Security and Privacy Issues with Telegram

Richard Hernandez

2021 June 30

Abstract

Telegram, a platform lauded for its "security" and "privacy," contains enough flaws in both domains to disqualify it as a secure or private platform. The default settings reveal the user's private information, including their telephone number, and the default authentication method (SMS) leaves room for unauthorized access and a complete hijacking of a user's account. For all their claims to an open-source frontend, the backend (servers) are completely closed-source. Despite their claims to "heavily encrypt" user data in their servers, the server-side encryption processes reveals the plaintext contents of default ("cloud") conversations while each message is in transit. Further, the protocol (MTProto, version 2.0 at the time of writing) suffers from a wide variety of attacks, all of which are modeled based on Telegram's own documentation. These protocol-related issues undermine the security of "secret chats" (end-to-end encrypted). A user or administrator cannot remedy these issues because the server-side framework is completely closed-source. Ironically, although an organization's administrator cannot view the contents of users' cloud conversations, the servers convert that information to plaintext during transit. In lieu of Telegram, users and organizations should use a platform that implements secure protocols and enforces two-factor authentication. Security policies should reject a platform like Telegram for serious consideration for an organization's communication. Recommendations are provided for two applications — Signal and Microsoft Teams — which meet busi-

ness needs and security standards. Signal is recommended for individuals (for example, operating outside of an organization) who wish to communicate securely. Teams is acceptable for an organization where an administrator should be able to view the contents of conversations and implement organization-wide policies.

Introduction

Telegram gained a lot of popularity after big-data companies (like Facebook's Messenger or WhatsApp) started to dominate the market. Facebook's hacks and WhatsApp's implementation of end-to-end encryption left many individuals and organizations concerned about which platforms to use for secure, private communication (with an emphasis on security over privacy). Many people looked to Telegram as a secure, private alternative to these platforms. The platform cites a variety of reasons to use it: end-to-end encryption, server-side encryption, and ross-platform support. Recently, Telegram included video chats, a response to individuals' and businesses' needs during COVID restrictions.

Unfortunately, the security and privacy of Telegram is still very undesirable. Its security model offers no real advantages compared to other platforms. The only argument in favor of its privacy is the ability for users to browse with a custom "username," and the fact that Telegram does not serve advertisements to its userbase.[1] In the last few years, its weak security model (in particular, its encryption protocol) has been the focus of news headlines and peer-reviewed research.

Telegram's insecurities become immediately clear when assessing them on a scale of "default settings" (worst) to "hardened settings" (best):

- **Default settings**. Telegram's default settings allow for a session hijack, assuming an adversary can get either the SIM card or cellular network stream.

[1] To that end, it bears emphasizing that, despite their open-source client, their servers operate on a closed-source basis. Telegram is not obligated to report any information about their server-side processes. Thus, one cannot determine the extent to which Telegram may serve special interests.

- **Hardened settings**. A user has enabled the highest security and privacy settings in the app (thereby solving the issues noted with the "default settings"). Even then, an attacker could expose flaws in the security protocol: by "hacking" or "paying off" a backend administrator, and tampering with the encryption process. This would reveal the in-transit data between any two users' chats, including secret chats.

Analysis of the problem

This section analyzes several problems in Telegram's security model:

1. The issues with its default authentication model (SMS)
2. The vulnerabilities with its encryption protocol for data at-rest and in-transit (between chat clients)
3. The extent to which the user's privacy settings can actually keep the user private

Problems with the default authentication

Any client that uses the default settings (security and privacy) is vulnerable to unauthorized access and, from there, a total account hijack. Telegram's default authentication is just an SMS verification code, which it sends to the account's associated phone number. Note that SMS authentication is considered one of the least secure multifactor authentication methods [**?**].

Unauthorized access

First, an attacker needs to intercept the SMS message.[2] They can do this through a variety of means[3]:

[2] Attaining the SMS message requires reconnaissance on the attacker's part. Specific information on how an attacker might gain this knowledge is beyond the scope of this paper.

[3] This list is by no means exhaustive. The point is that cellular communication is inherently insecure. Thus, SMS-based authentication is also an insecure way to authenticate a user's identity.

- **A SIM-swap**. This is when the attacker physically removes and places the victim's SIM card into the attacker's own device [Puig]. If a carrier PIN has not been set, the attacker can log in using the same number, and the SMS verification is sent to the attacker's device.
- **Using a mock cellular access point**. This could allow attacker to perform a "man in the middle" attack. (Note that SMS sends messages in plaintext across cellular networks.) In this case, the attacker triggers the SMS authentication from their own device. As the SMS is sent to the victim's device, the attacker intercepts the message — and, thus, its contents. They can then use this code to log in on their own device.
- **Gaining physical access to the phone itself**. In this case, the attacker can trigger the SMS authentication through their own device, and read the authentication code directly on the victim's device. (They could even take it a step further and log-out the victim, or even delete the application from the victim's device.)
- **Gaining access to the phone number**. If the victim's phone number is returned to the ISP (for example, if they stop paying for the line). If the attacker knows this, they could either ascertain the number for themselves, or find the number's current owner and attempt to attack them before the (original) victim transfers their Telegram account to a new number.
- **Non-technical workarounds**. This includes social engineering. For example, a white-hat hacker was able to gain access to a victim's entire text message stream by lying to a third-party vendor who works for T-Mobile [**?**].[4]

To access the account, an attacker simply follows the application's login procedures: enters the victim's phone number; requests an SMS or phone-call verification; and enters the code in the app. A successful login redirects the attacker to the victim's chat history.

[4]The vendor, Sakari, and T-Mobile have taken steps to rectify this issue. It should also be noted that this issue arose because of the differences in phone-number authorization between the US and other countries.

With any of these methods, the attacker is now in the victim's account. They have full access to conversation history, saved contacts, and non-secret chats. They can now hijack the account.

Hijacking the account

The initial access relies on the victim *not* having two-factor authentication. Unsurprisingly, the hijack relies on this, too.

To *maintain control* of the account, the attacker can follow the application's steps to enable two-factor authentication ["Setting a new 2FA password" and "Email verification"]. They need only enter their own password and recovery email. Once these are enabled, the account will no longer provide SMS authentication. Further, if the victim tries to log in again, they will be asked for the attacker's credentials. The attacker controls the account.[5]

Issues with in-transit encryption

The "security" and "privacy" of Telegram suffers from well-deserved scrutiny. Telegram's security model relies on a home-made protocol, MTProto[6](version 2.0 at the time of writing). MTProto is responsible for encrypting chat conversations: end-to-end ("secret chats") and device-to-server ("cloud" conversations, the default) ["End to End Encryption"]. Unfortunately, the protocol's design makes the platform inherently insecure. By default, the servers act as a "man in the middle." To make matters worse, the organization describes their platform in a way that underplays the amount of privileges performed in their server-side processes. To further the issues, the protocol itself suffers a variety of security flaws.

Telegram's servers decrypt non-secret conversations on their servers.[7]On the

[5]The victim could take any number of measures to try and regain access. For example, they could attempt a "counter-hack" the attacker to acquire the attacker's password or email code. The effort required to do so is beyond the scope of this paper. The author advocates for users to set multifactor authentication so as to avoid the need to regain control of one's account.

[6]A detailed description of the protocol is outlined on their website [**?**].

other hand, secret conversations are not implemented in a way that is intuitive for users. The nature of MTProto implies that end-to-end encryption is possible for cloud chats as well, but Telegram does not implement this by default. Thus, the servers should be considered untrusted (for example, when sending or receiving sensitive information).

Cloud chats are insecure

Cloud chats are stored in Telegram's servers. A user can access these conversations from any device. This type suffers from the greatest amount of security and privacy issues.

The backend servers can decrypt normal (cloud) conversations at any time. In fact, for cloud chats, the server converts "encrypted" messages between clients: from the sender to the server, and from the server to the recipient. This is a feature of their protocol. For any conversation, Telegram's servers create at least three keys (one for the clients, and one for the server). In this way, Telegram's in-transit security resembles a man-in-the-middle. The servers can read — and, therefore, intercept — all traffic. It should therefore be considered as an untrusted source [Miculan, 2]. Paradoxically, a user must trust Telegram's servers for normal conversations, which are not end-to-end encrypted [Albrecht, 31].

Telegram claims to *store* conversations and resources in a way that is, in their words, "heavily encrypted" ["Cloud Chats"]. This is true of how they store conversations *at rest*. Nonetheless, the servers decrypt the content *in-transit*. In other words, the way its stored is less important than the way it's used: at some point, the server will see the conversation in the clear. To this end, "encryption" itself is less important than the *way* their encryption works.

Telegram tries to inspire confidence in their privacy model by noting that the

[7] Some sources report that still stores information in the clear [Greenberg]. Telegram officially claims that it encrypts data on its servers: "All data is stored heavily encrypted" ["Cloud Chats"]. Determining whether Telegram encrypts regular chats is outside the scope of this briefing, but it is certainly worth clarification in further research.

server keys are kept in “other data centers in different jurisdictions.” However, as noted, the server converts the conversation content to plaintext as a feature of the conversation itself. An adversary could therefore get the conversation in the clear using any number of means: through a cyberattack, legal force, or financial incentives. The ideal server would *not* be able to decrypt a conversation — ever. For Telegram, this is a feature, and a highly vulnerable one under the wrong conditions.

Secret chats are unintuitive

Telegram’s “secret chats” are designed to alleviate the aforementioned concerns.[8]However, these chat suffer their own issues. First, they are still bound to the flaws of the protocol. Second, they follow a different set of rules which affect the user experience. Secret chats are stored only on the device, not the server. A secret conversation can only be viewed on the mobile client, not the desktop client. When the application is uninstalled or reinstalled, the secret conversation disappears forever.

Although secret chats offer some privacy benefits, it nonetheless raises the question: if Telegram could store normal chats encrypted on their servers, without it hindering the protocol, why don’t they? Telegram has no formal comment to this end. Likewise, the author makes no speculation or assumptions about their reasoning. The question merely highlights the choice that Telegram imposes on its users: on one hand, you can access your chats from any device, but they follow a weak security model in-transit; or, you could use a more secure, private scheme, at the expense of never being able to access it outside of that device exclusively.[9]

[8]Secret chats are stored encrypted end-to-end: that is, only on the client devices. The process noted for normal chats is the same, with the exception that, when the server "decrypts" the chat between clients, the "decrypted" content is the ciphertext. It can only be converted to cleartext in each client. This article raises some concerns with the *way* Telegram implements its end to end encryption, and what it says about the way they store secret chats.

[9]Compare this to an application like Signal, which allows *all* chats to be linked to another, authorized device.

Protocol vulnerabilities (MTProto 2.0)

The MTProto implementation is the subject of a wide breadth of critiques. Studies have highlighted its vulnerabilities to a *large variety* of attacks. The implementation for cloud chats is the most vulnerable. In short, MTProto should raise some alarms. Any security benefits are moot compared to the privacy and security concerns which it introduces.

The following attacks are possible within the current implementation of MTProto:

- **Reordering messages**. Telegram's documentation clarifies that secret chats are protected against this, but it makes no mention as to whether cloud chats are also protected. This attack type would be most effective from client-to-server messages. The servers will accept messages sent prior to the timestamp of its most recent message in a given conversation. This could cause a reordering or a delay of messages [Albrecht, 9].
- **Deleting messages**. Messages sent from the sender or recipient can be "silently dropped."\footnote{Albrecht et al. note that MTProto does not prevent deletion, but such an attack is incredibly unlikely in practice. Receivers use other delivery mechanisms which make it harder for someone to identify "repeated encryptions." The weak point, msg_id, is enclosed in a fresh ciphertext with random padding. However, they note that Telegram's desktop app, as the sender, is a viable candidate for this kind of attack: that is, its padding length is based on the message length (and, therefore, more predictable).}This occurs when a client or server does not get acknowledgement of a message and tries to resend it. If an attacker can distinguish "repeated encryptions," they can continuously drop the same payload [Albrecht, 10].
- **Re-encrypting messages**. Messages not acknowledged within a certain timeframe are re-encrypted using the same header data and fresh random padding. The same state\footnote{In this context, "state" refers to the message ID and

message sequence number (\texttt{msg_id} and \texttt{msg_seq_no}) from the header information.} can be used for two different encryptions. This is intended as a mitigation against the deletion attack noted previously. However, this also results in multiple ciphertext messages which contain the same data. An attacker could use this information to distinguish an “un-acked” message from a new one [Albrecht, 9, 11].

- **Unknown key-share**. The unknown key share attack would happen for an existing conversation: between some victim and their original intended recipient.[10] An adversary could collude with the intended recipient of such a conversation in order to pull off the exploit (for example, if they “hack” or “pay off” that person). The rekeying process could generate keys for the attacker in such a way that the attacker can send information (messages) to the victim. Further, those messages would appear to come from the original intended recipient. The protocol provides no way (for example, forensic artifacts) for any party involved to “prove” that the exploit happened, let alone that it happened through collusion. Thus, MTProto would obfuscate the attacker and the person whom they paid to help them accomplish this goal. Either of the offending parties could claim they were “hacked” with no substantiating evidence one way or the other. [Miculan, 17].

The exact nature of what an attacker would do with this information is outside the scope of this paper. Further, the models used to determine the possibility of these attacks differed from trial to trial.[11] Nonetheless, each trial implemented MTProto as described in Telegram’s own documentation. Any issues with the results are, therefore, flaws in one of two domains: Telegram’s documentation or its implementation.

[10] At the time of writing, the authors note that this attack relies on a "theoretical" implementation. Yet, the importance of this finding is substantial. When considered alongside the other discovered vulnerabilities, the integrity of mtproto2 is seriously undermined.

[11] Albrect et al. tested the message reordering, deletion, and re-encryption attacks using the model outlined in their analysis. The "unknown key-share" attack was deemed possible in the model used by Miculan, but they emphasized that it was largely "theoretical." [Miculan, 17].

While a messenger's protocol is not the *only* vector for an attacker, it is nonetheless a crucial one. Weak protocols should be rejected unless absolutely necessary. The potential for these vulnerabilities calls MTProto's integrity into question. Other protocols — for example, the Signal protocol, or even TLS — offer all the alleged advantages of Telegram's protocol, but they contain none of the issues.

Limited space to enforce policies

With all this in mind, consider that the IT administrator has no control over Telegram's security or privacy policies. This leaves the entire security and privacy administration under the purview of two groups of people: Telegram's developers and server staff for the backend; and the user for the frontend (client-side settings) [Kaspersky]. This introduces serious questions about the integrity of data stored on Telegram's servers.

To emphasize the importance of such intervention, consider the issues with the server-side encryption and the chosen protocol. Someone with access to the servers, or who could implement a new protocol, could effectively remedy the aforementioned problems. Telegram could alleviate many server-side concerns by open-sourcing the cryptographic processes which occur on their servers. Protocol concerns could be alleviated by fully migrating to a rigorously-tested protocol, like TLS. Until Telegram makes such changes, it should not be considered an "encrypted" platform [Albrecht, 31]. Likewise, one should avoid calling Telegram a "secure" messenger.

Recommendations

This section outlines mitigating strategies and alternative messaging platforms. All suggestions are founded on zero-trust policies and solve the security issues introduced by Telegram. These can be used by the individual for their personal devices; or, it can be adopted by security policymakers in creating or revising an organiza-

tion's policies.

Mitigations

The organization's security policies should reflect the following standards:

Use a platform that enforces two-factor authentication.

- Enforce the use of platforms that can use an "authenticator app" or a physical security key in addition to a strong password.
- At the bare minimum, enforce email authentication codes.
- *Do not use an application that performs authentication with only SMS codes!*

If the application requires a phone number for the username, ensure that the number is in your posession.

- If a SIM card is required, set a carrier PIN to prevent SIM-swapping.
- If the phone number is virtual, ensure that the virtual provider uses two-factor authentication (for example, to access the dashboard). Enable the strongest multifactor authentication settings for that account.
- Monitor the status of each phone-account mapping. (That is, if a phone number has changed, update any apps which rely on this number as the username.)
- Delete any unused accounts (for example, if a user leaves the organization) which are associated with an organization's phone number.

Use platforms that secure user data.

- When possible, use platforms which use end-to-end encryption and a secure protocol. Base these decisions off peer-reviewed research, which highlight objective findings about the platform's security model.
- Use platforms which do not present opportunities for someone else to view message contents. End-to-end encryption should prevent anyone except the conversation's intended recipients from viewing chat resources or conversation history. For organizations, ensure that no one *outside the organization* can access this information.

- For organizations, use a platform that leaves the user no opportunities to reduce their own security or privacy settings: for example, platforms that employ strong default settings, or ones which rely on an administrator's policy and settings.

Alternative platforms

The following platforms should be recommended to the organization's users:

- **Signal**. The actual content is end-to-end encrypted (not stored on servers)[Mora], the PIN provides a second authentication factor ["Signal Pin"], and the Signal protocol has succeeded rigorous security tests [Miculan, 2]. Use Signal if *the organization's IT department does not need to view chat content*.
- **Microsoft Teams**. IT administrators can review and audit message content. They can also enforce multi-factor authentication for users [Microsoft]. Use Teams if *the organization's IT department must view chat content*.

References

Albrecht, Martin R. et al. "Four Attacks and a Proof for Telegram." *Security Analysis of Telegram (Symmetric Part).* 16 Jul. 2021. https://mtpsym.github.io/paper.pdf, pp. 1.

Brodkin, Jon. "$16 attack shows how easy carriers make it to intercept text messages." *Arstechnica.* 16 Mar. 2021. https://arstechnica.com/information-technology/2021/03/16-attack-let-hacker-intercept-a-t-mobile-users-text-messages/.

"End-to-End Encryption, Secret Chats." *Telegram*. 29 Jul. 2021. http://web.archive.org/web/20210126013030/https://core.telegram.org/api/end-to-end.

Greenberg, Andy. "Fleeing WhatsApp for Better Privacy? Don't Turn to Telegram." *Wired*. 27 Jan. 2021. https://www.wired.com/story/telegram-encryption-whatsapp-settings

Kaspersky Team. *Kaspersky Daily.* "Telegram security and privacy tips." 15 Jan. 2021. https://www.kaspersky.com/blog/telegram-privacy-security/38444/.

"Set up multifactor authentication." *Microsoft 365 Admin Help Center.* 12 Jul. 2021. https://docs.microsoft.com/en-us/microsoft-365/admin/security-and-compliance/set-up-multi-factor-authentication?view=o365-worldwide.

Miculan, Marino and Nicola Vitacolonna. *Automated Symbolic Verification of Telegram's MTProto 2.0.* Version 2, 30 Apr. 2021. https://arxiv.org/abs/2012.03141, pp. 1-17.

Mora, Justino. "Demystifying the Signal Protocol for End-to-End Encryption (E2EE)." *Medium.* 17 Aug. 2017. https://medium.com/@justinomora/demystifying-the-signal-protocol-for-end-to-end-encryption-e2ee-ad6a567e6cb4.

"Multifactor Authentication". *University of Southern Indiana.* 2021. https://www.usi.edu/it/multifactor-authentication/.

Puig, Alvaro. "SIM Swap Scams: How to Protect Yourself." *Federal Trade Commission: Consumer Information*. 23 Oct 2019. https://www.consumer.ftc.gov/blog/2019/10/sim-swap-scams-how-protect-yourself.

"Signal Pin." *Signal Support.* Accessed 21 Jul. 2021. https://support.signal.org/hc/en-us/articles/360007059792-Signal-PIN.

"Mobile Protocol: Detailed Description." *Telegram.* 29 Jul. 2021. http://web.archive.org/web/20210126200309/https://core.telegram.org/mtproto/description.

"Telegram Privacy Policy: Cloud Chats." *Telegram*. 14 Aug. 2018. https://telegram.org/privacy.

"Two-factor authentication." *Telegram.* Accessed 24 Jul. 2021. https://core.telegram.org/api/srp.