CryptoFilter: Privacy-Preserving Traffic Analysis of Weak Transport Layer Encryption at Internet Gateways

Benjamin Mixon-Baca
Arizona State University
Tempe, Arizona, USA
Breakpointing Bad
Albuquerque, New Mexico, USA
bmixonba@asu.edu

Diwen Xue
University of Michigan
Ann Arbor, Michigan, USA
diwenx@umich.edu

Roya Ensafi
Computer Science and Engineering
University of Michigan
Ann Arbor, Michigan, USA
ensafi@umich.edu

Jedidiah R. Crandall
Arizona State University
Tempe, Arizona, USA
jedimaestro@asu.edu

Abstract

Transport-layer encryption is fundamental to protecting the basic rights of Internet users from tracking, surveillance, and machine-in-the-middle (MITM) attacks by adversaries along the network path. Despite decades of progress in applied cryptography and an increasing adoption of HTTPS, a substantial amount of network traffic continues to lack proper transport-layer encryption, threatening the privacy of end users and the integrity of the networks they inhabit. Researchers and network operators examining the issue, however, face a dilemma – in order to understand and mitigate these risks, they often have to examine the very information they aim to protect, such as sensitive user data exposed in plaintext. This tension raises ethical concerns, impedes research efforts, and limits our understanding of the threats posed by unencrypted traffic.

In this paper, we present CryptoFilter, a system designed for scalable, protocol-agnostic, privacy-preserving analysis of network traffic with weak transport-layer encryption at network gateways. CryptoFilter incrementally reduces gateway-volume traffic (tens of Gbps) in real-time (scalable) to a manageable amount of actionable information, while preserving user privacy (raw packet payloads never leave memory) and protocol semantics (protocol agnostic). Over a month-long deployment at a regional ISP, we use CryptoFilter to reason about network traffic composition, finding that a concerning amount of traffic is unencrypted and not associated with web browsers. We perform privacy threat mining to attribute subsets of the bytes within traffic to potential surveillance (e.g., geolocation information) or infrastructure threats (e.g., plaintext javascript). Finally, we identify applications not previously analyzed and confirm through manual reverse engineering that they leak sensitive information or contain other privacy and security exposures.

CCS Concepts

• **Security and privacy** → *Security protocols*; • **Networks** → *Transport protocols.*

Keywords

Network Security, Passive Analysis, Privacy

ACM Reference Format:
Benjamin Mixon-Baca, Diwen Xue, Roya Ensafi, and Jedidiah R. Crandall. 2025. CryptoFilter: Privacy-Preserving Traffic Analysis of Weak Transport Layer Encryption at Internet Gateways. In *Proceedings of the 2025 Workshop on Privacy in the Electronic Society (WPES '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3733802.3764059

1 Introduction

Despite decades of advancement in applied cryptography and the growing adoption of HTTPS, civil society worldwide continues to face various threats from adversaries exploiting applications that transmit data with missing or weak encryption. The presence of such traffic poses a threat to end-users as well as the networks they inhabit. For end-users, applications transmitting Personally Identifiable Information (PII) in plaintext violate their privacy and expose them to tracking by eavesdroppers on the network path [26, 29, 52]. For applications, the lack of transport-layer encryption during software updates leaves them vulnerable to machine-in-the-middle (MITM) and code injection attacks [27]. For the networks through which such traffic traverses, the lack of transport-layer protection can jeopardize the integrity of the network, as illustrated by past denial-of-service attacks where on-path adversaries injected malicious JavaScript into unencrypted traffic [32]. Finally, for society as a whole, traffic exposing unique identifiers enables mass surveillance [2, 52], motivating agencies like the Federal Trade Commission (FTC) Office of Technology to pose a series of open questions such as “In what ways are firms engaging in commercial surveillance?” and “What notable trends exist in the collection of location data?” [16, 17]

Most of these past vulnerabilities were surfaced through a targeted approach – researchers reverse-engineered specific applications to assess the threats they posed, selecting targets based on factors like download counts, user base, or historical evidence of problematic security behaviors within certain market sectors. While these efforts have been successful in raising awareness of insecure communication practices, the insights derived from individual apps do not necessarily address the broader concerns of all stakeholders: network operators and ISPs are not merely interested in cataloging insecure applications; they need to understand the types of insecure traffic patterns that are actually *present* in their networks in order to protect their users and infrastructure. Likewise, for advocacy and enforcement of better security practices, stakeholders must know what insecure traffic currently is out there, yet selecting individual targets from the vast "long tail" of insecure applications is simply not feasible [14, 47]. All of this calls for a paradigm shift from application-centric analysis to an exploratory, network-level method that, through the lens of ISPs and network gateways, discover insecure traffic patterns without predefined targets.

Adopting such an approach to investigate insecure traffic at internet gateways, however, introduces significant challenges. Internet traffic is characterized by many proprietary protocols, ranging from those explicitly designed as protocols (e.g., mmTLS for Tencent apps [35, 54]) to those which are simply artifacts of application design (e.g., passing GPS coordinates to a server using a URL parameter called "latlng"). These protocols are often built on top of existing protocols, such as HTTP. Meaningfully characterizing unencrypted traffic requires reasoning about this full protocol stack, including the multitude of proprietary protocols. We therefore define our first requirement: the analysis system should be *protocol agnostic*. That is, it should work for protocols that are unknown, reasoning about such protocols on the fly.

Furthermore, any examination of real user traffic brings up serious privacy concerns as we hypothesize that sensitive user data is co-located with application and protocol-specific byte patterns, such as protocol framing. This concern is particularly critical in our situation, where we target precisely those traffic flows that lack proper encryption mechanisms. Analyzing such traffic increases the chances of finding sensitive data – like Personally Identifiable Information (PII) – that is exposed in plaintext. The second requirement that we define is that the analysis system should be *privacy-preserving*.

This scenario creates a paradox: network operators and security researchers have a legitimate need to characterize and mitigate the threats posed by unencrypted traffic; yet the data required for such analysis is the same data whose examination could violate the privacy of the users who send it. This is the tension that motivated our study. We ask: *Can we analyze unencrypted traffic from real users in a privacy-preserving way?* But, more to the point, the requirements that the analysis system be both protocol agnostic and privacy-preserving combine to introduce a unique challenge for meeting the requirement of *scalability*. This is because, while the protocol agnostic requirement implies a deep analysis of every byte of packet payloads, the privacy-preserving requirement precludes the possibility of storing packet raw payloads for later analysis. In other words, all analysis must be performed in real time, at traffic line rates.

In this paper, we present **CryptoFilter**, a system designed to meet all three of these challenges and facilitate privacy-preserving analysis of network traffic with weak transport-layer encryption at network gateways. CryptoFilter marks a departure from traditional flow-based analysis by adopting a content-centric approach, where fixed-size payload segments (chunks, variants, and invariants described further in § 3.1) serve as the primary unit of analysis. The system operates through a multi-stage filtering pipeline, where the initial stage discards traffic that appears to be fully encrypted or unlikely to contain payloads of interest, incrementally reducing high-volume gateway traffic to a set of patterns that are most relevant from a security perspective. The subsequent sifting stage isolates *invariant* segments of the traffic – consistent byte sequences that recur across multiple traffic flows – while ensuring that data specific to individual users (variants) are not accessed or retained beyond volatile memory.

What sets CryptoFilter apart from existing approaches is its emphasis on privacy: **privacy-preserving** analysis is not just considered; it is a fundamental design goal. CryptoFilter is explicitly developed with the understanding that unencrypted traffic can include sensitive personal data that must never be viewed by humans or stored in persistent memory. At the same time, it still aims to grant researchers some extent of visibility into packet payloads in an anonymized and aggregated form. Additionally, **scalability** is a major requirement for analyzing the vast amounts of gateway traffic at line-rate, eliminating the need to write data to persistent memory for offline processing. Finally, CryptoFilter is **protocol-agnostic**. In contrast to traditional solutions that rely on parsing specific protocols, CryptoFilter is capable of adaptively identifying segments of traffic with missing encryption. This allows it to target not just standard plaintext protocols like HTTP or SIP but also custom or proprietary protocols that may lack proper encryption.

We demonstrate CryptoFilter's practical utility through both in-lab, controlled experiments as well as a month-long deployment at a major Point-of-Presence (PoP) of a regional ISP. Analyzing traffic generated by real users, CryptoFilter 's multi-stage content sifting pipeline progressively reduced the high volume of gateway traffic – measured in tens of gigabits per second – by several orders of magnitude, isolating a manageable set of invariant patterns that are prevalent across the network but unrelated to any particular user. We clustered these discovered invariant patterns and performed follow-up manual analysis. Among the clusters, we identified patterns indicating privacy leaks, such as PII and location data, as well as plaintext JavaScript susceptible to code injection attacks. Furthermore, we conducted an end-to-end case study where we attributed the invariant patterns to specific applications responsible for generating them. Through manual analysis, we confirmed critical security and privacy vulnerabilities in these applications due to their inadequate transport-layer security practices, such as transmitting sensitive data over unencrypted channels.

The remainder of this paper is structured as follows. §2 provides background and related work on traffic analysis. §3 describes privacy considerations when conducting research on real network traffic. §4 describes the technical design of CryptoFilter. We evaluate CryptoFilter in §5. §6 describes ethical considerations for our research and we conclude with a discussion in §7.

2 Background & Related Work

We now provide an overview of passive traffic analysis and the type of traffic and applications CryptoFilter is designed to analyze, including examples from the literature.

2.1 Passive Traffic Analysis

Studies aimed at measuring the Internet and examining network traffic can be broadly categorized into active and passive approaches. While active measurement involves injecting new traffic to observe how a system or host responds, passive measurement involves only observing in-situ traffic to extract insights without changing the environment. Our work with CryptoFilter aligns with the latter approach. Passive traffic analysis leverages information from network packets to infer properties about the network or to characterize the behaviors of users within the network. Unlike active measurements, which are typically conducted from network edges, passive traffic analysis is often performed from network gateways, where aggregated traffic from multiple hosts enables large-scale analysis and provides more representative insights. Such analyses may rely on traffic metadata (e.g., packet timing, size, or inter-host connection graphs) or may involve examining content from packet payloads to identify specific byte patterns. CryptoFilter falls into this latter category of content-based analysis.

Prior work has used passive traffic analysis as a technique to address security and privacy-related research questions. Seminal work from the early 2000s demonstrated how mining invariant payload fingerprints could detect worm propagation and generate early alerts [23, 36, 48, 53]. Perdisci et al. and Aresu et al. further developed network-level clustering systems to automatically generate signatures for known malware families based on structural similarities in malicious plaintext HTTP traffic [4, 42]. For the detection of unknown malware or covert backdoor communications, Bortolameotti et al. introduced DECANTeR, a system for passively extracting fingerprints of covert HTTP communications [6]. On the privacy front, Fu et al. identified suspicious location-related transmissions over plaintext HTTP and classified whether these transmissions were intended by the user [18]. Closer to our work in spirit, Jain et al. proposed a passive traffic analysis technique for extracting latent identifiers from network traffic that could be exploited for covert tracking of users [21].

User privacy matters, even when there is a legitimate research need. Since passive analysis captures in-situ data from communicating parties, there is often no practical mechanism to notify these parties or provide an opt-out option [39, 41]. This presents ethical challenges, as sensitive personal information could be among the payloads that are inspected during the analysis. To mitigate these privacy concerns, some prior research has opted to completely discard packet payloads from their analysis. For example, Huang et al. and Prakash et al. curated a crowd-sourced dataset to investigate the security and privacy practices of smart home IoT devices [20, 43]. The researchers explicitly chose not to collect traffic payloads for privacy reasons, but acknowledged that discarding payloads limited the extent of possible analysis. Alternatively, when specific features of interest are known in advance, researchers may choose to extract and log only these features, rather than analyzing the entire payload. For example, DeKoven et al. and Ukani et al. extracted features such as "device X is updating antivirus product Y" from campus network traffic to explore the online behaviors of students [11, 51]. Our goal with CryptoFilter is to present an alternative: to preserve user privacy while providing researchers with *limited* visibility into packet payloads (layers 5 and above) in an anonymized and aggregated manner.

```
 POST mtapi/v6/home/tabs?wm_appversion=********utm_medium=a
ndroid&wmUuidDeregistration=12&wm_seq=-1&uuid=********&use
r_id=****&wm_longitude=********&wm_latitude=*******
```

```
 POST /rest/n/nearby/map/bubble?ll=********&earphoneMode=1&
mod=Google%28Pixel%207a%29&appver=*********&egid=********&
thermal=10000&net=WIFI&lat=&******&ANDROID_PHONE&bottom_na
vigation=false&ver=11.11&android_os=0&oDid=ANDROID_*******
```

Figure 1: CryptoFilter separates the variants (blue) from variants (red) to facilitate privacy-preserving traffic analysis. Top: packet from *Meituan*, a popular Chinese shopping platform. Bottom: a popular Chinese short video platform.

2.2 Applications with Insecure Transport

Despite the growing adoption of HTTPS, a significant number of applications still lack proper transport-layer encryption, threatening the privacy of their users. Studies as recent as 2020 have shown that many websites, particularly regional and governmental ones, do not support or enforce encryption [14, 47]. With mobile becoming ubiquitous, a substantial body of literature has examined the lack of encryption and privacy leaks in mobile apps [10, 50, 55]. In particular, Vanrykel et al. investigated how mobile apps can be leveraged for surveillance through the transmission of unique identifiers over unencrypted connections [52]. Their study of 1,260 apps found that a global passive adversary could cluster up to 57% of a user's unencrypted traffic using identifiers such as Android ID, MAC address, or IMEI. Knockel et al. and Rodriguez et al. evaluated the security practices of popular web browsers, identifying insecure transmission of sensitive data (e.g., location and browsing history) without encryption [27, 44]. Kujath et al. analyzed eight widely used mobile apps in Latin America and found issues such as plaintext HTTP connections and leaks of personally identifiable information (PII) [29]. In a 2020 report, CNCERT/CC analyzed 50 banking miniapps and found that over 60% of them transmitted personal data without encryption [9]. Apart from missing encryption, prior studies have also highlighted instances where proprietary, non-standard encryption schemes contained flaws that allowed adversaries on the network path to decrypt communications [22, 26, 35]. CryptoFilter has identified some of these previously reported issues in this work.

Applications with insecure transport not only endanger user privacy but also compromise the integrity of the networks they traverse. For example, Marczak et al. examined a large-scale denial-of-service attack enabled by on-path adversaries who intercepted and injected malicious JavaScript into unencrypted traffic [32].

3 Privacy Considerations in Network Research with Real-User Traffic

Our work is motivated by a paradox in network security and privacy research: much of the data needed for such research must be derived from real networks and users, yet the very act of examining and analyzing network traffic can infringe upon user privacy. This tension is particularly clear when targeting traffic that is weakly encrypted – on one hand, network operators and security researchers have a legitimate need to analyze data packets traversing network boundaries to assess vulnerabilities associated with unencrypted or weakly encrypted traffic. On the other hand, merely monitoring such unencrypted traffic can be ethically problematic, as sensitive information, such as Personally Identifiable Information (PII), may be exposed in plaintext.

Consider, for example, the question: "Are there applications in my network that transmit sensitive user information, such as PII, in plaintext?" Researchers or network operators seeking to answer this question face a dilemma: in order to identify applications leaking sensitive data to on-path entities (e.g., transit ISPs) due to insufficient encryption, they must examine the very information they seek to protect. The common justification for such research is often that the potential benefits outweigh the privacy risks, or that only processed logs generated by automated packet analyzers are stored rather than raw packet data [7, 11, 51]. While not storing raw payloads offers some degree of privacy protection, for most general-purpose traffic analyzers (e.g., Zeek), there is simply no guarantee that PII will not survive through processing or that the generated logs will not contain any data that can be personally identifying.

This dilemma impedes research efforts that involve sharing network traffic with third parties, such as security researchers outside of the organization managing the network. Prior work has highlighted the difficulties researchers had when seeking access to network traffic from network operators [7], with raw traffic payloads being (understandably) the least likely to be shared. This is consistent with our own experience. While we do not advocate for open access to network traffic or for establishing research exceptions in privacy regulations – network operators have a moral obligation to safeguard users' privacy – we *do* hope that through this exploratory study, we can address some of the more serious ethical/privacy concerns, such as the risk of PII exposure, and thereby alleviate the tension between security/research needs and privacy considerations.

3.1 Variants vs. Invariants

A key motivation behind our CryptoFilter system lies in the observation that privacy needs do not necessitate *complete* freedom from observation – the primary privacy concerns in this type of research is due to the potential exposure of sensitive information that can be personally identifiable (PII) within unencrypted or weakly encrypted traffic being examined [46]. Information such as UUIDs, location data, online identifiers, or other information *specific* to a natural person [1], are collectively referred to as `variants` in this paper [1], as they are defined by their nature of varying across individuals or devices. In contrast, `invariants` refer to those parts of the payload that are consistent across all traffic streams following a specific protocol or across all users communicating through a particular application. Examples include headers, protocol framing, or application identifiers. For answering security and privacy-related research questions, invariants often offer more insights as they allow researchers to identify trends and cluster violating applications or protocols regardless of the identities of the communicating parties.

As examples, Figure 1 shows packet payloads from two applications whose traffic appeared in CryptoFilter's outputs during our evaluation in § 5. In both cases, the application client transmits unencrypted data containing private information. In the top example (Meituan), the payload consists of a header and key-value pairs, including a unique user identifier and geographic coordinates. Assuming multiple devices with this app installed are present in a network, portions of their network traffic will overlap (i.e., `invariants`), while other portions will be unique to each device or individual (i.e., `variants`). An ideal traffic analysis technique would filter out the variant portions of the payload while retaining the invariant portions that are neither unique nor attributable to specific users.

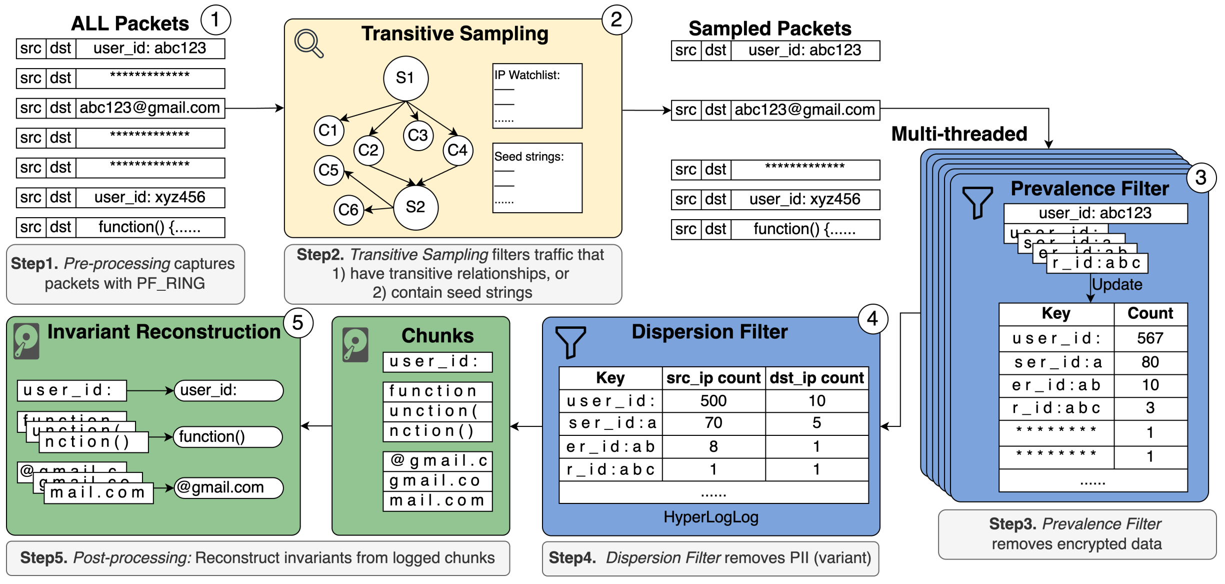
4 CryptoFilter: System Design

CryptoFilter is designed to facilitate privacy-preserving analysis of network traffic with *weak* transport-layer encryption at network gateways (e.g., residential ISPs, enterprise networks, university campus gateways). The system is guided by two primary objectives:

Goal#1: Isolation of weak/missing encryption The primary goal is to identify flows that likely lack encryption or use weak transport-layer encryption, while excluding flows that are properly encrypted. This approach reduces the search space, a necessary step given the substantial volume of traffic passing through a network gateway. Examples of *weak* transport-layer encryption include invariant assets (e.g., embedded public keys), protocol framing associated with plaintext communication, or insecure, non-standard encryption schemes that produce repeating ciphertexts (e.g., block ciphers using Electronic Codebook (ECB) mode).

Goal#2: Privacy-Preserving Analysis: No PII The system is designed to uphold user privacy throughout the analysis process. It should be noted that the traffic targeted by the above analysis potentially has more significant privacy implications than network traffic in general, as it likely lacks proper encryption to protect potentially Personally Identifiable Information (PII). Specifically, the content of the network traffic that *could* contain potential PII must not be accessible to human eyes or stored persistently (to disk).

[1]The concepts of "PII" and "variants" are not exactly synonymous. Variants include anything unique to a device or individual's traffic over a particular period, which makes this definition broader and more conservative – while personal data is explicitly specific to an individual, not all unique data qualifies as personal data. For example, properly encrypted ciphertext has a degree of uniqueness as it should be indistinguishable from random data, yet not all ciphertext is considered personal data.



Key	Count
user_id:	567
ser_id:a	80
er_id:ab	10
r_id:abc	3
********	1
********	1
......	

Key	src_ip count	dst_ip count
user_id:	500	10
ser_id:a	70	5
er_id:ab	8	1
r_id:abc	1	1
......		

Figure 2: System Overview of CryptoFilter. CryptoFilter facilitates privacy-preserving analysis of traffic with weak transport-layer security by using a two-stage content sifting pipeline that incrementally filters out traffic unlikely to provide usable insights for researchers (e.g., either fully encrypted or may contain privacy-sensitive data).

A direct implication of Goal#1 is that the system must be **protocol-agnostic**. The system should not rely on manually curated protocol specifications for parsing; rather, it should be capable of autonomously detecting traffic components indicative of weak encryption without having to parse them first. While certain plaintext protocols (e.g., HTTP) have well-defined structures, identifying new or emerging applications using custom protocols that lack proper encryption demands an automated, protocol-independent approach.

For Goal#2, a key implication is that analysis must be **scalable** so that gateway traffic can be processed in real-time without having to write the entire flow to non-volatile memory for offline processing. Additionally, the internal states maintained for the analysis across flows and times should use moderate amounts of memory.

Non-goal The privacy guarantees of CryptoFilter are focused on ensuring that its outputs do not contain content that is personally identifying *on its own* (i.e., myname=johnwick should not show up in outputs). However, the system does not aim to address the broader privacy implications of the presence or absence of specific content in its output. In other words, while CryptoFilter aims to filter out contents that could directly identify individuals, the inclusion of particular content in its output (e.g., language=russian) might still provide indirect insights into the composition of the traffic. Addressing these potential inferences lies outside the scope of CryptoFilter's design.

4.1 System Overview

We begin with a high-level overview of CryptoFilter before explaining each stage in detail. CryptoFilter is designed to analyze live network traffic at gateways, process it in a privacy-preserving way, and produce a set of *invariants* that can be ethically used to answer security questions about the traffic within a network. As shown in Figure 2, the system processes raw network traffic through multiple stages to generate the final outputs.

(1) Pre-processing We use PF_RING [37] to capture packets at the monitored (or mirrored) network interface. We also use PF_RING's zero-copy extension to further improve performance.This step also applies filtering rules to the packets, such as restricting the capture to specific subnets(e.g., only capturing traffic from student dorms on a campus network). PF_RING also supports scaling the system by distributing packets across multiple CPU cores, which is essential for processing gateway traffic volume in real-time.

(2) Transitive Sampling Modern traffic rates and volumes necessitate downsampling for systems that aim to process traffic in real-time. Without sampling, a moderate 60 Gbps feed of gateway traffic would fill 256 GB of RAM in just 30 seconds. A simple random sampling approach using flow identifiers would disproportionately favor short-lived, high-concurrency traffic patterns, such as those from network scanning tools (e.g., Zmap and MassScan [12, 19]). To address this disparity, we devise a novel sampling algorithm called *Transitive Sampling*, that dynamically selects one-to-many and many-to-one traffic patterns with particular characteristics found in client-server communications.

(3) Content Prevalence CryptoFilter aims to isolate traffic with weak or missing encryption. At this stage, a sliding window divides the payloads from flows into fixed-size chunks, then accumulates observations of each chunk.The goal is to remove chunks that are likely encrypted, as properly encrypted data should *not* repeat often compared to unencrypted data. At this point, the concept of "flows" becomes less relevant, and the analysis becomes content-centric – specifically the chunks are used as keys on which we maintain counters.

(4) Address Dispersion This stage measures how widely each chunk from the previous stage is dispersed across the network's address space. Specifically, we measure the number of unique IP addresses associated with each chunk and retain only those that are transmitted across a large number of addresses. If a chunk is common to many IP addresses (e.g., invariants from Figure 1), then they, by definition, do not uniquely identify specific users. The final output is a list of `invariants` and meta-data (i.e. packet numbers, offsets, and anonymized ports and IP addresses) that can be used for downstream tasks.

(5) Post-processing The raw output has applications including app or threat identification performed either manually or automatically using machine learning.

4.2 Sifting: (3) Content Prevalence

We begin by describing the design of the first sifting stage, which evaluates the prevalence of specific content within a traffic stream. Although the two sifting stages occur *after* transitive sampling, we describe them first, as they implement the privacy preserving component of CryptoFilter.

Chunk After a packet is pre-processed with `PF_RING`, we iterate over its payload using a sliding window of k bytes, generating a stream of k-byte `chunks`. Since each `chunk` is mapped to a counter to track its frequency, a smaller k would require less memory but be less effective at filtering out encrypted content (e.g., for $k = 1$, the system would only need 64 counters, but it would not be useful for filtering). On the other hand, a larger k might miss out `invariants` shorter than k (e.g., for $k = 10$, the invariant parts of userkey=******uid=****** cannot be isolated).Based on insights from prior work [21], we use a 64-bit (8 bytes) chunk size.

It is worth emphasizing that from this point onwards, the analysis in CryptoFilter focuses on these `chunks` – they serve as keys for maintaining counters in both of our high-pass filters (i.e., prevalence and dispersion filters). Compared to flow-based analysis, this approach is more efficient as it reduces the need to maintain extensive per-flow metadata. Additionally, using a small chunk size allows our approach to dynamically identify frequently occurring substrings without having to parse the entire payload at once, therefore does not require prior knowledge of the protocol (protocol-agnostic).

For the content prevalence filter, we use the contents (byte values) of each chunk as the key in an associative array of counters that track how many times each chunk appears in the traffic stream. If a counter exceeds a threshold T_P, the corresponding chunk is passed to the job queue of the next filtering stage (§ 4.3). We note that even with $T_P = 2$, the vast majority of properly encrypted traffic will be filtered out at this stage, and there is strong evidence to suggest that chunks remaining after this filter indicate that at least part of the corresponding TCP session is unencrypted[2].

4.3 Sifting: (4) Address Dispersion

Similar to the content prevalence stage, the address dispersion stage also maintains an associative array with k-sized `chunks` as keys, but with the number of unique sender/receiver pairs associated with each chunk as values. Importantly, only those chunks that pass

[2]A properly encrypted 8-byte should only recur once in 2^{64} chunks. Accounting for the birthday paradox, the expected amount of traffic before a collision is approximately every 40 Gigabytes of traffic. While collisions are likely to occur given the traffic volume at gateways, the preceding sampling stage and following dispersion filter limit total volume and impose a much higher threshold.

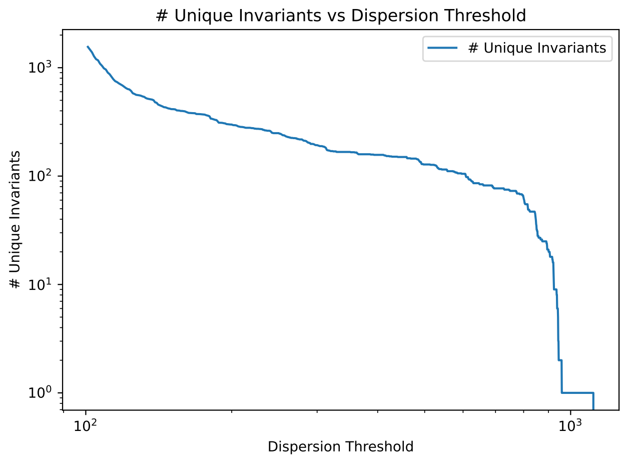


Figure 3: Number of unique invariants versus dispersion threshold (log scaled on both axes).

the content prevalence stage are tracked in this address dispersion table. If the count for a chunk exceeds a predefined threshold, T_D, it is passed on to the post-processing stage for logging.

This dispersion stage is designed to filter out chunks that originate from a small number of IPs. For example, consider the raw payload userkey=******. The chunk userkey= is likely common across multiple flows of this particular application across different clients, whereas chunks like serkey=* and the ones that follow contain user-specific byte values, therefore, less dispersed across the address space. The dispersion filter helps distinguish between payload segments that are unique to specific IPs that could potentially contain PII (e.g., user IDs or geographic coordinates) from those that are common across many IPs (e.g., protocol framing or static resources such as images or icons served to all users). By definition, the content of chunks that are observed in transmissions involving many different addresses should not be able to uniquely identify individual users.

A major challenge in tracking address dispersion across the address space is the memory required to store a set of unique IP addresses for each chunk. To address this, we implement the counters in the address dispersion table as HyperLogLog (HLL) [15], a probabilistic data structure designed for efficient cardinality estimation. HLL works by hashing elements into a fixed-size array of buckets, then tracking the maximum number of leading zeros among these hashes to approximate cardinality, all while using minimal memory. In our system, each chunk serves as a key, mapped to an HLL that tracks the unique IP pairs involved in its transmission. After each update to an HLL, we compare the estimated cardinality for the corresponding chunk against the threshold T_D.

It is important to acknowledge that IP addresses do not always directly correspond to individuals due to mechanisms like Network Address Translation (NAT) and users roaming across networks. CryptoFilter does *not* reason about NAT in any special way, as NAT generally results in chunks appearing less dispersed, thereby increasing confidence in the non-PII determination. On the other hand, when a single user roams across networks/addresses, it may reduce the apparent uniqueness of their data, making their chunks appear more dispersed than they are in reality. However, since CryptoFilter is designed to operate at the gateway of a specific network, the impact of user roaming on the dispersion analysis is limited to movement within that monitored network. To address this, the dispersion table can be periodically reset, with the reset interval adjusted based on the size of the monitored network and dispersion threshold. In our deployment, we reset the dispersion table every 10 minutes at most, with a conservative threshold of $T_D = 100$. Figure 3 shows the average number of unique invariants for different values of T_D. We settled on 100 for two reasons. First, having a sufficiently high lower bound ameliorates the potential for erroneous results due to collisions that might occur. Second, it gives the the highest number of unique invariants which permit us to cluster and classify applications more effectively. Finally, we emphasize that in our approach, the dispersion of IP addresses serves only as a heuristic for identifying chunks that are sufficiently common across the address space, rather than as a precise measurement of unique individuals.

4.4 (2) Transitive Sampling

A direct consequence of the requirement for the CryptoFilter system to process gateway network traffic in real-time, without storing packets to non-volatile memory, is the need for sampling traffic as it reaches the system. A naive approach, such as uniformly random sampling based on flow 5-tuples, would likely disproportionately favor short-lived, high-concurrency traffic patterns, such as those generated by network scanning tools, while discarding a large fraction of persistent client-server traffic, which is generally of more relevance for security analysis.

We improve upon random sampling by proposing and developing *transitive sampling*. At a high level, transitive sampling leverages the transitive relationships between clients, servers, and applications that initiate connections between them to explore a population of IPs with potentially similar protocol characteristics. For example, consider a scenario where two clients, C_1 and C_2, are using the same application to communicate with a server S_1. Once we establish that missing or weak encryption exists in the network flow $C_1 \leftrightarrow S_1$, it is reasonable to expect that other clients interacting with S_1 may also exhibit similar insecure communication patterns, as the underlying application is the same. Thus, when we observe a flow $C_2 \leftrightarrow S_1$, we should sample this flow so that `invariants` from this application can ultimately pass through the two high-pass filters. With the transitive relationship $C_1 \rightarrow S_1 \rightarrow C_2$, CryptoFilter effectively explores populations of IPs associated with a single application.

While the server-based transitive relationship allows CryptoFilter to explore communities related to a specific application, this form of sampling alone does not facilitate the discovery of new or, in particular, unknown applications or protocols. For this, we use a client-based transitive relationship. Whereas server-based transitivity assumes that the same application server will send/receive the same `invariants` to/from any client using the app, client-based transitivity assumes that each client is likely to have multiple applications, some of which might not be unknown to researchers but contain potential privacy violations. Continuing from the previous example, if C_1 communicates with S_1 over an insecure transport,

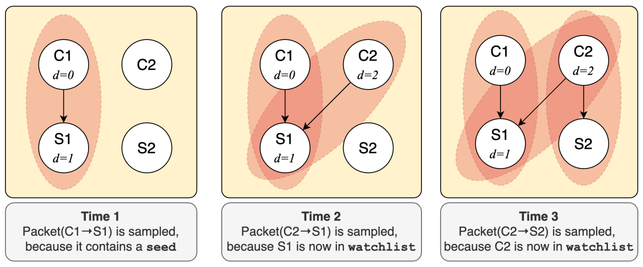

C1
d=0
C2
S1
d=1
S2
Time 1
Packet(C1→S1) is sampled, because it contains a seed
C1
d=0
C2
d=2
S1
d=1
S2
Time 2
Packet(C2→S1) is sampled, because S1 is now in watchlist
C1
d=0
C2
d=2
S1
d=1
S2
Time 3
Packet(C2→S2) is sampled, because C2 is now in watchlist

Figure 4: Transitive Sampling leverages the transitive relationships between clients, servers, and applications that initiate connections between them to explore a population of IPs with potentially similar protocol characteristics.

then the communication between C_1 and another server, S_2, involving a different application, may also be of interest to CryptoFilter. Client-transitivity also exploits the fact that a single application may contain advertising libraries or communicate with backend servers other than a primary app server.

Watchlist At the core of our implementation of *transitive sampling* is an associative array of IP addresses, referred to as the `watchlist`, whose traffic should be sampled and analyzed. Each IP address in the watchlist has an associated distance metric, d, representing the level of similarity or relationship between IPs. When a packet enters the sampling stage, if either the source or destination IP is in the watchlist, the packet is passed to the subsequent sifting stages. Additionally, if only one of the IPs is in the watchlist with distance d, the other IP is assigned a distance of $d+1$. If this distance is below a predefined threshold (to prevent indefinite growth of the watchlist), the IP is added to the watchlist. If neither IP is in the watchlist, the packet is discarded. As illustrated in Figure 4, the `watchlist` effectively captures both types of transitive relationships between IPs. We emphasize that the `watchlist` never leaves volatile memory and is never inspected by humans.

Packet Decay To prevent the same set of IPs being analyzed continuously and to create opportunities for exploring new IPs with higher distances, each IP in the `watchlist` is associated with a counter called Packet Decay. This counter decreases by one for every packet sent or received by the corresponding IP. When Packet Decay reaches zero, the IP is removed from the watchlist. Packet Decay serves as a low-pass filter, specifically designed to prevent IPs that generate large volumes of low-priority traffic from depleting system resources (e.g., network scanners generating high-volume probes from the same addresses). Without this mechanism, CryptoFilter would not be able to expand its scope and explore a broad range of unique applications and processes.

Cold Start Systems such as transitive sampling that rely on historical data or past relationships may lack sufficient information to make decisions when they are first initialized. Specifically, the `watchlist` relies on existing associations between IPs to expand and populate itself, but it requires a mechanism to bootstrap the watchlist upon initial startup. To address this, we use a small set of `Seeds` to help CryptoFilter target specific applications or traffic properties of interest and quickly start making meaningful sampling decisions.

Table 1 Seed Sets: The list of seeds used to bootstrap transitive sampling.

Seed Sets					
User Agent	**Location**	**HTTP Methods**	**Javascript**	**Control App**	
"User-Agent"	"latitude"	"POST /"	"javascript"	"GDTADNetClient"	"AiMeiTuan"
"user-agent"	"longitude"	"GET"	"function="	"AliXAdSDK"	"__weibo__"
"UserAgent"	"lng="			"iqiyi"	"douyin"
"User Agent"	"lat="			"MicroMessenger"	"HCDNLivenet"
"user agent"	"latlng="			"MQQBrowser"	"QQMusic"
	"CityId"			"QTP;QTP"	"QYPlayer"
	"City_Id"			"TBAndroid"	"Youku-ntk"
	"cityid"			"com.ss.android.ugc.aweme"	"Netease"
	"city_id"			"com.sankuai.meituan"	

We constructed five sets of Seeds, as summarized in Table 1. The "JavaScript" seeds are used to identify applications that transmit plaintext JavaScript. The "HTTP Method" seeds target IP communities that use plaintext HTTP transport – previous research has shown that many applications with weak transport layer security often embed their communications within HTTP [24, 31]. The "Location" seeds are associated with the transmission of location information. The "User-Agent" seeds prioritize sampling traffic generated by applications that explicitly identify themselves. These seeds are derived from domain expertise and findings from previous work [21]. Finally, the "Control App" seeds are byte sequences derived from user-agent strings of the control applications used in our evaluation in § 5.

While these seeds help address the cold start problem and expedite the construction of the watchlist, they are not indispensable components of CryptoFilter. As an alternative, random sampling could be used to initially populate the watchlist. We implemented both approaches and provide a comparison their performance (seed-based versus random) in Appendix A.1.

4.5 (5) Post-Processing & Applications

Once chunks have passed the dispersion stage they are invariant and are considered relevant from a security standpoint, with reasonable confidence that their contents are not personally identifying on their own. The invariants are logged to a file along with the timestamp, direction, source and destination port, IP address, packet number, packet length, and packet offset. If the invariant overlaps with those from our control data, they are logged directly, otherwise, a salted SHA256 (HMAC-SHA256) is applied prior to logging, as recommended by [38] to mitigate chosen plaintext attacks. The same anonymization technique is applied to the internal port and IP.

4.6 Prototype Implementation

The initial prototype of CryptoFilter was implemented on top of Zeek [56], an open-source traffic analysis framework. However, due to performance considerations, we later transitioned to a standalone implementation in C++. The standalone CryptoFilter system consists of roughly 28K lines of code and supports both PCAP mode and live traffic processing. The system interfaces with `PF_RING` with Zero-Copy support for efficient packet processing. The primary dependencies of CryptoFilter include the HyperLogLog library and libraries for `PF_RING` support during compilation.

The content prevalence stage is implemented using a multi-threaded design, with the number of threads configurable at compile time to match the number of available CPU cores. After transitive sampling, a prevalence thread is selected at random, and the packet is added to its thread-safe work queue. Each thread processes packets from its work queue by iterating through the payload to generate a stream of chunks. Each chunk is first checked against entries in the address dispersion filter, updating the corresponding entry if a match is found. Otherwise, the prevalence counter corresponding to the chunk is updated. Note that each thread in the prevalence stage maintains its own prevalence table, tracking the occurrences of different chunks independently of the other threads. *We plan to open-source our prototype at publication time under the GPLv3 license.*

4.7 Limitations and Potential Extension

As with any effort to enhance privacy, our system does not guarantee complete privacy in the broadest sense. Our goal with CryptoFilter is to improve upon current practices, and set a higher standard for privacy preservation during the analysis of weakly encrypted or plaintext traffic, especially since this type of traffic is inherently more vulnerable to scrutiny. CryptoFilter's focus is to provide a single guarantee: that the invariants emitted by the system, by themselves, do not directly identify any individual. However, even this seemingly simple goal needs to be qualified – it's widely understood that with enough surrounding context, almost any piece of information can potentially become personally identifying. We believe our system's invariant-centric approach to traffic analysis minimizes the contextual information that can be associated with or inferred from the system's output. For example, with a dispersion threshold of $T_D = 100$, each invariant is guaranteed to have at least 100 distinct IP-based contexts associated with it, therefore reducing the risk of associating the output with a specific individual.

It is important to emphasize that our design for CryptoFilter as presented in this paper errs on the side of being extremely conservative about user privacy. Although invariants have crossed thresholds that make them unlikely to contain PII, we still apply a salted cryptographically secure hash such that the plaintext of those invariants can only be recovered if we already know the plaintext. In other words, the unhashed packet contents are only visible to us as analysts if the same byte pattern also appeared in our control data, otherwise the invariant can only be used in a "bag of words" sense for clustering the data. Future work could relax these requirements to give analysts more visibility into the invariants that caused thresholds to be triggered, but because CryptoFilter is still in an experimental stage of development, our access to the network gateway is for research purposes only, and there have been no previous studies providing insight into what apps have poor transport security at an Internet gateway.

5 Evaluation

To assess the practical utility of our system, we deployed CryptoFilter within a real-world ISP – ISP-M– and conducted an initial deployment run to gain qualitative insights into the system's capabilities in a live network environment. The following sections describe our deployment process and initial findings. We begin by demonstrating the system's ability to reduce the vast traffic volume – measured in tens of Gbps – into manageable subsets more tenable

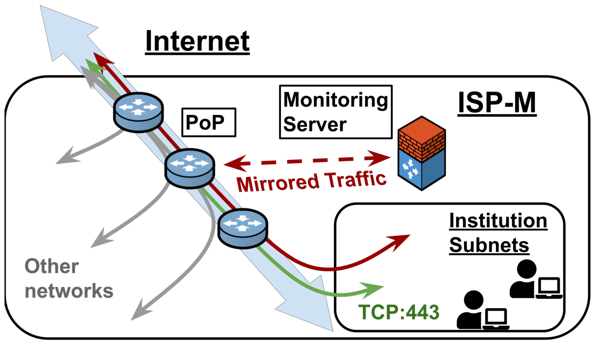


Figure 5: CryptoFilter Deployment inside ISP-M.

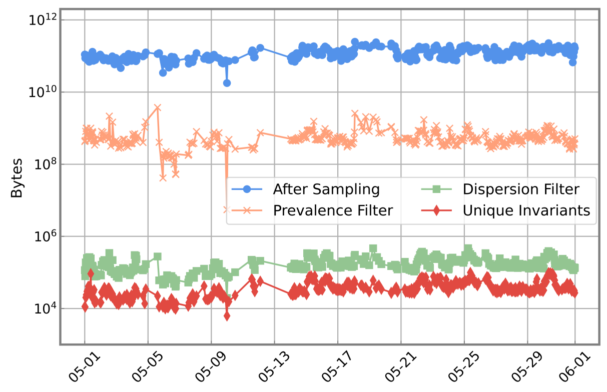


Figure 6: Traffic Volume that Passed CryptoFilter's Various Processing Stages. Both the prevalence and dispersion filters are able to reduce the volume of traffic by several orders of magnitude. $T_P = 2, T_D = 100$.

for an analyst to process. Next, we present an end-to-end case study in four stages: (§ 5.3.1) showcase CryptoFilter's ability to preserve semantically meaningful protocol information, (§ 5.3.2) identify potential privacy leakage (e.g., PII, location data, *etc.*) due to missing encryption while preserving user privacy, (§ 5.3.3) identify previously unanalyzed applications (those not previously analyzed or included in our control experiments), and (§ 5.3.4) reverse engineer eleven previously unanalyzed applications.

5.1 Setup

Our setup is two-fold: first, we adopt the perspective of a network operator by deploying CryptoFilter within an operational ISP, ISP-M [3]. This real-world deployment is augmented by a set of control experiments, which are designed to ensure strict adherence to ethical guidelines (see § 6) by allowing us to validate system outputs in a controlled environment.

5.1.1 ISP-M Deployment. We used a prevalence threshold of $T_P = 2$ and a conservative dispersion threshold of $T_D = 100$. To mitigate the effect of user roaming on the dispersion analysis, we reset both tables every 10 minutes. We ran CryptoFilter on the Monitoring Server for 31 days, collecting output `chunks` that passed through both sifting stages.

5.1.2 Control Experiments. We complemented our ISP deployment with control experiments conducted in a lab setting. Specifically, we selected 30 mobile applications – half from the Google Play Store and half from the Tencent Appstore (a major app store in China) – with apps chosen from the top categories in each store. Each app was downloaded and installed on six Android phones, and each instance was run with automated interactions using Android Debug Bridge (ADB), combined with manual interactions from researchers (for initial setups such as registration and agreeing to Terms of Service). Network traffic was captured, with flow-to-app attribution performed using Pcapdroid [13]. The raw traffic capture was then passed through CryptoFilter, with prevalence and dispersion thresholds set to $T_P = 2$ and $T_D = 6$, respectively. Finally, the output `chunks` were reconstructed into `invariants` following the procedure outlined in § 4.5.

[3]Regional ISP serving upwards of 2M users. Anonymized for submission.

5.2 Scalability & Privacy Preservation

Scalability is a primary obstacle for all but the most basic, byte-level pattern matching network analysis system. CryptoFilter's privacy-preserving constraint appears at face value to make this more challenging, but our solution to privacy preservation, namely, removing unique and likely encrypted communications *via* transitive sampling and content sifting, contributes significantly to solving the scalability problem. Figure 6 presents traffic statistics for a 31-day period, showing the reduction in traffic volume at each processing stage. Note that the reduction starts not with the raw traffic volume received at the mirroring interface (≈ 20 Gbps), but rather *after* transitive sampling is applied (≈ 4 Gbps, on average) — a 5x reduction right from the start. These results showcase CryptoFilter's ability to scale while processing data in memory. Following the initial 80% reduction from transitive sampling, the prevalence and the dispersion filters are able to reduce the traffic volume further by several orders of magnitude – specifically, the prevalence filter, with a permissive threshold of $T_P = 2$, removes over 99.6% of the sampled traffic, while the dispersion filter, configured with a conservative privacy threshold of $T_D = 100$, further reduces the remaining traffic by over 99.97%. Combined, this progressive refinement yields hundreds or thousands of `invariants`, rather than the billions of raw chunks that would have been produced without refinement. The reduction makes it significantly more manageable and ethical for subsequent analysis, both manual and automated, as not only are there fewer chunks, but they are the most relevant from a security analyst's perspective and cannot be attributed to unique IP addresses.

5.3 End-to-End Analysis

The data can serve both broad and deep analyses of the network and specific applications within it. An analysis cycle might start by analyzing aggregate protocol composition and similarities in the network, followed by mining potential privacy threats. An analyst could then search for and identify applications with potential privacy violations, and finally, perform deeper analyses *via* manual reverse engineering of threats actually present in the network.

For analyzing protocol composition, techniques in topic modeling algorithms, such as Tf-Idf [49], LDA [5], LSA [30], or Top2Vec [3] can be used to cluster traffic. Privacy threat mining can either be conducted on the raw output, or using a packet reconstruction algorithm such as the one detailed in Algorithm 1 in Appendix A.2. App identification can be performed by correlating CryptoFilter logs those from complementary tools such as Zeek, Snort, or Tshark. We opted to correlate CryptoFilter with user-agents and domain names extracted using Tshark. Finally, specific applications can be analyzed more deeply using manual reverse engineering to confirm the presence of security and privacy threats and exposures.

The modular processing pipeline could be incorporated into security units, (SoC team, machine learning, purple team) to proactively triage exposures or for cyber-risk insurance policies.

5.3.1 Aggregate Protocol View. We visualize CryptoFilter's ability to extract semantically meaningful protocol information, even when protocols are unknown or nested within known protocols like HTTP, by first applying Top2Vec [3] to the extracted invariants to generate an embedding representation, where each invariant is a word and each flow a document. The combined topic-/word-embedding representation preserves semantic relationships between different application and protocol invariants, unlike other topic models (LDA [5], LSI [30]) which do not. We then apply UMap [33] to the embedding for dimensionality reduction, yielding the two-dimensional projection in Figure 7. Colored clusters contain labeled flows (flows with a user agent) while black clusters contain unlabeled flows (no associated user agent string). We used HDBScan [8] to cluster flows.

We include specific applications in the annotations to highlight relationships between specific applications and protocols. For example, MicroMessenger is largely distinct from other applications which is consistent with the fact that it uses the custom "mmTLS" protocol and is heavily used in Tencent mini-Apps [35, 54]. Mozilla and "Other" (unlabeled) flows cluster together in proporation to Mozilla's representation in the labeled data (67%). Microsoft processes cluster together and near "Other" and Mozilla flows, suggesting HTTP is a base layer. Finally, various Sogou services cluster together and also near browser flows, which is consistent with findings that Sogou is built on top of HTTP [24, 26, 28]. These findings highlight CryptoFilter's ability to preserve semantically meaningful protocol information, even when the application-specific protocol is unknown or embedded within HTTP. *More critically, our results show that a large proportion of insecure Internet traffic cannot be attributed to browsers.*

5.3.2 Privacy Threat Investigation. Much the same way a sluice separates gold from dirt and various cruft, CryptoFilter separates potentially serious privacy violations by extracting invariant bytes adjacent to privacy leaks. To illustrate this capability, we use CryptoFilter's output to *sluice* for privacy threats automatically. First, we reconstruct contiguous invariants for flows as outlined in § 4.5. Next, we apply an LLM to the reconstructed patterns, instructing it to categorize the patterns into security-related categories (**PII:id, PII:device, PII: others, Location, Javascript, Useragent, HTTP/HTML, Other**). Table 2 presents a sample of reconstructed patterns categorized in this fashion. This highlights CryptoFilter's ability to simultaneously preserve data utility and user privacy.

Table 2 A sample of invariants discovered by CryptoFilter.

PII:id	**PII:device**	**PII:others**	**Location**
username	"device_id="	PhoneNumber=	countryCode=
urn:uuid:	device_ost:	User-Ages=	this.geolocation
X-Log-Uid=	androidUserInfo	organization=	timezone=
entityId	Fce_id":"	membership=	"cityid":
Javascript	**Useragent**	**HTTP/HTML**	**Other**
return this.	User-Agent:	Cache-Control	font-size:
function() {	Mozilla/5.0	onclick=	-pixel-ratio:
<script type=	Log-Network:	<input class=	upnp:rootdevice
.getElement ById	Micro Messenger	HTTP/1.1 Accept:	Allow- Credentials:

Table 3 Candidate applications identified traversing ISP-M.

Control-Related	Netease	QQBrowser	WeChat	QQ Music
	TC-Video	Youdao	QQ News	TC-Comics
	DYZB	Bilibili	Taobao	Nat.Karaoke
Previously Analyzed	Sogou	UCBrowser	dyplayer	Tencent
	Douyin	ttplayer	MQQBrowser	Weibo
	Tiktok	KakaoTalk	Tencent Mobile Manager	
Unknown	NewsApp	Flush	Fliggy	Ctrip
	netdisk	ESPN	Meituan	Iqiyi
	QTP;QTP	metwatchApp	Discover	Ting
	douban	JinShanCiBa	Kuaishou	AnQuanWeiShi
	ZhihuHybrid	Tizen	Huion	Huorong
	BossZP	Youku	Lavf	Hpplay
	Thunder	Dianping	SpeedinVPN	QuarkBrowser

5.3.3 App Investigation. After identifying potential privacy threats, the next step is to identify a list of candidate applications for analysis. Table 3 presents the list of candidate applications identified in the gateway traffic. Unlike for our cluster analysis, which used only user-agent strings for labels, we adopted a simplified version of the methodology proposed in [34]. Specifically, we extracted the protocol stack from the packet and used any available domain/URLs, substrings of HTTP header fields (e.g., User-Agent), or JavaScript libraries (if any) to label the traffic.

We used invariants and user agent strigns as search terms to identify applications. As expected, some of these identified applications were from our control experiments, as we restricted inspection of plaintext `invariants` to those also found in the control experiments. Nevertheless, many chunks were generic (e.g., protocol framing, common keys), allowing our us to identify apps beyond the control set. Interestingly, CryptoFilter also identified several applications that had previously been analyzed by other researchers and were confirmed to have issues (e.g., exposed PII or non-standard, unencrypted transport) [22, 24, 26, 35].

5.3.4 Privacy Threat Confirmation. Just because an application showed up in Table 3, does not mean it has actual vulnerabilities, although many of these apps had been previously confirmed to have transport-layer security issues. Confirmation *via* manual reverse engineering is thus necessary. After identifying the list of candidate applications, we performed manual reverse engineering for eleven of the applications. We did not include applications that had been previously analyzed, were not part of our control experiments, and were not browsers as those are frequently analyzed. For applications

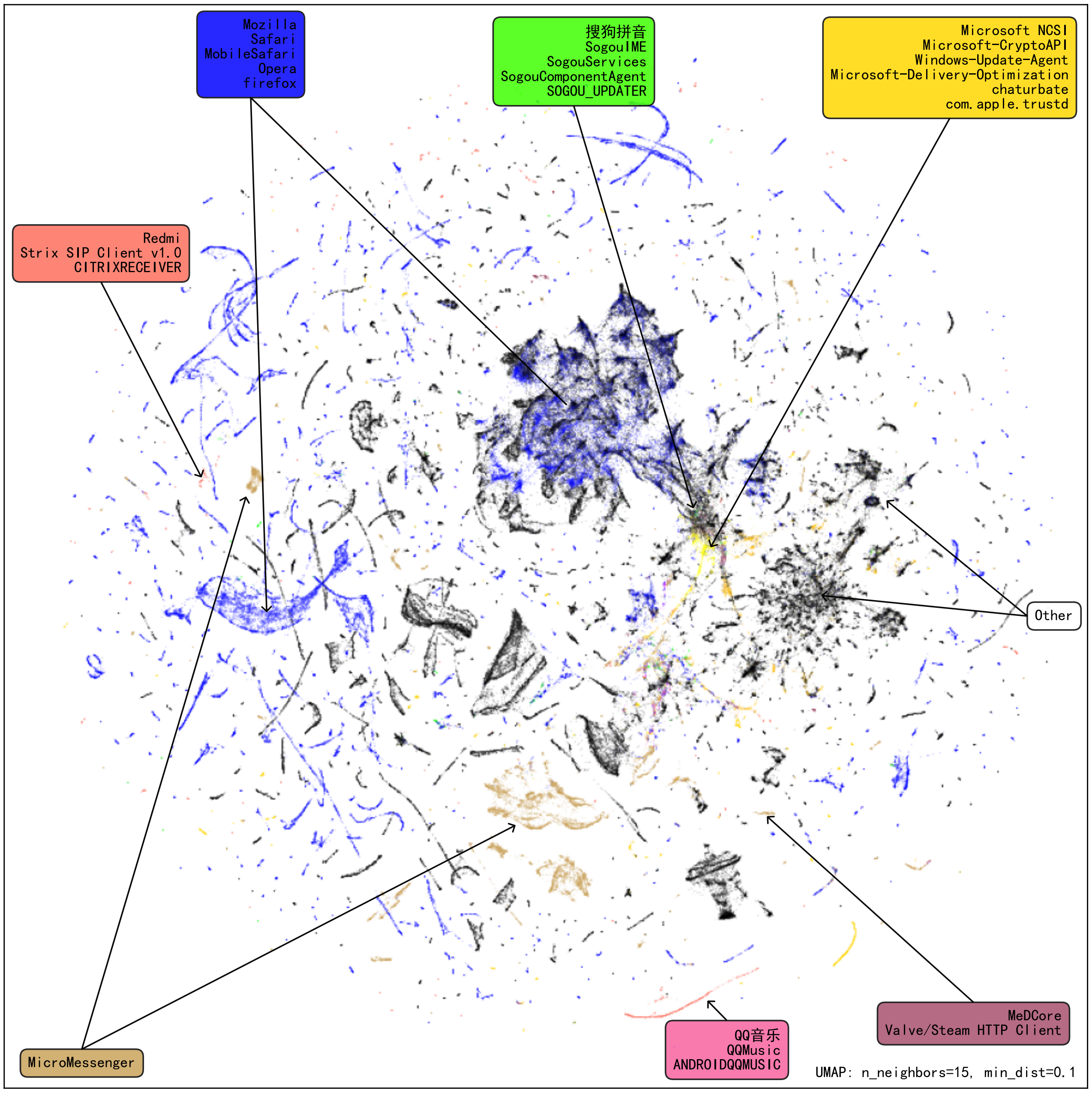


Figure 7: Top2Vec UMAP Plot: Top2Vec embedding of invariants for 50 apps. Dimensionality reduction using UMAP. Visualization representation of unencrypted/poorly encrypted apps present in ISP-M.

originating from the Chinese market, we downloaded them from the Tencent Appstore and used Google Lens for translation [4].

[4]Many Chinese apps require real-name verification to access full functionality. Although we were able to register for some of these apps using a U.S. phone number, prior work has shown that app (security) behavior may vary depending on the region of registration.

Each target application was analyzed on a Google Pixel 7a device using a combination of static and dynamic analysis techniques. We explored different views within the apps to exercise different code paths, while collecting packet captures that we later manually analyzed for instances of sensitive data (e.g., IP/MAC addresses, identifiers, geographic coordinates, and/or non-HTTPS URLs or HTTP redirects) transmitted in plaintext. We found that eight out of the eleven applications leaked a broad range of PII. Below, we provide an assessment for the most egregiously violating application, Kuaishou, with the remaining findings provided in Appendix A.3.

Kuaishou: Kuaishou stood out due to the breadth of information transmitted without encryption. The Kuaishou app includes several advertising and location libraries, such as AMap, which use HTTP to transmit user telemetry to their servers for targeted advertising. Further investigation revealed that it transmits various identifiers, coarse (e.g., country and city IDs) and fine-grained (e.g., GPS coordinates) geographic information, network connection type (e.g., Wi-Fi), the client's public IP address, device OS, model, processor architecture, and other metadata related to phone and app configuration. This information is transmitted to remote servers in different geographic locations. An adversary on the network path could exploit this information to track users across networks. An example packet payload of Kuaishou is included in Figure 1.

6 Ethical Considerations

Raw network traffic that contains real user data is inherently sensitive. Here, we describe the ethical considerations involved in our work, and outline the procedural and technical measures implemented to mitigate these risks.

Foremost among the our ethical considerations processing user traffic in an upstream Internet gateway. For this, we sought approval from our institution's Institutional Review Board (IRB) and received a "Not Regulated" determination. We also cleared our research with the institution's network operators and the officers from the Information Technology department.We note that all live user traffic processing was conducted on a dedicated monitoring server managed by ISP-M, which has well-established ethics and privacy protocols to guide such projects. A ISP-M representative provided oversight throughout the deployment and testing phases, and ensured that the access was restricted to select members of the team on a least-privilege basis. The monitoring server received only mirrored copies of the traffic, meaning the actual routing and service quality of end-users remained unaffected.

Our work distinguishes itself from earlier studies (e.g., [10, 11, 21, 51]) by committing to neither storing nor inspecting raw packet payloads except for the `invariants` that have demonstrated sufficient dispersion across users of the network. Nonetheless, given that some members of our team are external researchers to the ISP (including two non-citizens of the country where the ISP is located), even viewing these invariants can still raise ethical concerns. For this, we have implemented additional safeguards to set a higher standard for similar research. Specifically, we generate a control set of invariants in a controlled environment, following the procedure outlined in § 5.1.2. Next, when CryptoFilter generates invariants from ISP gateway traffic, we restrict ourselves to viewing only those invariants that also exist in the control set. This ensures that we only log or view invariants that are known a priori not to correlate with any individual on the public network.

7 Discussion and Conclusion

The question we stared with was, *Can we analyze unencrypted traffic from real users in a privacy-preserving way?* Our design and results answer this question in the affirmative. Specifically, this paper described a design and results that meet the three basic requirements of being protocol agnostic (§ 5.3.1), privacy-preserving (§ 4.3), and scalable (§ 5.2).

The biggest challenge moving forward will be to identify a balance between the privacy-preserving requirement and the utility of the data for empowering network administrators to identify the threats to their users and networks. A limitation of our approach in this paper is that if an invariant is not co-located with known invariants then we have no visibility into what it is or its semantic meaning. CryptoFilter, as presented in this paper, is an excellent complement to existing approaches such as filtering rules for known protocol fields (e.g., user agents strings), but CryptoFilter also gives us visibility into proprietary unencrypted protocols that our current design cannot attribute to specific applications. 90% of the unencrypted or poorly encrypted traffic that CryptoFilter identified had no user agent string. A focus of future work will be to identify these threats to users and networks.

Despite this limitation, CryptoFilter provided valuable insights into threats to the users and networks served by the Internet gateway where we deployed it for this study. Figure 7 gives a clear picture of what, among apps that include a user agent string, the network administrators for this network should be aware of. This includes apps previously known to have problems with poor or missing encryption [35] (e.g., Sogou [25], com.appled.trustd [40], KakaoTalk [45], QQ Browser [22], and Redmi Browser [44]), and others that warrant further investigation (e.g., ESPN, MeDCore, Roku, RedDownload, and chaturbate.com). It includes update mechanisms that are most likely authenticated correctly out of band, but should be investigated (e.g., Debian APT-HTTP and WindowsUpdateAgent). It also includes apps that were not previously known to expose PII in and unencrypted or poorly encrypted form, but clearly can be seen to do so with minimal reverse engineering effort (e.g., QQ Music or iQIYI). And, while much of the traffic identified in Figure 7 can be attributed to users going to unencrytped websites with browsers (e.g., Safari, or Mozilla – which is reported as the user agent string for Chrome and other Chrome-based browsers), the overwhelming majority of apps identified are not browsers. Not only do non-browser apps represent a large fraction of apps identified by CryptoFilter as lacking proper transport-layer encryption, but a large fraction of those (not shown in the figure because they lack a user-agent string) use custom protocols that are not built on top of HTTP. Empowering network analysts with visibility into all these different types of unencrypted and poorly encrypted traffic on their networks is critical for security and privacy, and we see CryptoFilter as a significant first step towards achieving this goal.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Number CNS-2141547 and the Open Technology Fund's Information Controls Fellowship Program.

References

[1] 2016. Art.4 GDPR – Definitions - General Data Protection Regulation (GDPR) — gdpr-info.eu. online. https://gdpr-info.eu/art-4-gdpr/
[2] Nader Ammari, Gustaf Björksten, Peter Micek, and Deji Olukotun. 2023. The Rise of Mobile Tracking Headers: How Telcos Around the World Are Threatening Your

Privacy. online. https://www.accessnow.org/wp-content/uploads/2023/07/AIBT-Report.pdf

[3] Dimo Angelov and Diana Inkpen. 2024. Topic Modeling: Contextual Token Embeddings Are All You Need. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 13528–13539. "https://aclanthology.org/2024.findings-emnlp.790"

[4] Marco Aresu, Davide Ariu, Mansour Ahmadi, Davide Maiorca, and Giorgio Giacinto. 2015. Clustering android malware families by http traffic. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 128–135.

[5] David Blei, Andrew Ng, and Michael Jordan. 2001. Latent dirichlet allocation. *Advances in neural information processing systems* 14 (2001).

[6] Riccardo Bortolameotti, Thijs van Ede, Marco Caselli, Maarten H Everts, Pieter Hartel, Rick Hofstede, Willem Jonker, and Andreas Peter. 2017. Decanter: Detection of anomalous outbound http traffic by passive application fingerprinting. In *Proceedings of the 33rd Annual computer security applications Conference*. 373–386.

[7] William Brockelsby and Rudra Dutta. 2019. A graded approach to network forensics with privacy concerns. In *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 292–297.

[8] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. 2013. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 160–172.

[9] CNCERT/CC. 2020. 2020 China Internet Network Security Report. online. https://www.cert.org.cn/publish/main/upload/File/2020%20Annual%20Report.pdf

[10] Shuaifu Dai, Alok Tongaonkar, Xiaoyin Wang, Antonio Nucci, and Dawn Song. 2013. Networkprofiler: Towards automatic fingerprinting of android apps. In *2013 proceedings ieee infocom*. IEEE, 809–817.

[11] Louis F DeKoven, Audrey Randall, Ariana Mirian, Gautam Akiwate, Ansel Blume, Lawrence K Saul, Aaron Schulman, Geoffrey M Voelker, and Stefan Savage. 2019. Measuring security practices and how they impact security. In *Proceedings of the Internet Measurement Conference*. 36–49.

[12] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. 2013. {ZMap}: fast internet-wide scanning and its security applications. In *22nd USENIX Security Symposium (USENIX Security 13)*. 605–620.

[13] Emanuele Faranda. [n. d.]. GitHub - emanuele-f/PCAPdroid: No-root network monitor, firewall and PCAP dumper for Android — github.com. online. https://github.com/emanuele-f/PCAPdroid

[14] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. 2017. Measuring HTTPS Adoption on the Web. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1323–1338. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/felt

[15] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete mathematics & theoretical computer science* Proceedings (2007).

[16] FTC. 2024. How "location, location, location" can lead to "enforcement, enforcement, enforcement" — ftc.gov. online. https://www.ftc.gov/business-guidance/blog/2024/01/how-location-location-location-can-lead-enforcement-enforcement-enforcement

[17] FTC. 2024. P = NP? Not exactly, but here are some research questions from the Office of Technology. — ftc.gov. online. https://www.ftc.gov/policy/advocacy-research/tech-at-ftc/2024/05/p-np-not-exactly-here-are-some-research-questions-office-technology

[18] Hao Fu, Zizhan Zheng, Aveek K Das, Parth H Pathak, Pengfei Hu, and Prasant Mohapatra. 2016. Flowintent: Detecting privacy leakage from user intention to network traffic mapping. In *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 1–9.

[19] Robert Graham. 2013. GitHub - robertdavidgraham/masscan: TCP port scanner, spews SYN packets asynchronously, scanning entire Internet in under 5 minutes. github.com. online. https://github.com/robertdavidgraham/masscan

[20] Danny Yuxing Huang, Noah Apthorpe, Frank Li, Gunes Acar, and Nick Feamster. 2020. Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 2 (2020), 1–21.

[21] Sakshi Jain, Mobin Javed, and Vern Paxson. 2016. Towards Mining Latent Client Identifiers from Network Traffic. *Proc. Priv. Enhancing Technol.* 2016, 2 (2016), 100–114.

[22] Ron Deibert Jeffrey Knockel, Adam Senft. 2016. WUP! There It Is: Privacy and Security Issues in QQ Browser - The Citizen Lab — citizenlab.ca. online. https://citizenlab.ca/2016/03/privacy-security-issues-qq-browser/

[23] Hyang-Ah Kim and Brad Karp. 2004. Autograph: Toward Automated, Distributed Worm Signature Detection.. In *USENIX security symposium*, Vol. 286. San Diego, CA.

[24] Jeffrey Knockel, Zoë Reichert, and Mona Wang. 2023. *"Please do not make it public": Vulnerabilities in Sogou Keyboard encryption expose keypresses to network eavesdropping*. Technical Report. https://citizenlab.ca/2023/08/vulnerabilities-in-sogou-keyboard-encryption/

[25] Jeffrey Knockel, Zoë Reichert, and Mona Wang. 2023. *"Please do not make it public": Vulnerabilities in Sogou Keyboard encryption expose keypresses to network eavesdropping*. Technical Report. https://utoronto.scholaris.ca/items/ab81bbdd-3bd0-4aab-8558-bc5a95e6b967

[26] Jeffrey Knockel, Zoë Reichert, and Mona Wang. 2024. *The not-so-silent type: Vulnerabilities across keyboard apps reveal keystrokes to network eavesdroppers*. Technical Report. https://citizenlab.ca/2024/04/vulnerabilities-across-keyboard-apps-reveal-keystrokes-to-network-eavesdroppers/

[27] Jeffrey Knockel, Adam Senft, and Ronald Deibert. 2016. Privacy and Security Issues in BAT Web Browsers. In *6th USENIX Workshop on Free and Open Communications on the Internet (FOCI 16)*. USENIX Association, Austin, TX. https://www.usenix.org/conference/foci16/workshop-program/presentation/knockel

[28] Jeffrey Knockel, Mona Wang, and Zoë Reichert. 2023. *Chinese Keyboard App Vulnerabilities Explained*. Technical Report. https://citizenlab.ca/2024/04/chinese-keyboard-app-vulnerabilities-explained/

[29] Beau Kujath, Jeffrey Knockel, Paul Aguilar, Diego Morabito, Masashi Crete-Nishihata, and Jedidiah R Crandall. 2024. Analyzing Prominent Mobile Apps in Latin America. *Free and Open Communications on the Internet* (2024).

[30] Thomas K Landauer and Susan T Dumais. 1997. A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review* 104, 2 (1997), 211.

[31] Pellaeon Lin. 2021. *TikTok vs Douyin A Security and Privacy Analysis*. Technical Report.

[32] Bill Marczak, Nicholas Weaver, Jakub Dalek, Roya Ensafi, David Fifield, Sarah McKune, Arn Rey, John Scott-Railton, Ron Deibert, and Vern Paxson. 2015. An Analysis of China's "Great Cannon". In *5th USENIX Workshop on Free and Open Communications on the Internet (FOCI 15)*. USENIX Association, Washington, D.C. https://www.usenix.org/conference/foci15/workshop-program/presentation/marczak

[33] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. 2018. UMAP: Uniform Manifold Approximation and Projection. *The Journal of Open Source Software* 3, 29 (2018), 861.

[34] Stanislav Miskovic, Gene Moo Lee, Yong Liao, and Mario Baldi. 2015. Appprint: automatic fingerprinting of mobile applications in network traffic. In *Passive and Active Measurement: 16th International Conference, PAM 2015, New York, NY, USA, March 19-20, 2015, Proceedings 16*. Springer, 57–69.

[35] Jeffrey Knockel Mona Wang, Pellaeon Lin. 2024. Should We Chat, Too? Security Analysis of WeChat's MMTLS Encryption Protocol - The Citizen Lab — citizenlab.ca. online. https://citizenlab.ca/2024/10/should-we-chat-too-security-analysis-of-wechats-mmtls-encryption-protocol/

[36] James Newsome, Brad Karp, and Dawn Song. 2005. Polygraph: Automatically generating signatures for polymorphic worms. In *2005 IEEE Symposium on Security and Privacy (S&P'05)*. IEEE, 226–241.

[37] ntop.org. [n. d.]. PF_RING — ntop.org. online. https://www.ntop.org/products/packet-capture/pf_ring/

[38] Ruoming Pang and Vern Paxson. 2003. A high-level programming environment for packet trace anonymization and transformation. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. 339–351.

[39] Craig Partridge and Mark Allman. 2016. Ethical considerations in network measurement papers. *Commun. ACM* 59, 10 (2016), 58–64.

[40] Jeffrey Paul. 2020. Your Computer Isn't Yours. online. https://sneak.berlin/20201112/your-computer-isnt-yours/ Accessed: 9 March 2025.

[41] Eric Pauley and Patrick McDaniel. 2023. Understanding the ethical frameworks of internet measurement studies. In *2nd International Workshop on Ethics in Computer Security*.

[42] Roberto Perdisci, Wenke Lee, and Nick Feamster. 2010. Behavioral clustering of http-based malware and signature generation using malicious network traces.. In *NSDI*, Vol. 10. 14.

[43] Vijay Prakash, Sicheng Xie, and Danny Yuxing Huang. 2022. Inferring software update practices on smart home iot devices through user agent analysis. In *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*. 93–103.

[44] Esther Rodriguez, Lobsang Gyatso, Tenzin Thayai, and Jedidiah R Crandall. 2025. Revisiting BAT Browsers: Protecting At-Risk Populations from Surveillance, Censorship, and Targeted Attacks. *Free and Open Communications on the Internet* (2025).

[45] Dawin Schmidt. 2016. *A Security and Privacy Audit of KakaoTalk's End-to-End Encryption*. Technical Report. https://www.diva-portal.org/smash/get/diva2:1046438/FULLTEXT01.pdf

[46] Darren Shou. 2012. Ethical considerations of sharing data for cybersecurity research. In *Financial Cryptography and Data Security: FC 2011 Workshops, RLCPS and WECSR 2011, Rodney Bay, St. Lucia, February 28-March 4, 2011, Revised Selected Papers 15*. Springer, 169–177.

[47] Sudheesh Singanamalla, Esther Han Beol Jang, Richard Anderson, Tadayoshi Kohno, and Kurtis Heimerl. 2020. Accept the risk and continue: Measuring the long tail of government https adoption. In *Proceedings of the ACM Internet*

Measurement Conference. 577–597.

[48] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. 2004. Automated Worm Fingerprinting. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6* (San Francisco, CA) *(OSDI'04)*. USENIX Association, USA, 4.

[49] Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation* 28, 1 (1972), 11–21.

[50] Ryan Stevens, Clint Gibler, Jon Crussell, Jeremy Erickson, and Hao Chen. 2012. Investigating user privacy in android ad libraries. In *Workshop on Mobile Security Technologies (MoST)*, Vol. 10. 195–197.

[51] Alisha Ukani, Ariana Mirian, and Alex C Snoeren. 2021. Locked-in during lockdown: undergraduate life on the internet in a pandemic. In *Proceedings of the 21st ACM Internet Measurement Conference*. 480–486.

[52] Eline Vanrykel, Gunes Acar, Michael Herrmann, and Claudia Diaz. 2017. Leaky birds: Exploiting mobile application traffic for surveillance. In *Financial Cryptography and Data Security: 20th International Conference, FC 2016, Christ Church, Barbados, February 22–26, 2016, Revised Selected Papers 20*. Springer, 367–384.

[53] Ke Wang, Gabriela Cretu, and Salvatore J Stolfo. 2005. Anomalous payload-based worm detection and signature generation. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 227–246.

[54] Knockel J. Wang M., Lin P. [n. d.]. Should We Chat? Privacy in the WeChat Ecosystem. https://citizenlab.ca/2023/06/privacy-in-the-wechat-ecosystem-full-report/ Accessed : 31 May 2024.

[55] Ning Xia, Han Hee Song, Yong Liao, Marios Iliofotou, Antonio Nucci, Zhi-Li Zhang, and Aleksandar Kuzmanovic. 2013. Mosaic: Quantifying privacy leakage in mobile networks. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. 279–290.

[56] Zeek [n. d.]. The Zeek Network Security Monitor. online. https://zeek.org/

A Appendix

A.1 App Attribution

Table 4 Flow Attribution. Aggregate percent of flows attributed either a Domain, User Agent, or Javascript library using different bootstrapping seeds. Percent of flows labeled.

	Total	Domains	User Agents	JavaScript
Random	15%	.38%	.29%	15%
Control Apps	74%	23%	19%	52%
Javascript	76%	47%	46%	28%
HTTP Methods	55%	17%	9%	39%
Location	64%	19%	14%	44%
User-Agent	60%	20%	12%	41%

Table 4 summarizes the percentage of flows we were able to label using the simplified methodology described in § 5.3.3. Each experiment type (rows) had a percentage of flows that we could associate to specific Domain names, applications *via* its User Agent string, or JavaScript libraries. In the case of JavaScript, we used the decoded invariants to check for their presence in JavaScript files downloaded in plaintext from the control applications. This approach has two limitations. First, a single or small number of invariants could come from non-JavaScript sources. We address this by setting a threshold of twenty for labeling flows with specific JavaScript files names. The intuition being that the more invariants that occur from a specific JavaScipt file, the less likely it is to have occurred erroneously from a different source. The second limitation is that JavaScript is ubiquitous on the Internet. This fact makes it unlikely that any real JavaScript transmitted came from a specific source from our control data, even with thresholding.

Algorithm 1 Reconstructing Invariants from chunks

Input: List of chunks within a packet `p`, each with an `offset`, sorted by `offset`; chunk size k
Output: List of reconstructed merged invariants

```
1: Initialize start as the lowest offset
2: Initialize end as start + $k$
3: Initialize invar as p[start:end]
4: while there are unprocessed offset do
5:     while nextoff $\leq$ end do
6:         invar += p[nextoff:nextoff+$k$]
7:         Update end to max(end, nextoff + $k$)
8:     end while
9:     Add invar to the list of reconstructed invariants
10:    Set start to the lowest unprocessed offset
11:    Set end to start + $k$
12:    Reset invar to p[start:end]
13: end while
14: return List of reconstructed invariants
```

A.2 Post-processing

Algorithm 1 describes the partial payload reconstruction algorithm we used during the threat mining use-case in 5.3.2.

One application of post-processing is mining privacy threats. To do this, we first have to reconstruct flows from the invariant portions of the packets. We note that `invariants` may still contain overlapping segments. For an m-byte partial payload (e.g., UniqueUserID=), CryptoFilter will likely output each of its k-byte invariants (e.g., UniqueUs, niqueUse, *etc.*) after the sifting stages. To reconstruct partial payloads from its component invariants, the surrounding context of each chunk needs to be considered. Specifically, every time an invariant is observed in a passing packet, the system records the flow identifier (the 5-tuple), the packet number within the flow, and the offset of the invariant within the packet payload. Next, adjacent or overlapping invariants are merged to form the partial payload.

A.3 Privacy Threat Confirmation

Table 5 summarizes the privacy and threat exposures we manually confirmed for eleven applications from § 5.3.4. To identify the specific applications, we used their user agent string and domain names present in their traffic as search terms in Google, Google Play store, Brave, GitHub, Reddit, and X. Upon identifying the applications, we downloaded them onto a Google Pixel 7a for analysis. We then created an account if need, agreed to terms of service, and other pre-initialization requirements. We then selected the application from the PCAPdroid user interface to capture its traffic. We then ran explored the application's user interface for 5-10 minutes as in the control experiments. To identify privacy and threat exposures, we searched for ASCII encoded strings both by searching from previously known strings from our control experiments and by running the Linux "strings" command on the packet captures. We then loaded the APK into jadx and searched for instances of the exposures to confirm they were present in the application and understand their semantics. The exposures are separated in three broad categories: PII (personally identifiable information, Loc (location), and Other.

PII: includes privacy exposures originating from various identifiers (IDs), typically keys from key-values pairs with the string "id" or "ID" in the name, MAC addresses, Wifi MAC addresses,

Table 5 Summary of Identified Insecure Transmission for the Manually Analyzed Apps. 🟥 indicates a confirmed instances of a specific privacy risk type.

Applications		Kuaishou	Fliggy	Quark Browser	Flush	Ctrip	Thunder	Dianping	Meituan	Iqiyi	MedCore	ESPN
PII	IDs	🟥	🟥	🟥	🟥	⬜	🟥	⬜	⬜	⬜	🟥	🟥
	MAC	🟥	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜
	Wifi MAC	🟥	🟥	🟥	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜
	IMSI/IMEI	⬜	⬜	🟥	⬜	⬜	⬜	⬜	⬜	🟥	⬜	⬜
	Client IP	🟥	🟥	🟥	⬜	⬜	⬜	🟥	🟥	🟥	⬜	⬜
Loc	MCC+MNC	🟥	🟥	🟥	⬜	⬜	⬜	🟥	⬜	⬜	⬜	⬜
	GPS	🟥	🟥	🟥	⬜	⬜	🟥	⬜	🟥	⬜	⬜	⬜
	LAC / Zip Code	⬜	⬜	⬜	⬜	⬜	🟥	🟥	⬜	⬜	⬜	🟥
Other	HTTPSite	🟥	🟥	🟥	⬜	⬜	🟥	⬜	⬜	⬜	⬜	🟥
	Redirects	🟥	⬜	🟥	⬜	⬜	⬜	⬜	⬜	⬜	⬜	🟥
	Hardware Info	🟥	🟥	⬜	⬜	⬜	🟥	🟥	⬜	⬜	⬜	⬜
	OS Info	🟥	🟥	🟥	🟥	⬜	🟥	🟥	🟥	🟥	⬜	🟥
	Device Info	🟥	🟥	🟥	🟥	⬜	🟥	🟥	🟥	🟥	⬜	🟥
	Screen Size	🟥	⬜	⬜	⬜	🟥	⬜	🟥	🟥	⬜	⬜	🟥
	Net. Acc. Type	🟥	🟥	🟥	⬜	⬜	🟥	⬜	⬜	🟥	⬜	⬜
	ELF	⬜	🟥	⬜	⬜	⬜	🟥	⬜	⬜	⬜	⬜	⬜
	Network Info	🟥	🟥	🟥	⬜	⬜	🟥	🟥	🟥	⬜	⬜	⬜

IMEI/IMSI, and Client IP addresses (the public address of our analysis device). An attacker could use this information to profile specific users and track them as they traverse networks, changing their IP address.

Loc: Encompasses location exposures, such as MCC+MNC codes, GPS coordinates, and Loc or Zip codes. This information is especially concerning, especially for at-risk users like activists, journalists, or dissidents as there location could be used for targeting by an adversary.

Other: Encompasses any other type of privacy or threat exposure. In this category, we observed direct and indirect references to unencrypted URLs, HTTP redirect messages. These could be used force a server to download compromised resources or redirect them to an attacker server. Hardware, Operating System (OS), Device information, screen size, Network Info, and Net. Acc. Type (Network Access Type), for example, Wifi or Verizon, which could provide additional profiling features or reconnaissance and later targeted exploitation. Finally, we observed ELF (Executable and Linkable Format) files being received by some of the applications. If the downloading of such a file is not properly authenticated, integrity and version checked, an attacker could potentially replace the ELF with malware or a vulnerable version of the software.