SNAP-SAFE Re-Imagining IM Security*

Wali Ahmed Usmani University of British Columbia
wusmani@cs.ubc.ca

ABSTRACT

Instant Messaging (IM) services have seen a boom in the past few years [8] and have come in a variety of different flavors allowing users to share text, emoticons, animations, music, pictures and videos. Phones and other portable smart devices like tablets and computers which often serve as a platform for such services, are not only valuable pieces of hardware but also contain huge variety of private information about their owners and unsurprisingly, are often the target of theft. According to Consumer Reports, in 2013, 3.1 million phones were stolen in the US alone [9], an alarming number which seems to only be growing. This makes the need to ensure user data remains private, and intact in the case of a phone being stolen even more pressing.

This paper presents Snap-Safe, a redesign of the way IM security is managed. The crucial flaw identified in the current systems is that the key used to encrypt IM data is located in the device memory itself, making it vulnerable to any adversary model that allows access to said memory, a very real possibility in the case of the device being stolen. Snap-Safe rectifies this issue leveraging a modified version of the Kerberos[11] 3rd party authentication and security mechanism to live fetch keys at request-time. Snap-Safe provides further advantage in the form of a fine grained access control scheme where users can grant, revoke and control the number of times a specific media item can be viewed. The system is evaluated in terms of the security guarantee it provides and its impact on how IM services respond to data access times

General Terms

Security, Instant Messaging, Internet

*(Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.

Keywords

SnapSafe, Latency, IM security, Encryption, Kerberos, TGT

1. INTRODUCTION

The popularity of mobile device based messaging services has seen a sharp rise in the last decade. The trend started off as SMS or short message service provided by the telecom companies powering the SIM cards of the mobile (phones) but as internet connectivity became a common feature, services have taken the form of pc based instant messaging services which provide real time information transfer between its users. Some examples powering the 7 billion [6] mobile devices in existence today are Whatsapp [5], Viber [4], Line [2] and Snapchat[3]. As the use of mobile devices as IM messengers becomes more common, the hardware itself becomes a treasure trove of information of its owner as it can contain private conversations, images and other forms of media, users may have shared with each other.

As a consequence, this data in the wrong hands could be dangerous as it could be used against them in a variety of ways including harassment, blackmail, and ridicule making it important to ensure its safety. This threat is a very real one since mobile device theft has seen a massive rise [9] where reports suggest 1 in 10 users are victims of mobile theft [7]. Current countermeasures are severely lacking in effectiveness because most depend on users locking their phones using short 4-6 digit pin numbers or touch screen pattern combinations (so that they are easy to memorize). However, they can quickly be brute forced if the adversary has access to the Password-Based Key Derivation Function (PBKDF).

Current 'secure' IM applications offer end to end encryption base on XMPP and Jabber protocol to ensure that user messages cannot be intercepted but do not offer protection in the case the user device is stolen, since the phone cannot differentiate between user and attacker apart from standard lock screens, which, as discussed before, have limited capabilities. Major mobile device software manufacturers like Google and Apple have taken note of this issue and in response introduced file system level encryption on their devices. While successful, these solutions often require specialized hardware uncommon in most mobile devices.

This paper presents Snap-Safe, a redesign of encryption based security in IM services that uses the concept of Kerberos based 3rd party authentication and encryption Ticket Granting Ticket to allow or disallow access to data in a conversation between users. Apart from secure and verifiable information transfer, using this scheme has multiple benefits. Firstly, it allows fine grained access control across various data types (a message containing text and one containing an image may require different levels of protection). Secondly, in terms of usability, the user does not need to remember any phrase, key or password to ensure that the data remains safe. To maintain keys off the local device, the scheme requires wireless connections either in the form of Internet through Wi-Fi, Bluetooth Modem, 3G or 2G radios or simple GSM connection with SMS capability. Snap-Safe is implemented on the Android OS (4.4 KitKat and below) based on a open source IM service [12]. The system design is based on IM semantics of Snapchat application [see Section 2] but the concept can be applied to any such service. Lastly, the data while safe is recoverable from the IM service if required by legal warrant, i.e. from law enforcement agencies, however, the protocol can be trivially changed if the user wants complete privacy.

In developing this solution, certain factors were kept in mind such as, it not requiring any special hardware, its ease of integration into existing IM services and its ability to avoid changing the way users currently use IM services in terms of the layout, and data access time. If the change is too drastic, users would be de-incentivized to use the mechanism. Snap-Safe, deployed in an Android environment is designed to meet all of these requirements.

To summarize, the contributions in this paper as a follows:

1. 1. We propose a Kerberos 3rd party based authentication and security mechanism to ensure privacy of IM service data in the case of the mobile device being stolen.

2. 2. A high granularity access control scheme adaptive to data type and data sets

3. 3. Safe but recoverable user data if required by legal authority (with a warrant) with an option to further modify the protocol for complete privacy

2. BACKGROUND

SnapSafe's design was inspired by the usage semantics of a popular IM messaging service called Snapchat which offers unique functionality from the user's perspective. The service stores all media in an encrypted format and guarantees that images/videos are 'discarded' [10] after they are viewed by the user they are sent to (and any attempt at taking a screen shot will result in the sender being notified).

While most users assume the entire transaction be secure, the reality is very different [10]. Before October 2014, Snapchat used simple Electronic Code Book encryption with a plaintext string serving as the password. Decrypting the media object was as simple as locating the string within the application APK.

After a revision of the security mechanism, Snapchat now generates a random IV and string for each snap for encryption in Cipher Block Chain format but stores all the keys in a root directory, in a file called 'bananas1', encrypted with an MD5 Hash of the phone's Android UID and constant string 'seems legit'. Thus once again, decrypting snaps is trivial after rooting (jail-breaking) the device (to gain access to the root user directory). [1]

SnapChat was chosen as the model application since it fails to provide the security guarantee it claims as well as lacking desirable features such as secure storage (of media). Snap-Safe is proposed as a solution to provide such features and guarantees security without requiring specialized tools to achieve it.

3. RELATED WORK

The current state of the art in IM security focuses on the communication of the application itself. A popular example the Guardian Project's Chat Secure which is based on XMPP protocol and guarantees end to end secure transmission of data by ensuring that all communication between two clients is encrypted with multiple levels of encryption including AES and as well as RSA. The system is based on an initial handshake after which symmetric session keys are exchanged making passive network monitoring attacks very difficult. The system however still lacks protection in the case that the device is stolen. While the app can be locked by a standard passkey based lock screen, this feature is switched off by default to allow unencumbered access to users who would find it bothersome to remember and repeatedly enter the password over and over again.

Solutions in other forms are also being presented in the form of Full Disk Encryption (FDE). A good example of this is the iOS 8 and Apples iPhone 6 where all the data stored on the phone is encrypted. The encryption key used is stored in Trusted Execution Environment (TEE) called Secure Enclave [18] to disallow an adversary to extract it. Since only the local device itself knows and can extract the key from the Secure Enclave, all decryption attempts must take place on the device itself and each attempt is computationally intensive at around 80ms.

While this measure may be effective, it is not currently widespread as there are many phones and operating systems where FDE is not enabled by default and there is a general lack of awareness of how to use it. Furthermore, FDE can have a high computational. While newer, powerful mobile devices may have computational resources, older devices may not, thus adversely affecting user experience due to sluggish performance.

Another method for secure key generation is finger print scanners, a scheme in which the user does not need to remember any particular password and can unlock and generate file system encryption key directly from their thumb imprint. However, very few phones and smart devices are equipped with such a technology for this to be a feasible solution as of today.

4. ADVERSARY MODEL

With the end goal of being able to access and recover user images, the adversary tackled in this paper has two capabilities. Each capability is described with a scenario of how it might be achieved in a real world scenario.

Firstly, the adversary can steal the victim's phone which gives him access to the phone's flash memory. If the phone is unprotected, a simple swipe should be sufficient to unlock it. The adversary could bypass screen locks in multiple ways including guessing, software bugs as well as instrumenting the boot image of the device to brute force the 4-6 character pin. Secondly, it is assumed that the adversary also has access to the unchanged application APK as well key derivation functions used by the mobile operating system directly from the application market where users download it from. The adversary is able decompile application code to gain access to any hard-coded information it may contain

In a logical extension to the aforementioned capabilities, writing, modifying and instrumenting the device contents is within the adversaries' power, however the adversary may not use wireless connections to impersonate the victim or allow the stolen device to communicate with a non-local host location due to the multitude of kill switch applications, GPS tracking and device location services in case it gets lost.

As mentioned above, FDE and finger print detection based lock screens are not considered [see Section 3].

5. SYSTEM DESIGN

This section will explain the Snap Safe protocol in detail as well as how it integrates into IM service model. The key concept of Snap-Safe is that the key used to encrypt data must not be kept on the device itself and this is achieved through a modified version of Kerberos protocol which uses TGT to provide authentication and secure information transfer. For the sake of simplicity we will consider two users, A, who is trying to send a private image to user B. As the scenario expands, B will be a victim of phone theft and it will be demonstrated how adversary C cannot access the image.

The trusted server uses SSL connections for all communications generated at the server initialization. Snap-Safe will already have the public key hardcoded into its APK along with the server URL. All messages to the server are encrypted with the servers' public key.

The system is evaluated upon the security guarantee it provides based on its protocol as well as its impact on user interaction and media access times using various means of communication such as 3G or Wi-Fi. Section VII covers the latter in greater detail

5.1 Signing Up for First Time Registry

Fig.1 shows how the user interacts when setting up an account with the server. The application uses a simple Key Generation Function to create a unique client key. The user then sends his user name and key along with a nonce to the server, encrypted with the server's public key. The server replies back with an acknowledgment and the nonce. The user has now registered for the IM service and from this point forward, all messages sent to the client are encrypted with the client key.

5.2 Sending A Snap

Fig 2 shows that when User A is ready to send a snap, the application will create a secure random string key and encrypt the picture with it. A message will be created containing the snap, the random key, an integer counter to control the number of times this image can be accessed, and the intended recipient. The original picture is then securely deleted (overwritten on by the encrypted image) and the key is discarded once the message is sent.

The server receives the message, stores it in its database and replies with an ACK as well as a Ticket Granting Ticket (TGT). This TGT is encrypted with a secret key that is only known to the server and contains the User A's username, the time when the initial message was received and the recipient id (User B's id). Since the secret key is known only to the server, only it may decrypt a TGT to reveal its contents.

5.3 Viewing A Snap

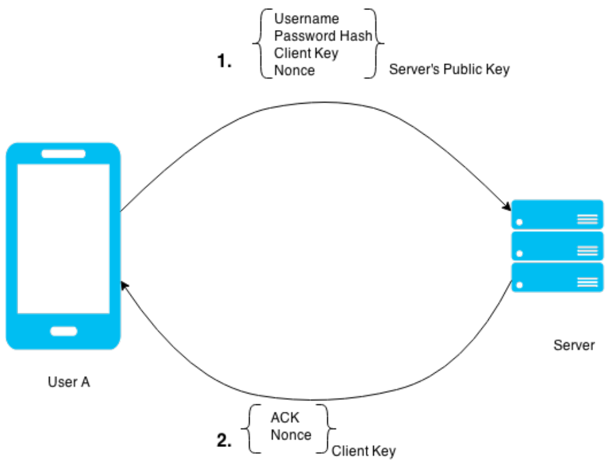

Figure 1: IM Registry

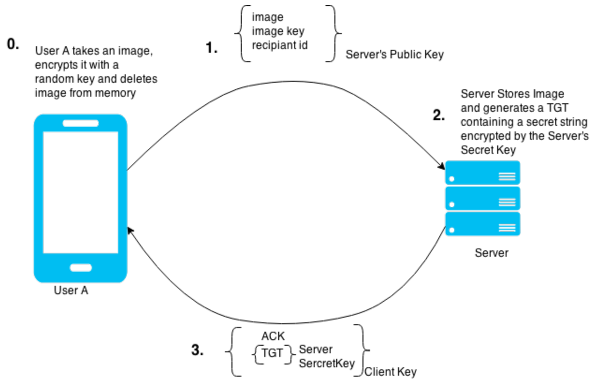
0.
User A takes an image, encrypts it with a random key and deletes image from memory
1.
image
image key
recipiant id
Server's Public Key
2.
Server Stores Image and generates a TGT containing a secret string encrypted by the Server's Secret Key
User A
Server
3.
ACK
TGT
Server SercretKey
Client Key

Figure 2: Sending a Snap

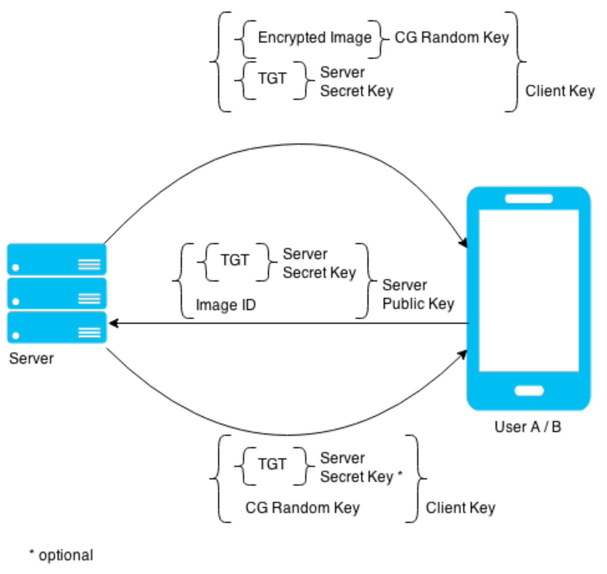

Figure 3: Receiving a Snap

Depending on push or pull based semantic, when the server is ready to deliver the image to the recipient, User B in this case, it will send the encrypted image, a unique image id, as well as a TGT containing sender details, recipient details, a unique TGT number and the integer counter. Once again, the TGT is encrypted with Server secret key.(Fig 3)

When User B is notified that he has received a message, he will open the IM application to view it. When he does this, the application sends a message to the server containing the image id, and the TGT. The server will then reply with the image key as well as a new TGT which will have the same information as before except that it will have a different TGT ID and the integer counter will be subtracted from.

If User A wants to view his own snap, he may do in the same way however User A is not forced to abide by the limited number of views since he is the owner of the said snap.

5.4 Device is Stolen

With the multitude of ways to track moderns smart devices, the most common action taken by a thief is to turn the device off (and often remove its battery) so that it is unable to communicate with tracking systems. However, even with the device off, extracting data from phone memory can be done in a variety of methods [detailed in Section IV]. Snap-Safe requires the owner to report the device stolen as soon as possible so if C steals User B's phone, User B would report their phone missing. This missing report would cause the server to blacklist all TGTs that list User B as a recipient so that if C tries to request a key to unlock an encrypted picture, the server will refuse and attempt to request location data to track device itself in an attempt to locate C. Since the key to decrypt the image is not on the device itself, the only option for C to access the picture is brute force which is not feasible.(Fig 4)

6. SYSTEM PROTOTYPE

We created a prototype system explore the practical implications of setting Snap Safe protocol. The implementation

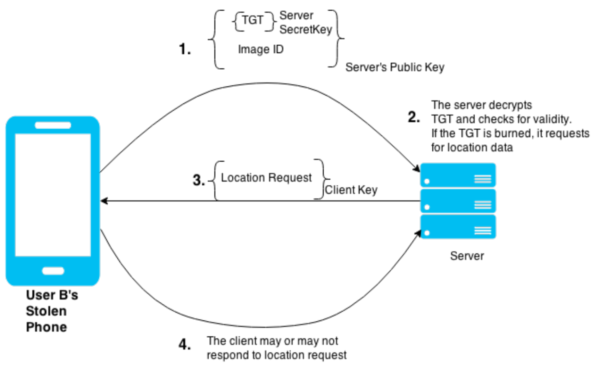
1.
TGT
Server SecretKey
Image ID
Server's Public Key
2.
The server decrypts TGT and checks for validity. If the TGT is burned, it requests for location data
3.
Location Request
Client Key
Server
User B's Stolen Phone
4.
The client may or may not respond to location request

Figure 4: Phone is Stolen

was divided into two portions, the Android Application and the server back-end system.

6.1 Snap-Safe Application

The Snap Safe application was written for Android up to API 19 (latest version of Android Kitkat) but is compatible with as low as 14 (Jelly Bean). The source was based on the simple Android IM project[12] on GitHub which provided the a bare-bone implementation of HTTP Post communication mechanism.

The code went through multiple modifications including the ability to handle and transmit image files, file encryption as well as message encryption. The communication mechanism was modified by adding a custom entry into the device certificate store to allow for HTTPS communication to the server at install time.

Advanced Encryption Standard (AES) 128 bit symmetric key encryption algorithm was chosen for image encryption because it offers fast performance and is reasonably secure at 3.4 x 10ˆ38 different combinations.

The selection of the random key was more of a challenge. Using a pseudo random number generator is not secure because if the seed is discovered, guessing the password becomes a real threat. Thus Java's Rand and Apache Common Lang's pseudorandom string generator was not suitable to generate a random key for image encryption. We used Java 1.7+'s Secure Random generator, a cryptographic-ally secure random number generator that complies with FIPS 140-2, Security Requirements for Cryptographic Modules. While secure random is computationally more expensive than generating a pseudo random string, it was necessary to make use of this since the entire point of the system would be lost if keys were vulnerable to informed guessing attacks.

6.2 Server

The prototype server is based on a 2 Ghz Core 2 Duo running a Linux 12.04 Ubuntu distribution. The server ran on the XAMPP server stack with PHP version 5.0+ and MYSQL database as message and user repository. Communication used SSL/TLS security with default configurations in the package and the certificate was installed in the phone's certificate repository before a successful connection could be made. Our understanding is that if the server was to be launched for commercial use, we would have to purchase a certificate The main interface was powered by PHP with each request specified by action id which in turn used the shell exec command to run a Java File that performs the required operations including storing and recovering files, handling encryption and generating TGTs.

7. EVALUATION

Our system is evaluated on its security guarantees as well as its performance in terms of the time it takes to open media objects. Since the security properties have already been explored in Section V (System Design), this section will focus more on the latter.

7.1 Evaluation Methodology

One of the foremost requirements of the Snap-Safe system was that it must not change the way IM is used, since most users are accustomed to this already and would be hesitant to change. Most current IM applications using little or no encryption usually tend to support instantaneous or near instantaneous conversation access since they store data in the plain on the user phone.

Snap-Safe on the other hand fetches keys for encrypted media and text, at request time forcing the user to wait between request and decryption. Thus, evaluation focuses on delay and measures it across different communication mediums, specifically through WiFi and 3G Internet Connection. Since the keys are only 128 bits long, SMS can also be used to fetch keys although this would take considerably longer and would be considered and emergency only mechanism. To test the delay between media item request and access, we implement code timing statements in Java, specifically System.currentTimeMillis();.

. When the media object, in this case, a view image icon for an encrypted image is tapped, the timer starts. At that point, the request key function is called through the network. When the key is downloaded, the specific image is decrypted, a new 'Image Viewing' Activity is launched and the timer is stopped.

The test was repeated 20 times on the UBC Campus 'UBC Secure' WiFi network as well as the 3G Network provided by Fido Canada at UBC ICICS Building as phone location. Due to the lack of a GSM module, SMS tests could not be performed with the server. Instead, timing code was injected into a simple sms application to test request response time of the Vancouver's Translink SMS Transit time service (Cell Number 33333) which serves as a suitable analog. A similar manual test (with stopwatch) was done on the popular IM apps, WhatsApp and SnapChat to see average media access times.

A user study to see how delays are perceived would have been preferred to validate whether Snap-Safe could be used as a viable alternative, however due to shortage of time we could not perform one.

7.2 Results

The results of the tests are provided in the figure 5. While most timings were closely grouped together, we noticed rare key retrieval instances (1 in 20) where the measured delay was uncharacteristically high.

7.3 Discussion of the Evaluation

The results show that apart from using SMS as a communication mechanism, key retrieval can almost always be done in lesser than 2.5 seconds, which we feel is a small price to pay for the added security benefit considering that

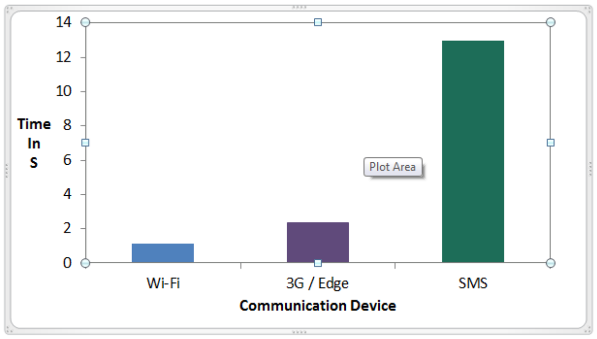

Figure 5: Results

in comparison, IM apps like SnapChat take around a second to display media as well. Furthermore, we believe that the rare instances of unexpected delay to retrieve keys were due to a disruption in the communication method, such as a loss of signal, re-obtaining a new IP after an expired lease from the local DHCP etc and was not due to the system itself. An aspect severely missing from the evaluation was server load. Since millions of users would be retrieving keys at any given time in an actual IM service, a single server would create a bottle neck to provide this key retrieval. Naturally, we feel that scaling would have to be performed by distributing the workload of the server among a cluster of suitable computers which could balance the requests and maintain the availability. Availability remains a challenge in this design because it can be very possible that the user may want to access old IM conversations while not having access to the network. Solutions in the form of Plausible Deniable Encryption exist to hide keys in seemingly plain sight however this technique is beyond the scope of this paper and is better explored in DEFY[14] and Mobilflage[13].

8. CONCLUSIONS

In this paper we presented Snap-Safe, which implements a live key fetch mechanism to control access to local data resources as well as provide security in the case of device being stolen.

We further provide ways of granting and revoking access to the same system might be used in various forms and adaptations to deploy the scheme in related and non related areas to provide fine grained access control to a set of objects.

We would like to acknowledge the solid foundation of work on top of which the current version of Kerberos stands today which, in this case, provides a solid foundation to base our work on.

Overall, the system has high applicability and we feel that it might fit well with the current architecture of Instant Messaging to benefit common users.

9. REFERENCES

[1] https://github.com/programa-stic/snapchat-decrypt.
[2] https://www.line.me/.
[3] https://www.snapchat.com/.
[4] https://www.viber.com/en.
[5] https://www.whatsapp.com/.
[6] Z. D. BOREN. There are officially more mobile devices than people in the world. 2014.

[7] T. Bradly. Study: Concern over mobile device theft on the rise. *PC World*, 2014.

[8] C. CLIFFORD. Top 10 apps for instant messaging. 2013.

[9] ConsumerReports. Smart phone thefts rose to 3.1 million last year, consumer reports finds.

[10] N. MOTT. Snapchat does what facebook couldnâĂŹt: Get publishers to trust it with their content. 2015.

[11] B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, 1994.

[12] D. Pingruber. simple-android-instant-messaging-application simple im application runs on android. 2012.

[13] A. Skillen and M. Mannan. Mobiflage: Deniable storage encryptionfor mobile devices. *Dependable and Secure Computing, IEEE Transactions on*, 11(3):224–237, May 2014.

[14] Z. N. J. P. Timothy M. Peters, Mark A. Gondree. Defy: A deniable, encrypted file system for log-structured storage. *NDSS*, 2015.