r2-store: A ransomware-resistant document store

Network and Computer Security

Alameda Group 41

André BREDA 89409 

Baltasar DINIS 89416 

Wilson PEREIRA 89561 

1 Problem

We wish to protect a document storage system, which backs an application where clients collaborate on documents, from ransomware attacks on the servers. We need to tolerate situations where an attacker can access the persistent storage backing this system, and replace it either with an outdated version or an encrypted version.

1.1 Requirements

The solution should:

- Tolerate a subset of servers with tampered persistent storage;
- Allow for deletions from the file (after being agreed by a sufficient subset of the collaborators);
- Tolerate a subset of malicious collaborators, which may try to delete the file or corrupt it;
- Allow for concurrent collaboration of documents;
- Provide confidentiality of the documents: it should be impossible for the servers, other clients or a malicious attackers to see the cleartext contents of the file;
- Provide integrity of the documents: it should be impossible for the servers, other clients or a malicious attackers to change the content of the file;
- Allow for membership changes: if a client is added, it can see the full history of the file; if a client is removed, it should not be able to see new modifications.
- Provide non-repudiability of specific document changes;

1.2 Trust Assumptions

- We assume the server code (written by us) to be correct.
- We assume that the persistent storage backing the server can be changed by a malicious administrator.
- Unauthenticated clients are fully untrusted by the servers.
- Authenticated clients are not assumed to follow the protocol correctly.
- Servers are fully untrusted by clients for confidentiality. For data integrity, clients do trust a server quorum.
- Clients partially trust their collaborators: it should be possible for a majority of clients to rollback changes performed by a malicious client.
- Collaborators fully trust the owner of the file to change the membership of the file.
- There is a trusted administrator which remotely provisions the servers with their keys for secure communication with clients/other servers.
- Clients and Servers trust the CA for certification of server's keys for the effects of having secure channels (in client-server and server-server communication).
- Everyone trusts the CA for certification of client's keys for non-repudiation of other's changes.

2 Solution

2.1 Overview

We partition our system in 3 compontents (showcased in Figure 1), plus a Certification Authority (**CA**):

- The clients, which may or may not collaborate in a document;
- The servers, which run a fault tolerant consensus protocol to provide the document service;

- The persistent storage, which backs each server.

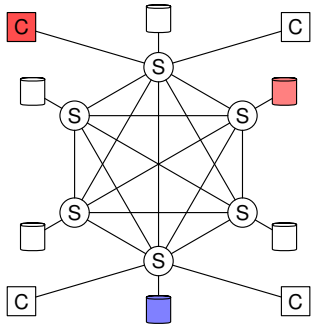

Figure 1: System overview. Storage in blue has been rolled back. Storage in red has been victim of a ransomware attack. Client in red is malicious.

2.2 Deployment

Each server and each client will be deployed on a separate Docker instance. Servers can communicate with each other over the internet or a (redundant) isolated network. A client can connect to any server.

2.3 Secure Channels

All communication will be protected using TLS. For simplicity it is assumed that the server has access to its certificates and the certificates (and location) of its peers.

Clients trust on servers is rooted on a signed certificate from a trusted Certificate Authority. They establish a connection with a server and verify the certificates.

The TLS library used will be `rustls`[3].

2.4 Secure Protocols

The protocol is based on the notion of applying diffs. Clients, when editing, will periodically send the diff to a server which will run a consensus protocol to propagate the diff to all servers. This way, a total order of diffs is established, making it possible to assign versions to the document.

Clients have asymmetric keys and are identified by their public keys. Their public key is stored by the CA, associated to their identity, for non-repudiability.

Diffs are signed by the author, making it possible to rollback a diff. However, the server only rollbacks a particular diff if it receives the requests from a majority of collaborators. This protects against client-issued ransomware: if a client decides, in a diff, to encrypt all the data, the rest of the clients can simply rollback the diff.

Each document has a *document key*. This key is only known to the collaborators and is used to encrypt diffs. This way, we achieve confidentiality of the documents. Integrity is achieved by employing the consensus protocol (which ensures that the diffs are totally ordered, and as such the document cannot be tampered by slightly reordering concurrent diffs) and by using an authenticated encryption scheme (namely GCM) with the *document key*. Note, however, that diffs are still signed with the author's public key for non-repudiation purposes.

The document key is chosen by the owner upon creation. To add collaborators, the owner simply encrypts the document key with the collaborator's public key and sends it to the servers. To remove a collaborator, the owner has to generate a new document key, decrypt all the diffs with the old key and re-encrypt all the diffs with the new key. This way, the purged collaborator cannot make or see new modifications.

It is possible to squash the document, by replacing the diff chain with a single diff. This requires approval from a majority of collaborators. This effectively makes it possible to remove something from the document permanentely, if the majority agrees. We believe this is a desirable feature, and it is the main reason we chose this solution versus, for instance, a blockchain based

Table 1: API provided by the servers

Function	Arguments	Result/Return
create	document name, ciphered document keys	creates the document
add_users	document name, ciphered document keys	adds collaborators
reset_users	document name, list of signed encrypted diffs, ciphered document keys	resets the collaborators
add	document name, signed encrypted diff	new version hash
get	document name	list of diffs and version hash
squash	document name, signed squashed document, version hash	registers a positive ack to squashing the document
rollback	document name, version hash	registers a positive ack to rollbacking a particular version

approach: if a collaborator uploads illegal/undesired content, it can be deleted safely.

Squashing is of course destructive: it makes it impossible to retrieve from the server the history of changes before the point in time when it happens. Should a client want to keep a log of changes for non-repudiation, they must be the one to do it. The space savings offered by squashing are fundamentally incompatible with author attribution to individual changes.

We consider that $\lfloor \frac{n-1}{3} \rfloor$ servers can behave in a Byzantine fashion, by having had their persistent storage tampered with.

We refer the reader to [1] for a description of a Byzantine consensus protocol. In terms of the state machine (ie: the API exposed by the servers) it is described in Table 1.

Note that in the case of the squash and rollback commands, any client can issue the first request and the server will in its turn ask the other clients. The clients may then either reject the request or reply positively, making the squash and rollback commands.

The implementation will be made in Rust[2]. There will be an application (the server), a client library and a sample client application for demonstrating the system.

3 Implementation Plan

3.1 Versions

Basic Version

- Implement document creation: assume pre shared key;
- Implement add/get interface for servers: confidentiality and integrity;
- Implement diff application on the clients;

Expected conclusion: November 20th.

Intermediate Version

- Implement document creation: creation and distribution of the key;
- Implement squash: remove illegal/unwanted content, performance boos;
- Implement rollback: protect against client-issued ransomware;

Expected conclusion: November 27th.

Advanced Version

- Implement change non-repudiation (requires implementing CA);

	AB	BSD	WP
Nov 20th	client diffs (40h)	consensus protocol (40h)	add/get + doc creation (40h)
Nov 25th	rollback (15h)	squash (16h)	document creation (18h)
Nov 30th	report (8h)	document membership changes (7h)	demonstration (5h)

- Implement document membership changes (grants are trivial, revokes not so much);

Expected conclusion: December 4th.

References

[1] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance". In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation*. OSDI '99. New Orleans, Louisiana, USA: USENIX Association, 1999, pp. 173–186. ISBN: 1880446391.

[2] *Rust: A language empowering everyone to build reliable and efficient software*. URL: rust-lang.org.

[3] *Rustls - a modern TLS library*. URL: https://docs.rs/rustls/0.18.1/rustls/index.html.