REPHRAIN

Protecting citizens online

Towards Human-Centric Endpoint Security

Jenny Blessing - University of Cambridge

Partha Das Chowdhury - University of Bristol

Maria Sameen - University of Bristol

Ross Anderson - University of Cambridge/University of Edinburgh

Joseph Gardiner - University of Bristol

Awais Rashid - University of Bristol

October 2023

UK Research and Innovation · University of Bristol · UCL · The University of Edinburgh · King's College London · University of Bath

Towards Human-Centric Endpoint Security

Jenny Blessing[2*], Partha Das Chowdhury[1*], Maria Sameen[1], Ross Anderson[2,3], Joseph Gardiner[1], and Awais Rashid[1]

[1] University of Bristol
[2] University of Cambridge
[3] University of Edinburgh

Abstract. In a survey of six widely used end-to-end encrypted messaging applications, we consider the post-compromise recovery process from the perspective of what security audit functions, if any, are in place to detect and recover from attacks. Our investigation reveals audit functions vary in the extent to which they rely on the end user. We argue developers should minimize dependence on users and view them as a residual, not primary, risk mitigation strategy. To provide robust communications security, E2EE applications need to avoid protocol designs that dump too much responsibility on naive users and instead make system components play an appropriate role.

1 Introduction

End-to-end encrypted (E2EE) messaging applications attract the attention of many adversarial entities interested in accessing communications and associated data. While it is in the interest of service providers to ensure that their applications can withstand common attacks, providers vary in which attacks they consider and their defenses against them. Users place a high degree of trust in these applications and regularly use them to send sensitive information, ranging from financial details of interest to generic attackers to private information relevant only to personal acquaintances and domestic partners. Law enforcement has long relied on widespread communications monitoring as a counterterrorism technique and has repeatedly proposed service providers implement mechanisms to circumvent the encryption [13,27].

In this paper, we investigate security *audit* functions in several widely used E2EE desktop client applications. We refer to the term auditing in the sense and meaning expressed by Christianson [8] to understand how E2EE messaging services communicate (in)security to users and what this says about the protective role they expect users to play. Christianson articulates auditing in a broad sense, describing mechanisms that go beyond flagging if something is wrong in system execution to present suggestions to revert the system back to a safe state. They consider three possible states of a system: permissible, possible and conceivable. "Permissible" states can be thought of as situations where the stated security

[*] These authors contributed equally to this work.

and privacy properties are satisfied, but where the system is still able to reach a separate set of possible states due to perturbations and/or adversarial behaviour. An audit function restores a (possibly breached) system back to a permissible state. The effectiveness of such an auditing notion lies in accounting for the possible states that an adversary can force a system to enter. Conceivable states refer to situations where a system can enter but are not possible according to its stated security policies. Security audit functions that can account for conceivable states, minimize the need to trust large parts of the system infrastructure.

We explore the post-compromise recovery process in six widely used E2EE client applications: Signal, WhatsApp, Element, Viber, Telegram and Wickr Me. Specifically, we explore what audit functions these applications use to detect, communicate, and recover from breaches, with a particular eye on their expectations of user involvement in the recovery process. We choose these six applications largely due to the diversity of their underlying protocols: WhatsApp and Signal are based on the Signal protocol [3], Viber [30] uses the Double Ratchet protocol, and Telegram [24] and Wickr Me [7] use bespoke messaging protocols. Element [11] likewise uses the Double Ratchet protocol and is the only decentralized system studied. We reached out to five of the six service providers[4] to better understand their thought processes around endpoint security and the role of the user; one provider consented to speak on the record. We obtained ethics approval from our respective institutions prior to conducting the interviews.

Our observations suggest that audit functions focus on the security of messages as they travel *between* endpoints, but fall short in ensuring security *at* the endpoints. Notifications to the user, when endpoints have potentially been compromised, are often confusing and ambiguously worded, if they are given at all. Most importantly, E2EE service providers have designed their systems with an expectation that each user will play an active role in verifying their own security, despite a large body of usable security research demonstrating that users are neither knowledgeable nor engaged enough to do so [29,31,2,12].

We propose that platforms should rethink underlying system and protocol designs, particularly around assumptions made with respect to additional devices. An effective audit mechanism must communicate clearly to the user when a compromise has occurred, and also take action to remedy whatever assumptions, if any, allowed the breach to occur.

2 Related Work

Prior work has demonstrated that E2EE messaging applications are widely vulnerable to mobile and desktop cloning when an adversary has temporary physical access to the device [1,9,10,16,6]. Such a scenario is increasingly common—for instance, law enforcement searches, managed systems, device repair [4], and proximity in domestic settings all allow 'legitimate' participants short-lived device access. An adversary is then able to copy either select files or the device's entire

[4] We were unable to make contact with Wickr Me.

file system to a separate, adversary-controlled machine. We analyse prior results to understand how E2EE messaging applications recover from breaches and communicate them to their users.

Specifically, Cremers et al. [10] showed that the use of the double-ratchet protocol does not by itself provide sufficient post-compromise security due to the way session handling is implemented in Signal messenger. A cloned mobile device can continue communicating as the victim without detection by the client. A similar study of E2EE messaging applications using the Signal protocol and other protocols showed that tolerance of non-malicious desynchronization makes the applications vulnerable to cloning attacks [9]. Albrecht et al. investigated the extent to which Matrix confidentiality and authentication guarantees persist in its prototype implementation Element [14], showing that a compromised home server can break fundamental confidentiality guarantees.

A recent study investigated the extent to which E2EE messaging service providers revisited their threat model while developing desktop clients to complement their existing mobile clients [6]. Chowdhury et al. experiment with short-lived adversarial access against the desktop clients of Signal, WhatsApp, Viber, Telegram, Element and WickrMe. They find that Signal, WhatsApp, and Telegram enable desktop cloning, while Element, Viber, and WickrMe have deployed technical mechanisms to detect and recover from potential breaches. In other words, some messaging applications scope a malicious insider in their threat model, while others do not.

3 Findings - Audit Mechanisms

All E2EE messaging applications have been designed to prevent or mitigate compromise at two points: as data is stored by the application service provider, and as data travels over the wire from one user to another. They differ substantially, however, in the ways they consider and communicate client security.

3.1 Account Compromise

Our analyses of prior work reveal distinct ways in which mobile applications and desktop clients detect, respond to, and communicate malicious behaviour.

1. *Breach Detection:* Cremers et al.[9] found that for most applications there are no explicit ways for users to detect a breach. The audit functions to recover are ineffective due to the fact that E2EE messaging applications trade strict synchronization for usability. Short-lived access to the desktop clients of Signal, WhatsApp and Telegram enables an adversary to copy files from one machine and set up access to a user's account on the attacker's machine. The ability to clone both mobile and desktop E2EE applications is clearly beyond the permissible state and breaks perfect forward secrecy. In short, there is no built-in audit mechanism to detect a breach of permissible state and revert back. A user needs to proactively detect any such breaches and

de-link cloned devices. In Signal's case, this design decision is a deliberate choice and has been previously discussed on Signal's community forums and GitHub issue tracker [20,21,22,23]. The desktop clients of Viber, WickrMe and Element do not depend on the user to detect breaches.

2. *Protocol Response:* The desktop clients of Viber, WickrMe, and Element each introduce technical mechanisms that prevent attackers from cloning accounts. Viber transfers its primary identity to any companion device that becomes active, making it obvious if an identity is copied or transferred. The assumption here is that the primary device is under the control of the legitimate account owner, whose client will initiate recovery when it notices any breach of the permissible state. Similarly, in WickrMe, any compromise is detected by associating each device with a device-specific identifier. Critically, there is no assumption of human involvement in this process. Element is not vulnerable to this type of compromise since keys are not exported from the device as they are not considered part of the application state [25]. However, Element suffers from leakage of communication metadata from simple cloning attacks such that an attacker can figure out the entities the victim communicated with, though they would not be able to read the content of the messages.

3. *Communicating Adversarial Account Access:* Here, we discuss what indications, if any, are given to a user when their account is cloned. Such notification is an industry standard for new account accesses (e.g., Gmail sends users an email warning of a suspicious login attempt when an account is accessed from a new device and/or IP address).

   In mobile messaging clients, Signal offers a somewhat ambiguous indication to the recipient by not decrypting the messages [9]. WhatsApp, Telegram and Wickr continue without any explicit indication to their users of a compromise. The documentation of Viber [30] claims that a red lock is shown to the user to represent an endpoint changing keys, but the experiments by Cremers et al. [9] report an absence of any visible indication to the user.

   The desktop clients of Signal and Telegram do not prevent the attack and do not clearly notify the user that a new device has been added to the account. Signal, for instance, displays a flashing yellow message to the user stating that there has been a network connection error. From a user's perspective, this can seem like a simple WiFi connectivity issue rather than a security breach. The Swiss messaging app Threema recently responded to a similar attack by introducing a warning message in cases where a new device begins using the same Threema ID [16,26]. While WhatsApp also does not prevent the attack, it does notify the legitimate account owner when cloning has occurred, stating plainly that another client instance is accessing the account. Since Viber, Wickr Me, and Element are designed in a way that prevents simple cloning attacks, their user interfaces give no indication that such an attack has occurred, abstracting the security details from the user. They rely primarily on technical audit mechanisms, not human ones.

3.2 Chat Compromise

The most straightforward way to compromise an end-to-end encrypted chat is to simply join the group, either through being invited to join by an existing group member or through being added by the service provider itself. The larger the group chat, the more difficult it is for an existing user in the group to know with whom they are communicating. There are two primary categories of warning messages a user receives relevant to chat membership: (1) when a new user has joined the chat and (2) when an existing user's keys have changed.

New member: Each of the E2EE applications surveyed has a different interface design around group membership. WhatsApp, for instance, by default displays only a newly added user's phone number, and displays the new user's WhatsApp name to an existing user only if the new user is a WhatsApp contact of the existing user or if the user has set a display name. For instance, a user in a WhatsApp group chat of several dozen people who are generally not contacts may see a message along the lines of "+44 4135 555111 added +44 4135 555222", which is sent as part of the group chat itself and can quickly get lost in the shuffle.

In Signal, Telegram, and WhatsApp, group admins control who can add or remove group members. Element's decentralized design gives home servers substantial control over group membership, enabling a malicious or compromised home server to indiscriminately add new users to a group chat [14]. Element has long shown a UI indicator when this occurs, displaying a red 'X' next to the room icon [15]. The current reliance on a UI indicator rather than a technical mechanism to prevent the attack, however, places the responsibility for verifying the security of their communications on the user rather than the application. Element has acknowledged as much both in our conversation with them and in their security advisory responding to this attack and is actively working on redesigning their model to include new users.

Key changes: When a user changes their public key, anyone with whom they're communicating receives a warning message stating that their contact's keys have changed. The problem is that this occurs each time a user gets a new mobile phone or simply deletes and re-downloads the messaging app, prompting all users to become desensitized to these types of warnings and liable to dismiss them altogether.

Although the vast majority of new user and key change messages are benign, their volume and ambiguity mask the potential for some to indicate a more sinister event. The quiet addition of an unwelcome participant, whether by the service provider or an ill-intentioned existing chat member, is not a theoretical concern: GCHQ has previously advocated for just such a system as a way for law enforcement to access encrypted communication [13].

Inundating users with warning messages is not a sound security strategy in response to so-called "ghost users" and other attacks. Rather, security researchers

and service providers should continue to explore protocol-driven approaches where the system design acknowledges and mitigates the potential of these sorts of compromises. Vasile et al. [28] previously proposed a framework for detecting a ghost user. Additional potential directions include a trust-on-first-use model, where new users are "proactively excluded" and assumed to be untrusted [15], or a single-hop transitive trust model where an existing group member would vouch for a new device in a room.

4 Discussion

Liability of true alerts: These findings raise the question of when it would be appropriate for messaging applications to be solely liable for detecting breaches without involving their users. Relying too heavily on the user can result in 'annoying' or false alerts, and so the experiments done by Cremers et al. et al. [9] show that most of the mobile messaging applications continue without any notification. Similar findings are reported in the context of desktop clients [6]. The threat models of most messaging applications assume that users will be able to take the responsibility of protecting their endpoints. This assumption seems to be misplaced, more so when the risks are also due to flawed implementations such as Signal's session handling [10] or Element's inadequate design [14]. Since prior work has indicated that the service providers do maintain a state between the communicating entities and their devices, a potential path forward can be tying the state information to the devices.

Usability & adversarial behavior evaluation: There are strong usability arguments in favour of a relaxed approach to synchronization between clients—but usability can facilitate adversarial behavior. For example, a recent study of mute buttons in video conferencing applications reveals that applications monitoring mute buttons send audio statistics to their telemetry servers. These statistics can in turn be used to infer user activities within their personal space [32]. In our conversations with Element, they similarly acknowledged that the approach of locking users out as a response to malicious home server attacks [14] will be a usability challenge.

Key management: E2EE messaging applications focus more on an eavesdropper threat model, leaving device security to the end user. However, this leaves users as the ones responsible for protecting the very artefacts they do not generate or have control over. The device-specific keys are not generated by the user, nor are they stored in a device specified by the user. E2EE messaging applications could consider giving more technically-savvy users greater control, such as allowing users to generate and manage their own root key pairs. Such a provision can be optional and would allow users to control the risks they are exposed to. For instance, Element [11] allows end users to store the keys in a location of their choice. Research into E2EE key management can look into leveraging protections such as hardware enclaves provided by iOS. E2EE applications can

also look for ways of automating key management, for instance by adopting key transparency. WhatsApp recently announced plans to roll out a publicly auditable key directory [19], a significant move that will hopefully prompt others to do the same.

Shift to technical audit mechanisms: The design of E2EE platforms demonstrates their expectation that end users will play an active role in securing their accounts and/or recovering from any breach. Specifically, the reliance on interface warnings reflects a misplaced faith that users will be able to understand and act on the message, an expectation which has been disproven in prior studies [18]. Instead, protocols should be designed in a manner that accepts responsibility for securing communication at all stages, shifting responsibility *away* from the user. To return to the auditing framework presented earlier, we should endeavour to shift from human audit mechanisms to technical audit mechanisms where possible. Involving the user in helping to manage residual risks (if any) requires accessible security communications and mechanisms [17]. Future research can explore how systems can be more inclusive by design [5].

5 Conclusion

Security audit functions of E2EE messaging applications largely depend on the user to protect endpoint devices and to recover from breaches, if any. This expectation can pose a real privacy hazard; misplaced assumptions coupled with flawed implementations exacerbate existing threats. The difference in existing audit functions among messaging applications is arguably due to differences in their target user base; for example, Element is focused more on enterprise users compared to WhatsApp. In many enterprise settings, every user is a potential threat compared to retail users. Thus, the synchronization requirements are strictly followed in the former and not the latter. This in turn impacts how the user interfaces and recovery mechanisms are designed. Our analysis shows that audit functions which minimize their reliance on the user to detect and recover from breaches perform better from a security standpoint than those that depend heavily on the user. User involvement, if any, should reflect users' real capabilities and interest in detecting and acting upon threats, not systems designers' idealized vision of user understanding.

Acknowledgements

- We thank Bruce Christianson for the discussions and feedback reflected in the paper.
- This University of Bristol team is supported by REPHRAIN: National Research centre on Privacy, Harm Reduction and Adversarial Influence online (EPSRC Grant: EP/V011189/1).

References

1. Vitor Ventura: in(Secure) messaging apps — How side-channel attacks can compromise privacy in WhatsApp, Telegram, and Signal. https://blog.talosintelligence.com/2018/12/secureim.html
2. Akgul, O., Bai, W., Das, S., Mazurek, M.L.: Evaluating {In-Workflow} messages for improving mental models of {End-to-End} encryption. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 447–464 (2021)
3. BBC: Moxie Marlinspike leaves encrypted-messaging app Signal. https://www.bbc.co.uk/news/technology-59937614
4. Ceci, J., Stegman, J., Khan, H.: No privacy in the electronics repair industry. arXiv preprint arXiv:2211.05824 (2022)
5. Chowdhury, P.D., Hernández, A.D., Ramokapane, M., Rashid, A.: From utility to capability: A new paradigm to conceptualize and develop inclusive pets. In: New Security Paradigms Workshop. Association for Computing Machinery (ACM) (2022)
6. Chowdhury, P.D., Sameen, M., Blessing, J., Boucher, N., Gardiner, J., Burrows, T., Anderson, R., Rashid, A.: Threat models over space and time: A case study of e2ee messaging applications. arXiv preprint arXiv:2301.05653 (2023)
7. Chris Howell, Tom Leavy, Joël Alwen: Wickr Messaging Protocol Technical Paper. https://wickr.com/wp-content/uploads/2019/12/WhitePaper_WickrMessagingProtocol.pdf
8. Christianson, B.: Auditing against impossible abstractions. In: International Workshop on Security Protocols. pp. 60–64. Springer (1999)
9. Cremers, C., Fairoze, J., Kiesl, B., Naska, A.: Clone detection in secure messaging: improving post-compromise security in practice. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1481–1495 (2020)
10. Cremers, C., Jacomme, C., Naska, A.: Formal analysis of session-handling in secure messaging: Lifting security from sessions to conversations. In: Usenix Security (2023)
11. Element: Matrix Specification. https://element.io/enterprise/end-to-end-encryption
12. Hu, H., Wang, G.: {End-to-End} measurements of email spoofing attacks. In: 27th USENIX Security Symposium (USENIX Security 18). pp. 1095–1112 (2018)
13. Ian Levy and Crispin Robinson: Principles for a More Informed Exceptional Access Debate. https://www.lawfareblog.com/principles-more-informed-exceptional-access-debate
14. Martin R. Albrecht, Sofia Celi, Benjamin Dowling, Daniel Jones: Practically-exploitable Cryptographic Vulnerabilities in Matrix. https://nebuchadnezzar-megolm.github.io/static/paper.pdf
15. Matrix: Upgrade now to address E2EE vulnerabilities in matrix-js-sdk, matrix-ios-sdk and matrix-android-sdk2. https://matrix.org/blog/2022/09/28/upgrade-now-to-address-encryption-vulns-in-matrix-sdks-and-clients
16. Paterson, K.G., Scarlata, M., Truong, K.T.: Three lessons from threema: Analysis of a secure messenger
17. Renaud, K., Coles-Kemp, L.: Accessible and inclusive cyber security: a nuanced and complex challenge. SN Computer Science **3**(5), 1–14 (2022)
18. Sasse, A.: Scaring and bullying people into security won't work. IEEE Security & Privacy **13**(3), 80–83 (2015)

19. Sean Lawlor and Kevin Lewi: Deploying key transparency at WhatsApp. https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/
20. Signal Community Forum: Vulnerabilities. https://community.signalusers.org/t/vulnerabilities/4548/7
21. Signal-Desktop GitHub: Add option to lock the application. https://github.com/signalapp/Signal-Desktop/issues/452#issuecomment-162622211
22. Signal-Desktop GitHub: All exported data (messages + attachments) are *NOT* encrypted on Disk during (and after) the upgrade process! https://github.com/signalapp/Signal-Desktop/issues/2815#issuecomment-433556965
23. Signal-Desktop GitHub: based upon Kevinsbranch encrypted key in config.json using cryptojs && start performance fix. https://github.com/signalapp/Signal-Desktop/pull/5465#issuecomment-923300524
24. Telegram: MTProto Mobile Protocol. https://core.telegram.org/mtproto/description
25. The Matrix.org Foundation: "Client-Server API (unstable), May 2021". https://spec.matrix.org/unstable/client-server-api/
26. Threema: Version history. https://threema.ch/en/versionhistory
27. UK Parliament: Online Safety Bill. https://bills.parliament.uk/bills/3137
28. Vasile, D.A., Kleppmann, M., Thomas, D.R., Beresford, A.R.: Ghost trace on the wire? using key evidence for informed decisions. In: Security Protocols XXVII: 27th International Workshop, Cambridge, UK, April 10–12, 2019, Revised Selected Papers 27. pp. 245–257. Springer (2020)
29. Vaziripour, E., Wu, J., O'Neill, M., Whitehead, J., Heidbrink, S., Seamons, K., Zappala, D.: Is that you, alice? a usability study of the authentication ceremony of secure messaging applications. In: Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017). pp. 29–47 (2017)
30. Viber: Viber Encryption Overview. https://www.viber.com/app/uploads/viber-encryption-overview.pdf
31. Wu, J., Gattrell, C., Howard, D., Tyler, J., Vaziripour, E., Zappala, D., Seamons, K.: " something isn't secure, but i'm not sure how that translates into a problem": Promoting autonomy by designing for understanding in signal. In: Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019). pp. 137–153 (2019)
32. Yang, Y., West, J., Thiruvathukal, G.K., Klingensmith, N., Fawaz, K.: Are you really muted?: A privacy analysis of mute buttons in video conferencing apps. Proceedings on Privacy Enhancing Technologies **3**, 373–393 (2022)