Design of Burnchat

December 11, 2016

G. Shaw – 38762118, T. Graham – 33756123,
M. Adria – 39636113, W. Qiu -42701136
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada

geoffshawbc@gmail.com, qiu@alumni.ubc.ca, tlgraham@glide.com, michaeladria@me.com

Abstract— **An implementation of a "user-proof" chat application is explored. The central idea is taking the best features in existing chat applications in order to demonstrate an application that is resilient to security breaches caused by misuse.**

I. Introduction

The market has shown to have accepted encrypted chat applications. The popular application WhatsApp has an estimated user base of over 1 million users, with the company reporting that in "one single day, over one billion messages were sent over WhatsApp" [9]. "Disposable" communication functionality has been shown to be popular with Snapchat sending approximately 400 million messages a day [2]. Many of the popular messaging application have security issues [9], but only one easily protects against a malicious user who has access to the device. Snapchat's chat functionality works in a disposable manner. Disposable messaging refers to messaging that does not store information about the conversation after the conversation has finished. This stop a malicious user from reading past messages even if they have access to the device on which the messages were sent and received. Currently, there are no chatting applications that combine solid security (end-to-end encryption), disposable messages, and ease-of-use.

Estimates have stated that by 2018 "IM apps will account for 75% of mobile traffic" [10]. The desire to have group chatting functionality is demonstrated by the popularity of applications such as WhatsApp, Apple's IMessage, Facebook Chat, Slack, and many more. These chatting applications can provide resilient defences against a wide range of attacks taking place over the network, but do not provide the strongest possible security against other types of undesired access. There have been reported cases of owners of electronic devices being arrested for refusal to provide access to their device [3]. This unauthorized access can be as simple as a curious friend borrowing a non-security conscious user's computer. The user in these situations would have much better peace of mind knowing that it is impossible to recover messages they have sent while not requiring extra user action.

Our solution provides multiple users the ability to chat securely between all users while preventing the storage of chat history, providing anonymity, and granting access to only those invited. The system is a web application that allows any user to initiate a chat. The system returns a one time use URL that can be given to another user to initiate the chat. The two clients establish a shared key, communicating through the server using https. The users are then able to chat using their shared key for end to end encryption. At any point, any user may request to add another user. The server provides another unique, one time use URL that can be used to join the chat. At this point, a new shared key is generated by the users, and the chat continues with the new user. Upon leaving the chat, a user is not able to rejoin the chat without a new URL being requested by a user currently in the chat. After all users leave the chat or the lifespan of the chat expires, no chat data is stored and the conversation is discarded.

We have combined the desirable elements found in popular messaging applications to provide a secure and easy to use application. We have taken the group messaging functionality of WhatsApp, iMessage, and others, the disposable messaging in snapchat, and the ease of use of Cyph. This results in an application that is user-friendly, minimizing the required user actions to remain secure.

Our evaluation methodology is split into two parts, security and usability. For the security evaluation we employ various attack vectors to gain access to the information within secure chat. We look at decrypting messages, man in the middle attacks to listen in, as well as cross site scripting to gain access. For usability, we look at whether or not our system can actually be adopted by the public. For this, we use user testing among a wide demographic.

After our evaluation, we found that certain aspects of our app were secure, while others were not. Cross-site-scripting and the re-use of chat URLs are prevented, but sharing URL invites unsecurely leaves our app vulnerable to a man-in-the-middle attack. In terms of usability, we discovered that sharing invite URLs via Facebook Messenger prevented users from successfully accessing a chat.

As a result of our evaluation, we believe that future iterations should include a method to verify the identity of an invited chat user, in order to prevent man-in-the-middle attacks. We would also change the invite process so that a

code is sent instead of a URL. This code would then be entered into a standard invite page.

Secure and private communication has always been important to humanity. Today, chat apps and chat services are a vital part of communication. Due to malicious actors, including powerful government agencies such as the NSA, there are many threats to private conversations. In the 20th century, it no longer became feasible for humans to encrypt messages by hand, due to the ease with which computers could break the ciphertext. Therefore, it is beneficial to society to have a service to provide secure communication without any effort from the user.

II. Related Work

Our design combines elements from several different existing works. The defining feature of our app, ephemerality, is inspired by Snapchat, as well as other online anonymous chatting services, like Cyph. Our task is to replicate this feature while ensuring that messages and conversations are irretrievable after the chat is ended, or a certain period of inactivity.

We will be using end-to-end encryption to maintain conversation confidentiality. Many services offer this security, notably iMessage, WhatsApp, and Cyph. Our method for encryption will be elliptic curve cryptography. Like many chat services, we want to offer group chat functionality, ie, having more than two people chatting together. We will need to implement a protocol to to create and share multiple keys as users are added to the chat group.

In many ways, the closest related work is Cyph: a user opens a room, is given a link to share (which expires in 10 minutes), both parties exchange keys, and then they can send encrypted messages. They use the Castle messaging protocol [4], which is as follows:

1) Asymmetric keys are exchanged under a layer of authenticated symmetric encryption.
2) For the cipher, Castle uses Poly1305-authenticated [5] Curve25519 [6] (Diffie-Hellman) with XSalsa20 [7] (stream cipher), plus an additional layer of Poly1305-authenticated NTRU-EES439EP1 [8] (asymmetric cipher) with XSalsa20.
3) This gives the cipher 256 bits of security, and gives Castle a good balance of efficiency and security: NTRU is a probably-quantum-safe cipher, with 20 years of research on it. [13]

Our challenge is to create a service with equivalent security, with the added feature of group chat.

III. Adversary Model

The objective of the adversary in our system is to obtain information that is shared in the chat. As this chat is designed to handle sensitive information, an attacker may infer that any given information transferred via our system has a high likelihood of being valuable. Access to the information could be though either decrypting ciphertext or gaining unauthorized access to a chat.

The initial capabilities of an adversary are access to the source code of the system, access to a standard user's capabilities on the system, and equipment capable of acting as a man in the middle. Access to the source code is available to the public due to Kerckhoffs's principle.

During an attack, the assumption is that an adversary has complete access to the network to intercept, modify, and forward messages. The adversary may also send any request to the server. This means an adversary has the ability to replicate any message that was sent by an authenticated user to the server. Further, the adversary can imitate a server to perform a man in the middle attack. The adversary also has access to a user's computer after the chat has ended.

IV. System Design

The key to providing client side encryption in a group chatting application is providing a symmetric key to all parties involved. The main feature we have implemented is an extension of Diffie-Hellman to allow for the dynamic extension of a symmetric key across a group of users, more specifically GDH.2. Reference [11] has shown that the shared symmetric key does not need to be re-calculated every time a new user enters the group, thus allowing for a less computationally expensive process to extend the secret to new users.

The initial shared secret will accessed through a link that the new user entering the group uses. The existing group member who shared this link keeps track of its state, and once the secret has been used once, it will refuse access with the same secret subsequent times. User will be encouraged to share this across a secure channel. To reduce the risk of an adversary brute forcing the secret from the time the secret was created, to the time the secret is used, the distributed link will only be available to be used for a short period of time before it expires.

We are using the Stanford Javascript Crypto Library, or SJCL. This library is light, secure, and optimized to be used in the browser [12]. The library also addresses the fact that native javascript's Math.Random() function is far from being adequate for cryptographic uses, and have implemented a pseudorandom number generator which "prevents an adversary who compromises the generator from recovering previously-generated data" [12].

Lastly to ensure that having the server be compromised does not allow for past messages to be leaked, we are not storing any data on the server. All messages are sent to the client by the server, but not logged anywhere, or stored in a database. The received messages exist as elements appended to the clients DOM, so that when the chatting window is closed, the past messages are gone.

The extension of Diffie-Hellman outlined [11] gives a computationally light way to extend the key to new users, and does not suffer from compromises when a user leaves the chatting session.

Although we are using an HTTPS connection to communicate between the server and client, we feel that providing client side encryption provides an extra layer of security to the messages being sent. This ensure that any malicious person with access to the server cannot just look directly at the messages being sent, and instead would have to decrypt them in order to read the messages.

We are passing the initial shared secret through the link as it provides an easy to use way to get an initial secret for the users, and doing this over an HTTPS connection will keep the secret save. We are limiting the URL to only be used once so that an attacker who finds the URL will be unable to join the chat room. This is an issue with the standard password authentication commonly used in chatrooms.

The lack of logs and stored information on the server prevents a malicious user who has access the the server from accessing the messages sent. In addition this prevents sql injection attacks, as there is no database to access.

Principles of fail-safe defaults, open design, psychological acceptability, and economy of mechanism are employed by our design. When a URL link is been accessed for the second time, a user is rejected to enter the chat room as default, even if the user if genuine. This is an example of fail-safe defaults in our design. The source code of our design is open source and available to the public, following the open design principle, because the security of our system does not reply on the secrecy of our design. Our design also follows the principle of psychological acceptability by using urls as a key to access to chat room, which does not place additional burden to the end users while providing them the security features. In addition, the principle of economy of mechanism is adopted where we keep the structure of our design at minimal by using simple and effective key-exchange protocol and storing as little information on the server as possible.

V. System Prototype

The web application is hosted online for the most realistic testing environment. The backend used Node.js. The application is deployed on the platform as a service, Heroku. The SSL certificate is provided by Heroku. The prototype is desktop based and built for modern web browsers. The implementation can be found at [14].

VI. System Evaluation

A. Evaluation Methodology

Our evaluation strategy is inspired by and generally follows OWASP Testing Guide 4.0. Specifically, our testing begins with attempting to retrieve information from a conversation we are not apart of. We set up a proxy between the client and server of an ongoing chat to intercept messages. From here, we see if it possible to decrypt the messages or, during key establishment, determine what the shared key is. We also attempt a man in the middle attack to convince a user they are in a secure chat, while a third person listens in. We try to alter the javascript that a user receives to display a different URL for a different chat where messages are relayed. We also try to alter the key sharing algorithm to provide a known key to the adversary. We then attempt to use cross site scripting to inject javascript code that sends user information and messages to an adversary. The third attack vector we try is via the URL. First, we attempt to use the URL to grant more than one user access. Second, we attempt to brute force the URL to gain access to an ongoing chat.

We also ensure that users will want to use our web application. Upon deployment, we use user testing to ensure ease of use and provide an outsider's perspective. We gain a wide perspective by getting testers of different age, education, and technologic backgrounds.

The source code is hosted on Github to allow future inspection and ongoing development as new security requirements arise.

B. Results of the evaluation

A man-in-the-middle attack is successful if the method of sharing the chat invite is compromised. If a third party can open a chat with the initial invite, prevent that invite from reaching the proper person, then open a new chat and inviting that person, the third person will successfully gain access to the chat.

Cross-site-scripting is prevented by escaping characters necessary for scripting.

We verified that each unique chat URL (both the initial user's URL and any invite URL) can only be used once.

The results of the usability study showed that the simple user interface was adequate while also demonstrating issues in URL sharing. When a user was sent the link and instructed to start a chat, all subjects intuitively began a chat, entered their name, and received a URL to invite someone. All users were able to send the URL, with the exception of one who did not know what a URL was. When it came to using the shared URL, users complained that selecting the URL by dragging the cursor would close the URL popup if the mouse was released outside of the popup. When users used email to send the URL code, they were able to chat without issue. However, if the link is sent via facebook, most of the time, the system would think that the URL has already been used.

C. Discussion of the evaluation results

The possibility of a man-in-the-middle attack suggests our design needs a method of authenticating whoever receives the invite URL. Further designs could include a verification system based on Google Authenticator or similar authentication application.

The user testing of the system showed much room for improvement. The three issues that became apparent are: unclear on how to share a URL, issues on selecting the URL, and inability to share via Facebook. The issue from Facebook comes from how URLs are sent in Facebook messenger. When a user sends a link, Facebook will access the URL to find information regarding the link to display to the recipient. In the case of Burnchat, this will invalidate the URL. The solution to this is to use a code instead of URL that can be

entered in a "Join Chat" option from the home page. This code would also clarify on how to share a chat. Lastly, the copying issue could be fixed by not having the box disappear when clicking outside the box.

VII. Discussion

The discussion of Burnchat's viability as a useful system begins with an overview of the positives and negatives:

Pros:
- Disposable
- Support for multiple participant in a chat
- Secure
- Provides anonymity
- Lightweight and efficient algorithm

Cons:
- Require secure methods of sharing urls
- No direct/automatic methods of authenticating other chat participant if url is compromised
- Lack of support for transferring images and videos
- Can do some user-based evaluation to further improve

The testing and evaluation process of our implementation of Burnchat reveals that it successfully meets the goals we planned at the beginning of the development cycle. However, there still exists several areas of possible improvement.

Burnchat has been proven to be a functional web application that facilitates online communication while maintaining security and disposability. Also, unlike Cyph, its support for multiple participants in a chat room makes it more suitable for a group of users. Furthermore, the option of anonymity widens possible use cases. Its choice of algorithm does not only offer security but also allow the client-side application to be fast and lightweight. Lightweight applications generally yield a faster rendering speed on end devices.

While the features mentioned above helps Burnchatt to differentiate itself from other chat applications and offer a unique experience to users, there are usability issues that could be improved upon. Firstly, distributing URLs securely can be a cumbersome job for users who value accessibility to the chat room more than its security aspects. Currently, users are responsible for sending the URL generated by Burnchat via some pre-existing secure communication platform, such as Facebook Messenger and Skype. It is questionable whether a user would be willing to compromise familiarity and switch to Burnchat for its extra security. Also, it is worth-noting that the security of the system is, to some extent, relying on the privacy of the URL. There is no built-in mechanism for the chat participants to authenticate an invitee, who newly entered the chat room, without using other secure communication methods to ask directly if the intended invitee is in the chat room or not. For example, when Alice sends a Burnchat invitation URL to Bob via Facebook, and Trudy, who has access to Bob's Facebook account, then the privacy of the chat room is compromised. Although Burnchat's policy of only allowing a URL to be used once will allow Bob to quickly detect the existence of Trudy's intrusion, people in the chat room have no way of identifying Trudy's attack without being informed by Bob directly.

Secondly, although the initial aim of Burnchat is to provide users a lightweight communication platform with minimal functionalities, the fact that it only allows messages in text to be transmitted may not always satisfy user needs. Support for multimedia messaging would almost certainly make Burnchat a better platform for group collaboration.

Here is a summary of improvements to make moving forward:
- Force https
- Improve client-server authentication
- Improve the generation of random number
- Use codes instead of URL for chat invites
- Extend messaging capabilities to multimedia

VIII. Conclusion

Our design and implementation of Burnchat provides users with a functional chatting platform. Similar to many instant-messaging applications that currently exist on the market, Burnchat secures user communications with crypto systems that are publicly published and examined. Furthermore, it successfully differentiates itself from others by supporting disposable group chat. As an entirety, Burnchat is an application that is able to stand its own and effectively provides users with liable and unique functionalities.

Further development of the system could result in adoption by security minded users. Increasing usability will make the system more intuitive and create awareness of the systems features. Further, building on the system's security and bringing in outside analysis will boost confidence in the application.

References

[1] *Play.google.com*, 2016. [Online]. Available: https://play.google.com/store/apps/details?id=org.thoughtcrime.securesms&hl=en. [Accessed: 12- Oct- 2016].
[2] J. Crook, "Snapchat Sees More Daily Photos Than Facebook", *TechCrunch*, 2016. [Online]. Available: https://techcrunch.com/2013/11/19/snapchat-reportedly-sees-more-daily-photos-than-facebook/. [Accessed: 12- Oct- 2016].
[3] C. Matyszczyk, "Man arrested for refusing to give phone passcode to border agents", *CNET*, 2016. [Online]. Available: https://www.cnet.com/news/man-charged-for-refusing-to-give-up-phone-passcode-to-canadian-border-agents/. [Accessed: 10- Nov- 2016].
[4] "Castle messaging protocol v1", *Google Docs*, 2016. [Online]. Available: https://docs.google.com/document/d/1XVh4ALXhbfxi70QSUY-xHclauob8O635bSySy6f1Ysk/edit?pref=2&pli=1. [Accessed: 10- Nov- 2016].
[5] D. J. Bernstein, "A state-of-the-art message-authentication code", *cr.yp.to,* 2005. [Online]. Available: http://cr.yp.to/mac.html. [Accessed: 10- Nov- 2016].
[6] D. J. Bernstein, "A state-of-the-art Diffie-Hellman function", *cr.yp.to,* 2005. [Online]. Available: https://cr.yp.to/ecdh.html. [Accessed: 10- Nov- 2016].

[7] “XSalsa20”, *Sodium crypto library,* 2016. [Online]. Available: https://download.libsodium.org/doc/advanced/xsalsa20.html. [Accessed: 10- Nov- 2016].
[8] “ntru-crypto”, *GitHub,* 2016. [Online]. Available: https://github.com/NTRUOpenSourceProject/ntru-crypto. [Accessed: 10- Nov- 2016].
[9] R. Mueller, S. Schrittwieser, P. Fruehwirt, P. Kieseberg and E. Weippl, "Security and privacy of smartphone messaging applications", *International Journal of Pervasive Computing and Communications*, vol. 11, no. 2, pp. 132-150, 2015.
[10] L. Piwek and A. Joinson, "“What do they snapchat about?” Patterns of use in time-limited instant messaging service", *Computers in Human Behavior*, vol. 54, pp. 358-367, 2016.
[11] G. Ateniese, M. Steiner and G. Tsudik, "New multiparty authentication services and key agreement protocols", *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4, pp. 628-639, 2000.
[12] E. Stark, M. Hamburg and D. Boneh, "Symmetric Cryptography in Javascript", *2009 Annual Computer Security Applications Conference*, 2009.
[13] N. Wolchover, “A Tricky Path to Quantum-Safe Encryption”, *Quanta Magazine,* 2015. [Online]. Available: https://www.quantamagazine.org/20150908-quantum-safe-encryption/. [Accessed: 10- Nov- 2016].
[14] “Burnchat”, 2016. [Online]. Available: https://burnchat.herokuapp.com/. [Accessed: 11-Dec-2016].