written `/tmp` stuff that exists now could be replaced with calls to the wrappers and (a) it would be more secure against attackers (b) it would be more robust against bad luck and (c) the code would be a lot simpler.

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

2 participants