Timing Attacks on a Centralized Presence Model

Ge Zhang
Karlstad University, Karlstad, Sweden
ge.zhang@kau.se

Abstract—Presence information (PI) represents the updated status, context and willingness of communication partners in Voice over IP systems. For instance, the action that Alice switches her status (e.g., from “idle” to “busy”) will trigger PI messages to notify her buddies this change. In a centralized presence service system, presence communications are managed by a presence server based on users’ buddylists. The privacy concern in this paper is that networking intermediaries, as adversaries, might be able to profile the buddy-relationship among the users by utilizing message arrival time. We found that the threat cannot be totally eliminated even if the server processes messages in batches. Attackers might observe the traffic in several rounds and thus profile the results. In this paper, we introduce the attacks and discuss potential countermeasures.

I. INTRODUCTION

People nowadays increasingly rely on the Internet to exchange information on a global scale supported by the services such as email, instant message, online social networks and voice over IP. Many implementations of the services are integrated with presence services to exchange presence information (PI) between users. Presence information includes not only boolean information whether someone is available or not but also a variety of rich information [12], for example, the reason for unavailable (e.g., being occupied by another conversation), the communication conditions (e.g., in a noisy area). When a user changes his/her status (e.g., from “idle” to “busy”), his/her user-agent will generate a message containing his/her updated PI and then multicast this message to his/her buddies. In this way, other users can get a brief overview of their perspective communication partner’s context before launching a call or sending an instant message. Therefore, presence services help users to initiate a proper action at a proper time and thus provide better quality of experience to users.

Most proposed presence service architectures are centralized [10], [11], [9], [13]. A presence server (PS) is deployed to collect and dispatch PI messages for users in this kind of architectures. One important benefit for doing so is bandwidth saving: Let us say that if Alice has ten buddies, she just needs to send one PI message to the PS instead of sending ten message copies to each buddy. In addition, it is easy for a centralized PS to manage user location and billing information. We follow the terminology defined in [10]: a user can play two roles, namely presentity and watcher. A presentity sends to the PS a PI message containing his/her PI. This kind of messages is called PUBLISH message. A watcher receives a PI message containing other’s PI from the PS. The messages are named NOTIFY messages.

PI is secret information thus needs additional protection. For example, the content of PI should not be disclosed when it is transmitted over open networks. Otherwise, attackers can intercept it and keep on tracking the changes of the PI of a particular victim to profile his/her daily behavior. This requirement can be met by encrypting PI messages. Moreover, a presentity would like his/her PI only to be revealed to trusted watchers because of privacy considerations. Therefore, each presentity maintains a buddylist containing users which can be trusted. To work properly on behalf of users, the PS also needs to know these buddylists. Please note that the buddy relationship in this model is unnecessary to be symmetric.

A working example is illustrated in Figure 1. There are five users (A, B, C, D, E) served by the server in the figure. Let us assume that C and E are A’s buddies. As a result, every time when the PS receives a PUBLISH message from A, it generates NOTIFY messages to C and E on behalf of A.

Another privacy concern is not on the content of messages, but on the buddylists since they disclose potential relationships among users. In this paper, we consider timing attacks: We assume a global adversary model in which attackers can wiretap all communication channels. The attackers cannot read the content of messages but they can detect the sender, recipient and arrival time of each message. The attacker can link one PUBLISH message with corresponding NOTIFY message(s) by using their arrival time, and thus they can profile the buddylist of the presentity. For instance, it can be deduced that A has C and E in his/her buddylist if attackers can observe the traffic flows illustrated in Figure 1. Thus, it is predictable that A has a close relationship to C and E than anyone else. For personal users, this information reveals their social networks. In business level, it might disclose some secrecy of a company (e.g., its customers or suppliers). Moreover, attackers can use the relationship information for further attacks (e.g., social engineering, spam, fraud). Although timing attacks has been well studied in many networking models (e.g., HTTP, Email, and VoIP), little related work has been done on this type of multicast model to the best of our knowledge. Moreover, presence users do not frequently change their buddylists, which means that the communication relationships between presentsities and watchers are highly stable. Thus, presence traffic is easier to be analyzed then other kinds of traffic. A successful anonymous services for presence traffics should take above issues into account.

The rest of this paper is organized as follows: Section II introduces the system model and adversary model.
The short term attack method and its prevention are represented in Section III. Based on the prevention of short term attacks, we discuss the long term attacks with corresponding countermeasures in Section IV. Related work on presence privacy and traffic analysis attacks are given in Section V. Finally, section VI concludes this paper.

II. MODELS

A. System model

We assume that there are \( n \) users (as a set \( \mathcal{N} = \{1, \cdots, n\} \)) served by one PS. Each user (user \( i, i \in \mathcal{N} \)) manages his/her buddylist (\( \mathcal{R}_i \subseteq \mathcal{N}/\{i\} \)). The server owns a copy of the buddylist for each user. When the server receives a PUBLISH message from user \( i \), the PS will extract PI and forward it in a NOTIFY message to each element in \( \mathcal{R}_i \). We also assume that the PS has high capability thus the time for message processing is rather small. In addition, security measurements for confidentiality and integrity have been applied on the model: (1) The PS share a secrete key with each user. The messages exchanged between users and the PS are encrypted; (2) Both PS and users can authenticate received messages as well as the source of messages; (3) Replayed PUBLISH messages can be detected and discarded by the PS; and (4) All messages in the system are padded to a constant size.

B. Threat model

In this paper, the asset that an attacker is interested in is the buddylist \( \mathcal{R}_i \) for a given user \( i \). We consider a global adversary model in which the attackers are able to wiretap all traffic to and from the PS. The attackers can learn the message direction, message arrival time. The attackers can distinguish PUBLISH and NOTIFY messages by their directions, as PUBLISH messages are always sent from users to the PS and the other way around for NOTIFY messages. Nevertheless, the attackers are unable to decipher the encrypted messages so that they cannot read the content of any message. We further assume that no user changes his/her buddylist during attacks. Finally, we assume that the PS is a trusted party.

To describe the attacks from an attacker’s point of view, we use the vector \( \mathbf{r}_i \) to denote the buddylist of the user \( i \). Please note that \( \mathbf{r}_i \) and \( \mathcal{R}_i \) actually indicate the same thing. The vector \( \mathbf{r}_i \) has \( n \) elements corresponding to the \( n \) users. The values corresponding to the elements that appeared in the \( \mathcal{R}_i \) are set to 1 and the remaining ones to 0. Any user can have none buddy (all elements in \( \mathbf{r}_i \) are 0) or \( m \) \((1 \leq m \leq n-1)\) buddy(s) \((m\ \text{element(s)}\ \text{in}\ \mathbf{r}_i\ \text{is(are)}\ 1)\). We employ a unit vector \( \mathbf{u}_i \) to indicate that \( i \) sent a PUBLISH message. The unit vector \( \mathbf{u}_i \) has \( n \) elements but only the \( i \) element is set to 1 and the remaining ones are set to 0. We use another vector \( \overrightarrow{\sigma} \) with also \( n \) elements to denote those who received a NOTIFY message. The elements in \( \overrightarrow{\sigma} \) indicate the recipient(s) of the NOTIFY message(s).

Let us take the scenario in Figure 1 as an example. \( A, B, C, D \) and \( E \) are represented by 1 to 5 respectively. \( A \) has \( C \) and \( E \) in his/her buddylist, thus \( \mathcal{R}_A = \{C, E\} \) and \( \mathbf{r}_A = (0, 0, 1, 0, 1) \). From the attackers’ view, they can observe a PUBLISH message sent from \( A \) to the PS, indicating as \( \mathbf{u}_A = (1, 0, 0, 0, 0) \). Also, they can observe two NOTIFY messages are sent at almost the same time to \( C \) and \( E \), indicating as \( \overrightarrow{\sigma} = (0, 0, 1, 0, 1) \).

III. SHORT TERM ATTACK AND ITS PREVENTION

A. The linkabilities

There are two linkabilities which might enable attackers to conduct traffic analysis attacks.

- Timing linkability (\( \overrightarrow{\sigma} \) and \( \overrightarrow{\mathbf{r}_i} \)): The PS will produce a \( \overrightarrow{\sigma} \) for \( \mathbf{u}_i \) without any intentional delay or waiting for other conditions. Thus, the time point for observing a \( \mathbf{u}_i \) and its corresponding \( \overrightarrow{\sigma} \) should be very close. In this way, attackers can link a \( \overrightarrow{\sigma} \) with \( \mathbf{u}_i \) by the arrival time of messages.

- Buddylist linkability (\( \overrightarrow{\mathbf{r}_i} \) and \( \overrightarrow{\mathbf{r}_o} \)): Only the users in the \( \mathcal{R}_i \) (with the corresponding value “1” in \( \mathbf{r}_i \)) will get a copy of NOTIFY message. Thus, from the view of an external observer, the PS works as a function mapping from a unit vector \( \mathbf{u}_i \) to an output vector \( \overrightarrow{\sigma} \):

\[
PS(\mathbf{u}_i) = \overrightarrow{\sigma}
\]

The mapping relationship between \( \mathbf{u}_i \) to \( \overrightarrow{\sigma} \) is deterministic as long as there is no change on \( \mathcal{R}_i \). This means that the attackers just need to find out the \( \overrightarrow{\sigma} \) linked to a given \( \mathbf{u}_i \). And the \( \mathbf{r}_i \) is exactly equal to \( \overrightarrow{\sigma} \). Let us take the example in Figure 1 again. When the attackers have observed \( \mathbf{u}_A = (1, 0, 0, 0, 0) \) and related \( \overrightarrow{\sigma} = (0, 1, 0, 0, 1) \). They can easily predict the \( \mathbf{r}_A = (0, 1, 0, 0, 1) \).

B. A defending batch scheme

Mixes were proposed by Chaum [3] to provide communication anonymity. A mix is a proxy that receives a certain number of messages as a batch, performs some ciphering operations on them to modify their original appearances, and forwards the results in random order. As a result, mixes decrease the linkability between inputs and outputs.

To prevent timing correlation between \( \mathbf{u}_i \) and corresponding \( \overrightarrow{\sigma}_i \), we use the batch approach which was introduced by Chaum mix [3]. As we assume that PS can be trusted, we let PS work as an anonymity server. To do
so, a PS does not process received PUBLISH messages until it gets $b$ of them. When the condition is met, the PS processes them and flushes corresponding NOTIFY messages at once. This operation is defined as one batch, with the size of $b$. As a result, the relationship between PUBLISH and NOTIFY messages are mixed. Given a user who received a NOTIFY message, it could be a buddy of any one of the $b$ presentities. Thus, the anonymity set of presentity is the set of senders of the $b$ PUBLISH messages. The larger of $b$, the higher anonymity can be achieved. However, a large $b$ also means longer average waiting time for a PUBLISH message before being processed. Therefore, the $b$ should be carefully selected by the service provider.

Another problem is that the server might receive more than one PUBLISH message from the same user in a single batch. Let us define a message is valid if the PS got more than one PUBLISH messages from the same user in a single batch. Let us define $\alpha$ as the waiting time for a PUBLISH message before being processed. The larger of $\alpha$, the higher anonymity can be achieved. However, a large $\alpha$ also means longer average waiting time for a PUBLISH message before being processed. Therefore, the $\alpha$ should be carefully selected by the service provider.

```
T = \emptyset;
/* T is a temporary set to hold the identity of presentities in one batch */
M = \emptyset;
/* M is a buffer to hold NOTIFY messages in one batch */
repeat
  if Receive a PUBLISH message from user $i$ then
    $T = T \cup i$;
    if $|T| = b$ then
      for all the $j \in T$ do
        for all the $x \in R_i$, do
          Initialize a NOTIFY message $m$ about $j$’s PI for $x$;
          $M = M \cup m$;
        end
      end
    end
    repeat
      Choose an arbitrary message $(z)$ from $M$;
      Send the message $z$;
      $M = M \setminus z$;
    until $M = \emptyset$;
    $T = \emptyset$;
  end
until exits;
```

Algorithm 1: The batch processing on a presence server

Nevertheless, the previous work is based on a classical unicast traffic model: the sender only sends one message to one recipient for each round. In contrast, our system model is a multicast traffic model: a PUBLISH message may invoke none, one or more NOTIFY messages. Thus, the previous attack methods cannot be directly applied in our work.

A simple attacking method in our system model is: The attacker can collect some $\overrightarrow{o}$ for the batches in which the victim participates. Then, the attackers can exclude some watchers which are not in the victim’s buddylist of all the $b$ presentities. Therefore, it is difficult for the attacks to be successful as long as the threshold $b$ is large enough.

IV. LONG TERM ATTACK AND ITS PREVENTION

The solution shown in Algorithm 1 is useful to defend against short term attacks, for the attackers who only observe traffic for one batch. Unfortunately, attackers usually can observe traffic for more than one batch in reality. In a long run, attackers might collect $\overrightarrow{o}$ in different batches and conclude the results. This kind of attacks is called long term timing attack (e.g., [5] and [4]).

1In $b - 1$ attack, the attacker can have a valid user account with an empty buddylist. When the attacker find a victim sends a message, the attacker continuously sends $b - 1$ messages to satisfy the requirement of a batch. As a result, the flushed NOTIFY messages are exclusively for the victim. In this way, attackers can bypass the protection of the batch mechanism.

Example 1: Assume the victim has two buddy pairs and two following presentities, $A = \{B, C\}$ and $B = \{A, D\}$, respectively. They are in $N$ and $R = \{x_1, x_2, \ldots, x_b\}$ respectively. The attacker continuously sends empty buddylist. When the attacker find a victim sends a message, the attacker makes sure that the victim has received at least one PUBLISH message from the attacker. Thus, we let unit vectors indicate which presentity sent a PUBLISH message. The larger of $\alpha$, the higher anonymity can be achieved. However, a large $\alpha$ also means longer average waiting time for a PUBLISH message before being processed. Therefore, the $\alpha$ should be carefully selected by the service provider.

The modified internal processing logic is listed in Algorithm 1.

The $\overrightarrow{o}$ represents the output for the batch. The PS should generate NOTIFY messages only depending on the buddylist of each presentity.

$\overrightarrow{o} = P S(\overrightarrow{u}_1 + \overrightarrow{u}_2 + \cdots + \overrightarrow{u}_b)$ (3)

Even though an external observer wiretapped $\overrightarrow{o}$, he/she can only profile the mixed buddylists of all the $b$ presentities. Therefore, it is difficult for the attacks to be successful as long as the threshold $b$ is large enough.

```
T = \emptyset;
/* T is a temporary set to hold the identity of presentities in one batch */
M = \emptyset;
/* M is a buffer to hold NOTIFY messages in one batch */
repeat
  if Receive a PUBLISH message from user $i$ then
    $T = T \cup i$;
    if $|T| = b$ then
      for all the $j \in T$ do
        for all the $x \in R_i$, do
          Initialize a NOTIFY message $m$ about $j$’s PI for $x$;
          $M = M \cup m$;
        end
      end
    end
    repeat
      Choose an arbitrary message $(z)$ from $M$;
      Send the message $z$;
      $M = M \setminus z$;
    until $M = \emptyset$;
    $T = \emptyset$;
  end
until exits;
```

Algorithm 1: The batch processing on a presence server

Example 1: Assume the victim has two buddy pairs and two following presentities, $A = \{B, C\}$ and $B = \{A, D\}$, respectively. They are in $N$ and $R = \{x_1, x_2, \ldots, x_b\}$ respectively. The attacker continuously sends empty buddylist. When the attacker find a victim sends a message, the attacker makes sure that the victim has received at least one PUBLISH message from the attacker. Thus, we let unit vectors indicate which presentity sent a PUBLISH message. The larger of $\alpha$, the higher anonymity can be achieved. However, a large $\alpha$ also means longer average waiting time for a PUBLISH message before being processed. Therefore, the $\alpha$ should be carefully selected by the service provider.

The modified internal processing logic is listed in Algorithm 1.

The $\overrightarrow{o}$ represents the output for the batch. The PS should generate NOTIFY messages only depending on the buddylist of each presentity.

$\overrightarrow{o} = P S(\overrightarrow{u}_1 + \overrightarrow{u}_2 + \cdots + \overrightarrow{u}_b)$ (3)

Even though an external observer wiretapped $\overrightarrow{o}$, he/she can only profile the mixed buddylists of all the $b$ presentities. Therefore, it is difficult for the attacks to be successful as long as the threshold $b$ is large enough.

```
T = \emptyset;
/* T is a temporary set to hold the identity of presentities in one batch */
M = \emptyset;
/* M is a buffer to hold NOTIFY messages in one batch */
repeat
  if Receive a PUBLISH message from user $i$ then
    $T = T \cup i$;
    if $|T| = b$ then
      for all the $j \in T$ do
        for all the $x \in R_i$, do
          Initialize a NOTIFY message $m$ about $j$’s PI for $x$;
          $M = M \cup m$;
        end
      end
    end
    repeat
      Choose an arbitrary message $(z)$ from $M$;
      Send the message $z$;
      $M = M \setminus z$;
    until $M = \emptyset$;
    $T = \emptyset$;
  end
until exits;
```

Algorithm 1: The batch processing on a presence server

Example 1: Assume the victim has two buddy pairs and two following presentities, $A = \{B, C\}$ and $B = \{A, D\}$, respectively. They are in $N$ and $R = \{x_1, x_2, \ldots, x_b\}$ respectively. The attacker continuously sends empty buddylist. When the attacker find a victim sends a message, the attacker makes sure that the victim has received at least one PUBLISH message from the attacker. Thus, we let unit vectors indicate which presentity sent a PUBLISH message. The larger of $\alpha$, the higher anonymity can be achieved. However, a large $\alpha$ also means longer average waiting time for a PUBLISH message before being processed. Therefore, the $\alpha$ should be carefully selected by the service provider.

The modified internal processing logic is listed in Algorithm 1.

The $\overrightarrow{o}$ represents the output for the batch. The PS should generate NOTIFY messages only depending on the buddylist of each presentity.

$\overrightarrow{o} = P S(\overrightarrow{u}_1 + \overrightarrow{u}_2 + \cdots + \overrightarrow{u}_b)$ (3)

Even though an external observer wiretapped $\overrightarrow{o}$, he/she can only profile the mixed buddylists of all the $b$ presentities. Therefore, it is difficult for the attacks to be successful as long as the threshold $b$ is large enough.
\( N \), we assume that \( V \) is the reference set in which the elements are not the attacker’s interest and \( X \) is the victim set containing elements which can be attacked (e.g., the victim is \( x_j, 1 \leq j \leq b \)). Attackers might be interested in one or several elements in the \( X \). Moreover, \( V \) and \( X \) must be mutually exclusive (\( V \cap X = \emptyset \)).

For a successful intersection attack, the attackers do not need to observe the output for all the possible input combinations, which is \( \binom{n}{b} \). The attackers just need to observe the outputs for \( b + 1 \) rounds of specific input combinations. The outputs for these rounds are denoted as: \( o^1, o^2, \ldots, o^{b+1} \). The rounds with input and output are listed as:

- **The 1st round**: \( PS(u_{x_1} + \sum_{i=1}^{b-1} u_{v_i}) = o^1 \)
- **The 2nd round**: \( PS(u_{x_2} + \sum_{i=1}^{b-1} u_{v_i}) = o^2 \)
  \[ \vdots \] (4)
- **The \( b \)th round**: \( PS(u_{x_b} + \sum_{i=1}^{b-1} u_{v_i}) = o^b \)
- **The \( b + 1 \)th round**: \( PS(\sum_{i=1}^{b} u_{v_i}) = o^{b+1} \)

Therefore, if we sum them from the first round to the \( b \) round and apply Eq 3, we have:

\[
\sum_{i=1}^{b} o^i = \sum_{i=1}^{b} r_{x_i} + b \cdot (\sum_{i=1}^{b-1} r_{v_i})
\]

(5)

Then if we take the \( b + 1 \) round and Eq 3 into account, we have

\[
\sum_{i=1}^{b} o^i = o^{b+1} + b \cdot (\sum_{i=1}^{b-1} r_{v_i})
\]

(6)

Thus,

\[
\sum_{i=1}^{b-1} r_{v_i} = \frac{\sum_{i=1}^{b} o^i - o^{b+1}}{b}
\]

(7)

Finally, the attackers can calculate the results based on Eq 3, Eq 4 and Eq 7.

\[
r_{x_j} = \frac{o^j - \sum_{i=1}^{b} o^j - o^{b+1}}{b}, \quad 1 \leq j \leq b
\]

(8)

In this way, attackers can profile the buddylist of any user in \( X \).

As said, the attackers first should select two mutually exclusive sets: \( V \) with \( |V| = b - 1 \) and \( X \) with \( |X| = b \). Thus \( |V \cup X| = 2b - 1 \). Since \( V \subset N \) and \( X \subset N \), the following condition must be satisfied to make the attacks successful:

\[
2 \cdot b - 1 \leq n \implies b \leq \frac{n + 1}{2}
\]

(9)

**A. How to make this attack easier?**

The attackers do not need to observe the output for all the possible combination of inputs, which is \( \binom{n}{b} \). The attackers just need to observe the output for selected \( b + 1 \) combinations. Now the problem is how the attackers can observe the traffic for their desired input combinations. If we consider a passive adversary model, attackers only can passively wait for their desired inputting combinations. However, this is not the case for an active adversary model. Here we extend the adversary model in which the attackers not only can wiretap traffic, but also can actively manipulate the traffic. For example, the attackers can intentionally prevent undesired PUBLISH messages from arriving the PS (e.g., delay or drop the message). Thus, it is easy for attackers to get their desired input combinations. For example, if attackers want to have an input batch with the input from A, B, and C, they can drop the packets from the users other than A, B, and C until the desired batch input is formed.

**B. A discussion of countermeasures**

Considering the nature of this threat, we discuss some countermeasures that can be used to prevent it.

1. **Set a large batch threshold**: We know that a fundamental precondition for the long term attack introduced in this paper is that \( b \leq \frac{n + 1}{2} \). Thus, to break the condition, we can set a large threshold \( b \), to make \( b > \frac{n + 1}{2} \). In this way, the attackers cannot find enough elements to form \( V \) and \( X \) to mount attacks. Unfortunately, however, setting a large \( b \) may introduce side-effects. For instance, the time required to form the batch increases, which means the watchers have to wait for a long time to receive a NOTIFY message.

2. **Employ pool mechanism instead of threshold batch mechanism**: The threshold batch mechanism waits for \( b \) PUBLISH messages and then process all of them at once. Instead of using threshold batch mechanism, the PS can employ pool batch mechanism: the PS always buffers \( b \) messages and processes a randomly selected messages from the pool when it receives a new one. This pool mechanism further make the relationship between a PUBLISH message with it related NOTIFY messages indeterministic since the sent one is randomly selected. However, the time interval between a PUBLISH message and its related NOTIFY messages is also highly unpredictable. It may result in the similar side effect of the previous method.

3. **Enforce dummy traffic**: Users can constantly send dummy PUBLISH messages to the PS. And the attackers should be unable to distinguish the difference between dummy messages and real messages by observing. Thus, whether a user initialized a real PUBLISH message is unobservable. However, since the users who send dummy messages can not directly get benefit from sending, an incentive mechanism is needed. Another alternative is to let the PS to produce dummy NOTIFY messages. Users do not need to actively participate in this case. Nevertheless,

\[ \footnote{Attackers only can observe. They do not replay, delay and drop messages} \]
it may introduce resource overhead on the PS. Moreover, dummy traffic mechanism wastes bandwidth resource thus difficult to be scale in the Internet.

V. Related Work

Some previous research results are focused on presence traffic anonymity and long-term timing attacks.

Presence traffic anonymity: Loesing et al., [8] proposed a P2P architecture based on Distributed Hash Table (DHT) with existing anonymous overlay networks (e.g., Tor [6]) to provide anonymity services for presence and message users. A user first configures his/her rendezvous address in an anonymous overlay network and later registers this address on a DHT. To extract the rendezvous address from DHT for communication, the user’s buddies need to share a common knowledge with the him/her. Danezis et al., [7] proposed Drac, a system designed to provide anonymity for instant message, and VoIP communications. Drac employs buddies as relays to connect to untrusted contacts. The communications between buddies are unobservable by applying heartbeat packets in a constant rate. However, the buddy-relationship in Drac is public. Our work is different to theirs: First, we consider a centralized presence model, rather than a P2P presence architecture; and our goal is to hide the buddy-relationship among users.

Long-term timing attacks: Despite the level of protection provided by batch design in Chaum mixes [3], a system is still vulnerable to long term traffic analysis attacks [2], [4], [1], [5]. These models exploit the fact that a user in reality only frequently communicates with a set of contacts. Thus, as long as the attackers observe enough rounds, they can learn more hints. In [4], [1], an adversary observes and records the recipient sets when a given victim sends messages. The adversary first identifies mutually disjoint sets of recipient sets and then intersects the disjoint sets with the recipient set of each round. When the intersection generates a set containing only one elements, which should be the recipient of the victim. Another kind of attack, the Statistical Disclosure Attack (SDA) [5], is a probability-based alternative. The attacker profiles the traffic statistical results with the participation of the victim and without respectively. Then the attacker learns the probability of recipient of the victim for each round based on the statistical model. Their long-term timing attacks are mainly focused on the classical unicast traffic model: a sender sends ONE message to ONE recipient. Different to the previous work, our research is based on multicast presence traffic model, in which one PUBLISH message to the PS might result in none or several NOTIFY message(s), depending on the buddylist. Moreover, the relationship between presentities and watchers are usually stable, which leads to the presence traffic patterns more deterministic. Therefore, the attacking methods on presence traffic model are different to those on the classical traffic model.

VI. Conclusion

In this paper, we demonstrated the traffic analysis attacks targeting on a centralized presence service model taking advantage of the linkabilities between a PUBLISH message and its triggered NOTIFY message(s). The goal of attackers is to learn the buddy-relationship among users. We discussed a threshold batch-based mechanism to prevent the threat: a PS does not process received PUBLISH messages until it receives enough messages. The idea behind is to unlink the timing relationships between the messages. However, we showed that the threshold batch-based mechanism is insufficient to discourage stronger adversary models. Presence system is different to other communicating systems in at least two perspectives:

1. One PUBLISH message may trigger none, one or more NOTIFY messages, and (2) the relationship between users are not frequently changed. We have shown that the unlinkability provided by the threshold batch-based mechanism can be broken if the attackers observe the traffic for several rounds. Finally, we relaxed adversary models and discussed methods to make the attacks more easier or difficult from the attackers and defenders perspectively.

REFERENCES