OBL4-OS

April 5, 2019

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 to gremlin.stud.iie.ntnu.no over SSH (Secure Shell). Working in groups is permitted, but submissions must be individual.

# **1 File systems**

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

**2. Name some examples of file metadata.**

# **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?

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

**(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).

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

**2. NTFS - Flexible tree with extents**

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

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

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

# **3 Security**

**1. Authentication**

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

You hash a password with a unique salt to make the password harder to crack. When storing passwords in a database, you don’t want to store the plain string, this is because everyone that has access to the database can see the user’s password. To avoid this, the password is hashed.

A “Hash” is a one-way function that generates a representation of the password.

When a user chooses a password, this password is stored as the generated hash in the database, rather than the actual string the user entered. When the password is run through a hashing function, it will always produce the same output.

When the user tries to log in with their email and password, the entered password is hashed again and then compared to what is stored in the database. If the two hashes are the same, the user has entered the correct password.

But hashing the user password on its own is not enough. It does not take very much computational power to generate a table of hashes of combinations of letters, numbers and symbols. Once you have this store of hashes, you can then compare the hash you want to crack and see if it matches. Once you find a match, you know the

password.

In order to make it more difficult to expose a hash, a salt is needed. Salting is where an extra bit of data is added to the password before hashing it. Salting is important because it adds a whole new level of required computational power in order to expose the hash. By adding a salt, you effectively render any lookup table useless.

**(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?**

**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.**

With the gets() library call the code can suffer from buffer overflow. A buffer overflow is where a program while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations. To avoid buffer overflow you can use fgets(). Fgets() makes sure that not more than MAX\_LIMIT characters are read (fgets(str, MAX\_LIMIT, stdin)).

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

* Separation of the drivers from the kernel (and each other) - a failing/malicious driver can't crash or even silently modify the kernel. The microkernels implementation of IPC by message passing instead of shared memory allows the kernel to sanitize the data transferred.
* Drivers being separate processes can run in a different protection ring than the microkernel and applications that do not require any hardware access at all.
* a failing driver can be restarted without the rest of the system being affected