MUSM

Minimal Universal Secure Messaging

Nikita Samsonau
samsonov@mit.edu

Allan Costa
allanc@mit.edu

Driss Hafdi
dhafdi@mit.edu

***Abstract*—Messaging apps became an essential part of modern communication. Very few of those apps, however, provide methods to verify privacy, and most users have no option but to trust that end-to-end encryption is provided. Cross-platform data sharing has been shown to be common practice, some of those platforms are forced to have back doors systems under specific political regimes. To ensure privacy, we propose MUSM (Minimal Universal Secure Messaging), an application that guarantees end-to-end encryption by overlaying existing messaging systems with RSA message encryption.**

I. Background and Motivation

Messaging apps are an ubiquitous and necessary technology for communicating today. Among teenagers, they are currently preferred over social media apps [1], and represent a rising trend of customer communication channel in the developed world [2]. Very few of those apps, however, provide methods for users to be able to verify privacy, and a number of those application have been reported to share data across different platforms, or have plans to do so [3]. Specific political situations [4] often enforce back door policies that leave user data exposed, and unreliable handling of encryption is pervasive among platforms [5]. Finally, the rapid adoption of so-called secure messaging apps such as Wire or Signal reveal a growing desire for privacy.

II. Threat Model

Whenever a message is sent, it passes through multiple points from the sender to the received: from browser, to router, to ISP, to the webserver, and then to the receiver. While TLS and similar approaches protect comunications from client to the webserver, it does not protect from the webserver itself. Even if we assume that apps like Facebook that claim high standards of security actually implement it, they still might be limited by the government regulations. For example, messages on Facebook can be retrieved by the government provided a subpoena. Therefore, we focus specifically on end-to-end encryption, meaning no unencrypted message leaves the user machine.

Since we need to display decrypted user messages in the browser, for the scope of this project, we assume website like Facebook and Whatsapp do not maliciously record user data and only store the sent texts. This limitation, however, can be lifted in the future, as described in the Future work section. Moreover, we need to rely on Google Chrome extensions framework - specifically, on its sandboxing mechanisms and storage. We assume no other pages or extensions have access to the code and environment of our chrome extension. This assumption is limited, however, since a malicious extension could still read the decrypted messages from the page. And of course, we assume the user's machine is not compromised, since that would defeat the purpose of this application.

III. Approach

We propose MUSM, a multi-platform, overlay extension that uses messaging apps to send encrypted messages between individual users and groups in order to provide strong and minimal security.

MUSM is a modular system that is able to provide End-to-End Encryption to any preexisting messaging service. The application works by locally intercepting messages before they are sent through a preexisting messaging infrastructure, encrypting it, and forwarding the encrypted message to the interlocutor. On the other side of the communication channel, the encrypted message is detected by the receivers app and decrypted. The app then re-renders the UI such that every encrypted message appears decrypted to the receiver.

MUSM provides a seamless interface to the user with a simple chrome extension menu, so that users impact in communication is minimal, and users are only aware of the application through tags that indicate successful encryption and decryption.

The proof of concept of MUSM was implemented as a chrome extension application and built to operate with Facebook Messenger. The extension was designed to be modular so that extending it to other UIs require minimal effort and ensures security. To prove the later point, we implemented a basic wrapper for WhatsApp in less than 1 hour, and although slightly buggy, it confirmed our hope of modularity and rapid prototyping. Moreover, as outlined in our threat model, we designed our extension to limit its interaction with any online resource. This means that we store locally the main user's private key, all of his/her pre-encrypted messages and every public key received from a given user.

IV. Implementation

The extension was designed to achieve two main goals: security and universality. Security is achieved through a modular and hierarchical approach to functionality, enforcing least privileges to each of the system's components. Communication with local storage, for example, is localized and constrained to a single component of the system.

In order for MUSM to be universal in a chrome extension level, it must be able to overlay and interface with any standard, HTML-based messaging app in the Web. For such, we developed an *UI Interface* class that abstracts the required methods for MUSM to operate. By building on this

class, simple interface-specific code can be written to indicate essential messaging functionality, such as what to encrypt, actions to intercept and user specific identifiers. As a proof of concept of this, we provide two example scripts that abstract basic Messenger and Whatsapp functionality, allowing MUSM to operate on those.

We now investigate MUSM's architecture and specific component features. MUSM has four layers of operation, each with minimized privilege for specific functionality. Communication between each component is done through Chrome's built-in *runtime* API, allowing the system to behave in a OKWS-like model [6]. Figure 1 depicts message passing through the system.

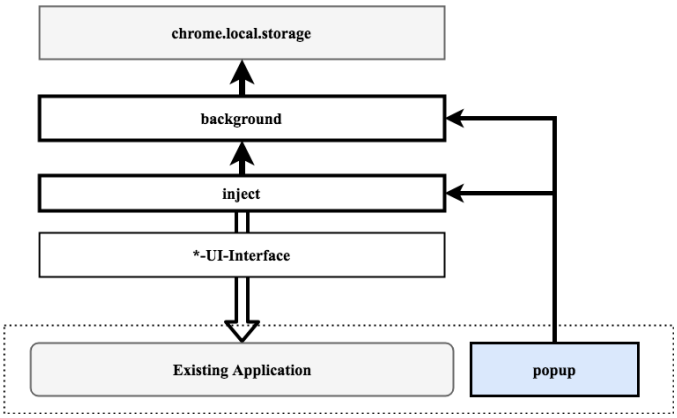

Fig. 1. The hierarchical structure of MUSM. Black arrows indicate requests-response message passing between the different layers. *Inject* code acts on the messaging app's DOM through methods provided by *interface*. Note that only the bottom two components (*Existing Application* and *popup*) have interfaces for the user to interact with.

A. Background:

Background is responsible mainly for handling low-level and application-oblivious functionality, such as communicating with Chrome's local storage, or responsively activating or deactivating MUSM according to the current visited website. Background also performs essential encryption operations, such as generating keys or making RSA pairs. It is independent of local conversations and runs continuously with the extension. It communicates with other parts of the application through message passing and it doesn't initiate requests on its own.

B. Inject:

Inject is responsible for setting up most of the interface overlay logic for the messaging application. It commands DOM operations by invoking methods defined in *interface* (see below), such as outputting RSA keys or alerting the user about key deletion. It also makes sure to include a nonce to the message before encryption, limiting potential attacks on the system.

When a conversation is detected, *Inject* builds conversation-specific, concrete classes which are based on a conversation's unique identifier and are responsible for most of the cryptographic methods. Through *interface*, *inject* can then operate on the DOM and intercept, encrypt and decrypt message passing.

C. User interface:

As previously outlined, modularity was a driving factor for MUSM. Hence, when designing the user interface code, we had to implement a basic abstract class that provides methods that will be required by any hypothetical messaging application.

It provides methods to intercept when the user presses the "Enter" key, extract the text in the input box, encrypt it and forward it again to the messaging service. Using a similar scheme, it also allows the user to share its public on the click of a button.

Another one of its functions is to provide ways to render encrypted messages in order to provide a seamless experience to the user. We append a given keyword at the beginning of any message sent by MUSM. When the user interface code detects one of the keywords, it re-renders the textbox in the conversation to display the appropriate text, whether it is a decrypted message or an information message telling the user that encryption is turned on/off for the current conversation.

As previously described, interface can be understood as a basis code that provides a foundation class on which interface-specific code is build. In order to implement a new messaging app specific wrapper, a programmer will have to modify the *addListener fillText* and *injectScripts* methods, updating the location of the input textbox in the html so that MUSM can send encrypted messages and share the user's public key. The *addListener* method will also need to include a way to differentiate sent and receive signals, since the rendering would differ in some aspects. Lastly, *updateUID* will also me modified so that MUSM can update who the current interlocutor is.

Our prototype contains two such interface-specific scripts, one for Messenger and one for Whatsapp. The Messenger version was our initial implementation and is bug free. On the other hand, we implemented the WhatsApp version mostly as a proof that our system was modular enough. Given how fast the implementation turned out to be, we are fairly confident in the modularity of our system and its ease of use for new programmers.

D. Popup:

Popup is the main form of interaction of MUSM with the user. It abstracts the extension functionality into a simple interface for controling encryption, and is responsible for keeping globals that report encryption status for the users. Through popup, the user is able to toggle global encryption, local/conversation-specific encryption, send his/her RSA key or delete conversation keys.

V. Future Work

Since the scope of the project was limited, we focused mostly on the confidentiality aspect of the application. A clear improvement would be adding signatures and MACs to the messages, so that we can ensure full integrity of the messages. As this would imply message overhead, such an implementation would require careful design and trade-off choices.

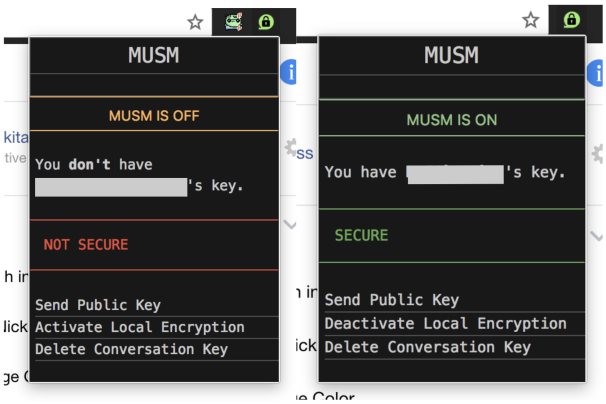

Fig. 2. MUSM's popup interface can be used to send public RSA keys, toggle conversation-specific encryption, or delete keys. The menu is also color responsive to indicate if messages are currently being encrypted or not.

As mentioned in the Thread model, we assume the website does not intercept data from the users screen. We can lift this assumption by employing iframe and relying on the browser security instead.

Since our architecture has been designed to be generic and modular, we can easily extend it to other services, and much of future work will come from expanding MUSM to other platforms. Moreover, we could create a generic way to create interfaces for other messaging apps - in particular, we are interested in text messaging apps that don't require internet connection.

The UI could definitely be more user friendly. As of now, MUSM requires a significant amount of user control and some basic knowledge of some cryptographic principles.

Finally, an essential future work would be expanding MUSM into mobile technology, since much activity of messaging apps rely on those. In this scenario, user-friendliness would require a form of coordination between mobile and computer devices, for which case a number of approaches are possible and for discussion.

VI. Conclusion

In this paper, we have proposed and described MUSM, a Minimal Universal Secure Messaging technology that overlays existing messaging applications to ensure privacy. We built a modular and resilient software system that is able to intercept, encrypt and decrypt messages while providing a simple and intuitive interface for users to operate it. One of the goals for the project was scalability and universality, which was further confirmed when we quickly expanded MUSM to WhatsApp from existing infrastructure. Much work is to be done for future work, but as a proof of concept MUSM was able to achieve its main goals.

Appendix A
Project Repository

https://github.mit.edu/dhafdi/6.858-Final-Project

References

[1] *Messaging apps are now bigger than social networks*. Business Insider. Sep, 2016.

[2] *Facebook study: 53% of consumers more likely to shop with a business they can message*. Campaign US. August 03, 2016.

[3] *WhatsApp will not share user data with Facebook until it complies with GDPR, ICO closes investigation*. Tech Crunch. Mar 14, 2018.

[4] *We(Chat) The People: Technology and Social Control in China*. Harvard Political Review, 2017. http://harvardpolitics.com/world/wechat-the-people-technology-and-social-control-in-china/

[5] *WhatsApp Security Flaws*. Wired Magazine, 2018. https://www.wired.com/story/whatsapp-security-flaws-encryption-group-chats/

[6] *Building Secure High-Performance Web Services with OKWS* https://css.csail.mit.edu/6.858/2018/readings/okws.pdf