Q1) System Description:
1st Assumption: The binary target files would have data exactly the same as the structure file_data; therefore, we will read and write them in the same manner

2nd Assumption: Folders have their own acl file, within the folder, with the name "folder_name" + "_acl". That is true for every folder and subfolder.

3rd Assumption: Our own programs, setfacl, getfacl, fgetc, fputc, create_dir and change_dir, have their own acl files too, named as "file_name" + "_acl"

4th Assumption: setuid bit of all the target binary are turned on. (setfacl, getfacl,..., create_dir have chmod u+sx binary_name)

In my system,
Getfacl: Check if the target is a file, folder, or none. Next, check if the Target user can read the file (check for owner, acl & other). Read the file, then display the content if the user has permission to read.

Setfacl: Check if the target is a file, folder, or none. Next, check if the user can write the file (check for owner, acl & other). Take input from the user and check whether they are correct (valid user and valid acl). Write the file if you have permission. If the target is a folder, write recursively in each subfile and subfolder the updated acls.

Fgetc: Check if the target is a file, folder, or none. Next check if the user can read the file (check for owner, acl & other). Read the file, then display the data if the user has permission to read.

Fputc: Check if the target is a file, folder, or none. Next check if the user can write the file (check for owner, acl & other). Take input from user. Write the file if you have permission. If the target is a folder, write recursively in each subfile and subfolder the updated acls.

Change_dir: Check if the target is file (return), folder (pass), or none (return). Check of the user has read and execute access to the folder they want to change to. If all checks pass, print the current working directory, then do chfir(target) command, then print current working directory again for proof.

Create_dir: Check if the target is file (return), folder (pass), or none (return). Check of the user has write and execute access to the folder they want to change to. Input from user the directory they want to create. Check if the input directory already exists. If all checks pass, create the directory using mkdir command, inside the same folder. Then create acl_path for that inherit dac and acls from the parent folder.

I have employed all checks:
For setfacl: if the user is owner and has write permission, if the owner has an acl entry and has w permission, if the user is not found in the above two, then does other have write permission. If none of the checks pass the gracefully fail. Also if user is updating the acl of an already registered user or we are creating a new acl entry.

For getfacl: if the user is owner and has read permission, if the owner has an acl entry and has w permission, if the user is not found in the above two, then does other have write permission. If none of the checks pass the gracefully fail. If all checks pass, show dac and acl.

For fgetc: if the user is owner and has read permission, if the owner has an acl entry and has w permission, if the user is not found in the above two, then does other have write permission. If none of the checks pass the gracefully fail. If all checks pass, show dac and acl.

For fputc: if the user is owner and has write permission, if the owner has an acl entry and has w permission, if the user is not found in the above two, then does other have write permission. If none of the checks pass the gracefully fail. Also if user is updating the acl of an already registered user or we are creating a new acl entry.

For create_dir: For setfacl: if the user is owner and has write and execute permission, if the owner has an acl entry and has w permission, if the user is not found in the above two, then does other have write permission. If none of the checks pass the gracefully fail. Also if user is updating the acl of an already registered user or we are creating a new acl entry.

For change_dir: if the user is owner and has read and execute permission, if the owner has an acl entry and has w permission, if the user is not found in the above two, then does other have write permission. If none of the checks pass the gracefully fail. If all checks pass, show dac and acl.

(if anywhere user is found and does not have appropriate permission, program fails gracefully)

I defended against: 1) Someone with not appropriate permissions trying to access the file/folder.
2) If each check is valid, then I am not adding a new entry for every acl or any change in dac. I am just updating if the entry if already present. I add only when the user is not present, both in dac and acl.
3) Before creating and changing dir, I am employing proper checks, as both write and execute or read and execute should be present.
4) I am applying seteuid only once, that is for the privileged operation of final read or write, after all checks have passed. Then I immediately switch back to invoking user.
5) There are very low, errors since I am checking permissions of the invoking user, and the input (s)he gives, then checks on the input too. Only then do the privileged operation.

To execute: I am using a caller program, which uses execv command, which takes in 2 arguments, first is our own created target program (setfacl, getfacl, etc.) and second is the argv[] array, whose first and only element is the path of the target file/folder.

Execv calls the target binary, which then passes on the argv array containing the target path, then according to the target binary, the program proceeds.

Commands: I am using, fread, fwrite, opendir, struct stat (and its related commands), fopen, fclose, malloc, realloc, getuid, getpwuid, seteuid, getcwd, chdir, mkdir, free, and various commands for strings (strcpy, strdup, strstr, etc.)

Q2) In simple sudo, I am taking in 2 arguments, first is the path to target file/folder, second path to the target binary.

Assumption 1): Target binary has their acl_file ready, i.e. “file_name” + “_acl”, in which all permissions (dac, acl) are stored / struct is written.
Assumption 2): setuid bit of all the target and this binary are turned on. (setfacl, getfacl ,...,simple_sudo have chmod u+sx binary_name)

First, I check if the user is calling for a valid target binary [should be one of the 6: setfacl, getfacl, fgetc, fputc, create_dir, change_dir]. Then fetch the acl path of the file (using get_acl_path function). Then checking for if the user has permission to execute the target binary or not.

I check, if the user is the owner and has execute, if not, then if they are present in acl and have execute, if not then if the other has execute permission. If none of the checks pass, program fails graceful. (If anywhere user is found, and is with inappropriate permissions, the program fails gracefully)

Using struct stat, I get the uid of the owner of the target program, then using setuid command I set the uid of the simple_sudo to the owner of the target file/folder.

Then I use execv command which takes in 2 arguments, first is our own created target program (setfacl, getfacl, etc.) and seconds is the argv[] array, whose first element is the path of the target file/folder. Then the target binary is called.

To execute: I am using a caller program, which uses execv command, which takes in 2 arguments, first is our own created simple_sudo binary and second is the argv[] array, whose has 2 elements, first element is the path of the target file/folder, second, our own created target program (setfacl, getfacl, etc.)

Execv calls the simple_sudo, which then passes on the argv array containing the target file/folder path and target binary path, then according to the simple_sudo, the program proceeds (whether to call target binary or not)

Commands: I am using, fread, fwrite, struct stat (and its related commands), fopen, fclose, malloc, getuid, setuid, free, and various commands for strings (strcpy, strdup, strstr, etc.)

I defended against: 1) Someone with not appropriate permissions trying to execute the target binary.
2) I am applying setuid only once, for the privileged operation of executing the target binary (execv), after all checks have passed. Then I immediately switch back to invoking user.
3) There are very low, errors since I am checking permissions of the invoking user, and the input (s)he gives, then checks on the input too. Only then do the privileged operation.