Distributed Systems
Project 2 - Report

Team name - Yew
Alexandra Fritzen (afritzen 866403)
afritzen@student.unimelb.edu.au
Annie Zhou (azhou 356076)
a.zhou@student.unimelb.edu.au
Borce Koteski (bkoteski 696567)
bkoteski@student.unimelb.edu.au
Hugh Edwards (hughe 584183)
hughe@student.unimelb.edu.au

1 Introduction

This report discusses the second iteration of EZShare, a distributed file sharing service written in java. The implementation of two features; secure connections and subscription, will be discussed along with tradeoffs and suggested improvements.

2 Security

The optional -secure flag controls the use of Secure Socket Layer (SSL) Sockets. Servers maintain a separate list of secure servers to ensure that the relay feature does not relay insecure requests on behalf of secure commands, and vice versa.

2.1 Benefits

The three primary benefits are integrity, authentication, and confidentiality.

Client and server authentication is achieved with certificates, signed by a trusted authority. Assuming the authority is not compromised, SSL guarantees that clients and servers are communicating with their intended recipients. This prevents certain classes of 'man in the middle' attacks.

Confidentiality is achieved with encryption. All messages between SSL Sockets are encrypted and are thus only readable by their intended recipient. Hence, clients/server interaction is confidential.

Integrity is maintained with message digests; which are hashes attached to each message. Receivers compute a hash of messages and compare the result with the digest to verify their integrity. This prevents message tampering.

2.2 Channel/User-Specific Issues

Whilst these secure properties hold for individual messages, they may not hold for the system as a whole due to the vulnerability of the channel/user system. Secondly, the channel/user system also limits user interaction.

The security provided by SSL sockets is undermined by the insecurity of the channel/user system. Channel and user names have no complexity requirement and thus may be easily 'guessable'. This is compounded by the unlimited incorrect guesses available to a host without being 'locked out', and a lack of expiry time on these secrets. Hence an attacker with sufficient resources and time could crack channel/user pairs and maliciously access confidential resources, or impersonate a user.

This insecurity can be alleviated with more stringent naming requirements, 'lockouts', and expiry times. The 'guessability' of channels and users can be reduced by requiring a level of naming complexity, such as a minimum of eight characters with a diversity of symbols. Secondly, hosts should be 'locked out' (their commands ignored) after a number of consecutive incorrect channel/user guesses. Thirdly, users or channels may expire after a certain time to reduce the time available for them to be cracked. These three measures would reduce the capacity of an attacker to 'brute force' secrets by making guessing more difficult, and reducing the number of guesses available (Ma et al. 2010: 586).

The channel/user system also limits usability. Currently, a channel/owner combination must be known for a resource to be queried. Hence for any channel other than the public channel, resources are effectively hidden from users. Whilst providing a degree of security, this limits user interaction with resources. A user might want resources to be available to a group of people; currently this can only be achieved by exchanging user/channel information outside of the application. Users might also want resources to be publically available, but still organised in channels (the public channel may become overcrowded with resources as the system scales).

Usability may be improved by the implementation of passwords, allowing users and channel information to be made available publically, and with the addition of a 'request' system. If accessing resources required a password, then channel and user information could be made public (optionally at the user's discretion). To query or access resources on a channel would optionally require a password, which a user could request from the channel owner. This would allow users to browse available channels and users, and request access to password-protected channels. Another usability measure could be to implement different levels of resource permissions; querying, fetching, modifying, etc. With these improvements, file sharing is encouraged between users without compromising security.

2.3 General Security Issues

Independent of the channel/user system, the security implementation has two other primary weaknesses; malicious file sharing, and Distributed Denial of Service attacks.

The system has no safeguards against the sharing of malicious files. Users may be vulnerable to inadvertently downloading harmful mobile code, especially in the public channel. This risk could be reduced with server-side virus scanning.

Secondly, whilst the -intervallimit flag provides some protection against Denial of Service (DOS) attacks, the server is still vulnerable to Distributed DOS attacks. Here an attacker floods a channel with requests from multiple hosts to consume server threads and prevent access by other users (Lau et al. 2000). Preventing these attacks, however, is complex and beyond the scope of this report (Lau et al. 2000).

3 Subscribe Command

The subscribe command allows a client to be alerted to resources matching a template over a period of time.

3.1 Current Approach

Subscriptions are connection-oriented. With the -relay flag, the server will make a connection with all servers in its list, and forward results to the client. This approach has two key advantages. Firstly, it is immediately consistent. Secondly, TCP connections are reliable; all messages will be delivered so long as the connection persists.

3.1.1 Drawbacks

This approach is limited in terms of fault tolerance and scalability.

The implementation is not fault-tolerant. If the server to which the client subscribes fails, then the subscribe fails completely, and no relayed results will arrive at the client.

Scalability is also limited, due to a large number of connections, and in bottlenecks. For every single active relay-subscribe request, a server maintains a connection with every other server in its list. Hence if there are n servers and k active relay subscriptions, $k*n$ subscription-related connections exist. A number of these connections may be redundant; since two servers maintain a separate connection between them for every single subscription.

Connections themselves also have an overhead in their establishment and in acknowledgments. Therefore, connections associated with relay subscriptions present a significant overhead.

Finally, bottlenecks are introduced by the 'centralised' approach, whereby all traffic associated with a given subscription is routed through one server before it arrives at the client. This may be exacerbated if many clients send subscription requests to the same server. Servers may rapidly run out of threads as the number of subscriptions increases. Hence scalability is limited by connection overhead and bottlenecks.

3.2 Alternative Approach

An improved approach may utilise connectionless and decentralised routing to improve fault tolerance and scalability, potentially at the cost of introducing eventual consistency.

Fault tolerance may be improved by decentralising the relay system. Instead of all servers communicating with a client through the originating server, servers may instead communicate directly with the client. If the originating server were to fail, these servers may continue to provide a subscription service, without relaying their messages through the failed server.

A decentralised implementation may also reduce bottlenecks. Since relayed messages can be sent directly to the client, they no longer need to pass through the originating server. This reduces the load on the originating server and makes the system more scalable.

Thirdly, utilising connectionless protocols, such as the User Datagram Protocol (UDP) and Internet Protocol (IP) multicast, may reduce overhead and improve scalability. Implementing subscriptions over UDP reduces overhead by eliminating acknowledgement. Further, IP multicast allows a server to send a single message to multiple recipients. Since for a given subscription, only one message needs to be sent every time a subscription is updated or a relevant resource found, the server load is greatly reduced.

Depending upon load, servers could also 'piggyback' messages; that is, multicasting a single message containing multiple subscription requests and resource hits every t seconds. This would reduce the total number of individual subscription-related messages sent by n servers to a worst-case of n, within a time period t. This represents a significant improvement in overhead.

3.2.1 Drawbacks

This alternative approach is more scalable, however, may only be eventually consistent. If messages are 'piggybacked', then clients may not be notified of all relevant resources to a subscription immediately.

Moreover, UDP is unreliable. If new subscriptions or resource hits are sent only once, then delivery failures may result in clients missing certain resources altogether. The importance of reliability depends upon how critical correctness is to users. If EZShare is to be used for sharing images of cats, eventual consistency or missed resources can be tolerated. This is less likely if EZShare is used for sharing important company data.

If reliability and consistency are considered vital, they may be improved at the cost of performance. Acknowledgements could be built into the application, which would increase the number of messages sent, but guarantee eventual consistency. Further, 'piggybacking' could be abandoned, and any new subscriptions or resources could be immediately sent. This would again increase overhead in the total number of messages sent, but ensure that the system remains more immediately consistent.

In summary, a connectionless and decentralised implementation may improve the scalability and fault tolerance of EZShare, however, certain tradeoffs need to be made regarding consistency.

4 Conclusion

This report has discussed the strengths and weaknesses of our implementation of security and subscription for EZShare, and suggested improvements.

5 References

Bailis, P & Ghodsi, A 2013, 'Eventual consistency today: Limits, extensions, and beyond', *ACM Queue*, vol. 11, no. 3.

Cavage, M 2013, 'There's no getting around it: You are building a distributed system', *Communications of the ACM,* vol. 56, no. 6, pp. 63-70.

Lau, F, Rubin, SH, Smith, MH, Trajkovic, L 2000, 'Distributed denial of service attacks', *Systems, Man, and Cybernetics*, vol. 3.

Ma, W, Campbell, J, Tran, D & Kleeman, D 2010, 'Password entropy and password quality', *Network and System Security,* pp.583-7.

Meeting minutes

Date: May 10th 2017, 8PM
Location: Skype
Attendees: Alexandra, Annie, Borce, Hugh
Goals:

- Discuss project 2 specs
- Divide project 2 into separate tasks
- Assign tasks to individuals

Results:

- Three main tasks: SUBSCRIBE command, security, and report
- Annie and Hugh will focus on security
- Alexandra and Borce will focus on the SUBSCRIBE command
- Hugh will sketch out report
- Everyone will contribute to report sketch
- Hugh will write report based on sketch

Date: May 24th 2017, 2PM
Location: Alice Hoy 211
Attendees: Alexandra, Borce, Hugh
Goals:

- Work out how to include the certificate in the keystore
- Discuss vague statements in the project specs

Results:

- Basic SUBSCRIBE command is working
- New "certificate_unknown" exception when server acts as client in exchange & relay

Date: May 26th 2017, 4PM
Location: Alice Hoy 236
Attendees: Alexandra, Annie
Goals:

- Fix "certificate_unknown" exception when server acts as client in exchange & relay
- Test security and subscribe features

Results:

- "certificate_unknown" exception is fixed
- New exception when trying to run ezshare.jar file from Alexandra's computer
- Alexandra and Annie try setting up maven to include the keystore in the jar file
- Alexandra and Annie will continue working on a solution separately with the hopes of one of them having a breakthrough

Other communication

- Slack group
- E-mail
- In person before/after DS lectures