User Search with Knowledge Thresholds in Decentralized Online Social Networks

Benjamin Greschbach, Gunnar Kreitz, and Sonja Buchegger

KTH Royal Institute of Technology
School of Computer Science and Communication
Stockholm, Sweden
{bgre, gkreitz, buc}@csc.kth.se

Abstract. User search is one fundamental functionality of an Online Social Network (OSN). When building privacy-preserving Decentralized Online Social Networks (DOSNs), the challenge of protecting user data and making users findable at the same time has to be met. We propose a user-defined knowledge threshold ("find me if you know enough about me") to balance the two requirements. We present and discuss protocols for this purpose that do not make use of any centralized component. An evaluation using real world data suggests that there is a promising compromise with good user performance and high adversary costs.

Keywords: Decentralized Online Social Networks, Privacy, User Search

1 Introduction

Popular Online Social Networks (OSNs) are logically centralized systems. The massive information aggregation at the central provider inherently threatens user-privacy. Data leakages, whether intentional (e. g., selling of user data to third parties) or unintentional (e. g., by attacks from outsiders), happen regularly[1]. Motivated by these insights, decentralization has been proposed to mitigate these threats. When decentralizing a system, two challenges have to be met: to implement equal functionality without centralized components, and to provide user privacy under a significantly different threat model.

Here, we look at the functionality of user search, i. e., the lookup of a system-specific user identifier (e. g., a URI of a profile) based on information about the user (e. g., name, city, affiliation). The ability to search for users, in conjunction with other ways of traversing the social graph (e. g., friendlist of friends), is a basic building block of an OSN that allows users to find each other and thereby establish links.

[1] To name only two examples: Twitter leaking data from 250K users in February 2013 (http://blog.twitter.com/2013/02/keeping-our-users-secure.html), Facebook selling user data (http://www.telegraph.co.uk/technology/facebook/8917836/Facebook-faces-EU-curbs-on-selling-users-interests-to-advertisers.html).

1.1 Our Contribution

We propose and evaluate protocols to support user search in a decentralized OSN that shield user data from searchers who know less than a user-specified threshold amount of information about the target. To our knowledge, this consideration is a novel form of knowledge-based access control. This type of restriction was inspired by an observation by Fong et al. [6] that being able to reach a user in an OSN is an integral part of access control in such systems.

We evaluate our protocols using real world data from the U.S. census to relate the performance for legitimate users to the costs of an adversary that tries to guess unknown information.

1.2 Related Work

To the best of our knowledge the privacy-findability tradeoff has not been investigated in this context. The closest example is user search in Skype. However, as far as we know, their protocol has not been described in detail, but only via external measurement studies, such as one by Baset and Schulzrinne [2].

Most user search functionalities, including ours, search for users within the global user database of the OSN, independently of who searches. In contrast, we note that recently, Facebook has debuted Graph search [4], which ties searching to the social graph, and where the goal is not only to find users, but also content. Several other approaches of personalized searching for content in an OSN have also be discussed, e. g., by Bai et al. [1] in a decentralized setting.

Although designed specifically to search for users in a DOSN, some challenges are shared with constructing a general purpose search in a peer-to-peer (P2P) setting. This has been studied by e. g., Li et al. [7], and Bender et al. [3]. There is also a commercial search engine using P2P, Faroo [5]. Two differences are our focus on access control and privacy, and the significantly smaller amount of information to be indexed in our setting. Similar to these proposals, we also build upon a Distributed Hash Table (DHT) as a core component to realize our functionality.

2 Decentralized User Search Protocol

We design a search protocol for a decentralized system, so we cannot assume any trusted third party or central search provider to be available. Instead, we use a DHT to register and look up search terms, as it is a common component of Decentralized Online Social Networks (DOSNs). As the DHT runs on nodes participating in the system, we must also protect the privacy of the participants against these nodes.

We propose two protocols, both designed to index and retrieve information in a DHT in a protected way. Our protocols provide two operations. A *register* operation, where users enter information that allows others to find them based on certain attributes, and a *search* operation that, given a set of search terms,

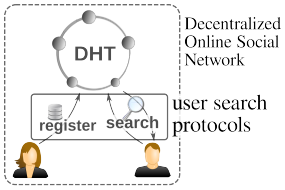

Fig. 1. System overview: The search protocols are one component of the DOSN and makes use of a DHT.

returns the set of matching user identifiers. In a next step, out of the scope of the search protocols described here, these user identifiers can be used to view public profiles, and to send a message or friend request to the sought user. Figure 1 illustrates the search functionality.

2.1 Protocol Specification

We consider a searcher, who wants to find a searchee. The searchee registers searchable information about herself in the DHT by choosing a number n of attribute labels l_i (e. g., lastname, firstname, city) and assigning each one value v_i. This label-value pair (denoted as attribute a_i) is mapped to a user identifier *uid* of the searchee. Upon registration the searchee specifies a threshold number t of attributes which the searcher must know in order to obtain the user identifier.

2.2 Storing Values in the DHT

The DHT holds a mapping from user attributes to user identifiers, but this mapping must be protected, also against the nodes in the DHT. To this end, we propose a protocol that alters how values are added and retrieved from the DHT. The required property is to retain standard DHT functionality, while nodes in the DHT do not learn plaintexts of keys or values.

When storing a *key-value* pair the *key* is fed into a Key Derivation Function (KDF) together with a global salt *gSalt*, yielding the DHT-key for the `put` and `get` operations of the DHT. The *value* is encrypted using a secret that is derived from a random salt *salt* and the *key* (the attribute information, in our case). The *salt* is stored together with the ciphertext on the right hand side of the mapping. In short, the mapping of a *key-value* pair in the DHT looks like this:

$$\mathrm{KDF}(gSalt,key) \mapsto salt||\mathrm{encrypt}_{\mathrm{KDF}(salt,key)}(value)$$

The *gSalt* has to be publicly available for all users to allow the lookup of any attributes. This invalidates the purpose of a salt, as pre-computing tables to reverse the left hand side becomes possible again. Nevertheless, we suggest to keep the *gSalt* as it at least requires the pre-computation attack to be targeted

to each specific instance of our system and out-of-the-shelf pre-computed tables for the used KDF cannot be employed.

The *salt* is an individual random number different for every entry. Note that it in particular has to be different from *gSalt* as otherwise any DHT node could decrypt the *value* of items it stores, using the left hand side (without knowing the *key*).

2.3 Scheme 1: Storing all Allowed Attribute Combinations

We want a searcher to prove knowledge of a threshold number of attributes before obtaining the user identifier. One direct approach to achieve this is to map the user identifier only from attribute concatenations of the threshold length. If the searchee registered e. g., seven attributes and specified that at least four of them are necessary to find her *uid*, we would store the following $\binom{7}{4} = 35$ combinations:
$a_1||a_2||a_3||a_4 \mapsto uid$
$a_1||a_2||a_3||a_5 \mapsto uid$
...
$a_4||a_5||a_6||a_7 \mapsto uid$

where $a_i = (u_i, v_i)$, u_i attribute labels and v_i attribute values. We assume there is a global order of attribute labels, and attributes are inserted sorted by label.

Algorithm 1 Registration (Scheme 1)

1: $l_1, \ldots, l_n \leftarrow$ User.input("Choose searchable attribute labels (e. g., name,city,...)")
2: $v_1, \ldots, v_n \leftarrow$ User.input("Enter values (your name, your city,...)")
3: $a_i \leftarrow l_i||v_i$ // for $i = 1 \ldots n$
4: $t \leftarrow$ User.input("Enter threshold number of attributes necessary to find you.")
5: **for** all ordered sequences $a_p|| \ldots ||a_q$ of length t **do**
6: $key \leftarrow a_p|| \ldots ||a_q$
7: $dhtkey \leftarrow$ KDF($gSalt$,key)
8: $salt \leftarrow$ generateSalt()
9: $value \leftarrow uid$
10: $dhtvalue \leftarrow salt||$ encrypt$_{\mathrm{KDF}(salt,key)}(value)$
11: DHT.put($dhtkey$,$dhtvalue$)
12: **end for**

Algorithms 1 and 2 describe the protocol in more detail. For registration, all attribute combinations of length t are mapped to the user identifier and stored in the DHT according to the procedure described in Section 2.2. When searching, all possible combinations of the provided search attributes are ordered and used to query the DHT (after the Section 2.2 transformation). The result will contain the user identifier of the searchee (and possibly more hits from other users that registered the same attributes).

Algorithm 2 Search (Scheme 1)

1: $l_1, \ldots, l_s \leftarrow$ User.input("Choose attribute labels to search for (e. g., name,city,...)")
2: $v_1, \ldots, v_s \leftarrow$ User.input("Enter attribute values (a name, a city,...)")
3: $a_i \leftarrow l_i || v_i$ // for $i = 1 \ldots s$
4: **for** $i \leftarrow s, \ldots, 1$ **do**
5: **for** all ordered sequences $a_p || \ldots || a_q$ of length i **do**
6: $key \leftarrow a_p || \ldots || a_q$
7: $dhtkey \leftarrow \mathrm{KDF}(gSalt, key)$
8: $salt, ciphertext \leftarrow \mathrm{DHT.get}(dhtkey)$
9: $uid \leftarrow \mathrm{decrypt}_{\mathrm{KDF}(salt,key)}(ciphertext)$ // decrypt successful = uid found
10: **end for**
11: **end for**

2.4 Scheme 2: Storing Each Attribute Individually

An alternative approach, overcoming the large DHT storage overhead of scheme 1, is to store each attribute individually. In order to require a threshold number of attributes to find the user identifier, a single attribute does not map directly to the *uid* but to an encrypted version. The key used for the encryption is derived using a secret sharing scheme and one share is stored with each of the attributes. The indirection via the KDF allows us to independently tune the costs for requesting shares for one attribute (determined by the DHT latency and the KDF described in Section 2.2) and for trying to combine them (determined by the KDF used here). Furthermore, a bloom filter bf_i is attached to each share, to help finding the right shares to combine with, which is important for popular attributes with large response sets:

$a_1 \mapsto s_1 || bf_1 || salt_1 ||$ $\mathrm{encrypt}_{\mathrm{KDF}(salt_1, sk)}(\mathrm{uid})$
...
$a_n \mapsto s_n || bf_n || salt_n ||$ $\mathrm{encrypt}_{\mathrm{KDF}(salt_n, sk)}(\mathrm{uid})$
where sk can be recovered with t of the shares $s_1 \ldots s_n$.

Encrypting the user identifier under different keys (due to different salts) also yields different ciphertexts. This is important as otherwise, searching e. g., for a certain firstname-lastname combination and getting the same ciphertext on the right hand side, reveals that there is a person with that firstname-lastname combination registered in the system, even if the person specified that more than two attributes are necessary to find her. The bloom filter that is stored with each share is created using all other $n-1$ shares belonging to the same key sk. The bloom filters will furthermore be randomized (expanded and filled with random values) in order to reduce the probability that two related shares have too similar bloom filters.

The full paper will explain this approach in more detail, including discussions of design decisions and properties as well as pseudocode.

3 Threat Model

We consider all information that the user gives away or generates while interacting with the system as possibly sensitive. This comprises general administrative information (existence in system, date of registration, user-identifiers), entered information during registration (attributes, i. e., label-value pairs) as well as search query data and behavioural data.

3.1 Adversaries and Their Capabilities

All agents in the system can possibly act in malicious ways. This comprises nodes involved in the DHT storage, passive traffic observers and active adversaries (malicious users that can perform search and register operations). Their capabilities range from sniffing traffic, crawling the DHT (performing massive search operations), to actively inserting data into the DHT, performing traffic analysis (e. g., analyzing query sizes), and analyzing data they might store.

3.2 Attack Scenario: Targeted crawling using brute-force

A comprehensive security and privacy analysis of the protocols, would go beyond the scope of this paper. We therefore focus on several specific attacks, and present one of them here (more in the full paper).

We investigate an attack scenario where the adversary tries to crawl the system for users with specific sensitive attributes (fixing them) and wants to find identifying information such as name and city (guessing them) for these users.

4 Privacy Evaluation

We expect the adversary to query the specific attributes of interest and then guessing the missing, identifying attributes by performing an exhaustive search on their value spaces. We assume that a person's first name, last name, and the city the person is located at are among the popular and identifying search attributes.

To get evaluation results reflecting realistic distributions of values for these attributes, data from the U.S. census was used as input for the following calculations; properties and shortcomings thereof are discussed in the full paper.

4.1 Brute-force Probabilities Scheme 1

We investigate the success probability of an adversary, when trying to guess one of the attributes by brute-force, i. e., searching the whole value space. We assume the adversary will try most likely values (those registered by most users according to the value distribution in the population) first. Figure 2 shows the number of combinations to test in order to cover a certain percentage of the

user population. This corresponds to the costs of an adversary as in scheme 1, to try one combination, one DHT `get` operation plus two KDF operations are necessary. For single attributes between 180 and 3000 combinations are enough to find a target with 50% success probability (4600 to 60 Million combinations for 100%). When the combination of two attributes has to be guessed, this increases to around 10^7 combinations for 50% success probability and up to 10^{15} combinations to search the whole value space.

Scheme 2, which we will discuss in more detail in the full paper, can be tuned to achieve comparable security to that of scheme 1, at the cost of slightly more work for legitimate users.

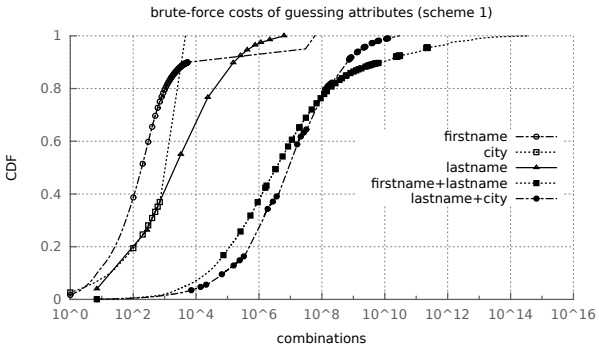

Fig. 2. CDF of brute-force success after trying a certain number of combinations (most likely ones first) for different attributes.

5 Discussion

The results presented in the previous section describe the gap between the search effort of a legitimate user and the cost of an adversary trying to find user identifiers despite knowing fewer attributes than required. For scheme 1, the former is constant in terms of DHT operations, the latter depends on the number and kind of unknown attributes, as shown in Figure 2. The adversary's costs for only one attribute are rather low, as expected. They can be tuned by KDF parameters but this will also affect the performance for legitimate users. The gap increases, however, combinatorially with the number of attributes the adversary has to guess. Already for two unknown attributes this might frustrate an attack: When tuning the KDF operations to take one second (delay for a legitimate user), an adversary with the same computational power as the user would need about 6 weeks to find the correct combination with 50% probability. The gap is not a global system parameter but can be tuned by each user individually (by choosing an individual threshold t for the registered information) but also depends on the adversary's knowledge about a target user.

In the full paper we analyze other kinds of attacks, such as determining the existence of a user in the system using fewer than t attributes and learning other attributes when already knowing t attributes.

6 Conclusion and Future Work

We presented two approaches to realize a targeted user search in a Decentralized Online Social Network. The search protocols implement a knowledge threshold, allowing the users to protect their user identifier from adversaries that do not possess enough information about them while legitimate users, who know enough about the searchee, are able to find her. We described the protocols in detail, sketched a threat model, and evaluated selected properties using real world data. The evaluation yielded insights into the brute-force costs of an adversary, which depend on the user defined knowledge threshold and the knowledge of the adversary about the target user. The results suggest that for common attributes, the proposed protocol offers promising protection against an adversary that knows at least two or three fewer attributes than required by the user.

In the full paper we will explore more attacks and evaluate the second protocol. Furthermore, we suggest improvements for the protocols such as weighting attributes differently when computing the threshold number and possibilities for combining the two approaches.

7 Acknowledgements

Oleksandr Bodriagov and Guillermo Rodríguez Cano contributed to joint discussions of the ideas in Section 2. Some of the ideas were also discussed with Thomas Paul.

References

1. Bai, X., Bertier, M., Guerraoui, R., Kermarrec, A.M., Leroy, V.: Gossiping personalized queries. In: Manolescu, I., Spaccapietra, S., Teubner, J., Kitsuregawa, M., Léger, A., Naumann, F., Ailamaki, A., Özcan, F. (eds.) EDBT. ACM International Conference Proceeding Series, vol. 426, pp. 87–98. ACM (2010)
2. Baset, S., Schulzrinne, H.: An analysis of the Skype peer-to-peer internet telephony protocol. CoRR abs/cs/0412017 (2004)
3. Bender, M., Michel, S., Triantafillou, P., Weikum, G., Zimmer, C.: Minerva: Collaborative p2p search. In: Böhm, K., Jensen, C.S., Haas, L.M., Kersten, M.L., Larson, P.Å., Ooi, B.C. (eds.) VLDB. pp. 1263–1266. ACM (2005)
4. Facebook: Introducing graph search (2013), https://www.facebook.com/about/graphsearch
5. Faroo: P2P search (2013), http://www.faroo.com/hp/p2p/p2p.html
6. Fong, P.W.L., Anwar, M.M., Zhao, Z.: A privacy preservation model for facebook-style social network systems. In: Backes, M., Ning, P. (eds.) ESORICS. LNCS, vol. 5789, pp. 303–320. Springer (2009)
7. Li, J., Loo, B.T., Hellerstein, J.M., Kaashoek, M.F., Karger, D.R., Morris, R.: On the feasibility of peer-to-peer web indexing and search. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS. LNCS, vol. 2735, pp. 207–215. Springer (2003)