|  |
| --- |
| http://www.wpi.edu/~vernescu/logo.png |
| Project 1: Password Wallet |
| CS 557: Software Secure Design and Analysis |
|  |
| **Tyler Carroll and James Silvia** |
| **9/25/2014** |

# Capabilities and Goals of the Adversary

We can expect the adversary to have several capabilities. Firstly we can expect that they will have access to the application and access to a program that will allow them to rapidly guess the master password of the user that grants access to the application. Secondly we can expect that since they have access to the machine that the program is running on, they will be able to search for and possibly find the file that holds the encrypted versions of the user’s passwords.

There are two main goals of the adversary. The first goal of the adversary is to obtain the passwords of the user in any way possible. The adversary will try to use programs to brute force attack the program and guess the user’s master password or they will find the encrypted version of the file and try to figure out the key that was used to encrypt the file. If they cannot access the user’s passwords then the second goal of the adversary will be to corrupt the user’s passwords in any way possible. They can achieve that by deleting, editing or moving the encrypted password file.

# System Goals

The intended functionality of the system is to provide a secure place to store the user’s passwords while providing the user with the option to auto-generate a password. If for some reason, the user does not want to auto-generate a random password, they can input their own password to store. The system will maintain association between the password description and password. The first time the user logs into the program, they are prompted to make a master password that they will use to access their passwords from now on. The program only supports one user at a time. After logging in the user has the ability to add a new password, look up a password that is already in the system or remove a password from the system.

The security goals we want to achieve here are the ability to store the passwords in a file safely that we know may be looked at by an untrusted party. We want to pick a key to encrypt the file that is not hard coded so each user has a different key. We also want to prevent an untrusted party from brute force attacking our system and guessing the user’s master password.

Passwords will never be displayed in cleartext unless the user has successfully logged in to the system with their master password and requested the proper name association. When a new password is generated, it is not printed to the screen, only encrypted and saved. Similarly, if a user decides to manually create their password via the CLI, this text will also be hidden.

# User Profile

There are three things that we expect the user to do in order to keep the system secure. Firstly we expect the user to run the program as a root user (or equivalent user level) so that no untrusted party will be able to run the application or read, write or remove the file that stores the passwords (only the user initiating the program can access the wallet). We expect the user to know to do that because the program provides an obvious banner describing the penalties for not following these directions. Secondly we expect the user to back up their encrypted password file to another secure location just in case an untrusted party corrupts it. We think the user will know how to do that because on a first time login when they create their master password and we create the password file, we display a banner message that informs the user to back up their password file periodically just in case an untrusted party gains root access on their machine and corrupts the password file. Lastly we expect the user to pick a strong master password and we expect this to happen because we have placed restrictions on the password that they can choose to ensure their password is strong.

# Security Mechanisms

There are several security mechanisms we have employed to protect the user’s password wallet. Firstly to ensure that the key is random for each user we decided to make the key we use to encrypt and decrypt the user’s passwords from their master password. We perform a one way hash function with a hard coded key on the master password, and then use the result to encrypt and decrypt each password in the wallet. We chose this method because even if the adversary obtains the source code and the hash function’s hard coded key, they would need the user’s master password in order to obtain the key used to decrypt the passwords. If they obtained the master password somehow to make the decryption key then it would not matter because they would already have gained access to the system.

Secondly to ensure that the adversary cannot perform a brute force attack on the system to gain the master password, we have put a limit to the number of attempts that the user can perform. The user can only perform five guesses for the master password; if they fail all five then we wipe the password file and exit the program. Yes this seems like an easy way for the adversary to corrupt the program but we inform the user to make a backup copy of their password file just in case so they can replace it if this occurs. The adversary will not be able to make a copy of the file and continually replace it to keep guessing passwords unless they have root access to the user’s system. We also make the brute force attack more difficult by putting restrictions on the type of password the user can enter as their master password.

Lastly we ensure the adversary cannot corrupt the password file by only giving permissions to root users and only allowing the user to run the program if they have root access.