Segurança Informática em Redes e Sistemas

Ransomware-resistant remote documents

Final Report

Grupo T01

Manuel Mascarenhas

90751

Miguel Levezinho

90756

Ricardo Fernandes

90775

1. Problem

The given scenario depends heavily on the reliable communication between a central server and a set of client machines, and thus suffers from the possibility of attacks to this channel, such as spoofing, replay, man-in-the-middle, and others.

Another worry to take into account is the threat of ransomware, since the main server will function as a storage for potentially shared files and documents, susceptible to malicious code. This code can propagate through the network and encrypt the files of users and servers, offering the key for decryption for a hefty amount, usually through the use of cryptocurrency such as Bitcoin.

Finally, there is also the possibility of a physical attack to the servers, where an intruder (attacker) might access the data servers and attempt to read or modify a document to which he has no permissions.

These are considered to be the three main points where security problems may arise and where security mechanisms will need to be implemented in order to guarantee a secure environment. Additionally, in this manner, not only is the risk of an attack minimized, but also the impact and damage of a successful one mitigated through the recovery of documents.

1.1. Requirements

- Authenticity, Integrity, Confidentiality and Freshness in communications between all parties;
- Client side file encryption for storage confidentiality;
- Detection and recovery from integrity and ransomware attacks to the shared documents;

1.2. Trust Assumptions

- We won't trust users if they don't have the correct permissions for a specified document or whose requests don't assure authenticity, integrity, confidentiality and freshness;
- We won't trust anyone with physical access to file servers in terms of both read and write access;
- We will trust that the servers won't become unavailable, due to malfunction or denial of service;
- We will trust that there will be no malicious physical access to the Logs Server;
- We will trust that there will be no malicious physical access to more than one file storing server at once. This includes the main File Server and the Backup Servers;

2. Proposed Solution

2.1. Overview

We will implement the file sharing system ourselves with Django for realizing the database (details in annex 2.) and application server-side. Django REST Framework will also be used for the communications between machines (details source in annex 3.). TLS will be configured in all these communications through the use of Apache2. NFS will be used in VPN to synchronize files between file servers, and dbbackup (a Django add-on) will be used to backup and restore files and DB's.

The solution will follow the diagram presented below (Fig.1) in terms of structure and communication between machines:

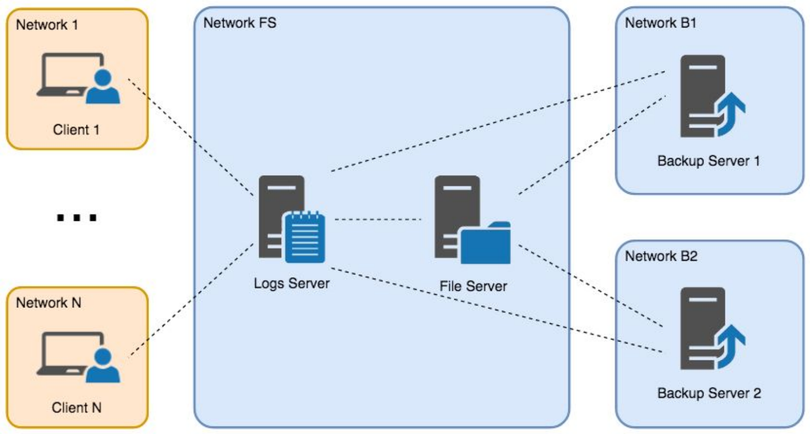

Fig. 1: System Structure

The File Server files will be stored encrypted. The encryption will be made by the Client Group that has access to the file, using a Group adaptation of the PGP protocol (GPGP). To use the system, clients will need to register themselves through the Logs Server, and can only communicate through it.

We will protect the system from physical integrity attacks and ransomware using a custom made protocol, which we will call Ransomware and Integrity Recovery (RIR), that will work in two phases:

- Detection, where illegal changes to files are detected using a log system stored in the Logs Server;
- Recovery, that will use redundancy by having the infrastructure setup in a 3-2 system way:
    - 3 servers, 2 of them for backups online, all of them in different private networks;

2.2. Deployment

Vagrant and VirtualBox will be used to deploy each machine in the system as a VM:

- The File Server machine will store and manage all files;
- The Logs Server machine will act as the entryway to the FS network. It will be split into:
    - A user registry system;
    - A log system for every file update;
- The Backup Server 1 and 2 will serve as backup servers for the files in the File Server machine;
- Two Client machines that will connect to the log server

Ansible will also be used to set up the environment, to install packages, distribute keys and deploy the code.

2.3. Secure channel(s) to configure

Communication between machines will be made using TLS (HTTPS). Communication paths are expressed in Fig.1 as dashed lines. As per the diagram, there will exist a certificate for all the server machines (public and private keys), and clients will have a generated RSA key pair as well. Since TLS will use AES, a symmetric key will be generated and used for each communication session.

2.4. Secure protocol(s) to develop

Both GPGP and RIR will be developed in Python. Each Client will have a pair of RSA keys. The public keys will be stored at the logs server upon registration (not at File Server since an attacker can access it physically) and be used by both protocols.

2.4.1. GPGP protocol

- Used to encrypt files, so they can be confidentially stored at the File Server;
- Used to decrypt files that a client has access to;

Protocol Specification:

The following two diagrams specify how the GPGP protocol will work at the client-side for encrypting and decrypting files, being N the number of clients that share file P:

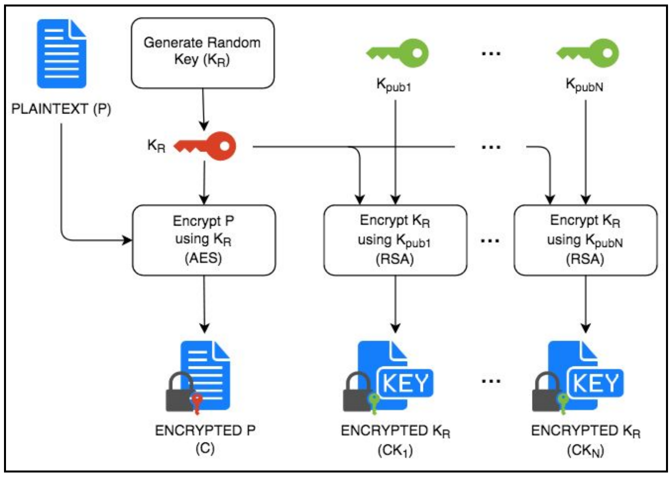
PLAINTEXT (P)
Generate Random Key (K_R)
K_{pub1}
K_{pubN}
K_R
Encrypt P using K_R (AES)
Encrypt K_R using K_{pub1} (RSA)
Encrypt K_R using K_{pubN} (RSA)
ENCRYPTED P (C)
ENCRYPTED K_R (CK_1)
ENCRYPTED K_R (CK_N)

Fig. 2.1: Client-side GPGP Encryption

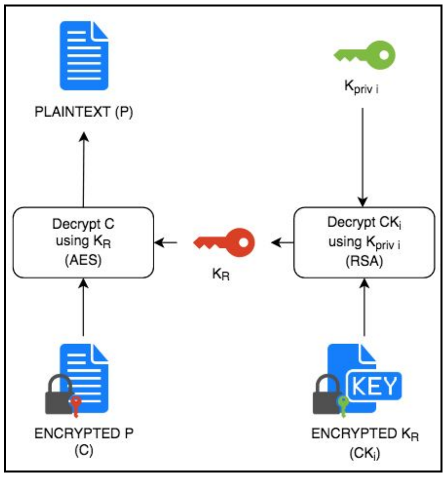

Fig. 2.2: Client-side GPGP Decryption

Besides C and $CK_{1\text{-}N}$, the client will also have to send the version number of the file, V, which is an integer that increments for each file update, and a signature, S, defined as $S = S_{RSA}(H_{SHA}(C + CK_{1\text{-}N} + V), K_{priv})$. Both will be stored in the Logs Server and be used for RIR.

RSA will use OAEP padding for encryption and decryption, and PSS for signing and signature verification. Both were chosen for being more robust padding schemes in comparison with PKCS1v1.5.

The AES cipher used to encrypt the file will be GCM. This cipher was chosen because it authenticates the data with an added tag. Although it adds redundancy since we will have S, it is useful to assure that the nonce used in the cipher (that will also be sent concatenated with each key $CK_{1\text{-}N}$) is authenticated.

2.4.2. RIR protocol

- Used to detect data integrity attacks to the files at the File Server, by using a log system;
- Used to recover from data integrity and ransomware attacks by use of redundancy, using backups (3-2 rule);
- Communications use TLS and NFS (using a VPN);

Protocol Specification: Detection Phase

The diagram that follows (Fig. 3) and its legend specifies how the RIR protocol will be able to verify storage integrity in the File Server and recover from corruption if necessary.

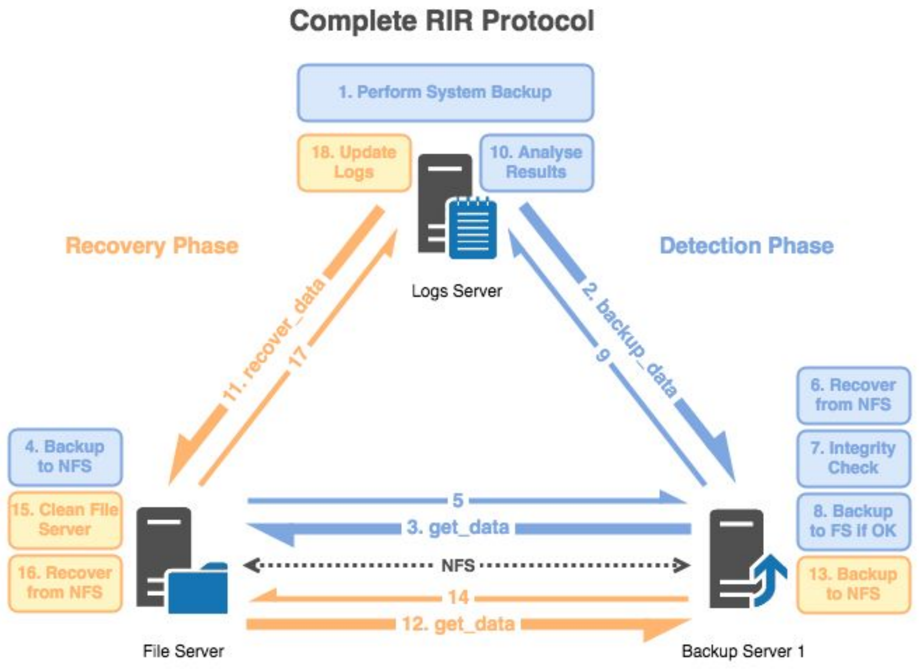

Fig. 3: RIR Protocol. In blue, the detection phase; In orange, the recovery phase (if needed). Colored arrows represent REST function calls, with ticker arrows representing requests, and slimer ones the responses. The dotted arrow represents an NFS connection that exists between File and Backup Servers (only one Backup Server in this example).

The Logs Server (LS), being the only secure server, will act as the controller during the protocol. A backup of the contents (media files and db state) of the Files Server (FS) is made when an admin user deems it so (1.). The RIR Detection Phase begins when the LS sends a request (2.) to all Backup Servers (BSs) to perform a backup of the FS.

This request contains information on the latest log of every file, which includes:

- file id, version (V), signature (S), user signing, its public key (K_{pub}), and file contributors. (i.)

The BSs will request the data from the FS (3.) and the FS will create a .dump representation of the db, and a .tar with all the files in the system. Both these files will be sent over NFS to the BSs (4.), and the FS response (5.) will contain information of of every file, which includes:

- file id, path, list of user keys, each with their key value (CK) and user id who owns it. (ii.)

The BSs will then restore the FS state in their environment using the .dump and .tar with (6) and will analyse the integrity of the FS in two steps (7.):

- Compare the information in (i.) and (ii.) to determine if tampering in the db happened. For example, if the number of keys to a file in FS is equal to the number of contributors reported in (i.) to that file;
- Confirm the integrity of files by verifying the signature sent in (i.): $V_{RSA}(S, H_{SHA}(C + CK_{1\text{-}N} + V), K_{pub})$;

If all verifications passed, BSs will backup the validated data (8.) to the File System. The verification status is then sent to the LS (9.), which will decide if recovery is necessary (10.). Three scenarios may happen:

- Both BSs report OK: The Recovery Phase is not started;
- Both BSs report NOK: The Recovery Phase is started;
- One BS reports OK and the other NOK: The Recovery Phase is not started (NOK-BS is compromised);

Protocol Specification: Recovery Phase

The LS will signal the FS to recover from a chosen BS (11.). To start, the FS will retrieve the latest backup from the BS in a similar but inverse way to steps 3. 4. 5. (12. 13. 14.). After cleaning the File System and flushing the db (15.), the FS will recover its state with the backup data (16.) and report to LS (17.). Finally, the LS has to also rollback its logs to the recovered state. To do this, it uses a timestamp each time a backup starts and stores it if it ends successfully. On rollback, it checks the latest backup timestamp and deletes all subsequent logs (18.).

Results

3.1. Achieved Version

The achieved project corresponds with the Proposed Advanced Version, which means it implements:

- Setup infrastructure and add secure communication between machines, configuring TLS for all communications.
- RIR protocol for integrity and ransomware detection and recovery.
- GPGP protocol at client side to enable file sharing between clients.

Additionally, the solution also includes 2 enhancements related to the RIR protocol:

- On uploads and updates to files, the LS will verify the sent signature against the sent data to prevent storing corrupted information, minimizing the overhead of the RIR recovery phase later on.
- At the client side, if GPGP decryption fails, the failure is reported to the LS, which will verify the claim (getting the data and checking with the signature) and start the RIR recovery phase if necessary.

3.2. Conclusions and Solution Analysis

The implemented solution represents what was proposed with added value. Besides what was already reported above, the solution also includes strengths in other small ways, such as password hashing (with added salt) for registered users, token authentication of users and user groups when calling resources, and permission checks when accessing files. Requests to the LS are also heavily validated for errors. The system is also scalable in terms of number of backup servers.

In terms of improvements, there are two that stood out after we concluded the project:

1. GPGP is not scalable to a large group of users, since each user will have a key and hashes need to include all of them, which can become highly inefficient;
2. The implemented recovery is not very efficient in that it can only backup/recover the whole system, and not individual files and resources, due to limitations in the used dbbackup library.

The proposed solutions for these problems would be:

1. Implement a group Diffie-Hellman on top of PGP, so that only one key is used per user group (details of some extra design work can be found in annex 4.);
2. This would require refactoring of the RIR protocol to work more closely to what, for example, rsync does.

Annexes

1. References

Django
Django REST Framework
Django dbbackup
Django sendfile
PGP
NFS
Data Integrity in storage
3-2-1 Backup Strategy for ransomware
OpenVPN

2. Database Structure

The following diagram (Fig. 4) represents the database structure used in the system:

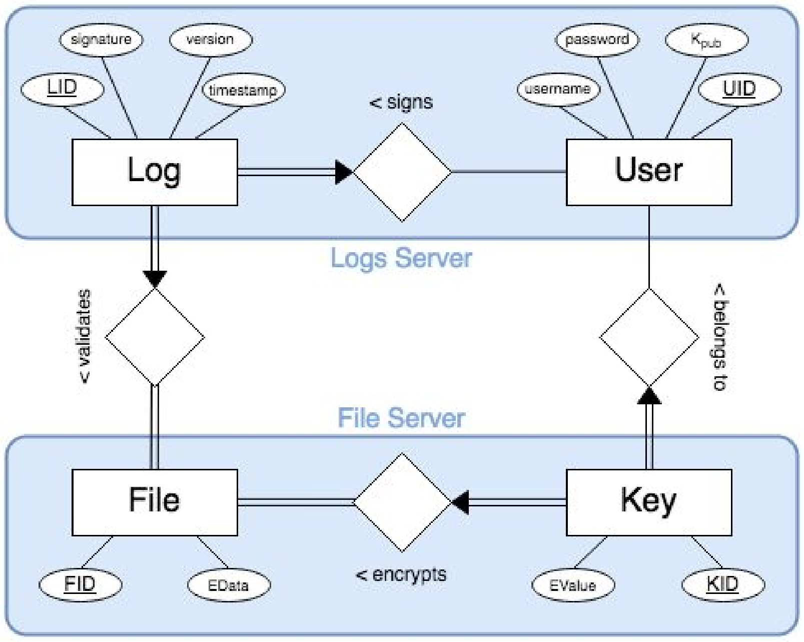

Fig. 4: EA diagram of the distributed database used in the system

The Backup Servers have an exact copy of the Files Server database model. The file data is not physically stored in the database for memory consumption reasons, the db stores only a reference to the file location. The Logs Server has an additional table, Backup, which is used to manage log deletion when RIR recovers files.

3. API REST calls

The definitions and behaviors for all the REST calls in the system can be found in the annexed file docs/API.md

4. Group Diffie-Hellman

This solution was partially designed as extra work for the project. The basic idea is to have clients that share a file also share the key that encrypts/decrypts that file.

For this, a client would first need to create a user group if he wanted to upload a shared file. He would need to send to the LS a generator (g) and prime modulus (p), which are the base stones for Diffie-Hellman, and define an order of communication in the client group. Each client would then grab these values from the LS, and start calculating intermediate values of the key.

Supposing client A, B and C (with order A->B->C) want to share a key, each would calculate $k_i = g^x mod\, p$, where $x = a, b, c$ are private values calculated at each client that stay private, and send it to the LS, in plaintext.

After that, a second round of exponentiation would begin, with each client calculating $k_j = (k_i)^x mod\, p$, where k_i would be the value calculated previously by the client next in order (for example, A would retrieve the k_i sent by B).

Finally, the third and final round would output the shared key, as $K = (k_j)^x mod\, p$, where k_j would be the value calculated previously by the client next in order.

The number of exponentiation rounds would be equal to the number of clients in the group.