TRACELESS

Nikhil Buduma, Pratheek Nagaraj
nkbuduma@mit.edu, pnagaraj@mit.edu

May 8, 2015

6.824 Project Report

1 Introduction

Messaging is a popular form of interpersonal communication. Most modern communication systems have clients communicate through a server. The concern arises when the server is curious and can thus peek at messages. To solve this concern, end to end encryption may be used. Though this isn't enough since information is leaked about users communicating with specific users. To our knowledge, Traceless is the first system to serve as a oblivious messaging platform.

2 Overview

Traceless is a cryptographically secure and oblivious messaging system. In this paper we examine the design of Traceless and for brevity exclude any performance analysis. Section 3 discusses the design goals of the system and assumptions. Section 4 explores the cryptographic considerations in Traceless. Section 5 examines the distributed aspect of the system. Section 6 discusses some aspects of the implemented system. Section 7 the concludes.

3 Design Goals and Assumptions

This section explores some of the main design goals involved in building Traceless as well as discusses some assumptions that are made in order to achieve these goals.

3.1 Design Goals

Obliviousness - Our design goal is to ensure that the server is provably incapable of determining whether two individuals ever had a conversation on the service (using a computational secrecy model). In particular, we claim that our design ensures that 1) Any message pushed to the server cannot be traced back to a user and 2) any pull request for a message stored on the server also cannot be traced back to a user.

Responsiveness - To reduce server load, we divide server state into multiple non-overlapping shards. Because users only have to communicate with the shard containing their respective information (determined by the message slot they want to pull/push from), we can multiplex responses proportional to the number of shards. Locations of these shards are managed a master server.

Fault Tolerance - To prevent the system from crashing when a single machine, each shard is replicated (total of two copies - primary and backup). The master server manages which of these shards is the primary and which of them is the backup (state is synchronized by a two-phase commit). This means that even if a primary server for a shard fails, the secondary shard can take over for the primary.

3.2 Assumptions

1. Computational Security - We make assumptions about what kinds of computation a machine can reasonably achieve. We make no claims of perfect secrecy.

2. IP Spoofing - We assume that actors are able to (and are responsible for) keeping their IP hidden. This is a realistic assumption that is made in the unlinkable serial transactions

4 Cryptographic Design

Traceless is designed as a client server model. Clients wish to communicate to other clients and the server is used as a means for passing such messages around. The system is dependent on unlinkable serial transactions (UST) [1] as well as the RSA cryptosystem. This section of the paper explores the different aspects associated with the cryptography of the system briefly.

4.1 Subscription

Clients ping the master server in order to join the service. If successful, the clients then generate a RSA public-private key pair for encyrption and a RSA public-private key pair for signing. The clients initialize a UST transaction by sending a blinded nonce to the server along with a username and the RSA public encyption and signing keys. Existing clients discover newly added clients by regularly pinging the master server and downloading the new client's username and public keys to a local copy.

4.2 Initializing Conversations

A new conversation is initialized by a client selecting an existing user to communicate with. The client side generates a read and write slot for the communication. The client then generates a message consisting of its username, the recipient's username, the read slot and write slot. The client then signs the message and encrypts the combination of the message and the signature. This client then sends these items as a new conversation object to the server. Other clients then regularly ping the master server to learn about new conversation objects and downloads any newly added ones. They first verify that the signing is valid and then attempt to decrypt the message. If successful they recover the sender and trivially that they are the recipient as well as the slot to read and write, noting that these slots are reversed for the recipient.

4.3 Message Push and Pull

If a conversation has successfully been established then either party can send a message to the other individual. Without loss of generality, let client A send a message to client B. Then client A will first select a slot for the next write. Client A then sends a message with the text it wishes to send along with the next write block. Client A then signs this message and encrypts the two sets using client B's public key. Then the client A contacts the corresponding shard server responsible for the slot and pushes this message object. Client B, regularly pings the shard at the first read slot that it had from the initializing conversation. The shard responds successfully and then client B can decrypt the message, verify the signer, update the next read block, and read the text. Note that all the transactions with the server involved UST mechanisms so that the server cannot identify the user but simply that the user is a valid and authenticated user in the system.

4.4 Other Cryptographic Mechanisms

There is the ability to reserve blocks and delete blocks however this feature is not discussed in detail here. The implementation excludes these features at the moment for simplicity and performance considerations.

5 Distributed Systems Discussion

Traceless implements some key distributed systems features on the client and server side. Each side is discussed below.

5.1 Client Side

The main distributed systems feature set is the retry function in case a request is dropped or is unsuccessful. The client uses a timeout based mechanism to detect when a packet is lost. Specifically, the code uses a unreliable parameter to simulate whether a packet has been dropped going to the server or on the way back. The client locks on key objects so that with multiple threads spinning there is no inconsistent state in the system.

In addition, the client side must be responsive to the server side implementation of sharding. Specifically, the client uses a thread to communicate with the server in order to update its view on which servers are responsible for which shard.

5.2 Server Side

The server side implements the majority of the distributed systems features. Namely, implementations of master-slave models, sharding of key value pairs, and primary-backup reliability.

5.2.1 Master Slave Model

The use of a master-slave model allows the system to maintain organization and privilege levels. The master is responsible for the user table, conversation table, as well as maintaining the shard organiztion. Specifically, with regard to shard organization, the master is responsible for assignment of shards as well as upgrading backups and changing to new views appropriately. The master communicates with the client as well as other slaves to make sure that their views are updated. Note that the client only gets a partial view in that they cannot contact backup shards. Further, the master is responsible for signing of blinded nonces including the initial of the slaves.

5.2.2 Sharding and Availability

The use of sharding is a key features of the system. Since this is a messaging platform availability is a key concern. In order to make sure that clients and push and pull in a timely manner, we implement sharding of the server message table so that slots are distributed across slave servers. Specifically, we shard a contiguous range of slot ids and assign to a slave server. The clients can then communicate directly with the slave server rather than going through the master. This allows for high availability of the system improving the overall performance of message pushes and pulls.

5.2.3 Primary Backup and Fault-Tolerance

The system uses a complete replication model as part of fault-tolerance. Specifically, if sufficient servers exist then a shard will both have a slave serving as the primary and the backup for the slot range it is assigned. When a message object is sent to the primary from the client, the primary will first send it to the backup, if one is assigned, to insert, and then insert into its own message table. In the case of the backup failing, the system proceeds as normal and assigns a new slave to serve as the backup if available. In the case that the primary fails then the master will designate the backup to be the successor primary for the shard.

6 Discussion

This section explores some of the security, performance, and limitations of the system. It concludes with possible future work. Figure 1 is a screenshot of Traceless.

6.1 Security

The server cannot discern the client from the requests as seen by the cryptographic encryption of the entire message data. With IP spoofing the clients would be relatively immune to discovery by the server. Some concerns are timing attacks which the solution might be to reserve blocks in a random or batched manner instead of when a message needs to be sent.

6.2 Performance

The system is perfomant in the small scale testing we provided. The use of threads allow the system to rapidly perform any command argument provided by a user while also updating user tables, conversation tables, message pulls, and server views. The parameters for waiting are simply relative weights on which threads should progress faster than others. The authors believe that we can reduce the latency and throughput of the system by both reducing the waiting time as well as optimizing the thread performance.

6.3 Limitations

Some limitations of the system include a possible master server bottleneck. In order to resolve this issue we consider splitting off some of the functions of the master into other slaves, such as the user table and conversation table. In addition, we could provide a read-only version of certain parts of the master.

Another limitation of the system is in the case of a complete shard partition or failure wherein both the primary and backup fail nearly simultaneously - or at least before the master can respond. In this case, Traceless is unable to recover from this state. The authors believe that this is more unlikely and suggest alternative sharding mechanisms such as partial sharding or distributing among more backups.

Figure 1: An image depicting Traceless with three clients.

6.4 Future Work

Some future work includes developing a better primary backup model so that slave count is reduced. At the moment a complete replication provides great write throughput but introducing a more distributed sharding mechanism will reduce the hardware cost of the system and serve as a smarter replication scheme.

In addition, the authors hope to add the reservation and deletion aspects into the cryptographic system. This would allow for better memory management. The theoretical derivation for cryptographic security is not presented here for brevity.

For feature set, we would like to explore the possibility of group conversations such that multiple users can communicate with one another simultaneously.

For further failure tolerance, we hope to implement a recovery system for clients such that they can retrieve their previous state from other clients and the server while not compromising the security of the system. The solution to this recovery mechanism is not presented here for brevity.

7 Conclusion

Traceless provides a mechanism for oblivious and secure messaging. The system is distributed in a manner that achieves high availability while ensuring fault tolerance.

8 Acknowledgements

The authors would like to thank Prof. Morris for his instruction and thoughts on the project. In addition, the authors thank Steven Allen for valuable discussions and critique on the design choices. Finally, the authors thank Mayuri Sridhar for providing much of the thought that went into the system and cryptographic client work; she is a co-author of the system.

9 Repository

Traceless can be downloaded from the git repository provided below.

`https://github.com/pratheeknagaraj/traceless`

Several packages must be downloaded and configurations must be made so that the distributed system can work; please see the README on proper installation.

References

[1] Stuart G. Stubblebine , Paul F. Syverson , David M. Goldschlag, Unlinkable serial transactions: protocols and applications, ACM Transactions on Information and System Security (TISSEC), v.2 n.4, p.354-389, Nov. 1999