CIS-481: Introduction to Information Security

InfoSec Chapter Exercise #7

Team: 5
Participants: James Hoagland, Ethan Grimes, Holli Grubbs, Gregory Ellis

Logistics

A. Get together with other students on your assigned team in person and virtually.
B. Discuss and complete this assignment in a collaborative manner. Don't just assign different problems to each teammate as that defeats the purpose of team-based learning.
C. Choose a scribe to prepare a final document to submit via Blackboard for grading, changing the file name provided to denote the number of your assigned **Team**.

Problem 1
Consider the logical access control needs for joint software development teams using a typical Linux environment. Roles must include Developers (that can commit changes made in the code), Testers, and Code Reviewers. The technical access control mechanisms that you design must reflect these organizational roles. Your access control solution must:

1. Protect the software being developed from outsiders stealing it
2. Protect against unauthorized changes (including from internal actors)
3. Ensure that we can trace *who* made each change

Situation 1: A small team on a single machine *(5 points)*
Logical access controls for a small team on a single machine may not require the use of a version control system. The access control solution of setting up access control lists on that computer protects against unauthorized users from accessing the software being developed. Developers can use the access control user interface on the computer to grants access rights to a user or group. Permissions such as Read and Write can be specified based on the groups or users. Developers would be able to have Read and Write permissions. Testers and reviewers would only need Read access. With administrator rights you can trace who made changes by object auditing which tells successful and failed access to objects.

Situation 2: A medium-to-large team on a LAN [Hint: Use of a version control system like Subversion is highly recommended] *(10 points)*

For the Version Control System on a LAN, the medium-to-large team will feature software developers utilizing the Mercurial VCS. Mercurial is an excellent tool for software developers, as it is supported on many OS's such as Windows, FreeBSD, macOS, and in this case Linux. But why use a VCS?

The reason for using a VCS such as Mercurial is for Auditability, which is for identifying who made which change? More reasons for using a VCS are the ability to analyze when a given change was made, rolling back to a known-clean version of the codebase, and analyzing what patches have been applied to which versions of the system.

The structure of a VCS such as Mercurial contains a repository, which has a master copy that records all changes, versions, etc. Another vital part of the structure of a VCS are the working copies, where developers seek out a specific version form the repository in order to make changes and commit to those changes.

Situation 3: A large, distributed team, including outsourced contractors *(10 points)*

For a large, distributed team, including outsourced contractors will have software developers using the git VCS. Git is free and open source VCS that was created to handle large projects. It is available for macOS, Linux, and Windows. Git is one of the fastest VCSs on the market that is intended for software developers. It is distributed, has multiple backups, and works with any workflow. It protects data by making it impossible for users to change/alter data without changing the IDs of everything after it. With a commit ID, all data put into the repository will be safe. It also documents who inserted data as well. Testers and code reviewers can use the staging area git provides to review and format code.

[Inspired by https://www.cs.columbia.edu/~smb/classes/f09/l08.pdf - many thanks to Columbia University for providing under Creative Commons!]