"Permission Denied!" Why?

Hiroki Nakajima, Kenta Ishiguro, Kenji Kono
Keio University

Motivation. Modern operating systems enforce complex access control over computing resources such as files. For example, file access control consists of two layers: capabilities and traditional `rwx` control. Even for experienced users, it is not always easy to comprehend the access control rules. When manipulating files and directories, the user often encounters a notorious error message: "permission denied". It is often hard to grasp the root cause of the denied permission, although the user with moderate familiarity with the access control should be able to understand why.

The following is an intricate example that needs a deep understanding of Linux file systems.

```
Taro% ls -l /tmp
lrwxrwxrwx  Taro   /tmp/lnk -> XXX
Taro% cat /tmp/lnk
I am Taro!
Taro% sudo cat /tmp/lnk
cat: /tmp/lnk: Permission denied
```

Without the intimate knowledge of Linux file access permissions, the user cannot understand the root cause of this "permission denied". In this example, the file accessible in the user privilege can not be accessed in the root privilege.

Goal. The goal of this work is to offer clues to the root causes of denied permissions. Error messages generated by operating systems can be ambiguous and provide insufficient information to diagnose the errors. In Linux, the number of error messages (e.g., errno) is 150 at most, which is too few to explain the root cause of each error. The functions in `fs/namei.c` have 15 statements to return the same errno (`EACCES`, so-called "permission denied"), each of which corresponds to a different root cause upon opening a file. Windows has more than 10,000 error codes, but not enough to ensure one-to-one correspondence between an error code and its root cause.

Approach. This poster proposes Permod, a compiler extension that augments the Linux source code to generate rich logging information, which helps the users understand the root cause of "permission denied". The fundamental approach is as follows. Permod inserts a logging statement that generates `x != 0` for statement like `if(x) return -EACCES;`. Beginning with a statement that returns errno, all the basic blocks are traversed backward up to a syscall entry point. During this traversal, all the conditions that reach the return statement are piled up. Finally, Permod inserts logging code that pretty-prints the piled-up conditions. Fig. 1 overviews Permod.

Preliminary Results. Permod has been implemented as a LLVM pass with LLVM-17.0.6 and Coccinelle [2]. For

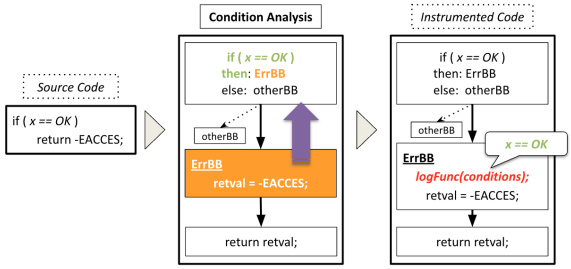

Figure 1. Overview of Permod

the example described above, Permod generates the log as follows:

```
sysctl_protected_symlinks != 0  // enabled
dir_mode == S_ISVTX  // 'sticky bit' set
```

The first line indicates `fs.protected_symlinks` is enabled in the Linux sysctl, and the second line represents the sticky bit of the directory is set (`dir_mode` is `S_ISVTX`). When the `protected_symlinks` is enabled, symbolic links in a sticky world-wide writable directory, such as `/tmp`, are protected from being followed by anyone but the owner. Therefore the root cannot access the symbolic link owned by `Taro`.

Related Work. There are several studies on logging improvement in software. Li et al. [1] proposed a deep-learning approach to suggest better logging locations. LogEnhancer [4] and SecLog [3] are static analysis tools to enhance log messages. They focus on the places where log statements already exist or must exist.

Summary & Future Work. This paper presents Permod, a compiler extension for OS-generated error messages by providing rich logging information. We plan to measure the effectiveness of the information and determine better granularity in the future.

References

[1] Zhenhao Li, Tse-Hsun (peter) Chen, and Weiyi Shang. 2021. Where shall we log? studying and suggesting logging locations in code blocks. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE '20)*. 361–372.

[2] Yoann Padioleau, Julia Lawall, René Rydhof Hansen, and Gilles Muller. 2008. Documenting and automating collateral evolutions in linux device drivers. In *Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2008 (Eurosys '08)*. 247–260.

[3] Bingyu Shen, Tianyi Shan, and Yuanyuan Zhou. 2023. Improving Logging to Reduce Permission {Over-Granting} Mistakes. In *USENIX Security Symposium (USENIX Security 23)*. 409–426.

[4] Ding Yuan, Jing Zheng, Soyeon Park, Yuanyuan Zhou, and Stefan Savage. 2011. Improving software diagnosability via log enhancement. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '11)*. 3–14.