Universität des Saarlandes
Lehrstuhl für Softwaretechnik
Prof. Andreas Zeller, Stephan Neuhaus
Design of Secure Software Systems
Exercise 2 - Solution

Total number of points obtainable: 33, plus 4 bonus points.

1 Password Aging

1.1 Two Scenarios to Foil Password Aging

- Append a number to your "base" password, and increment the number every time your password expires. For instance, *password1*, *password2*, *password3* etc.. [2 points]
- Keep only two (or three) passwords. Every time your password expires, switch to the next. When it expires again, switch back to the first. [2 points]

There are, of course, many other correct answers.

1.2 Closing the Loopholes

For loopholes identified in this sample solution:

- Apply string matching heuristics to compare the old and the new password. For example the new password must not contain more than n letters at the same positions. (Caution! Ideas like these need to compare passwords in plain-text, which is not a very 'secure' thing to do.) [2 points]
- Maintain a history of old passwords and ensure that the new password is not amongst the last n previous passwords. [2 points]

Since maximising security creates hassle for the users, and maximizing usability leaves the system (and its users) vulnerable to attacks, a tradeoff exists between security and usability. [4 points]

1.3 Threat Analysis

Password aging helps against brute-force attacks, if the time taken break the password is longer than the expiry time. It reduces the chances of damage caused by a compromised password (since the password may have expired by the time the attacker gets to use it). Also, in situations where someone gets your password and is interested only in monitoring your activities (such as reading your email), he/she will be able to do that for a limited time only. [2 points]

Whether the effort was worth it or not is a subjective question, and depends largely on the reasons you give. Amongst the different opinions expressed in the answers, one was that the more ?secure? you try to make the system, the more annoying it will become for the user, who may finally resort to writing down his password and defeat the whole purpose of enforcing such policies anyway. Another equally acceptable answer is that simple measures often do save the user from making password-related blunders, therefore such policies are better than having no security policy at all.

2 Password Storage

2.1 Password Entry

The password typed by the user will appear on the screen because the echo is not turned off. [2 points]

You can find how to do that on slide 12/45, lecture 3. [2 points]

2.2 *crypt*(3) Used Improperly

The first two characters of the password will appear as plain-text in the password file, and a given plain-text password will always have the same encrypted form. [2 points]

It is better to use a pseudo-random number, or even the time of day for the salt. The idea is that no two users should have the same salt, even if it is not truly random.

2.3 Plaintext Password in Memory

The contents of the memory may be transferred to disk as a result of a core dump, or when memory is swapped out to disk. [4 points]

2.4 Access to Password File

Non-root users would need read and write access to the password file/database (extremely dangerous!). [4 points]

To overcome this problem you can set the SUID bit for the program so that it runs as root. [2 points]

2.5 Plaintext Password Swapped Out

If the plaintext portion is swapped out to disk, it will remain there as long as it is not properly erased. [2 points]

Since normal users do not (or are not supposed to) have access to the swap file/partition, one would either need to have root access on the machine or physical access to the disk. Therefore the risk from such a threat is low. [1 point]