OBL4-OS

Kristoffer Longva Eriksen

> Done with Kristoffer Juelsen

August 12, 2020

This is a mandatory assignment. Use resources from the course to answer the following questions.Take care to follow the numbering structure of the assignment in your submission. Some questions may require a little bit of web searching. Some questions require you to have access to a Linux machine, for example running natively or virtually on your own PC, or by connecting togremlin.stud.iie.ntnu.no over SSH (Secure Shell). Working in groups ispermitted, but submissions must beindividual.

1 File systems

1. Name two factors that are important in the design of a file system.

Two important factors in the design of a file system are reliabaility and naming. Reliability is important to keep data safe and accessible and is achieved through the use of transactions. File systems use transactions to write to files, similarly to how the operating system uses critical sections to atomically operate on critical sections. Reliability is also improved through tracking checksums to provide integrity and storing files accross different hardware to prevent loss in case of hardware failures. Naming is important to help users keep track of and organize their files. They are usually set up in a tree like structure with folders grouping together related files.

2. Name some examples of file metadata.

Metadata is the information stored about the files, instead of the data making up the content of the files. Examples of metadata could be filesize, geographical information, time created etc.

2 Files and directories

1. Consider a Fast File System (FFS) like Linux's ext4.

(a) Explain the difference between a hard link and a soft link in this file system. What is the length of the content of a soft link file?

A hard link is created any time a new file is created, giving the operating system an absolute path to use to access the new file (an example could be `/home/krloer/general/ntnu/idatt2202`). This link will be viable no matter where we are in the operating system. A soft link is a link from a location in the tree structure to another location (an example could be `../idatt2103/Oving6`). This link is only usable in the directory it is created in, and does not work other places in the operating system. The length of the content of a soft link file is equal to the length of the original link to the file we are linking to.

It is possible to create multiple hard links to the same file, possibly creating a cyclic structure instead of a tree structure (if there is no protection against this). It is also possible to create a soft link from a to b, and then unlink b. This would create a dangling soft link as trying to open b using the soft link would fail.

(b) What is the minimum number of references (hard links) for any given folder?

The minimum number of hard links for any given folder is two. The first is the link from the parent directory, e.g. `/home/tmp/`, and the second is the link through the directory itself, e.g. `/home/tmp/.`.

(c) Consider a folder /tmp/myfolder containing 5 subfolders. How many references (hard links) does it have? Try it yourself on a Linux system and include the output. Use ls -ld /tmp/myfolder to view the reference count (hint, it's the second column in the output).

Output: `drwxr-xr-x 7 krloer krloer 4096 Nov 10 12:29 tmp/myfolder/`

It has seven hard links. The first two are the links described in the previous answer, one from the perspective of the parent folders, and one from within the directory itself. In addition to these it contains one hard link for each of the five subfolders.

(d) Explain how spatial locality is acheived in a FFS.

Spatial locality is achieved through locality heuristics. These help the operating systems with multiple things, including deciding where in the file system a file should be placed to optimize performance. They are based on free space maps and directories and index structures.

FFS in particular uses the locality heuristics "block group placement" and "reserved space". Block group placement divides the memory into blocks. Files are placed in the same block as its directory, while new directories are placed in different blocks than its parent directory to trade short term for long term locality. Reserved space means that FFS makes 10% of a disk unavailable to the user, to maintain good locality, as this would be very difficult on a full disk.

2. NTFS - Flexible tree with extents

(a) Explain the differences and use of resident versus non-resident attributes in NTFS.

NTFS uses extents, which are variable sized regions of memory, instead of blocks from FFS. These are stored as attribute records in the master file table, and keep control of where files are physically stored. The MFT stores an array of 1 KB entries, which means that if an entry is smaller than 1 KB it is stored as a resident attribute, but if its too large, it is stored elsewhere as non-resident attributes, with pointers to it in the MFT.

(b) Discuss the benefits of NTFS-style extents in relation to blocks used by FAT or FFS.

Extents allow more effective use of memory, as small files, dont take up an entire block, but instead use only the required space. They also prevent the file fragmentation that are created when files are larger than blocks and have to be split into different areas in memory. With extents these can be stored contiguously.

3. Explain how copy-on-write (COW) helps guard against data corruption.

Copy-on-write means that when a file is updated or changed, the file is copied to a new location and the copy in the new location is changed, instead of changing the original file. This means that if anything goes wrong when editing the file, the file from before the change started is stored safely. If the file itself was edited without a copy, errors could cause corruption in the file itself. COW systems can also use checksums to make sure the integrity of a file is maintained.

3 Security

1. Authentication

(a) Why is it important to hash passwords with a unique salt, even if the salt can be publicly known?

A common hash cracking attack uses a rainbow table. This is more adaptable than a previously computed hash lookup table as it can be used on all different hashes, but it is slower each attack as it has to compute each hash to match with its entries. As this is already a time consuming process, unique salts slowing it down even more can make it significantly less effective. The fact that the unique salts are publicly known, doesnt decrease the computation required to crack the hashes.

Unique salts ensure that the computation for cracking one users password can not be used to check another users password. If the password 12345 is listed in a rainbow table, and md5 hashes are used, the attacker will compute the md5 hash and match this against the password hash. If it matches, the hash has been cracked. If unique salts are not used, this information can also be used to check whether other users have 12345 as their password or not, saving one future computation for every other password for each computed hash cracking the first password.

However, when password are salted with unique salts, the computation to check the first hash, will not give any information about the remaining hashes. This means that in a worst (best) case scenario, the attacker has to calculate the hash for each password in its lookup table for every password hash its cracking. Without unique salts the ammount of computations to check all of the password in a database would be the ammount of known password in the attackers lookup table. With unique salts the ammount of computations would be the ammount of passwords in the lookup table times the ammount of hashed passwords.

(b) Explain how a user can use a program to update the password database, while at the same time does not have read or write permissions to the password database file itself. What are the caveats of this?

A program can use the setuid() and setgid() library calls to escalate the privileges of the program, unrelated to the privileges of the users calling the program. This allows a user to execute a program that increases the programs privileges, executes specified operations on the database file and then exits, without the user being able to read/write to the database itself.

This comes with some security risks, as its very important to make sure the code executing with escalated privileges can not be exploited. If it can be exploited, the attacker will gain higher privileges than the user and be able to execute commands that they shouldnt. For this reason, in addition to securing the code, we should also make sure that we escalate the privileges with the principle of least privilege (to limit the damage if the program is compromised).

2. Software vulnerabilities

(a) Describe the problem with the well-known gets() library call. Name another library call that is safe to use that accomplishes the same thing.

The gets() function does not allow us to set any limit on the ammount of input we receive. This causes what we call buffer overflows. Most commonly known are stack overflows, as the variable untrusted users write to often are stored as local variables on the stack. The stack also holds all other local variables which means that if a user can write to as much of the stack as they want, this allows them to alter parts of memory they shouldnt.

These targets can be variables that later will be used in conditional statements, or a very common target would be the return instruction of a called function. If this address is changed, the user can control the instruction pointer to call, unintended functions, or (knowing the libc version on the system) use it to fork a shell process on the server, getting the access of whoever was running the program.

This is why all programmers that wish to recieve input through the gets() library call should instead use fgets(). The fgets() call limits the ammount of user input on the stack to one less than the programmer specifies, and ends it with a null terminator to signal that the input string has ended. We still need to be careful to only allow as much input with fgets as we should, as allowing too much can cause overflows and unintended consequenses.

(b) Explain why a microkernel is statistically more secure than a monolithic kernel.

A monolithic kernel takes a lot of control, and prevents bad user code from compromising the kernel, however it does not prevent bad kernel code from causing damage. With a properly implemented microkernel, a lot of the code making the system function operates without inherent kernel privileges, meaning it can deal less damage. This means that considering all code can be buggy, a monolithic kernel has a larger attack surface, seeing that its is statistically more likely to contain exploitable bugs granting kernel privileges.