A Black-Box Privacy Analysis of Messaging Service Providers' Chat Message Processing

Robin Kirchner*
Technische Universität Braunschweig
Braunschweig, Germany

Simon Koch
Technische Universität Braunschweig
Braunschweig, Germany

Noah Kamangar
Technische Universität Braunschweig
Braunschweig, Germany

David Klein
Technische Universität Braunschweig
Braunschweig, Germany

Martin Johns
Technische Universität Braunschweig
Braunschweig, Germany

ABSTRACT

Online messaging has rapidly emerged as today's primary communication platform, extending from personal, to business and even to government channels. *But can these services be trusted to maintain the privacy of your communication?* This paper addresses this question by evaluating 105 different online messaging platforms.

Utilizing "*honey*" messages and active HTTP(S), WebSocket, and WebRTC traffic monitoring, along with continuous observation of honey token access, we determine which messaging services process user messages beyond mere transmission. We conduct a large-scale honey token-based study on 69 popular web and 36 mobile messaging applications. Our findings reveal that 34 % of messaging services show capabilities of server-side message analysis. Seven of these messengers evidently conduct an extended analysis of the messages, reusing the results hours to an observed maximum of a month after the chat concluded. This shows that one cannot automatically expect the same confidentiality when chatting via messengers compared to in-person communication.

KEYWORDS

Messenger privacy, privacy assessment, honey tokens.

1 INTRODUCTION

How much trust can we put into the confidentiality of our chat messages? Chat applications for person-to-person communication are ubiquitous, dominating the communication space. They are frequently replacing in-person communication. While in-person communication comes with clear confidentiality assumptions, including the absence of third-party surveillance and analytics. However, it is questionable whether those assumptions transfer to digital communication. This question becomes even more pressing when realizing that the usage of chat applications even extends into governmental channels. For example, the UK government puts sufficient trust in the security of their communication over WhatsApp that governmental topics are discussed via instant messaging [71].

*Corresponding author: Robin Kirchner, robin.kirchner@tu-braunschweig.de.

Proceedings on Privacy Enhancing Technologies 2024(3), 674–691

https://doi.org/10.56553/popets-2024-0099

Past work on mobile applications indicates that common assumptions about the privacy provided by applications do not match expectations [44, 61]. Regarding mobile apps, past measurements have shown that data collection even extends beyond public declarations and might be in breach of data protection regulations such as the GDPR [42, 43, 62]. Measurements for personal computer and web-based applications paint a similar picture [1, 75].

Given this mismatch between expectations and reality, the question arises if our chats are as private as we expect them to be. Up until now, this question has been addressed by studying the cryptographic properties of secure messaging [e.g., 4, 5, 8, 15, 72, 98]. While this allows to uncover flaws in the underlying cryptographic implementation, it is a manual approach and does not scale for dozens of services. To overcome this scaling issue, we propose a black-box, honey message-based approach.

By sending honey messages containing hard-to-guess monitored URLs, we are able to detect whether a messaging provider processes the content of chats to extract and analyze links. This allows us to reason about a lower bound on how privacy-invasive common messengers are. Despite having no insight into their implementation, any server-side visitation of our links provides clear evidence for ongoing message analysis. Furthermore, we are able to monitor our *"honeypages"* over an extended time period and can thus reason on whether the results of the message analysis are short-lived or kept and (re)used later, indicating persistent storage of the analysis results. We regard only security measures, spam protection, and link previews as explainable goals for such analysis.

In the same fashion, we also conduct our study on websites that deploy chats powered by messaging service providers (MSPs) which are often automated chats with companies. While support chats do not necessarily carry the same privacy assumptions as communication between private friends, we aim to answer whether users can reasonably expect to be talking to company employees, like they would in a phone support call, or if a third party analyzes their messages.

This leads to three research questions we answer in this work:

RQ1. Can we define indicators showing us that private messages are analyzed by messaging services?
RQ2. Can we detect who analyzes the messages and find reasons for such practices?
RQ3. Can users of mobile applications expect more privacy than web users?

To answer these research questions, we make the following four technical contributions:

- We develop the *honey messages* framework providing and managing monitored honey pages to enable token-based studies.
- In addition, to enable monitoring of web-based messengers, we present an augmented browser-based testing approach for client-side analysis of intercepted HTTP(S), Web Socket, and WebRTC traffic to detect client-side leaks, using string similarity matching with precomputed token-encodings.
- We conduct a study on 105 commercial messaging services, comprised of 36 apps, 28 messaging service provider (MSP) chats, 41 websites with chat functionality, including web messengers, dating platforms and social networks.
- We found evidence for message analysis for 58 % of the apps, 49 % of the web messengers, and 14 % of the MSP service chats. Notably, in an *extreme* case a messenger crawled links causing nearly 80 server-side requests (SSRs); and another accessed our page 30 days after the chat.

Our paper is structured as follows: First, we present background information about message processing and reasons for such practices in Section 2. Then, we establish our methodology in Section 3, where we also present our implementation of the honey messages framework (ref. 3.2) and experiment approach. We detail our study design, analysis approach and evaluation in Section 4. In Section 5, we interpret and discuss our results by defining acceptable messenger behavior (ref. 5.1) and giving details about specific conduct, before presenting our key takeaways (ref. 5.8). We cover related work in Section 6, and finally conclude our paper in Section 7.

2 BACKGROUND

People having private conversations consider the content to be private and kept—within expectations—confidential. Meanwhile, in the digital world, the number of messenger users is steadily increasing [10]. In this section, we explain how the expectation of privacy carries over into the digital world (ref. 2.1) and where, during message transmission, this assumption might be violated (ref. 2.2), before addressing reasons for the violation (ref. 2.3).

2.1 The Assumption of Confidentiality

To understand if chat messages are analyzed, we selected a set of two different chat models—*Messenger Chats* and *Service Chats*. We argue that either carries its own assumption of confidentiality, which is violated if messages are automatically analyzed and their content processed beyond transmission.

2.1.1 Messenger Chats. A user study of German teenage chat app users showed that they expect their chat conversations to be private between the involved parties [74] and Amnesty International created a privacy score ranking for chat service providers in 2021, indicating that Amnesty perceived an educational gap between the privacy afforded by a service versus the expectations by the users [37]. Either demonstrating that an assumption of confidentiality exists similar to in-person conversations. Consequently, we model the user to expect the same privacy when using a chat app to converse with a friend, acquaintance, or relative as if the conversation would happen in person. This implies that any third-party access and analysis, especially if this involves the storage of data, is in violation of this assumption.

2.1.2 Service Chats. Talking to a store clerk or service representative does not carry the same privacy assumptions as talking to a personal friend. However, we assert that for privacy-conscious users there are still privacy assumptions that carry over to the digital realm, similar to personal messages. Those exclude a permanent record of the conversation and later analysis of the conversations. This assumption is also backed by the GDPR [32] at least for hotlines. Companies even acknowledge this, as a common preamble to being connected to a representative is the question of whether the call may be monitored for analyzing the quality of the provided service. Consequently, we model the privacy-conscious user to expect the same privacy when using a service chat as if the conversation would happen in person or via the telephone, implying no analysis and permanent storage of message elements.

2.2 Analyzing Chat Messages

Transmitting a chat message commonly involves three steps: (1) sender writes a message and sends it to the server; (2) server receives the message and forwards it to the receiver; (3) the receiver receives the message and displays it to the user (ref. Figure 1). Each step carries its own potential for message analysis. The sender and receiver-client necessarily have full access to the message contents as they need to send it to the server (sender) or receive it and display the content to the user (receiver). Consequently, the application used could perform further processing besides their core task of sending and receiving. The server has access to the content of the message in case of incomplete encryption, i.e., if only the transmission of the message between sender and server or server and receiver is encrypted but not the message itself. While E2EE is highly desirable from a privacy standpoint, it brings its own challenges, which may be one of the reasons why not every chat service has this feature enabled by default [e.g., 24, 91] or even at all [e.g., 18, 48]. Consequently, there is potential to perform further processing on the server, going beyond the core task of relaying the message.

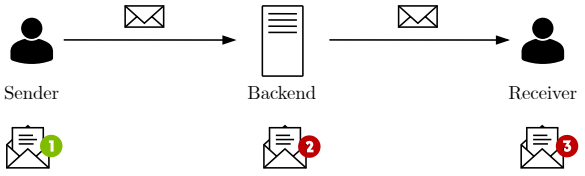

Figure 1: Points of potential message processing in a transport-encrypted setting.

2.3 Reasons for Analyzing Messages

Now that we have answered *where* messages might be accessed, in order to be able to answer RQ2, we need to consider *why* such access might happen. In our context of presumed private communication, a user can reasonably expect that their messages are confidential but might concede some confidentially in favor of security and usability. Overall we have identified two scenarios in which message analysis arguably takes place for the benefit of the user (*Spam, Scam & Security*, *Link Preview*) as well as two scenarios of analysis which are not beneficial to the user (*Surveillance*, *Business Incentives*).

2.3.1 Spam, Scam & Security. Analyzing chat messages could allow a messaging service to detect malicious patterns such as spam, scam, or even phishing campaigns. Given the prevalence of such campaigns on popular messengers such as WhatsApp [68] and their documented financial impact [25], the drive for such features is clear. It may also be reasonable to assume that some users would be willing to trade some privacy in favor of protection against scam.

However, any security check has a strict logical time limit, as the protective action has to happen before the receiving app displays the message. At the moment the message is received and read, the security mechanism is too late as the victim has possibly already been exploited. Consequently, any message processing has to happen instantaneously and should be without any permanent records if the message passes the check.

2.3.2 Link Previews. Link previews are a common convenience feature in messaging apps. They allow the user to gain an intuition about the content of the website they just received or sent. OpenGraph [64] and Twitter [93] define a semi-standardized set of meta-tag elements for this purpose.

However, previews need to occur at the time of sending or receiving a message to provide the user any benefits. Consequently, message processing has to happen instantaneously in these situations and thus does not require a permanent record.

2.3.3 Surveillance. There is also evidence that companies and government organizations are collecting user data with the help of messaging apps [22, 39, 66, 94]. This eavesdropping on chat communication can be part of criminal investigations—or—espionage. Past cases have shown that services deliver information about conversations when forced to [84], showing both the capability as well as volition to access chat messages beyond the scope of improving the security or experience of the user. Such an analysis of chat messages involves storing messages as well as analysis results and records can potentially be kept indefinitely.

2.3.4 Business Incentives. While the previous reasons generally offer an advantage for the user, monetary interests conflicting with the users' privacy cannot be ruled out. If the technical possibility exists, there would clearly be an incentive to gather users' personal thoughts and interests, e.g., for targeted advertising or to sell information to data brokers. Such an analysis would clearly exceed the scope a chat user expects.

3 BLACK-BOX PRIVACY ASSESSMENT

We covered that provider-level message analysis is theoretically possible and discussed potential reasons, the messengers' behavior in terms of permanent storage, and privacy implications. In this section, we first establish a high-level methodology capable of detecting message analysis before describing our implementation and experimental approach.

3.1 Methodology

Section 2.2 stated that user messages can be analyzed and processed in various stages of a message's transmission cycle. Unfortunately, most of the situations in which messages can be processed are not under our control. Specifically, the most privacy invasive side of such a process, the server side, is out of our grasp. To address this shortcoming, we propose an approach to detect server-side access to user messages' content, detailed in this section.

Messengers have various claims about their security measures. Verifying these claims and cryptographic properties would require a disproportional effort out-of-scope for this work. We design our methodology to be oblivious to the messengers' stated security protocol, thus independent of promises made by the vendor.

3.1.1 Behavioral Assessment. As static analysis is complex and possibly non transferable across messengers, we opt for a dynamic approach. We transmit specifically crafted messages containing honey tokens that we subsequently monitor. If access to the tokens inside a message is detected, it implies that at least parts of that message must have been extracted and undergone processing by the messaging application or service. This approach allows us to focus on observable behavior and provides proof for message access.

To complement the token-based approach, we deploy traffic interception for the mobile apps, as well as the website-based chats allowing us to further study their behavior based on recorded traffic.

In summary, we aim to transmit chat messages including traceable elements. These elements have to be enticing for extraction and analysis so that we subsequently can observe when they are further processed. In combination with the network traffic interception, we provide two-dimensional monitoring: we can observe access to the honey tokens from the client, backend, and receiver, as well as observe client-side leaks through the network.

3.2 Honey Messages Framework

We implement the core of our methodology as the *Honey Messages Framework*, forming the foundation of our black-box messenger study. The fundamental idea is to send unique URLs (honeypages) and email addresses as honey tokens via the chosen message providers pointing back to a web server controlled by us. The web server acts as a honeypot and is the key component of the framework. It is equipped with a REST API to programmatically request unique honey tokens for each experiment.

3.2.1 Honeypage. We coin the monitored web pages behind the URLs we send *"honeypages"*. Each page is served by a web server recording all access and interaction with it via HTTP and HTTPS. Figure 2 shows the structure of a honeypage URL. When a honeypage experiment is created, the framework generates a new unique URL referencing the experiment in the subdomain. We then transmit this root link, depicted in the first line of the figure, via the respective messenger. A honeypage links its subresources using the URL path, as shown in the second line of Figure 2. When the page is loaded, it tries to execute a script to create a fingerprint of the requesting browser, using FingerprintJS2 [26]. Thereby, we aim to detect very similar-appearing backends.

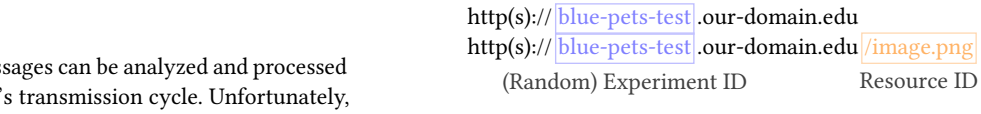

Figure 2: Honeypage subdomains (■) identify experiments and paths (■) identify a resource.

3.2.2 Experiment Variations. To find indicators for the message analysis reasons we discussed in Section 2.3, we use multiple types of honeypages in each experiment. Each type is based on the *regular honeypage* and adds specific characteristics to it, allowing us to uncover different ways a service handles these pages.

Regular Honeypage. Our starting point is a basic HTML website, including a title, style sheet, and a favicon in the HTML header. The page's body contains several typical resources, such as text blocks, an image, a PDF file, as well as JavaScript. To detect crawling, this kind of page includes links to two child pages, of which each also has two more children. Visiting this page with a modern browser results in ten HTTP requests before the page is fully loaded, which forms the baseline of a full load.

Honeypage with Meta Tags. One of the discussed purposes of message analysis is URL previews. During testing, we observed that several messengers struggled to create such previews for the regular honeypage. To aid this process, we supply a second type of honeypage, which adds common "meta tags" to the page's head, as well as the corresponding image resources. We deploy the *og:image*, *og:title*, and *og:description* meta tags. They are commonly used to present the core content of a site in a conveniently retrievable way. The intended audience for these meta tags are not visitors of the website, but machines like web crawlers that automatically extract information from it. By only processing the *<head>* section of the page, these tags allow the machine to grasp a short summary of the content without having to actually process all of it [64]. Similarly, we deploy more specific meta tags like *twitter:title*, *twitter:image*, and *twitter:description* specifically tailored for Twitter (X) cards [93].

Suspicious Honeypage. To check if messaging services conduct security checks on URLs, we add a third honeypage variation: pages hosting benign binaries flagged by antivirus solutions. This was done to check if a URL containing such content would be treated differently within the chat, perhaps indicating that the messaging service is performing content analysis using (3rd-party) security scanners. Secondly, we want to identify whether the messenger provides a visual alert, warning the receiver about potential dangers or even blocking the link if it is detected as potentially harmful.

In detail, the page includes links, binaries, and files that generate false positives for virus scanners. While they contain no malicious code, these files are flagged by the AVG virus scanner [6] and up to 11 of VirusTotal's [95] tests. We achieved this by compiling C++ programs with either an empty body, *printf*, or *cout* statements using Visual Studio 2019 and 2022. Lastly, we linked the Eicar homepage [20] and included a copy of the Eicar [20] antivirus test file, which is harmless but designed to be flagged by antivirus.

Blocklisted URL. Another possible form of security scanning would be to warn users about phishing attempts. To simulate such a phishing attack we utilized VirusTotal's academic API and checked links from a current Pi-hole [67] phishing blocklist [33] against VirusTotal's database. Our selection process prioritized a URL that was flagged as malicious by at least 25 security vendors and is included in multiple blocklists. We opted for a page that, despite being included in the blocklists, now returns HTTP 404.

We selected a—now—unavailable page for the same reason we did not include any actually malicious files: As we have no insight into the postprocessing done on the server of the service provider, we aim to avoid causing unintended side effects. Thus, we opted to select a page that does not actually serve harmful content yet is still block-listed. This choice allows us to safely assess whether a messaging service's link analysis feature cross-references exchanged URLs against known lists of malicious websites.

Honey Email Address. Finally, we transmit a unique email address via the chat. We use a fresh Gmail address and append the experiment's identifier after the plus sign [63], e.g., honey+red-pets-test@gmail.com. We monitor the address inbox looking for incoming mail bound to our honey token email addresses.

3.3 Web Chat Instrumentation

To assist in the analysis of the interactions with web-based messengers, we created a browser extension to automate repetitive steps and instrument the messengers' execution within the browser. To bootstrap an individual experiment, our extension connects with our honey messages framework to register the currently visited chat application and to retrieve the honey tokens. Then during the execution of the experiment, the extension records outgoing HTTP(S) and WebSocket traffic on the browser level using the *chrome.debugger* API [13], thus, no further man-in-the-middle (MITM) interception is required. This removes the requirement to break or circumvent SSL like in the mobile setup. Similarly, WebRTC traffic is recorded by the browser extension. To achieve this, the respective constructors are instrumented using content scripts and code injection. After each step in the experiment protocol (ref. 3.5), the extension takes screenshots of the chat for later analysis of the displayed previews.

3.4 Mobile App Instrumentation

For our investigations of mobile messengers, we used the app instrumentation framework by Koch et al. [41, 42]. The framework automatically installs the app and provides it with the required permissions. In addition, it also offers the option to inject an MITM proxy [69] into the internet connection and disables SSL certificate checks via Objection and Frida [31, 77]. We extend the tool by connecting it to the honey messages framework aiding our experiments. An Appium connection is utilized to paste the messages to transmit directly into the phone's clipboard.

3.5 Experiment Protocol

The two-person chats under investigation are carried out by one researcher using one device for the sender and another for the receiver, both connected to different networks with distinct IP addresses. Both accounts are logged in, and the chat is opened and stays open for both parties. While the receiver observes and keeps the chat window in focus, the sender account is sending our messages. MSP experiments require only the sender device and begin with a notification (ref. Appendix C). This notification informs and instructs any potential human reader to disregard our experiments, preventing unnecessarily consuming their time.

For each messaging experiment, we conduct five *sub-experiments* (ref. 3.2.2). Thereby, we generate three unique honeypage URLs and a honey email address sequentially. Finally, we use a fixed blocklisted URL. We begin our experiments with the non-suspicious tokens and finish with the blocklisted URL, to mitigate potential

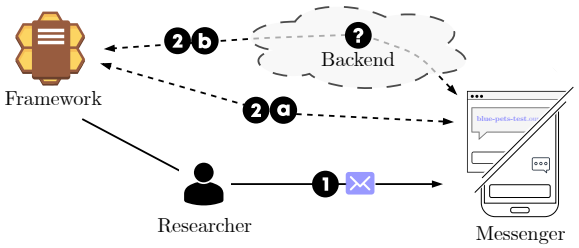

Figure 3: ❶ The framework provides a monitored URL to paste into the messenger's web or app chat. ❷ Afterwards, ⓐ the messenger may request the URL via client-side request, or ⓑ via SSR through unknown backend processes.

blockage that could arise due to suspicious actions. The experiments proceed with the link to the honeypage containing meta tags, followed by the link to the page without meta tags. Subsequently, we transmit the email address, followed by the link to the page with suspicious content, and finally, the blocklisted link. The blocklisted link stays the same for each experiment, as it is only required to check for visual indicators or potential blockage of the message. The web (ref. 3.3) and app instrumentations (ref. 3.4) aid in carrying out the experiments.

Figure 3 depicts the simplified experiment concept that applies to each sub-experiment involving honey tokens. In step (1), the researcher uses the respective tool—browser extension or app instrumentation—to register the current experiment with the honey messages framework. They then receive text with a new unique honey token (URL or email address) to copy and paste. After pasting the prepared message in the messenger's input field, the researcher has to wait for at least five seconds before sending the message to allow potential client-side previews (2a) to occur. Following the transmission of each message, the researcher pauses for another five seconds to wait for potential server-side processes (2b) to finish. Then, they capture a screenshot, recording the presence or absence of a link preview and other visual artifacts. Not depicted, the messenger serves the chat message to our controlled receiver client, where the researcher does not further interact with the message.

Our experiments rely on the core guarantee that the framework (ref. 3.2) ensures each experiment receives its own set of unique honey tokens and that the researcher only transmits and views the message for each experiment once. This is achieved by closing conversations of concluded experiments and never reopening them.

4 STUDY DESIGN AND EVALUATION

In this section, we detail our experiments and results. We first discuss our target selection (ref. 4.1), the specifics of our experiment (ref. 4.2), and our approach to analyzing the experiments (ref. 4.3). To this end, Table 1 forms the backbone, as it aggregates our findings across all analyzed messengers. We conclude this section with our evaluation of the results (ref. 4.3), which are interpreted in Section 5.

4.1 Target Selection

Our target selection comprises a total of 105 messaging services comprised of three different categories: 36 mobile messaging apps, 41 websites with chat functionality, as well as chats via 28 customers of 12 broadly used messaging service providers (MSPs). A full overview of all evaluated services is provided in Tables 3 to 5. The selection criteria for each category are detailed below.

4.1.1 Web-based Messengers. To gain a sample of messaging services, we first reviewed the top 250 websites listed on Tranco [46], using website categories from Symantec Sitereview [89] to identify popular messaging-related websites. We supplemented our list with additional services found in online rankings of messaging and dating platforms [e.g., 2, 3]. In our selection process, we excluded services that were not available in our region, services that required payment, services with extensive verification procedures, and services that necessitated the provision of foreign mobile numbers or addresses for registration. Additionally, we could only test dating platforms where we were able to match our two profiles in a reasonable time frame. This resulted in 41 web messenger targets.

Some messengers like WhatsApp tether their web app to the mobile device. This way, communication is relayed through the phone. Other services use standalone web apps. In general, multiple services have both web and mobile clients. We decided to facilitate our target selection independently of whether one or more types of clients exist. Instead, we conducted our target selection individually for each category of messengers we considered.

4.1.2 Mobile Messaging Apps. We visually inspected the Google App Store for the top 100 apps contained in the categories Communication, Social, and Dating for indicators that a chat feature is provided. The filtered apps were then retrieved from a data set downloaded on the 14th of April 2023 using an open source downloader [40], resulting in 36 APKs being available for further analysis.

4.1.3 Messaging Service Providers (MSPs). There is a myriad of different services available for developers and maintainers of websites to build and manage their web applications and businesses. Among these services, there are messaging service providers (MSPs) that specialize in offering messaging capabilities as a service. Thus, they could also be called chat-as-a-service providers. These providers offer to handle the entire messaging feature that a developer might want to embed in their page. The range of services supplied by MSPs can vary, with some focusing on bot-driven chats, customer support, live chats, or help desks. From a user's point of view, these services are often presented in chat boxes comparable to those of other messenger categories. Thus, we evaluate whether their behavior differs. Following an online ranking [35], we initially identified 7 common third-party providers for chats as a service. We semi-automatically extracted customer websites, i.e., websites using and embedding the service's chat feature, from the provider's "customer success story" pages and verified that a chat window was available on the customer website. Upon closer investigation of the network traffic, we noticed that some customers have migrated to other providers. Thus, our final list encompasses 12 MSPs: Salesforce [73], LivePerson [51], tawk.to [90], Intercom [36], Solvvy [79], LiveAgent [49], Drift [19], Zendesk [100], LiveHelpNow [52], re:amaze [70], LiveChat [50], and optimise-it [65]. Our list of tested websites can be seen in Table 4.

4.2 Experiment Description

Our experiments were carried out by three researchers over the course of one week in May 2023. We conducted follow-up experiments in November 2023. For our experiments, we used Chrome 113 on MacOS, Google Chrome 113 on Windows 10, Firefox 113 on Ubuntu, as well as Chrome 118 on MacOS. We used two rooted Google Pixel 6A and created two novel Google accounts for either smartphone using our SIM cards.

4.2.1 Experiment Preparation. To prevent unwanted side effects, we created three pairs of personas, one for intercepted, and another for app experiments without traffic interception. The last pair is used to register accounts for web-based experiments. Each persona received its own email address and phone number using three pairs of new prepaid SIM cards. Based on the requirements of the service, we created a new account for either user.

To establish a one-to-one chat session, there might be additional preliminary steps which we detail in the following. The most common requirement is to befriend both accounts. However, some services required additional steps to establish a chat session between our accounts. In the case of dating apps using a swipe-and-match concept [92], this involved swiping sufficiently long until both accounts were able to match each other. To eliminate the impact on the service or the other users, we did not interact with other users at all and strictly kept communication and exchange of honey links between the two experiment accounts. Other services, like TikTok, required us to first use their service for a couple of hours to lift a shadow-ban which prevented chatting and sending friend requests. Once all preparatory actions were complete, we continued to actually perform the experiment.

4.2.2 Experiment Realization. For each messenger, the five sub-experiments (ref. 3.2.2, 3.5) were conducted sequentially, separated by a waiting period between sending messages.

The honey messages framework (ref. 3.2) continuously monitors all HTTP requests to our monitoring domain. Before our evaluation, each experiment underwent at least 30 days of monitoring, and data collection is still ongoing afterward. For our evaluation, we manually encoded the visible resources by analyzing the screenshots taken during each experiment and noted visible indicators regarding the suspicious and blocklisted pages.

4.3 Analysis Approach

For this study, we tested 105 messengers, comprised of 36 app, 41 web, and 28 MSP customer targets. Following our experiment protocol (Section 3.5), we transmit four controlled honeypage URLs, one block-listed URL, and for the MSP messengers, an additional introductory message. Consequently, we sent at least 553 messages. Since the start of our first experiment in June 2023, we receive 805 relevant individual HTTP requests, of which 700 were third-party SSRs, and 105 client-side requests from our devices.

4.3.1 Analysis of Honey Token Access. To analyze the messengers' behavior we checked all Web requests received by our honeypot and cross-referenced them with the honeypage URLs we transmit. From each request we learn when which resource was requested by which IP address. Using GeoIP [54], allows to additionally attribute IPs to an Autonomous System Number (ASN) and its corresponding location. Since our regular honeypages have links to other honeypages, we can detect whether a service is crawling the URL, by looking for access to the child-pages. Using the timestamps of our experiments, we can derive whether a request was received after the chat was concluded. We choose a threshold of one minute after a message was sent until we count a request as late. Since each HTTP request carries the sender's IP, we can cross-reference them with our device IPs to learn if a request came from the sender or receiver client, or if it was an SSR.

Furthermore, as a check if visitors execute scripts, our honeypages contain JavaScript. In the process, we utilize FingerprintJS2 (FPJS2) [26] to recognize identical or very similar visitors. Next, to find associations between services, we cross-checked IP addresses that sent requests to our server. We found 15 messengers sharing at least one IP address with another. Finally, to simplify the process, we manually checked the email inbox for messages addressed to our honey token email addresses.

4.3.2 Traffic Interception. In addition to monitoring external requests to our honeypages, we analyzed the outgoing traffic during our experiments. If any of the transmitted URLs would be sent to a third party, this would constitute a severe privacy violation. We, therefore, scanned all outgoing traffic for these URLs, taking evasion techniques such as CNAME cloaking [17] into account. While we were able to frequently recognize CNAME redirect chains, namely for 60 % of the analyzed messengers, we were not able to detect any requests to obviously nefarious third parties. Most of these chains ended at various content delivery networks or cloud hosting providers where we are unable to attribute the destination service.

In this process, we followed best practices and precomputed common encodings and hashes for the relevant parts of our honey tokens. In alignment with state-of-the-art research [12, 21, 76, 82, 83], we used string similarity matching to look for the precomputed values. Similar to Starov and Nikiforakis [83], we found that next to plain text, tokens in the traffic were most frequently transmitted URL-encoded, followed by Base64-encoding.

4.4 Evaluation

Table 1 provides a comprehensive overview of our results. It shows all 45 messengers for which our honey tokens revealed practices with privacy implications. This constitutes 43 % of the 105 tested messengers. With 21 of 36 samples (58 %), we found indication for message analysis by the majority of the apps. Nearly half of the web messengers—20 of 41—access some of our honey tokens, followed by 14 % of the MSP customer websites. The tokens of the remaining 15 apps, 21 web and 24 MSP messengers received no access.

We found that a total of 39 tested services (37 %) visibly conduct link previews. With 56 %, apps have the highest ratio of messengers that render a preview, followed by 39 % of web-based messengers and only 7 % of MSPs. We found content blocking regarding URLs by eight messengers, as well as blocking of email addresses by two messengers, half app and half web, each. However, we did not receive any emails at any of the addresses provided.

4.4.1 User-Agents. It can be considered fair practice to mark automated requests using the *User-Agent* HTTP header (UA), e.g., *PreviewBot*. To this end, we analyzed the UAs from SSRs related to

Table 1: Aggregated findings of the messenger experiments with access to the honeypages.

	Messenger Name	Preview	Regular HP #SSRs	Regular HP #Visible	Regular HP #Requested	Regular HP Crawled	Meta HP #SSRs	Meta HP #Visible	Meta HP #Requested	Repeated	Late	Latest	#IPs	#ASNs	Requests from Sender	Requests from Backend	Requests from Receiver	Location
Web Messenger	Badoo	◑	2	0	1	○	4	2	2	●	○		2	1	○	●	○	
	Bumble	◑	2	0	1	○	4	2	2	●	○		2	1	○	●	○	
	Discord	◑	1	0	1	○	3	2	2	●	○		4	1	○	●	○	
	ICQ	●	18	3	6	○	21	3	7	●	○		12	1	○	●	○	
	Imgur	◑	1	0	1	○	3	2	2	●	○		4	3	○	●	○	
	Lark	●	9	2	9	○	9	2	9	○	○		5	1	○	●	○	
	LinkedIn	●	79	2	27	●	48	2	10	●	●	>4 min	9	3	○	●	○	
	MeetMe	○	1	0	1	○	1	0	1	○	○		2	1	○	●	○	
	Pinterest	○	1	0	1	○	1	0	1	○	○		2	1	○	●	○	
	Quora	●	4	1	1	○	4	1	1	●	○		6	1	○	●	○	
	Skype	●	3	2	2	○	5	3	3	●	○		2	1	○	●	○	
	Slack	◑	2	0	2	○	10	3	3	●	●	>1 min	14	2	○	●	○	
	Snapchat	●	3	2	2	○	4	3	3	●	○		9	2	○	●	○	
	SoundCloud	○	1	0	1	○	1	0	1	○	○		1	1	○	●	○	
	Steam	◑	13	0	1	○	8	2	2	●	●	>30 d	20	2	○	●	○	
	TamTam	●	11	1	9	○	12	2	10	●	○		9	1	○	●	○	
	Telegram	◑	1	0	1	○	2	2	2	○	○		3	1	○	●	○	
	Twitter	◑	5	0	2	○	11	2	4	●	○		6	2	○	●	○	
	VK	●	8	1	6	○	8	2	7	●	○		16	2	○	●	○	
	WhatsApp	●	0	1	1	○	0	2	2	○	○		1	1	●	○	○	
Messenger App	Bigo	○	1	0	1	○	1	0	1	○	●	>8 h	1	1	○	●	○	
	Botim	●	0	1	1	○	0	2	2	○	○		1	1	●	○	○	
	LINE	●	2	1	1	○	4	2	2	●	○		6	1	○	●	○	
	MoboNitro	◑	1	0	1	○	2	2	2	○	○		3	1	○	●	○	
	OK	●	12	1	9	○	13	2	10	●	●	>11 h	8	1	○	●	○	
	REALITY	◑	0	0	1	○	0	2	2	●	○		2	2	●	○	●	
	Robik Anti Filter	◑	1	0	1	○	2	2	2	○	○		3	1	○	●	○	
	Robik Zed Filter	◑	1	0	1	○	2	2	2	○	○		3	1	○	●	○	
	Signal	●	0	2	2	○	0	2	2	○	○		1	1	●	○	○	
	Skype	●	2	2	2	○	3	3	3	○	○		1	1	○	●	○	
	Snapchat	●	2	2	2	○	5	3	3	●	○		9	2	○	●	○	
	Talaei	◑	1	0	1	○	2	2	2	○	○		3	1	○	●	○	
	Telegram	◑	1	0	1	○	2	2	2	○	○		3	1	○	●	○	
	VK	●	7	1	6	○	9	2	8	●	○		17	2	○	●	○	
	Viber	●	0	2	2	○	0	2	2	●	○		2	2	●	○	●	
	WhatsApp	●	0	1	1	○	0	2	2	○	○		1	1	●	○	○	
	WhatsApp Business	●	0	1	1	○	0	2	2	○	○		1	1	●	○	○	
	Wink	◑	1	0	1	○	1	2	2	●	○		3	3	●	●	●	
	imoHD	●	0	2	2	○	0	2	2	●	○		2	2	●	○	●	
	nebenan.de	●	3	0	2	○	4	2	2	●	○		3	1	○	●	○	
	sms.messages	●	0	2	2	○	0	2	2	●	○		2	2	●	○	●	
MSP	absorblms.com (Drift)	◑	4	0	2	○	7	2	3	●	●	>23 min	8	3	●	●	○	
	deputy.com (Drift)	◑	15	0	2	○	22	2	3	●	●	>11 h	36	5	●	●	○	
	waterpik.com (LiveHelpNow)	○	11	0	10	○	10	0	10	●	●	>9 h	6	3	○	●	○	
	x-cart.com (Intercom)	○	7	0	7	○	7	0	7	●	●	>3 d	5	4	○	●	○	

● (yes), ○ (no). ◑ means only the meta-honey page receives a preview. Red and green encode bad and good practice, respectively. A service **crawled**, if it followed links on the honeypage. They conduct **repeated** requests, if the same resource was accessed multiple times by different IPs or by the same IP with a minimum delta of 1 second to account for potential network-related resending. **Late** requests arrived > 1 min after the message was sent.

app and web messenger chats, as we have full control over both participants in these chats. Since no interaction with honey tokens happens from our side and no eavesdropping is expected, all requests must come from automated tasks. 31 web and app messengers qualify for this analysis as they send SSRs with a User-Agent header. MeetMe is left out, as it sends SSRs, but without the header. Note that 12 messengers (39 %) use more than one UA for their requests, with the upper bound being LinkedIn, which utilizes 7 different UAs, including various Chrome versions (86 to 104). In this scope, we received 47 unique UAs which consist of 23 marked as automated and 24 other UAs. We count UAs as automated if they mentioning a messenger's name, the term bot, or a tool, e.g., *python-requests*. Of the unique UAs, 40 % mention a messenger, 32 % have the string *bot*, and 4 % mention a tool. Note that some contain both a name and *bot*, e.g., *TwitterBot*. Regarding the messengers, the majority marked all UAs as automated, while 10 messengers have at least one unmarked UA. Four messengers—SoundCloud, Lark, ICQ, nebenan.de App—stand out, as none of their UAs indicate that they come from automated visits. Oddly, another six messengers mark some, but not all UAs. These include Twitter, Discord, LinkedIn, Quora, TamTam, and OK app. In Section 5, we further discuss the interesting cases and complement our analysis in Appendix B.

4.4.2 JavaScript Execution. Our honeypage contains JavaScript to test if visitors execute scripts, which only happened for four messengers—Lark, LinkedIn, TamTam, and the OK app. We utilize FingerprintJS2 (FPJS2) [26] to recognize identical or very similar appearing visitors. Such a similarity appears, as OK app and TamTam share a common fingerprint, indicating a connection between them (ref. 5.5). While all visits from Lark carry the same fingerprint, TamTam's and LinkedIn's visits leave multiple different ones. Noteworthy, LinkedIn carries 15 different fingerprints over 16 requests.

5 DISCUSSION

In this section, we interpret and discuss the messengers' behavior from the position of a high-privacy standard. We first argue which behavior is acceptable for which analytic purpose. Then, we discuss the behavior of each group of messengers.

5.1 Acceptable Message Analysis

We previously (ref. 2.3) stated which reasons and motivations a messaging provider can have to analyze a user's private messages, namely spam, scam & security (ref. 2.3.1), link previews (ref. 2.3.2), surveillance (ref. 2.3.3), and business incentives (ref. 2.3.4). While surveillance and business incentives are unacceptable behavior for us from a privacy standpoint, we concede that there could be a case for security and usability. Here, we lay the foundation for our discussion by defining practices we consider acceptable.

5.1.1 Acceptable Spam, Scam & Security Analysis. Analyzing the content of a private message is a concerning practice. However, spam and scam protection are goals for which a privacy-preserving solution can be found. For instance, bulk sending of messages to unknown accounts can be distinguished from regular traffic by a minimal metadata analysis, possible even if the messages themselves are encrypted. To this end, the services have no need to review the message content. Further practices, like URL cross-checking with spam and block-lists, are feasible from the client-side. While it is possible that such lists are incomplete and easier to update on a server, we do not consider this to be sufficient reason to access user messages on the server-side, breaking their confidentiality.

Since targeted scams and phishing are harder to detect, as evidenced by spam mails that regularly evade sophisticated spam filters, we can give no implementation recommendations for these goals. Finding a client-side and privacy-preserving solution is a research gap left open for future work, and we argue that a sacrifice in perceived security in favor of privacy may be worth it. We consider opt-in based chat moderation provided by messengers like WhatsApp or Facebook Messenger an acceptable compromise. There, users are warned when being contacted by an unknown number and given the option to report the chat in case of suspected fraud or spam. The initial measure is possible without access to the clear text and without permanent record of the chat. For the more in-depth countermeasure, one of the involved parties has to explicitly opt-in, as both messengers state they only gain access to recent messages if one of the participants reports the conversation [23, 96].

5.1.2 Acceptable Link Preview Conduct. Evidently, creating link previews is a common goal of many messengers, as about two out of five display a form of link preview (ref. 4.3). The percentage of services with a preview mechanism is highest for messaging apps. We tried making it as easy as possible for the messaging service to conduct privacy-preserving previews—i.e., conducting only the necessary amount of SSRs—by providing meta tags in our honeypage variation (ref. 3.2.2). It is evident from our findings that, indeed, creating a link preview without meta tags seems to be a challenge for many messengers, as 17 of 39 (44 %) messengers that generally display previews, only display them for pages that supply meta tags. However, our results also show that even with such tags, ten messengers still conduct more requests than necessary.

In our view, requests from the server-side are more privacy-intrusive than requests from the client-side. This is because, for SSRs, the URL to be visited has to be extracted from the personal message, transmitted to a server, and visited from there. It is non-transparent for the user what else happens on the server-side when a link is processed, so we discourage SSRs. The repeated requests days to weeks after the chat highlight that long-term analysis of the chats is done, proving the issue of server-side analysis abuse. In contrast, when client-side requests happen, independently of their purpose, the messaging server does not gain access to the message, or URLs that were transmitted in the chat. However, receiver-side previews can introduce additional security risks if the sender can entice the receiver's device to visit an arbitrary URL. Additionally, it can pose a privacy risk, e.g., by leaking the receiver's IP address to the URL's server. The same theoretically applies to sender-side requests; however, in this case, the user at least knowingly typed or pasted the URL into the chat box themselves. Thus, only sender-side requests combine privacy and security advantages.

To answer if and how many SSRs are necessary for a link preview, we need to differentiate between web and app-based messengers.

Web-based Previews. The Web and modern browsers bring many security guarantees, strengthening security and privacy for end-users. Specifically, the Same-origin policy (SOP) [57] by default isolates web documents from different origins from another. Since

our honeypot does not set any access control (CORS) headers [56] that can relax the SOP, a website from a foreign origin is unable to programmatically access the content of our honeypages. Thus, a strictly client-side preview of a page transmitted in the chat of a browser-based web messenger is not directly possible. Theoretically, it would be possible for the messenger to embed the website of the URL in an iframe to display it for the user. Which, however, would constitute a complete visit to the page. Also, we found no messenger in our targets that does this.

Instead, if previews are desired, we suggest that a browser-based messenger perform the preview with a single SSR. Still, the messenger first needs to extract a URL from the chat and transmit it to the server from where an SSR requests the page's HTML content. Parsing this HTML serves two purposes. First, the title and description can be extracted from the page's header if available. Since the preview process is not standardized, alternative strategies are possible, like extracting the first heading and image from the page. Second, parsing the HTML reveals if meta tags are present and if they include a preview image (*og:image*, *twitter:image*), or if a favicon is set. The title, description, and URL of the desired image can then be transmitted to the client, which can conduct a privacy-friendly client-side request to fetch the image. As a result, one SSR collects enough information for the client to render a preview that is in line with many previews visible in our study.

App-based Previews. Apps do not need to comply with policies like the SOP, which makes app-based client-side requests possible. Thus, apps can conduct the necessary requests in order to craft a preview entirely from the client-side. Consequently, we consider zero SSRs acceptable from the perspective of an app messenger.

To summarize, our privacy setting only considers *one* SSR acceptable to preview a website in a browser-based chat and allows *no* SSRs for app-based messengers. Following, we discuss the observed conduct of the different messenger types in more detail.

5.2 Web Messengers

Looking at Table 1—as expected (ref. 5.1.2)—nearly all web messengers with access requested our honeypage from the server-side. The only exception is WhatsApp, which creates a preview entirely from the sender-side. However, this is a special case that cannot be compared to the others, as the WhatsApp web client is tethered to the mobile device running the app, meaning the requests come from the phone. Giving the web client the advantages of an app.

Overall, six of the messengers did render a link preview for at least one honeypage type but requested more resources than they show, namely ICQ, Lark, LinkedIn, TamTam, Twitter, and VK. Additional three messengers—MeetMe, Pinterest, and SoundCloud—visit links in the chat while not showing a visual representation of the page. A discrepancy our different honeypage types can highlight is that 8/17 web messengers that generally conduct previews only render them for pages that have meta tags, as seen in Table 1 (◐).

We observed an unexpectedly high number of distinct IP addresses from the web-based messengers. Steam, VK, and Slack, for instance, respectively use 20, 16, and 14 distinct IPs in our honeypage experiments. The messengers with a high number of IPs seemingly process links in the form of a queue, where varying IPs request different resources. VK requests the honeypage's root once, and then subsequent requests to the subresources follow from various IPs. Slack operates differently, as it requests the honeypage's root *twice* from different countries, as derived from the IP location. Subsequent requests also use different IPs to request two pictures four times each. With 70 %, the majority of browser-based messengers were found to request the same resources multiple times. The reasons for the messengers' conduct are unclear. However, such practices can cause unnecessary load for website operators, especially regarding comparatively larger resources like images.

Of the 19 messengers conducting SSRs, 10 marked their UAs as bots. From the remaining messengers, three—SoundCloud, ICQ, and Lark—stand out, as they showed no bot-indicators in their UAs. MeetUp is also special, as it never sent a User-Agent header. Notably, the remaining messengers marked some of their UAs, but also sent requests with regular-appearing agents. Half of the messengers used more than one UA for their requests.

We found measures to combat spam for three messengers. The MeetMe dating website visually censors links in the chat while the link is still visited once by MeetMe's servers. Meetup and Pinterest blocked us from sending our links. Another website, Quora, allowed links but blocked our email address.

Message analysis with the goal of security checks was only apparent for one web messenger, namely Twitter. As shown in Section 4.3, only Twitter blocked the phishing link from being sent. While related work [85] found that LinkedIn may also show a Google Safe Browsing [34] warning, the blocklisted page we used was not blocked. Interestingly, Google Safe Browsing indeed finds no malicious content on the site, which could explain LinkedIn's behavior. However, VirusTotal and PhishTank [14] do flag the page.

5.2.1 LinkedIn. The web messenger that stands out most is LinkedIn because it is the only one on our list that crawled the honeypage. Specifically, it visited all links to other honeypages on the landing page of our regular honeypage. Links on the resulting pages were not followed, so LinkedIn crawled one level of child pages. Interestingly, the service was one of the four messengers that executed JavaScript and sent us an FPJS2 fingerprint (ref. 4.4.2). The purpose of LinkedIn's visits is not apparent and would require speculation. What we can observe is a relatively high impact of sending a link via LinkedIn, as one link caused nearly 80 SSRs distributed over the linked website and child websites in a time frame of four minutes after sending the link. Amongst these visits, the service also requested all images on the page, which can cause increased load for website operators. The crawling practice, the large number of different IP addresses, UAs (ref. 4.4.2), and fingerprints are indicators for a specialized system in their backend, whose conduct clearly cannot be explained by link previews alone. Interestingly, among the browser versions signaled on LinkedIn's UAs are browsers with known exploits, e.g., CVE-2020-16015 for Chrome 86 [60]. We could not find indicators for effective security checks that would warrant such intense visitation practice, as neither our suspicious honeypage, which is packed with resources and links flagged by anti-virus tools, nor the blocklisted link were detected.

5.2.2 Steam. Another messenger that shows unprecedented behavior is Steam. The gaming platform shows a pattern of revisits and repeatedly requested resources, as highlighted in Figure 4. We received a series of reoccurring visits to our honeypage, each only

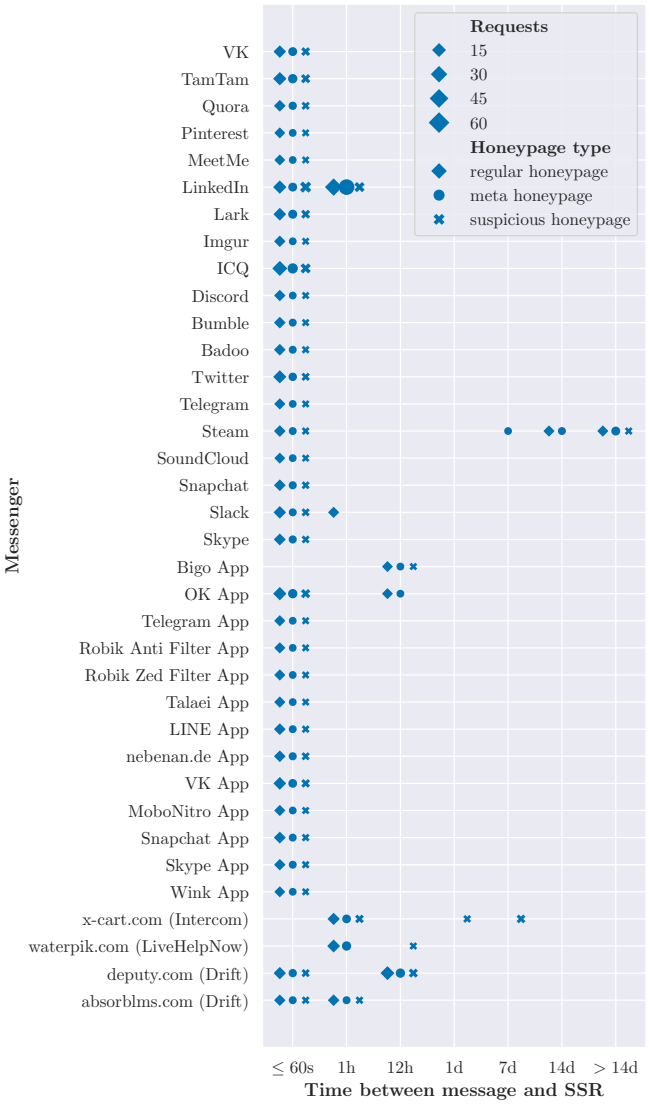

Figure 4: Time-bucket visualization of the amount of SSRs received with delay after sending a honey message, separated by the type of honeypage.

requesting the landing page and no further resources apart from the HTML. After the initial visits, directly when sending the message, the sequence commenced four days later, with one to seven days between the visits. The final visit to our regular honeypage was 30 days after sending our message. Interestingly, visits to our meta and suspicious honeypage stopped after about half that time. While the reasons are unclear, the requests necessitate that our URLs are stored long-term to be regularly processed. After our Steam chat, the respective honeypages—and no others—received visits from Canadian IPs with an unusual UA. It explained that a company, *Expanse*, regularly visits IPv4 addresses to identify "customers' presences on the Internet". While the exact meaning of that is unclear, it seems like Steam shares URLs in chats with a third party that ends up scanning the respective links. Additionally, after the Steam chats, we found indication that IPs previously seen in Steam experiments and other requests carrying the Expanse UA, probe honeypot subdomains unrelated to any experiments. Namely two subdomains are probed, *ns*, which indicates a DNS probe and *api*, with unknown purpose.

5.3 Apps

Apps are less restricted in terms of the SOP, compared to websites. With regard to RQ3, our results show more variation in behavior among the apps compared to the web messengers amongst each other. Apps conducted requests from all possible positions in the timeline of Figure 1—sender-client, server, and receiver-client. We can group the apps based on their request origin:

- Only 4 messengers are in line with our recommendations from Section 5.1.2, Botim, Signal, WhatsApp, and WhatsApp Business, since they create previews only with sender-side requests.
- Twelve apps, Bigo, LINE, MoboNitro, OK, Robik Anti Filter, Robik Zed Filter, Skype, Snapchat, Talaei, Telegram, VK, and nebenan.de conduct solely SSRs to URLs in the chat. Thereby, these messengers fall short compared to privacy-preserving client-side previews.
- There is also a mix that can be observed: REALITY, Viber, and "Messages, SMS, Text Messaging" (sms.messages) show similar practices. All three seem to create a link preview from the sender *and* the receiver-side. We discouraged receiver-side requests in Section 5.1.2.
- One service—Wink—also follows the sender-and-receiver approach, but additionally causes seemingly unnecessary requests from a server, with no clear signs about the intention (ref. 5.5).

An apparent difference between web-based and app messengers is that apps show fewer repeated requests. However, with 48 %, still, nearly half of the apps follow this practice, but apps give more indicators for the underlying reasons. If sender *and* client-side previews are done, as is the case for REALITY, Viber, Wink, imoHD, and sms.messages, both clients need to access the resource, which explains repetitions. However, LINE, OK, Snapchat, VK, and nebenan.de solely conduct SSR, which makes repetitions similarly hard to explain compared to web messengers. Overall, we recorded fewer distinct IP addresses for the app-based messengers, which is biased due to the client-side requests only apps do. Regarding the SSR-only messengers, web-based messengers used on average six backend IPs, and app messengers an average of five IPs, indicating similar server-side mechanisms.

Meetup, just like its web version, blocked us from sending links. Similarly, Likee Lite and nebenan.de disallowed URLs in the chat. Interestingly, while preventing us from sending the message, nebenan.de shows a preview of the link as long as it stays in the input field of the chat. Walkietalkie, in turn, blocked our email token.

We could not find evidence that providers conduct server-side security checks, i.e., the suspicious page was never blocked.

Only two apps visited our honeypage a considerable time after the chat was closed, Bigo and OK. The former stands out, as it conducts its first and only request eight hours after the chat concluded. Remarkably, they only issue one SSR from a Singapore IP to the root of the honey page. The latter, in turn, visited our page even later. With 11 h after the chat, it issued the latest request from all apps. Additionally, it conducted the most SSR compared to the other

apps and thereby loaded all ten resources of the honeypage. Similar to the web messengers, most apps used bot indicators in their SSRs' User-Agent headers. Only nebenan.de and OK did not make that fact obvious. In contrast to web, most apps (77 %) utilize one UA.

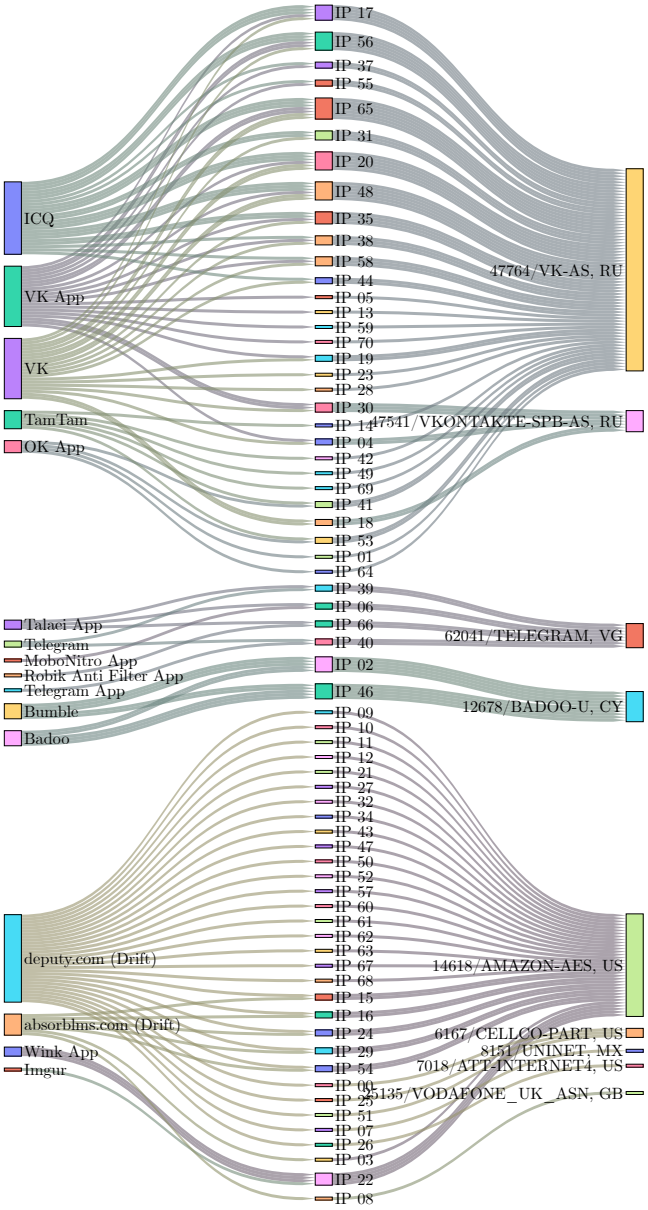

Figure 5: Connections of messengers, (pseudonymized) IPs and ASNs. Includes only messengers with SSRs from IP addresses shared with another messenger.

5.4 MSP Chats

In our mental model, described in Section 2.1, privacy-conscious users can reasonably expect to be communicating with the company whose website they are visiting. For example, it is expected to be talking to, e.g., a furniture store's employees when using the company's chat function. Our findings show that this is partially the case, but we can also find indicators for access from the MSP.

Clearly, our honey pages were frequently accessed, resulting from chats with companies using MSP services. Drift customers were the only ones in our list that displayed previews, specifically only when meta tags were present. Notably, all four companies that visited our page conducted more requests than necessary. While it was obvious in some instances that we talked only with bots, this is not certain in some cases. Nevertheless, it is possible that a human chat partner visited the links even though they were asked not to click them. However, the high amount of distinct IP addresses and the requesting of only selected resources by Drift and Intercom customers indicates automatic processes.

What stands out is the two companies that use Drift as their MSP share five AWS IP addresses, which indicates the access is associated with the Drift MSP, which is not expected in our user's mental model. The remaining requests from non-shared IPs are more likely to come from the companies themselves—the expected behavior. Thus, likely, both the company and the MSP have access to messages exchanged in the chat on the company's website.

5.5 Messenger Affiliation

We already discussed that the two Drift customers share IP addresses, which likely belong to the common MSP. Upon further investigating the IP addresses that request our honeypages in the chats, additional apparent connections between messengers present themselves. We display the messengers with at least one shared IP address in Figure 5 and show the connection to the respective ASNs. A shared ASN is not uncommon for cloud hosting providers. However, two services using the same IP address is a strong indicator of technical proximity. It shows that at least parts of the user message, i.e., the URLs, are transmitted to a common network, indicating the possibility of data sharing between the services.

We found multiple groups of services that are connected by common IPs and ASNs. Directly obvious are the apparent affiliation of the Russian messengers VK, ICQ, TamTam, and the OK app, all sharing the VK ASN. This view is strengthened by the browser fingerprint shared between OK and TamTam (ref. 4.4.2). The affiliation of the VK app and its browser-based pendant is no surprise, as are their similarities in our results. Both their versions display a nearly identical amount of requests and a similar number of utilized distinct IPs, making a shared backend likely.

Another apparent connection concerns Telegram and Talaei, MoboNitro, Robik Anti Filter, which visually appear to be Telegram clones. This is strengthened as they share IPs, a common ASN associated with Telegram, and appear identical in our evaluation.

Badoo and Bumble's server-side practices also appear similar (ref. Table 1), which is a connection users may not be aware of. Their requests share IPs from the Badoo ASN in Cyprus.

Interestingly, Imgur and Wink app use a common IP to access our honeypage. Notably, these requests mention getstream.io in the User-Agent header. This is interesting, as apparently Stream.IO—a third-party in-app chat messaging provider [88]—visits links in both chats. Since Wink already conducts sender- and receiver-side requests, the SSR by Stream.IO is not necessary for their link preview. Moreover, while the exact connection of these companies is

unapparent to us, Stream.IO lists both others in its featured clients list and claims to have build-in Imgur support. Some additional services do also share common UA strings. However, our analysis did not unveil further information beyond the more robust IP/ASN analysis, thus Appendix B features our full analysis.

Upon investigating the IP addresses, we cross-referenced all IPs with the *firehol_level1* [27] blocklist as of November 2023 to identify possible known malicious IPs. Fortunately, no matches were found.

5.6 Ethical Considerations

We made sure that our study had a very low operational impact on the services we analyzed. First, we conducted the one-on-one messaging experiments in a controlled environment, meaning that we sent our messages between two fresh accounts we controlled, which makes all submissions ethically non-challenging. Second, we evaluated customers of MSP providers, which were mostly comprised of bot-driven chats but could involve live chats. To spare potential peoples' time, we started each conversation with a short introduction, explaining the academic project, instructing the reader to ignore our links, and giving the option to opt-out. When the chat was not aborted by the other side, we continued with the experiment. While we were mostly ignored and did not receive any opt-out replies, two chats were closed prematurely by the other side.

In order to avoid drawing unnecessary attention from users towards our profiles, we did not engage with other accounts during the dating platform experiments. We only used their services to match both our accounts in order to be able to test the chat feature.

5.7 Limitations and Outlook

This section covers limitations and future directions of our black-box approach regarding visitor attribution, chat automation, other honey token types, and the potential for false positives.

5.7.1 Visitor Attribution. Since our approach leaks unique URLs to services, we control who first learns about these URLs and consequently who appears to leak them once we receive visits to it. As we have no insight into processes happening on the server-side, a black-box approach is inherently limited in providing detailed information for honeypage visitor attribution, i.e., if it is a third-party or the messenger itself. Our most robust approach is IP/ASN-based attribution. While requests from cloud-networks are not attributable to individual parties without further insight, IP addresses shared by multiple messengers are an indicator for technical proximity of services. User-Agent headers complement our IP-analysis, as they sometimes carry information provided by the visiting service. However, since UA strings can be freely set, e.g., spoof a known UA [59], they are less reliable than IP addresses. While using UAs to detect software versions is broadly discouraged [58], they still commonly include the names of other services in order to be treated similarly. For example, Signal uses *"WhatsApp/2"* as their UA [78].

5.7.2 Chat Automation. Our research covers a wide range of different messengers—browser-based and apps. We have explored automation via APIs, app-automation tools like Appium, and browser-automation tools like Playwright and Puppeteer. However, we quickly noticed that this approach was infeasible due to frequent changes in the applications, like changing API endpoints and element IDs, as well as bot protection mechanisms, like CAPTCHAs and bot bans. Thus, the manual approach currently scales best.

5.7.3 Non-text Honey Tokens. The messengers in scope of this study do not have uniform support for alternative types of messages like file attachments or images. Consequently, we opted for text-messages for our honey probes. However, non-text messages may be an interesting angle for future research in this domain.

5.7.4 False Positives. Any Internet-facing server, including ours, can be subject to web scans. While we regularly received probing attempts to well-known paths, e.g., */robots.txt*, or to common subdomains, like *owa*, *secure*, *ssl*, or *webmail*, these requests are easy to exclude from our findings. Since we utilize a previously unused and hard-to-guess subdomain for each experiment, to produce a false positive, an actor would have to correctly guess a subdomain associated with an experiment and visits it at the right time in an unobtrusive way. This is unlikely in multiple ways. For once, there is no feedback from our system that the visited subdomain belongs to an experiment, as we serve monitored honeypages on all subdomains and the association of visits to experiments is done afterwards. Second, large amounts of probing attempts would stand out, and third, we would catch previous visits to later used subdomains. As a sanity check, we assured that no requests to any experiment subdomains were received before the respective experiment subdomain was allocated by us.

5.8 Key Takeaways

The main takeaway of our analysis is that there are *no privacy guarantees* in online messaging and service providers do conduct server-side message analysis. Some providers evidently store and revisit transmitted URLs hours to weeks after initial transmission, showing a clear violation of the user's expected privacy. For the majority of messengers, we found indicators that transmitted messages are at least partially processed on the server side. Concerning our research questions, we can summarize our findings as follows:

RQ1: It is possible to establish a lower bound for message analysis by service providers using honey pages and transmitting corresponding unique URLs via chat messages.

RQ2: Partial inference about ongoing analysis and the corresponding actors can be made based on observed network traffic. Especially if late visits cannot be explained by usability or security considerations. However, reasons for immediate message analysis are not always obvious and might be overshadowed by previews.

RQ3: Apps, compared to web messengers, do show a more promising behavior from a privacy perspective. However, there are still app-based messengers conducting SSRs to links in the chat, which is not required and gives room for improvement.

6 RELATED WORK

Most research related to messaging services focuses on the analysis of the underlying protocols. Especially the implementation of messaging encryption is a particularly active research topic [e.g., 4, 5, 8, 15, 72, 98]. While broken cryptography does harm privacy, such research is orthogonal to our work. Black-box approaches, like ours, can even give signals of problematic cryptographic primitives

or the existence of out-of-band channels. Thus, black-box testing can guide future protocol analysis research.

Recent previous work concerning privacy-violating data exfiltration of mobile applications focused on analyzing traffic on startup of the application or after very limited interactions. Honey tokens were seeded across the smartphone prior to the conducted studies and looked for in the intercepted traffic. Such studies were done for Android [61] as well as iOS [43] or both [44]. More interactive studies were done by Nguyen et al. [62] for Android as well as by Koch et al. [42] for iOS and Android, during which privacy consent dialogues were extracted, analyzed, and interacted with to assess their impact. However, none of those studies involved intricate app interaction or the explicit transmission of honey data. Neither did they reason about the further usage of observed transmitted data beyond classifying whether the receiver domain is a known tracker.

As stated in Section 2.3, one reason for automatically interacting with URLs inside chat messages is to provide link previews. In 2019, Stivala and Pellegrino [85] studied the implementation of such previews and researched ways for an attacker to hide malicious content behind benign-looking previews. While parts of our study also concern previews, we only detect their usage to analyze what the message provider does on top, a question unanswered in the literature so far. The concept of honeypots dates back to 1989 when it was first described in literature [86] and was subsequently broadly used in deception systems for intrusion detection. In recent years, honeypots have proven to be a valuable tool for attack detection and cyber forensics [e.g., 28–30, 47, 81]. The concept has been extended to honeynets—networks consisting of multiple honeypot systems designed to be compromised by hackers—that are, apart from deception and intrusion detection, also beneficial for studying the attackers' tools and practices [80]. In Stoll's book, bait files were additionally deployed to draw the attackers' interest, keeping them occupied while they were unknowingly being traced. Most likely inspired by that, security researcher de Barros [16] coined the term "honey token" in 2003 for fictional data that is monitored similarly to honeypots. Such data-based mechanisms generalize the idea of honeypots and offer a more flexible way of trapping attackers into revealing themselves. Yuill et al. [99] deployed enticing bait files on a file server armed with alarms that are triggered when the files are accessed. While we do not configure alarms for resources, bait files roughly translate to the multitude of files, links, and subdomains we serve with the honey messages framework.

Since then, the concept of honey tokens has been applied to trace the source of phishing attempts by submitting links to supervised resources to phishing forms to detect access by the phisher [55]. A similar concept was later used to equip Google spreadsheets with false bank account details and fund transfer links to identify accessors in case the sheets were leaked [45].

Other approaches use tokens like bogus credentials in database systems [11, 38, 87], which either requires hackers to exfiltrate and manually use the honey tokens in order for access to be detected or requires setting alarms on database-level whenever the data items are queried which is again susceptible to false positives. honey tokens have also been used in form of images [53, 55], email addresses [97] and documents [9, 99]. All of these are not adapted to targeted attacks but rather, more generally, offer a second line of defense to detect an attacker during system exploration. In our work, we focus on text-based communication, which is why email tokens are also included as honeydata. We exclude sending credentials with our service because that would require more targeted and sophisticated attacks, which we rule out of scope.

Furthermore, it is a common concept to deploy hard-to-reproduce watermarks on physical currency and government documents to obstruct counterfeiting attempts because replicas with missing watermarks can then be identified as fake. This concept also finds adaptation for the world of inherently easily reproducible digital items, but instead of discouraging plagiarism by making the process of replication more complicated, the digital watermarks may be hidden and can later be used to recognize unlicensed copies. Popular examples are trap streets and phantom settlements, which are non-existent or misrepresented streets and cities, respectively, that mapmakers include in their maps for the purpose of revealing copyright infringements [e.g., 101]. Bercovitch et al. [7] apply this idea by generating convincingly genuine-appearing data items from tabular data used in production. The data is generated specifically for each employee to "post-mortem" trace who leaked the data to the public. Transferred to our work, such specifically generated data translates to the generated URLs we created for each experiment. Same as in the related approach, access to our URLs can be attributed to the messenger the URL was submitted with.

To summarize, in contrast to previous work, we do not build a deceptive system diverting attention from a production environment, nor do we address intrusion detection in terms of hacking attempts. Instead, we combine the flexibility of honey tokens with the detection capabilities of honeypots and actively expose specific messaging providers to monitored tokens as opposed to positioning the honeypot as a deception and waiting for access attempts.

7 CONCLUSION

In this paper, we presented the honey messages framework, a tool capable of facilitating honey token-based experiments in both browser and app-based online communication platforms.

Utilizing our framework, we conducted an extensive evaluation of 105 platforms across both web-based and app-based environments. This examination focused on the behavior of these services regarding transmitted URLs and email addresses in chat conversations. 34 % of the examined messaging services conduct server-side access of URLs within private conversations, thus demonstrating the capability and intent of server-side message analysis. Amongst them, seven messengers showed concerning practices that significantly encroach on user privacy expectations. Most alarmingly, we observed requests to URLs mentioned in the chat, days after the conversation was concluded. One messaging service even crawled website links in the chat, and another conducted recurrent visits, even a month after the conversation ended.

To our knowledge, our study represents the largest black-box messenger privacy study to date. Our findings underscore the diverse and sometimes invasive behavior of messaging services with respect to privacy, emphasizing the importance of ongoing research and user awareness in this area. While beyond our scope, in order to steer the community towards increased privacy awareness, we recommend choosing open-source E2EE-*only* messengers, some of which were included and positively stood out in our study.

AVAILABILITY

We published our Honeymessages framework, alongside a Python API connector to interact with the framework's REST API, and our app analysis plugin on GitHub https://github.com/honeymessages.

ACKNOWLEDGMENTS

We are thankful for the valuable feedback and suggestions of our anonymous shepherd and reviewers. We gratefully acknowledge funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2092 CASA – 390781972 as well as from the European Union's Horizon 2020 research and innovation programme under project TESTABLE, grant agreement No 101019206.

REFERENCES

[1] Gunes Acar, Steven Englehardt, and Arvind Narayanan. 2020. No boundaries: data exfiltration by third parties embedded on web pages. In *Proc. of Privacy Enhancing Technologies Symposium (PETS)*, Vol. 2020.4. De Gruyter.

[2] Aelieve Digital Marketing. 2023. Website Rankings For The Best Free Online Chatting Sites. Online https://aelieve.com/rankings/websites/category/online-communities/best-free-online-chatting-sites/. (2022-05-19).

[3] Aelieve Digital Marketing. 2023. Website Rankings For The Best Online Dating Sites. Online https://aelieve.com/rankings/websites/category/online-communities/best-online-dating-sites/. (2022-05-19).

[4] Martin R. Albrecht, Lenka Mareková, Kenneth G. Paterson, and Igors Stepanovs. 2022. Four Attacks and a Proof for Telegram. In *Proc. of IEEE Symposium on Security and Privacy*.

[5] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. 2019. The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol.. In *Proc. of Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*.

[6] Avast Software s.r.o. 2023. AVG. Online https://avg.com. (2023-05-25).

[7] Maya Bercovitch, Meir Renford, Lior Hasson, Asaf Shabtai, Lior Rokach, and Yuval Elovici. 2011. HoneyGen: An Automated Honeytokens Generator. In *Proc. of IEEE International Conference on Intelligence and Security Informatics*.

[8] Alexander Bienstock, Jaiden Fairoze, Sanjam Garg, Pratyay Mukherjee, and Srinivasan Raghuraman. 2022. A More Complete Analysis of the Signal Double Ratchet Algorithm. In *Proc. of Advances in Cryptology (CRYPTO)*.

[9] Brian M Bowen, Shlomo Hershkop, Angelos D Keromytis, and Salvatore J Stolfo. 2009. Baiting inside Attackers Using Decoy Documents. In *Proc. of International Conference on Security and Privacy in Communication Networks (SECURECOMM)*.

[10] Laura Ceci. 2023. Number of mobile phone messaging app users worldwide from 2019 to 2025. https://www.statista.com/statistics/483255/number-of-mobile-messaging-users-worldwide/. (2023-11-20).

[11] A Cenys, Darius Rainys, L Radvilavicius, and Nikolaj Goranin. 2005. Implementation of Honeytoken Module in Dbms Oracle 9ir2 Enterprise Edition for Internal Malicious Activity Detection. *IEEE Computer Society's TC on Security and Privacy* (2005).

[12] Quan Chen and Alexandros Kapravelos. 2018. Mystique: Uncovering Information Leakage from Browser Extensions. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*.

[13] Chrome Developers. 2023. API reference: chrome.debugger. Online https://developer.chrome.com/docs/extensions/reference/debugger/. (2022-05-19).

[14] Cisco Talos. 2023. PhishTank. Online https://phishtank.org. (2023-11-22).

[15] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. 2017. A Formal Security Analysis of the Signal Messaging Protocol. In *Proc. of IEEE Symposium on Security and Privacy*.

[16] Augusto Paes de Barros. 2007. DLP and Honeytokens. (2020-05-22).

[17] Yana Dimova, Gunes Acar, Lukasz Olejnik, Wouter Joosen, and Tom van Goethem. 2021. The CNAME of the Game: Large-scale Analysis of DNS-based Tracking Evasion. *Proc. of Privacy Enhancing Technologies Symposium (PETS)* (2021).

[18] Discord. 2023. Official Website. Online https://discord.com/. (2023-11-29).

[19] Drift. 2023. Drift | Everything Starts With a Conversation. Online https://www.drift.com/. (2023-10-26).

[20] Eicar. 2022. Anti Malware Testfile. Online https://www.eicar.org/download-anti-malware-testfile/. (2024-03-07).

[21] Steven Englehardt, Jeffrey Han, and Arvind Narayanan. 2018. I never signed up for this! Privacy implications of email tracking. In *Proc. of Privacy Enhancing Technologies Symposium (PETS)*.

[22] Europol. 2023. Dismantling Encrypted Criminal EncroChat Communications Leads to over 6.500 Arrests and Close to EUR 900 Million Seized. Online https://www.europol.europa.eu/media-press/newsroom/news/dismantling-encrypted-criminal-encrochat-communications-leads-to-over-6-500-arrests-and-close-to-eur-900-million-seized. (2023-11-11).

[23] Facebook Help Center. 2023. Report a conversation in Messenger. Online https://www.facebook.com/help/messenger-app/833709093422928. (2023-11-21).

[24] Facebook Help Center. 2023. What end-to-end encryption on Messenger means and how it works. Online https://www.facebook.com/help/messenger-app/786613221989782. (2023-11-28).

[25] Financial Ombudsman Service. 2023. Bank said victim of text message scam was grossly negligent and won't refund lost money. Online https://www.financial-ombudsman.org.uk/decisions-case-studies/case-studies/bank-said-victim-text-message-scam-grossly-negligent. (2023-11-19).

[26] FingerprintJS. 2023. Browser fingerprinting library. Online https://github.com/fingerprintjs/fingerprintjs. (2023-11-29).

[27] FireHOL. 2023. FireHOL IP Lists. Online https://iplists.firehol.org/. (2023-11-28).

[28] Daniel Fraunholz, Daniel Krohmer, Simon Duque Anton, and Hans Dieter Schotten. 2017. Investigation of Cyber Crime Conducted by Abusing Weak or Default Passwords with a Medium Interaction Honeypot. In *Proc. of International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*.

[29] Daniel Fraunholz, Marc Zimmermann, Simon Duque Anton, Jorg Schneider, and Hans Dieter Schotten. 2017. Distributed and Highly-Scalable WAN Network Attack Sensing and Sophisticated Analysing Framework Based on Honeypot Technology. In *Proc. of International Conference on Cloud Computing, Data Science & Engineering-Confluence (Confluence)*.

[30] Daniel Fraunholz, Marc Zimmermann, Alexander Hafner, and Hans D Schotten. 2017. Data Mining in Long-Term Honeypot Data. In *Proc. of IEEE International Conference on Data Mining Workshops (ICDMW)*.

[31] Frida.re. 2023. Frida - A World Class Instrumentation Toolkit. Online https://frida.re/.

[32] GDPRhub. 2023. BVwG - W274 2251055-1/5E. Online https://gdprhub.eu/index.php?title=BVwG_-_W274_2251055-1/5E. (2024-03-07).

[33] GitHub. 2023. GitHub: pihole-blocklists. Online https://github.com/topics/pihole-blocklists. (2023-05-22).

[34] Google. 2023. Safe Browsing. Online https://safebrowsing.google.com/. (2023-11-22).

[35] Harry Guiness. 2022. The 7 Best Live Chat Apps for Customer Support in 2023. Online https://web.archive.org/web/20230414081634/https://zapier.com/blog/best-customer-support-chat-apps/. (2023-10-26).

[36] Intercom. 2023. The Best of Automation and Human Customer Service. Online https://www.intercom.com. (2023-10-26).

[37] Amnesty International. 2021. How private are your favourite messaging apps? Online https://www.amnesty.org/en/latest/campaigns/2016/10/which-messaging-apps-best-protect-your-privacy/.

[38] Ari Juels and Ronald L Rivest. 2013. Honeywords: Making Password-Cracking Detectable. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*.

[39] Brian Klais. 2022. New Research Across 200 iOS Apps Hints That Surveillance Marketing Is Still Going Strong. Online https://app.urlgeni.us/blog/new-research-across-200-ios-apps-hints-surveillance-marketing-may-still-be-going-strong.

[40] Simon Koch. 2023. GitHub: App-Downloader. Online https://github.com/the-ok-is-not-enough/app-downloader.

[41] Simon Koch. 2023. GitHub: Scala-Appanalyzer. Online https://github.com/the-ok-is-not-enough/scala-appanalyzer.

[42] Simon Koch, Benjamin Altpeter, and Martin Johns. 2023. The OK Is Not Enough: A Large Scale Study of Consent Dialogs in Smartphone Applications. In *Proc. of USENIX Security Symposium*.

[43] Simon Koch, Malte Wessels, Benjamin Altpeter, Madita Olvermann, and Martin Johns. 2022. Keeping Privacy Labels Honest. In *Proc. of Privacy Enhancing Technologies Symposium (PETS)*.

[44] Konrad Kollnig, Anastasia Shuba, Reuben Binns, Max van Kleek, and Nigel Shadbolt. 2022. Are iPhones Really Better for Privacy? Comparative Study of iOS and Android Apps. In *Proc. of Privacy Enhancing Technologies Symposium (PETS)*.

[45] Martin Lazarov, Jeremiah Onaolapo, and Gianluca Stringhini. 2016. Honey Sheets: What Happens to Leaked Google Spreadsheets?. In *Proc. of Workshop on Cyber Security Experimentation and Test (CSET)*.

[46] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened against Manipulation. In *Proc. of Network and Distributed System Security Symposium (NDSS)*.

[47] Corrado Leita, Marc Dacier, and Frederic Massicotte. 2006. Automatic Handling of Protocol Dependencies and Reaction to 0-Day Attacks with ScriptGen based Honeypots. In *Proc. of International Symposium on Recent Advances in Intrusion Detection (RAID)*.

[48] LinkedIn Security. 2023. Data encryption. Online https://security.linkedin.com/content/security/global/en_us/index/our-security-practices. (2023-11-28).
[49] LiveAgent. 2023. Simple Customer Support Software for Teams. Online https://www.liveagent.com/. (2023-10-26).
[50] LiveChat. 2023. Web Live Chat Software & Online Customer Support. Online https://www.livechat.com/. (2023-10-26).
[51] LivePerson. 2023. The Best Conversational AI Platform for Business. Online https://www.liveperson.com/. (2023-10-26).
[52] LiveHelpNow LLC. 2023. Support Solutions for Better Business Communications. Online https://livehelpnow.net/. (2023-10-26).
[53] David Martin, Hailin Wu, and Adil Alsaid. 2003. Hidden Surveillance by Web Sites: Web Bugs in Contemporary Use. *Commun. ACM* (2003).
[54] MaxMind. 2020. GeoIP2 Databases. Online https://www.maxmind.com/en/geoip-databases. (2020-05-22).
[55] Craig M McRae and Rayford B. Vaughn. 2007. Phighting the Phisher: Using Web Bugs and Honeytokens to Investigate the Source of Phishing Attacks. In *Proc. of Annual Hawaii International Conference on System Sciences (HICSS)*.
[56] MDN. 2023. Cross-Origin Resource Sharing (CORS). Online https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. (2023-11-21).
[57] MDN. 2023. Same-origin policy. Online https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy. (2023-11-21).
[58] MDN. 2024. Browser detection using the user agent. Online https://developer.mozilla.org/en-US/docs/Web/HTTP/Browser_detection_using_the_user_agent. (2024-02-06).
[59] Monperrus, Martin. 2023. GitHub: monperrus/crawler-user-agents. Online https://github.com/monperrus/crawler-user-agents/tree/4288b18. (2024-02-06).
[60] Marius Musch, Robin Kirchner, Max Boll, and Martin Johns. 2022. Server-Side Browsers: Exploring the Web's Hidden Attack Surface. In *Proc. of ACM Asia Conference on Computer and Communications Security (ASIA CCS)*. (2022-12-13).
[61] Trung Tin Nguyen, Michael Backes, Ninja Marnau, and Ben Stock. 2021. Share First, Ask Later (or Never?) Studying Violations of GDPR's Explicit Consent in Android Apps. In *Proc. of USENIX Security Symposium*.
[62] Trung Tin Nguyen, Michael Backes, and Ben Stock. 2022. Freely given Consent?: Studying Consent Notice of Third-Party Tracking and Its Violations of GDPR in Android Apps. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*.
[63] Official Gmail Blog. 2008. 2 hidden ways to get more from your Gmail address. Online https://gmail.googleblog.com/2008/03/2-hidden-ways-to-get-more-from-your.html. (2023-11-25).
[64] ogp.me. 2023. The Open Graph Protocol. Online https://ogp.me/. (2020-06-07).
[65] Optimise-it. 2023. Chat Software. Online https://www.optimise-it.de/.
[66] Thomas Perkins. 2022. It's Their Word against Their Source Code - TikTok Report - Internet 2.0. Online https://internet2-0.com/.
[67] Pi-hole LLC. 2023. Network-wide Ad Blocking. Online https://pi-hole.net/. (2023-05-22).
[68] Police Service of Nothern Ireland. 2023. WhatsApp Family Impersonation Scam. Online https://www.psni.police.uk/safety-and-support/keeping-safe/protecting-yourself/scams-and-fraud/whatsapp-family-impersonation. (2023-11-19).
[69] Mitmproxy Project. 2023. Mitmproxy - an Interactive HTTPS Proxy. Online https://mitmproxy.org/.
[70] Re:amaze. 2023. Customer Service, Live Chat, and Helpdesk Solutions for Online Businesses. Online https://www.reamaze.com/. (2023-10-26).
[71] Reuters. 2022. UK watchdog seeks review into government use of WhatsApp, messaging apps. Online https://www.reuters.com/world/uk/uk-watchdog-seeks-review-into-government-use-whatsapp-messaging-apps-2022-07-11/. (2022-7-11).
[72] Paul Rösler, Christian Mainka, and Jörg Schwenk. 2018. More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema. In *Proc. of IEEE Symposium on Security and Privacy*.
[73] Salesforce. 2023. The Customer Company. Online https://www.salesforce.com/. (2023-10-26).
[74] Leonie Schaewitz, Cedric A. Lohmann, Konstantin Fischer, and M. Angela Sasse. 2022. Bringing Crypto Knowledge to School: Examining and Improving Junior High School Students' Security Assumptions About Encrypted Chat Apps. In *Proc. of International Workshop on Socio-Technical Aspects in Security (STAST)*.
[75] Asuman Senol, Gunes Acar, Mathias Humbert, and Frederik J. Zuiderveen Borgesius. 2022. Leaky Forms: A Study of Email and Password Exfiltration Before Form Submission. In *Proc. of USENIX Security Symposium*.
[76] Asuman Senol, Gunes Acar, Mathias Humbert, and Frederik Zuiderveen Borgesius. 2022. Leaky Forms: A Study of Email and Password Exfiltration Before Form Submission. In *Proc. of USENIX Security Symposium*.
[77] sensepost. 2023. Github: Objection - Runtime Mobile Exploration. Online https://github.com/sensepost/objection.
[78] Signal. 2024. GitHub: signalapp/Signal-Android user-agent. Online https://github.com/signalapp/Signal-Android/blob/71850e1e/app/src/main/java/org/thoughtcrime/securesms/linkpreview/LinkPreviewRepository.java#L93. (2024-02-06).
[79] Solvvy. 2023. Artificial Intelligence Customer Service & Support. Online https://web.archive.org/web/20230125194004/https://solvvy.com/. (2023-10-26).
[80] Lance Spitzner. 2003. The Honeynet Project: Trapping the Hackers. *IEEE Security & Privacy* (2003).
[81] Lance Spitzner. 2003. *Honeypots: Tracking Hackers*. Vol. 1. Addison-Wesley Reading.
[82] Oleksii Starov, Phillipa Gill, and Nick Nikiforakis. 2016. Are You Sure You Want to Contact Us? Quantifying the Leakage of PII via Website Contact Forms. In *Proc. of Privacy Enhancing Technologies Symposium (PETS)*.
[83] Oleksii Starov and Nick Nikiforakis. 2017. Extended Tracking Powers. In *Proc. of the International World Wide Web Conference (WWW)*.
[84] Martin Steiger. 2018. Telegram liefert IP-Adressen und Telefonnummern an Sicherheitsbehoerden. (2020-06-01).
[85] Giada Stivala and Giancarlo Pellegrino. 2020. Deceptive Previews: A Study of the Link Preview Trustworthiness in Social Platforms. In *Proc. of Network and Distributed System Security Symposium (NDSS)*.
[86] Clifford Stoll. 1989. *The Cuckoo's Egg. Tracking a Spy through the Maze of Computer Espionage*.
[87] Dominic Storey. 2009. Catching Flies with Honey Tokens. *Network Security* (2009).
[88] Stream.IO, Inc. 2024. Design + Build Any Chat Use Case. Online https://getstream.io/chat. (2024-02-16).
[89] Symantec. 2023. Sitereview. Online https://sitereview.bluecoat.com/.
[90] Tawk.to. 2023. Live Chat Software, Ticketing & Knowledge Base. Online https://www.tawk.to. (2023-10-26).
[91] Telegram Support Force. 2023. End-to-End Encryption FAQ. Online https://tsf.telegram.org/manuals/e2ee-simple. (2023-11-28).
[92] Tinder. 2023. Tinder Help. Online https://www.help.tinder.com.
[93] Twitter. 2023. Summary Card. Online https://developer.twitter.com/en/docs/twitter-for-websites/cards/overview/summary/. (2022-05-27).
[94] ValdikSS. 2023. Encrypted Traffic Interception on Hetzner and Linode Targeting the Largest Russian XMPP (Jabber) Messaging Service —. Online https://notes.valdikss.org.ru/jabber.ru-mitm/. (2023-11-11).
[95] VirusTotal. 2023. VirusTotal. Online https://virustotal.com. (2023-05-25).
[96] WhatsApp Help Center. 2023. About Blocking and Reporting Contacts. Online https://faq.whatsapp.com/414631957536067/?helpref=hc_fnav. (2023-11-21).
[97] Ben Whitham. 2017. Automating the Generation of Enticing Text Content for High-Interaction Honeyfiles. In *Proc. of Annual Hawaii International Conference on System Sciences (HICSS)*.
[98] Jan Wichelmann, Sebastian Berndt, Claudius Pott, and Thomas Eisenbarth. 2021. Help, My Signal has Bad Device! – Breaking the Signal Messenger's Post-Compromise Security Through a Malicious Device. In *Proc. of Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*.
[99] Jim Yuill, Mike Zappe, Dorothy Denning, and Fred Feer. 2004. Honeyfiles: Deceptive Files for Intrusion Detection. In *Proc. of IEEE SMC Information Assurance Workshop (IAW)*.
[100] Zendesk. 2023. Customer Service Software & Sales CRM. Online https://www.zendesk.com. (2023-10-26).
[101] Sarah Zhang. 2015. The Fake Places That Only Exist to Catch Copycat Mapmakers. Online https://gizmodo.com/the-fake-places-that-only-exist-to-catch-copycat-cartog-1695414770. (2023-10-23).

APPENDIX

A MESSENGERS

For reference, a complete listing of tested online messengers, messaging apps, and MSP chats is given in Tables 3 to 5. Especially the app's package names are relevant for clear identification of the corresponding app. The apps' names and the MSP providers the websites use were collected in May 2023.

A.1 Messengers Without Access

The paper mainly discussed messengers that accessed our honey tokens, as this proves some form of message analysis. Table 2 presents the list of 60 messengers (57 %) we were able to conduct experiments for, but did not find any indicators for message processing. The list is comprised of 21 web messengers, 15 apps, and 24 MSP chats. This means, neither of these messengers accessed our honeypage, was found to exfiltrate it in their network traffic, or caused an email to the honey token email address.

Table 2: Messengers without indicators for private message analysis.

Web Messengers (21)
TikTok, Tumblr, Twitch, csdn.net, DeviantArt, Etsy, Facebook, Flickr, gutefrage.net, KingsChat, Kleinanzeigen, Medium, Meetup, Okcupid, Pixiv, Reddit, Tinder, Trillian, Vinted, Voxer, Zoom
App messengers (15)
Dogorama, Uplive, Google Meet, purp, Knuddels Chat, Meetup, Vibes, TikTok, Tellonym, Walkie Talkie, noteit widget, PokeRaid, message.me, social.free, Likee-Lite
MSP messengers (24)
adidas.com (SalesForce), anyvan.de (re:amaze), barracuda.com (SalesForce), calm.com (Solvvy), cashflowtool.com (re:amaze), catapultsports.com (Zendesk), chevrolet.com (LivePerson), chupi.com (Zendesk), gelighting.com (SalesForce), getresponse.com (LiveChat), greenbeanbattery.com (tawk.to), headspace.com (Zendesk), ikea.com (optimise-it), kinguin.net (LiveChat), moo.com (Intercom), nascar.com (LiveAgent), otrmeals.com (re:amaze), printful.com (Zendesk), ryanair.com (Zendesk), bancsabadell.com (Zendesk), slido.com (LiveAgent), soundcloud.com (Solvvy), thunes.com (Intercom), trailersuperstore.com (LiveHelpNow)

B MESSENGER USER-AGENTS

Figure 6 displays a mapping of messengers to their utilized User-Agent HTTP headers. Included are all messengers that issued SSRs to our framework after the experiments. The full user agent strings are listed in Table 6.

Some messengers did not include the header in their requests, resulting in UA 46, "None". MeetMe never sent a User-Agent header, while Twitter, LinkedIn and Steam left out the header in some of their SSRs. Most of the time, services with an obvious connection used similar UAs. This includes the app and web version of Snapchat, Skype, and VK, as well as the Telegram clones. Furthermore, some connections we found in the IP/ASN analysis (ref. 5.5) are supported by common user agents. As such, Badoo and Bumble declared themselves as *BadooBot*, while TamTam and OK used *OdklBot*. The more interesting cases of Steam, as well as Imgur and Wink app are discussed in Section 5.2.2 and Section 5.3, respectively. Furthermore, ICQ and LinkedIn share a common browser user agent (UA 09), but there is no grounds to suspect anything else than a coincidence.

Section 4.4.1 discussed that overall about half of the distinct UAs we received are marked as automated, i.e., they directly reveal that they come from automated requests. The 23 UAs we consider marked are coloured light blue on the right side of Figure 6. The remaining 24 UAs are considered unmarked and coloured grey, while the empty UA is coloured white.

It is also interesting to look at the messengers' UAs in isolation to account for the shared UAs used by multiple messengers, e.g., the Telegram clones. For this, we regard the distinct UAs of each messenger, disregarding empty UAs. In Figure 6, these are the 61 links from messenger (left) to UA (right). Clearly marking automated requests as such is better for transparency, but still 41 % have no indication. Positively, we see that 38 % of UAs clearly mention their respective messenger in the string. Another 18 % do not mention the correct messenger, but are still clearly marked as automated, due to mentioning the term *bot*. This share includes MoboNitro, Talaei, and the Robik apps utilizing the *TelegramBot* string, which does not match the messengers' names. An additional 3 %—UA 35 and UA 40—do not name a messenger or have the term *bot*, but still show that the requests are automated by indicating tools like Python's *requests* library.

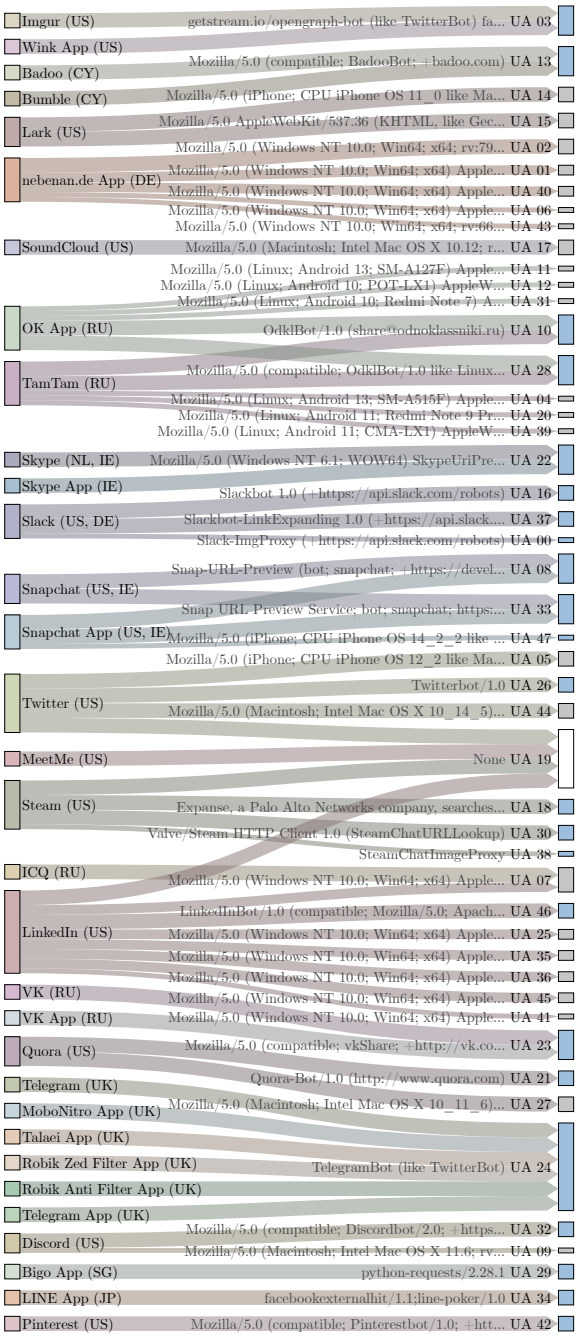

Figure 6: Connections of messengers and their UAs. UAs marked as automated are coloured light blue (■). Messengers have varied unordered colours for distinctiveness.

C MESSAGES SENT TO CUSTOMER SERVICE AGENTS

We prepared two different message templates. The first was chosen if no length constraints were imposed by the website, and the second was chosen if such constraints were in place.

> *"Dear person,*
> *Feel free to ignore this message, as it belongs to an ongoing research project by the «*Institute for Anonymous*» of «*Anonymous*». Ignore the links below. There is nothing you need to do. Let us know if you want us to exclude you from our evaluation. If possible, we will mark this chat as resolved and give you a positive rating, if supported by this chat.*
> *Thank you for your cooperation."* (Long Version)

> *"Hi, we don't want to waste your time, you can ignore this chat. We are researchers from «*Anonymous*». Ignore the following links. Let us know if you want to opt-out. Thank you."* (Shorter Version)

Table 3: Full list of all tested web-based messengers.

All Web Messengers (41)
Skype, Slack, Snapchat, SoundCloud, Steam, Telegram, TikTok, Tumblr, Twitch, Twitter, WhatsApp, Badoo, Bumble, csdn.net, DeviantArt, Discord, Etsy, Facebook, Flickr, gutefrage.net, ICQ, Imgur, KingsChat, Kleinanzeigen, Lark, LinkedIn, Medium, MeetMe, Meetup, Okcupid, Pinterest, Pixiv, Quora, Reddit, TamTam, Tinder, Trillian, Vinted, VK, Voxer, Zoom

Table 4: Full list of all tested mobile MSP messengers.

All MSP messengers (28)
absorblms.com (Drift), adidas.com (SalesForce), anyvan.de (re:amaze), barracuda.com (SalesForce), calm.com (Solvvy), cashflowtool.com (re:amaze), catapultsports.com (Zendesk), chevrolet.com (LivePerson), chupi.com (Zendesk), deputy.com (Drift), gelighting.com (SalesForce), getresponse.com (LiveChat), greenbeanbattery.com (tawk.to), headspace.com (Zendesk), ikea.com (optimise-it), kinguin.net (LiveChat), moo.com (Intercom), nascar.com (LiveAgent), otrmeals.com (re:amaze), printful.com (Zendesk), ryanair.com (Zendesk), bancsabadell.com (Zendesk), slido.com (LiveAgent), soundcloud.com (Solvvy), thunes.com (Intercom), trailersuperstore.com (LiveHelpNow), waterpik.com (LiveHelpNow), x-cart.com (Intercom)

Table 5: Full list of all tested mobile app messengers.

All Apps *(package name)* (36)
Dogorama—The Dog Community *(app.dogorama)*
Wink—Friends & More *(co.ninecount.wink)*
Uplive—Live Stream, Go Live *(com.asiainno.uplive)*
Google Meet *(com.google.android.apps.tachyon)*
purp—Make new friends *(com.hubolabs.hubo)*
imo HD—Video Calls and Chats *(com.imo.android.imoimhd)*
Knuddels Chat: Find friends *(com.knuddels.android)*
Meetup: Social Events & Groups *(com.meetup)*
Messages, SMS, Text Messaging *(com.messenger.messaging.sms.messages)*
Skype *(com.skype.raider)*
Snapchat *(com.snapchat.android)*
Mobo nitro *(com.telmemjplus.messenger)*
Messenger Viber: Chats & Calls *(com.viber.voip)*
VK: music, video, messenger *(com.vkontakte.android)*
WhatsApp Business *(com.whatsapp.w4b)*
WhatsApp Messenger *(com.whatsapp)*
Vibes—Dating, Match, Meet Up *(com.xpartner)*
TikTok *(com.zhiliaoapp.musically)*
nebenan.de *(de.nebenan.app)*
Tellonym: Anonymous Q&A *(de.tellonym.app)*
Botim—Video and Voice Call *(im.thebot.messenger)*
Walkie Talkie—All Talk *(io.walkietalkie)*
LINE: Calls & Messages *(jp.naver.line.android)*
noteit widget—by sendit *(me.bukovitz.noteit)*
PokeRaid—Worldwide Remote Ra *[sic] (me.pokeraid)*
REALITY—Become an Anime Avatar *(net.wrightflyer.le.reality)*
Talaei messenger *(org.eitagra.massenger)*
Robik Zed Filter *(org.massengrrobik.grazed)*
Robik Anti Filter *(org.messenger.robi)*
Telegram Messenger *(org.telegram.messenger.web)*
Signal *(org.thoughtcrime.securesms)*
OK: Social Network *(ru.ok.android)*
Bigo Live—Live Stream, Go Live *(sg.bigo.live)*
Messages: Chat & Message App *(sms.app.messages.app.message.box.message.me)*
SMS Messenger for Text & Chat *(sms.messenger.social.free)*
Likee Lite—Funny videos *(video.like.lite)*

Table 6: Companion table for Figure 6 with peudonyms, raw User-Agent strings, and the messengers using it.

ID	Raw User-Agent String	Messengers
UA 00	Mozilla/5.0 (compatible; vkShare; +http://vk.com/dev/Share)	VK, VK App
UA 01	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0	nebenan.de App
UA 02	Valve/Steam HTTP Client 1.0 (SteamChatURLLookup)	Steam
UA 03	Expanse, a Palo Alto Networks company, searches across the global IPv4 space multiple times per day to identify customers' presences on the Internet. If you would like to be excluded from our scans, please send IP addresses/domains to: scan-info@paloaltonetworks.com	Steam
UA 04	LinkedInBot/1.0 (compatible; Mozilla/5.0; Apache-HttpClient +http://www.linkedin.com)	LinkedIn
UA 05	Twitterbot/1.0	Twitter
UA 06	Mozilla/5.0 (compatible; Pinterestbot/1.0; +http://www.pinterest.com/bot.html)	Pinterest
UA 07	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.135 Safari/537.36	nebenan.de App
UA 08	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0	nebenan.de App
UA 09	Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 Safari/537.36	Lark
UA 10	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36	LinkedIn
UA 11	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36	LinkedIn
UA 12	Mozilla/5.0 (Linux; Android 13; SM-A127F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Mobile Safari/537.36	OK App
UA 13	Slackbot 1.0 (+https://api.slack.com/robots)	Slack
UA 14	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36 OPR/56.0.3051.52	nebenan.de App
UA 15	Mozilla/5.0 (compatible; OdklBot/1.0 like Linux; klass@odnoklassniki.ru)	TamTam, OK App
UA 16	Mozilla/5.0 (compatible; BadooBot; +badoo.com)	Badoo, Bumble
UA 17	Snap-URL-Preview (bot; snapchat; +https://developers.snap.com/robots)	Snapchat, Snapchat App
UA 18	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134 Safari/537.36	LinkedIn
UA 19	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.198 Safari/537.36	LinkedIn
UA 20	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.67 Safari/537.36	LinkedIn
UA 21	Mozilla/5.0 (Linux; Android 13; SM-A515F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Mobile Safari/537.36	TamTam
UA 22	Slackbot-LinkExpanding 1.0 (+https://api.slack.com/robots)	Slack
UA 23	getstream.io/opengraph-bot (like TwitterBot) facebookexternalhit/1.1 (+http://www.facebook.com/externalhit_uatext.php)	Imgur, Wink App
UA 24	Mozilla/5.0 (iPhone; CPU iPhone OS 11_0 like Mac OS X) AppleWebKit/604.1.38 (KHTML, like Gecko) Version/11.0 Mobile/15A372 Safari/604.1	Lark
UA 25	python-requests/2.28.1	Bigo App
UA 26	SteamChatImageProxy	Steam
UA 27	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.127 Safari/537.36	LinkedIn
UA 28	Mozilla/5.0 (Linux; Android 10; POT-LX1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Mobile Safari/537.36	OK App
UA 29	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:51.0) Gecko/20100101 Firefox/51.0	SoundCloud
UA 30	Mozilla/5.0 (iPhone; CPU iPhone OS 12_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.1 Mobile/15E148 Safari/604.1	Twitter
UA 31	OdklBot/1.0 (share@odnoklassniki.ru)	TamTam, OK App
UA 32	Mozilla/5.0 (iPhone; CPU iPhone OS 14_2_2 like Mac OS X) AppleWebKit/537.1.43 (KHTML, like Gecko) Version/14.2 Mobile/15E148 Snapchat/11.8.0.30 (like Safari/537.1, panda)	Snapchat App
UA 33	Quora-Bot/1.0 (http://www.quora.com)	Quora
UA 35	Mozilla/5.0 (Windows NT 6.1; WOW64) SkypeUriPreview Preview/0.5 skype-url-preview@microsoft.com	Skype, Skype App
UA 36	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari/537.36	nebenan.de App
UA 37	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36	Quora
UA 38	TelegramBot (like TwitterBot)	Telegram, MoboNitro App, Talaei App, Robik Zed Filter App, Robik Anti Filter App, Telegram App
UA 39	Mozilla/5.0 (Linux; Android 11; CMA-LX1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Mobile Safari/537.36	TamTam
UA 40	Mozilla/5.0 (Macintosh; Intel Mac OS X 11.6; rv:92.0) Gecko/20100101 Firefox/92.0	Discord
UA 41	Mozilla/5.0 (compatible; Discordbot/2.0; +https://discordapp.com)	Discord
UA 42	Mozilla/5.0 (Linux; Android 11; Redmi Note 9 Pro) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Mobile Safari/537.36	TamTam
UA 43	Slack-ImgProxy (+https://api.slack.com/robots)	Slack
UA 44	facebookexternalhit/1.1;line-poker/1.0	LINE App
UA 45	Snap URL Preview Service; bot; snapchat; https://developers.snap.com/robots	Snapchat, Snapchat App
UA 46	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36	Twitter
UA 47	Mozilla/5.0 (Linux; Android 10; Redmi Note 7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Mobile Safari/537.36	OK App