Notes on "On the Security of UNIX" by D. M. Ritchie

G. Lettieri

21 September 2023

"On the Security of UNIX"[1] is a short paper by D. M. Ritchie, first published in the 6th edition of "The UNIX Programmer's Manual" (1975) and then adapted for the 7th edition (1979). The main message of the paper is that

> UNIX was not developed with security [...] in mind.

The paper lists several examples to support the above statement. Most of what Ritchie says is still true today, in some form.

The first two examples deal with denial of service problems caused by overuse of system resources. The first example shows a four-line script that attempts to create an arbitrarily deep directory structure, and will only stop when either the i-node table is full or all disk blocks have been used[2]. The second example exhausts system resources (either swap space or process table slots) by creating an excessive amount of background processes (the syntax used is the same as today). The V6 UNIX would either panic (i.e., crash) or become unusable in such circumstances. V7 UNIX introduced limits on the number of processes that a user can create, and modern Unix and Unix-like systems can do the same; Ritchie, however, duly notes that this is not sufficient and the system can still become essentially unusable, or even crash, if these processes use too many resources.

Ritchie then goes on to explain file permissions. There are a few problems he sees in this area: i) permission checks are skipped for the root user; ii) default permissions are too liberal; iii) the permissions on directories are not intuitive. Problem ii) can mostly be fixed by following Ritchie's advice, or by

[1] `http://www.tom-yam.or.jp/2238/ref/secur.pdf`

[2] Modern readers may not recognize the syntax of the V6 script: pre-v7 UNIX still used Ken Thompson's minimalist shell. This shell could only run scripts by redirecting its standard input from a file. The "`goto` *label*" statement was actually an external command that used `seek()` on its own standard input file pointer, looking for a line starting with a colon and the string *label*. Since the file pointer is shared between the shell and its children, after `goto` the shell would go back and read the command that followed the "`:` *label*" statement. The colon itself was another external command that did nothing. The `goto` command has disappeared, but the colon has remained (as a built-in command) in modern shells, where it is used either as a fast "`true`", or when you are only interested in expanding its arguments. The V7 version of the paper uses the more familiar Bourne Shell syntax.

remembering to set the `umask` that was introduced in V7, but problems i) and iii) still exist, and i) has gotten worse: the root user has acquired a bewildering set of capabilities that go far beyond the ability to skip filesystem checks. This is a problem when we are forced to run a program as root because the program needs one of these capabilities. For example, to give a web server the right to open port 80, we must also give it the right to wipe out the entire filesystem.

Ritchie then moves on to the topic of passwords. He says that here UNIX behaves better than most other systems of the time, since passwords are always stored in encrypted form[3] This allowed the `passwd` file to be world-readable, so that it could also be used as a database mapping usernames to uids (replacing an older `uids` file). This database is used, e.g., by `ls`, `ps`, `chown` and so on. However, Ritchie points out that a world-readable `passwd` file opens the door to brute-force/dictionary attacks, so users must choose strong passwords. This is even more true today, of course. However, to help prevent brute-force and dictionary attacks the encrypted passwords have now been removed from `passwd`, which must remain world-readable for compatibility, and are now stored in the `shadow` file, which is readable only by root.

Ritchie also mentions an old trick that works on UNIX too: run a fake "login" program on a terminal, and wait for an unsuspecting user to come in and type in their password. The fake program can now store the password somewhere (or mail it to the attacker) and then perform the normal login. This is still true today, and it's not much of a problem only because we no longer log into our systems from public-access terminals. Windows NT is safer in this regard: the Ctrl+Alt+Del key combination cannot be intercepted by user programs and provides a "trusted path" from the user to Windows itself.

Then Ritchie's touches on the very important topic of set-UID/set-GID programs (his own invention, and the only UNIX patent). First, he notes that set-UID/set-GID programs must not be writable by attackers (a problem related to permissions, as above). This is a general note: the permissions of a file (including the set-UID and set-GID flags) are stored in its i-node: they don't change when you write into the file. More importantly, he notes that these programs must be "sufficiently careful of what is fed into them". This is still extremely important, and we will spend some time on the attacks that target this very feature of UNIX(-like) systems. The exact example Ritchie gives is no longer directly applicable, since it is based on an ancient version of `mail`[4],

[3] The encrypted-passwords idea was suggested by M. V. Wilkes in '68, but come to Unix via Multics, where it was developed in the aftermath of a funny incident in CTSS (the precursor of Multics). CTSS stored passwords in plain text, in a file readable only by administrators, like early versions of Unix. One day, two administrators at MIT were editing the message-of-the-day file and the password file, without knowing about each other. Since the editor program always used the same name for temporary files, the two files were accidentally mixed up. Until the system was stopped (by exploiting another bug to force it to crash) all users logging into the system were greeted with everyone's passwords in clear text.

[4] You can find the sixth edition of the Unix Programmer's Manual online, and read the man page of `mail`: the program allowed the user to append a message to the `.mail` file in the recipient's home directory or any other directory. Interestingly, the v6 `mail` command is vulnerable to the integrity attack, but apparently not to the privacy attack described by Ritchie, since it always sends only its standard input. The v5 `mail` command, instead, also

but the attacks use techniques that are still valid today: misuse of program arguments and filesystem links. Moreover, V7 actually exacerbated the problem with set-UID/set-GID programs with the introduction of environment variables, as we will see. The V7 version of Ritchie's paper doesn't show any awareness of this fact yet.

The final `mount`-related problems also apply, in some form, to modern systems. The `mount` operation has finally become privileged[5] but personal computer users still need a way to do this, if they want to use their pen-drives or virtual disks and so on. Much care must be taken, before trusting the contents of these file systems.

accepts the name of a file to send, and was therefore vulnerable to both attacks. On the other hand, v5 mail used a different name for the target mailbox, instead of `.mail`. Ritchie seems to be referring to an interim version of `mail` that was never officially released. As Ritchie says in the V7 version of the paper, the V7 `mail` command is very different from the older ones, and it is not vulnerable to either attack.

[5]The first revisions of the Programmer's Manual included the phrase "This call should be restricted to the super-user" in the BUGS sections of the `mount` and `umount` entries.