2014 978-1-4503-3001-5/14/125

Security and Privacy of Smartphone Messaging Applications[1]

Robin Mueller
Vienna University of Technology, Austria
e0926507@student.tuwien.ac.at

Sebastian Schrittwieser
St. Poelten University of Applied Sciences, Austria
sebastian.schrittwieser@fhstp.ac.at

Peter Fruehwirt
SBA Research, Austria
pfruehwirt@sba-research.org

Peter Kieseberg
SBA Research, Austria
pkieseberg@sba-research.org

Edgar Weippl
SBA Research, Austria
eweippl@sba-research.org

In recent years mobile messaging and VoIP applications for smartphones have seen a massive surge in popularity, which has also sparked the interest in research related to the security and privacy of these applications. Various security researchers and institutions have performed in-depth analyses of specific applications or vulnerabilities. This paper gives an overview of the status quo in terms of security for a number of selected applications in comparison to a previous evaluation conducted two years ago, as well as performing an analysis on some new applications. The evaluation methods mostly focus on known vulnerabilities in connection with authentication and validation mechanisms but also describe some newly identified attack vectors. The results show a predominantly positive trend for new applications, which are mostly being developed with robust security and privacy features, while some of the older applications have shown little to no progress in this regard or have even introduced new vulnerabilities in recent versions. In addition, this paper shows privacy implications of smartphone messaging that are not even solved by today's most sophisticated "secure" smartphone messaging applications, as well as discuss methods for protecting user privacy during the creation of the user network.

[1] This paper is an extended version of []

D.4.6 Security and ProtectionAccess controls Security, Experimentation Mobile Security, Smartphone Messengers, Transport Layer Encryption

1 Introduction & Related Work

With the ever increasing popularity of OTT (over-the-top) messaging in recent years and massively successful applications such as WhatsApp, Line and WeChat claiming to have active monthly user bases of up to 400 million users or more [13, 10, 12], large numbers of similar applications have emerged on the mobile app market trying to imitate those huge successes. In 2012 the number of messages sent over OTT networks had eclipsed the number of SMS messages, with researchers projecting OTT messages to exceed SMS by a factor of 4 by the year 2017 [18]. The fast growth and large number of available applications in a relatively young field naturally causes many of them being developed without sufficient security in mind. Schrittwieser et al. [15] and Cheng et al. [3] describe various attack scenarios and possible implications of security vulnerabilities related to these kinds of applications. Other research focused further on the consequences of vulnerabilities in those applications, e.g. privacy[17] or the system architecture [2]. The security functionality of smartphone operating systems are widely studied 5, 6, 7, 4, 8], app specific vulnerabilities further exist.

The goal of this paper is to follow up previous research by re-evaluating existing applications to show advances in the security field as well as examining newly emerged ones for known or potentially new vulnerability patterns. User authentication is a popular field of ongoing research 16, 1], especially in web services [9] and distributed systems [11]. We re-evaluate the security of the authentication system of mobile messaging apps two years after the publication of critical vulnerabilities.

As the number of OTT messaging applications is very large, we focus only on a subset of the available applications, based on the sample of previous evaluations as well as their install base.

2 Messaging Applications

Similar to the previous work of Schrittwieser et al. [15] we are focussing at applications that solely rely on the users' phone number in the verification process (in Section 5 we extend our research selcted other messengers that also support user accounts). Generally this means that a new user has to enter his phone number when registering an account. The application will use this number as a means of identifying the user. To prevent malicious attackers from simply entering arbitrary phone numbers to impersonate their target, most applications include a verification process to make sure that the entered phone number actually belongs to the user. The way this verification is done varies between applications, but it usually involves some kind of authentication token (in most cases this is simply a 4 to 6-digit number) being communicated between the server and the phone in a way that enables the server to establish the authenticity of the entered phone number. This is almost universally done through SMS, although the actual protocol can be vastly different in terms of implementation and security. Most applications will simply send a short verification code per SMS to the number that the user is trying to register which they then have to copy into the application in order to prove that they are actually the owner of the given phone number. The individual protocols and their identified flaws will be outlined in Section 4.

3 Evaluation

3.1 Methods

The actual evaluation consisted of two groups of applications - first we re-evaluated all of the applications that had previously been analyzed by [15] to check for any improvements, and then we looked for new applications that have emerged in the last two years and checked those for any vulnerabilities.

Table 1 lists all applications, their basic features and the estimated size of their user base. Whenever possible we used publicly available information from the application vendor, otherwise the user base was estimated from the numbers accumulated from the Google Play Store (which provides a rather wide range on the approximate number of Android installs) and Xyo[2] (a service that provides estimated download numbers for iPhone applications). The following section lists and shortly describes different vulnerabilities that the evaluated applications were tested against. The categories are based on [15].

3.2 Common Vulnerabilities

Authentication and Account Hijacking

Arguably the most dangerous class of vulnerabilities allows an attacker to take over a victim's account or impersonate it by circumventing the authentication mechanism of an application. Most applications prompt the user to enter their phone number first (some Android applications will extract the phone number automatically and ask the user to confirm its correctness) and then send a SMS to that number containing an (usually 4 to 6-digit) authentication code which the user has to enter. Some applications use different methods, which will be described in detail in the appropriate sections. We tested and analyzed the protocols used for identifying and linking the user's phone number to their account and attempted to circumvent them. Another related vulnerability deals with the unauthorized de-registration or deactivation of existing accounts - one instance of which has been identified during research.

Sender ID Spoofing/Message Manipulation

This vulnerability class deals with an attacker manipulating or forging messages and sender information without hijacking the entire account. This usually involves creating and sending messages with a fake (spoofed) sender ID by bypassing user-identification mechanisms inside the application. This class of vulnerabilities is rather uncommon and we were not able to

[2] http://xyo.net/iphone/, last accessed: 1st Oct. 2014

identify any affected applications. The applications that showed this sort of vulnerability in the past (according to [15]) have since been fixed or discontinued.

Unrequested SMS/Phone Calls

As most applications use passive SMS-based verification (and some even use passive phone calls) during sign-up, it is possible to generate unwanted messages or even phone calls to arbitrary phone numbers. Although most applications include mechanisms to prevent the sending of too many of such requests, combining multiple applications with an automated system could still generate considerable amounts of spam. It should be noted though that the content of those messages can generally not be modified which makes the concept less attractive for spammers.

Enumeration

Pretty much all applications allow the user to upload their phone book to identify other registered users. The server usually replies with a list of contacts that are also registered for the service. By uploading specific phone numbers an attacker can gain knowledge about whether the targeted person uses the service. This information can potentially be used for further attacks such as impersonation or spoofing attacks. In another scenario an attacker could systematically upload large amounts of different phone numbers to enumerate parts of the application userbase, for example uploading all possible numbers with a specific country code would give them an overview of all users in that country. This can potentially be a large privacy concern. For further reading see [3] where Cheng et al. have conducted rather extensive research on this particular issue.

3.3 Experimental Setup

For the actual research we used an iPhone 3GS running iOS 6.1.3 and a Samsung Galaxy S3 Mini running rooted Android 4.1.2. All tested applications were available for both iOS as well as Android and have been tested on both platforms. To read and modify the encrypted HTTPS traffic between the application and the server we used mitmproxy[3] – an SSL proxy and man-in-the-middle-tool for intercepting and modifying HTTP traffic on the fly. Furthermore, we used sslsplit[4] in a similar fashion to be able to read some of the SSL encrypted non-HTTP traffic.

(Version Android/iOS)	VoIP	Text Messages	Number Verification
eBuddy XMS (2.21.1/2.3.1)	no	yes	SMS, active SMS
EasyTalk (2.2.6/2.1.1)	yes	yes	SMS
Forfone (1.5.7/3.4.2)	yes	yes	SMS, active SMS
HeyTell (3.1.0.384/3.1.2.458)	yes	no	none
Tango (3.3.69998/3.3.71425)	yes	yes	SMS
Viber (4.1.1.10/4.1)	yes	yes	SMS

[3] http://mitmproxy.org/index.html, last accessed: 1st Oct. 2014
[4] http://www.roe.ch/SSLsplit, last accessed: 1st Oct. 2014

WhatsApp (2.11.152/2.11.7)	no	yes	SMS, passive phone call
fring (4.5.1.1/6.5.0)	yes	yes	SMS
GupShup (2.6/2.6)	no	yes	SMS
hike (2.6.16/2.4.1)	no	yes	SMS
JaxtrSMS (03.02.00/3.0.9)	no	yes	Active SMS, validation link, passive phone call
KakaoTalk (4.2.3/3.9.5)	yes	yes	SMS, passive phone call
Line (3.10.1/3.10.1)	yes	yes	SMS
textPlus (5.9.1.4671/5.4.0)	yes	yes	SMS
WeChat (5.0.3.1/5.1.0.6)	yes	yes	SMS

Table 1: Overview of messaging applications, 8 re-evaluated applications, followed by 9 new ones

(Version Android/iOS)	Phone Book Upload	Status Messages	Estimated User Base
eBuddy XMS (2.21.1/2.3.1)	yes	no	7.3-12.3M
EasyTalk (2.2.6/2.1.1)	yes	no	0.48-0.88M
Forfone (1.5.7/3.4.2)	yes	no	2.8-6.8M
HeyTell (3.1.0.384/3.1.2.458)	no	no	17.6-57.6M
Tango (3.3.69998/3.3.71425)	yes	no	110-510M
Viber (4.1.1.10/4.1)	yes	no	133-533M
WhatsApp (2.11.152/2.11.7)	yes	yes	350M
fring (4.5.1.1/6.5.0)	yes	no	29-69M
GupShup (2.6/2.6)	yes	yes	0.1-0.5M
hike (2.6.16/2.4.1)	yes	yes	5.3-10.3M
JaxtrSMS (03.02.00/3.0.9)	yes	no	0.9M-1.4M
KakaoTalk (4.2.3/3.9.5)	yes	no	58M-108M
Line (3.10.1/3.10.1)	yes	yes	300M
textPlus (5.9.1.4671/5.4.0)	yes	no	44-84M
WeChat (5.0.3.1/5.1.0.6)	yes	no	270M

Table 2: Continuation of the overview

4 Results

This section presents the results of the evaluation process based on the vulnerability categories described in Section 3. In general, we will limit ourself to mentioning applications with specific vulnerabilities or noteworthy findings. Table 2 provides a per-App overview of the vulnerabilities identified in the individual applications now and in 2012 (from [15]).

Application	Account Hijacking	Unrequested SMS	Enumeration	Other Vulnerabilities
eBuddy XMS	yes (no)	yes	Yes	no
EasyTalk	yes* (yes)	yes	Yes	no
Forfone	yes (no)	yes	yes	no (yes)
HeyTell	yes	no	limited	no
Tango	yes	yes	yes	no (yes)
Viber	no	yes	yes	no
WhatsApp	no (yes)	yes	yes	no (yes)
fring	no	yes	yes	no
GupShup	no	yes	yes	no
hike	no	yes	yes	no
JaxtrSMS	no*	yes	no	no
KakaoTalk	no	yes	yes	no
Line	no	yes	limited	no
textPlus	no	yes	yes	no
WeChat	no*	yes	limited	no

Table 3: Overview of vulnerabilities (in case of differences to [15], the old value is shown in parentheses)

* potential vulnerability, see details in the respective sections

4.1 Authentication and Account Hijacking

This section will describe practical and theoretical attacks against the analyzed applications that could be used to circumvent the authentication and validation process to allow an attacker to register a different person's phone number. Generally, this can be done by either using a new, not-yet-registered number or by hijacking an existing account.

eBuddy XMS

The XMS' authentication mechanism is very different between the Android and iOS versions and includes distinct weaknesses which will be described separately.

iOS The iOS version uses a simple SMS-based authentication approach where the device sends an authentication request to the server, which in turn sends a SMS message containing a random, 3-digit code to the registered phone number. The user then has to enter this code on the device which sends it to the server where the code is checked and the device is authenticated. While the protocol itself seems safe and does not allow circumventing the mechanism, the usage of a code of only 3 digits length is very alarming. Coupled with the fact that there appears to be no lockout when entering too many invalid codes and no time limit when entering them either, an attacker can reliably guess the code after an average of 500 tries. Increasing the code length and implementing a limit on the allowed number of attempts are basic measures for preventing brute forcing of access codes that should be present in every application that uses an authentication scheme such as this one.

Android For some reason the verification process in Android is very different from the iOS approach. Firstly, when registering a number for the first time the application will not attempt to validate it at all. Only when trying to register an already-registered number the application will attempt to do some form of SMS-based authentication. This is obviously a poor scheme, as it allows an attacker to impersonate arbitrary people, given that they have not registered for the XMS service yet. Combined with an enumeration attack (as described in later sections) to find out whether someone is using the service, this could be used to register someone without them ever knowing, as there will be no SMS traffic generated on a first-time-registration. Secondly, the verification process when registering an already known number is somewhat broken as well. The application locally generates a 10-digit authentication code and sends it via active SMS (text message charges apply) to the entered phone number. When used legitimately, this will result in the phone sending a text message to itself, which is then intercepted by the application and the code is verified locallyWhen entering a foreign number that person will receive a text message containing the verification code. Sending a reply message from that number including the received verification code should authenticate the device. While this scheme appears alright at first sight, we will describe a theoretical approach that could be used to exploit it.

The basic idea of the attack is to somehow gain access to the code inside the SMS (by reading the outgoing message) and then using some form of SMS sender spoofing mechanism to create a fake response message. This response message has to include the activation code and has to appear to be originating from the number the attacker is trying to register. The process is visualized in Figure 2. This requires two things: Firstly, intercepting the outgoing message with the code. The problem here is that in Android text messages sent through the messaging API from within applications will not show up in the normal SMS outbox. There might be a way to programmatically intercept or log the outgoing messages to retrieve the verification code or else the attacker could attempt to intercept the message at the hardware or carrier level. After obtaining the code, the attacker would have to use an SMS spoofer (there are various such services available on the internet, such as spoofsms [5]) to send a fake message which includes the code and has its sender set to the number the attacker is trying to register. This should make the application believe that the message actually originated from the entered number and it should complete the authentication process. While these approaches would potentially require rather sophisticated methods, they should be feasible as the entire authentication process happens locally.

[5] http://spoofsms.net, last accessed: 1st Oct. 2014

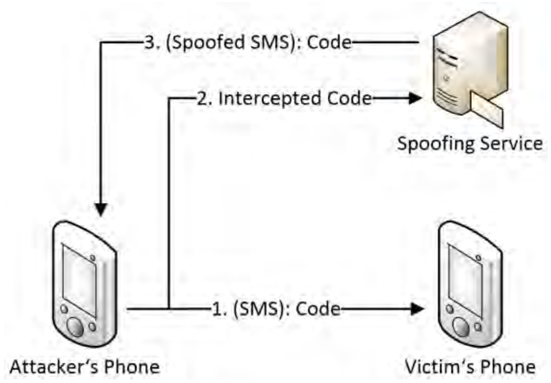


Figure 2: Theoretical exploit approach against XMS and JaxtrSMS

One positive aspect that stood out, was the fact that if someone registered a second account using a specific number, the owner of the original account would get a notification that someone else has registered another device with that number. That way the real owner would at least have an indication that something was wrong. In the end it seems surprising that the Android version would use such a vastly different and rather unusual authentication approach, when the iOS version uses a pretty simple and robust protocol (aside from the brute-force issue). One thing that all applications have in common is the fact that authentication is only as strong as its weakest version, so having a proper authentication mechanism on one platform is useless when one of the other platforms is susceptible to simple attacks, as an attacker can simply choose to use a device based on the easier-to-circumvent platform to carry out the attacks.

EasyTalk

Basically, EasyTalk uses a passive 4-digit SMS-based authentication scheme like many of the other applications. However, in our tests its authentication mechanism seemed not very reliable and on iOS the application simply crashed when started with an active proxy (even when in transparent proxy mode). On Android the verification process would simply get stuck most of the time when trying with an active proxy - without a proxy the process seemed to work, but the SMS with the code only really arrived in around 1 out of 20 attempts. During later testing the registration process stopped functioning entirely which made any further analysis virtually impossible. In [15] the authors describe an exploit that can be used to circumvent the authentication mechanism completely, but since the application did not function correctly in our analysis scenario it was not possible to verify the continued presence of this vulnerability.

Forfone

Forfone uses the same authentication mechanism on both platforms. However, it seems to have undergone significant changes compared to the way the mechanism was described in [15]. While the option to do a secure and well-implemented passive SMS-authentication is still there, it will only be used if the default authentication process fails. This default process is outlined in Figure 3 and works as follows:

The device generates a seemingly random "reference token" (a 32-digit hexadecimal number) which is sent to the server via HTTPS request. The server replies with a

HTTPS-response including an "authentication token" (another 32-digit hexadecimal number). The application then attempts to send this token using an active SMS from the phone to a Forfone service number. If the sending of the message is successful and the authentication token is correct, the account will be successfully registered using the received message's sender number. This means that the user does not enter the phone number at all during the process, but rather it is extracted from the message sent to the server. Only when the sending of the active SMS fails the application will revert to a passive SMS authentication scheme, where a common 4-digit code is sent to an user-provided number and then has to be entered manually. The entered code is then transmitted and verified server-side which is not susceptible to a simple impersonation attack.

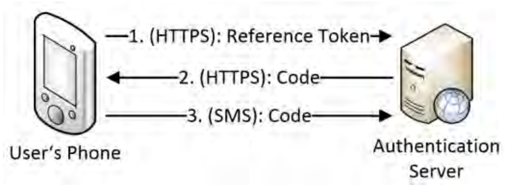


Figure 3: Forfone authentication during a legitimate attempt

The default authentication scheme on the other hand can be exploited quite easily as shown in Figure 4 (especially on iOS) - an attacker can simply copy the authentication token from the SMS before it is sent (since iOS requires the user to manually send the message themselves, all the application can and will do is open the SMS messaging app and pre-populate the recipient and message fields with the authentication code) or intercept the HTTPS response from the server and extract the token from there. After the attacker has obtained the token they need to create a spoofed SMS message which appears to be coming from the number they are trying to register and include the authentication token in that message. We used *spoofsms.net* for spoofing the sender ID which worked flawlessly in our mobile network.

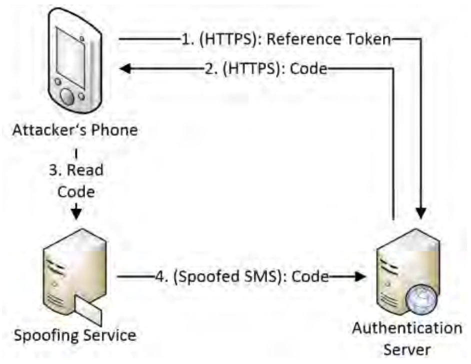


Figure 4: Spoofing attack against Forfone

It seems curious that Forfone would opt to use such an insecure validation mechanism as its default scheme (or at all) when it also features a secure, passive SMS mechanism. We would imagine this is done for price reasons, as active messages sent from the user's phone incur no cost to Forfone's operators, although this seems to be the wrong place to save costs seeing how it causes such a massive security flaw – especially when considering the low SMS messaging rates in most countries today.

HeyTell

HeyTell still does not have any sort of number verification whatsoever. A registrant can simply enter an arbitrary number along with a name when registering for the service. The system allows for multiple users to be registered using the same number. When another user attempts to add a phone number to their contacts, they will be presented with a choice of all users' names that are registered using that specific number. This system has two major ramifications: Impersonating someone who is not using the service yet is extremely easy due to the lack of any number verification. Hijacking an existing account on the other hand is not possible - users that already have someone's legitimate account in their contacts will continue to do so, all the attacker can do is to simply create a second account using the same number, so that anyone who attempts to add that number to their contacts from this point onward would be presented with two choices – the legitimate, and the fake one.

Tango

Tango's authentication mechanism appeared to be fundamentally broken - during early stages of research when doing some rudimentary testing we did get a validation SMS (4-digit PIN) when registering a device. However, when attempting to do further research at a later point the application did not attempt to do any sort of number verification whatsoever. We were able to freely change the phone number associated with an account without having to verify it at all.

Viber

Viber uses a 4-digit passive SMS authentication scheme which was not susceptible to traffic interception or other impersonation attacks. An example of such a scheme is outlined in Figure 5.

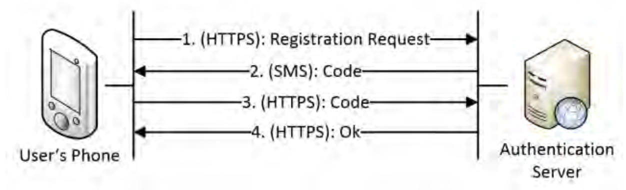


Figure 5: Secure authentication scheme as used by numerous applications (Viber, WhatsApp, fring, GupShup, hike, KakaoTalk, Line, textPlus and WeChat)

WhatsApp

WhatsApp completely re-hauled their authentication and messaging protocols since Schrittwieser et al. conducted their research [15]. The verification code (6 digits) is no longer sent to the device allowing for easy impersonation and hijacking, but rather the entered code is sent to the server and checked for validity there.

fring

Fring uses a 4-digit passive SMS authentication scheme which was not susceptible to traffic interception or other impersonation attacks.

GupShup

Similar to many of the other applications, GupShup also uses a well-implemented passive SMS authentication scheme (using a 6-digit code).

hike

Hike uses a 4-digit passive SMS authentication scheme which was not susceptible to traffic interception or other impersonation attacks.

JaxtrSMS

JaxtrSMS is another application that uses two entirely different and rather uncommon authentication schemes on both platforms. In addition to that, JaxtrSMS also supports passive call based verification for both platforms which becomes available after the default mechanism fails for some reason.

iOS The iOS authentication mechanism is essentially a passive SMS system as used by many other applications, with the difference that it does not send a verification code to the user but rather a verification link (as is often used in e-mail address verification). The user then has to open that link to activate their account. While this is a system not seen in any other app during research, it is essentially a tried-and-tested scheme that is usually used for verifying the e-mail addresses of newly registered accounts in virtually all online services, except that in this case the communication medium is SMS instead of e-mail. As such it was not susceptible to any traffic interception or other impersonation attacks.

Android The Android version implements a different authentication scheme and while we did not manage to exploit it in our tests, we cannot rule out the possibility of it being exploitable. It works as follows: After the user has entered the phone number the device will attempt to send a SMS message to the entered number containing a verification code. During legitimate use this would result in the application sending a message to itself, which is then intercepted and used to authenticate the user (similar to XMS, see Figure 1).

Now theoretically an attacker should be able to exploit this scheme by intercepting/reading the outgoing message and its code (for doing this see the eBuddy XMS section above, the same problems apply) and then creating a spoofed reply message which includes this code and appears to be coming from the target number (see Figure 2). In practice, this did not work for some reason though - we tried to register a second phone by simply sending the received authentication code back to the Android device, but the application ignored that SMS. We had no knowledge about the internal algorithm the application uses to do the authentication, but one possible reason for the attempt failing could be that it not only checks the sender number on the received message, but also the destination number. During a legitimate registration those two would be identical as the message is sent from the phone to itself, but when trying to impersonate another number with a spoofed message the target number will always be the number of the attacker’s phone. This is obviously just speculation though, further research would need to be conducted in order to establish whether or not the authentication scheme can actually be exploited.

KakaoTalk

KakaoTalk uses a 4-digit passive SMS authentication scheme which was not susceptible to traffic interception or other impersonation attacks. In case the SMS-based system fails the application also offers the option to do a passive call-based authentication.

Line

Line also uses a 4-digit passive SMS authentication scheme which was not susceptible to traffic interception or other impersonation attacks.

textPlus

textPlus also uses a 4-digit passive SMS authentication scheme which was not susceptible to traffic interception or other impersonation attacks.

WeChat

WeChat uses a classic 4-digit passive SMS authentication scheme, with the difference that after establishing the authenticity of the user's phone number it is possible to set a password in order to be able to log into the account from other devices. However, it is also possible to register the same number multiple times, effectively overwriting existing accounts under that number.

According to research done by Roberto Paleari, WeChat uses a custom communication protocol which is not based on typical HTTP/S but uses a combination of RSA for key exchange and subsequent AES for encrypting individual messages. A weakness in the application's debugging infrastructure allowed any application installed on the same Android device to extract a hash of the user's password. Detailed information on this exploit can be found on Paleari's blog [14].

4.2 Sender ID Spoofing/Message Manipulation

In this section we will discuss the evaluation of the applications' messaging protocols. We attempted to exploit the protocols in order to send unauthorized messages or messages with a spoofed sender ID. Most of the applications rely on the Extensible Messaging and Presence Protocol (XMPP) [6] for messaging and as such are not susceptible to sender ID spoofing. While a few of them use custom and mostly HTTPS based protocols such as JaxtrSMS and Forfone, even those applications included security features to prevent the sending of spoofed messages. Overall, we were not able to find any sender ID spoofing vulnerabilities in the analyzed applications.

Forfone

While according to Schrittwieser et al. [15] Forfone had contained a sender spoofing vulnerability, it appears to have been fixed since then. The application no longer uses the IMSI

[6] http://xmpp.org, last accessed: 1st Oct. 2014

or UDID for authenticating the sender but rather the randomly generated "reference token" as described in authentication hijacking section. While this makes message spoofing unfeasible, the other vulnerabilities described in the last section allow hijacking the entire Forfone account, arguably removing the necessity to create spoofed messages.

JaxtrSMS

The reason we wanted to mention JaxtrSMS at this point is because it follows a slightly different approach than most applications by being completely HTTP/S based - message sending is done through HTTPS requests and message receiving is done by periodically querying the server for any new messages. This simple protocol is secured by using a random user ID which is generated when a user signs up for the service. Every message sending request includes the recipient's phone number as well as the sender's user ID. This user ID appears to be secret and is known only to the server and the client itself and is used to authenticate the sender of the message.

4.3 Unrequested SMS/phone calls

Due to the nature of the authentication mechanisms of most applications it is possible to generate authentication requests for arbitrary phone numbers, which results in the system sending verification messages to the targeted number(s). An attacker could set up an automated system to generate lots of such requests to flood the target with spam messages. Although most applications include a limit of some sort on how often such requests can be sent, combining the authentication systems of multiple applications could still generate considerable amounts of spam. It should be noted though that it is not possible to change the contents of such an authentication message as it gets delivered directly from the service provider's infrastructure without possibilities for interception or modification. Therefore, such a system is pretty much unsuitable for commercial spammers and only useful as a disruption or annoyance. The exception being applications that rely on active authentication SMS sent from the registrants' phone to the targeted phone number. These messages are sent at the cost of the user and also have the user's phone number as the sender, which makes them unsuitable to be used as spam. Some applications such as WhatsApp, JaxtrSMS or KakaoTalk even allow for phone-call-based authentication, where the user receives a short phone call during which a computer-generated voice reads the verification code to the user. In case the phone call is missed, the system will speak the code onto the receivers message box. In all applications where call-based authentication is possible it only becomes available after the SMS-based authentication has failed. As opposed to most of the authentication messages which usually originated from the requesting country (or showed a spoofed sender) the origin of the phone calls usually was in the USA. We could imagine that generating numerous international calls in an automated fashion could cause considerable costs on the operators' part.

4.4 Enumeration

Most applications allow the user to upload their phone book to the server to automatically identify other users of the service. This can have various security implications as described in Section 3. The feasibility of such an attack was previously demonstrated in [15] by abusing WhatsApp's phone book uploading feature. By programmatically crafting custom HTTP requests that included ranges of phone numbers they were able to obtain information about whether the uploaded phone numbers were registered for WhatsApp. Almost all of the analyzed applications appear to be vulnerable to such an attack, although for some of them it might be harder to automate as they do not use HTTP requests for synchronizing the address book but custom (often TCP-based) protocols. While it should be possible to reverse-engineer these protocols and implement a rogue client to automatically upload phone numbers, it would potentially involve a lot of work. Furthermore some of the applications are either more cumbersome to enumerate (due to the way they work) or include privacy features that prevent individual users (if they had chosen the appropriate settings) from being identified by a mass-enumeration attack. Those special cases will be highlighted in the following section. Countermeasures for preventing enumeration attacks from being feasible have been proposed by Cheng et al. [3], but it is additionally advisable to impose a limit on the number of contacts that can be uploaded within a certain time period. Some of the analyzed applications might actually impose such limits, but attempting automated enumeration attacks against every single application to find out which ones do was out of scope for this project.

Forfone

We used Forfone as an example to demonstrate the feasibility of an enumeration attack due to its rather simple, HTTPS-based contact synchronization. The user simply has to upload a list of contacts using a POST request (this request is validated with the user's reference token, see Section 4.1 for details). The server responds with the same list, but for every contact entry it includes a flag that indicates whether that phone number is a registered Forfone user. Since Forfone does not limit the amount of requests that can be sent, we were able to enumerate arbitrary phone number ranges using a simple Java script (see appendix [sec:appendix]A) that automatically generates HTTPS requests and sends them to the Forfone server.

HeyTell

HeyTell allows users to change their privacy settings - using any setting other than "low" prevents random people from adding them to their friend list. In other words, people are unable to find out whether or not they are using the service (for example on "medium" only friends of friends are able to add them). This can prevent the enumeration of individual accounts, but most users will probably go with the default setting of being visible to everyone.

This feature does however include a weakness - if someone knows another person's user ID they can add them to their friend list regardless of their privacy setting by simply sending a crafted HTTPS-request with the target's ID as a POST parameter. While it does not seem possible to find out someone's user ID without them being on one's friend list, effectively preventing the "blind" adding or enumeration of random accounts, this flaw could be abused in other scenarios. For example, after blocking/ignoring an unwanted user and changing the privacy settings to prevent said user from finding or contacting again, that user could still be in possession of the blocking user's ID, create a new account and use the described vulnerability to add the blocking user again.

JaxtrSMS

JaxtrSMS does not identify users of the service beforehand – it only does so after someone attempts to send them a message. In case the recipient also uses the service, the message will be delivered through the applications network, otherwise an error message will be thrown. While this does not entirely prevent enumeration or identification of active users, it does prevent it from happening without the target user knowing. An attacker could still attempt to systematically send automatically generated messages to different numbers to enumerate users that way, although that would generate a lot of potentially unwanted traffic. While the application does not seem to utilize the user's contact list, for some reason it will still require the permission to upload it to the server - considering it is not used in any apparent fashion after being uploaded this seems like a totally unnecessary privacy intrusion.

Line

Line allows users to change their visibility settings – that means users can prevent other users from finding them using their phone number. While the default setting is to allow finding by phone number, the inclusion of such a feature is still a good step into the right direction. It is probably not going to prevent an attacker from enumerating large parts of the userbase, as most users would not bother to change their default privacy setting, but it gives privacy-conscious users the chance of staying hidden and avoiding being identified as Line users.

WeChat

Similar to Line, WeChat allows users to change their visibility setting to prevent others from being able to find them using only their phone number.

5 Privacy considerations of smartphone messaging

In this section, we further analyzed privacy considerations of today's generation of smartphone messengers in order to compare these with the applications outlined in the previous chapters and give a comparison on the respective security and privacy related issues.

5.1 Facebook Messenger

The Facebook Messenger[7] differs greatly from the previously discussed applications in that it does not use the phone number to identify the user, but requires registration and login. Furthermore, the list of contacts is populated with Facebook friends rather than numbers from the address book, still also this application allows importing contacts from a smartphone and storing them on Facebook's servers.

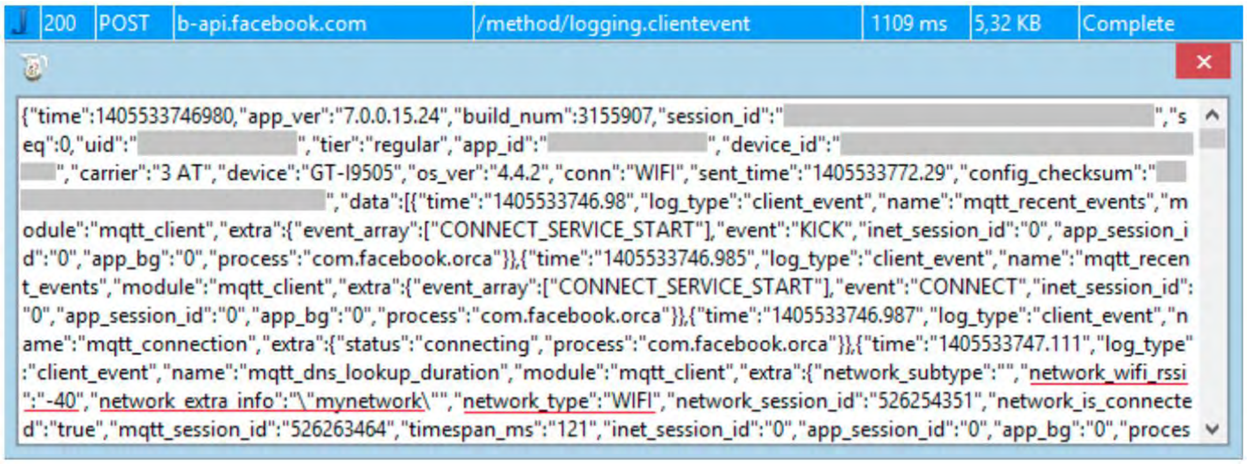
The messenger's communications are sent encrypted via HTTPS but can nevertheless be examined with the Charles Proxy. Our analysis revealed that a number of pieces of information are being transmitted in addition to the actual communication, including:

- date and time of communication

[7] https://www.facebook.com/mobile/messenger, last access 25 July 2014

- app version and build number
- network operator
- device information, such as the type of smartphone and the version of its OS
- network information: which type of network is being used (e.g. WIFI), the SSID[8] of the WIFI and its signal strength

The majority of these metadata is not necessary for the actual communication, so it is unclear why they are sent to Facebook's servers, especially network information, such as the SSID of the WIFI.
As this example shows, the SSID (in this case "mynetwork"), network type (in this case WIFI) and the RSSI[9] are transmitted in addition to standard information such as time of communication and version number of the application.



```
200   POST   b-api.facebook.com   /method/logging.clientevent   1109 ms   5,32 KB   Complete
```

```
{"time":1405533746980,"app_ver":"7.0.0.15.24","build_num":3155907,"session_id":"[redacted]","s
eq":0,"uid":"[redacted]","tier":"regular","app_id":"[redacted]","device_id":"[redacted]
","carrier":"3 AT","device":"GT-I9505","os_ver":"4.4.2","conn":"WIFI","sent_time":"1405533772.29","config_checksum":"[redacted]
","data":[{"time":"1405533746.98","log_type":"client_event","name":"mqtt_recent_events","m
odule":"mqtt_client","extra":{"event_array":["CONNECT_SERVICE_START"],"event":"KICK","inet_session_id":"0","app_session_i
d":"0","app_bg":"0","process":"com.facebook.orca"}},{"time":"1405533746.985","log_type":"client_event","name":"mqtt_recen
t_events","module":"mqtt_client","extra":{"event_array":["CONNECT_SERVICE_START"],"event":"CONNECT","inet_session_id":
"0","app_session_id":"0","app_bg":"0","process":"com.facebook.orca"}},{"time":"1405533746.987","log_type":"client_event","n
ame":"mqtt_connection","extra":{"status":"connecting","process":"com.facebook.orca"}},{"time":"1405533747.111","log_type"
:"client_event","name":"mqtt_dns_lookup_duration","module":"mqtt_client","extra":{"network_subtype":"","network_wifi_rssi
":"-40","network_extra_info":"\"mynetwork\"","network_type":"WIFI","network_session_id":"526254351","network_is_connecte
d":"true","mqtt_session_id":"526263464","timespan_ms":"121","inet_session_id":"0","app_session_id":"0","app_bg":"0","proces
```

Fig. 6: Facebook Messenger – transmission of information

5.2 XMPP through anonymization networks

One of the protocols that are used quite frequently with smartphone messengers is XMPP [11, pp. 1-18]. With XMPP, every user has a unique ID, which is constructed similarly to an e-mail address. Generally, we can distinguish between two types of communication with XMPP: client-server and server-server communication. When it comes to storing metadata, it is not sufficient to encrypt only the message or to use end-to-end encryption, as the XMPP servers would still have access to the information on who communicated with whom and when. If the system avoids using the phone number or e-mail address as identification, as in this case, it is of course harder to match a self-selected ID to a person. Nevertheless, it is theoretically possible via the service provider using IP addresses. The

[8] Service Set Identifier: For distinguishing between several WIFIs

[9] Received Signal Strength Indicator: Received field strength of wireless network

only way to prevent this is the use of anonymization networks such as Tor[10].This makes it impossible to determine the source or destination of a packet. Within the Tor network, there are hidden services that can only be reached via the Tor network itself, so that it is not possible to determine where the server providing the service is located. Should an XMPP service be operated as a hidden service inside Tor, only the server's operator could see who communicates with whom. Neither the Internet provider nor any other user could access this information. This means that a user would either have to trust the operator of the hidden service or operate their own XMPP server. However, the fundamental structure becomes slightly more difficult if this type of XMPP server has to communicate with non-anonymous XMPP servers, but even that would be possible. One of the disadvantages of anonymization networks such as Tor is that it takes longer to establish the connection. Marcel Heupel [12, p. 56] tested the usability of Tor for Android smartphones and found that it affected the speed considerably. It usually took approximately 1.5 seconds longer to establish a connection, and in some cases, it would take up to 20 seconds for a simple query to the server. One of the ways in which Tor can be used on the Android OS is the Orbot[11] application, which makes it possible to transport either the entire network traffic of the smartphone or only the traffic of specified applications via the Tor network. However, the unlimited use of this app requires root access on the smartphone. Without these rights, the application can only be used for specific other apps that support the use of a proxy. It should also be added that most smartphone messengers use the Google Cloud Messaging service. It delivers the message via a Google service, which hinders anonymization.

5.3 Google Cloud Messaging

Google Cloud Messaging for Android [13] is a free service by Google. It allows developers to send messages from their server to the smartphone app and to receive messages from the app. The message transmitted can either be the actual content of the communication or an empty packet. This so-called send-to-sync message can be useful when the server simply wants to inform the app about an event, e.g. that a new message is available for download. In this scenario, only an empty packet would have to be sent instead of the complete message. The service also makes it possible to deliver messages even when the app is not currently running on the smartphone. In this case, the send-to-sync message could "wake up" the app and tell it that a new message has arrived. Google only transmits the raw data, while any processing and displaying of the message must be done by the app itself. This service is used by many smartphone messengers on Android, as it is very comfortable for developers to use existing technology. Otherwise, the app would also have to query the server regularly to see whether new messages have arrived, which would of course influence the battery life of the smartphone considerably. By using Google Cloud

[10] https://www.torproject.org/, last access 25 July 2014

[11] Download at https://play.google.com/store/apps/details?id=org.torproject.android, last access 25 July 2014

Messaging, the server can inform the smartphone immediately that new data has arrived. The authors of the privacy preserving messenger application Threema[12] say that it sends an empty message via Google Cloud Messaging to inform the application of the existence of a new message. According to the developers, TextSecure[13] currently transmits the entire message via the service, but encrypts it. The app relies completely on Google to deliver it. Telegram[14] is the only app to use its own protocol, but says that this can lead to higher battery use and offers users the option of switching to Google Cloud Messaging.

6 Mitigation of phone book disclosure

The value [10] of a social network depends on its size, since most people would not join a social network if none of their friends were already using it. This creates a conundrum when building a social network – if nobody has joined yet, nobody will want to join. In order to avoid this problem when creating a new app, many developers decide to build on an existing social graph, such as Facebook. When the application is intended for a smartphone, however, there is another existing social graph that can be used – the user's address book. When a new app identifies users via their e-mail addresses or phone numbers - i.e., information that is already stored in address books – it can recognize quite easily which of the contacts already use the app. This means that users do not have to look for their friends, as they immediately show up as contacts. This mitigates the problem with the creation of a new social network as described above. The problem with this approach however is the following question: How does the app know which contacts already use the service? In most cases, the entire address book is sent to the server, where each contact is checked against an index of all phone numbers that use the service. But what options are there other than sending the entire address book to a server? The address book might contain sensitive information regarding contacts, or people may feel uncomfortable simply because all information is sent to some server.

6.1 Hash values

One option [10] that initially seems like an answer is not to use the phone number but its hash[15]. This way, it would not be necessary to transmit the individual contacts to the server, only their hashes, so that the server would have no access to the actual information, i.e., phone numbers or names and other personal details. The problem with this approach is the relatively low number of possible phone numbers. This means that it would be possible

[12] https://threema.ch, last access 26 February 2015

[13] https://whispersystems.org, last access 26 February 2015

[14] https://telegram.org, last access 26 February 2015

[15] A one-way function that can map a text of arbitrary length to a character string of fixed length. The hash cannot be reversed to gain plaintext.

to try all phone numbers in an acceptable time by brute force. It is not possible to use a salt in this case, as the other smartphones would have to use the same salt when sending their address books to the server. The number of possible e-mail addresses is considerably larger, but still not enormous.

6.2 Bloom filters

One strategy for mitigating the problem outlined in Section 6.1 would be the use of Bloom filters [10].The basic idea is that if the server were to send the entire database of registered users to the client, the client could verify the data locally without the need to query the server. Bloom filters could be used to optimize network efficiency in such a case. One problem, however, would be that the entire database could be read on the client. This can be avoided by using encrypted Bloom filters. The client could not simply search for a certain number in the Bloom filter, but would have to first request a blind signature from the server. This way the server would retain access control to the Bloom filter while not knowing what the client is searching for. The problem with Bloom filters in general, however, is the data volume that needs to be transmitted. As Marlinspike [10] writes on his blog, this approach cannot be used for the TextSecure messenger. If approximately 10 million users were to use the service and would update the bloom filter only once a day, this would still amount to 40 MB being queried from the server 116 times a second. This problem aside, a daily update of approx. 40 MB can still be quite costly, depending on the individual mobile contract of the respective user.

7 Conclusion

Generally speaking, the re-evaluation of the eight previously analyzed applications showed almost no improvement - while one of the flawed authentication mechanisms was fixed along with most of the other vulnerabilities present in the application (WhatsApp) and one completely broken application is off the market entirely (Voypi), new authentication weaknesses have been identified or introduced in both Forfone and XMS.

The newly evaluated applications on the other hand paint a much better picture: Virtually all of them use a seemingly well-implemented passive SMS authentication approach and with the exception of WeChat's logging vulnerability (as described in [14]) and a potential weakness in JaxtrSMS (which we were not able to exploit though) we could not identify any serious vulnerabilities. In regards to privacy and enumeration, two currently very popular applications (Line and WeChat) incorporate privacy settings that allow users to stay hidden from random people. This appears like a good privacy-preserving feature and the inclusion of similar mechanisms into some of the more popular messaging applications would be a desirable development for the near future.More privacy preserving smartphone messaging could be accomplished under certain conditions, but this would influence usability severely in most cases. The need to create one's own contact list, high battery use, or a slower connection, as with Tor, limit usability and therefore probably lead to lower user numbers compared to products such as WhatsApp, Viber and Facebook. Furthermore, in the last section of the paper we discussed several techniques for protecting the sensitive data inside the users' phone books while still allowing the application provider to establish a network between the individual users of the application. While currently no solution working in real-life environments has been found the pros and cons regarding the existing solutions can be seen as valuable starting points for future developments in this area.

References

[1] M. Bishop. Computer Security: Art and Science. Addison-Wesley, 2002.

[2] A. Braga. Integrated technologies for communication security on mobile devices. In MOBILITY 2013, The Third International Conference on Mobile Services, Resources, and Users, pages 47{51, 2013.

[3] Y. Cheng, L. Ying, S. Jiao, P. Su, and D. Feng. Bind your phone number with caution: automated user profiling through address book matching on smartphone. In Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, pages 335{340. ACM, 2013.

[4] L. Davi, A. Dmitrienko, A. Sadeghi, and M. Winandy. Privilege escalation attacks on android. Information Security, pages 346{360, 2011.

[5] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. Pios: Detecting privacy leaks in ios applications. In Network and Distributed System Security Symposium (NDSS), 2011.

[6] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A study of android application security. In Proc. of the 20th USENIX Security Symposium, 2011.

[7] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In Proceedings of the 16th ACM conference on Computer and communications security, pages 235{245. ACM, 2009.

[8] A. Felt, H. Wang, A. Moshchuk, S. Hanna, E. Chin, K. Greenwood, D. Wagner, D. Song, M. Finifter, J. Weinberger, et al. Permission re-delegation: Attacks and defenses. In 20th Usenix Security Symposium, San Fansisco, CA, 2011.

[9] K. Fu, E. Sit, K. Smith, and N. Feamster. Dos and Don'ts of Client Authentication on the Web. In Proceedings of the 10th conference on USENIX Security Symposium-Volume 10, pages 19{19. USENIX Association, 2001.

[10] J. Koum. 400 million stories. http://blog.whatsapp. com/index.php/2013/12/ 400-million-stories/, 2013. Accessed: 2014-08-04.

[11] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. ACM Transactions on Computer Systems (TOCS), 10(4):265{310, 1992.

[12] S. Millward. Line reveals latest user numbers in japan, thailand, taiwan, indonesia. http://www.techinasia.com/line-user-numbers-thailand-indonesia-japan-taiwan-august-2013/,

2013. Accessed: 2014-08-04.
[13] S. Millward. Tencent: Wechat now has 271.9 million monthly active users around the world. http://www.techinasia.com/tencent-wechat-272-million-activer-users-q3-2013/, 2013. Accessed: 2014-08-04.
[14] R. Paleari. A look at wechat security. http://blog.emaze.net/2013/09/a-look-at-wechat-security.html, 2013. Accessed: 2014-08-04.
[15] S. Schrittwieser, P. Fr • uhwirt, P. Kieseberg, M. Leithner, M. Mulazzani, M. Huber, and E. R. Weippl. Guess who's texting you? evaluating the security of smartphone messaging applications. In NDSS, 2012.
[16] W. Stallings. Cryptography and network security: principles and practice. Prentice Hall Press, 2010.
[17] P. Stirparo and I. Kounelis. The mobileak project: Forensics methodology for mobile application privacy assessment. In Internet Technology And Secured Transactions, 2012 International Conference for, pages 297{303. IEEE, 2012.
[18] K. Whitfield. 17 incredible facts about mobile messaging that you should know. http://www.portioresearch.com/en/blog/2013/17-incredible-facts-about-mobile-messaging-that-you-should-know.aspx, 2013. Accessed: 2014-08-04.