CS 1653 Project P5 Writeup

Brian Lester, bdl20@pitt.edu
Ryan Conley, rgc11@pitt.edu
Carmen Condeluci, crc73@pitt.edu

Overview

Threat Model

Here we characterize the behavior of the principles that may be present in our system.

- **Group Server**: The group server is entirely trusted. In this phase of the project, this means the group server will only issue tokens to properly authenticated clients and will properly enforce the constraints on group creation, deletion, and management specified in previous phases of the project. The group server is not assumed to share secrets with the file servers in the system.
- **File Server**: In this phase of the project, file servers will be assumed to be largely untrusted. In particular, file servers might leak files, attempt to steal user tokens, or modify files on disk. Note that this does not include file deletion, as such a guard would provide little benefit to the system.
- **Clients**: We will assume the clients are completely untrustworthy.
- **Attackers**: All communication within the system might be intercepted by an active that can insert, reorder, replay, or modify messages.
- **Server Environment**: Assume that the machines that the servers run on are largely untrusted. This means that the sensitive files which should remain known only to the server could be viewed by an unauthorized user by an oversight from the untrusted server admin.

Threat T8: Theft of Server State

This threat model is that a System Admin (or anyone that is using the machine that the servers are running on) could read or modify the GroupList, UserList, and/or the FileList that the various servers use to save state. This could be due to multiple reasons such as improperly configured servers saving their files in shared system directories or accounts being left logged-in and unlocked on unshared machines.

Threat T9: Theft of the server's private key

This threat is that since keys are currently trusted to be stored as plaintext files on the machine, that they may be stolen by another user or a system admin, as outlined in T8 where the server admin cannot be trusted to manage the secrecy of their local account, and by extension a server's private key. If this private key is stolen, an attacker could then impersonate a server.

Threat T10: File Modification

Due to the untrusted nature of the file server and the untrusted machine it is running on a file that is saved to disk could be modified.

Threat T11: Information Leakage via Group Naming

This threat can be distinctly divided into two subthreats. The first being the threat of allowing for reuse of a group name, whereby a user could incidentally or purposefully gain access to an old deleted group by using the same name.

This initial threat was partially addressed in a prior phase of the project with the disallowance of repeated group names, but this also is a source of information leakage. By providing information about group names that are in use, it compromises the assumed privacy of the system, where all group information should only be known to members of said group.

Threat T12: Denial of Service

Many users connecting to the server will create a flood of requests that utilize server state, leaving the server borderline unusable by legitimate users. This may also bring the server completely down or severely cripple it.

Attacks

Attack leveraged via Threat 8

An attacker could load up the GroupList, FileList and/or the UserList into their own server implementation (this is possible because the GroupList, FileList, and UserList are unencrypted and the implementation of the objects that underlying these lists is known due to transparency of mechanism) and start maliciously modify these files. The attacker could add a new account and then add that new user to each group. This is dangerous because they could access any sensitive files in the groups they add themselves too. They could even add themselves to the admin group. Modifying the FileList also allows them to grant themselves access to a group's files. While the files are all encrypted, the file names are still leaked. With access to these file lists and by impersonating a server, an attacker effectively gains free reign over the entire system.

Attack leveraged via Threat 9

"groupserverprivate.key" is unencrypted on the disk so if a user on that machine were to steal that file they would have the group servers private key. This could be used to create fake tokens. These fake tokens could then be signed with this key which would cause them to be accepted. "fileserverprivate.key" is unencrypted on the disk so if a user steals the private key for the file server they can impersonate the file server. This can lead to the clients uploading a sensitive file to the fake server.

Attack leveraged via Threat 10

Files on the disk for the file server could be modified. This modification would not be detected so any changes could junk the file or worse they could be changed and the changes would not be noticed which could lead to incorrect information. An attack via modification is more important to protect against than deletion because deletion will be known as soon as a file is requested, with nothing being returned. Deletion also serves no purpose as an attack other than messing with the users, which could also be done by just cutting the cord. Modification, on the other hand, could be used as an actual attack. Modification could also be carried out by swapping blocks of the real file with blocks from an older version of the file where the key has yet to be evolved. This can cause legible changes outside of small errors at block boundaries.

Attack leveraged via Threat 11

There are two manifestations of this attack: the first being a user choosing a group name that is already in use on the system. This causes an error to be returned and the group is not created. This leaks information in that the user trying to create a group knows the name of another group known on the system, which goes against the privacy of the system.

The second attack is the user choosing the name of a group that had previously been deleted. When logging in to a file server that this group used, this new group could access files that belonged to the old group. This threat is slightly mitigated due to the backwards secrecy provided by group keys, however information is still leaked in file names and file sizes.

Attack leveraged via Threat 12

A DoS attack on the system causes a build-up of state and excessive resource consumption on the server. This slows down the speed of real requests to the server. Using our own DoS and benchmarking programs (Dos.java and BenchMark.java respectively), we have shown that our server processed 1/40th of the speed during a simple DoS attack (27,000ms vs. 700ms). This increased slowdown is an attack on availability, which is an attack on a part of the CIA triad that we have not yet considered.

Countermeasures

Mechanism to Defend Against Threat 8

In order to defend against this threat, a mechanism must be developed to protect the existence of the server's state files (GroupList.bin, UserList.bin, FileList.bin) on disk, so that even if the state files are physically stolen, they cannot be used or modified by an attacker to gain information. This can be achieved through the use of the "starter's" (server administrator's) private key. On the execution of the server, the starter's private RSA key (which the starter will type in manually) is used to generate a confidentiality and integrity key that will be used to encrypt and verify the server's state files when it is loaded from disk. The two keys are $\mathbf{k_c}$, used for encryption, and $\mathbf{k_i}$ which is used for integrity checks. The keys are generated by hashing, using SHA-256, the starter's private RSA key concatenated with "confidentiality" and "integrity" respectively. After the server private key is successfully loaded and verified, it is used as normal throughout the system as outlined in our previous phases.

This mechanism is sufficient for defending against this threat. The server administrator (starter) is tasked with the responsibility of keeping his/her private RSA key (used to create $\mathbf{k_c}$ and $\mathbf{k_i}$ to protect the server private RSA key) safe from compromise. This means that the key used to protect the server state files is not present on the system, and will not be stolen in the result of a machine compromise of any verified server. It is acceptable to require the starter of a server to type in a full RSA key as both servers are assumed to have long uptimes, and this operation only needs to occur once on server startup. This administrator is also tasked with verifying the server's public key out-of-band with users, therefore he can be trusted to do this too.

Mechanism to Defend Against Threat 9

In order to defend against this threat, a mechanism must be developed to protect the existence of the server's private RSA key on disk, so that even if the key is physically stolen, it cannot be used by an attacker to host a fake server. This can be achieved through the use of the "starter's" (server administrator's) private key. On the execution of the server, the starter's private RSA key (which the starter will type in manually) is used to generate a confidentiality and integrity key that will be used to encrypt and verify the server's private RSA key when it is loaded from disk. The two keys are $\mathbf{k_c}$, used for encryption, and $\mathbf{k_i}$ which is used for integrity checks. The keys are generated by hashing, using SHA-256, the starter's private RSA key concatenated with "confidentiality" and "integrity" respectively. After the server private key is successfully loaded and verified, it is used as normal throughout the system as outlined in our previous phases.

This mechanism is sufficient for defending against this threat. The server administrator (starter) is tasked with the responsibility of keeping his/her private RSA key (used to create $\mathbf{k_c}$ and $\mathbf{k_i}$ to protect the server private RSA key) safe from compromise. This means that the key used to protect the server private key is not present on the system, and will not be stolen in the result of a machine compromise of any verified server. It is acceptable to require the starter of a server to type in a full RSA key as both servers are assumed to have long uptimes, and this operation only needs to occur once on server startup. This administrator is also tasked with verifying the server's public key out-of-band with users, therefore he can be trusted to do this too.

Mechanism to Defend Against Threat 10

When users upload a file to the file server, they create a confidentiality and integrity key from the evolving key that they are provided by the group server. The confidentiality key is used to encrypt the file, while the integrity key is used to key an HMAC based on SHA-256. This

HMAC is then uploaded to the file server along with the file.

This mechanism is sufficient to defend against the threat as the file server can no longer change the file without causing a mismatch in the compared HMACs. The HMAC is keyed with an integrity key that is secret from the file server, therefore the file server cannot recompute this HMAC.

Mechanism to Defend Against Threat 11

The mechanism to defend against this threat is to simply add a new field to a group object that identifies it with a unique identifier rather than a group name. This allows for any number of groups with any given alias to exist. This new identifier is then used with the group name (now the group alias) for all operations to identify specific groups. This identifier is formatted as the group alias concatenated with a randomly generated string.

This mechanism is sufficient for defending against the threat due to users no longer being able to create a new group that possesses the same unique identifier as a previously deleted group. A user could also now create any number of groups with any given alias, so they would never receive an error message that would indicate if a group with a given identifier already existed. The random string is generated using Java.Security.SecureRandom and is 128 bits long. This ensures that two groups will not end up with the same random string.

Mechanism to Defend Against Threat 12

The mechanism to defend this threat is the addition of computational puzzles to the connection process. When a client tries to connect to a group server, they will be given a hash of **n** bits along with the solution of the puzzle concatenated with a timestamp, encrypted with the server's public key. They must solve this puzzle and include the answer and the encrypted chunk with their login request.

This mechanism is sufficient to address this threat because very little state is used on the server-side while a adequate of computational resources are used on the client side. The encrypted answer is sent to the user (forcing them to keep track of the state) to minimize the resource disparity between the client and server. Due to the setting where this system will be deployed in (a business environment where all computers are pretty much the same and also decently powerful), a computational puzzle such as this is reasonable to have a client solve. While this sort of defense does not adequately stop an attack by large nation-states and motivated attackers, some defense is far greater than none.

Discussion

Conclude with a paragraph or two discussing your threat model and countermeasure mechanisms. How realistic is your threat model for a real-life file sharing system? Comment on the design process that your group followed. Did you discuss other ideas that didn’t pan out before settling on the above-documented approach? Did you design attacks that you were unable to mitigate, or that you think are impossible to mitigate? Use this space to show off your hard work!

Finally, spend a paragraph discussing the interplay between the countermeasures for your proposed attacks and your techniques for mitigating threats T1–T7 from Phases P3 and P4 of the project. Did you have to design your countermeasures in a particular way so as not to re-introduce these older threats?

If your group implemented the countermeasures or did any extra credit, please discuss these additional contributions in this section, as well.

Extra stuff

- DDoS, write a program that does this.
- Group name collisions that I discuss in the TODO.