|  |
| --- |
| http://www.wpi.edu/~vernescu/logo.png |
| Project 2: TLS Distributed Filesystem |
| CS 557: Software Secure Design and Analysis |
|  |
| **Tyler Carroll, James Silvia, Tom Strott** |
| **10/15/2014** |

# Capabilities and Goals of the Adversary

We expect the adversary to have a few capabilities. The adversary might be attacking the system from the outside, in which case they will be monitoring transmissions between each peer, hoping to impersonate one or the other and get some files. Alternatively, the adversary might be on a machine that can run this application. In that case, it is important to prevent them from successfully getting into the application. Finally, each peer stores files, and we need to make sure the adversary cannot get the contents of the files from the peer’s filesystem.

There are two main goals of the adversary. The first goal of the adversary is to obtain any files in any way possible. The adversary can attack the system in any of the three ways outlined above to achieve this. If they cannot access the files then the secondary goal of the adversary will be to corrupt the files in any way possible. They can succeed in that by deleting, editing or moving the encrypted files once the files are located.

# System Goals

The intended functionality of the system is to provide a secure system to store and retrieve files across peered connections. The files are modeled as numerical values, but with a few tweaks they could easily be strings or actual files. The user logs in to the system by typing in a valid username and password, and then is able to access other peers to store and retrieve files. Any peer can connect to any other peer provided both are logged in at the same time. The filesystem is shared, meaning that a peer stores files based on an index and any peer can access them (as long as it is stored on a peer – not you). The program supports four users at a time.

The security goals we want to achieve here are based on what we think the attacker will do. We cannot let an attacker impersonate a valid user and gain access to the system, so each user has their own password that is created on the first login. We implemented TLS to prevent an adversary from falsely impersonating a peer as well as using an application like Wireshark to see packet transmissions and reconstruct the files. Finally, when a peer stores a file, that file is encrypted to ensure that an attacker cannot see the contents of those files.

Files will never be displayed in plaintext unless the user has successfully logged in to the system with their password and requested to load or store a file.

# User Profile

There are three things that we expect the user to do in order to keep the system secure. First of all, 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 files created by the application. We expect the user to know to do that because the program provides a warning describing that an attacker may be able to corrupt your files if the user is not a root-level user. Secondly, if the user ignores our warning, we have three potential peers for a user to store files on. That way the user can back up each one of their files three times and an adversary needs to compromise each peer in order to corrupt the user’s data. We suggest that the user does this. Lastly we expect the user to pick a strong password and there are restrictions in place to prevent weak passwords.

# Security Mechanisms

There are several security mechanisms we have employed to protect the user’s distributed filesystem. 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 files from their password. We perform a one way hash function with a hard coded key on the password, and then use the result to encrypt and decrypt each file on the peer. 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 password in order to obtain the key used to decrypt the files. If they obtained the password somehow to make the decryption key then it would not matter because they would already have gained access to the system.

TLS is implemented to create a secure connection between the two peers, encrypting any transmission. We have our own certificate authority to ensure that an attacker cannot forge credentials and impersonate a peer. A peer cannot read its own data, so the only way to get a file is to request it from another peer with a valid certificate. Our certificates use 2048-bit RSA.

Each transaction is a discrete event, so as soon as a file is sent or received, the connection between the peers is terminated. This prevents an attacker from using an idle, connected machine to get files. If someone managed to connect to the peer with a certificate signed by our CA and attempt to interact with it, the peer will only respond to valid commands and give no response of any kind otherwise.

Lastly we ensure the adversary cannot corrupt the files as long as the user is a root user by only giving permissions to root users.